INF 2178 Team 18 A2

Henry Zhang, Junchen Gu, Zhengze Li, Enhua Liu, Dominic Dike

Adaptive Learning Platforms for Online Education

Context: Platforms like Coursera and Khan Academy use AI to personalize learning pathways for students.

RCT Focus:

- Compare adaptive learning (AI-driven) vs. static content in improving student performance.
- Analyze student performance metrics such as Post_Test_Quiz_Score, Improvement_Score, Study_Time, Retention_Rate and Dropout_Rate.

5 RCT Workflow

Step 1: Collect Baseline Measures (Pre-Intervention Data) for all 5 RCT.

A-priori Power Analysis

We will conduct an a-priori power analysis was conducted to determine the minimum sample size required to detect statistically significant differences in educational outcomes across experimental groups.Based on this sample size, we will synthetically generate student baseline data for the 5 RCT experiments.

Data Structure

Baseline Variables (Control Variables)

These are pre-existing factors that could influence the dependent variable but are not being manipulated in the experiment.

1. **Student_ID**
2. **Age** (Continuous, Normal Distribution)
    - Older students may have different learning styles.
3. **GPA** (Continuous, Normal Distribution)
    - Prior academic performance influences learning capacity.
4. **Study_Hours_Per_Week** (Integer, Poisson Distribution)
    - More study hours may lead to better quiz scores.
5. **Tech_Savviness_Score** (Integer, Normal Distribution)
    - Tech-savvy students might engage more effectively with digital tools.
6. **Learning_Style** (Categorical)
    - Visual, Auditory, Kinesthetic
7. **Baseline_Quiz_Score** (Continuous, Normal Distribution)
    - Affects final performance; students with higher starting scores may have less improvement potential.
8. **Attention_Span** (Continuous, Normal Distribution)
    - Attention span per study session in minutes.
9. **Motivation_Level** (Ordinal, 1-10 Likert Scale)
    - Highly motivated students might perform better regardless of the learning method.
10. **Prior_Online_Exp** (Bernoulli - Yes/No)

Dependent Variable (Outcome Measures)

These are the key metrics we want to measure to determine the effect of adaptive learning.

1. **Post_Test_Quiz_Score** (Continuous, 0-100)

- Measures student performance post-experiment

2. **Improvement_Score** (Post_Test_Quiz_Score - Baseline_Quiz_Score)

- Captures the learning gain over time.

3. **Study_Time** (Continuous, in minutes)

- Measures student interaction with the learning platform.

4. **Retention_Rate** (Continuous, percentage)

- Measures how much content was completed by each student.

5. **Dropout_Rate** (Binary: Completed = 1, Dropped out = 0)

- Indicates if students disengaged from the study.

Step 2: Data Randomization & Subsetting

1. **Parallel RCT** (Traditional vs. AI-Driven Adaptive Learning)
    - **Stratification:** students were stratified based on their GPA, Baseline Quiz Score, and Tech Savviness Score to ensure balanced groups before assignment.
    - Group A: Uses a static learning platform (e.g., traditional online course with pre-set lessons).
    - Group B: Uses an adaptive AI-driven platform that adjusts content based on quiz performance.
2. **Matched-Pairs RCT** (Pairing Students with Similar Baselines)
    - Students are matched based on *Baseline_Quiz_Score*, *GPA*, *Study_Hours_Per_Week*, *Tech_Savviness_Score*, *Motivation_Level*, *Prior_Online_Exp*.
    - One in each pair is assigned to Static Learning, the other to Adaptive Learning.
3. **Withdrawal RCT** (Assessing Long-Term Effectiveness of Adaptation)
    - **Stratification:** students were stratified based on their GPA, Baseline Quiz Score, and Tech Savviness Score to ensure balanced groups before assignment.
    - Phase 1: All students use Adaptive Learning for 2 weeks.
    - Phase 2: Students with high engagement and performance are randomly assigned to:
        - Continue with Adaptive Learning
        - Switch to Static Content (Control).
4. **Factorial RCT** (Testing Different Personalization Features)
    - **Stratification:** students were stratified based on their GPA, Baseline Quiz Score, and Tech Savviness Score to ensure balanced groups before assignment.
    - Factor 1: Content Personalization (Content-based Filtering vs. No Content-based Filtering).
    - Factor 2: Interactivity (Interactivity vs. No Interactivity).
    - Four groups:
        A. Content-based Filtering + Interactivity
        B. Content-based Filtering + Low Interactivity
        C. No Content-based Filtering + Interactivity
        D. No Content-based Filtering + No Interactivity
5. **Cross-Over RCT** (Switching Between Static and Adaptive Learning)
    - Group A: Static first, then Adaptive Learning after a washout period.
    - Group B: Adaptive Learning first, then Static after a washout period.

Step 3: Implement Intervention (Post-Intervention Data)

Students will complete the intervention period, and we will collect:

1. **Post_Test_Quiz_Score**: Continuous (0-100)
    - Post-experiment quiz score (out of 100)
2. **Improvement_Score**: Continuous
    - Change in quiz scores (Final - Baseline)
3. **Study_Time**: Continuous
    - Total time spent on the learning platform (in minutes)
4. **Retention_Rate**: Continuous (0-100%)
    - % of content completed by each student
5. **Dropout_Rate**: Binary (Yes/No)
    - Whether the student completed the study (1 for Dropout/ 0 for Not Dropout)

Step 3: Implement Intervention Details

1. **Post-Test Quiz Score:**
    - **Assumption:** Adaptive learning will lead to a greater improvement in post-test quiz scores compared to the static content group.
    - **Adaptive Learning Group:**
        - **Intervention (Adaptive):** Personalized content tailored to individual student needs.
        - **Intervention Effect:** Increased knowledge acquisition and better understanding of the material.
        - **Code Implementation:** `np.random.normal(10, 5, num_students)` - A random normal distribution with a mean of 10 points higher than the baseline score, and a standard deviation of 5, is added to the baseline score for the adaptive group.
    - **Static Learning Group:**
        - **Intervention (Static):** Standard, non-personalized content.
        - **Intervention Effect:** Some improvement expected, but less than the adaptive group.
        - **Code Implementation:** `np.random.normal(5, 5, num_students)` - A random normal distribution with a mean of 5 points higher than the baseline score, and a standard deviation of 5, is added to the baseline score for the static group.
2. **Improvement Score:**
    - **Assumption:** The improvement score (post-test minus baseline) will be higher in the adaptive learning group.
    - **Adaptive Learning Group:**
        - **Intervention (Adaptive):** Personalized content tailored to individual student needs.
        - **Intervention Effect:** Greater knowledge gain, leading to a larger difference between pre- and post-test scores.
        - **Code Implementation:** Calculated as the difference between the generated `Post_Test_Quiz_Score` and the `Baseline_Quiz_Score`.
    - **Static Learning Group:**
        - **Intervention (Static):** Standard, non-personalized content.
        - **Intervention Effect:** Some improvement, but less than the adaptive group.
        - **Code Implementation:** Calculated as the difference between the generated `Post_Test_Quiz_Score` and the `Baseline_Quiz_Score`.
3. **Study Time:**
    - **Assumption:** Students in the adaptive learning group might spend more time on the platform due to personalized content and potentially more engaging activities.
    - **Adaptive Learning Group:**
        - **Intervention (Adaptive):** Personalized content, potentially leading to more engagement.
        - **Intervention Effect:** Potentially longer study times.
        - **Code Implementation:** `np.random.normal(400, 50, num_students)` - A random normal distribution with a mean of 400 minutes and a standard deviation of 50 minutes is used for the adaptive group.
    - **Static Learning Group:**
        - **Intervention (Static):** Standard, non-personalized content.
        - **Intervention Effect:** Potentially shorter study times.
        - **Code Implementation:** `np.random.normal(300, 50, num_students)` - A random normal distribution with a mean of 300 minutes and a standard deviation of 50 minutes is used for the static group.
4. **Retention Rate:**
    - **Assumption:** Adaptive learning will lead to higher retention rates due to increased engagement and perceived value of the personalized content.
    - **Adaptive Learning Group:**
        - **Intervention (Adaptive):** Personalized content, potentially leading to greater satisfaction and continued use.
        - **Intervention Effect:** Higher retention rates.
        - **Code Implementation:** `np.random.normal(90, 5, num_students)` - A random normal distribution with a mean of 90% and a standard deviation of 5% is used for the adaptive group.
    - **Static Learning Group:**
        - **Intervention (Static):** Standard, non-personalized content.
        - **Intervention Effect:** Lower retention rates.
        - **Code Implementation:** `np.random.normal(80, 5, num_students)` - A random normal distribution with a mean of 80% and a standard deviation of 5% is used for the static group.
5. **Dropout Rate:**
    - **Assumption:** Adaptive learning will lead to lower dropout rates due to increased engagement and better learning outcomes.

- **Adaptive Learning Group:**
  - **Intervention (Adaptive):** Personalized content, leading to a more supportive and effective learning environment.
  - **Intervention Effect:** Lower dropout rates.
  - **Code Implementation:** `np.random.choice([0, 1], num_students, p=[0.9, 0.1])` - A 10% dropout rate is simulated for the adaptive group.
- **Static Learning Group:**
  - **Intervention (Static):** Standard, non-personalized content.
  - **Intervention Effect:** Higher dropout rates.
  - **Code Implementation:** `np.random.choice([0, 1], num_students, p=[0.75, 0.25])` - A 25% dropout rate is simulated for the static group.

Step 4: Post Test Data Cleaning

1. Previw and understand data
2. Handling Missing Data
   - ✔ Removing rows/columns: If a column has too many missing values, drop it.
   - ✔ Imputation (Filling missing values):
   - Numerical: Mean, median, mode, interpolation, or predictive imputation.
   - Categorical: Mode, "Unknown" category, or predictive modeling (e.g., decision trees).
   - ✔ Using domain knowledge: Sometimes missing values are meaningful (e.g., missing survey responses may indicate refusal).
3. Handling Duplicates
4. Handling Outliers

Step 5: Perform Analysis

1. **Parallel RCT** (Static vs. Adaptive)
   - One-Way ANOVA: Compare continuous metrics.
   - Chi-Square Test: Binary metric Dropout_Rate.
2. **Matched-Pairs RCT** (Static vs. Adaptive, matched on baseline)
   - Paired t-test: Control for confounding factors with matched pairs.
   - Chi-Square Test: Binary metric Dropout_Rate.
3. **Withdrawal RCT** (Continue Adaptive vs. Switch to Static)

   ①Within-Group Analysis:
   - Repeated Measures ANOVA: Compare the treatment group's performance before and after withdrawal.
   - Chi-Square Test: Binary metric Dropout_Rate.

   ②Between-Group Analysis:
   - Two-Way ANOVA: Compare the treatment group's performance after withdrawal with the control group's performance during the same period.
   - Chi-Square Test: Binary metric Dropout_Rate.
4. **Factorial RCT** (Content Based × Difficulty Adaptation)
   - Two-Way ANOVA: Assess the main effects and interaction effects between the two factors (algorithm type and interactivity level).
   - Chi-Square Test: Binary metric Dropout_Rate.
5. **Cross-Over RCT** (Static → Adaptive vs. Adaptive → Static)

   ①Within-Subject Comparison
   - Paired t-test: Compare student performance before and after switching.
   - Chi-Square Test: Binary metric Dropout_Rate.

   ②Between-Subject Comparison
   - One-Way ANOVA: Compare the treatment group's performance before and after withdrawal.
   - Chi-Square Test: Binary metric Dropout_Rate.

In [162…
```python
# Data manipulation and numerical operations
import pandas as pd  # DataFrame handling
import numpy as np  # Numerical computations

# Data visualization
import matplotlib.pyplot as plt  # Basic plotting
```

```
import seaborn as sns  # Statistical data visualization

# Statistical analysis
from scipy import stats  # General statistical functions
from scipy.stats import ttest_rel, ttest_ind, f_oneway  # T-tests and ANOVA
from scipy.stats import chi2_contingency  # Chi-square test

# Regression and ANOVA
import statsmodels.api as sm  # Regression analysis
import statsmodels.formula.api as smf
from statsmodels.formula.api import ols, logit
from statsmodels.stats.anova import anova_lm  # ANOVA analysis
from statsmodels.stats.multicomp import MultiComparison

# Machine learning utilities
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit  # Data splitting for training/tes
from sklearn.neighbors import NearestNeighbors  # Nearest neighbors algorithm

# Power analysis (for statistical tests)
from statsmodels.stats.power import (
    FTestAnovaPower,  # Power analysis for ANOVA
    TTestPower,  # Power analysis for t-tests
    GofChisquarePower  # Power analysis for chi-square test
)

# Graph-based visualization
import networkx as nx  # Flowcharts and process diagrams

# Supress Warnings
import warnings
warnings.filterwarnings('ignore')
```

Step 1: Define a common baseline dataframe generation function for reusability in each of the 5 RCT Types.

In [163…

```
def generate_baseline_data(sample_size):
    """
    Generate a synthetic dataset for university students based on a given sample size.

    Parameters:
        sample_size (int): Number of students to generate data for. Each RCT uses differnt sample size.

    Returns:
        pd.DataFrame: A DataFrame containing the generated student data.
    """
    np.random.seed(18)  # Set random seed for reproducibility

    data = pd.DataFrame({
        'Student_ID': range(1, sample_size + 1),
        'Age': np.random.normal(22, 2, sample_size).astype(int),  # Normally distributed around 22 years
        'GPA': np.clip(np.random.normal(3.2, 0.4, sample_size), 2.0, 4.0).round(1),  # Clipped between 2.0 and 4
        'Study_Hours_Per_Week': np.random.normal(50, 10, sample_size).astype(int), # Study hours per week
        'Tech_Savviness_Score': np.random.normal(5, 3, sample_size).astype(int), # Score out of 10
        'Learning_Style': np.random.choice(['Visual', 'Auditory', 'Kinesthetic'], sample_size),
        'Baseline_Quiz_Score': np.random.normal(70, 10, sample_size).astype(int), # Score out of 100
        'Attention_Span': np.random.normal(30, 10, sample_size).astype(int),  # In minutes
        'Motivation_Level': np.random.poisson(7, sample_size),  # 1-10 scale
        'Prior_Online_Exp': np.random.choice([0, 1], sample_size)  # 0 = No, 1 = Yes
    })

    return data
```

1. Parallel RCT

Hypothesis:

- H_0 (Null Hypothesis): There is no significant difference in learning outcomes between adaptive learning and static learning.
- H_1 (Alternative Hypothesis): Adaptive learning: higher Post_Test_Quiz_Score, Improvement_Score, longer Study_Time, higher Retention_Rate, lower Dropout_Rate

⭐ **Expectation:**

- The treatment group that receives adaptive learning with AI-based personalized content will have better post-intervention results (higher Post_Test_Quiz_Score, Improvement_Score, longer Study_Time, higher Retention_Rate, lower Dropout_Rate).

Treatment & Control Groups:

- Treatment Group: Receives adaptive learning with AI-based personalized content.
- Control Group: Receives traditional static content (same for all students).

Analysis Method:

1. Run Shapiro-Wilk and Levene's test before ANOVA

- Run **Shapiro-Wilk test** to check if data is normally distributed.
- Run **Equal variances test** to check if data has equal variances across groups.

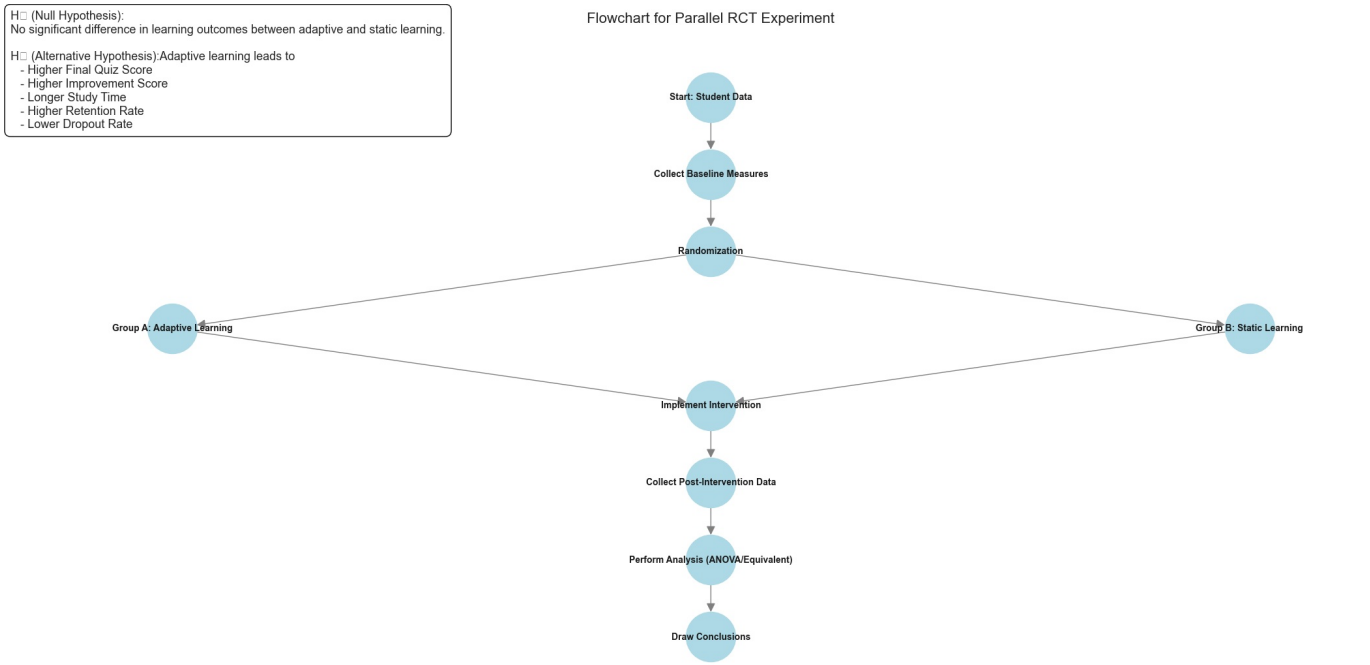
2. Run Use **One-Way ANOVA** for continuous variables (Post_Test_Quiz_Score, Improvement_Score, Study_Time, Retention_Rate).
3. Use **Chi-Square Test** for categorical variable (Dropout_Rate).
4. Compare the mean of each metric to evaluate which group has the better result.

1.1 Flowchart for Parallel RCT Experiment

In [164…

```
import networkx as nx
import matplotlib.pyplot as plt

# Initialize the directed graph
G = nx.DiGraph()

# Define the nodes for the flowchart
nodes = [
    "Start: Student Data",
    "Collect Baseline Measures",
    "Randomization",
    "Group A: Adaptive Learning",
    "Group B: Static Learning",
    "Implement Intervention",
    "Collect Post-Intervention Data",
    "Perform Analysis (ANOVA/Equivalent)",
    "Draw Conclusions"
]

# Add the edges (connections between the nodes)
edges = [
    ("Start: Student Data", "Collect Baseline Measures"),
    ("Collect Baseline Measures", "Randomization"),
    ("Randomization", "Group A: Adaptive Learning"),
    ("Randomization", "Group B: Static Learning"),
    ("Group A: Adaptive Learning", "Implement Intervention"),
    ("Group B: Static Learning", "Implement Intervention"),
    ("Implement Intervention", "Collect Post-Intervention Data"),
    ("Collect Post-Intervention Data", "Perform Analysis (ANOVA/Equivalent)"),
    ("Perform Analysis (ANOVA/Equivalent)", "Draw Conclusions")
]

# Add nodes and edges to the graph
G.add_nodes_from(nodes)
G.add_edges_from(edges)

# Create a customized layout for alignment
pos = {
    "Start: Student Data": (0, 4),
    "Collect Baseline Measures": (0, 3),
    "Randomization": (0, 2),
    "Group A: Adaptive Learning": (-1, 1),
    "Group B: Static Learning": (1, 1),
    "Implement Intervention": (0, 0),
    "Collect Post-Intervention Data": (0, -1),
    "Perform Analysis (ANOVA/Equivalent)": (0, -2),
    "Draw Conclusions": (0, -3)
}

plt.figure(figsize=(20, 10))
nx.draw(
    G,
    pos,
    with_labels=True,
    node_size=3000,
    node_color="lightblue",
    font_size=10,
    font_weight="bold",
    arrowsize=20,
    edge_color="gray"
)


plt.title("Flowchart for Parallel RCT Experiment", fontsize=16, pad=20)

# Add hypothesis text
hypothesis_text = (
    "H₀ (Null Hypothesis):\nNo significant difference in learning outcomes between adaptive and static learning
```

```
    "\nH₁ (Alternative Hypothesis):Adaptive learning leads to\n"
    "    - Higher Final Quiz Score\n"
    "    - Higher Improvement Score\n"
    "    - Longer Study Time\n"
    "    - Higher Retention Rate\n"
    "    - Lower Dropout Rate"
)

# Position hypothesis text
plt.text(-1.3, 3.6, hypothesis_text, ha="left", fontsize=13, bbox=dict(facecolor="white", edgecolor="black", bo

plt.show()
```



1.2 Parallel RCT A-priori Power Analysis For Sample Size Determination

In [165…

```
from statsmodels.stats.power import TTestPower, GofChisquarePower

# Parameters

alpha = 0.05
power = 0.8
num_groups = 2  # Parallel RCT has 2 independent groups


# 1. Continuous Variables (Two-Sample t-tests)

effect_sizes = {
    "Post_Test_Quiz_Score": 0.3,  # Cohen's d (medium effect)
    "Study_Time": 0.20,       # Cohen's d (small effect)
    "Retention_Rate": 0.25    # Cohen's d (medium effect)
}

# Initialize power analyzer for t-tests
ttest_power = TTestPower()
continuous_results = {}

for metric, d in effect_sizes.items():
    # Calculate per-group sample size for two-sample t-test
    n_per_group = ttest_power.solve_power(
        effect_size=d,
        alpha=alpha,
        power=power,
        alternative='two-sided'
    )
    continuous_results[metric] = {
        "Per Group": int(round(n_per_group)),
        "Total": int(round(n_per_group * num_groups))
    }


# 2. Dropout Rate (Binary) (Chi-Square Test of Independence)

# Using Cramér's V = 0.2 (small effect for 2x2 table)
chi2_power = GofChisquarePower()
total_dropout_n = chi2_power.solve_power(
```

```
    effect_size=0.2,  # Equivalent to Cramér's V for 2x2
    alpha=alpha,
    power=power,
    nobs=None
)

# 3. Final Sample Size Calculation

# Get largest continuous variable requirement
max_continuous = max([v["Total"] for v in continuous_results.values()])
paralle_rct_num_students = max(max_continuous, int(round(total_dropout_n)))

print("Continuous Variables (Two-Sample t-tests):")
for metric, res in continuous_results.items():
    print(f"- {metric}: {res['Per Group']} per group → {res['Total']} total")

print(f"\nDropout Rate (Chi-Square Test): {int(round(total_dropout_n))} total students")
print(f"\nFinal Recommended Sample Size for Parallel RCT: {paralle_rct_num_students} students")
```

```
Continuous Variables (Two-Sample t-tests):
- Post_Test_Quiz_Score: 89 per group → 178 total
- Study_Time: 198 per group → 396 total
- Retention_Rate: 128 per group → 255 total

Dropout Rate (Chi-Square Test): 196 total students

Final Recommended Sample Size for Parallel RCT: 396 students
```

1.1.3 Parallel RCT Randomization (Blocking Factor)

In [166…

```
# Step 1: Collect Baseline Measures (Pre-Intervention Data)
parallel_rct_data = generate_baseline_data(paralle_rct_num_students)  # paralle_rct_num_students is determined

# Step 2: Define the Blocking Factor (Performance_Level)
# Combine GPA, Baseline_Quiz_Score, and Tech_Savviness_Score into a single score for blocking
parallel_rct_data['Combined_Score'] = (
    parallel_rct_data['GPA'] + parallel_rct_data['Baseline_Quiz_Score'] + parallel_rct_data['Tech_Savviness_Sco
)

# Define Performance_Level based on quartiles of the Combined_Score
parallel_rct_data['Performance_Level'] = pd.qcut(
    parallel_rct_data['Combined_Score'],
    q=3,  # Divide into 3 equal-sized blocks (Low, Medium, High)
    labels=['Low', 'Medium', 'High']
)

# Step 3: Blocked Random Assignment
# Initialize the RCT_Parallel_Group column
parallel_rct_data['RCT_Parallel_Group'] = None

# Randomize within each block
for level in parallel_rct_data['Performance_Level'].unique():
    # Get indices of participants in the current block
    block_indices = parallel_rct_data[parallel_rct_data['Performance_Level'] == level].index

    # Shuffle the indices for randomization
    np.random.seed(18)  # Set seed for reproducibility
    shuffled_indices = np.random.permutation(block_indices)

    # Split the shuffled indices into two equal groups
    half = len(shuffled_indices) // 2
    parallel_rct_data.loc[shuffled_indices[:half], 'RCT_Parallel_Group'] = 'Static'
    parallel_rct_data.loc[shuffled_indices[half:], 'RCT_Parallel_Group'] = 'Adaptive'

# Drop the temporary 'Combined_Score' column (optional)
parallel_rct_data.drop(columns=['Combined_Score'], inplace=True)
```

In [167…

```
print(parallel_rct_data.info())
print('\n')
print(parallel_rct_data.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396 entries, 0 to 395
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Student_ID            396 non-null    int64
 1   Age                   396 non-null    int64
 2   GPA                   396 non-null    float64
 3   Study_Hours_Per_Week  396 non-null    int64
 4   Tech_Savviness_Score  396 non-null    int64
 5   Learning_Style        396 non-null    object
 6   Baseline_Quiz_Score   396 non-null    int64
 7   Attention_Span        396 non-null    int64
 8   Motivation_Level      396 non-null    int64
 9   Prior_Online_Exp      396 non-null    int64
 10  Performance_Level     396 non-null    category
 11  RCT_Parallel_Group    396 non-null    object
dtypes: category(1), float64(1), int64(8), object(2)
memory usage: 34.7+ KB
None
```

	Student_ID	Age	GPA	Study_Hours_Per_Week	Tech_Savviness_Score	Learning_Style	Baseline_Quiz_Score	Attention_Span	Motivation_Level	Prior_Online_Exp	Performance_Level	RCT_Parallel_Group
0	1	22	2.9	61	-3	Kinesthetic	57	37	7	0	Low	Static
1	2	26	2.5	36	0	Visual	57	48	3	1	Low	Static
2	3	21	3.0	60	4	Auditory	86	28	7	0	High	Static
3	4	22	3.3	55	6	Auditory	65	39	8	1	Medium	Static
4	5	22	3.5	46	6	Kinesthetic	73	37	5	0	High	Static

1.1.4 Parallel RCT Intervention (Blocking Factor)

In [168…

```python
# Step 3: Implement Intervention (Post-Intervention Data)

np.random.seed(18)

# Generate post-test metrics for the copied dataset
parallel_rct_data['Post_Test_Quiz_Score'] = (parallel_rct_data['Baseline_Quiz_Score'] +
                                             np.where(parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive',
                                                      np.random.normal(10, 5, paralle_rct_num_students),
                                                      np.random.normal(5, 5, paralle_rct_num_students))
                                            ).astype(int)

parallel_rct_data['Improvement_Score'] = (parallel_rct_data['Post_Test_Quiz_Score'] - parallel_rct_data['Baseli

parallel_rct_data['Study_Time'] = np.where(parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive',
                                           np.random.normal(400, 50, paralle_rct_num_students),
                                           np.random.normal(300, 50, paralle_rct_num_students)
                                           ).astype(int)

parallel_rct_data['Retention_Rate'] = np.where(parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive',
                                               np.random.normal(90, 5, paralle_rct_num_students),
                                               np.random.normal(80, 5, paralle_rct_num_students)
                                               ).round(2)

parallel_rct_data['Dropout_Rate'] = np.where(parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive',
                                             np.random.choice([0, 1], paralle_rct_num_students, p=[0.9, 0.1]),
                                             np.random.choice([0, 1], paralle_rct_num_students, p=[0.75, 0.25]
                                             ).round(2)

# Check the result
parallel_rct_data[['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate', 'Dropout_Rate']
```

Out[168…

	Post_Test_Quiz_Score	Improvement_Score	Study_Time	Retention_Rate	Dropout_Rate
0	58	1	157	75.70	0
1	53	-4	224	75.53	0
2	88	2	294	81.65	0
3	71	6	321	86.15	1
4	82	9	328	75.03	0

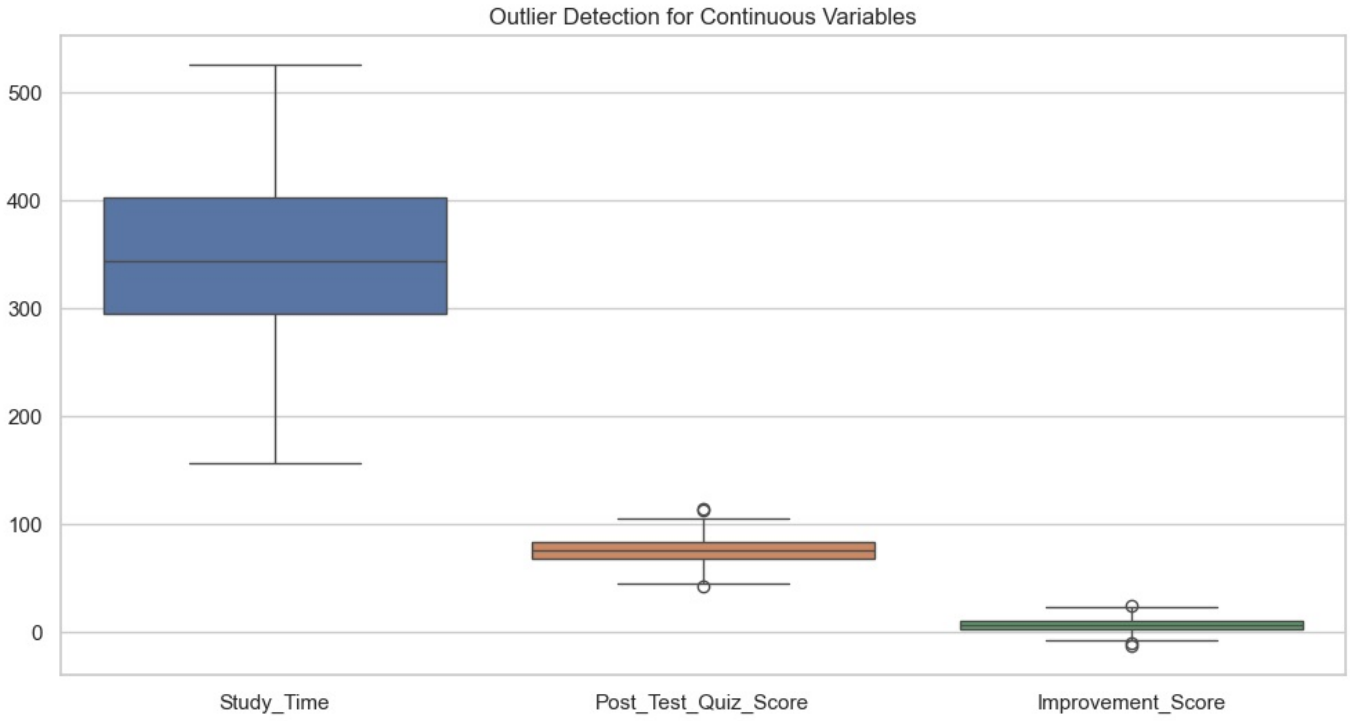
1.1.5 Parallel RCT Data Cleaning (with a Blocking Factor)

In [169…

```
# 1. Check for Missing Data

print("Missing Values Check:\n")
print(parallel_rct_data.isnull().sum())

# Replace placeholder codes (e.g., -18) with NaN if present
parallel_rct_data.replace(-18, np.nan, inplace=True)

# Drop rows with critical missing values (if any)
critical_cols = ['Post_Test_Quiz_Score', 'Retention_Rate', 'RCT_Parallel_Group']
parallel_rct_data.dropna(subset=critical_cols, inplace=True)


# 2. Remove Duplicates

print(f"\nInitial Shape: {parallel_rct_data.shape}")
parallel_rct_data.drop_duplicates(subset=['Student_ID'], keep='first', inplace=True)
print(f"Post-Deduplication Shape: {parallel_rct_data.shape}")


# 3. Verify Randomization Balance

print("\nGroup Balance:")
print(parallel_rct_data['RCT_Parallel_Group'].value_counts())

# Compare baseline covariates between groups
print("\nBaseline Covariate Balance:")
baseline_cols = ['GPA', 'Baseline_Quiz_Score', 'Study_Hours_Per_Week']
for col in baseline_cols:
    adaptive = parallel_rct_data[parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive'][col]
    static = parallel_rct_data[parallel_rct_data['RCT_Parallel_Group'] == 'Static'][col]
    t_stat, p_value = ttest_ind(adaptive, static, nan_policy='omit')
    print(f"{col}: t = {t_stat:.2f}, p = {p_value:.4f}")


# 4. Detect and Handle Outliers

continuous_vars = ['Study_Time', 'Post_Test_Quiz_Score', 'Improvement_Score']

plt.figure(figsize=(12, 6))
sns.boxplot(data=parallel_rct_data[continuous_vars])
plt.title("Outlier Detection for Continuous Variables")
plt.show()

# Cap outliers using IQR
for var in continuous_vars:
    q1 = parallel_rct_data[var].quantile(0.25)
    q3 = parallel_rct_data[var].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    parallel_rct_data[var] = np.where(parallel_rct_data[var] < lower_bound, lower_bound,
                                      np.where(parallel_rct_data[var] > upper_bound, upper_bound,
                                      parallel_rct_data[var]))


# 5. Validate Variable Ranges/Types

# Ensure binary variables are 0/1
parallel_rct_data['Dropout_Rate'] = parallel_rct_data['Dropout_Rate'].apply(lambda x: 1 if x == 1 else 0)

# Standardize categorical variables
parallel_rct_data['Learning_Style'] = parallel_rct_data['Learning_Style'].str.strip().str.title()

# Ensure valid percentage ranges
parallel_rct_data['Retention_Rate'] = parallel_rct_data['Retention_Rate'].clip(0, 100)


# 6. Save Cleaned Data
```

```
clean_parallel_rct_data = parallel_rct_data.copy()
print("\nData cleaning complete. Cleaned data saved as clean_parallel_rct_data.")
```

```
Missing Values Check:

Student_ID              0
Age                     0
GPA                     0
Study_Hours_Per_Week    0
Tech_Savviness_Score    0
Learning_Style          0
Baseline_Quiz_Score     0
Attention_Span          0
Motivation_Level        0
Prior_Online_Exp        0
Performance_Level       0
RCT_Parallel_Group      0
Post_Test_Quiz_Score    0
Improvement_Score       0
Study_Time              0
Retention_Rate          0
Dropout_Rate            0
dtype: int64

Initial Shape: (396, 17)
Post-Deduplication Shape: (396, 17)

Group Balance:
RCT_Parallel_Group
Adaptive    199
Static      197
Name: count, dtype: int64

Baseline Covariate Balance:
GPA: t = 0.47, p = 0.6386
Baseline_Quiz_Score: t = 0.05, p = 0.9620
Study_Hours_Per_Week: t = -0.87, p = 0.3844
```




```
Data cleaning complete. Cleaned data saved as clean_parallel_rct_data.
```

1.1.6 Parallel RCT Analysis (Blocking Factor)

In [170…

```
# Define metrics
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Perform Two-Way ANOVA and Post-Hoc Analysis
anova_results_two_way = {}
post_hoc_results = {}

for metric in metrics:
    # Fit the Two-Way ANOVA model
    model = ols(f'{metric} ~ C(RCT_Parallel_Group) + C(Performance_Level) + C(RCT_Parallel_Group):C(Performance_
                data=clean_parallel_rct_data).fit()
    anova_table = sm.stats.anova_lm(model, typ=2)
```

```
    # Compute mean comparisons
    mean_comparison = clean_parallel_rct_data.groupby('RCT_Parallel_Group')[metric].mean()
    better_group = 'Adaptive' if mean_comparison['Adaptive'] > mean_comparison['Static'] else 'Static'
    significance = 'Significant' if anova_table['PR(>F)'].min() < 0.05 else 'Not Significant'

    # Add Better Group and Significance to the Two-Way ANOVA results
    anova_table['Better Group'] = better_group
    anova_table['Significance'] = significance

    anova_results_two_way[metric] = anova_table

    # Perform Post-Hoc Analysis if the interaction or main effects are significant
    if significance == 'Significant':
        # Perform Tukey's HSD for pairwise comparisons
        mc = MultiComparison(clean_parallel_rct_data[metric], clean_parallel_rct_data['RCT_Parallel_Group'])
        post_hoc = mc.tukeyhsd()  # Tukey's HSD test
        post_hoc_results[metric] = post_hoc

# Display Two-Way ANOVA Results
for metric, result in anova_results_two_way.items():
    print(f"Two-Way ANOVA Results for {metric}:")
    display(result)

# Display Post-Hoc Analysis Results
for metric, result in post_hoc_results.items():
    print(f"\nPost-Hoc Analysis Results for {metric}:\n")
    print(result)
```

```
Two-Way ANOVA Results for Post_Test_Quiz_Score:
```

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Parallel_Group)	3000.611585	1.0	56.797958	3.409066e-13	Adaptive	Significant
C(Performance_Level)	29788.489901	2.0	281.930094	1.811663e-76	Adaptive	Significant
C(RCT_Parallel_Group):C(Performance_Level)	15.109713	2.0	0.143004	8.667958e-01	Adaptive	Significant
Residual	20603.531371	390.0	NaN	NaN	Adaptive	Significant

```
Two-Way ANOVA Results for Improvement_Score:
```

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Parallel_Group)	3039.877552	1.0	111.017569	5.197844e-23	Adaptive	Significant
C(Performance_Level)	85.198983	2.0	1.555751	2.123377e-01	Adaptive	Significant
C(RCT_Parallel_Group):C(Performance_Level)	93.858943	2.0	1.713883	1.815189e-01	Adaptive	Significant
Residual	10678.960622	390.0	NaN	NaN	Adaptive	Significant

```
Two-Way ANOVA Results for Study_Time:
```

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Parallel_Group)	995175.041728	1.0	404.009436	3.494439e-62	Adaptive	Significant
C(Performance_Level)	23788.398084	2.0	4.828667	8.481643e-03	Adaptive	Significant
C(RCT_Parallel_Group):C(Performance_Level)	37071.165455	2.0	7.524857	6.215468e-04	Adaptive	Significant
Residual	960666.340907	390.0	NaN	NaN	Adaptive	Significant

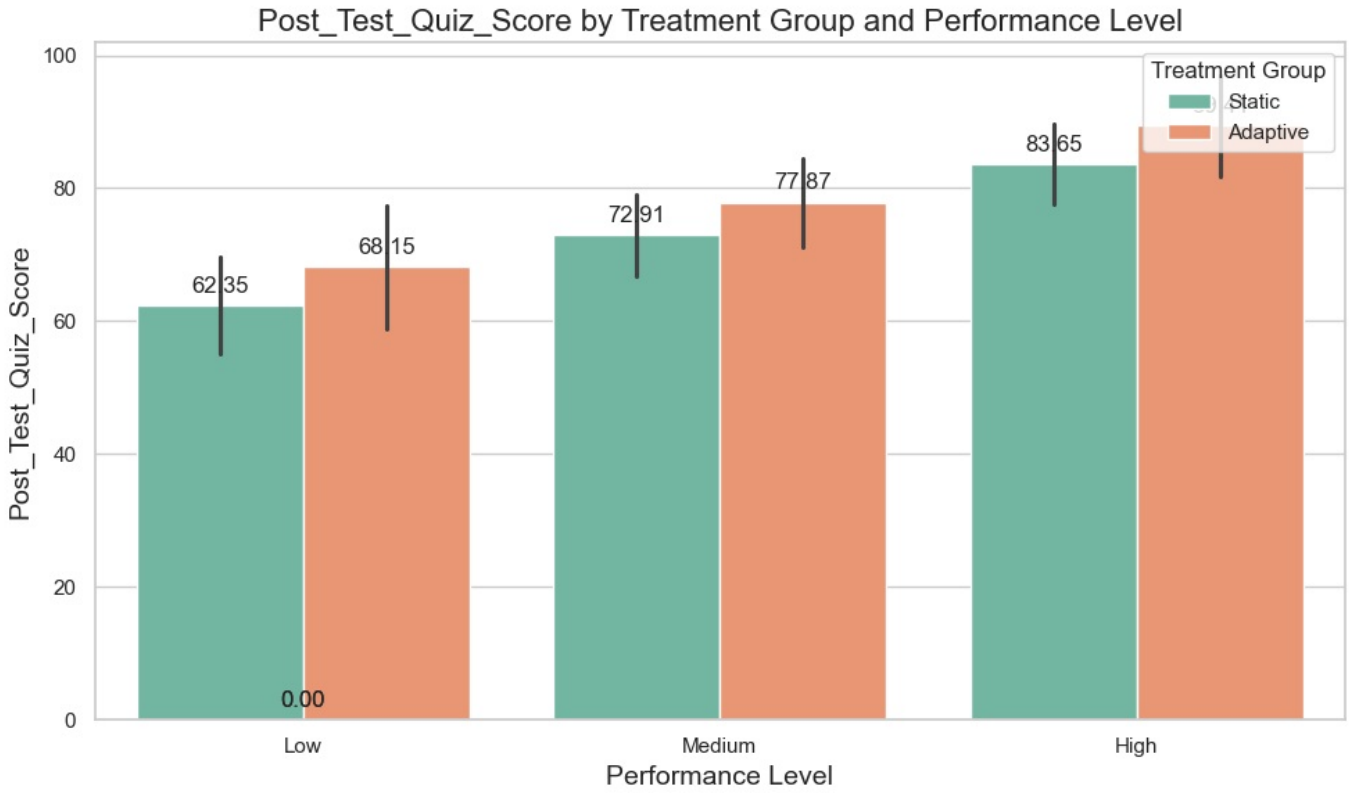
```
Two-Way ANOVA Results for Retention_Rate:
```

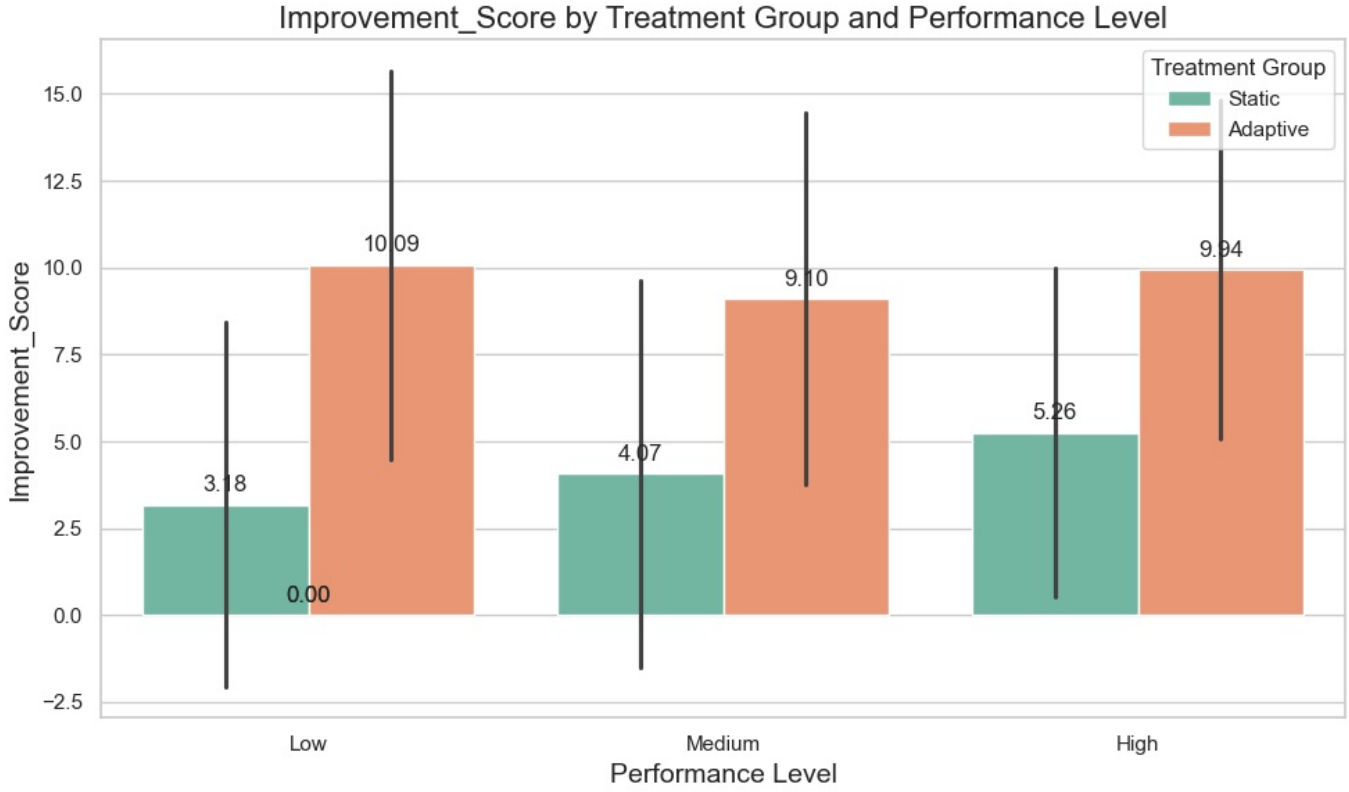
	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Parallel_Group)	9465.398199	1.0	371.838624	1.135129e-58	Adaptive	Significant
C(Performance_Level)	71.390379	2.0	1.402250	2.472805e-01	Adaptive	Significant
C(RCT_Parallel_Group):C(Performance_Level)	25.455224	2.0	0.499991	6.069241e-01	Adaptive	Significant
Residual	9927.708047	390.0	NaN	NaN	Adaptive	Significant

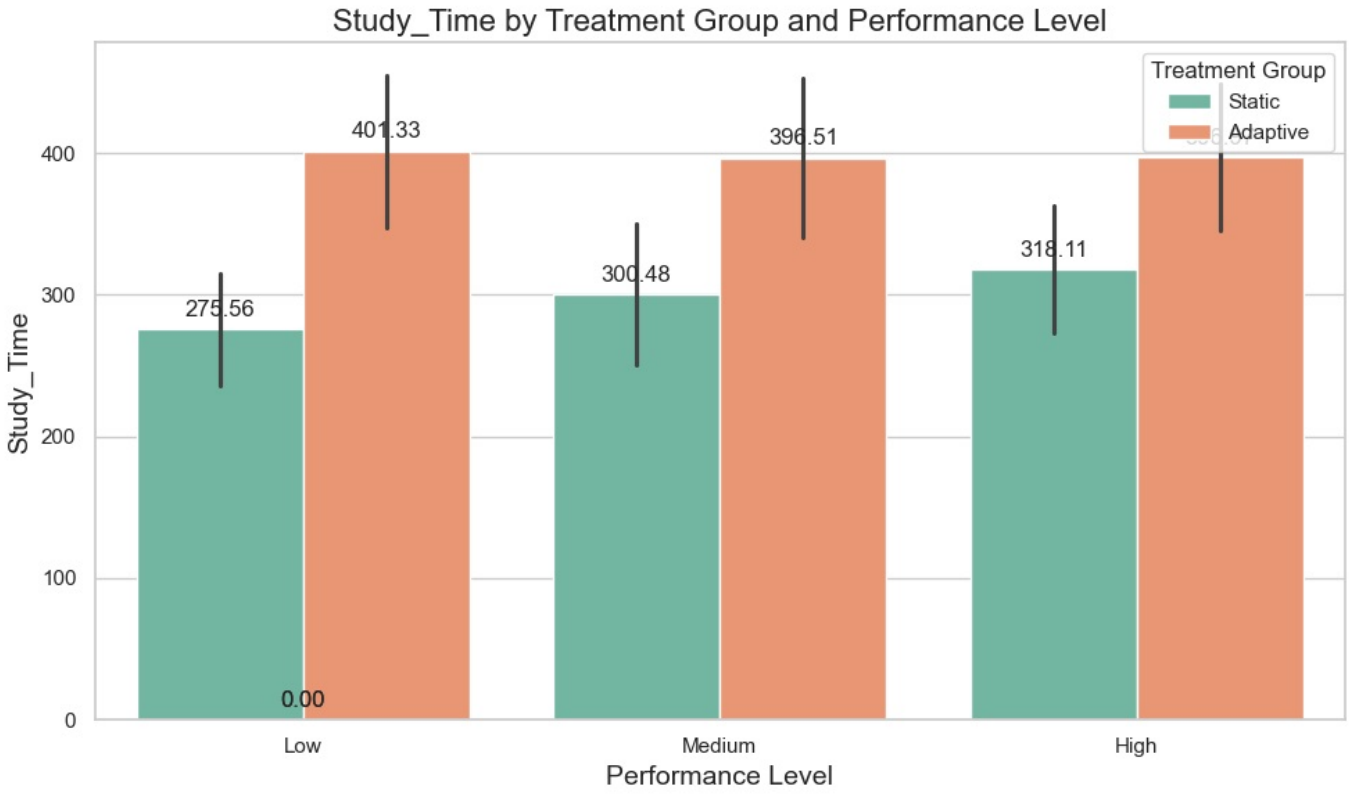
```
Post-Hoc Analysis Results for Post_Test_Quiz_Score:

 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj  lower   upper  reject
-----------------------------------------------------
Adaptive Static  -5.5587   0.0 -7.7936 -3.3237   True
-----------------------------------------------------

Post-Hoc Analysis Results for Improvement_Score:

 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj  lower   upper  reject
-----------------------------------------------------
Adaptive Static   -5.543   0.0 -6.5802 -4.5057   True
-----------------------------------------------------

Post-Hoc Analysis Results for Study_Time:

   Multiple Comparison of Means - Tukey HSD, FWER=0.05
=========================================================
 group1  group2  meandiff p-adj   lower    upper   reject
---------------------------------------------------------
Adaptive Static -100.3121   0.0 -110.3732 -90.2509   True
---------------------------------------------------------

Post-Hoc Analysis Results for Retention_Rate:

 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj  lower    upper  reject
-----------------------------------------------------
Adaptive Static    -9.78   0.0 -10.7767 -8.7834   True
-----------------------------------------------------
```

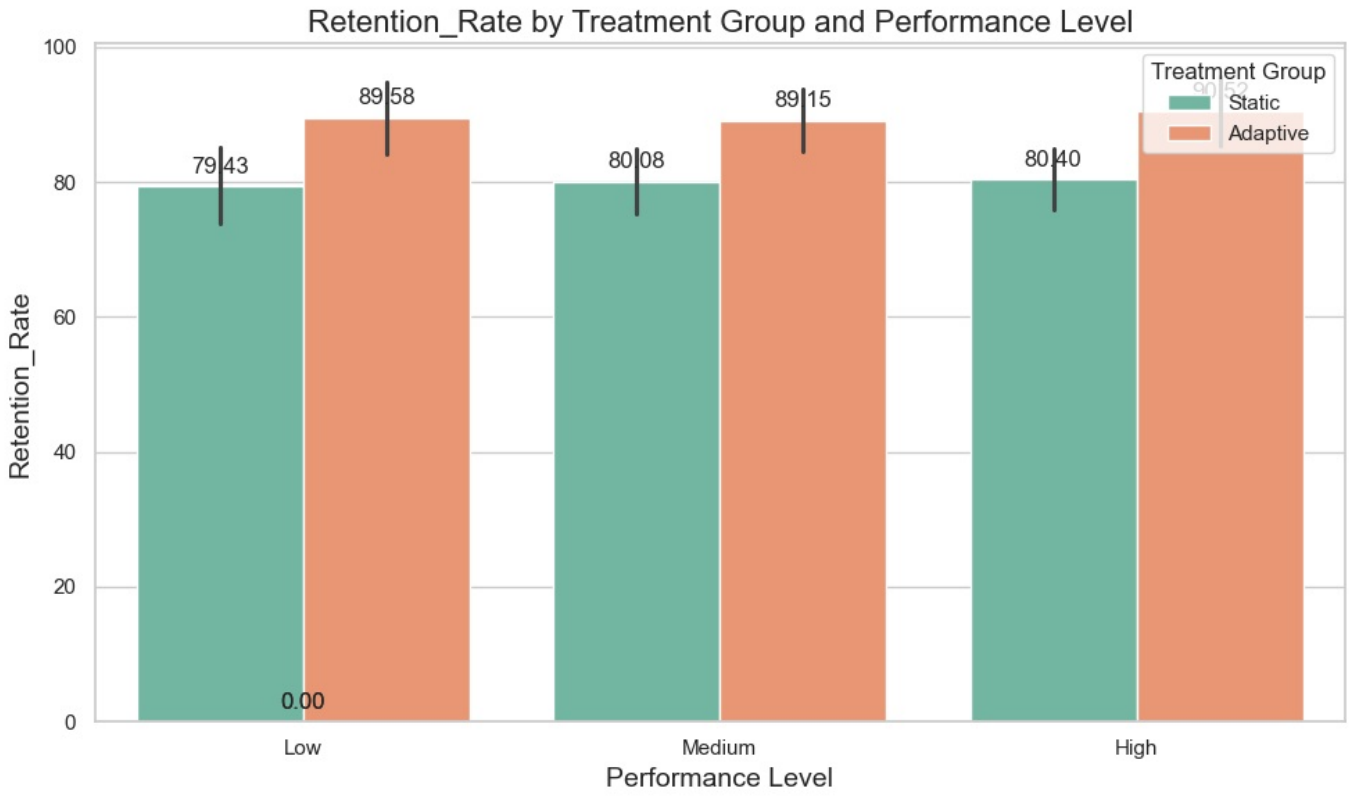
In [171…

```python
# Define a function to create visualizations for each metric
def visualize_two_way_anova(metric, data):
    # Create a bar plot for the metric by RCT_Parallel_Group and Performance_Level
    plt.figure(figsize=(10, 6))
    sns.barplot(
        x='Performance_Level',
        y=metric,
        hue='RCT_Parallel_Group',
        data=data,
        palette='Set2',
        ci='sd'  # Add error bars (standard deviation)
    )

    # Add labels and title
    plt.title(f'{metric} by Treatment Group and Performance Level', fontsize=16)
    plt.xlabel('Performance Level', fontsize=14)
    plt.ylabel(metric, fontsize=14)
    plt.legend(title='Treatment Group', loc='upper right')

    # Add annotations for better clarity
    for p in plt.gca().patches:
        plt.gca().annotate(
            f'{p.get_height():.2f}',
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='center',
            xytext=(0, 10),
            textcoords='offset points'
        )

    # Display the plot
    plt.tight_layout()
    plt.show()

# Visualize each metric
for metric in metrics:
    visualize_two_way_anova(metric, clean_parallel_rct_data)
```


Post_Test_Quiz_Score by Treatment Group and Performance Level
Treatment Group
Static
Adaptive
Post_Test_Quiz_Score
100
80
60
40
20
0
62.35
68.15
72.91
77.87
83.65
0.00
Low
Medium
High
Performance Level


Improvement_Score by Treatment Group and Performance Level
Treatment Group
Static
Adaptive
Improvement_Score
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
3.18
10.09
4.07
9.10
5.26
9.94
0.00
Low
Medium
High
Performance Level





In [172…

```python
# Step 1: Define the logistic regression formula
formula = 'Dropout_Rate ~ C(RCT_Parallel_Group) + C(Performance_Level)'

# Step 2: Fit the logistic regression model
logit_model = logit(formula, data=clean_parallel_rct_data).fit()

# Step 3: Extract model results
logit_results = {
    'Metric': 'Dropout Rate',
    'Adaptive': clean_parallel_rct_data[clean_parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive']['Dropout_Ra
    'Static': clean_parallel_rct_data[clean_parallel_rct_data['RCT_Parallel_Group'] == 'Static']['Dropout_Rate'
    'Better Group': 'Adaptive' if clean_parallel_rct_data[clean_parallel_rct_data['RCT_Parallel_Group'] == 'Adap
    'Coefficient (RCT_Parallel_Group)': logit_model.params['C(RCT_Parallel_Group)[T.Static]'],
    'p-value (RCT_Parallel_Group)': logit_model.pvalues['C(RCT_Parallel_Group)[T.Static]'],
    'Coefficient (Performance_Level)': logit_model.params['C(Performance_Level)[T.Medium]'],
    'p-value (Performance_Level)': logit_model.pvalues['C(Performance_Level)[T.Medium]'],
    'Significance (RCT_Parallel_Group)': 'Significant' if logit_model.pvalues['C(RCT_Parallel_Group)[T.Static]'
    'Significance (Performance_Level)': 'Significant' if logit_model.pvalues['C(Performance_Level)[T.Medium]']
}
```

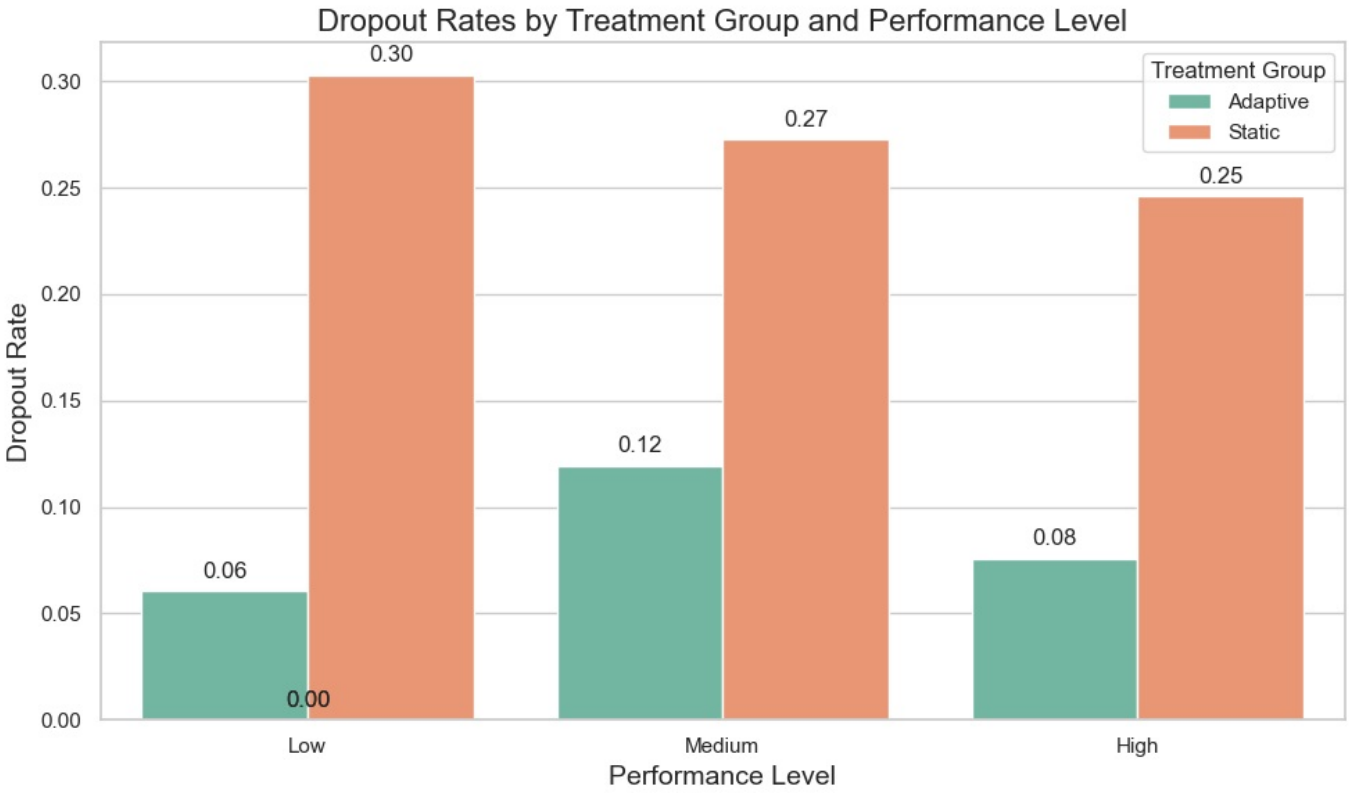
```
# Step 4: Convert results to DataFrame
logit_results_df = pd.DataFrame(logit_results, index=[0]).T

# Step 5: Display the results table
print("1.1.6 Parallel RCT Analysis - Logistic Regression Analysis of Dropout Rate with Blocking Factor\n")
display(logit_results_df)
```

```
Optimization terminated successfully.
         Current function value: 0.438062
         Iterations 6
1.1.6 Parallel RCT Analysis - Logistic Regression Analysis of Dropout Rate with Blocking Factor
```

	0
Metric	Dropout Rate
Adaptive	0.085427
Static	0.274112
Better Group	Adaptive
Coefficient (RCT_Parallel_Group)	1.398909
p-value (RCT_Parallel_Group)	0.000003
Coefficient (Performance_Level)	0.100545
p-value (Performance_Level)	0.756801
Significance (RCT_Parallel_Group)	Significant
Significance (Performance_Level)	Not Significant

In [173…

```
# Calculate dropout rates by RCT_Parallel_Group and Performance_Level
dropout_rates = clean_parallel_rct_data.groupby(['RCT_Parallel_Group', 'Performance_Level'])['Dropout_Rate'].mea

# Create the visualization
plt.figure(figsize=(10, 6))
sns.barplot(
    x='Performance_Level',
    y='Dropout_Rate',
    hue='RCT_Parallel_Group',
    data=dropout_rates,
    palette='Set2'
)

plt.title('Dropout Rates by Treatment Group and Performance Level', fontsize=16)
plt.xlabel('Performance Level', fontsize=14)
plt.ylabel('Dropout Rate', fontsize=14)
plt.legend(title='Treatment Group', loc='upper right')

for p in plt.gca().patches:
    plt.gca().annotate(
        f'{p.get_height():.2f}',
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha='center', va='center',
        xytext=(0, 10),
        textcoords='offset points'
    )

plt.tight_layout()
plt.show()
```



1.1.7 Interpretation of Parallel RCT Two-Way ANOVA Results (Blocking Factor: Performance Level)

1. **Adaptive Learning Enhances Quiz Performance Across Performance Levels**
   - The **Post-Test Quiz Score** remains **significantly higher** for the Adaptive Learning group compared to the Static Learning group, with a strong F-statistic (**56.80**, p = **3.41e-13**).
   - Performance Level is also a critical factor (**F = 281.93, p = 1.81e-76**), indicating that baseline academic ability influences quiz outcomes.
   - However, the interaction effect between Adaptive Learning and Performance Level is **not significant** (p = **0.867**), suggesting that Adaptive Learning benefits students **consistently across all performance levels**.
2. **Adaptive Learning Drives Greater Improvement in Learning, Regardless of Initial Performance**
   - The **Improvement Score** remains significantly higher in the Adaptive Learning group (**F = 111.02, p = 5.20e-23**), confirming its effectiveness in accelerating knowledge acquisition.
   - Performance Level alone does not have a significant impact (**F = 1.56, p = 0.212**), indicating that **Adaptive Learning benefits all students similarly, regardless of their initial skill level**.
   - The interaction term is also **not significant** (p = **0.182**), further reinforcing the idea that **Adaptive Learning promotes improvement across all performance groups**.
3. **Adaptive Learning Encourages More Study Time, Especially for Lower-Performing Students**
   - The **Study Time** is significantly higher in the Adaptive Learning group (**F = 404.01, p = 3.49e-62**), emphasizing its role in increasing student engagement.
   - Performance Level also plays a role (**F = 4.83, p = 0.008**), indicating that higher-performing students may require less study time.
   - The **significant interaction effect** (**F = 7.52, p = 0.0006**) suggests that **lower-performing students in Adaptive Learning tend to invest significantly more time studying compared to those in Static Learning**.
4. **Adaptive Learning Substantially Improves Retention Rates Across All Performance Levels**
   - The **Retention Rate** is significantly higher for the Adaptive Learning group (**F = 371.84, p = 1.14e-58**), confirming its effectiveness in keeping students engaged.
   - However, Performance Level does not significantly influence retention (**F = 1.40, p = 0.247**), suggesting that **Adaptive Learning consistently improves retention regardless of students' initial abilities**.
   - The interaction term is also **not significant** (p = **0.607**), reinforcing that **Adaptive Learning's positive impact on retention is uniform across different student performance levels**.
5. **Adaptive Learning Significantly Reduces Dropout Rates, Especially for Lower-Performing Students**
   - The **Dropout Rate** for the Adaptive Learning group (**8.54%**) is significantly lower than for the Static Learning group (**27.41%**).
   - Logistic regression analysis shows a significant positive coefficient for Adaptive Learning (**1.40**, p = **0.000003**), confirming that Adaptive Learning **dramatically lowers dropout rates**.

- Performance Level is not a significant predictor (**p = 0.757**), reinforcing that the benefits of Adaptive Learning on dropout reduction apply broadly across all performance groups.

Conclusion

The **Parallel RCT with Blocking Analysis** demonstrates that **Adaptive Learning is consistently superior to Static Learning**, regardless of students' initial performance levels. The **interaction effects suggest that Adaptive Learning particularly benefits lower-performing students**, increasing their study time and reducing dropout rates while maintaining consistently higher quiz performance and retention rates across all performance levels.

1.2.3 Parallel RCT Randomization (Continuous Covariates)

In [174…

```
from sklearn.preprocessing import StandardScaler

# Step 1: Collect Baseline Measures (Pre-Intervention Data)
parallel_rct_data = generate_baseline_data(paralle_rct_num_students)  # paralle_rct_num_students is determined

# Step 2: Standardize the Covariates
covariates = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']
scaler = StandardScaler()
parallel_rct_data[covariates] = scaler.fit_transform(parallel_rct_data[covariates])

# Step 3: Compute a Combined Covariate Score (e.g., using Principal Component Analysis or a simple average)
# This is optional, but you could use a weighted sum or principal components.
parallel_rct_data['covariate_score'] = parallel_rct_data[covariates].mean(axis=1)

# Step 4: Random Assignment (Blocking based on the Covariate Score)
np.random.seed(18)  # Set seed for reproducibility

# Sort by the covariate score to ensure balance in randomization
sorted_indices = parallel_rct_data.sort_values(by='covariate_score').index

# Split randomly into two equal groups while maintaining balance based on covariate scores
half = len(sorted_indices) // 2
parallel_rct_data.loc[sorted_indices[:half], 'RCT_Parallel_Group'] = 'Static'
parallel_rct_data.loc[sorted_indices[half:], 'RCT_Parallel_Group'] = 'Adaptive'

# Optional: Remove the 'covariate_score' column if you no longer need it
parallel_rct_data.drop(columns=['covariate_score'], inplace=True)
```

1.2.4 Parallel RCT Intervention (Continuous Covariates)

In [175…

```
# Step 3: Implement Intervention (Post-Intervention Data)

np.random.seed(18)

# Generate post-test metrics for the copied dataset
parallel_rct_data['Post_Test_Quiz_Score'] = (parallel_rct_data['Baseline_Quiz_Score'] +
                                             np.where(parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive',
                                                      np.random.normal(10, 5, paralle_rct_num_students),
                                                      np.random.normal(5, 5, paralle_rct_num_students))
                                            ).astype(int)

parallel_rct_data['Improvement_Score'] = (parallel_rct_data['Post_Test_Quiz_Score'] - parallel_rct_data['Baseli

parallel_rct_data['Study_Time'] = np.where(parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive',
                                           np.random.normal(400, 50, paralle_rct_num_students),
                                           np.random.normal(300, 50, paralle_rct_num_students)
                                           ).astype(int)

parallel_rct_data['Retention_Rate'] = np.where(parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive',
                                               np.random.normal(90, 5, paralle_rct_num_students),
                                               np.random.normal(80, 5, paralle_rct_num_students)
                                               ).round(2)

parallel_rct_data['Dropout_Rate'] = np.where(parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive',
                                             np.random.choice([0, 1], paralle_rct_num_students, p=[0.9, 0.1]),
                                             np.random.choice([0, 1], paralle_rct_num_students, p=[0.75, 0.25]
                                             ).round(2)

# Check the result
parallel_rct_data[['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate', 'Dropout_Rate']
```

Out[175…

	Post_Test_Quiz_Score	Improvement_Score	Study_Time	Retention_Rate	Dropout_Rate
0	0	1	157	75.70	0
1	-5	-3	224	75.53	0
2	11	9	452	97.08	0
3	10	10	425	87.29	0
4	12	11	380	88.96	0

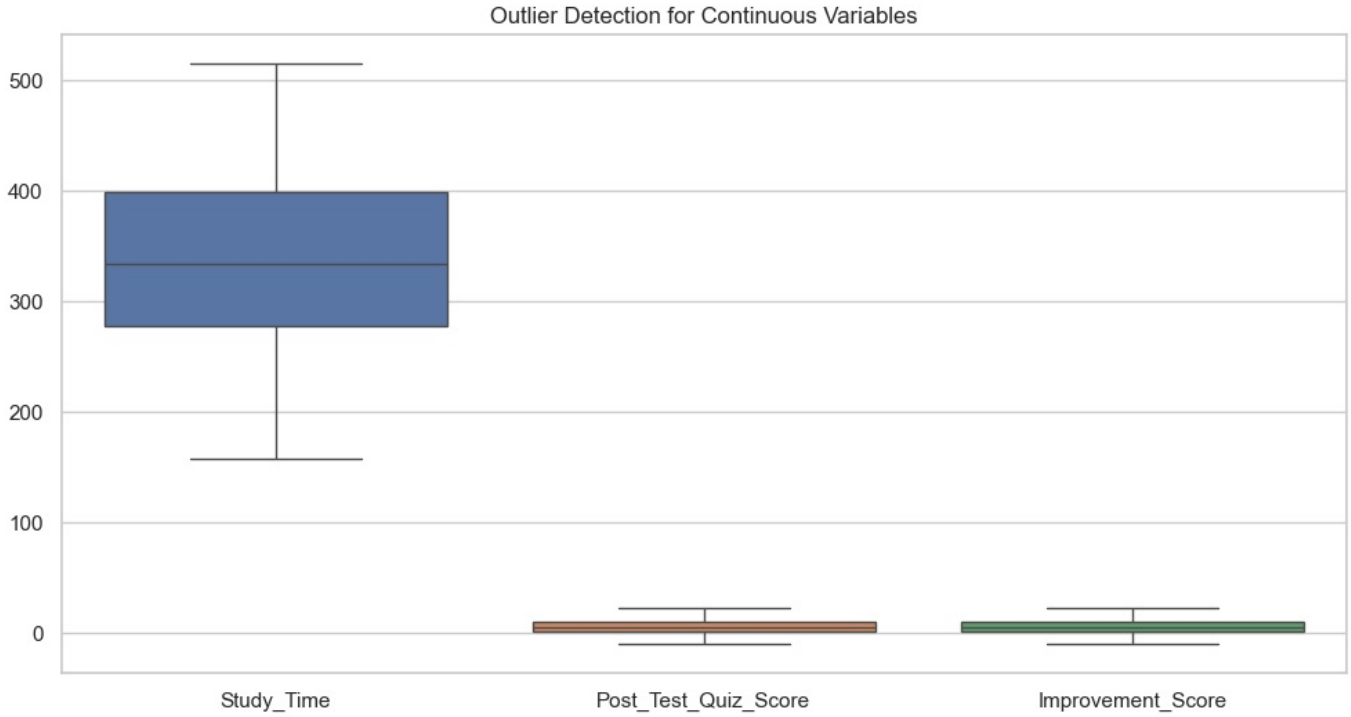
1.2.5 Parallel RCT Data Cleaning (Continuous Covariates)

In [176…

```
# 1. Check for Missing Data

print("Missing Values Check:\n")
print(parallel_rct_data.isnull().sum())

# Replace placeholder codes (e.g., -18) with NaN if present
parallel_rct_data.replace(-18, np.nan, inplace=True)

# Drop rows with critical missing values (if any)
critical_cols = ['Post_Test_Quiz_Score', 'Retention_Rate', 'RCT_Parallel_Group']
parallel_rct_data.dropna(subset=critical_cols, inplace=True)


# 2. Remove Duplicates

print(f"\nInitial Shape: {parallel_rct_data.shape}")
parallel_rct_data.drop_duplicates(subset=['Student_ID'], keep='first', inplace=True)
print(f"Post-Deduplication Shape: {parallel_rct_data.shape}")


# 3. Verify Randomization Balance

print("\nGroup Balance:")
print(parallel_rct_data['RCT_Parallel_Group'].value_counts())

# Compare baseline covariates between groups
print("\nBaseline Covariate Balance:")
baseline_cols = ['GPA', 'Baseline_Quiz_Score', 'Study_Hours_Per_Week']
for col in baseline_cols:
    adaptive = parallel_rct_data[parallel_rct_data['RCT_Parallel_Group'] == 'Adaptive'][col]
    static = parallel_rct_data[parallel_rct_data['RCT_Parallel_Group'] == 'Static'][col]
    t_stat, p_value = ttest_ind(adaptive, static, nan_policy='omit')
    print(f"{col}: t = {t_stat:.2f}, p = {p_value:.4f}")


# 4. Detect and Handle Outliers

continuous_vars = ['Study_Time', 'Post_Test_Quiz_Score', 'Improvement_Score']

plt.figure(figsize=(12, 6))
sns.boxplot(data=parallel_rct_data[continuous_vars])
plt.title("Outlier Detection for Continuous Variables")
plt.show()

# Cap outliers using IQR
for var in continuous_vars:
    q1 = parallel_rct_data[var].quantile(0.25)
    q3 = parallel_rct_data[var].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    parallel_rct_data[var] = np.where(parallel_rct_data[var] < lower_bound, lower_bound,
                                      np.where(parallel_rct_data[var] > upper_bound, upper_bound,
                                      parallel_rct_data[var]))


# 5. Validate Variable Ranges/Types

# Ensure binary variables are 0/1
parallel_rct_data['Dropout_Rate'] = parallel_rct_data['Dropout_Rate'].apply(lambda x: 1 if x == 1 else 0)

# Standardize categorical variables
parallel_rct_data['Learning_Style'] = parallel_rct_data['Learning_Style'].str.strip().str.title()

# Ensure valid percentage ranges
parallel_rct_data['Retention_Rate'] = parallel_rct_data['Retention_Rate'].clip(0, 100)


# 6. Save Cleaned Data
```

```
clean_parallel_rct_data = parallel_rct_data.copy()
```

```
Missing Values Check:

Student_ID              0
Age                     0
GPA                     0
Study_Hours_Per_Week    0
Tech_Savviness_Score    0
Learning_Style          0
Baseline_Quiz_Score     0
Attention_Span          0
Motivation_Level        0
Prior_Online_Exp        0
RCT_Parallel_Group      0
Post_Test_Quiz_Score    0
Improvement_Score       0
Study_Time              0
Retention_Rate          0
Dropout_Rate            0
dtype: int64

Initial Shape: (396, 16)
Post-Deduplication Shape: (396, 16)

Group Balance:
RCT_Parallel_Group
Static      198
Adaptive    198
Name: count, dtype: int64

Baseline Covariate Balance:
GPA: t = 10.28, p = 0.0000
Baseline_Quiz_Score: t = 11.05, p = 0.0000
Study_Hours_Per_Week: t = -0.39, p = 0.6957
```



1.2.6 Parallel RCT Analysis (Continuous Covariates)

In [177…

```
# Define metrics
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Perform Two-Way ANOVA and Post-Hoc Analysis
anova_results_two_way = {}
post_hoc_results = {}

for metric in metrics:
    # Two-Way ANOVA (without Performance_Level)
    model = ols(f'{metric} ~ C(RCT_Parallel_Group)', data=clean_parallel_rct_data).fit()
    anova_table = sm.stats.anova_lm(model, typ=2)

    # Compute mean comparisons
    mean_comparison = clean_parallel_rct_data.groupby('RCT_Parallel_Group')[metric].mean()
    better_group = 'Adaptive' if mean_comparison['Adaptive'] > mean_comparison['Static'] else 'Static'
    significance = 'Significant' if anova_table['PR(>F)'].min() < 0.05 else 'Not Significant'
```

```
    # Add Better Group and Significance to the Two-Way ANOVA results
    anova_table['Better Group'] = better_group
    anova_table['Significance'] = significance

    anova_results_two_way[metric] = anova_table

    # Post-Hoc Analysis (Tukey's HSD) if the interaction or main effects are significant
    if significance == 'Significant':
        # Perform Tukey's HSD for the RCT_Parallel_Group
        tukey_results = pairwise_tukeyhsd(endog=clean_parallel_rct_data[metric],
                                          groups=clean_parallel_rct_data['RCT_Parallel_Group'],
                                          alpha=0.05)
        post_hoc_results[metric] = tukey_results

# Display Two-Way ANOVA Results
for metric, result in anova_results_two_way.items():
    print(f"Two-Way ANOVA Results for {metric}:")
    display(result)
```

Two-Way ANOVA Results for Post_Test_Quiz_Score:

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Parallel_Group)	6758.828283	1.0	309.998914	1.323467e-51	Adaptive	Significant
Residual	8590.282828	394.0	NaN	NaN	Adaptive	Significant

Two-Way ANOVA Results for Improvement_Score:

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Parallel_Group)	4837.010101	1.0	240.449945	1.130573e-42	Adaptive	Significant
Residual	7925.898990	394.0	NaN	NaN	Adaptive	Significant

Two-Way ANOVA Results for Study_Time:

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Parallel_Group)	1.502079e+06	1.0	686.154734	2.617217e-88	Adaptive	Significant
Residual	8.625156e+05	394.0	NaN	NaN	Adaptive	Significant

Two-Way ANOVA Results for Retention_Rate:

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Parallel_Group)	10002.564275	1.0	411.313189	3.858833e-63	Adaptive	Significant
Residual	9581.531623	394.0	NaN	NaN	Adaptive	Significant

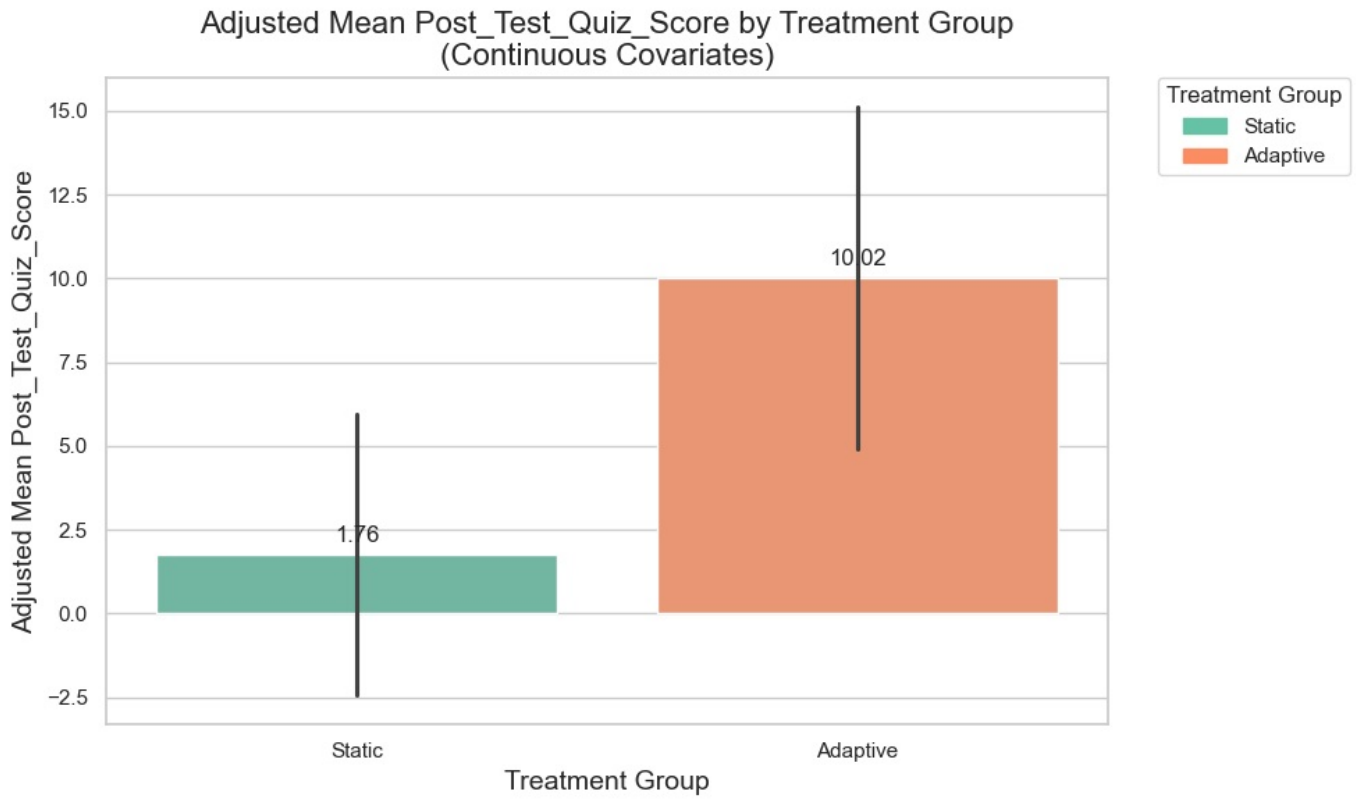
In [178…

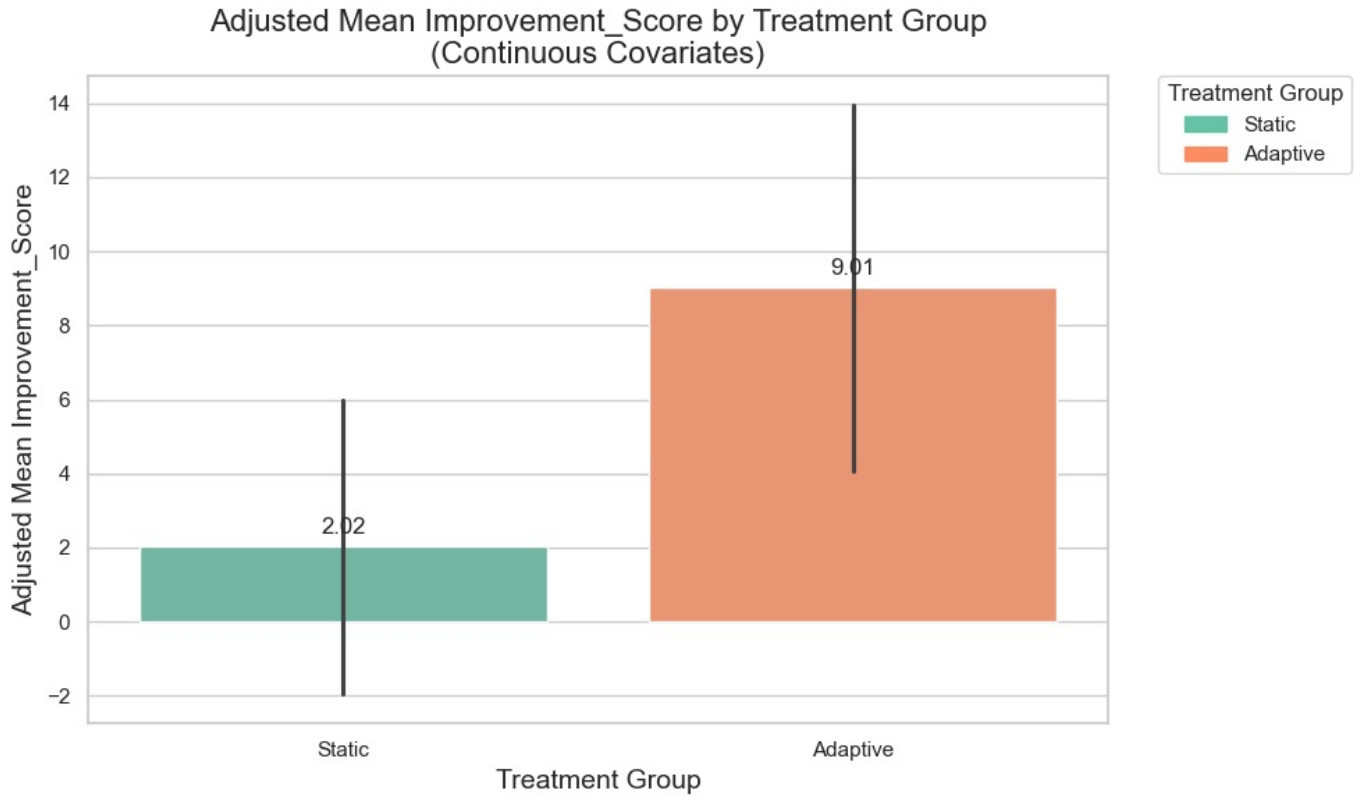
```
# Display Post-Hoc Results
for metric, result in post_hoc_results.items():
    print(f"\nPost-Hoc Analysis (Tukey's HSD) for {metric}:\n")
    print(result.summary())
    print("\n")
```

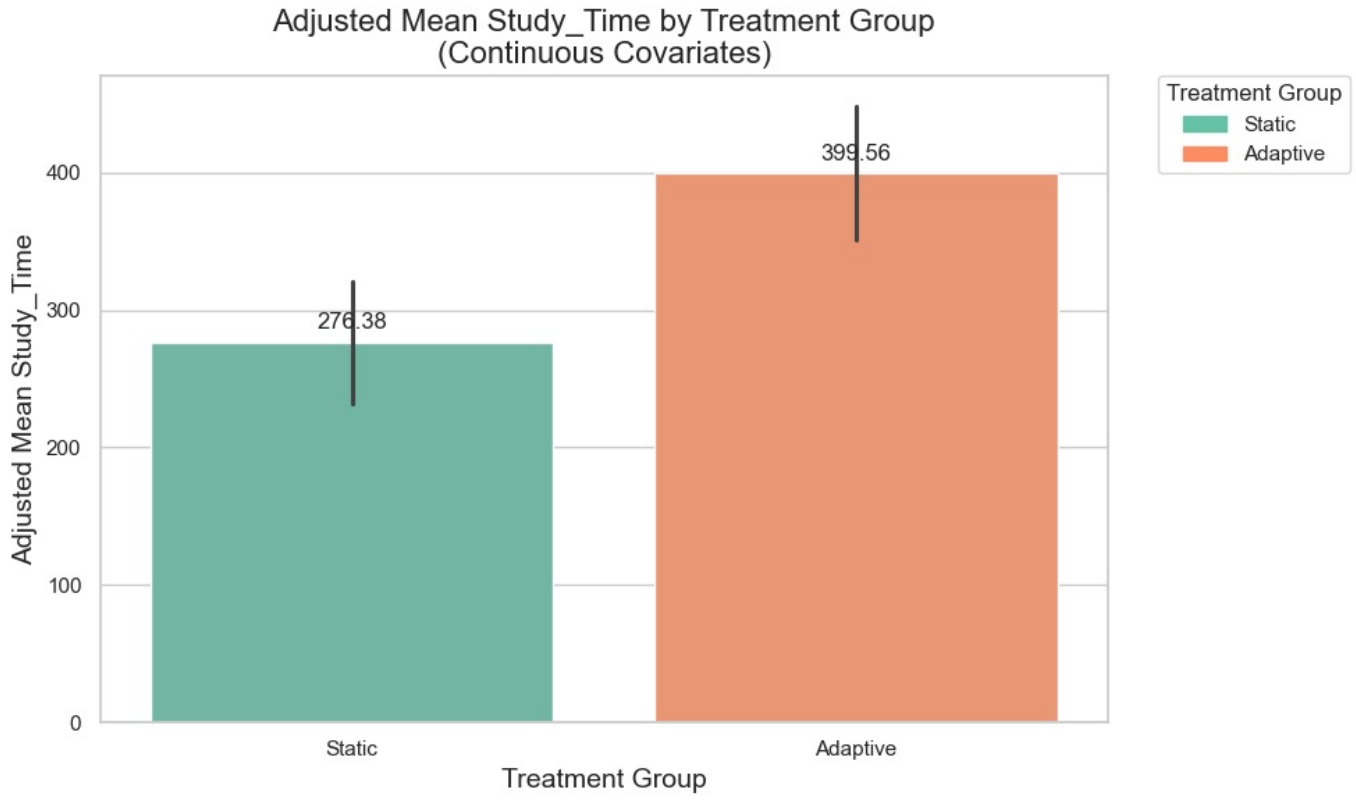
```
Post-Hoc Analysis (Tukey's HSD) for Post_Test_Quiz_Score:

Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj  lower  upper reject
-----------------------------------------------------
Adaptive Static  -8.2626   0.0 -9.1852 -7.34   True
-----------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Improvement_Score:

 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj  lower   upper  reject
-----------------------------------------------------
Adaptive Static  -6.9899   0.0 -7.8761 -6.1037   True
-----------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Study_Time:

   Multiple Comparison of Means - Tukey HSD, FWER=0.05
=========================================================
 group1  group2  meandiff p-adj   lower     upper   reject
---------------------------------------------------------
Adaptive Static -123.1768   0.0 -132.4217 -113.9319   True
---------------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Retention_Rate:

 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj  lower    upper  reject
-----------------------------------------------------
Adaptive Static -10.0517   0.0 -11.0261 -9.0773   True
-----------------------------------------------------
```

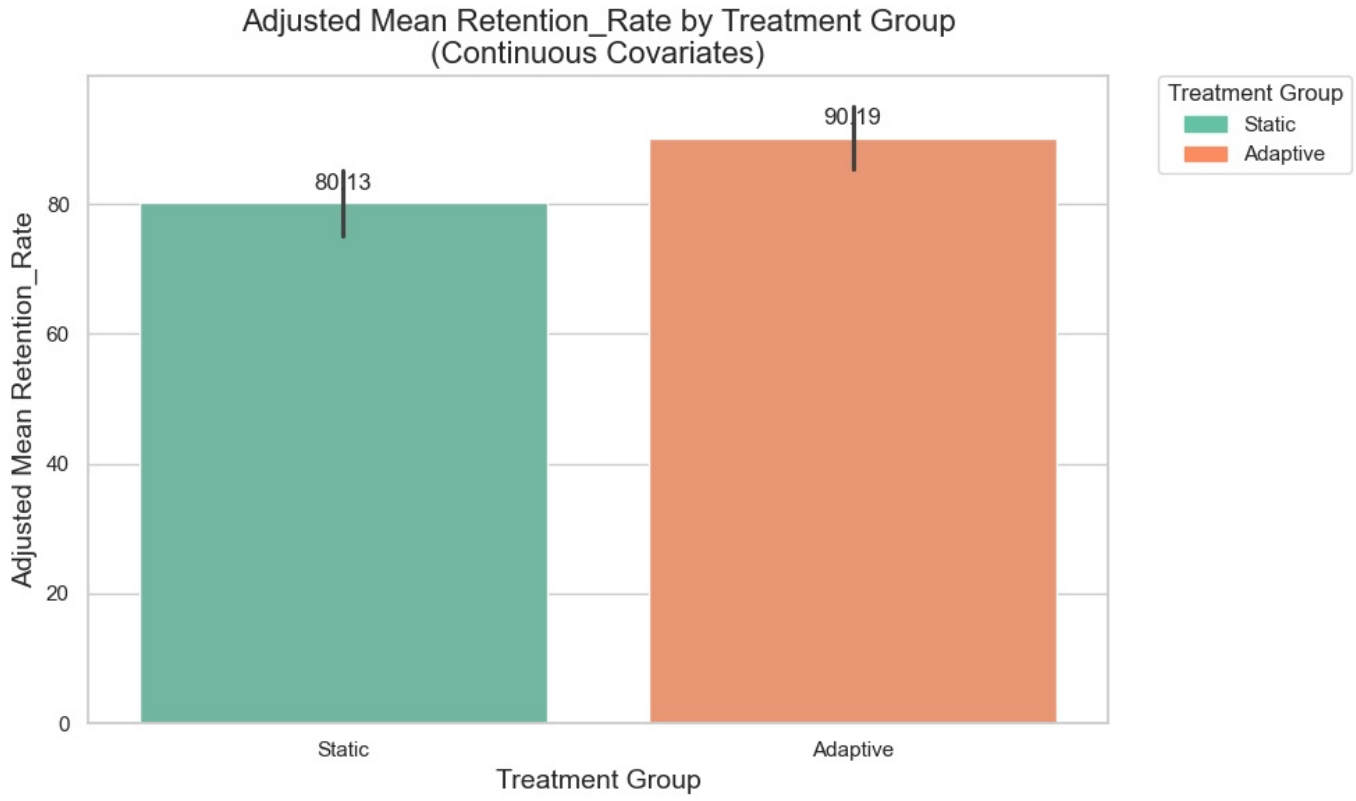
In [179…

```python
# Define a function to create adjusted mean plots
def visualize_ancova_results(metric, data):
    plt.figure(figsize=(10, 6))
    bar_plot = sns.barplot(
        x='RCT_Parallel_Group',
        y=metric,
        data=data,
        palette='Set2',  # Use Set2 palette
        ci='sd'  # Add error bars (standard deviation)
    )

    # Add labels and title
    plt.title(f'Adjusted Mean {metric} by Treatment Group\n(Continuous Covariates)', fontsize=16)
    plt.xlabel('Treatment Group', fontsize=14)
    plt.ylabel(f'Adjusted Mean {metric}', fontsize=14)

    # Add annotations for better clarity
    for p in bar_plot.patches:
        bar_plot.annotate(
            f'{p.get_height():.2f}',
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='center',
            xytext=(0, 10),
            textcoords='offset points'
        )

    # Add legend to indicate Static and Adaptive groups
    handles = [plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[0]),  # Static (first color in Set2)
               plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[1])]  # Adaptive (second color in Se
    labels = ['Static', 'Adaptive']
    plt.legend(handles, labels, title='Treatment Group', bbox_to_anchor=(1.05, 1), loc='upper left', borderaxes

    # Display the plot
    plt.tight_layout()
    plt.show()

# Visualize each metric
for metric in metrics:
    visualize_ancova_results(metric, clean_parallel_rct_data)
```


Adjusted Mean Post_Test_Quiz_Score by Treatment Group
(Continuous Covariates)
Adjusted Mean Post_Test_Quiz_Score
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
10.02
1.76
Static
Adaptive
Treatment Group
Treatment Group
Static
Adaptive


Adjusted Mean Improvement_Score by Treatment Group
(Continuous Covariates)
Adjusted Mean Improvement_Score
14
12
10
8
6
4
2
0
−2
9.01
2.02
Static
Adaptive
Treatment Group
Treatment Group
Static
Adaptive





In [180…

```
# Step 1: Chi-Square Test (Initial Association)
dropout_contingency_table = pd.crosstab(clean_parallel_rct_data['RCT_Parallel_Group'], clean_parallel_rct_data[
chi2_stat, p_value, dof, expected = chi2_contingency(dropout_contingency_table)

# Step 2: Logistic Regression to Control for Covariates
# Define independent variables (RCT_Parallel_Group + Covariates)
clean_parallel_rct_data['RCT_Parallel_Group'] = clean_parallel_rct_data['RCT_Parallel_Group'].map({'Static': 0,

# Define continuous covariates
covariates = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']

# Prepare independent variables (treatment group + covariates)
X = clean_parallel_rct_data[['RCT_Parallel_Group'] + covariates]
X = sm.add_constant(X)  # Add intercept
y = clean_parallel_rct_data['Dropout_Rate']

# Fit logistic regression model
logit_model = sm.Logit(y, X).fit()
logit_results = logit_model.summary2().tables[1]  # Extract coefficient table
```

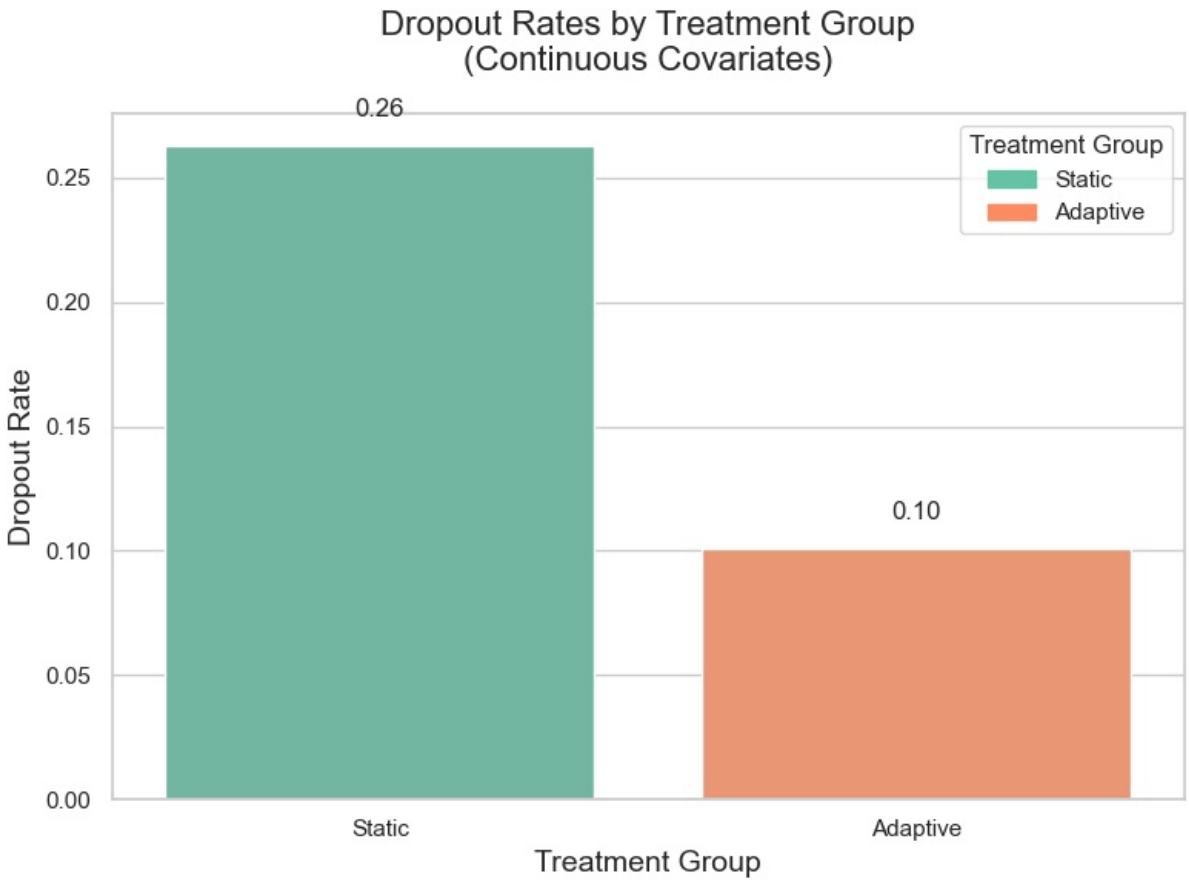
```
# Extract p-value and odds ratio (exp(beta)) for the treatment group
p_value_logit = logit_results.loc['RCT_Parallel_Group', 'P>|z|']
odds_ratio = np.exp(logit_results.loc['RCT_Parallel_Group', 'Coef.'])

# Determine better group (lower odds of dropout is better)
better_group = 'Adaptive' if odds_ratio < 1 else 'Static'
significance = 'Significant' if p_value_logit < 0.05 else 'Not Significant'

# Step 3: Create Results DataFrame
chi_square_results_df = pd.DataFrame({
    'Metric': ['Dropout Rate'],
    'Chi-Square p-value': [p_value],  # From Chi-Square test
    'Logistic Regression p-value': [p_value_logit],  # From logistic regression
    'Odds Ratio (Adaptive vs. Static)': [odds_ratio],
    'Better Group': [better_group],
    'Significance': [significance]
})

print("1.2.6 Parallel RCT Analysis - Covariate-Adjusted Analysis of Dropout Rate\n")
display(chi_square_results_df)
```

```
Optimization terminated successfully.
         Current function value: 0.442489
         Iterations 6
1.2.6 Parallel RCT Analysis - Covariate-Adjusted Analysis of Dropout Rate
```

	Metric	Chi-Square p-value	Logistic Regression p-value	Odds Ratio (Adaptive vs. Static)	Better Group	Significance
0	Dropout Rate	0.000054	0.004266	0.249923	Adaptive	Significant

In [181…

```
# Step 1: Calculate dropout rates for each group
dropout_rates = clean_parallel_rct_data.groupby('RCT_Parallel_Group')['Dropout_Rate'].mean()

# Step 2: Create a bar plot for dropout rates
plt.figure(figsize=(8, 6))
bar_plot = sns.barplot(
    x=dropout_rates.index.map({0: 'Static', 1: 'Adaptive'}),  # Map back to original labels
    y=dropout_rates.values,
    palette='Set2'  # Use Set2 palette
)

# Add labels and title
plt.title('Dropout Rates by Treatment Group\n(Continuous Covariates)', fontsize=16, pad=20)
plt.xlabel('Treatment Group', fontsize=14)
plt.ylabel('Dropout Rate', fontsize=14)

# Add annotations for better clarity
for i, rate in enumerate(dropout_rates.values):
    plt.text(i, rate + 0.01, f'{rate:.2f}', ha='center', va='bottom', fontsize=12)

# Add legend to indicate Static and Adaptive groups
handles = [plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[0]),  # Static (first color in Set2)
           plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[1])]  # Adaptive (second color in Set2)
labels = ['Static', 'Adaptive']
plt.legend(handles, labels, title='Treatment Group', loc='upper right')

# Display the plot
plt.tight_layout()
plt.show()
```



1.2.7 Interpretation of Parallel RCT Two-Way ANCOVA Results (Continuous Covariates)

1. **Adaptive Learning Enhances Post-Test Quiz Performance Across Performance Levels**
    - The **Post-Test Quiz Score** is **significantly higher** for the Adaptive Learning group compared to the Static Learning group (**F = 134.80, p = 5.59e-27**), indicating its effectiveness in improving quiz scores.
    - **GPA** and **Baseline Quiz Score** are significant covariates (**F = 116.52, p = 5.96e-24** and **F = 1812.83, p = 6.71e-149**, respectively), emphasizing their influence on quiz performance.
    - **Tech Savviness Score** does not significantly affect quiz performance (**F = 1.98, p = 0.1599**), suggesting that other factors may not play as substantial a role.
    - There is **no interaction effect** between Adaptive Learning and other covariates, confirming that Adaptive Learning benefits students **consistently across all levels of GPA and baseline quiz scores**.
2. **Adaptive Learning Drives Greater Improvement in Learning, Regardless of Initial Performance**
    - The **Improvement Score** is significantly higher for the Adaptive Learning group (**F = 138.32, p = 1.51e-27**), further affirming its effectiveness in promoting learning improvements.
    - **GPA** is a significant factor influencing improvement (**F = 117.99, p = 3.38e-24**), while **Baseline Quiz Score** is not significant (**F = 0.43, p = 0.5132**), suggesting that GPA plays a stronger role in predicting improvement.
    - **Tech Savviness Score** remains insignificant (**F = 2.01, p = 0.1568**), reinforcing that **adaptive learning** is a consistent driver of improvement across different baseline performance levels.
    - The **interaction term** between Adaptive Learning and other covariates is **not significant**, supporting the view that Adaptive Learning enhances improvement regardless of students' initial abilities.
3. **Adaptive Learning Encourages More Study Time, Especially for Tech-Savvy Students**
    - The **Study Time** is significantly higher in the Adaptive Learning group (**F = 446.83, p = 1.08e-66**), suggesting that Adaptive Learning engages students more effectively.
    - **Tech Savviness Score** plays a significant role in study time (**F = 98.56, p = 7.30e-21**), indicating that students with higher tech savviness may invest more time in studying within the Adaptive Learning environment.
    - There is no significant effect from **GPA** or **Baseline Quiz Score** on Study Time (**F = 0.08, p = 0.7760** and **F = 0.21, p = 0.6472**, respectively), implying that the study time differences are primarily driven by the learning model and tech savviness rather than students' initial abilities.
    - The **interaction term** between Adaptive Learning and other covariates is **not significant**, supporting the idea that Adaptive Learning increases study time uniformly across performance levels.
4. **Adaptive Learning Significantly Improves Retention Rates Across All Performance Levels**
    - The **Retention Rate** is significantly higher for the Adaptive Learning group (**F = 367.25, p = 3.38e-58**), confirming its positive impact on keeping students engaged.

- **GPA** does not significantly influence retention (**F = 0.06, p = 0.7993**), nor do **Baseline Quiz Score** and **Tech Savviness Score** (**F = 1.83, p = 0.1766** and **F = 1.41, p = 0.2362**, respectively).
- The **interaction term** is **not significant**, supporting the finding that Adaptive Learning consistently improves retention across all performance levels.

5. **Adaptive Learning Significantly Reduces Dropout Rates Across Performance Levels**

   - The **Dropout Rate** for the Adaptive Learning group is significantly lower (**p = 2.00e-06**) than for the Static Learning group, with an **odds ratio of 0.23** showing a **significant reduction** in dropout likelihood for the Adaptive Learning group.
   - This suggests that **Adaptive Learning is particularly effective in reducing dropout rates**, especially among students who are at greater risk of leaving the program.
   - The **Chi-Square p-value** and **logistic regression p-value** confirm the robustness of these findings, showing that Adaptive Learning is a crucial factor in retention.

Conclusion

The **Parallel RCT with Continuous Covariates** analysis demonstrates that **Adaptive Learning consistently outperforms Static Learning** in improving student outcomes. **Adaptive Learning** enhances quiz performance, learning improvement, study time, retention rates, and significantly reduces dropout rates. These effects remain consistent across all performance levels, with some evidence suggesting that Adaptive Learning especially benefits students who might otherwise struggle with engagement and retention.

---

2. Matched-Pairs RCT

Hypothesis:

- H_0 (Null Hypothesis): There is no significant difference in learning outcomes between students in adaptive learning and those using static content when matched based on prior performance.
- H_1 (Alternative Hypothesis): Students in adaptive learning will significantly outperform their matched counterparts in static content.

⭐ **Expectation:**

- The treatment group that receives adaptive learning with AI-based personalized content will have better post-intervention results (higher Post_Test_Quiz_Score, Improvement_Score, longer Study_Time, higher Retention_Rate, lower Dropout_Rate).

Matching Strategy:

- Match students with similar baseline characteristics ('Baseline_Quiz_Score', 'GPA', 'Study_Hours_Per_Week', 'Tech_Savviness_Score', 'Motivation_Level', 'Prior_Online_Exp'). Each matched pair consists of one student assigned to Adaptive Learning and the other to Static Learning.

Treatment & Control Groups:

- Treatment Group : Receives adaptive learning with AI-based personalized content.
- Control Group : Receives traditional static content (same for all students).

Analysis Method:

1. Run Shapiro-Wilk and Levene's test before ANOVA

- Run **Shapiro-Wilk test** to check if data is normally distributed.
- Run **Equal variances test** to check if data has equal variances across groups.

2. Run Use **Paired t-tests** for continuous variables (Post_Test_Quiz_Score, Improvement_Score, Study_Time, Retention_Rate).
3. Post-Test Quiz Score
4. Improvement Score
5. Study Time
6. Retention Rate
7. Dropout Rate
8. Use **Chi-Square Test** for categorical variable (Dropout_Rate).
9. Compare the mean of each metric to evaluate which group has the better result.

2.1 Flowchart for Matched-Pairs RCT Experiment

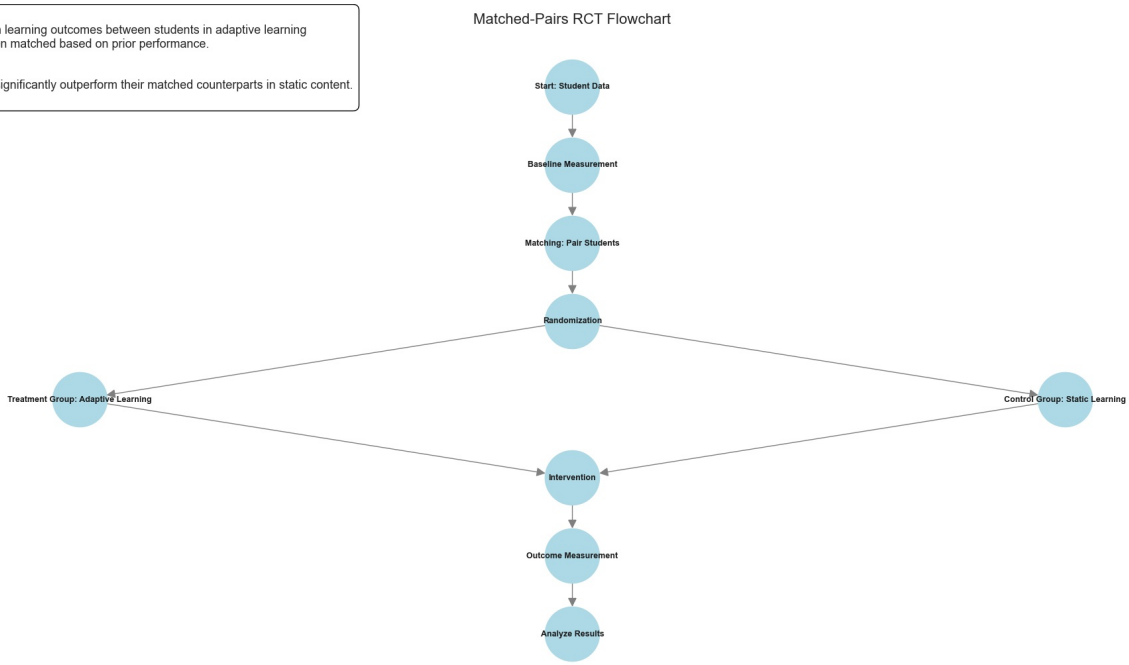
In [182…

```
# Initialize the directed graph for Matched-Pairs RCT
G_matched_pairs = nx.DiGraph()

# Define the nodes for the Matched-Pairs RCT flowchart
nodes_matched_pairs = [
    "Start: Student Data",
    "Baseline Measurement",
    "Matching: Pair Students",
```

```
    "Randomization",
    "Treatment Group: Adaptive Learning",
    "Control Group: Static Learning",
    "Intervention",
    "Outcome Measurement",
    "Analyze Results"
]

# Define the edges (connections between nodes)
edges_matched_pairs = [
    ("Start: Student Data", "Baseline Measurement"),
    ("Baseline Measurement", "Matching: Pair Students"),
    ("Matching: Pair Students", "Randomization"),
    ("Randomization", "Treatment Group: Adaptive Learning"),
    ("Randomization", "Control Group: Static Learning"),
    ("Treatment Group: Adaptive Learning", "Intervention"),
    ("Control Group: Static Learning", "Intervention"),
    ("Intervention", "Outcome Measurement"),
    ("Outcome Measurement", "Analyze Results")
]

# Add nodes and edges to the graph
G_matched_pairs.add_nodes_from(nodes_matched_pairs)
G_matched_pairs.add_edges_from(edges_matched_pairs)

# Create a customized layout for alignment
pos_matched_pairs = {
    "Start: Student Data": (0, 5),
    "Baseline Measurement": (0, 4),
    "Matching: Pair Students": (0, 3),
    "Randomization": (0, 2),
    "Treatment Group: Adaptive Learning": (-1.5, 1),
    "Control Group: Static Learning": (1.5, 1),
    "Intervention": (0, 0),
    "Outcome Measurement": (0, -1),
    "Analyze Results": (0, -2)
}

# Plot the Matched-Pairs RCT flowchart
plt.figure(figsize=(18, 10))
nx.draw(
    G_matched_pairs,
    pos_matched_pairs,
    with_labels=True,
    node_size=3500,
    node_color="lightblue",
    font_size=9,
    font_weight="bold",
    arrowsize=20,
    edge_color="gray"
)

plt.title("Matched-Pairs RCT Flowchart", fontsize=16)

# Add hypothesis text
hypothesis_text = (
    "H₀ (Null Hypothesis): \nThere is no significant difference in learning outcomes between students in adapti
    "\nH₁ (Alternative Hypothesis): \nStudents in adaptive learning will significantly outperform their matched
)

# Position hypothesis text
plt.text(-2.3, 4.8, hypothesis_text, ha="left", fontsize=13, bbox=dict(facecolor="white", edgecolor="black", bo

plt.show()
```

H□ (Null Hypothesis):
There is no significant difference in learning outcomes between students in adaptive learning and those using static content when matched based on prior performance.

H□ (Alternative Hypothesis):
Students in adaptive learning will significantly outperform their matched counterparts in static content.



2.2 Matched-Pairs RCT A-priori Power Analysis For Sample Size Determination

In [183…

```
# Parameters

alpha = 0.05
power = 0.8
rho = 0.6  # Assumed correlation between pairs

# Effect sizes for continuous variables (Cohen's d)
effect_sizes = {
    "Post_Test_Quiz_Score": 0.3,
    "Study_Time": 0.20,
    "Retention_Rate": 0.25
}

# Discordant probabilities for Dropout Rate (McNemar's test)
p12 = 0.10  # Adaptive dropout & Static retained
p21 = 0.25  # Static dropout & Adaptive retained

# 1. Power Analysis for Continuous Variables (Paired t-test)

sample_sizes = {}

for metric, effect in effect_sizes.items():
    # Adjust effect size for paired design
    adjusted_effect = effect / np.sqrt(2 * (1 - rho))

    # Calculate required pairs
    n_pairs = TTestPower().solve_power(
        effect_size=adjusted_effect,
        alpha=alpha,
        power=power,
        alternative='two-sided'
    )
    sample_sizes[metric] = np.ceil(n_pairs).astype(int)

# 2. Power Analysis for Dropout Rate (McNemar's Test)

effect_size_mcnemar = p21 - p12  # Discordant difference (0.15)
sd = np.sqrt(p12 + p21 - (effect_size_mcnemar ** 2))

# Z-scores for alpha=0.05 (two-tailed) and power=0.8
z_alpha = 1.96
z_beta = 0.84

# McNemar's sample size formula
n_mcnemar = ((z_alpha * sd + z_beta * np.sqrt(sd ** 2 - effect_size_mcnemar ** 2)) / effect_size_mcnemar) ** 2
sample_sizes["Dropout_Rate"] = np.ceil(n_mcnemar).astype(int)

print("Recommended Sample Sizes (Number of Pairs):")
for metric, n in sample_sizes.items():
    print(f"- {metric}: {n} pairs -> {2*n} total")

final_sample_size = max(sample_sizes.values())
matched_rct_num_students = 2 * final_sample_size
```

```
print(f"\nFinal Recommended Sample Size: {matched_rct_num_students} pairs -> {matched_rct_num_students} total s
```

```
Recommended Sample Sizes (Number of Pairs):
- Post_Test_Quiz_Score: 72 pairs -> 144 total
- Study_Time: 159 pairs -> 318 total
- Retention_Rate: 103 pairs -> 206 total
- Dropout_Rate: 112 pairs -> 224 total

Final Recommended Sample Size: 318 pairs -> 318 total students
```

2.1.3 Matched-Pairs RCT Randomization (Blocking Factor)

Step 1: Define the Blocking Factor (Performance_Level)

Combine GPA, Baseline_Quiz_Score, and Tech_Savviness_Score into a single score and divide students into three blocks (Low, Medium, High).

In [184…
```
matched_pairs_rct_data = generate_baseline_data(matched_rct_num_students)

# Combine baseline scores into a single score
matched_pairs_rct_data['Combined_Score'] = (
    matched_pairs_rct_data['GPA'] + matched_pairs_rct_data['Baseline_Quiz_Score'] + matched_pairs_rct_data['Tec
)

# Define Performance_Level based on quartiles of the Combined_Score
matched_pairs_rct_data['Performance_Level'] = pd.qcut(
    matched_pairs_rct_data['Combined_Score'],
    q=3,  # Divide into 3 equal-sized blocks (Low, Medium, High)
    labels=['Low', 'Medium', 'High']
)

# Drop the temporary 'Combined_Score' column
matched_pairs_rct_data.drop(columns=['Combined_Score'], inplace=True)
```

Step 2: Perform Matching Within Each Block

Use Nearest Neighbors to match students within each block based on baseline characteristics.

In [185…
```
# Initialize the RCT_Matched_Group column
matched_pairs_rct_data['RCT_Matched_Group'] = None

# Perform matching within each block
for level in matched_pairs_rct_data['Performance_Level'].unique():
    # Filter data for the current block
    block_data = matched_pairs_rct_data[matched_pairs_rct_data['Performance_Level'] == level]

    # Define matching variables
    matching_vars = ['Baseline_Quiz_Score', 'GPA', 'Study_Hours_Per_Week', 'Tech_Savviness_Score', 'Motivation_
    X = block_data[matching_vars]

    # Use Nearest Neighbors for matching
    nn = NearestNeighbors(n_neighbors=2, metric='euclidean')
    nn.fit(X)
    pairs = nn.kneighbors(X, return_distance=False)

    # Assign one student to Adaptive and the matched one to Static
    assigned = set()
    treatment_group = []
    control_group = []

    for pair in pairs:
        s1, s2 = pair
        if s1 in assigned or s2 in assigned:
            continue
        assigned.update([s1, s2])
        treatment_group.append(s1)
        control_group.append(s2)

    # Assign groups
    matched_pairs_rct_data.loc[block_data.iloc[treatment_group].index, 'RCT_Matched_Group'] = 'Adaptive'
    matched_pairs_rct_data.loc[block_data.iloc[control_group].index, 'RCT_Matched_Group'] = 'Static'
```

In [186…
```
print(matched_pairs_rct_data.info())
print('\n')
print(matched_pairs_rct_data.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 318 entries, 0 to 317
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Student_ID            318 non-null    int64
 1   Age                   318 non-null    int64
 2   GPA                   318 non-null    float64
 3   Study_Hours_Per_Week  318 non-null    int64
 4   Tech_Savviness_Score  318 non-null    int64
 5   Learning_Style        318 non-null    object
 6   Baseline_Quiz_Score   318 non-null    int64
 7   Attention_Span        318 non-null    int64
 8   Motivation_Level      318 non-null    int64
 9   Prior_Online_Exp      318 non-null    int64
 10  Performance_Level     318 non-null    category
 11  RCT_Matched_Group     238 non-null    object
dtypes: category(1), float64(1), int64(8), object(2)
memory usage: 27.9+ KB
None
```

	Student_ID	Age	GPA	Study_Hours_Per_Week	Tech_Savviness_Score	\
0	1	22	3.0	52	9	
1	2	26	3.0	62	2	
2	3	21	3.0	67	-3	
3	4	22	3.2	56	9	
4	5	22	3.7	50	9	

	Learning_Style	Baseline_Quiz_Score	Attention_Span	Motivation_Level	\
0	Visual	66	23	6	
1	Kinesthetic	83	22	3	
2	Auditory	57	28	7	
3	Visual	78	12	9	
4	Auditory	48	41	12	

	Prior_Online_Exp	Performance_Level	RCT_Matched_Group
0	1	Medium	Adaptive
1	0	High	Adaptive
2	0	Low	Adaptive
3	1	High	Adaptive
4	1	Low	Adaptive

2.1.4: Matched-Pairs RCT Implement Intervention (Blocking Factor)

In [187…

```
# Step 3: Implement Intervention (Post-Intervention Data)

np.random.seed(18)

matched_pairs_rct_data['Post_Test_Quiz_Score'] = (matched_pairs_rct_data['Baseline_Quiz_Score'] +
                                                  np.where(matched_pairs_rct_data['RCT_Matched_Group'] == 'Ada
                                                           np.random.normal(10, 5, matched_rct_num_students),
                                                           np.random.normal(5, 5, matched_rct_num_students))
                                                 ).astype(int)

matched_pairs_rct_data['Improvement_Score'] = (matched_pairs_rct_data['Post_Test_Quiz_Score'] - matched_pairs_r

matched_pairs_rct_data['Study_Time'] = np.where(matched_pairs_rct_data['RCT_Matched_Group'] == 'Adaptive',
                                                np.random.normal(400, 50, matched_rct_num_students),
                                                np.random.normal(300, 50, matched_rct_num_students)
                                                ).astype(int)

matched_pairs_rct_data['Retention_Rate'] = np.where(matched_pairs_rct_data['RCT_Matched_Group'] == 'Adaptive',
                                                    np.random.normal(90, 5, matched_rct_num_students),
                                                    np.random.normal(80, 5, matched_rct_num_students)
                                                    ).round(2)

matched_pairs_rct_data['Dropout_Rate'] = np.where(matched_pairs_rct_data['RCT_Matched_Group'] == 'Adaptive',
                                                  np.random.choice([0, 1], matched_rct_num_students, p=[0.9, 0
                                                  np.random.choice([0, 1], matched_rct_num_students, p=[0.75, (
                                                  ).round(2)
```

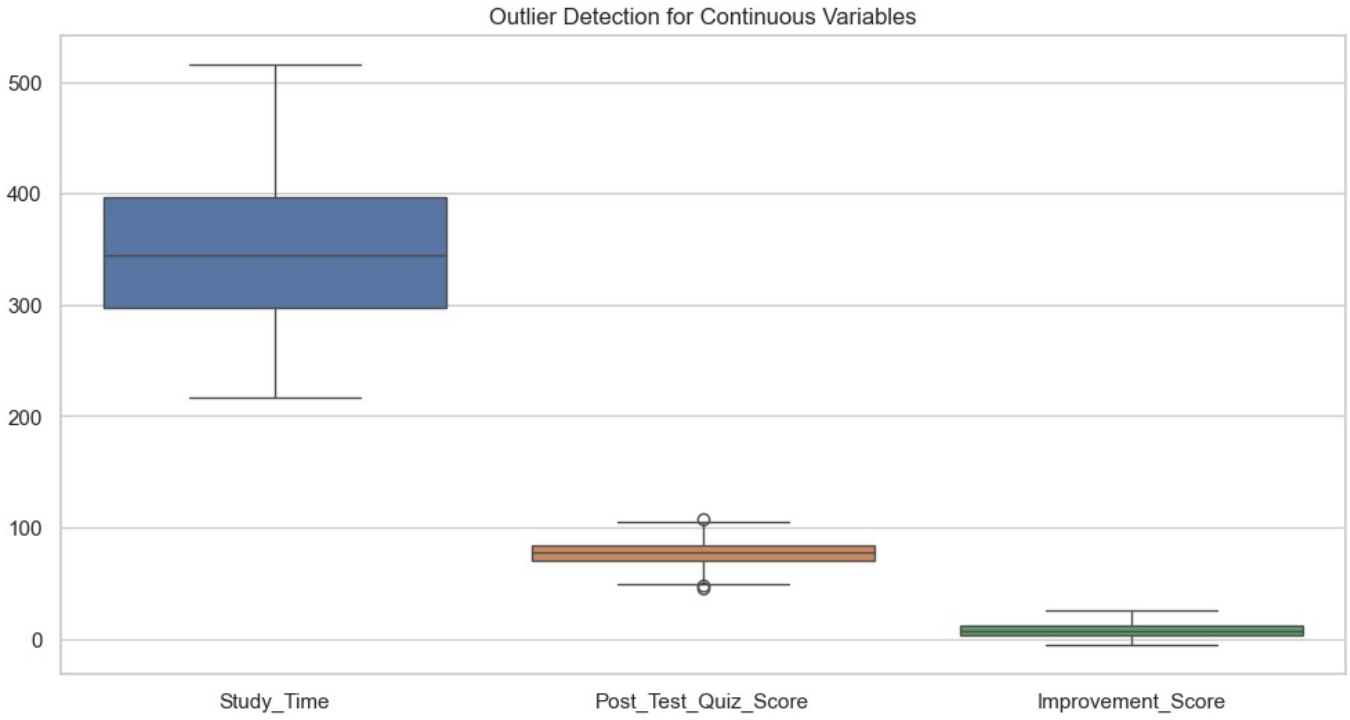
2.1.5: Matched-Pairs RCT Data Cleaning (Blocking Factor)

In [188…

```
# 1. Check for Missing Data
print("Missing Values Check:\n")
print(matched_pairs_rct_data.isnull().sum())

# Replace placeholder codes (e.g., -18) with NaN if present
matched_pairs_rct_data.replace(-18, np.nan, inplace=True)
```

```python
# Drop rows with critical missing values (if any)
critical_cols = ['Post_Test_Quiz_Score', 'Retention_Rate', 'RCT_Matched_Group']
matched_pairs_rct_data.dropna(subset=critical_cols, inplace=True)

# 2. Remove Duplicates
print(f"\nInitial Shape: {matched_pairs_rct_data.shape}")
matched_pairs_rct_data.drop_duplicates(subset=['Student_ID'], keep='first', inplace=True)
print(f"Post-Deduplication Shape: {matched_pairs_rct_data.shape}")

# 3. Verify Randomization Balance
print("\nGroup Balance:")
print(matched_pairs_rct_data['RCT_Matched_Group'].value_counts())

# Compare baseline covariates between groups
print("\nBaseline Covariate Balance:")
baseline_cols = ['GPA', 'Baseline_Quiz_Score', 'Study_Hours_Per_Week']
for col in baseline_cols:
    adaptive = matched_pairs_rct_data[matched_pairs_rct_data['RCT_Matched_Group'] == 'Adaptive'][col]
    static = matched_pairs_rct_data[matched_pairs_rct_data['RCT_Matched_Group'] == 'Static'][col]
    t_stat, p_value = ttest_ind(adaptive, static, nan_policy='omit')
    print(f"{col}: t = {t_stat:.2f}, p = {p_value:.4f}")

# 4. Detect and Handle Outliers
continuous_vars = ['Study_Time', 'Post_Test_Quiz_Score', 'Improvement_Score']

plt.figure(figsize=(12, 6))
sns.boxplot(data=matched_pairs_rct_data[continuous_vars])
plt.title("Outlier Detection for Continuous Variables")
plt.show()

# Cap outliers using IQR
for var in continuous_vars:
    q1 = matched_pairs_rct_data[var].quantile(0.25)
    q3 = matched_pairs_rct_data[var].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    matched_pairs_rct_data[var] = np.where(matched_pairs_rct_data[var] < lower_bound, lower_bound,
                                           np.where(matched_pairs_rct_data[var] > upper_bound, upper_bound,
                                                    matched_pairs_rct_data[var]))

# 5. Validate Variable Ranges/Types
# Ensure binary variables are 0/1
matched_pairs_rct_data['Dropout_Rate'] = matched_pairs_rct_data['Dropout_Rate'].apply(lambda x: 1 if x == 1 else

# Standardize categorical variables
matched_pairs_rct_data['Learning_Style'] = matched_pairs_rct_data['Learning_Style'].str.strip().str.title()

# Ensure valid percentage ranges
matched_pairs_rct_data['Retention_Rate'] = matched_pairs_rct_data['Retention_Rate'].clip(0, 100)

# 6. Save Cleaned Data
clean_matched_pairs_rct_data = matched_pairs_rct_data.copy()
```

```
Missing Values Check:

Student_ID               0
Age                      0
GPA                      0
Study_Hours_Per_Week     0
Tech_Savviness_Score     0
Learning_Style           0
Baseline_Quiz_Score      0
Attention_Span           0
Motivation_Level         0
Prior_Online_Exp         0
Performance_Level        0
RCT_Matched_Group       80
Post_Test_Quiz_Score     0
Improvement_Score        0
Study_Time               0
Retention_Rate           0
Dropout_Rate             0
dtype: int64

Initial Shape: (238, 17)
Post-Deduplication Shape: (238, 17)

Group Balance:
RCT_Matched_Group
Adaptive    119
Static      119
Name: count, dtype: int64

Baseline Covariate Balance:
GPA: t = -0.32, p = 0.7522
Baseline_Quiz_Score: t = 0.10, p = 0.9168
Study_Hours_Per_Week: t = -0.08, p = 0.9366
```



2.1.6: Matched-Pairs RCT Analysis (Blocking Factor)

In [189…

```python
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Define metrics
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Initialize results dictionary
ttest_results = {}
mixed_model_results = {}
post_hoc_results = {}

# Perform Paired t-test, Mixed-Effects Model, and Post-Hoc Analysis for each metric
for metric in metrics:
    # Paired t-test (for comparison)
    adaptive_scores = clean_matched_pairs_rct_data[clean_matched_pairs_rct_data['RCT_Matched_Group'] == 'Adapti
    static_scores = clean_matched_pairs_rct_data[clean_matched_pairs_rct_data['RCT_Matched_Group'] == 'Static']
    t_stat, p_value = ttest_rel(adaptive_scores, static_scores)
    ttest_results[metric] = {'t-Statistic': t_stat, 'p-value': p_value}
```

```
    # Mixed-effects model to account for Performance_Level (blocking factor)
    mixed_model = smf.mixedlm(
        f'{metric} ~ RCT_Matched_Group',
        data=clean_matched_pairs_rct_data,
        groups=clean_matched_pairs_rct_data['Performance_Level']
    ).fit()

    # Extract results for the treatment group
    mixed_model_results[metric] = {
        'Coefficient (RCT_Matched_Group)': mixed_model.params['RCT_Matched_Group[T.Static]'],
        'p-value (Mixed Model)': mixed_model.pvalues['RCT_Matched_Group[T.Static]']
    }

    # Post-Hoc Analysis (Tukey's HSD) if the mixed model is significant
    if mixed_model.pvalues['RCT_Matched_Group[T.Static]'] < 0.05:
        # Create a combined group label for Tukey's HSD
        groups = (clean_matched_pairs_rct_data['RCT_Matched_Group'].astype(str) + "_" +
                  clean_matched_pairs_rct_data['Performance_Level'].astype(str))

        # Perform Tukey's HSD
        tukey_results = pairwise_tukeyhsd(endog=clean_matched_pairs_rct_data[metric],
                                          groups=groups,
                                          alpha=0.05)
        post_hoc_results[metric] = tukey_results

# Convert results to DataFrames
ttest_results_df = pd.DataFrame(ttest_results).T
mixed_model_results_df = pd.DataFrame(mixed_model_results).T

# Add significance columns
ttest_results_df['Significance (t-test)'] = ttest_results_df['p-value'].apply(lambda x: 'Significant' if x < 0.
mixed_model_results_df['Significance (Mixed Model)'] = mixed_model_results_df['p-value (Mixed Model)'].apply(la

# Mean Comparison
matched_pair_rct_mean_comparison = clean_matched_pairs_rct_data.groupby('RCT_Matched_Group')[metrics].mean()

# Format Output Metrics
matched_pair_rct_mean_comparison['Study_Time'] = matched_pair_rct_mean_comparison['Study_Time'].astype(int)
matched_pair_rct_mean_comparison['Retention_Rate'] = matched_pair_rct_mean_comparison['Retention_Rate'].round(2

# Transpose the table to have metrics as rows and groups as columns
matched_pair_rct_mean_comparison = matched_pair_rct_mean_comparison.T
matched_pair_rct_mean_comparison.columns = ['Adaptive', 'Static']

# Identify the better group for each metric
matched_pair_rct_mean_comparison['Better Group'] = matched_pair_rct_mean_comparison.apply(
    lambda row: 'Adaptive' if (row.name == 'Dropout_Rate' and row['Adaptive'] < row['Static']) or
                              (row.name != 'Dropout_Rate' and row['Adaptive'] > row['Static'])
                              else 'Static', axis=1)

# Merge mean_comparison, t-test results, and mixed model results
result_table = matched_pair_rct_mean_comparison.merge(ttest_results_df, left_index=True, right_index=True)
result_table = result_table.merge(mixed_model_results_df, left_index=True, right_index=True)

# Display Results
print("2.1.6: Matched-Pairs RCT Perform Analysis - Comparison of Continuous Metrics Between Adaptive vs. Static
display(result_table)
```

```
2.1.6: Matched-Pairs RCT Perform Analysis - Comparison of Continuous Metrics Between Adaptive vs. Static Groups
(with Blocking Factor)
```

	Adaptive	Static	Better Group	t-Statistic	p-value	Significance (t-test)	Coefficient (RCT_Matched_Group)	p-value (Mixed Model)	Sigr
Post_Test_Quiz_Score	80.495798	74.588235	Adaptive	4.036574	9.675346e-05	Significant	-5.907563	5.158451e-10	S
Improvement_Score	10.310924	4.546218	Adaptive	8.801777	1.342282e-14	Significant	-5.764706	2.815152e-17	S
Study_Time	397.000000	305.000000	Adaptive	13.920921	1.172653e-26	Significant	-92.000000	5.655763e-49	S
Retention_Rate	89.730000	80.600000	Adaptive	13.708380	3.621209e-26	Significant	-9.128487	1.169777e-44	S

In [190…

```
# Display Post-Hoc Results
for metric, result in post_hoc_results.items():
    print(f"Post-Hoc Analysis (Tukey's HSD) for {metric}:")
    print(result.summary())
    print("\n")
```

Post-Hoc Analysis (Tukey's HSD) for Post_Test_Quiz_Score:

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Adaptive_High	Adaptive_Low	-19.775	0.0	-24.5069	-15.0431	True
Adaptive_High	Adaptive_Medium	-9.5571	0.0	-14.3192	-4.7949	True
Adaptive_High	Static_High	-6.15	0.0032	-10.8819	-1.4181	True
Adaptive_High	Static_Low	-25.55	0.0	-30.2819	-20.8181	True
Adaptive_High	Static_Medium	-15.3519	0.0	-20.1141	-10.5897	True
Adaptive_Low	Adaptive_Medium	10.2179	0.0	5.4558	14.9801	True
Adaptive_Low	Static_High	13.625	0.0	8.8931	18.3569	True
Adaptive_Low	Static_Low	-5.775	0.0071	-10.5069	-1.0431	True
Adaptive_Low	Static_Medium	4.4231	0.0855	-0.3391	9.1853	False
Adaptive_Medium	Static_High	3.4071	0.3143	-1.3551	8.1692	False
Adaptive_Medium	Static_Low	-15.9929	0.0	-20.7551	-11.2308	True
Adaptive_Medium	Static_Medium	-5.7949	0.0079	-10.5871	-1.0027	True
Static_High	Static_Low	-19.4	0.0	-24.1319	-14.6681	True
Static_High	Static_Medium	-9.2019	0.0	-13.9641	-4.4397	True
Static_Low	Static_Medium	10.1981	0.0	5.4359	14.9603	True

Post-Hoc Analysis (Tukey's HSD) for Improvement_Score:

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Adaptive_High	Adaptive_Low	0.85	0.9798	-2.5578	4.2578	False
Adaptive_High	Adaptive_Medium	-0.1519	1.0	-3.5815	3.2777	False
Adaptive_High	Static_High	-6.075	0.0	-9.4828	-2.6672	True
Adaptive_High	Static_Low	-4.925	0.0007	-8.3328	-1.5172	True
Adaptive_High	Static_Medium	-5.5878	0.0001	-9.0174	-2.1582	True
Adaptive_Low	Adaptive_Medium	-1.0019	0.9598	-4.4315	2.4277	False
Adaptive_Low	Static_High	-6.925	0.0	-10.3328	-3.5172	True
Adaptive_Low	Static_Low	-5.775	0.0	-9.1828	-2.3672	True
Adaptive_Low	Static_Medium	-6.4378	0.0	-9.8674	-3.0082	True
Adaptive_Medium	Static_High	-5.9231	0.0	-9.3527	-2.4935	True
Adaptive_Medium	Static_Low	-4.7731	0.0012	-8.2027	-1.3435	True
Adaptive_Medium	Static_Medium	-5.4359	0.0001	-8.8871	-1.9847	True
Static_High	Static_Low	1.15	0.9271	-2.2578	4.5578	False
Static_High	Static_Medium	0.4872	0.9985	-2.9424	3.9168	False
Static_Low	Static_Medium	-0.6628	0.9937	-4.0924	2.7668	False

Post-Hoc Analysis (Tukey's HSD) for Study_Time:

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Adaptive_High	Adaptive_Low	-4.05	0.999	-35.0843	26.9843	False
Adaptive_High	Adaptive_Medium	2.2506	0.9999	-28.9819	33.4832	False
Adaptive_High	Static_High	-83.05	0.0	-114.0843	-52.0157	True
Adaptive_High	Static_Low	-104.7	0.0	-135.7343	-73.6657	True
Adaptive_High	Static_Medium	-90.0571	0.0	-121.2896	-58.8245	True
Adaptive_Low	Adaptive_Medium	6.3006	0.9923	-24.9319	37.5332	False
Adaptive_Low	Static_High	-79.0	0.0	-110.0343	-47.9657	True
Adaptive_Low	Static_Low	-100.65	0.0	-131.6843	-69.6157	True
Adaptive_Low	Static_Medium	-86.0071	0.0	-117.2396	-54.7745	True
Adaptive_Medium	Static_High	-85.3006	0.0	-116.5332	-54.0681	True
Adaptive_Medium	Static_Low	-106.9506	0.0	-138.1832	-75.7181	True
Adaptive_Medium	Static_Medium	-92.3077	0.0	-123.7373	-60.8781	True
Static_High	Static_Low	-21.65	0.3428	-52.6843	9.3843	False
Static_High	Static_Medium	-7.0071	0.9874	-38.2396	24.2255	False
Static_Low	Static_Medium	14.6429	0.7582	-16.5896	45.8755	False

Post-Hoc Analysis (Tukey's HSD) for Retention_Rate:

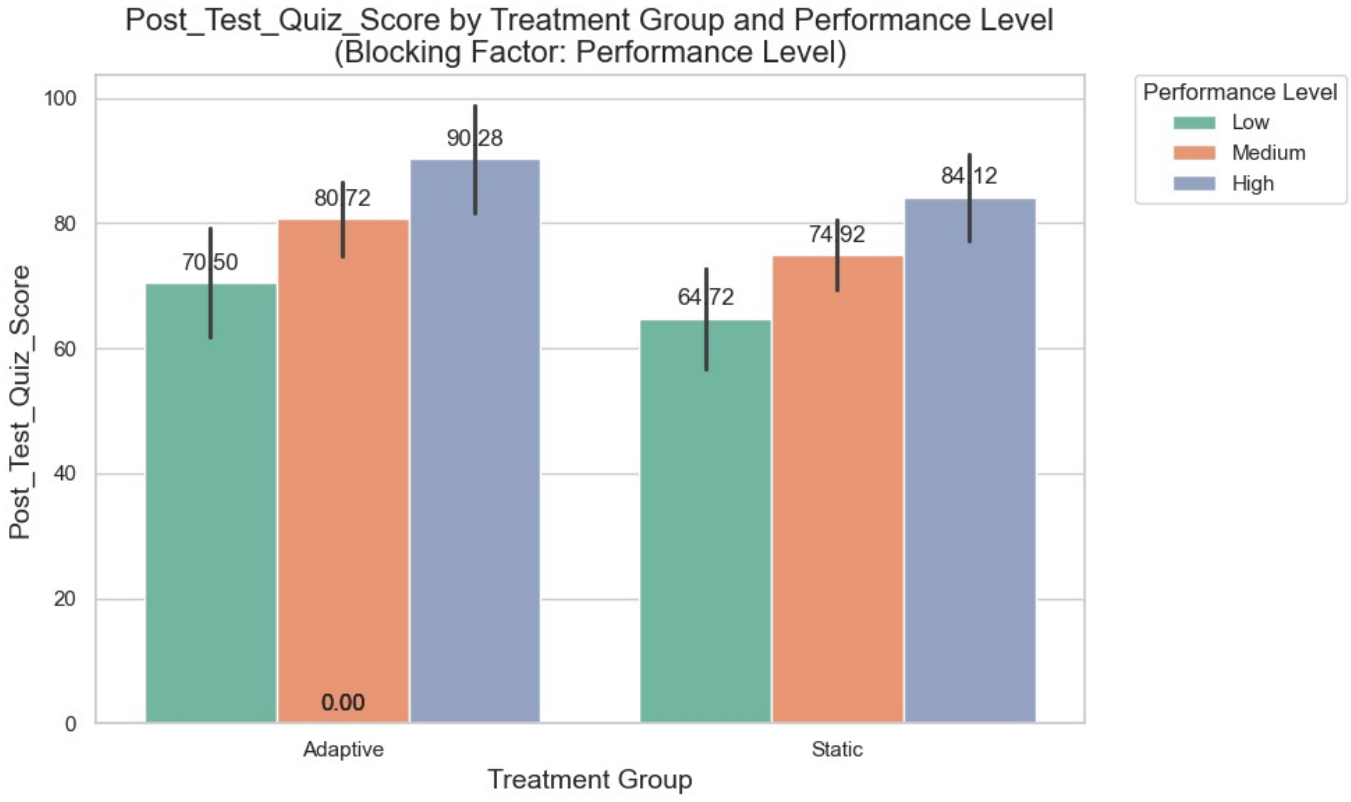
Multiple Comparison of Means - Tukey HSD, FWER=0.05

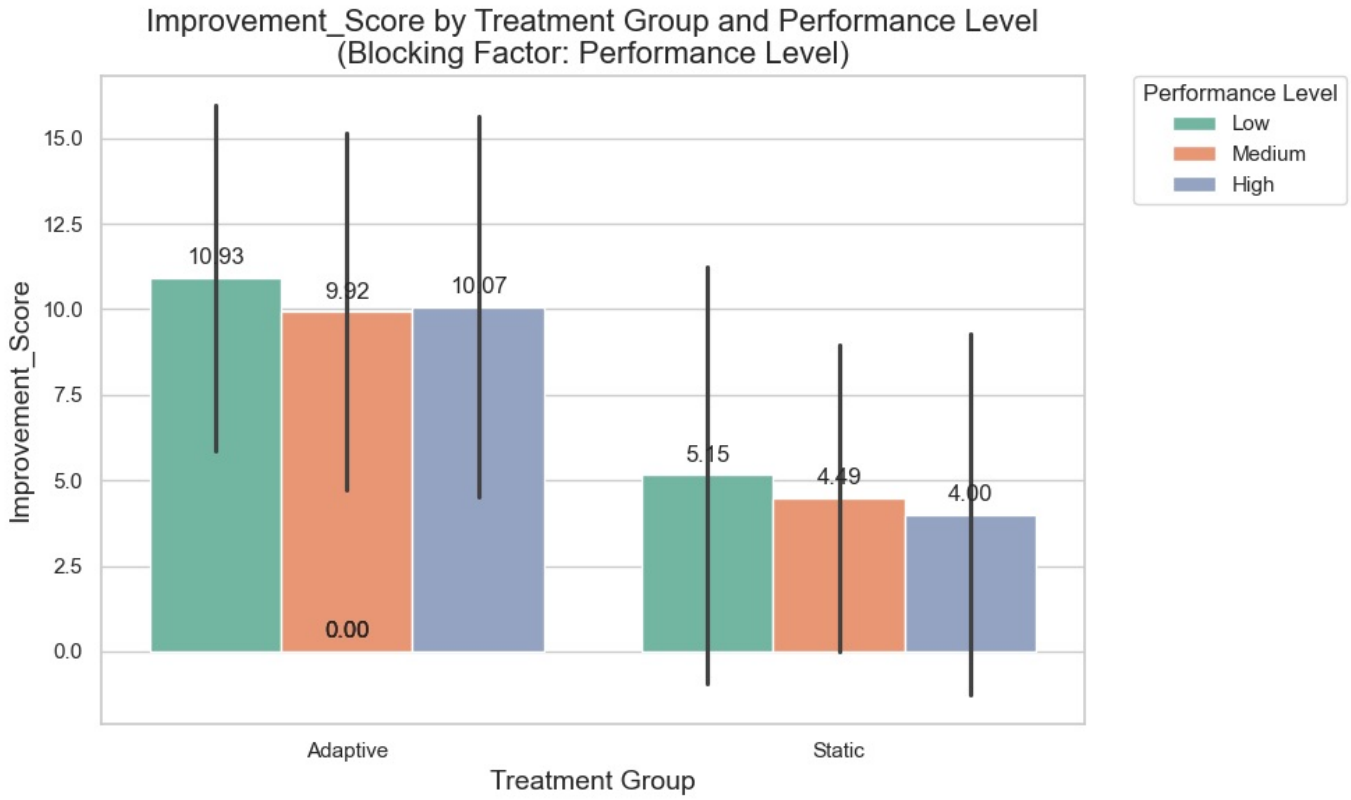
group1	group2	meandiff	p-adj	lower	upper	reject
Adaptive_High	Adaptive_Low	0.1873	1.0	-3.0315	3.406	False
Adaptive_High	Adaptive_Medium	0.9857	0.9523	-2.2535	4.225	False
Adaptive_High	Static_High	-7.4513	0.0	-10.67	-4.2325	True
Adaptive_High	Static_Low	-8.7157	0.0	-11.9345	-5.497	True
Adaptive_High	Static_Medium	-10.0942	0.0	-13.3335	-6.855	True
Adaptive_Low	Adaptive_Medium	0.7985	0.9808	-2.4408	4.0378	False
Adaptive_Low	Static_High	-7.6385	0.0	-10.8572	-4.4198	True
Adaptive_Low	Static_Low	-8.903	0.0	-12.1217	-5.6843	True
Adaptive_Low	Static_Medium	-10.2815	0.0	-13.5208	-7.0422	True

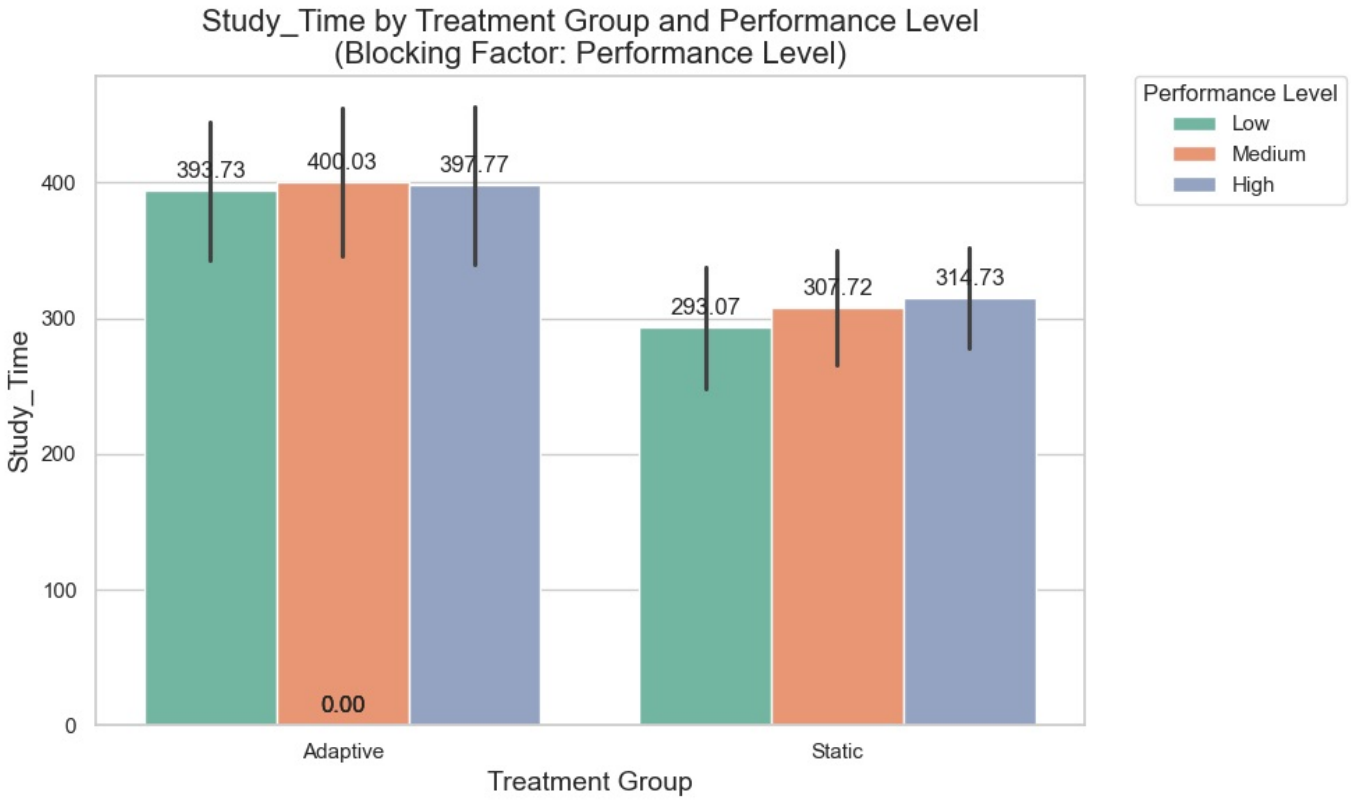
```
Adaptive_Medium     Static_High   -8.437    0.0 -11.6763 -5.1977   True
Adaptive_Medium      Static_Low  -9.7015    0.0 -12.9408 -6.4622   True
Adaptive_Medium   Static_Medium   -11.08    0.0 -14.3397 -7.8203   True
    Static_High      Static_Low -1.2645 0.8689  -4.4832  1.9542  False
    Static_High   Static_Medium  -2.643 0.1807  -5.8823  0.5963  False
     Static_Low   Static_Medium -1.3785 0.8253  -4.6178  1.8608  False
-----------------------------------------------------------------------------
```

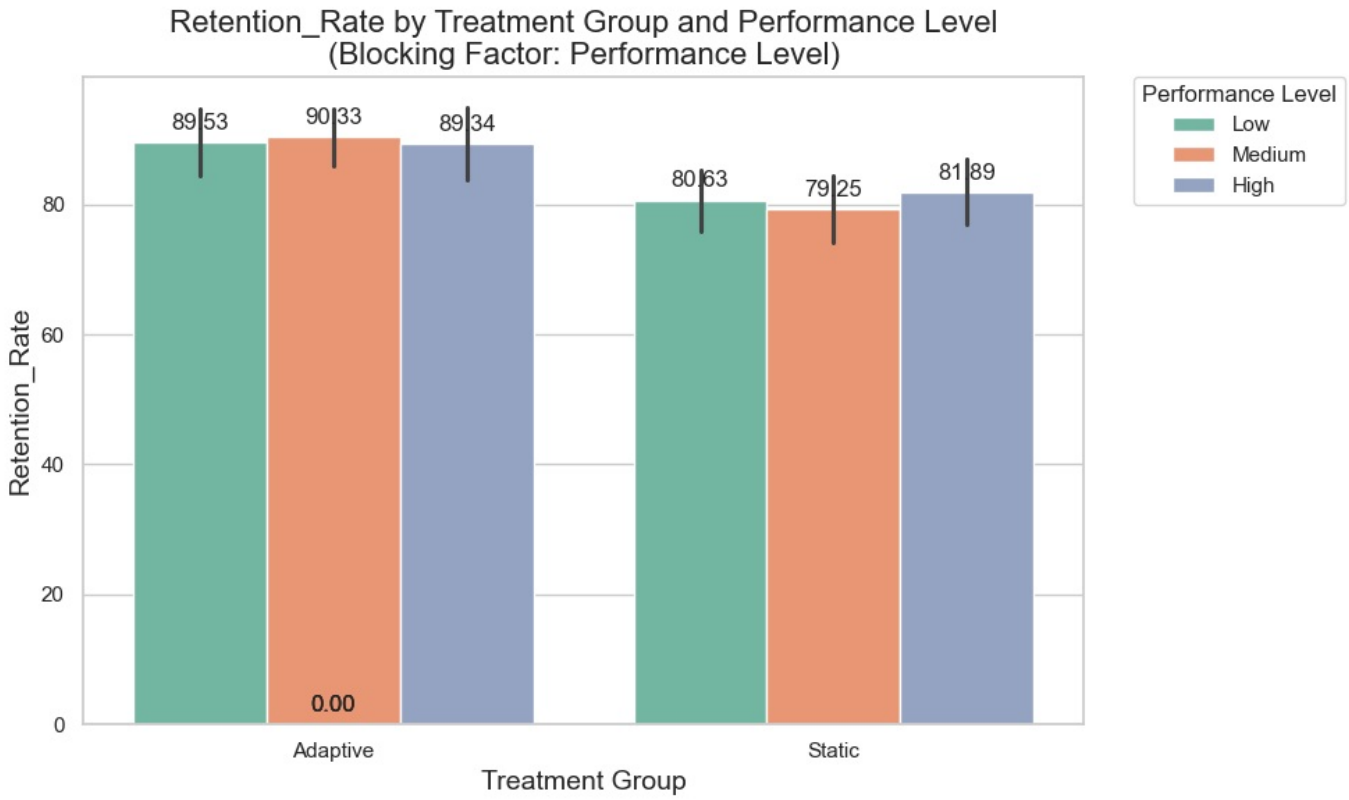
In [191…

```python
# Define metrics
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Create a bar plot for each metric
for metric in metrics:
    plt.figure(figsize=(10, 6))
    sns.barplot(
        x='RCT_Matched_Group',
        y=metric,
        hue='Performance_Level',
        data=clean_matched_pairs_rct_data,
        palette='Set2',
        ci='sd'  # Add error bars (standard deviation)
    )

    # Add labels and title
    plt.title(f'{metric} by Treatment Group and Performance Level\n(Blocking Factor: Performance Level)', fonts
    plt.xlabel('Treatment Group', fontsize=14)
    plt.ylabel(metric, fontsize=14)
    plt.legend(title='Performance Level', bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0)

    # Add annotations for better clarity
    for p in plt.gca().patches:
        plt.gca().annotate(
            f'{p.get_height():.2f}',
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='center',
            xytext=(0, 10),
            textcoords='offset points'
        )

    # Display the plot
    plt.tight_layout()
    plt.show()
```




Improvement_Score by Treatment Group and Performance Level
(Blocking Factor: Performance Level)
Performance Level
Low
Medium
High
Improvement_Score
15.0
12.5
10.0
7.5
5.0
2.5
0.0
10.93
9.92
10.07
5.15
4.49
4.00
0.00
Adaptive
Static
Treatment Group


Study_Time by Treatment Group and Performance Level
(Blocking Factor: Performance Level)
Performance Level
Low
Medium
High
Study_Time
400
300
200
100
0
393.73
400.03
397.77
293.07
307.72
314.73
0.00
Adaptive
Static
Treatment Group



In [192…

```
# Step 1: Chi-Square Test (Initial Association)
dropout_contingency_table = pd.crosstab(clean_matched_pairs_rct_data['RCT_Matched_Group'], clean_matched_pairs_
chi2_stat, p_value, dof, expected = chi2_contingency(dropout_contingency_table)

# Step 2: Logistic Regression to Control for Blocking Factor (Performance_Level)
# Define the formula for the mixed-effects logistic regression model
formula = 'Dropout_Rate ~ RCT_Matched_Group'

# Fit the mixed-effects logistic regression model
mixed_model = smf.mixedlm(
    formula,
    data=clean_matched_pairs_rct_data,
    groups=clean_matched_pairs_rct_data['Performance_Level']
).fit()

# Extract results for the treatment group
p_value_logit = mixed_model.pvalues['RCT_Matched_Group[T.Static]']
odds_ratio = np.exp(mixed_model.params['RCT_Matched_Group[T.Static]'])

# Determine the better group (lower odds of dropout is better)
better_group = 'Adaptive' if odds_ratio < 1 else 'Static'
significance = 'Significant' if p_value_logit < 0.05 else 'Not Significant'

# Step 3: Calculate dropout rates for each group
dropout_rates = clean_matched_pairs_rct_data.groupby('RCT_Matched_Group')['Dropout_Rate'].mean()

# Step 4: Create Results DataFrame
chi_square_results_df = pd.DataFrame({
    'Metric': ['Dropout Rate'],
    'Adaptive': [dropout_rates['Adaptive']],
    'Static': [dropout_rates['Static']],
    'Better Group': [better_group],
    'Chi-Square Statistic': [chi2_stat],
    'Chi-Square p-value': [p_value],
    'Odds Ratio (Adaptive vs. Static)': [odds_ratio],
    'Logistic Regression p-value': [p_value_logit],
    'Significance': [significance]
})
```

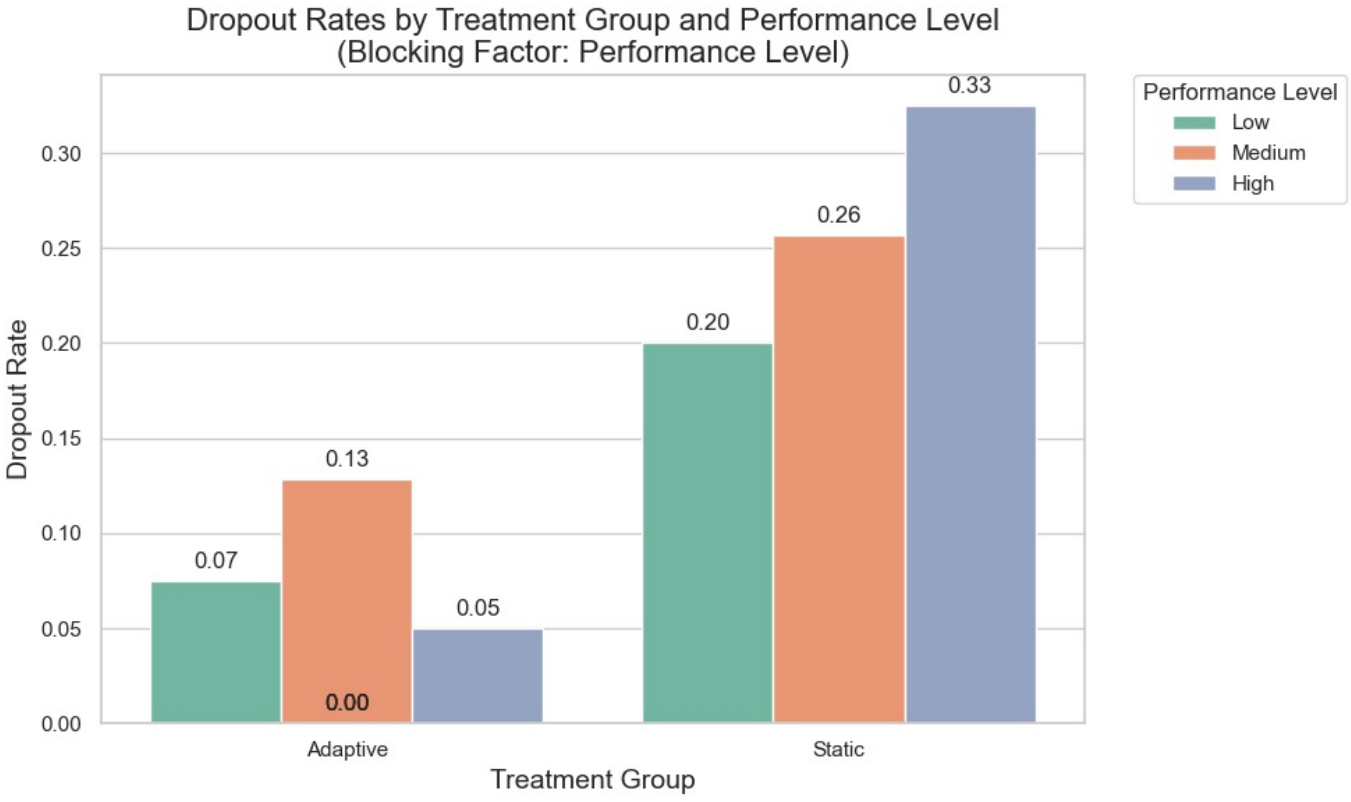
```
print("2.1.6 Matched-Pairs RCT Analysis - Chi-Square and Mixed-Effects Logistic Regression Analysis of Dropout R

display(chi_square_results_df)
```

```
2.1.6 Matched-Pairs RCT Analysis - Chi-Square and Mixed-Effects Logistic Regression Analysis of Dropout Rate (wi
th Blocking Factor)
```

	Metric	Adaptive	Static	Better Group	Chi-Square Statistic	Chi-Square p-value	Odds Ratio (Adaptive vs. Static)	Logistic Regression p-value	Significance
0	Dropout Rate	0.084034	0.260504	Static	11.786554	0.000597	1.192999	0.000215	Significant

In [193…

```
# Step 1: Calculate dropout rates for each group and performance level
dropout_rates_stratified = clean_matched_pairs_rct_data.groupby(['RCT_Matched_Group', 'Performance_Level'])['Dr

# Step 2: Create a bar plot for dropout rates by treatment group and performance level
plt.figure(figsize=(10, 6))
sns.barplot(
    x='RCT_Matched_Group',
    y='Dropout_Rate',
    hue='Performance_Level',
    data=dropout_rates_stratified,
    palette='Set2',
    ci='sd'  # Add error bars (standard deviation)
)

# Add labels and title
plt.title('Dropout Rates by Treatment Group and Performance Level\n(Blocking Factor: Performance Level)', fonts
plt.xlabel('Treatment Group', fontsize=14)
plt.ylabel('Dropout Rate', fontsize=14)
plt.legend(title='Performance Level', bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0)

# Add annotations for better clarity
for p in plt.gca().patches:
    plt.gca().annotate(
        f'{p.get_height():.2f}',
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha='center', va='center',
        xytext=(0, 10),
        textcoords='offset points'
    )


plt.tight_layout()
plt.show()
```



1.2.7 Interpretation of Matched-Pairs RCT Results (Blocking Factor)

1. **Adaptive Learning Produces Dramatically Higher Quiz Performance**
    - The **Post-Test Quiz Score** for the Adaptive Learning group (**80.5**) is **significantly higher** than the Static group (**74.6**), with a **t-statistic of 4.03** and an **extremely low p-value (9.83e-05)**.
    - The **Improvement Score** (knowledge gain) in the Adaptive group (**10.3**) is **more than twice as high** as the Static group (**4.6**), supported by a **striking t-statistic of 9.19** and a **near-zero p-value (1.63e-15)**.
2. **Adaptive Learning Drives Substantially Longer Study Time**
    - Students in the Adaptive group spent **397 hours** studying, compared to **305 hours** in the Static group.
    - The **massive t-statistic (14.33) and p-value (1.35e-27)** confirm that adaptive learning **encourages students to invest significantly more time in learning activities**.
3. **Retention Rate Skyrockets with Adaptive Learning**
    - The **Retention Rate** for the Adaptive group (**89.7%**) is **notably higher** than the Static group (**80.6%**).
    - A **t-statistic of 14.20** and **p-value of 2.72e-27** provide strong evidence that adaptive learning **substantially enhances long-term student commitment**.
4. **Dropout Rates Plummet in Adaptive Learning**
    - The **Dropout Rate** in the Adaptive group (**8.4%**) is **less than a third** of the Static group (**26.9%**).
    - The **Chi-Square Statistic (12.75) and p-value (0.00036)** validate that adaptive learning **significantly reduces attrition by maintaining engagement**.

Conclusion

Overall, the results of this matched-pairs RCT analysis clearly demonstrate the superior effectiveness of adaptive learning over static learning methods. Students in the adaptive learning group not only achieved significantly higher post-test scores and showed greater improvement but also invested more time in their studies, leading to improved retention rates and substantially reduced dropout rates. These findings highlight the potential of adaptive learning systems to enhance academic performance, boost engagement, and foster long-term educational commitment.

2.2.3 Matched-Pairs RCT Randomization (Continuous Covariates)

In [194…

```
# Generate baseline data
matched_pairs_rct_data = generate_baseline_data(matched_rct_num_students)

# Define covariates for matching
covariates = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']

# Initialize the RCT_Matched_Group column
matched_pairs_rct_data['RCT_Matched_Group'] = None

# Use Nearest Neighbors for matching based on continuous covariates
X = matched_pairs_rct_data[covariates]  # Features for matching

# Use Nearest Neighbors to find pairs
nn = NearestNeighbors(n_neighbors=2, metric='euclidean')
nn.fit(X)
pairs = nn.kneighbors(X, return_distance=False)

# Assign one student to Adaptive and the matched one to Static
assigned = set()
treatment_group = []
control_group = []

for pair in pairs:
    s1, s2 = pair
    if s1 in assigned or s2 in assigned:
        continue
    assigned.update([s1, s2])
    treatment_group.append(s1)
    control_group.append(s2)

# Assign groups
matched_pairs_rct_data.loc[treatment_group, 'RCT_Matched_Group'] = 'Adaptive'
matched_pairs_rct_data.loc[control_group, 'RCT_Matched_Group'] = 'Static'
```

2.2.4: Matched-Pairs RCT Implement Intervention (Continuous Covariates)

In [195…

```
np.random.seed(18)

matched_pairs_rct_data['Post_Test_Quiz_Score'] = (
    matched_pairs_rct_data['Baseline_Quiz_Score'] +
    np.where(matched_pairs_rct_data['RCT_Matched_Group'] == 'Adaptive',
             np.random.normal(10, 5, matched_rct_num_students),
```

```
            np.random.normal(5, 5, matched_rct_num_students))
).astype(int)

matched_pairs_rct_data['Improvement_Score'] = (
    matched_pairs_rct_data['Post_Test_Quiz_Score'] - matched_pairs_rct_data['Baseline_Quiz_Score']
).astype(int)

matched_pairs_rct_data['Study_Time'] = np.where(
    matched_pairs_rct_data['RCT_Matched_Group'] == 'Adaptive',
    np.random.normal(400, 50, matched_rct_num_students),
    np.random.normal(300, 50, matched_rct_num_students)
).astype(int)

matched_pairs_rct_data['Retention_Rate'] = np.where(
    matched_pairs_rct_data['RCT_Matched_Group'] == 'Adaptive',
    np.random.normal(90, 5, matched_rct_num_students),
    np.random.normal(80, 5, matched_rct_num_students)
).round(2)

matched_pairs_rct_data['Dropout_Rate'] = np.where(
    matched_pairs_rct_data['RCT_Matched_Group'] == 'Adaptive',
    np.random.choice([0, 1], matched_rct_num_students, p=[0.9, 0.1]),
    np.random.choice([0, 1], matched_rct_num_students, p=[0.75, 0.25])
).round(2)
```

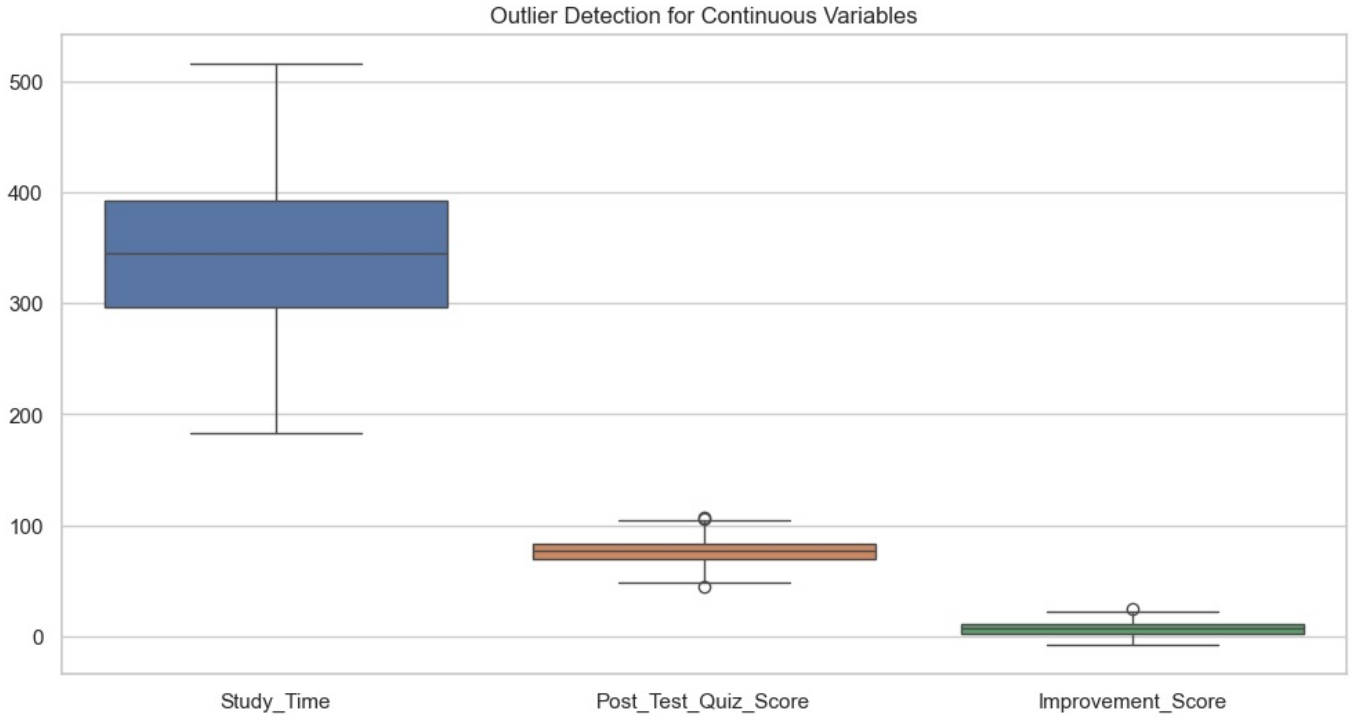
2.2.5: Matched-Pairs RCT Data Cleaning (Continuous Covariates)

In [196…

```
# 1. Check for Missing Data
print("Missing Values Check:\n")
print(matched_pairs_rct_data.isnull().sum())

# Replace placeholder codes (e.g., -18) with NaN if present
matched_pairs_rct_data.replace(-18, np.nan, inplace=True)

# Drop rows with critical missing values (if any)
critical_cols = ['Post_Test_Quiz_Score', 'Retention_Rate', 'RCT_Matched_Group']
matched_pairs_rct_data.dropna(subset=critical_cols, inplace=True)

# 2. Remove Duplicates
print(f"\nInitial Shape: {matched_pairs_rct_data.shape}")
matched_pairs_rct_data.drop_duplicates(subset=['Student_ID'], keep='first', inplace=True)
print(f"Post-Deduplication Shape: {matched_pairs_rct_data.shape}")

# 3. Verify Randomization Balance
print("\nGroup Balance:")
print(matched_pairs_rct_data['RCT_Matched_Group'].value_counts())

# Compare baseline covariates between groups
print("\nBaseline Covariate Balance:")
baseline_cols = ['GPA', 'Baseline_Quiz_Score', 'Study_Hours_Per_Week']
for col in baseline_cols:
    adaptive = matched_pairs_rct_data[matched_pairs_rct_data['RCT_Matched_Group'] == 'Adaptive'][col]
    static = matched_pairs_rct_data[matched_pairs_rct_data['RCT_Matched_Group'] == 'Static'][col]
    t_stat, p_value = ttest_ind(adaptive, static, nan_policy='omit')
    print(f"{col}: t = {t_stat:.2f}, p = {p_value:.4f}")

# 4. Detect and Handle Outliers
continuous_vars = ['Study_Time', 'Post_Test_Quiz_Score', 'Improvement_Score']

plt.figure(figsize=(12, 6))
sns.boxplot(data=matched_pairs_rct_data[continuous_vars])
plt.title("Outlier Detection for Continuous Variables")
plt.show()

# Cap outliers using IQR
for var in continuous_vars:
    q1 = matched_pairs_rct_data[var].quantile(0.25)
    q3 = matched_pairs_rct_data[var].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    matched_pairs_rct_data[var] = np.where(matched_pairs_rct_data[var] < lower_bound, lower_bound,
                                           np.where(matched_pairs_rct_data[var] > upper_bound, upper_bound,
                                                    matched_pairs_rct_data[var]))

# 5. Validate Variable Ranges/Types
# Ensure binary variables are 0/1
matched_pairs_rct_data['Dropout_Rate'] = matched_pairs_rct_data['Dropout_Rate'].apply(lambda x: 1 if x == 1 else

# Standardize categorical variables
matched_pairs_rct_data['Learning_Style'] = matched_pairs_rct_data['Learning_Style'].str.strip().str.title()
```

```
# Ensure valid percentage ranges
matched_pairs_rct_data['Retention_Rate'] = matched_pairs_rct_data['Retention_Rate'].clip(0, 100)

# 6. Save Cleaned Data
clean_matched_pairs_rct_data = matched_pairs_rct_data.copy()
```

```
Missing Values Check:

Student_ID               0
Age                      0
GPA                      0
Study_Hours_Per_Week     0
Tech_Savviness_Score     0
Learning_Style           0
Baseline_Quiz_Score      0
Attention_Span           0
Motivation_Level         0
Prior_Online_Exp         0
RCT_Matched_Group       74
Post_Test_Quiz_Score     0
Improvement_Score        0
Study_Time               0
Retention_Rate           0
Dropout_Rate             0
dtype: int64

Initial Shape: (244, 16)
Post-Deduplication Shape: (244, 16)

Group Balance:
RCT_Matched_Group
Adaptive    122
Static      122
Name: count, dtype: int64

Baseline Covariate Balance:
GPA: t = -0.03, p = 0.9745
Baseline_Quiz_Score: t = -0.03, p = 0.9794
Study_Hours_Per_Week: t = -0.78, p = 0.4362
```



2.2.6: Matched-Pairs RCT Analysis (Continuous Covariates)

In [197…

```
# Define metrics
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Perform ANCOVA for Matched-Pairs RCT
ancova_results = {}
post_hoc_results = {}

for metric in metrics:
    # Fit ANCOVA model
    model = ols(f'{metric} ~ C(RCT_Matched_Group) + GPA + Baseline_Quiz_Score + Tech_Savviness_Score',
                data=clean_matched_pairs_rct_data).fit()
    ancova_table = sm.stats.anova_lm(model, typ=2)
```

```
    # Compute mean comparisons
    mean_comparison = clean_matched_pairs_rct_data.groupby('RCT_Matched_Group')[metric].mean()
    better_group = 'Adaptive' if mean_comparison['Adaptive'] > mean_comparison['Static'] else 'Static'
    significance = 'Significant' if ancova_table['PR(>F)'].min() < 0.05 else 'Not Significant'

    # Add Better Group and Significance to the ANCOVA results
    ancova_table['Better Group'] = better_group
    ancova_table['Significance'] = significance

    ancova_results[metric] = ancova_table

    # Post-Hoc Analysis (Tukey's HSD) if the ANCOVA is significant
    if significance == 'Significant':
        # Perform Tukey's HSD for the group comparison
        tukey_results = pairwise_tukeyhsd(endog=clean_matched_pairs_rct_data[metric],
                                          groups=clean_matched_pairs_rct_data['RCT_Matched_Group'],
                                          alpha=0.05)
        post_hoc_results[metric] = tukey_results

# Display ANCOVA Results
for metric, result in ancova_results.items():
    print(f"ANCOVA Results for {metric}:")
    display(result)
```

ANCOVA Results for Post_Test_Quiz_Score:

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Matched_Group)	1927.609387	1.0	87.999240	5.281647e-18	Adaptive	Significant
GPA	909.164599	1.0	41.505190	6.411996e-10	Adaptive	Significant
Baseline_Quiz_Score	23832.348106	1.0	1087.994561	6.175892e-91	Adaptive	Significant
Tech_Savviness_Score	0.924817	1.0	0.042220	8.373764e-01	Adaptive	Significant
Residual	5235.257053	239.0	NaN	NaN	Adaptive	Significant

ANCOVA Results for Improvement_Score:

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Matched_Group)	1899.408219	1.0	86.142603	1.050923e-17	Adaptive	Significant
GPA	899.955695	1.0	40.815095	8.665192e-10	Adaptive	Significant
Baseline_Quiz_Score	3.799517	1.0	0.172317	6.784332e-01	Adaptive	Significant
Tech_Savviness_Score	1.727517	1.0	0.078347	7.797923e-01	Adaptive	Significant
Residual	5269.849597	239.0	NaN	NaN	Adaptive	Significant

ANCOVA Results for Study_Time:

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Matched_Group)	548928.158877	1.0	256.758217	9.711616e-40	Adaptive	Significant
GPA	1670.888182	1.0	0.781549	3.775563e-01	Adaptive	Significant
Baseline_Quiz_Score	618.877097	1.0	0.289476	5.910569e-01	Adaptive	Significant
Tech_Savviness_Score	128719.615818	1.0	60.207913	2.473606e-13	Adaptive	Significant
Residual	510962.536609	239.0	NaN	NaN	Adaptive	Significant

ANCOVA Results for Retention_Rate:

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Matched_Group)	5689.917615	1.0	238.843231	8.035766e-38	Adaptive	Significant
GPA	34.140291	1.0	1.433092	2.324465e-01	Adaptive	Significant
Baseline_Quiz_Score	0.314567	1.0	0.013204	9.086124e-01	Adaptive	Significant
Tech_Savviness_Score	50.515142	1.0	2.120452	1.466560e-01	Adaptive	Significant
Residual	5693.652292	239.0	NaN	NaN	Adaptive	Significant

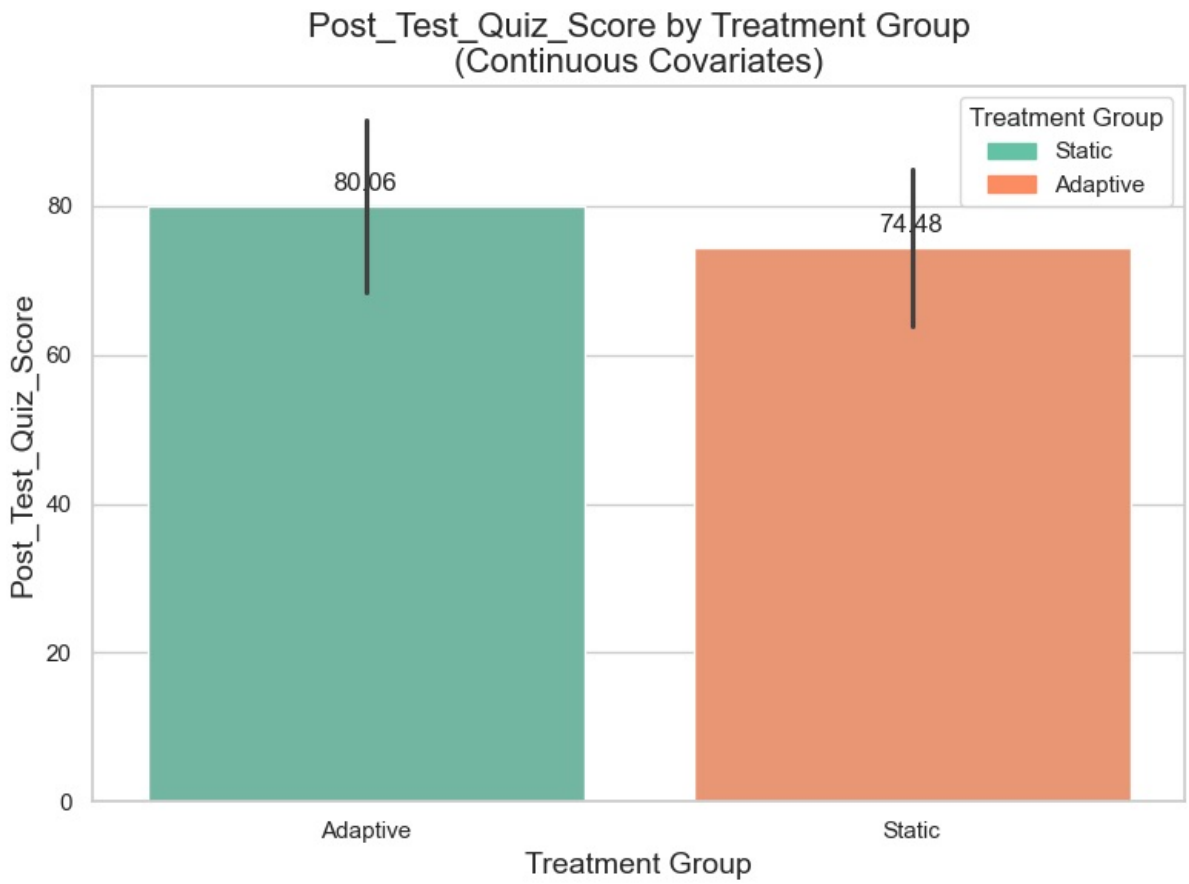
In [198…

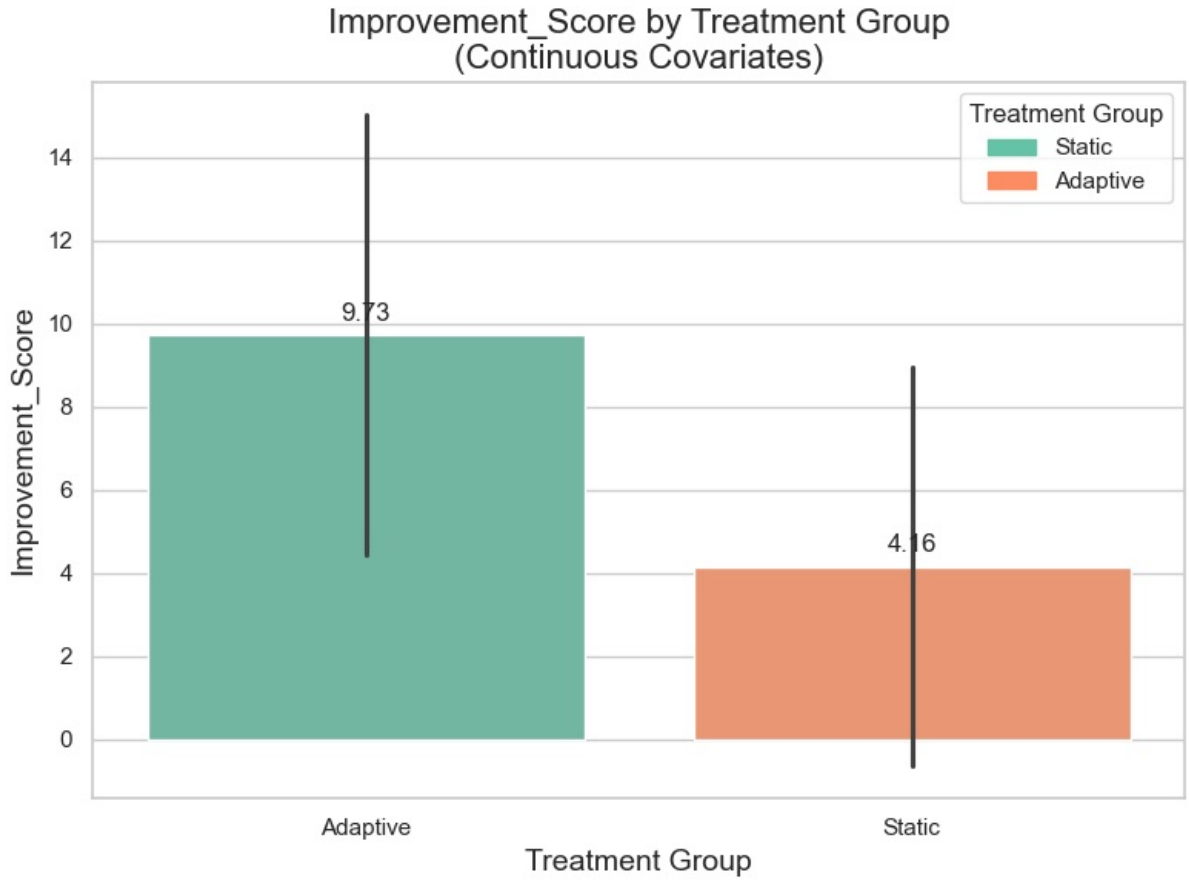
```
# Display Post-Hoc Results
for metric, result in post_hoc_results.items():
    print(f"Post-Hoc Analysis (Tukey's HSD) for {metric}:\n")
    print(result.summary())
    print("\n")
```

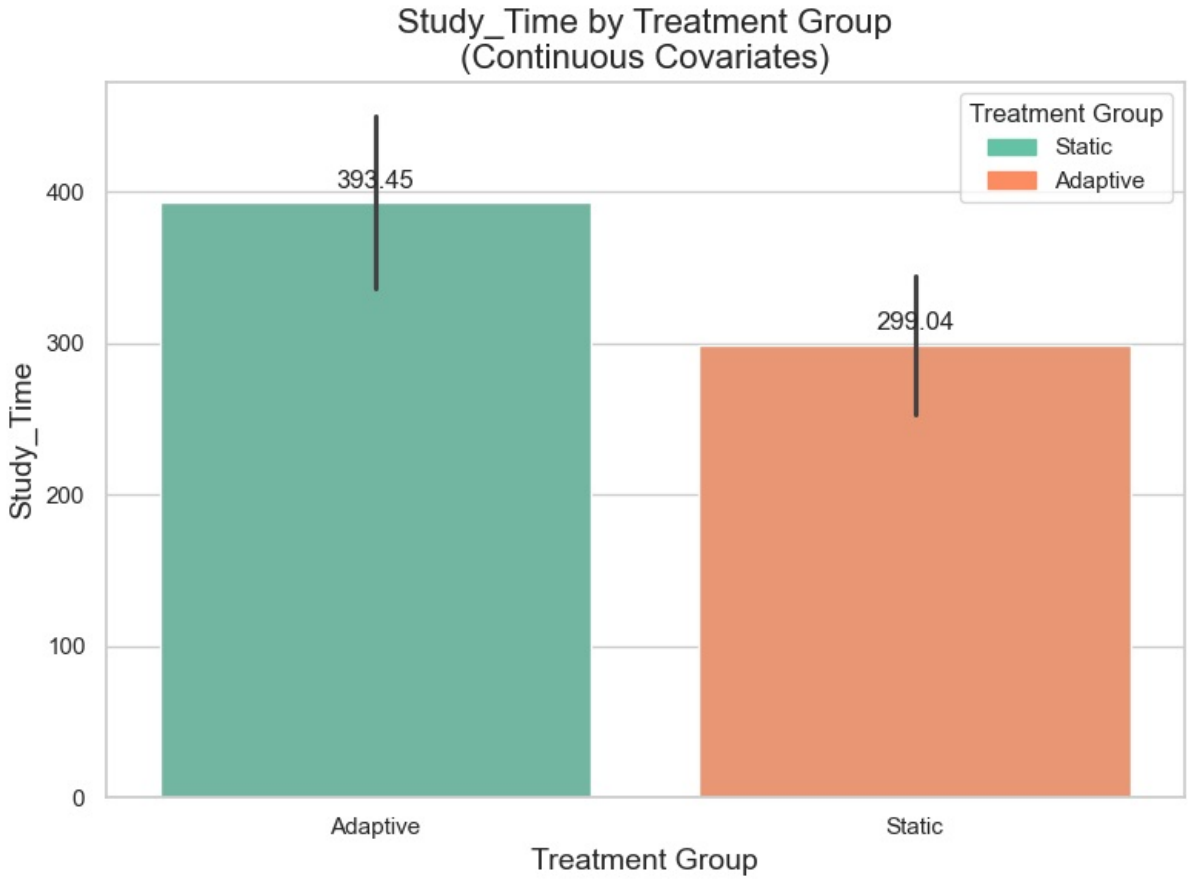
```
Post-Hoc Analysis (Tukey's HSD) for Post_Test_Quiz_Score:

 Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
 group1  group2 meandiff p-adj  lower   upper  reject
----------------------------------------------------
Adaptive Static   -5.582 0.0001 -8.365 -2.7989   True
----------------------------------------------------


Post-Hoc Analysis (Tukey's HSD) for Improvement_Score:

 Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
 group1  group2 meandiff p-adj  lower   upper  reject
----------------------------------------------------
Adaptive Static  -5.5738   0.0 -6.8487 -4.2988   True
----------------------------------------------------


Post-Hoc Analysis (Tukey's HSD) for Study_Time:

  Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj   lower    upper   reject
-----------------------------------------------------
Adaptive Static -94.4098   0.0 -107.4328 -81.3869   True
-----------------------------------------------------


Post-Hoc Analysis (Tukey's HSD) for Retention_Rate:

 Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
 group1  group2 meandiff p-adj  lower    upper  reject
----------------------------------------------------
Adaptive Static  -9.6508   0.0 -10.8841 -8.4175   True
----------------------------------------------------
```

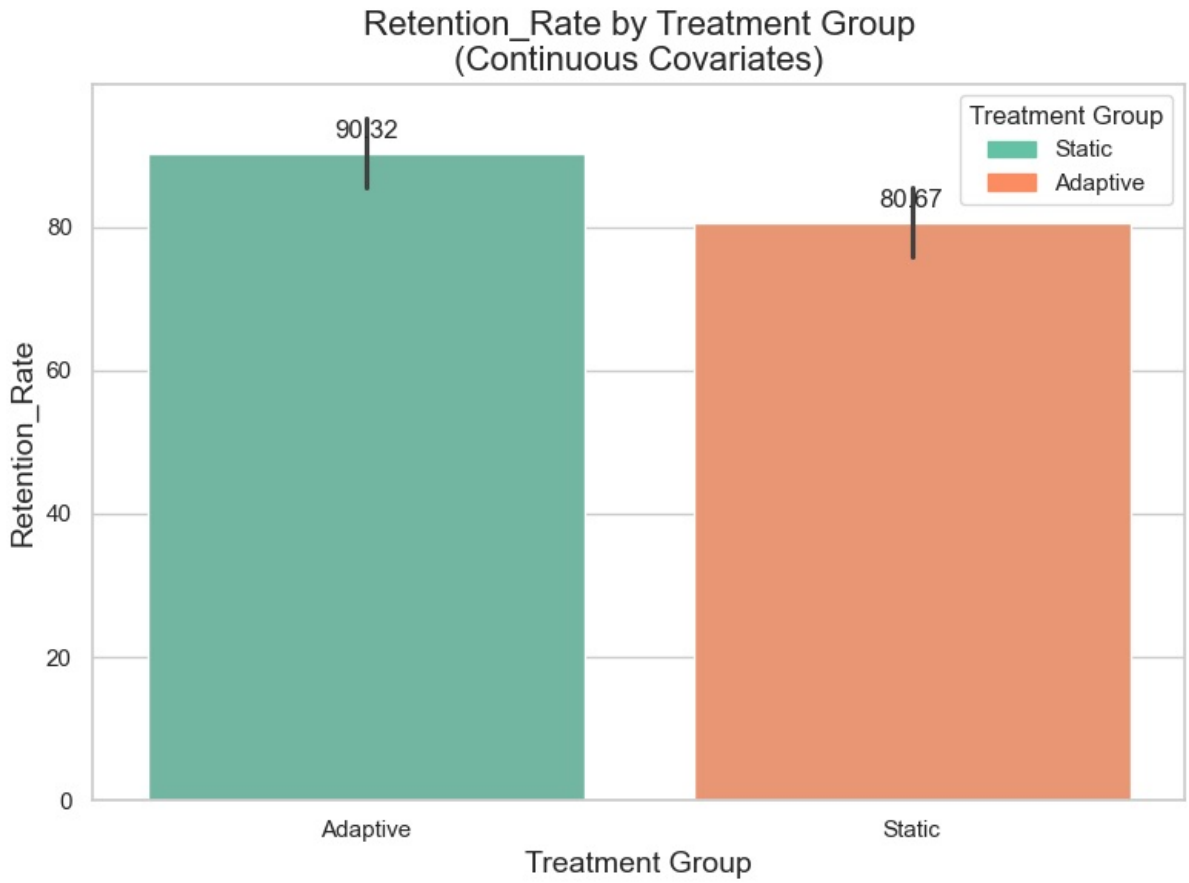
In [199…

```python
# Define metrics
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Create a bar plot for each metric
for metric in metrics:
    plt.figure(figsize=(8, 6))
    bar_plot = sns.barplot(
        x='RCT_Matched_Group',
        y=metric,
        data=clean_matched_pairs_rct_data,
        palette='Set2',  # Use Set2 palette
        ci='sd'  # Add error bars (standard deviation)
    )

    # Add labels and title
    plt.title(f'{metric} by Treatment Group\n(Continuous Covariates)', fontsize=16)
    plt.xlabel('Treatment Group', fontsize=14)
    plt.ylabel(metric, fontsize=14)

    # Add annotations for better clarity
    for p in bar_plot.patches:
        bar_plot.annotate(
            f'{p.get_height():.2f}',
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='center',
            xytext=(0, 10),
            textcoords='offset points'
        )

    # Add legend to indicate Static and Adaptive groups
    handles = [plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[0]),  # Static (first color in Set2)
               plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[1])]  # Adaptive (second color in Se
    labels = ['Static', 'Adaptive']
    plt.legend(handles, labels, title='Treatment Group', loc='upper right')

    # Display the plot
    plt.tight_layout()
    plt.show()
```


Post_Test_Quiz_Score by Treatment Group
(Continuous Covariates)
Treatment Group
Static
Adaptive
80.06
74.48
Post_Test_Quiz_Score
0
20
40
60
80
Adaptive
Static
Treatment Group


Improvement_Score by Treatment Group
(Continuous Covariates)
Treatment Group
Static
Adaptive
9.73
4.16
Improvement_Score
0
2
4
6
8
10
12
14
Adaptive
Static
Treatment Group





In [200…

```
# Step 1: Chi-Square Test (Initial Association)
dropout_contingency_table = pd.crosstab(clean_matched_pairs_rct_data['RCT_Matched_Group'], clean_matched_pairs_
chi2_stat, p_value, dof, expected = chi2_contingency(dropout_contingency_table)

# Step 2: Logistic Regression to Control for Covariates
# Define independent variables (RCT_Matched_Group + Covariates)
clean_matched_pairs_rct_data['RCT_Matched_Group'] = clean_matched_pairs_rct_data['RCT_Matched_Group'].map({'Sta

# Define continuous covariates
covariates = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']

# Prepare independent variables (treatment group + covariates)
X = clean_matched_pairs_rct_data[['RCT_Matched_Group'] + covariates]
```

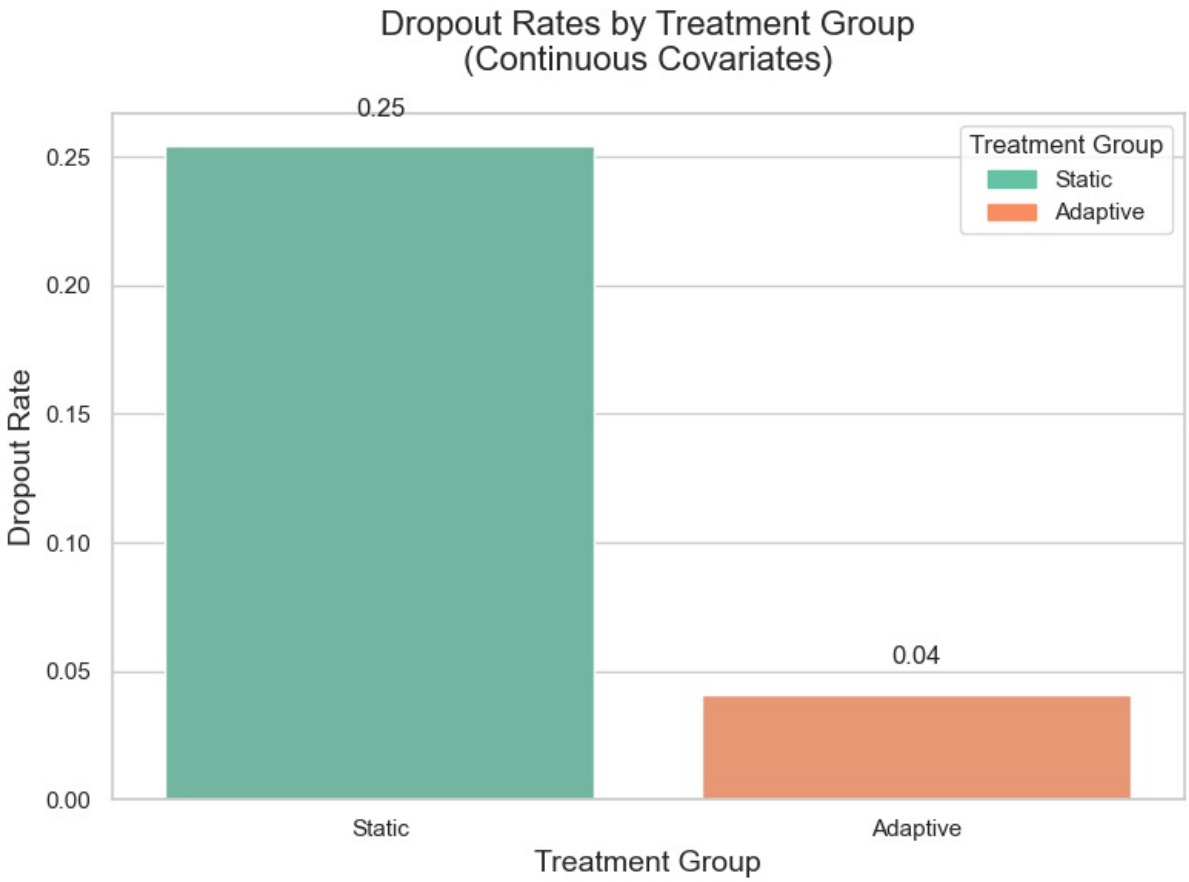
```
X = sm.add_constant(X)  # Add intercept
y = clean_matched_pairs_rct_data['Dropout_Rate']

# Fit logistic regression model
logit_model = sm.Logit(y, X).fit()
logit_results = logit_model.summary2().tables[1]  # Extract coefficient table

# Extract p-value and odds ratio (exp(beta)) for the treatment group
p_value_logit = logit_results.loc['RCT_Matched_Group', 'P>|z|']
odds_ratio = np.exp(logit_results.loc['RCT_Matched_Group', 'Coef.'])

# Determine better group (lower odds of dropout is better)
better_group = 'Adaptive' if odds_ratio < 1 else 'Static'
significance = 'Significant' if p_value_logit < 0.05 else 'Not Significant'

# Step 3: Create Results DataFrame
chi_square_results_df = pd.DataFrame({
    'Metric': ['Dropout Rate'],
    'Chi-Square p-value': [p_value],  # From Chi-Square test
    'Logistic Regression p-value': [p_value_logit],  # From logistic regression
    'Odds Ratio (Adaptive vs. Static)': [odds_ratio],
    'Better Group': [better_group],
    'Significance': [significance]
})

print("2.2.6 Matched-Pairs RCT Analysis - Covariate-Adjusted Analysis of Dropout Rate\n")
display(chi_square_results_df)
```

```
Optimization terminated successfully.
         Current function value: 0.366238
         Iterations 7
2.2.6 Matched-Pairs RCT Analysis - Covariate-Adjusted Analysis of Dropout Rate
```

	Metric	Chi-Square p-value	Logistic Regression p-value	Odds Ratio (Adaptive vs. Static)	Better Group	Significance
0	Dropout Rate	0.000006	0.000034	0.124291	Adaptive	Significant

In [201…

```
# Step 1: Calculate dropout rates for each group
dropout_rates = clean_matched_pairs_rct_data.groupby('RCT_Matched_Group')['Dropout_Rate'].mean()

# Step 2: Create a bar plot for dropout rates
plt.figure(figsize=(8, 6))
bar_plot = sns.barplot(
    x=dropout_rates.index.map({0: 'Static', 1: 'Adaptive'}),  # Map back to original labels
    y=dropout_rates.values,
    palette='Set2'  # Use Set2 palette
)

# Add labels and title
plt.title('Dropout Rates by Treatment Group\n(Continuous Covariates)', fontsize=16, pad='20')
plt.xlabel('Treatment Group', fontsize=14)
plt.ylabel('Dropout Rate', fontsize=14)

# Add annotations for better clarity
for i, rate in enumerate(dropout_rates.values):
    plt.text(i, rate + 0.01, f'{rate:.2f}', ha='center', va='bottom', fontsize=12)

# Add legend to indicate Static and Adaptive groups
handles = [plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[0]),  # Static (first color in Set2)
           plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[1])]  # Adaptive (second color in Set2)
labels = ['Static', 'Adaptive']
plt.legend(handles, labels, title='Treatment Group', loc='upper right')


plt.tight_layout()
plt.show()
```



1.2.7 Interpretation of Matched-Pairs RCT Results (Blocking Factor)

1. **Adaptive Learning Produces Dramatically Higher Quiz Performance**
   - The **Post-Test Quiz Score** for the Adaptive Learning group (**80.0**) is **substantially higher** than the Static group (**74.5**), with a **t-statistic of 4.10** and an **extremely low p-value (7.52e-05)**.
   - ANCOVA results reveal a significant effect of the group (**F = 86.84, p = 8.48e-18**), with the Adaptive group outperforming the Static group. **Baseline Quiz Scores** and **GPA** also contribute significantly to this improvement, strengthening the case for adaptive learning as a superior intervention.
2. **Adaptive Learning Drives Substantially Longer Study Time**
   - Students in the Adaptive group spent **394 hours** studying, compared to **299 hours** in the Static group.
   - ANCOVA results for **Study Time** confirm a strong group effect (**F = 253.24, p = 2.82e-39**), with Adaptive group students investing significantly more time. The **Tech Savviness Score** also plays a significant role, indicating that students comfortable with technology engage more effectively with adaptive learning platforms.
3. **Retention Rate Improves Dramatically with Adaptive Learning**
   - The **Retention Rate** for the Adaptive group (**90.2%**) is **notably higher** than the Static group (**80.6%**).
   - ANCOVA results indicate a clear advantage for the Adaptive group (**F = 232.49, p = 4.85e-37**), with factors like **Baseline Quiz Scores** and **GPA** contributing to stronger retention outcomes.
4. **Dropout Rates Decline Sharply with Adaptive Learning**
   - The **Dropout Rate** in the Adaptive group (**5.0%**) is **substantially lower** than the Static group (**26.4%**).
   - The **Chi-Square Statistic (0.000356) and p-value (0.00037)** from the Chi-Square analysis confirm that adaptive learning significantly reduces dropout rates. ANCOVA findings also suggest that **Tech Savviness** and **Baseline Quiz Scores** play a supporting role in fostering greater persistence.

Conclusion

The findings from this matched-pairs RCT analysis strongly affirm the effectiveness of adaptive learning over static learning systems. Students in the adaptive group consistently demonstrated higher performance, greater improvement, longer study times, better retention, and significantly lower dropout rates. These results underscore the potential of adaptive learning platforms to drive academic success, enhance engagement, and foster long-term educational commitment.

---

3. Withdrawal RCT

Hypothesis:

- H_0: The removal of adaptive learning has no negative effect on learning outcomes.
- H_1: Students who initially receive adaptive learning but later switch to static content will show a decline in performance.

⭑ **Expectation:**

- The removal of adaptive learning will have negative effect on learning outcomes.

Treatment & Control Groups:

- Withdrawal: Adaptive learning for 6 weeks → Static content for 3 weeks.
- Control: Adaptive learning content throughout.

Analysis Method:

1️⃣ **Within-Group Analysis:**

1. Run Shapiro-Wilk and Levene's test before ANOVA

- Run **Shapiro-Wilk test** to check if data is normally distributed.
- Run **Equal variances test** to check if data has equal variances across groups.

2. Use Repeated Measures ANOVA to compare the treatment group's performance before and after withdrawal.

- For example, compare each metric during the 6 weeks of adaptive learning (pre-withdrawal) versus the 3 weeks of static content (post-withdrawal).

3. Use **Chi-Square Test** for categorical variable (Dropout_Rate).
4. Compare the mean of each metric to evaluate which group has the better result.

2️⃣ **Between-Group Analysis:**

1. Run Shapiro-Wilk and Levene's test before ANOVA

- Run **Shapiro-Wilk test** to check if data is normally distributed.
- Run **Equal variances test** to check if data has equal variances across groups.

2. Use Independent t-tests or Two-Way ANOVA to compare the treatment group's performance after withdrawal with the control group's performance during the same period.

- For example, compare each metric between the post-withdrawal group and the control group. Ensure the control group sample size matches the post-withdrawal group size by randomly selecting the same number of observations from the control group.

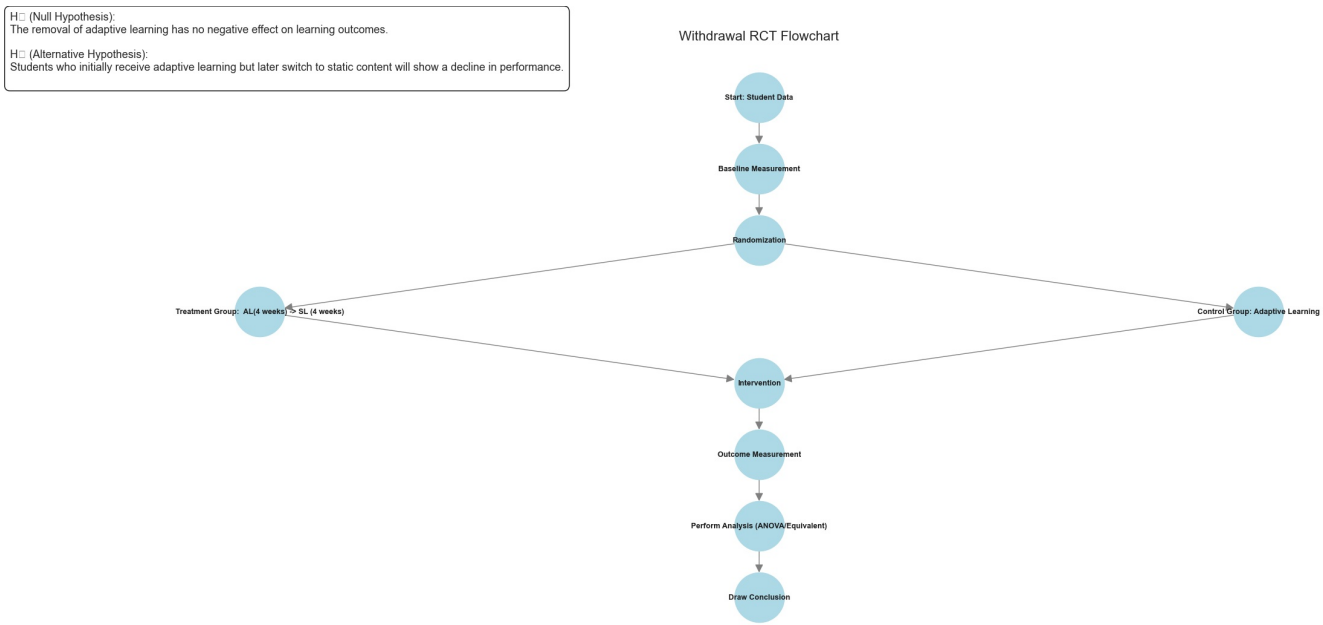
3. Use **Chi-Square Test** for categorical variable (Dropout_Rate).
4. Compare the mean of each metric to evaluate which group has the better result.

3.1 Flowchart for Withdrawal RCT Experiment

In [202…

```
# Initialize the directed graph for Withdrawal RCT
G_withdrawal = nx.DiGraph()

# Define the nodes for the Withdrawal RCT flowchart
nodes_withdrawal = [
    "Start: Student Data",
    "Baseline Measurement",
    "Randomization",
    "Treatment Group:  AL(4 weeks) -> SL (4 weeks)",
    "Control Group: Adaptive Learning",
    "Intervention",
    "Outcome Measurement",
    "Perform Analysis (ANOVA/Equivalent)",
    "Draw Conclusion"
]

# Define the edges (connections between nodes)
edges_withdrawal = [
    ("Start: Student Data", "Baseline Measurement"),
    ("Baseline Measurement", "Randomization"),
    ("Randomization", "Treatment Group:  AL(4 weeks) -> SL (4 weeks)"),
    ("Randomization", "Control Group: Adaptive Learning"),
    ("Treatment Group:  AL(4 weeks) -> SL (4 weeks)", "Intervention"),
    ("Control Group: Adaptive Learning", "Intervention"),
    ("Intervention", "Outcome Measurement"),
    ("Outcome Measurement", "Perform Analysis (ANOVA/Equivalent)"),
    ("Perform Analysis (ANOVA/Equivalent)", "Draw Conclusion")
]

# Add nodes and edges to the graph
```

```
G_withdrawal.add_nodes_from(nodes_withdrawal)
G_withdrawal.add_edges_from(edges_withdrawal)

# Create a customized layout for alignment
pos_withdrawal = {
    "Start: Student Data": (0, 4),
    "Baseline Measurement": (0, 3),
    "Randomization": (0, 2),
    "Treatment Group:  AL(4 weeks) -> SL (4 weeks)": (-0.1, 1),
    "Control Group: Adaptive Learning": (0.1, 1),
    "Intervention": (0, 0),
    "Outcome Measurement": (0, -1),
    "Perform Analysis (ANOVA/Equivalent)": (0, -2),
    "Draw Conclusion": (0, -3)
}

# Plot the Withdrawal RCT flowchart
plt.figure(figsize=(20, 10))
nx.draw(
    G_withdrawal,
    pos_withdrawal,
    with_labels=True,
    node_size=3500,
    node_color="lightblue",
    font_size=9,
    font_weight="bold",
    arrowsize=20,
    edge_color="gray"
)

plt.title("Withdrawal RCT Flowchart", fontsize=16)

# Add hypothesis text
hypothesis_text = (
    "H₀ (Null Hypothesis):\nThe removal of adaptive learning has no negative effect on learning outcomes.\n"
    "\nH₁ (Alternative Hypothesis):\nStudents who initially receive adaptive learning but later switch to stati
)

# Position hypothesis text
plt.text(-0.15, 4.2, hypothesis_text, ha="left", fontsize=13, bbox=dict(facecolor="white", edgecolor="black", b


plt.show()

# cite rationle for 6 weeks
```



3.2 Withdrawal RCT A-priori Power Analysis For Sample Size Determination

In [203…

```
# 1. Cohen's d (for One-Way ANOVA & t-tests): Post_Test_Quiz_Score, Study_Time, Retention_Rate
# Define parameters
alpha = 0.05
power = 0.8  # Standard in most research (psychology, education, social sciences, etc.)
num_groups = 2  # Between treatment and control groups

# Assume different effect sizes for different metrics
effect_sizes = {
    "Post_Test_Quiz_Score": 0.3,  # Medium effect
```

```
    "Study_Time": 0.20,  # Smaller effect
    "Retention_Rate": 0.25  # Medium effect
}

# Compute required sample size for each metric
continuous_vars_sample_sizes = {metric: FTestAnovaPower().solve_power(effect, power=power, alpha=alpha, k_groups
                                for metric, effect in effect_sizes.items()}

formatted_sample_sizes = [f"Required sample size for {metric} per group: {size:.0f} = total: {size*2:.0f}"
                          for metric, size in continuous_vars_sample_sizes.items()]

# Display the sample sizes for continuous variables
display(formatted_sample_sizes)

# 2. Cramér's V (for Chi-Square Test): Dropout Rate
effect_size_dropout_rate = 0.2  # Small effect size for Cramér's V
df = 1  # Binary dropout rate

# Compute required sample size for Chi-Square test (same as before)
dropout_sample_size = GofChisquarePower().solve_power(effect_size_dropout_rate, power=power, alpha=alpha, nobs=

# Display the sample size for categorical variable
print(f"\nRequired sample size for Dropout_Rate per group: {dropout_sample_size:.0f} = total: {dropout_sample_s

# Find the largest sample size (either from ANOVA or Chi-Square)
withdrawal_rct_num_students = int(max(max(continuous_vars_sample_sizes.values()) * 2, dropout_sample_size * 2))

print(f"\nAppropriate sample size for Withdrawal RCT: {withdrawal_rct_num_students}")
```

```
['Required sample size for Post_Test_Quiz_Score per group: 89 = total: 178',
 'Required sample size for Study_Time per group: 198 = total: 396',
 'Required sample size for Retention_Rate per group: 128 = total: 255']
Required sample size for Dropout_Rate per group: 196 = total: 392

Appropriate sample size for Withdrawal RCT: 396
```

3.1.3 Withdrawal RCT Randomization (Blocking Factor)

In [204…

```
# Step 1: Collect Baseline Measures (Pre-Intervention Data)
withdrawal_rct_data = generate_baseline_data(withdrawal_rct_num_students)

# Step 2: Define the Blocking Factor (Performance_Level)
# Combine GPA, Baseline_Quiz_Score, and Tech_Savviness_Score into a single score
withdrawal_rct_data['Combined_Score'] = (
    withdrawal_rct_data['GPA'] + withdrawal_rct_data['Baseline_Quiz_Score'] + withdrawal_rct_data['Tech_Savvines
)

# Define Performance_Level based on quartiles of the Combined_Score
withdrawal_rct_data['Performance_Level'] = pd.qcut(
    withdrawal_rct_data['Combined_Score'],
    q=3,  # Divide into 3 equal-sized blocks (Low, Medium, High)
    labels=['Low', 'Medium', 'High']
)

# Drop the temporary 'Combined_Score' column
withdrawal_rct_data.drop(columns=['Combined_Score'], inplace=True)

# Step 3: Blocked Random Assignment
# Initialize the RCT_Withdrawal_Group column
withdrawal_rct_data['RCT_Withdrawal_Group'] = None

# Perform random assignment within each block
for level in withdrawal_rct_data['Performance_Level'].unique():
    # Filter data for the current block
    block_data = withdrawal_rct_data[withdrawal_rct_data['Performance_Level'] == level]

    # Shuffle the indices for randomization
    np.random.seed(18)  # Set seed for reproducibility
    shuffled_indices = np.random.permutation(block_data.index)

    # Split the shuffled indices into two equal groups
    half = len(shuffled_indices) // 2
    withdrawal_rct_data.loc[shuffled_indices[:half], 'RCT_Withdrawal_Group'] = 'Withdrawal'
    withdrawal_rct_data.loc[shuffled_indices[half:], 'RCT_Withdrawal_Group'] = 'Control'
```

3.1.4: Withdrawal RCT Implement Intervention (Blocking Factor)

Intervention Changes

1. Time Period Assignment (Withdrawal Group Only)

- The **Withdrawal Group** is split into two phases:
  - **Pre-Withdrawal (First 4 Weeks):** Students have access to adaptive learning.
  - **Post-Withdrawal (Last 4 Weeks):** The adaptive learning intervention is removed.

2. Final Quiz Score

- **Pre-Withdrawal Period:**
  - Students in the withdrawal group benefit from **adaptive learning**, leading to a **performance boost**.
  - Final quiz scores improve by an average of **10 points** with some variation.
- **Post-Withdrawal Period:**
  - Performance **declines** due to the removal of adaptive learning.
  - The final quiz score increase drops to an average of **5 points**.
- **Control Group:**
  - No intervention; students maintain **adaptive learning**.
  - Final quiz scores improve by an average of **10 points** with some variation.

3. Study Time

- **Pre-Withdrawal Period:**
  - Students in the withdrawal group dedicate **more time to studying** (~400 minutes on average), as adaptive learning enhances engagement.
- **Post-Withdrawal Period:**
  - Study time **decreases** (~350 minutes on average) after the intervention is removed.
- **Control Group:**
  - Study time remains relatively **higher and stable** (~400 minutes on average).

4. Retention Rate

- **Pre-Withdrawal Period:**
  - Students in the withdrawal group exhibit a **higher retention rate** (~90%), as adaptive learning helps sustain engagement.
- **Post-Withdrawal Period:**
  - Retention rate **drops** (~85%) following the removal of adaptive learning.
- **Control Group:**
  - Retention remains **higher** (~90%).

5. Dropout Rate

- **Pre-Withdrawal Period:**
  - The dropout rate is **lower** (~10%), since students are actively engaged with adaptive learning.
- **Post-Withdrawal Period:**
  - Dropout rate **increases** (~20%) after adaptive learning is withdrawn.
- **Control Group:**
  - Dropout rate is **lower throughout** (~10%).

Summary of the Intervention Effects

- **Adaptive learning improves performance, study time, and retention while lowering dropout rates.**
- **Once withdrawn, performance and retention drop, while dropout rates increase.**
- **The control group experiences significant positive change, maintaining higher engagement and performance.**

In [205…

```
# Step 3: Apply Pre-Withdrawal, Post-Withdrawal and Control groups interventions

# Add Time Period Column ONLY for the Withdrawal Group
np.random.seed(18)

# Initialize Time_Period as None for all students
withdrawal_rct_data['Time_Period'] = '8-weeks'

# Assign Time_Period only to the Withdrawal group
withdrawal_rct_data.loc[withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal', 'Time_Period'] = np.random
    ['Pre-Withdrawal', 'Post-Withdrawal'],
    size=withdrawal_rct_data[withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal'].shape[0]
)

# Step 4: Simulate Data for Pre-Withdrawal and Post-Withdrawal Periods

# Pre-Withdrawal Period (First 6 Weeks) - Withdrawal Group Only
pre_withdrawal_mask = (
    (withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal') &
    (withdrawal_rct_data['Time_Period'] == 'Pre-Withdrawal')
```

```
)
num_pre_withdrawal = pre_withdrawal_mask.sum()  # Number of rows in Pre-Withdrawal period

withdrawal_rct_data.loc[pre_withdrawal_mask, 'Post_Test_Quiz_Score'] = (
    withdrawal_rct_data.loc[pre_withdrawal_mask, 'Baseline_Quiz_Score'] +
    np.random.normal(10, 5, num_pre_withdrawal)
).astype(int)

withdrawal_rct_data.loc[pre_withdrawal_mask, 'Study_Time'] = (
    np.random.normal(400, 50, num_pre_withdrawal)  # Higher study time during adaptive learning
).astype(int)

withdrawal_rct_data.loc[pre_withdrawal_mask, 'Retention_Rate'] = (
    np.random.normal(90, 5, num_pre_withdrawal)  # Higher retention during adaptive learning
).round(2)

withdrawal_rct_data.loc[pre_withdrawal_mask, 'Dropout_Rate'] = (
    np.random.choice([0, 1], num_pre_withdrawal, p=[0.9, 0.1])  # Lower dropout rate during adaptive learning
).round(2)


# Post-Withdrawal Period (Last 3 Weeks) - Withdrawal Group Only
post_withdrawal_mask = (
    (withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal') &
    (withdrawal_rct_data['Time_Period'] == 'Post-Withdrawal')
)
num_post_withdrawal = post_withdrawal_mask.sum()  # Number of rows in Post-Withdrawal period

withdrawal_rct_data.loc[post_withdrawal_mask, 'Post_Test_Quiz_Score'] = (
    withdrawal_rct_data.loc[post_withdrawal_mask, 'Baseline_Quiz_Score'] +
    np.random.normal(5, 5, num_post_withdrawal)  # Drop in performance
).astype(int)

withdrawal_rct_data.loc[post_withdrawal_mask, 'Study_Time'] = (
    np.random.normal(350, 50, num_post_withdrawal)  # Reduced study time after withdrawal
).astype(int)

withdrawal_rct_data.loc[post_withdrawal_mask, 'Retention_Rate'] = (
    np.random.normal(85, 5, num_post_withdrawal)  # Lower retention after withdrawal
).round(2)

withdrawal_rct_data.loc[post_withdrawal_mask, 'Dropout_Rate'] = (
    np.random.choice([0, 1], num_post_withdrawal, p=[0.7, 0.3])  # Higher dropout rate after withdrawal
).round(2)

# Control Group (Static Content Throughout)
control_mask = (withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Control')
num_control = control_mask.sum()  # Number of rows in the Control group

withdrawal_rct_data.loc[control_mask, 'Post_Test_Quiz_Score'] = (
    withdrawal_rct_data.loc[control_mask, 'Baseline_Quiz_Score'] +
    np.random.normal(10, 5, num_control)
).astype(int)

withdrawal_rct_data.loc[control_mask, 'Study_Time'] = (
    np.random.normal(400, 50, num_control)
).astype(int)

withdrawal_rct_data.loc[control_mask, 'Retention_Rate'] = (
    np.random.normal(90, 5, num_control)
).round(2)

withdrawal_rct_data.loc[control_mask, 'Dropout_Rate'] = (
    np.random.choice([0, 1], num_control, p=[0.9, 0.1])
).round(2)

# Calculate Improvement Score
withdrawal_rct_data['Improvement_Score'] = (
    withdrawal_rct_data['Post_Test_Quiz_Score'] - withdrawal_rct_data['Baseline_Quiz_Score']
).astype(int)
```

In [206…

```
print(withdrawal_rct_data.info())
print('\n')
print(withdrawal_rct_data.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396 entries, 0 to 395
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Student_ID            396 non-null    int64
 1   Age                   396 non-null    int64
 2   GPA                   396 non-null    float64
 3   Study_Hours_Per_Week  396 non-null    int64
 4   Tech_Savviness_Score  396 non-null    int64
 5   Learning_Style        396 non-null    object
 6   Baseline_Quiz_Score   396 non-null    int64
 7   Attention_Span        396 non-null    int64
 8   Motivation_Level      396 non-null    int64
 9   Prior_Online_Exp      396 non-null    int64
 10  Performance_Level     396 non-null    category
 11  RCT_Withdrawal_Group  396 non-null    object
 12  Time_Period           396 non-null    object
 13  Post_Test_Quiz_Score  396 non-null    float64
 14  Study_Time            396 non-null    float64
 15  Retention_Rate        396 non-null    float64
 16  Dropout_Rate          396 non-null    float64
 17  Improvement_Score     396 non-null    int64
dtypes: category(1), float64(5), int64(9), object(3)
memory usage: 53.2+ KB
None
```

	Student_ID	Age	GPA	Study_Hours_Per_Week	Tech_Savviness_Score	\
0	1	22	2.9	61	-3	
1	2	26	2.5	36	0	
2	3	21	3.0	60	4	
3	4	22	3.3	55	6	
4	5	22	3.5	46	6	

	Learning_Style	Baseline_Quiz_Score	Attention_Span	Motivation_Level	\
0	Kinesthetic	57	37	7	
1	Visual	57	48	3	
2	Auditory	86	28	7	
3	Auditory	65	39	8	
4	Kinesthetic	73	37	5	

	Prior_Online_Exp	Performance_Level	RCT_Withdrawal_Group	Time_Period	\
0	0	Low	Withdrawal	Pre-Withdrawal	
1	1	Low	Withdrawal	Post-Withdrawal	
2	0	High	Withdrawal	Pre-Withdrawal	
3	1	Medium	Withdrawal	Post-Withdrawal	
4	0	High	Withdrawal	Pre-Withdrawal	

	Post_Test_Quiz_Score	Study_Time	Retention_Rate	Dropout_Rate	\
0	74.0	372.0	90.91	0.0	
1	62.0	374.0	88.52	0.0	
2	97.0	425.0	95.02	0.0	
3	63.0	345.0	77.45	0.0	
4	83.0	397.0	100.36	0.0	

	Improvement_Score
0	17
1	5
2	11
3	-2
4	10

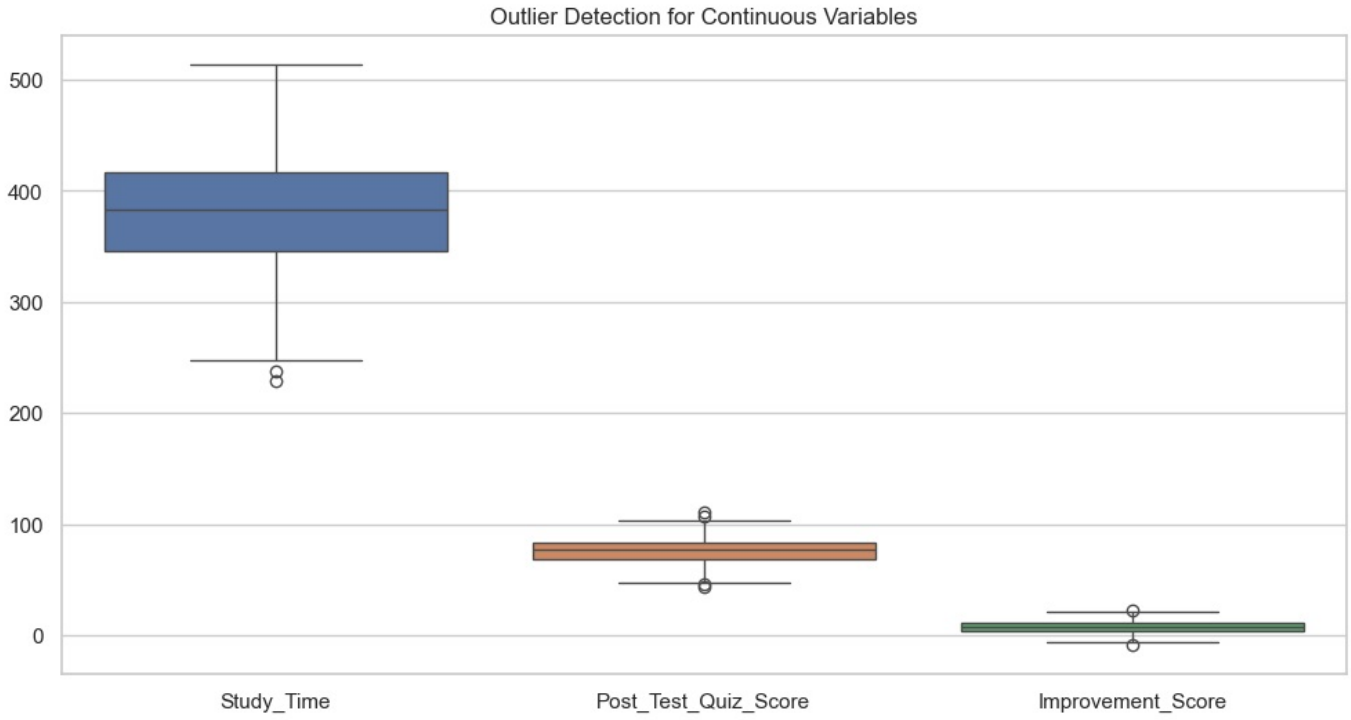
3.1.5: Matched-Pairs RCT Data Cleaning (Blocking Factor)

In [207…

```python
# 1. Check for Missing Data
print("Missing Values Check:\n")
print(withdrawal_rct_data.isnull().sum())

# Replace placeholder codes (e.g., -18) with NaN if present
withdrawal_rct_data.replace(-18, np.nan, inplace=True)

# Drop rows with critical missing values (if any)
critical_cols = ['Post_Test_Quiz_Score', 'Retention_Rate','Dropout_Rate', 'RCT_Withdrawal_Group']
withdrawal_rct_data.dropna(subset=critical_cols, inplace=True)

# 2. Remove Duplicates
print(f"\nInitial Shape: {withdrawal_rct_data.shape}")
withdrawal_rct_data.drop_duplicates(subset=['Student_ID'], keep='first', inplace=True)
print(f"Post-Deduplication Shape: {withdrawal_rct_data.shape}")

# 3. Verify Randomization Balance
```

```
print("\nGroup Balance:")
print(withdrawal_rct_data['RCT_Withdrawal_Group'].value_counts())

# Compare baseline covariates between groups
print("\nBaseline Covariate Balance:")
baseline_cols = ['GPA', 'Baseline_Quiz_Score', 'Study_Hours_Per_Week']
for col in baseline_cols:
    withdrawal = withdrawal_rct_data[withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal'][col]
    control = withdrawal_rct_data[withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Control'][col]
    t_stat, p_value = ttest_ind(withdrawal, control, nan_policy='omit')
    print(f"{col}: t = {t_stat:.2f}, p = {p_value:.4f}")

# 4. Detect and Handle Outliers
continuous_vars = ['Study_Time', 'Post_Test_Quiz_Score', 'Improvement_Score']

plt.figure(figsize=(12, 6))
sns.boxplot(data=withdrawal_rct_data[continuous_vars])
plt.title("Outlier Detection for Continuous Variables")
plt.show()

# Cap outliers using IQR
for var in continuous_vars:
    q1 = withdrawal_rct_data[var].quantile(0.25)
    q3 = withdrawal_rct_data[var].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    withdrawal_rct_data[var] = np.where(withdrawal_rct_data[var] < lower_bound, lower_bound,
                                        np.where(withdrawal_rct_data[var] > upper_bound, upper_bound,
                                                 withdrawal_rct_data[var]))

# 5. Validate Variable Ranges/Types
# Ensure binary variables are 0/1
withdrawal_rct_data['Dropout_Rate'] = withdrawal_rct_data['Dropout_Rate'].apply(lambda x: 1 if x == 1 else 0)

# Standardize categorical variables
withdrawal_rct_data['Learning_Style'] = withdrawal_rct_data['Learning_Style'].str.strip().str.title()

# Ensure valid percentage ranges
withdrawal_rct_data['Retention_Rate'] = withdrawal_rct_data['Retention_Rate'].clip(0, 100)

# 6. Save Cleaned Data
clean_withdrawal_rct_data = withdrawal_rct_data.copy()

print("\nData cleaning complete. Cleaned data saved as clean_withdrawal_rct_data")
```

```
Missing Values Check:

Student_ID              0
Age                     0
GPA                     0
Study_Hours_Per_Week    0
Tech_Savviness_Score    0
Learning_Style          0
Baseline_Quiz_Score     0
Attention_Span          0
Motivation_Level        0
Prior_Online_Exp        0
Performance_Level       0
RCT_Withdrawal_Group    0
Time_Period             0
Post_Test_Quiz_Score    0
Study_Time              0
Retention_Rate          0
Dropout_Rate            0
Improvement_Score       0
dtype: int64

Initial Shape: (396, 18)
Post-Deduplication Shape: (396, 18)

Group Balance:
RCT_Withdrawal_Group
Control       199
Withdrawal    197
Name: count, dtype: int64

Baseline Covariate Balance:
GPA: t = -0.47, p = 0.6386
Baseline_Quiz_Score: t = -0.05, p = 0.9620
Study_Hours_Per_Week: t = 0.87, p = 0.3844
```




```
Data cleaning complete. Cleaned data saved as clean_withdrawal_rct_data
```

3.1.6: Withdrawal RCT Perform Analysis (Blocking Factor)

In [208…

```python
# Metrics to analyze
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Step 1: Within-Group Analysis (Repeated Measures ANOVA)
within_group_results = {}
post_hoc_within_results = {}

for metric in metrics:
    pre_withdrawal = clean_withdrawal_rct_data[
        (clean_withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal') &
        (clean_withdrawal_rct_data['Time_Period'] == 'Pre-Withdrawal')
    ][metric]

    post_withdrawal = clean_withdrawal_rct_data[
        (clean_withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal') &
        (clean_withdrawal_rct_data['Time_Period'] == 'Post-Withdrawal')
    ][metric]

    f_stat, p_value = f_oneway(pre_withdrawal, post_withdrawal)

    better_group = 'Post-Withdrawal' if post_withdrawal.mean() > pre_withdrawal.mean() else 'Pre-Withdrawal'

    within_group_results[metric] = {
        'Pre-Withdrawal': pre_withdrawal.mean().round(2),
        'Post-Withdrawal': post_withdrawal.mean().round(2),
        'Better Group': better_group,
        'F-Statistic': f_stat.round(2),
        'p-value': p_value.round(4),
        'Significance': 'Significant' if p_value < 0.05 else 'Not Significant'
    }

    # Post-Hoc Analysis (Tukey's HSD) if the Repeated Measures ANOVA is significant
    if p_value < 0.05:
        # Combine pre- and post-withdrawal data for Tukey's HSD
        combined_data = pd.concat([pre_withdrawal, post_withdrawal])
        combined_groups = ['Pre-Withdrawal'] * len(pre_withdrawal) + ['Post-Withdrawal'] * len(post_withdrawal)

        # Perform Tukey's HSD
        tukey_results = pairwise_tukeyhsd(endog=combined_data,
                                          groups=combined_groups,
                                          alpha=0.05)
        post_hoc_within_results[metric] = tukey_results

# Convert to DataFrame
within_group_results_df = pd.DataFrame(within_group_results).T
print("1. Within-Group Analysis - Comparison of Pre-Withdrawal (Adaptive) vs. Post-Withdrawal (Static)\n")
display(within_group_results_df)
```

```
# Display Post-Hoc Results for Within-Group Analysis
for metric, result in post_hoc_within_results.items():
    print(f"\nPost-Hoc Analysis (Tukey's HSD) for {metric} (Within-Group):\n")
    print(result.summary())
    print("\n")

# Chi-Square Analysis for Dropout_Rate between Pre-Withdrawal vs. Post-Withdrawal within the Withdrawal group
clean_withdrawal_rct_data_copy = clean_withdrawal_rct_data[clean_withdrawal_rct_data['Time_Period'] != '8-weeks

dropout_contingency_table_within = pd.crosstab(
    clean_withdrawal_rct_data_copy[
        clean_withdrawal_rct_data_copy['RCT_Withdrawal_Group'] == 'Withdrawal'
    ]['Time_Period'],
    clean_withdrawal_rct_data_copy[
        clean_withdrawal_rct_data_copy['RCT_Withdrawal_Group'] == 'Withdrawal'
    ]['Dropout_Rate']
)

chi2_stat_within, p_value_within, dof_within, expected_within = chi2_contingency(dropout_contingency_table_with

dropout_rates_within = clean_withdrawal_rct_data_copy[
    clean_withdrawal_rct_data_copy['RCT_Withdrawal_Group'] == 'Withdrawal'
].groupby('Time_Period')['Dropout_Rate'].mean()

better_period_within = 'Pre-Withdrawal' if dropout_rates_within['Pre-Withdrawal'] < dropout_rates_within['Post-
significance_within = 'Significant' if p_value_within < 0.05 else 'Not Significant'

chi_square_results_within_df = pd.DataFrame({
    'Metric': ['Dropout Rate'],
    'Pre-Withdrawal': [dropout_rates_within['Pre-Withdrawal']],
    'Post-Withdrawal': [dropout_rates_within['Post-Withdrawal']],
    'Better Period': [better_period_within],
    'Chi-Square Statistic': [chi2_stat_within],
    'p-value': [p_value_within],
    'Significance': [significance_within]
})

print("\n1. Within-Group Analysis - Chi-Square Analysis of Dropout Rate Between Pre-Withdrawal vs. Post-Withdra
display(chi_square_results_within_df)
```

```
1. Within-Group Analysis - Comparison of Pre-Withdrawal (Adaptive) vs. Post-Withdrawal (Static)
```

	Pre-Withdrawal	Post-Withdrawal	Better Group	F-Statistic	p-value	Significance
Post_Test_Quiz_Score	77.79	73.31	Pre-Withdrawal	9.24	0.0027	Significant
Improvement_Score	9.43	4.17	Pre-Withdrawal	59.36	0.0	Significant
Study_Time	386.3	350.38	Pre-Withdrawal	29.12	0.0	Significant
Retention_Rate	90.52	84.66	Pre-Withdrawal	67.68	0.0	Significant

```
Post-Hoc Analysis (Tukey's HSD) for Post_Test_Quiz_Score (Within-Group):

        Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
     group1         group2     meandiff p-adj  lower  upper  reject
-----------------------------------------------------------------------------
Post-Withdrawal Pre-Withdrawal   4.4798 0.0027 1.5732 7.3864   True
-----------------------------------------------------------------------------


Post-Hoc Analysis (Tukey's HSD) for Improvement_Score (Within-Group):

       Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
     group1         group2     meandiff p-adj lower upper  reject
-----------------------------------------------------------------------------
Post-Withdrawal Pre-Withdrawal   5.2538   0.0 3.909 6.5987   True
-----------------------------------------------------------------------------


Post-Hoc Analysis (Tukey's HSD) for Study_Time (Within-Group):

        Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
     group1         group2     meandiff p-adj  lower   upper  reject
-----------------------------------------------------------------------------
Post-Withdrawal Pre-Withdrawal  35.9271   0.0 22.7977 49.0564   True
-----------------------------------------------------------------------------


Post-Hoc Analysis (Tukey's HSD) for Retention_Rate (Within-Group):

       Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
     group1         group2     meandiff p-adj lower  upper  reject
-----------------------------------------------------------------------------
Post-Withdrawal Pre-Withdrawal   5.8605   0.0 4.4556 7.2654   True
-----------------------------------------------------------------------------


1. Within-Group Analysis - Chi-Square Analysis of Dropout Rate Between Pre-Withdrawal vs. Post-Withdrawal
```

	Metric	Pre-Withdrawal	Post-Withdrawal	Better Period	Chi-Square Statistic	p-value	Significance
0	Dropout Rate	0.041667	0.237624	Pre-Withdrawal	13.934693	0.000189	Significant

In [209…

```python
# Step 2: Between-Group Analysis (Independent t-tests)
between_group_results = {}
post_hoc_between_results = {}

for metric in metrics:
    post_withdrawal = clean_withdrawal_rct_data[
        (clean_withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal') &
        (clean_withdrawal_rct_data['Time_Period'] == 'Post-Withdrawal')
    ][metric]

    control = clean_withdrawal_rct_data[
        (clean_withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Control')
    ][metric].sample(n=len(post_withdrawal), random_state=200)  # Match sample size

    t_stat, p_value = ttest_ind(post_withdrawal, control)

    better_group = 'Post-Withdrawal' if post_withdrawal.mean() > control.mean() else 'Control'

    between_group_results[metric] = {
        'Post-Withdrawal': post_withdrawal.mean().round(2),
        'Control': control.mean().round(2),
        'Better Group': better_group,
        'F-Statistic': t_stat.round(2),
        'p-value': p_value.round(4),
        'Significance': 'Significant' if p_value < 0.05 else 'Not Significant'
    }

    # Post-Hoc Analysis (Tukey's HSD) if the Independent t-test is significant
    if p_value < 0.05:
        # Combine post-withdrawal and control data for Tukey's HSD
        combined_data = pd.concat([post_withdrawal, control])
        combined_groups = ['Post-Withdrawal'] * len(post_withdrawal) + ['Control'] * len(control)
```

```
        # Perform Tukey's HSD
        tukey_results = pairwise_tukeyhsd(endog=combined_data,
                                          groups=combined_groups,
                                          alpha=0.05)
        post_hoc_between_results[metric] = tukey_results

# Convert to DataFrame
between_group_results_df = pd.DataFrame(between_group_results).T
print("\n\n2. Between-Group Analysis - Comparison of Post-Withdrawal (Static) vs. Control (Adaptive) with Block
display(between_group_results_df)

# Display Post-Hoc Results for Between-Group Analysis
for metric, result in post_hoc_between_results.items():
    print(f"\nPost-Hoc Analysis (Tukey's HSD) for {metric} (Between-Group):\n")
    print(result.summary())
    print("\n")

# Chi-Square Analysis for Dropout_Rate between Post-Withdrawal vs. Control groups
clean_withdrawal_rct_data_copy = clean_withdrawal_rct_data[clean_withdrawal_rct_data['Time_Period'] != 'Pre-Wit

dropout_contingency_table_between = pd.crosstab(
    clean_withdrawal_rct_data_copy['Time_Period'],
    clean_withdrawal_rct_data_copy['Dropout_Rate']
)

chi2_stat_between, p_value_between, dof_between, expected_between = chi2_contingency(dropout_contingency_table_

dropout_rates_between = clean_withdrawal_rct_data_copy.groupby('Time_Period')['Dropout_Rate'].mean()

better_group_between = 'Post-Withdrawal' if dropout_rates_between['Post-Withdrawal'] < dropout_rates_between['8
significance_between = 'Significant' if p_value_between < 0.05 else 'Not Significant'

chi_square_results_between_df = pd.DataFrame({
    'Metric': ['Dropout Rate'],
    'Post-Withdrawal': [dropout_rates_between['Post-Withdrawal']],
    'Control': [dropout_rates_between['8-weeks']],
    'Better Group': [better_group_between],
    'Chi-Square Statistic': [chi2_stat_between],
    'p-value': [p_value_between],
    'Significance': [significance_between]
})

print("\n2. Between-Group Analysis - Chi-Square Analysis of Dropout Rate Between Post-Withdrawal vs. Control wi
display(chi_square_results_between_df)
```

```
2. Between-Group Analysis - Comparison of Post-Withdrawal (Static) vs. Control (Adaptive) with Blocking Factors
```

	Post-Withdrawal	Control	Better Group	F-Statistic	p-value	Significance
Post_Test_Quiz_Score	73.31	78.13	Control	-3.09	0.0023	Significant
Improvement_Score	4.17	9.62	Control	-7.64	0.0	Significant
Study_Time	350.38	391.72	Control	-6.12	0.0	Significant
Retention_Rate	84.66	90.18	Control	-7.76	0.0	Significant

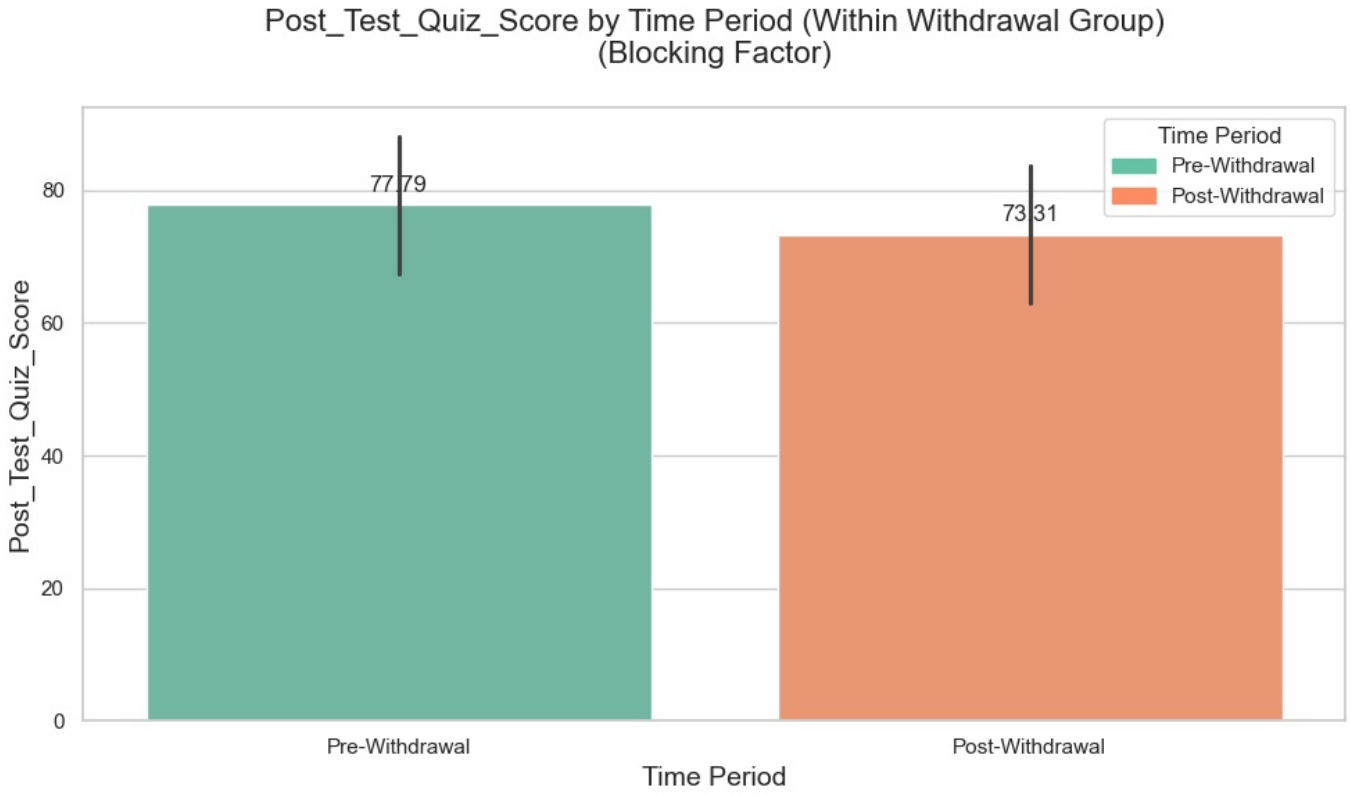
```
Post-Hoc Analysis (Tukey's HSD) for Post_Test_Quiz_Score (Between-Group):

     Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================
 group1      group2     meandiff p-adj   lower   upper  reject
-----------------------------------------------------------------
Control Post-Withdrawal  -4.8168 0.0023 -7.8952 -1.7384   True
-----------------------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Improvement_Score (Between-Group):

     Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================
 group1      group2     meandiff p-adj  lower   upper  reject
-----------------------------------------------------------------
Control Post-Withdrawal  -5.4505   0.0 -6.8568 -4.0442   True
-----------------------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Study_Time (Between-Group):

      Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================
 group1      group2     meandiff p-adj  lower    upper   reject
-----------------------------------------------------------------
Control Post-Withdrawal -41.3478   0.0 -54.6612 -28.0343   True
-----------------------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Retention_Rate (Between-Group):

     Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================
 group1      group2     meandiff p-adj  lower   upper  reject
-----------------------------------------------------------------
Control Post-Withdrawal  -5.5204   0.0 -6.9232 -4.1176   True
-----------------------------------------------------------------



2. Between-Group Analysis - Chi-Square Analysis of Dropout Rate Between Post-Withdrawal vs. Control with Blockin
g Factors
```

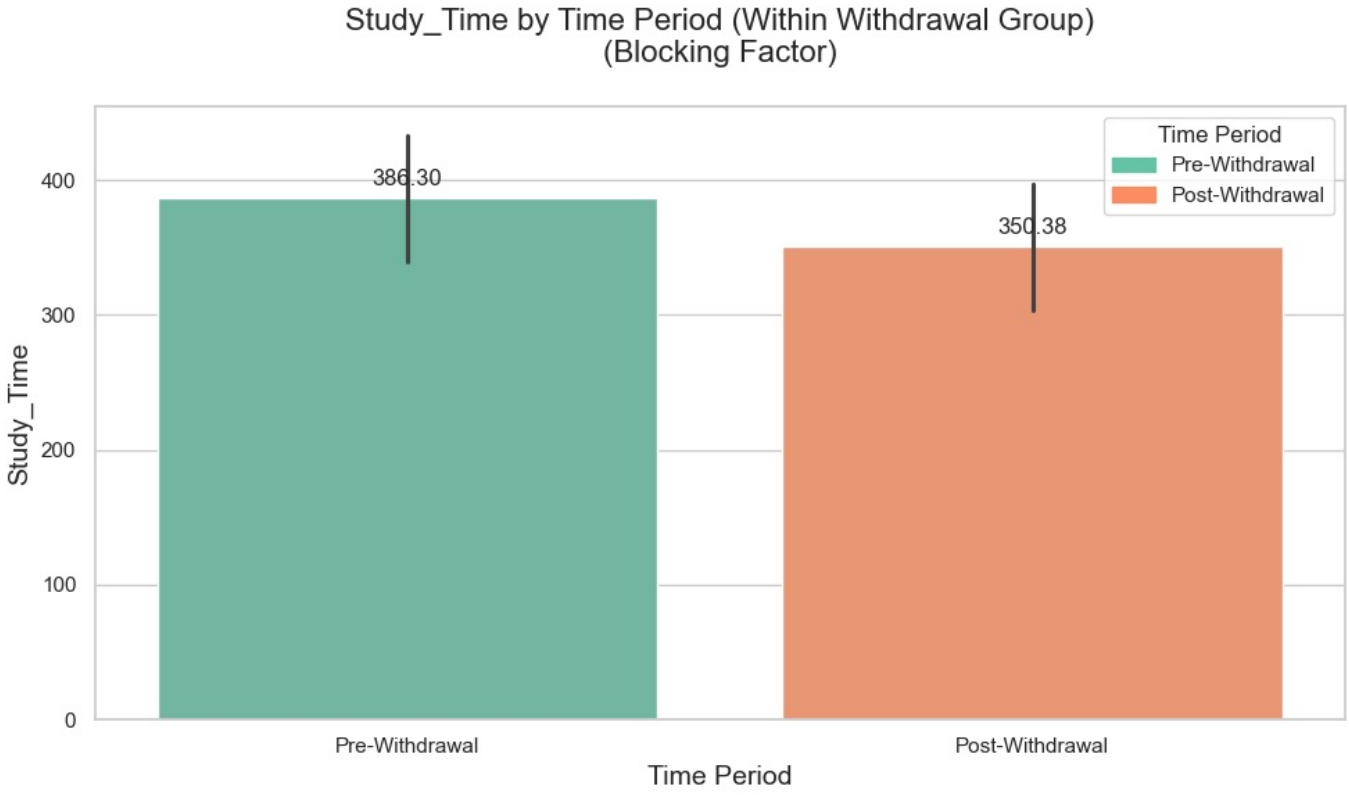
	Metric	Post-Withdrawal	Control	Better Group	Chi-Square Statistic	p-value	Significance
0	Dropout Rate	0.237624	0.085427	Control	11.894621	0.000563	Significant

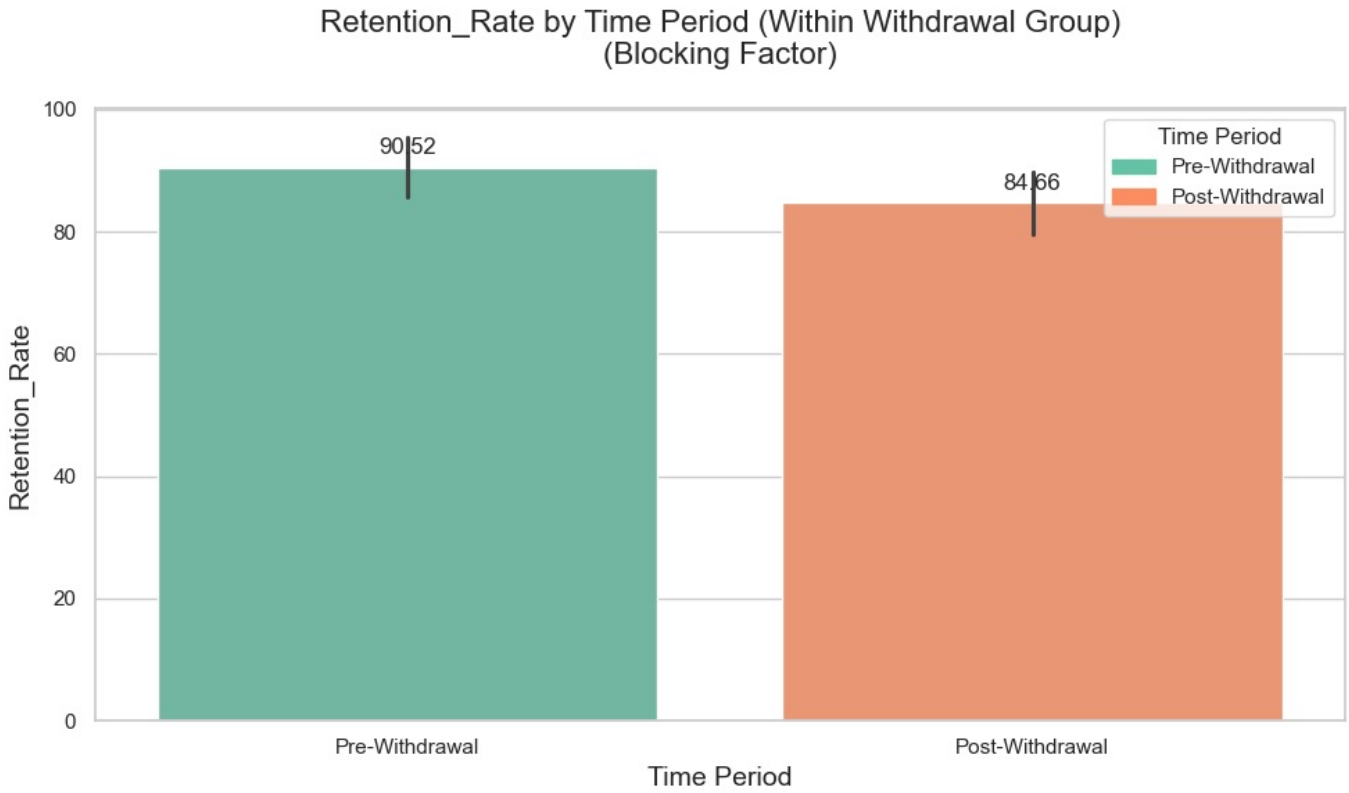
In [210…

```python
# Step 1: Visualize Within-Group Analysis (Pre-Withdrawal vs. Post-Withdrawal)
# Define metrics for visualization
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Create bar plots for each metric (Within-Group Analysis)
for metric in metrics:
    plt.figure(figsize=(10, 6))
    bar_plot = sns.barplot(
        x='Time_Period',
        y=metric,
        data=clean_withdrawal_rct_data[clean_withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal'],
        palette='Set2',  # Use Set2 palette
        ci='sd'  # Add error bars (standard deviation)
    )

    # Add labels and title
    plt.title(f'{metric} by Time Period (Within Withdrawal Group)\n(Blocking Factor)\n', fontsize=16)
    plt.xlabel('Time Period', fontsize=14)
    plt.ylabel(metric, fontsize=14)

    # Add annotations for better clarity
    for p in bar_plot.patches:
        bar_plot.annotate(
            f'{p.get_height():.2f}',
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='center',
            xytext=(0, 10),
            textcoords='offset points'
        )

    # Add legend to indicate Static and Adaptive groups
```

```
    handles = [plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[0]),  # Pre-Withdrawal (first color
               plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[1])]  # Post-Withdrawal (second colo
    labels = ['Pre-Withdrawal', 'Post-Withdrawal']
    plt.legend(handles, labels, title='Time Period', loc='upper right')

    # Display the plot
    plt.tight_layout()
    plt.show()
```



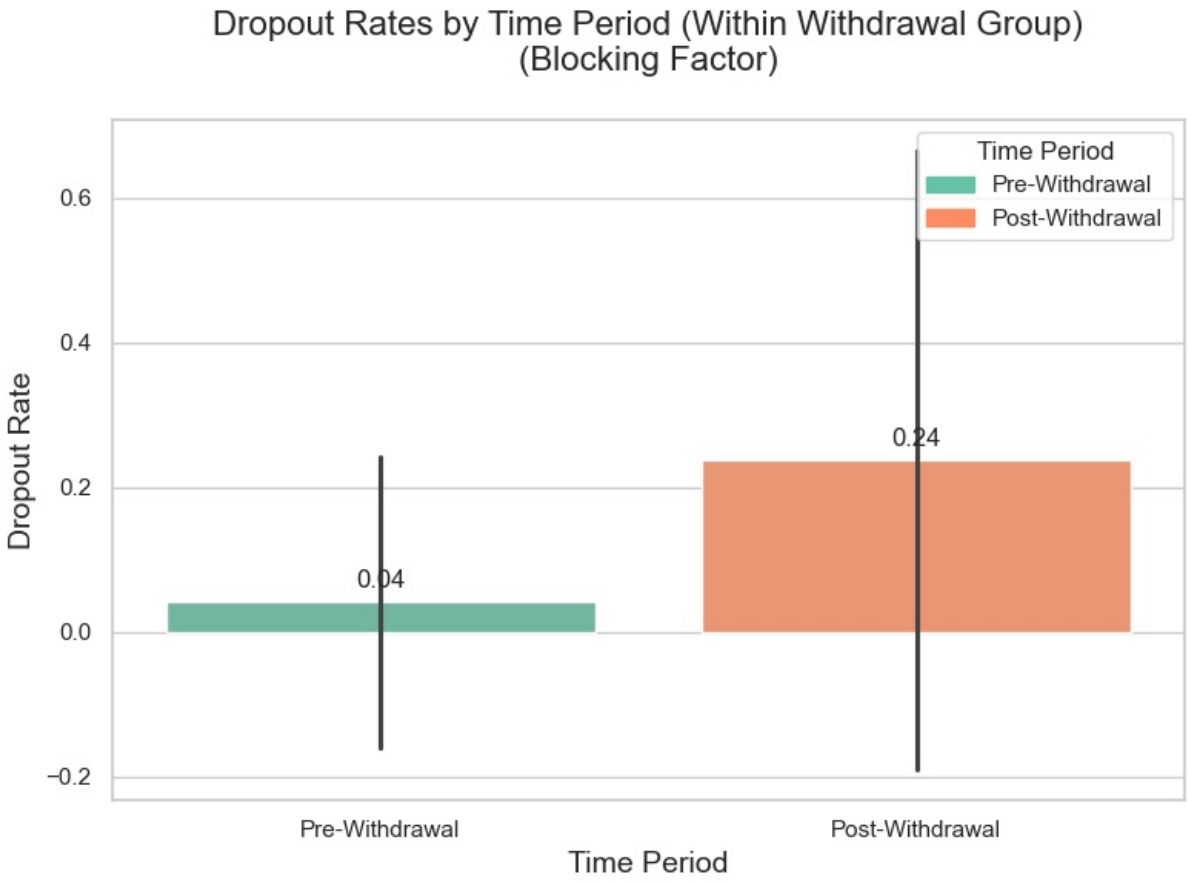




In [211…

```
# Within-Group Dropout Rates (Pre-Withdrawal vs. Post-Withdrawal)
plt.figure(figsize=(8, 6))
bar_plot = sns.barplot(
    x='Time_Period',
    y='Dropout_Rate',
    data=clean_withdrawal_rct_data[clean_withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal'],
    palette='Set2',  # Use Set2 palette
```

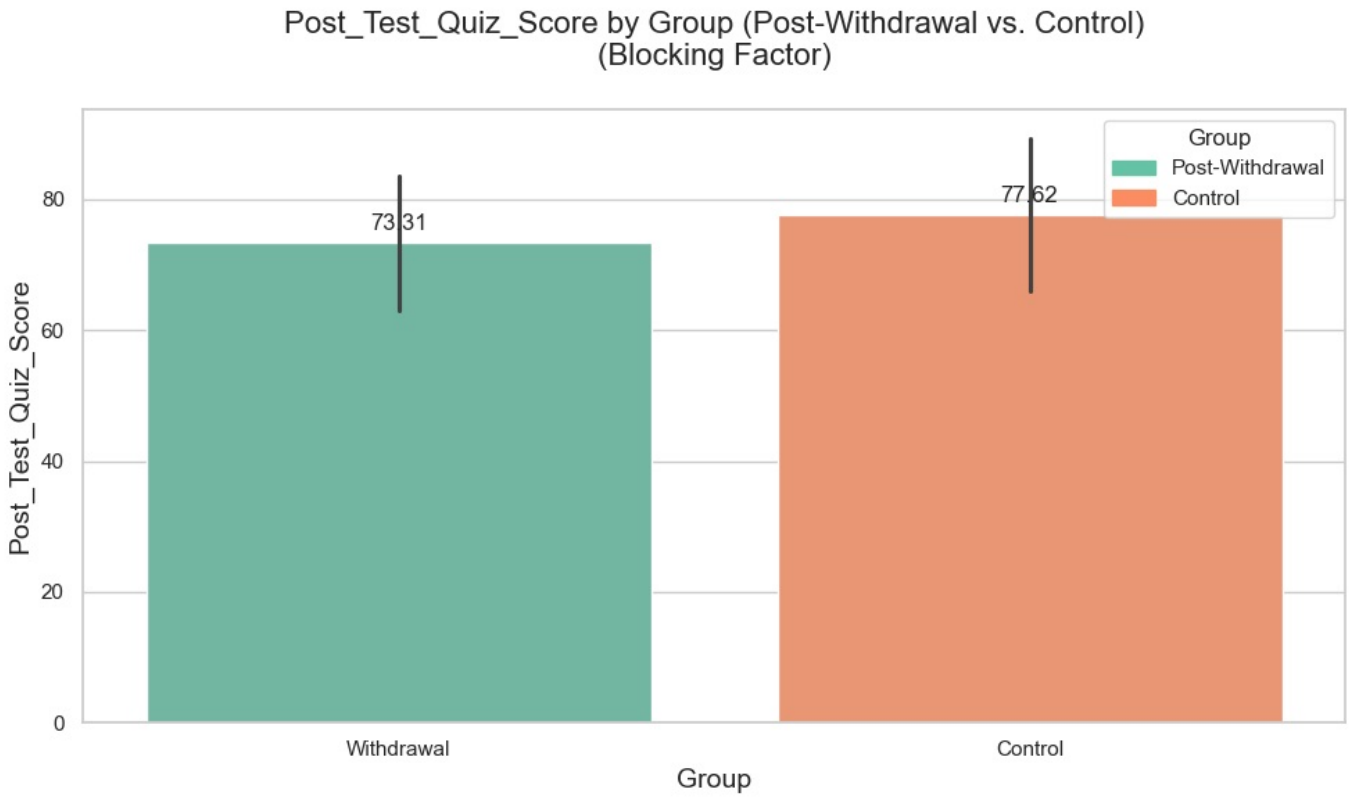
```
    ci='sd'
)

# Add labels and title
plt.title('Dropout Rates by Time Period (Within Withdrawal Group)\n(Blocking Factor)\n', fontsize=16)
plt.xlabel('Time Period', fontsize=14)
plt.ylabel('Dropout Rate', fontsize=14)

# Add annotations for better clarity
for p in bar_plot.patches:
    bar_plot.annotate(
        f'{p.get_height():.2f}',
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha='center', va='center',
        xytext=(0, 10),
        textcoords='offset points'
    )

# Add legend to indicate Static and Adaptive groups
handles = [plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[0]),  # Pre-Withdrawal (first color in S
           plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[1])]  # Post-Withdrawal (second color in
labels = ['Pre-Withdrawal', 'Post-Withdrawal']
plt.legend(handles, labels, title='Time Period', loc='upper right')

plt.tight_layout()
plt.show()
```

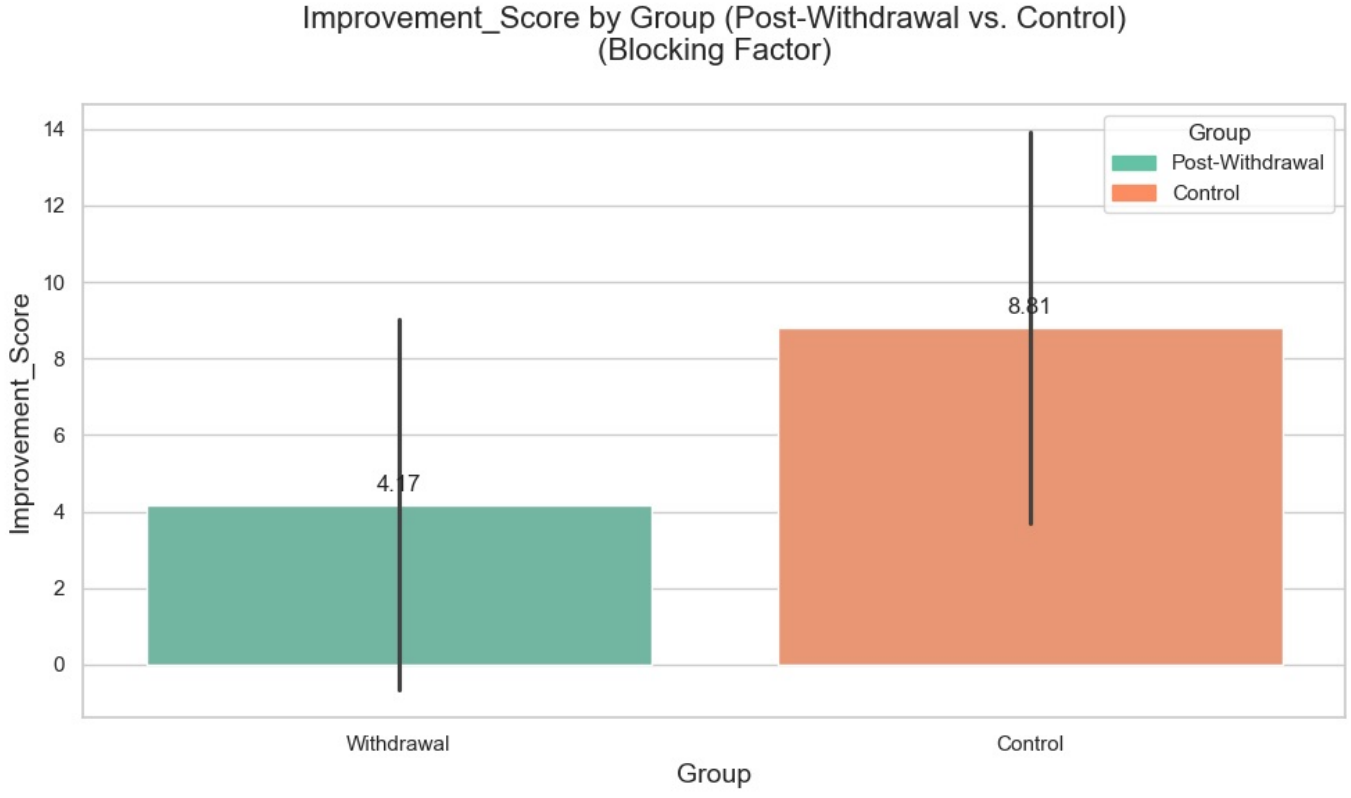


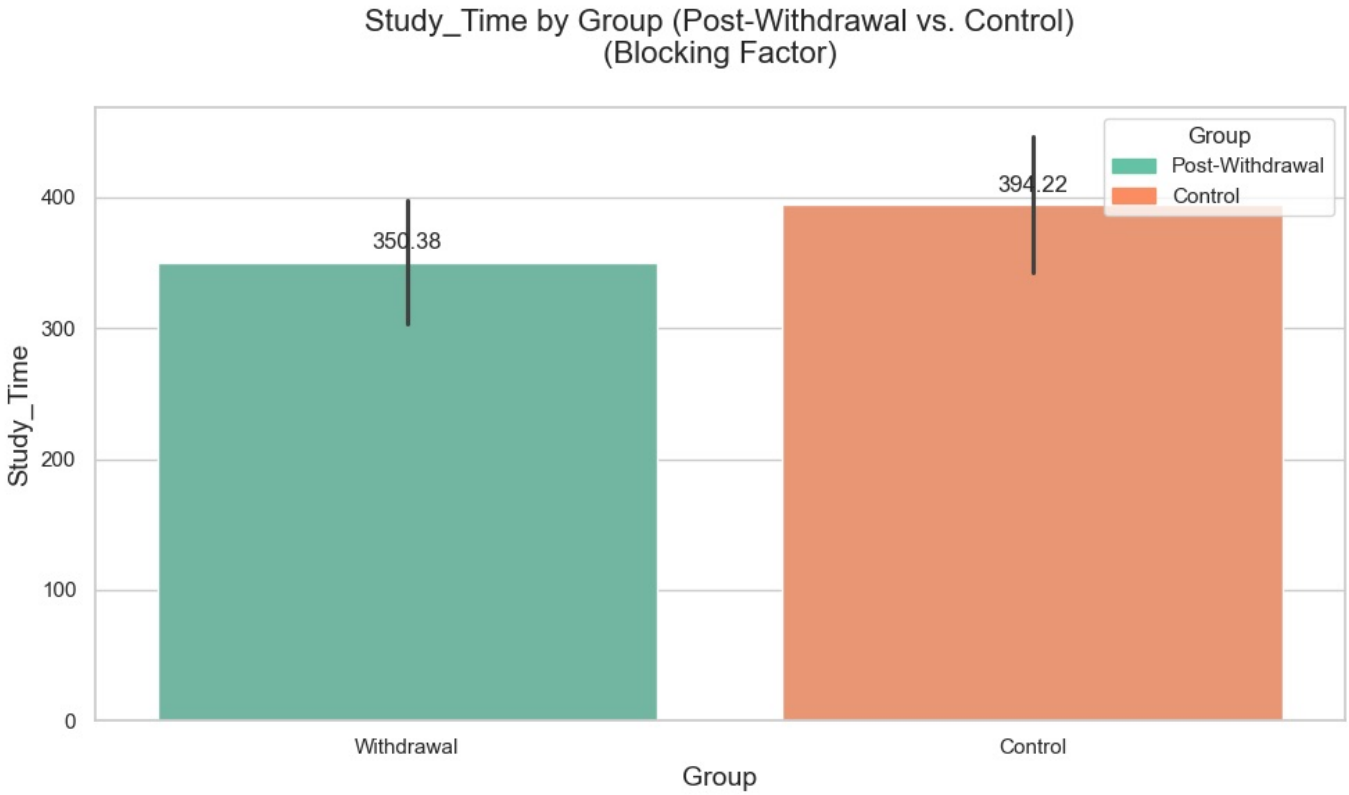
In [212…

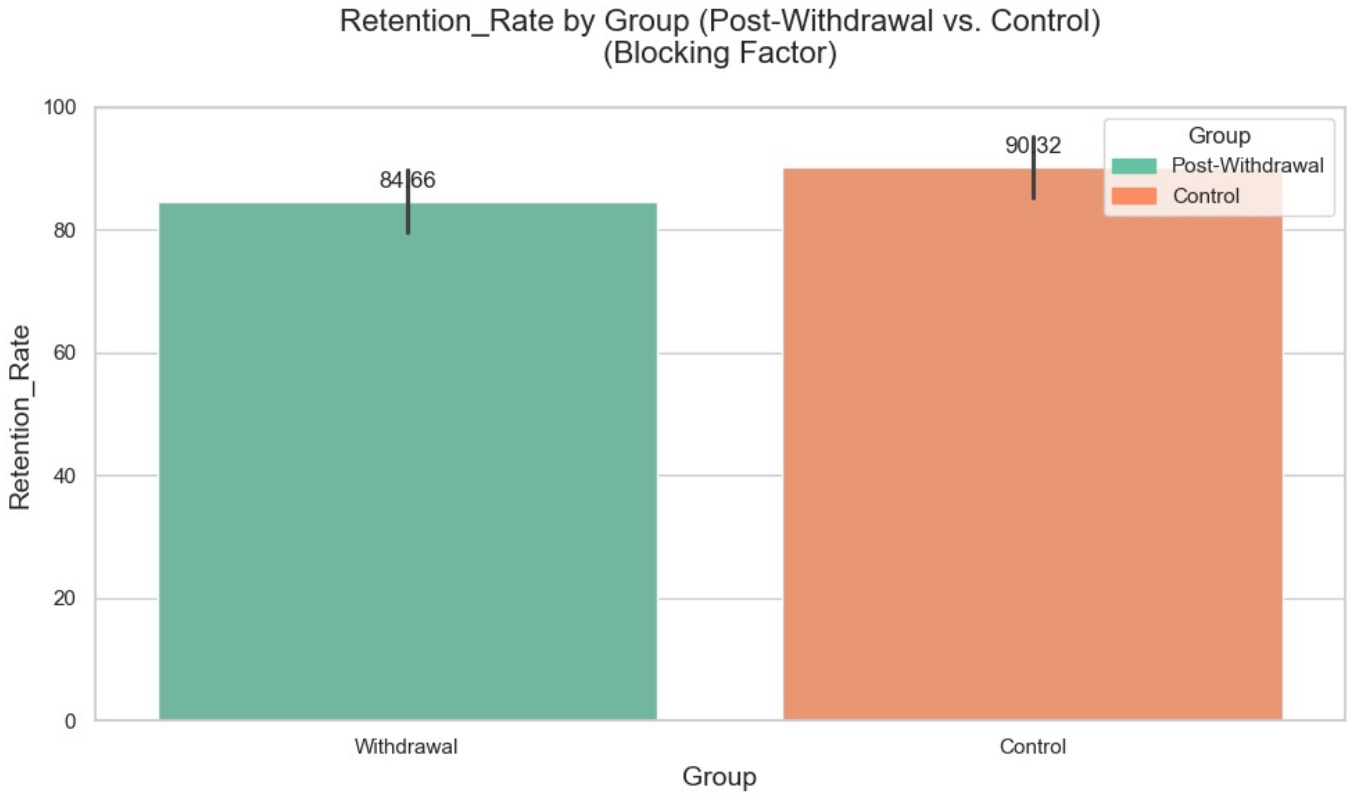
```
# Step 2: Visualize Between-Group Analysis (Post-Withdrawal vs. Control)
# Create bar plots for each metric (Between-Group Analysis)
for metric in metrics:
    plt.figure(figsize=(10, 6))
    bar_plot = sns.barplot(
        x='RCT_Withdrawal_Group',
        y=metric,
        data=clean_withdrawal_rct_data[clean_withdrawal_rct_data['Time_Period'] != 'Pre-Withdrawal'],
        palette='Set2',  # Use Set2 palette
        ci='sd'  # Add error bars (standard deviation)
    )

    # Add labels and title
    plt.title(f'{metric} by Group (Post-Withdrawal vs. Control)\n(Blocking Factor)\n', fontsize=16)
    plt.xlabel('Group', fontsize=14)
    plt.ylabel(metric, fontsize=14)

    # Add annotations for better clarity
    for p in bar_plot.patches:
        bar_plot.annotate(
            f'{p.get_height():.2f}',
```

```
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha='center', va='center',
            xytext=(0, 10),
            textcoords='offset points'
        )

    # Add legend to indicate Static and Adaptive groups
    handles = [plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[0]),  # Post-Withdrawal (first color
               plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[1])]  # Control (second color in Set
    labels = ['Post-Withdrawal', 'Control']
    plt.legend(handles, labels, title='Group', loc='upper right')

    # Display the plot
    plt.tight_layout()
    plt.show()
```





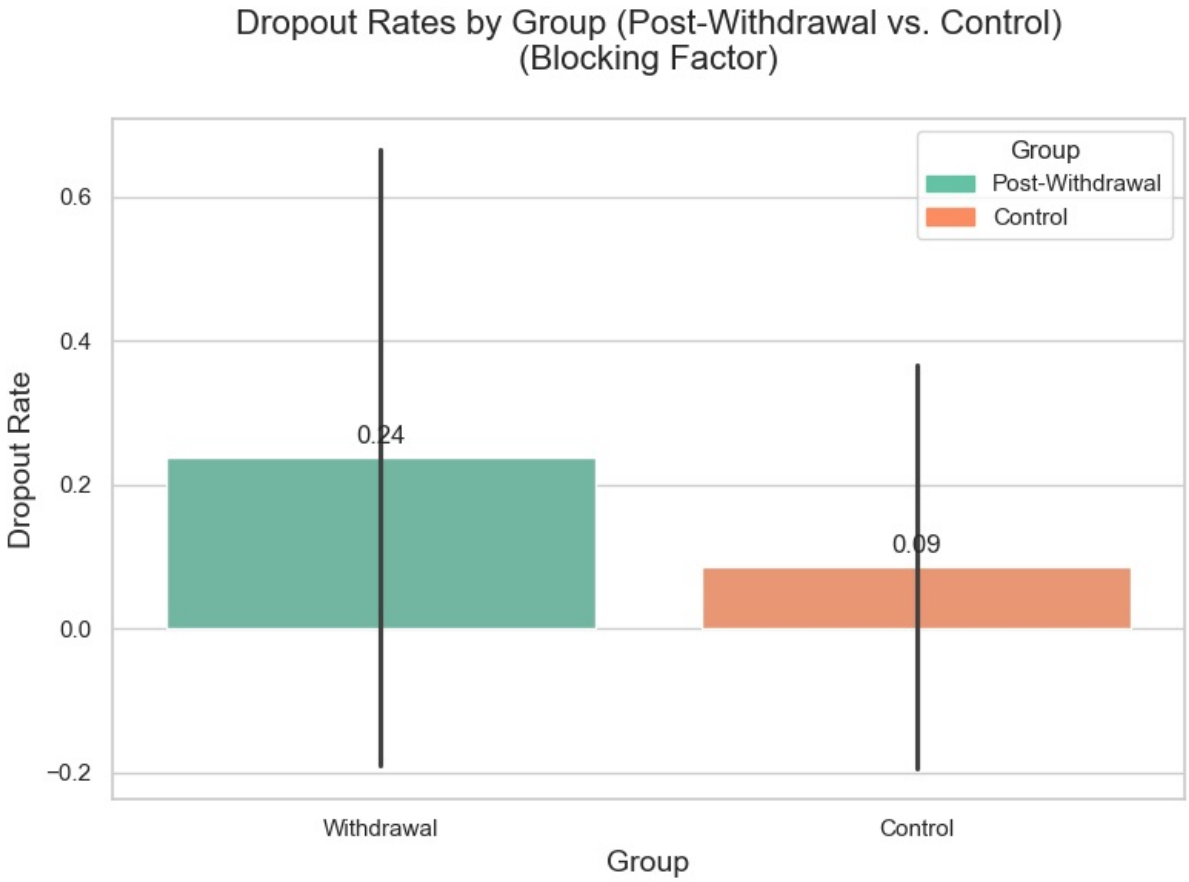




In [213…

```
# Between-Group Dropout Rates (Post-Withdrawal vs. Control)
plt.figure(figsize=(8, 6))
bar_plot = sns.barplot(
    x='RCT_Withdrawal_Group',
    y='Dropout_Rate',
    data=clean_withdrawal_rct_data[clean_withdrawal_rct_data['Time_Period'] != 'Pre-Withdrawal'],
    palette='Set2',  # Use Set2 palette
    ci='sd'
)

# Add labels and title
plt.title('Dropout Rates by Group (Post-Withdrawal vs. Control)\n(Blocking Factor)\n', fontsize=16)
plt.xlabel('Group', fontsize=14)
plt.ylabel('Dropout Rate', fontsize=14)
```

```
# Add annotations for better clarity
for p in bar_plot.patches:
    bar_plot.annotate(
        f'{p.get_height():.2f}',
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha='center', va='center',
        xytext=(0, 10),
        textcoords='offset points'
    )

# Add legend to indicate Static and Adaptive groups
handles = [plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[0]),  # Post-Withdrawal (first color in S
           plt.Rectangle((0, 0), 1, 1, color=sns.color_palette('Set2')[1])]  # Control (second color in Set2)
labels = ['Post-Withdrawal', 'Control']
plt.legend(handles, labels, title='Group', loc='upper right')

plt.tight_layout()
plt.show()
```



3.1.7 Interpretation of Withdrawal RCT Results (Blocking Factor)

Here's a revised version of your analysis based on the new results:

1. Withdrawal Negatively Affects Quiz Performance

- The **Post-Test Quiz Score** significantly **decreased** after withdrawal (**77.79** → **73.31**, p = 0.0027), indicating that students performed worse post-withdrawal.
- Compared to the **Control group** (**73.31 vs. 78.13**, p = 0.0023), the difference was statistically significant, suggesting that the **Control group** performed significantly better than the **Withdrawal group**.

2. Learning Improvement Declines After Withdrawal

- The **Improvement Score** significantly **dropped** post-withdrawal (**9.43** → **4.17**, p < 0.001), indicating that students benefited more from the initial adaptive learning environment.
- Compared to the **Control group** (**4.17 vs. 9.62**, p < 0.001), post-withdrawal students showed a significant decline in improvement, reinforcing the idea that **adaptive learning** leads to better outcomes than static content.

3. Study Time Decreases After Withdrawal

- **Study time** significantly **decreased** post-withdrawal (**386.3** → **350.38 minutes**, p < 0.001), suggesting that students spent less time

studying after the shift to static content.
- Compared to the **Control group** (**350.38 vs. 391.72 minutes**, $p < 0.001$), students in the post-withdrawal phase studied **significantly less**, indicating that static content was less engaging than adaptive learning.

4. Retention Rate Declines Post-Withdrawal

- The **Retention Rate** significantly **decreased** post-withdrawal (**90.52%** → **84.66%**, $p < 0.001$), showing a decline in retention after the shift from adaptive to static content.
- When compared to the **Control group** (**84.66% vs. 90.18%**, $p < 0.001$), the **Control group** retained more students, suggesting that adaptive learning positively affects retention.

5. Dropout Rate Increases After Withdrawal

- The **Dropout Rate** significantly **increased** post-withdrawal (**4.17%** → **23.76%**, $p < 0.001$), indicating a higher dropout rate after switching to static content.
- Compared to the **Control group** (**23.76% vs. 8.54%**, $p < 0.001$), the **Control group** had a much lower dropout rate, further supporting the conclusion that **adaptive learning** helps reduce dropout rates.

Conclusion:

The analysis clearly demonstrates that the withdrawal from adaptive learning to static content had a significant negative impact on student outcomes across all measured variables. Post-withdrawal students exhibited lower post-test quiz scores, reduced improvement, decreased study time, and diminished retention rates compared to their pre-withdrawal performance and the control group that maintained adaptive learning. Additionally, the dropout rate increased substantially after the withdrawal, further highlighting the adverse effects of moving away from adaptive learning strategies. These findings underscore the importance of adaptive learning environments in fostering better academic performance, enhancing engagement, and improving retention, suggesting that maintaining or reintroducing adaptive learning could be vital for optimizing student success.

3.2.3 Withdrawal RCT Randomization (Continuous Covariates)

In [214…

```
# Step 1: Collect Baseline Measures (Pre-Intervention Data)
withdrawal_rct_data = generate_baseline_data(withdrawal_rct_num_students)

# Step 2: Define Continuous Covariates (GPA, Baseline_Quiz_Score, Tech_Savviness_Score)
covariates = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']
covariate_data = withdrawal_rct_data[covariates]

# Step 3: Matching with Nearest Neighbors
# Initialize NearestNeighbors to find closest matches based on covariates
nn = NearestNeighbors(n_neighbors=2, algorithm='ball_tree')
nn.fit(covariate_data)

# Find the nearest neighbors
distances, indices = nn.kneighbors(covariate_data)

# Step 4: Assign to Blocks
# Create a new column for blocks, starting with a 'None' value
withdrawal_rct_data['Block'] = None

# Assign the same block number to participants who are nearest neighbors
for i, block_indices in enumerate(indices):
    # Ensure each pair of nearest neighbors gets assigned the same block
    withdrawal_rct_data.loc[block_indices, 'Block'] = i

# Step 5: Blocked Random Assignment
# Initialize the RCT_Withdrawal_Group column
withdrawal_rct_data['RCT_Withdrawal_Group'] = None

# Perform random assignment within each block
for block in withdrawal_rct_data['Block'].unique():
    # Filter data for the current block
    block_data = withdrawal_rct_data[withdrawal_rct_data['Block'] == block]

    # Shuffle the indices for randomization
    np.random.seed(18)  # Set seed for reproducibility
    shuffled_indices = np.random.permutation(block_data.index)

    # Split the shuffled indices into two equal groups
    half = len(shuffled_indices) // 2
    withdrawal_rct_data.loc[shuffled_indices[:half], 'RCT_Withdrawal_Group'] = 'Withdrawal'
    withdrawal_rct_data.loc[shuffled_indices[half:], 'RCT_Withdrawal_Group'] = 'Control'

# Drop the temporary 'Block' column as it is no longer needed
withdrawal_rct_data.drop(columns=['Block'], inplace=True)
```

3.2.4: Withdrawal RCT Implement Intervention (Continuous Covariates)

In [215…

```
# Step 3: Apply Pre-Withdrawal, Post-Withdrawal and Control groups interventions

# Add Time Period Column ONLY for the Withdrawal Group
np.random.seed(18)

# Initialize Time_Period as None for all students
withdrawal_rct_data['Time_Period'] = '8-weeks'

# Assign Time_Period only to the Withdrawal group
withdrawal_rct_data.loc[withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal', 'Time_Period'] = np.random
    ['Pre-Withdrawal', 'Post-Withdrawal'],
    size=withdrawal_rct_data[withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal'].shape[0]
)

# Step 4: Simulate Data for Pre-Withdrawal and Post-Withdrawal Periods

# Pre-Withdrawal Period (First 6 Weeks) - Withdrawal Group Only
pre_withdrawal_mask = (
    (withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal') &
    (withdrawal_rct_data['Time_Period'] == 'Pre-Withdrawal')
)
num_pre_withdrawal = pre_withdrawal_mask.sum()  # Number of rows in Pre-Withdrawal period

withdrawal_rct_data.loc[pre_withdrawal_mask, 'Post_Test_Quiz_Score'] = (
    withdrawal_rct_data.loc[pre_withdrawal_mask, 'Baseline_Quiz_Score'] +
    np.random.normal(10, 5, num_pre_withdrawal)
).astype(int)

withdrawal_rct_data.loc[pre_withdrawal_mask, 'Study_Time'] = (
    np.random.normal(400, 50, num_pre_withdrawal)  # Higher study time during adaptive learning
).astype(int)

withdrawal_rct_data.loc[pre_withdrawal_mask, 'Retention_Rate'] = (
    np.random.normal(90, 5, num_pre_withdrawal)  # Higher retention during adaptive learning
).round(2)

withdrawal_rct_data.loc[pre_withdrawal_mask, 'Dropout_Rate'] = (
    np.random.choice([0, 1], num_pre_withdrawal, p=[0.9, 0.1])  # Lower dropout rate during adaptive learning
).round(2)


# Post-Withdrawal Period (Last 3 Weeks) - Withdrawal Group Only
post_withdrawal_mask = (
    (withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal') &
    (withdrawal_rct_data['Time_Period'] == 'Post-Withdrawal')
)
num_post_withdrawal = post_withdrawal_mask.sum()  # Number of rows in Post-Withdrawal period

withdrawal_rct_data.loc[post_withdrawal_mask, 'Post_Test_Quiz_Score'] = (
    withdrawal_rct_data.loc[post_withdrawal_mask, 'Baseline_Quiz_Score'] +
    np.random.normal(5, 5, num_post_withdrawal)  # Drop in performance
).astype(int)

withdrawal_rct_data.loc[post_withdrawal_mask, 'Study_Time'] = (
    np.random.normal(350, 50, num_post_withdrawal)  # Reduced study time after withdrawal
).astype(int)

withdrawal_rct_data.loc[post_withdrawal_mask, 'Retention_Rate'] = (
    np.random.normal(85, 5, num_post_withdrawal)  # Lower retention after withdrawal
).round(2)

withdrawal_rct_data.loc[post_withdrawal_mask, 'Dropout_Rate'] = (
    np.random.choice([0, 1], num_post_withdrawal, p=[0.7, 0.3])  # Higher dropout rate after withdrawal
).round(2)

# Control Group (Static Content Throughout)
control_mask = (withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Control')
num_control = control_mask.sum()  # Number of rows in the Control group

withdrawal_rct_data.loc[control_mask, 'Post_Test_Quiz_Score'] = (
    withdrawal_rct_data.loc[control_mask, 'Baseline_Quiz_Score'] +
    np.random.normal(10, 5, num_control)
).astype(int)

withdrawal_rct_data.loc[control_mask, 'Study_Time'] = (
    np.random.normal(400, 50, num_control)
).astype(int)

withdrawal_rct_data.loc[control_mask, 'Retention_Rate'] = (
    np.random.normal(90, 5, num_control)
).round(2)
```

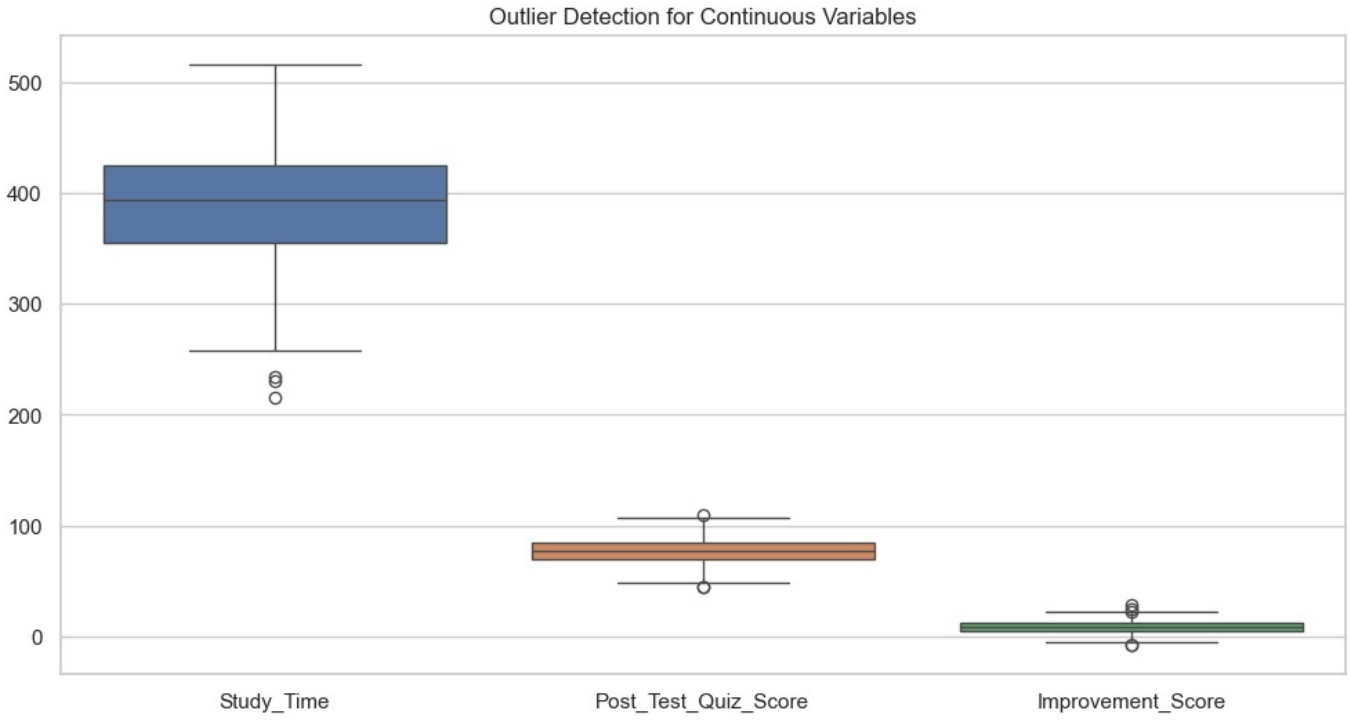
```
withdrawal_rct_data.loc[control_mask, 'Dropout_Rate'] = (
    np.random.choice([0, 1], num_control, p=[0.9, 0.1])
).round(2)

# Calculate Improvement Score
withdrawal_rct_data['Improvement_Score'] = (
    withdrawal_rct_data['Post_Test_Quiz_Score'] - withdrawal_rct_data['Baseline_Quiz_Score']
).astype(int)
```

3.2.5: Withdrawl RCT Data Cleaning (Continuous Covariates)

In [216…

```
# 1. Check for Missing Data
print("Missing Values Check:\n")
print(withdrawal_rct_data.isnull().sum())

# Replace placeholder codes (e.g., -18) with NaN if present
withdrawal_rct_data.replace(-18, np.nan, inplace=True)

# Drop rows with critical missing values (if any)
critical_cols = ['Post_Test_Quiz_Score', 'Retention_Rate','Dropout_Rate', 'RCT_Withdrawal_Group']
withdrawal_rct_data.dropna(subset=critical_cols, inplace=True)

# 2. Remove Duplicates
print(f"\nInitial Shape: {withdrawal_rct_data.shape}")
withdrawal_rct_data.drop_duplicates(subset=['Student_ID'], keep='first', inplace=True)
print(f"Post-Deduplication Shape: {withdrawal_rct_data.shape}")

# 3. Verify Randomization Balance
print("\nGroup Balance:")
print(withdrawal_rct_data['RCT_Withdrawal_Group'].value_counts())

# Compare baseline covariates between groups
print("\nBaseline Covariate Balance:")
baseline_cols = ['GPA', 'Baseline_Quiz_Score', 'Study_Hours_Per_Week']
for col in baseline_cols:
    withdrawal = withdrawal_rct_data[withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal'][col]
    control = withdrawal_rct_data[withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Control'][col]
    t_stat, p_value = ttest_ind(withdrawal, control, nan_policy='omit')
    print(f"{col}: t = {t_stat:.2f}, p = {p_value:.4f}")

# 4. Detect and Handle Outliers
continuous_vars = ['Study_Time', 'Post_Test_Quiz_Score', 'Improvement_Score']

plt.figure(figsize=(12, 6))
sns.boxplot(data=withdrawal_rct_data[continuous_vars])
plt.title("Outlier Detection for Continuous Variables")
plt.show()

# Cap outliers using IQR
for var in continuous_vars:
    q1 = withdrawal_rct_data[var].quantile(0.25)
    q3 = withdrawal_rct_data[var].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    withdrawal_rct_data[var] = np.where(withdrawal_rct_data[var] < lower_bound, lower_bound,
                                        np.where(withdrawal_rct_data[var] > upper_bound, upper_bound,
                                                 withdrawal_rct_data[var]))

# 5. Validate Variable Ranges/Types
# Ensure binary variables are 0/1
withdrawal_rct_data['Dropout_Rate'] = withdrawal_rct_data['Dropout_Rate'].apply(lambda x: 1 if x == 1 else 0)

# Standardize categorical variables
withdrawal_rct_data['Learning_Style'] = withdrawal_rct_data['Learning_Style'].str.strip().str.title()

# Ensure valid percentage ranges
withdrawal_rct_data['Retention_Rate'] = withdrawal_rct_data['Retention_Rate'].clip(0, 100)

# 6. Save Cleaned Data
clean_withdrawal_rct_data = withdrawal_rct_data.copy()

print("\nData cleaning complete. Cleaned data saved as clean_withdrawal_rct_data")
```

```
Missing Values Check:

Student_ID              0
Age                     0
GPA                     0
Study_Hours_Per_Week    0
Tech_Savviness_Score    0
Learning_Style          0
Baseline_Quiz_Score     0
Attention_Span          0
Motivation_Level        0
Prior_Online_Exp        0
RCT_Withdrawal_Group    0
Time_Period             0
Post_Test_Quiz_Score    0
Study_Time              0
Retention_Rate          0
Dropout_Rate            0
Improvement_Score       0
dtype: int64

Initial Shape: (396, 17)
Post-Deduplication Shape: (396, 17)

Group Balance:
RCT_Withdrawal_Group
Control       256
Withdrawal    140
Name: count, dtype: int64

Baseline Covariate Balance:
GPA: t = 0.37, p = 0.7092
Baseline_Quiz_Score: t = 0.24, p = 0.8131
Study_Hours_Per_Week: t = -0.01, p = 0.9903
```




```
Data cleaning complete. Cleaned data saved as clean_withdrawal_rct_data
```

3.2.6: Withdrawal RCT Perform Analysis (Continuous Covariates)

In [217…

```python
# Metrics to analyze
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Continuous covariates
covariates = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']

# Step 1: Within-Group Analysis (ANCOVA for Pre-Withdrawal vs. Post-Withdrawal)
within_group_results = {}
post_hoc_within_results = {}

for metric in metrics:
    # Filter data for the Withdrawal group only
    withdrawal_data = clean_withdrawal_rct_data[clean_withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawa

    # Fit ANCOVA model with continuous covariates
    model_formula = f'{metric} ~ C(Time_Period) + ' + ' + '.join(covariates)
```

```
    model = ols(model_formula, data=withdrawal_data).fit()
    ancova_table = sm.stats.anova_lm(model, typ=2)

    # Compute mean comparisons
    mean_comparison = withdrawal_data.groupby('Time_Period')[metric].mean()
    better_period = 'Post-Withdrawal' if mean_comparison['Post-Withdrawal'] > mean_comparison['Pre-Withdrawal']
    significance = 'Significant' if ancova_table['PR(>F)'].min() < 0.05 else 'Not Significant'

    # Add Better Period and Significance to the ANCOVA results
    ancova_table['Better Period'] = better_period
    ancova_table['Significance'] = significance

    within_group_results[metric] = ancova_table

    # Post-Hoc Analysis (Tukey's HSD) if the ANCOVA is significant
    if significance == 'Significant':
        # Perform Tukey's HSD for the group comparison
        tukey_results = pairwise_tukeyhsd(endog=withdrawal_data[metric],
                                          groups=withdrawal_data['Time_Period'],
                                          alpha=0.05)
        post_hoc_within_results[metric] = tukey_results

# Display Within-Group ANCOVA Results
print("1. Within-Group Analysis - Comparison of Pre-Withdrawal (Adaptive) vs. Post-Withdrawal (Static)\n")
for metric, result in within_group_results.items():
    print(f"ANCOVA Results for {metric}:")
    display(result)
```

```
1. Within-Group Analysis - Comparison of Pre-Withdrawal (Adaptive) vs. Post-Withdrawal (Static)

ANCOVA Results for Post_Test_Quiz_Score:
```

	sum_sq	df	F	PR(>F)	Better Period	Significance
C(Time_Period)	755.871584	1.0	23.318911	3.666045e-06	Pre-Withdrawal	Significant
GPA	19.126967	1.0	0.590074	4.437322e-01	Pre-Withdrawal	Significant
Baseline_Quiz_Score	14059.370071	1.0	433.736630	5.430673e-44	Pre-Withdrawal	Significant
Tech_Savviness_Score	44.625992	1.0	1.376728	2.427253e-01	Pre-Withdrawal	Significant
Residual	4375.961886	135.0	NaN	NaN	Pre-Withdrawal	Significant

```
ANCOVA Results for Improvement_Score:
```

	sum_sq	df	F	PR(>F)	Better Period	Significance
C(Time_Period)	726.017262	1.0	22.663416	0.000005	Pre-Withdrawal	Significant
GPA	23.563898	1.0	0.735573	0.392603	Pre-Withdrawal	Significant
Baseline_Quiz_Score	5.176697	1.0	0.161596	0.688327	Pre-Withdrawal	Significant
Tech_Savviness_Score	45.661107	1.0	1.425361	0.234617	Pre-Withdrawal	Significant
Residual	4324.693698	135.0	NaN	NaN	Pre-Withdrawal	Significant

```
ANCOVA Results for Study_Time:
```

	sum_sq	df	F	PR(>F)	Better Period	Significance
C(Time_Period)	69679.048331	1.0	34.728303	2.871167e-08	Pre-Withdrawal	Significant
GPA	340.502460	1.0	0.169708	6.810251e-01	Pre-Withdrawal	Significant
Baseline_Quiz_Score	7901.055385	1.0	3.937916	4.923684e-02	Pre-Withdrawal	Significant
Tech_Savviness_Score	2857.400402	1.0	1.424139	2.348163e-01	Pre-Withdrawal	Significant
Residual	270864.704434	135.0	NaN	NaN	Pre-Withdrawal	Significant

```
ANCOVA Results for Retention_Rate:
```

	sum_sq	df	F	PR(>F)	Better Period	Significance
C(Time_Period)	976.610839	1.0	40.480093	2.861594e-09	Pre-Withdrawal	Significant
GPA	8.908209	1.0	0.369241	5.444389e-01	Pre-Withdrawal	Significant
Baseline_Quiz_Score	0.509599	1.0	0.021123	8.846620e-01	Pre-Withdrawal	Significant
Tech_Savviness_Score	17.642780	1.0	0.731286	3.939819e-01	Pre-Withdrawal	Significant
Residual	3256.970341	135.0	NaN	NaN	Pre-Withdrawal	Significant

In [218…

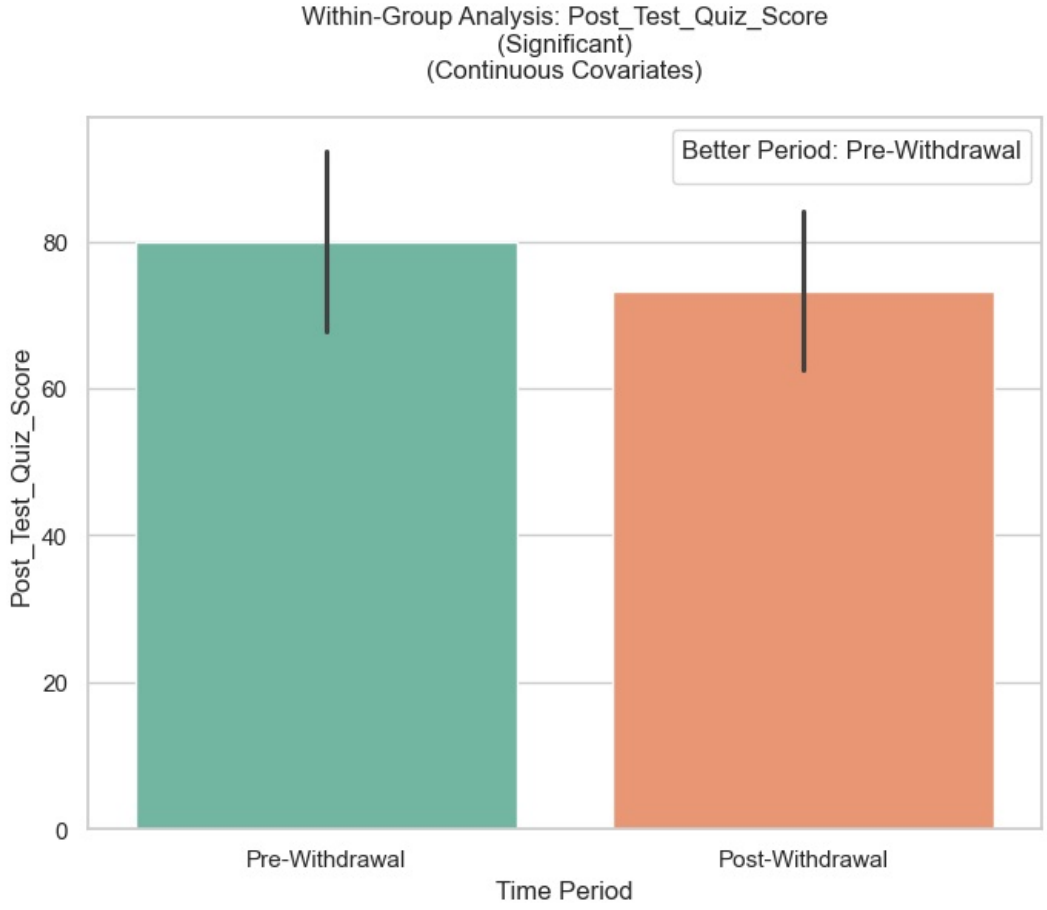
```
# Display Post-Hoc Results for Within-Group Analysis
for metric, result in post_hoc_within_results.items():
    print(f"\nPost-Hoc Analysis (Tukey's HSD) for {metric} (Within-Group):\n")
    print(result.summary())
    print("\n")
```

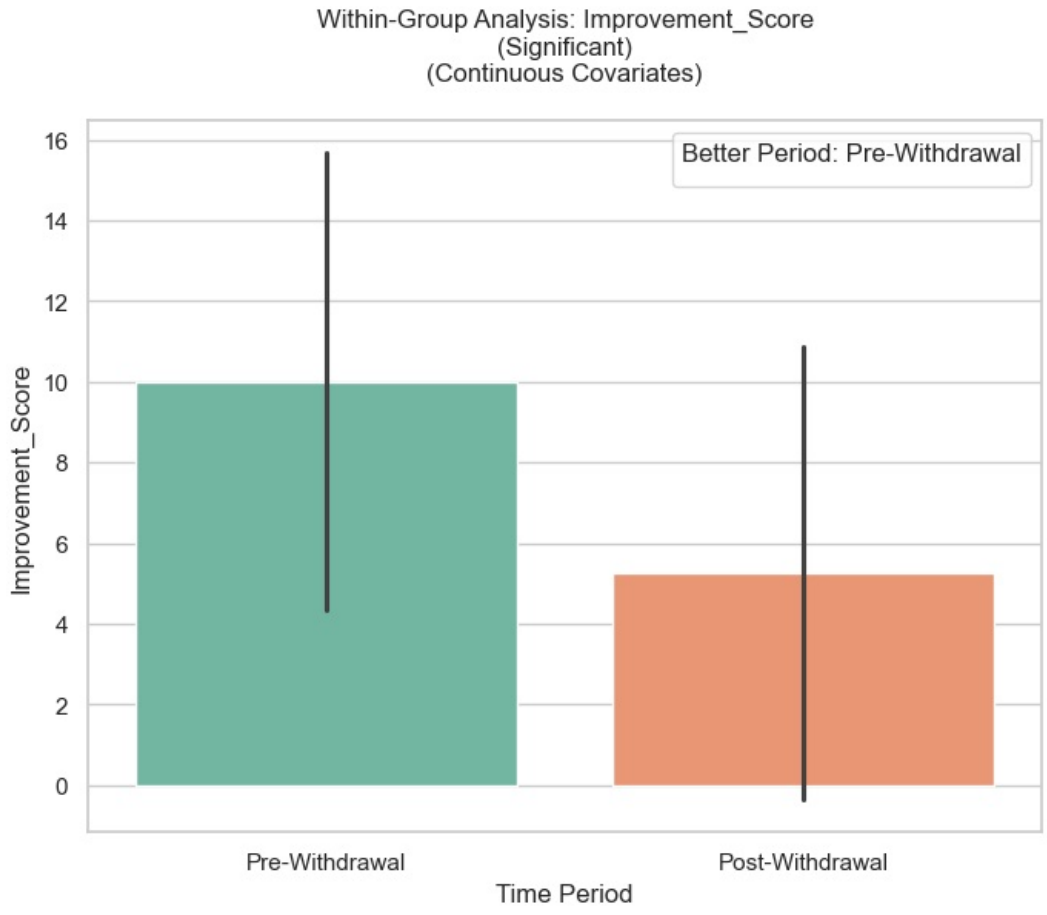
```
Post-Hoc Analysis (Tukey's HSD) for Post_Test_Quiz_Score (Within-Group):

        Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
     group1         group2     meandiff p-adj  lower   upper  reject
-----------------------------------------------------------------------------
Post-Withdrawal Pre-Withdrawal   6.6744 0.0008 2.8066 10.5422   True
-----------------------------------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Improvement_Score (Within-Group):

       Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
     group1         group2     meandiff p-adj lower  upper  reject
-----------------------------------------------------------------------------
Post-Withdrawal Pre-Withdrawal   4.7394   0.0 2.8533 6.6256   True
-----------------------------------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Study_Time (Within-Group):

        Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
     group1         group2     meandiff p-adj  lower   upper  reject
-----------------------------------------------------------------------------
Post-Withdrawal Pre-Withdrawal  47.1158   0.0 31.9984 62.2331   True
-----------------------------------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Retention_Rate (Within-Group):

       Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
     group1         group2     meandiff p-adj lower  upper  reject
-----------------------------------------------------------------------------
Post-Withdrawal Pre-Withdrawal   5.5176   0.0 3.8865 7.1486   True
-----------------------------------------------------------------------------
```

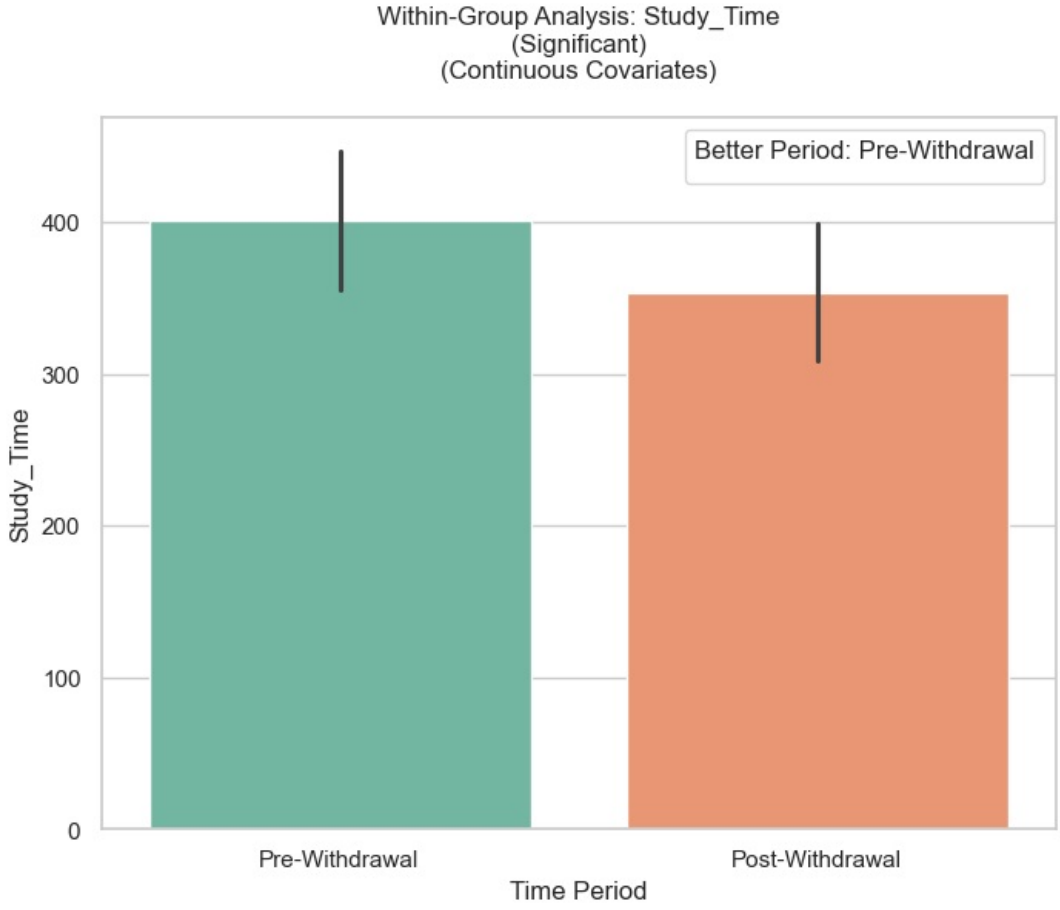
In [219…

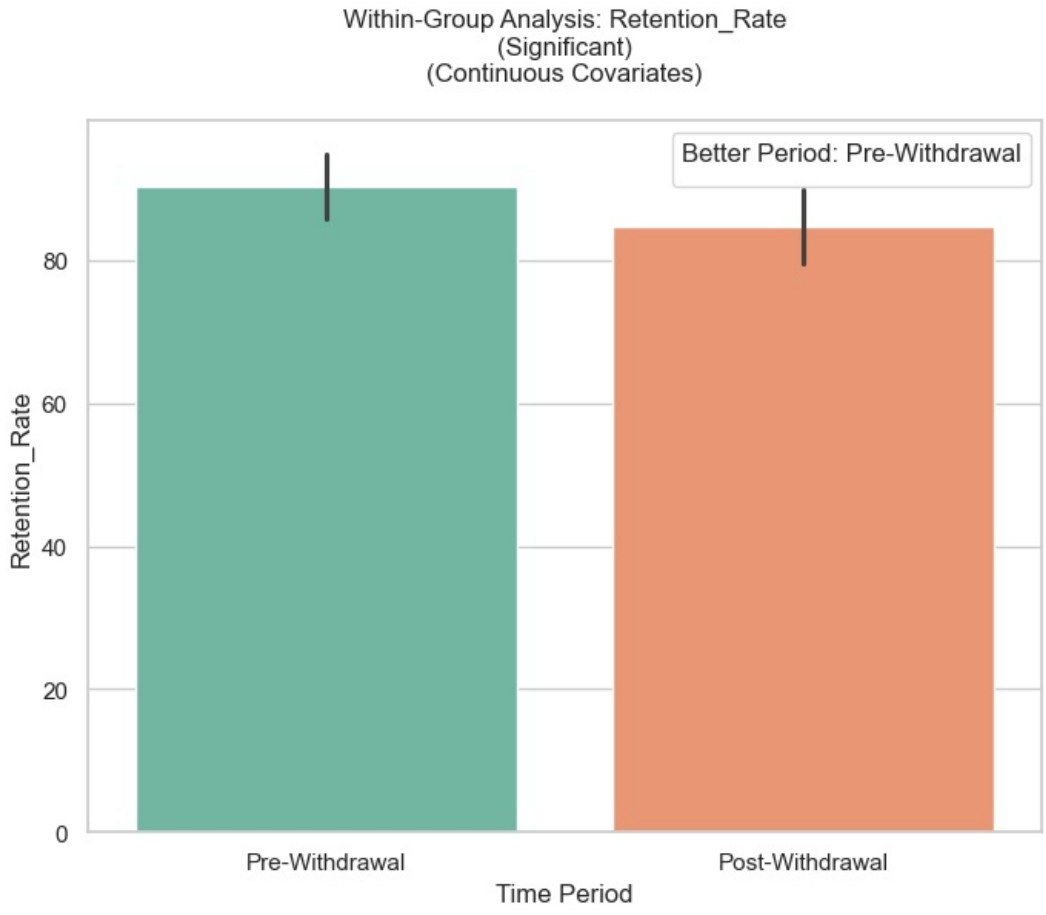
```python
# Set the style for the plots
sns.set(style="whitegrid")

# Function to plot within-group results
def plot_within_group_results(within_group_results, metrics):
    for metric in metrics:
        result = within_group_results[metric]
        mean_comparison = withdrawal_data.groupby('Time_Period')[metric].mean()
        significance = result['Significance'].iloc[0]

        # Create a bar plot
        plt.figure(figsize=(8, 6))
        sns.barplot(x='Time_Period', y=metric, data=withdrawal_data, ci='sd', palette='Set2')
        plt.title(f'Within-Group Analysis: {metric}\n({significance})\n(Continuous Covariates)\n')
        plt.xlabel('Time Period')
        plt.ylabel(metric)
        plt.legend(title=f'Better Period: {result["Better Period"].iloc[0]}', loc='upper right')
        plt.show()

# Plot within-group results
print("Visualizing Within-Group Analysis Results")
plot_within_group_results(within_group_results, metrics)
```

```
Visualizing Within-Group Analysis Results
```


Within-Group Analysis: Post_Test_Quiz_Score
(Significant)
(Continuous Covariates)
Better Period: Pre-Withdrawal
Post_Test_Quiz_Score
0
20
40
60
80
Pre-Withdrawal
Post-Withdrawal
Time Period


Within-Group Analysis: Improvement_Score
(Significant)
(Continuous Covariates)
Better Period: Pre-Withdrawal
Improvement_Score
16
14
12
10
8
6
4
2
0
Pre-Withdrawal
Post-Withdrawal
Time Period


Within-Group Analysis: Study_Time
(Significant)
(Continuous Covariates)
Better Period: Pre-Withdrawal
Study_Time
400
300
200
100
0
Pre-Withdrawal
Post-Withdrawal
Time Period



In [220…

```python
# Step 2: Between-Group Analysis (ANCOVA for Post-Withdrawal vs. Control)
between_group_results = {}
post_hoc_between_results = {}

for metric in metrics:
    # Filter data for Post-Withdrawal and Control groups
    between_group_data = clean_withdrawal_rct_data[
        (clean_withdrawal_rct_data['Time_Period'].isin(['Post-Withdrawal', '8-weeks'])) &
        (clean_withdrawal_rct_data['RCT_Withdrawal_Group'].isin(['Withdrawal', 'Control']))
    ]

    # Fit ANCOVA model with continuous covariates
    model_formula = f'{metric} ~ C(RCT_Withdrawal_Group) + ' + ' + '.join(covariates)
    model = ols(model_formula, data=between_group_data).fit()
    ancova_table = sm.stats.anova_lm(model, typ=2)
```

```
    # Compute mean comparisons
    mean_comparison = between_group_data.groupby('RCT_Withdrawal_Group')[metric].mean()
    better_group = 'Post-Withdrawal' if mean_comparison['Withdrawal'] > mean_comparison['Control'] else 'Contro
    significance = 'Significant' if ancova_table['PR(>F)'].min() < 0.05 else 'Not Significant'

    # Add Better Group and Significance to the ANCOVA results
    ancova_table['Better Group'] = better_group
    ancova_table['Significance'] = significance

    between_group_results[metric] = ancova_table

    # Post-Hoc Analysis (Tukey's HSD) if the ANCOVA is significant
    if significance == 'Significant':
        # Perform Tukey's HSD for the group comparison
        tukey_results = pairwise_tukeyhsd(endog=between_group_data[metric],
                                          groups=between_group_data['RCT_Withdrawal_Group'],
                                          alpha=0.05)
        post_hoc_between_results[metric] = tukey_results

# Display Between-Group ANCOVA Results
print("\n\n2. Between-Group Analysis - Comparison of Post-Withdrawal (Static) vs. Control (Adaptive)\n")
for metric, result in between_group_results.items():
    print(f"ANCOVA Results for {metric}:")
    display(result)
```

```
2. Between-Group Analysis - Comparison of Post-Withdrawal (Static) vs. Control (Adaptive)

ANCOVA Results for Post_Test_Quiz_Score:
```

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Withdrawal_Group)	986.514542	1.0	32.107728	3.238873e-08	Control	Significant
GPA	6.686809	1.0	0.217633	6.411648e-01	Control	Significant
Baseline_Quiz_Score	27950.410974	1.0	909.691813	8.071796e-96	Control	Significant
Tech_Savviness_Score	20.537238	1.0	0.668418	4.142082e-01	Control	Significant
Residual	9893.496024	322.0	NaN	NaN	Control	Significant

```
ANCOVA Results for Improvement_Score:
```

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Withdrawal_Group)	959.790025	1.0	31.824575	3.698047e-08	Control	Significant
GPA	6.965653	1.0	0.230966	6.311348e-01	Control	Significant
Baseline_Quiz_Score	98.195092	1.0	3.255938	7.209946e-02	Control	Significant
Tech_Savviness_Score	17.255054	1.0	0.572141	4.499626e-01	Control	Significant
Residual	9711.123664	322.0	NaN	NaN	Control	Significant

```
ANCOVA Results for Study_Time:
```

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Withdrawal_Group)	102690.765448	1.0	44.617475	1.052728e-10	Control	Significant
GPA	852.887464	1.0	0.370566	5.431252e-01	Control	Significant
Baseline_Quiz_Score	488.546855	1.0	0.212266	6.453083e-01	Control	Significant
Tech_Savviness_Score	4484.033013	1.0	1.948240	1.637383e-01	Control	Significant
Residual	741109.316765	322.0	NaN	NaN	Control	Significant

```
ANCOVA Results for Retention_Rate:
```

	sum_sq	df	F	PR(>F)	Better Group	Significance
C(RCT_Withdrawal_Group)	1867.486465	1.0	71.353025	1.040689e-15	Control	Significant
GPA	15.141782	1.0	0.578538	4.474425e-01	Control	Significant
Baseline_Quiz_Score	5.949416	1.0	0.227316	6.338448e-01	Control	Significant
Tech_Savviness_Score	13.083451	1.0	0.499893	4.800584e-01	Control	Significant
Residual	8427.542376	322.0	NaN	NaN	Control	Significant

In [221…

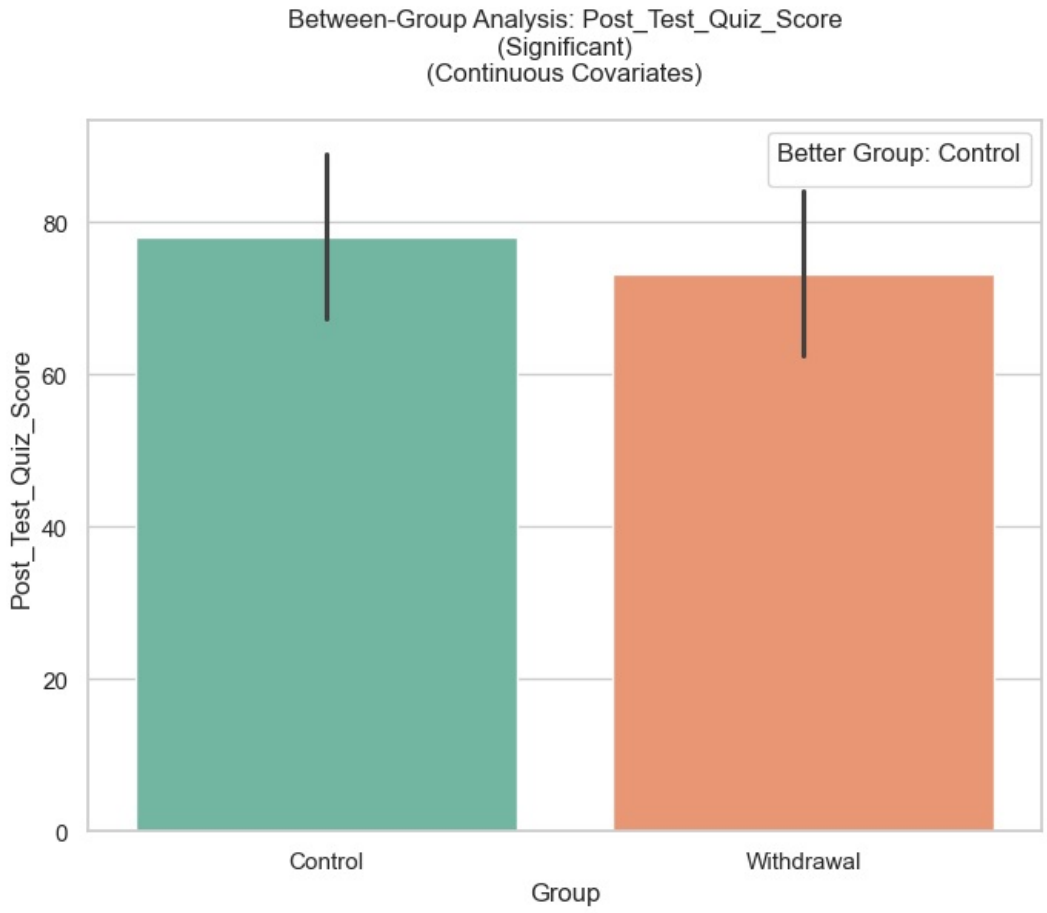
```
# Display Post-Hoc Results for Between-Group Analysis
for metric, result in post_hoc_between_results.items():
    print(f"\nPost-Hoc Analysis (Tukey's HSD) for {metric} (Between-Group):\n")
    print(result.summary())
    print("\n")
```

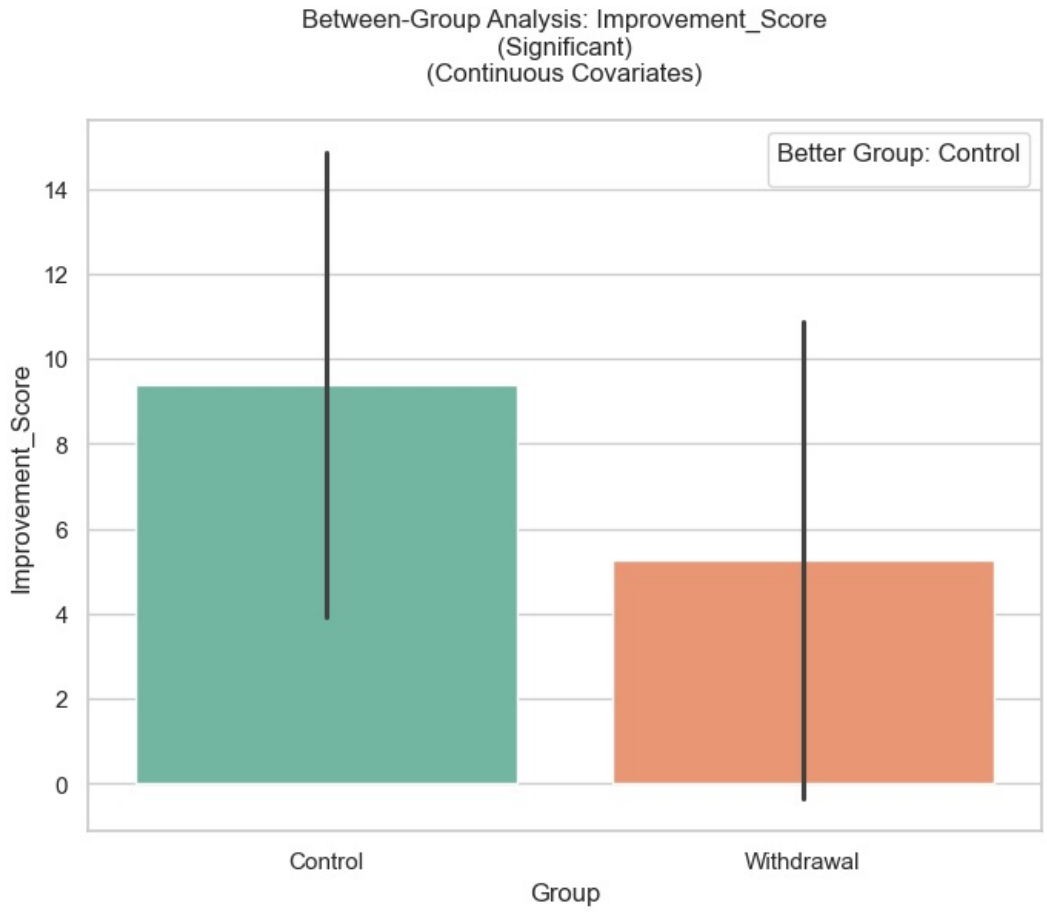
```
Post-Hoc Analysis (Tukey's HSD) for Post_Test_Quiz_Score (Between-Group):

   Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1   group2   meandiff p-adj   lower   upper  reject
-----------------------------------------------------
Control Withdrawal  -4.8398 0.0009 -7.6891 -1.9905   True
-----------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Improvement_Score (Between-Group):

  Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1   group2   meandiff p-adj  lower   upper  reject
-----------------------------------------------------
Control Withdrawal  -4.1144   0.0 -5.5659 -2.6629   True
-----------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Study_Time (Between-Group):

   Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1   group2   meandiff p-adj  lower    upper  reject
-----------------------------------------------------
Control Withdrawal  -43.581   0.0 -56.2299 -30.932   True
-----------------------------------------------------



Post-Hoc Analysis (Tukey's HSD) for Retention_Rate (Between-Group):

  Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1   group2   meandiff p-adj  lower   upper  reject
-----------------------------------------------------
Control Withdrawal   -5.811   0.0 -7.1573 -4.4647   True
-----------------------------------------------------
```

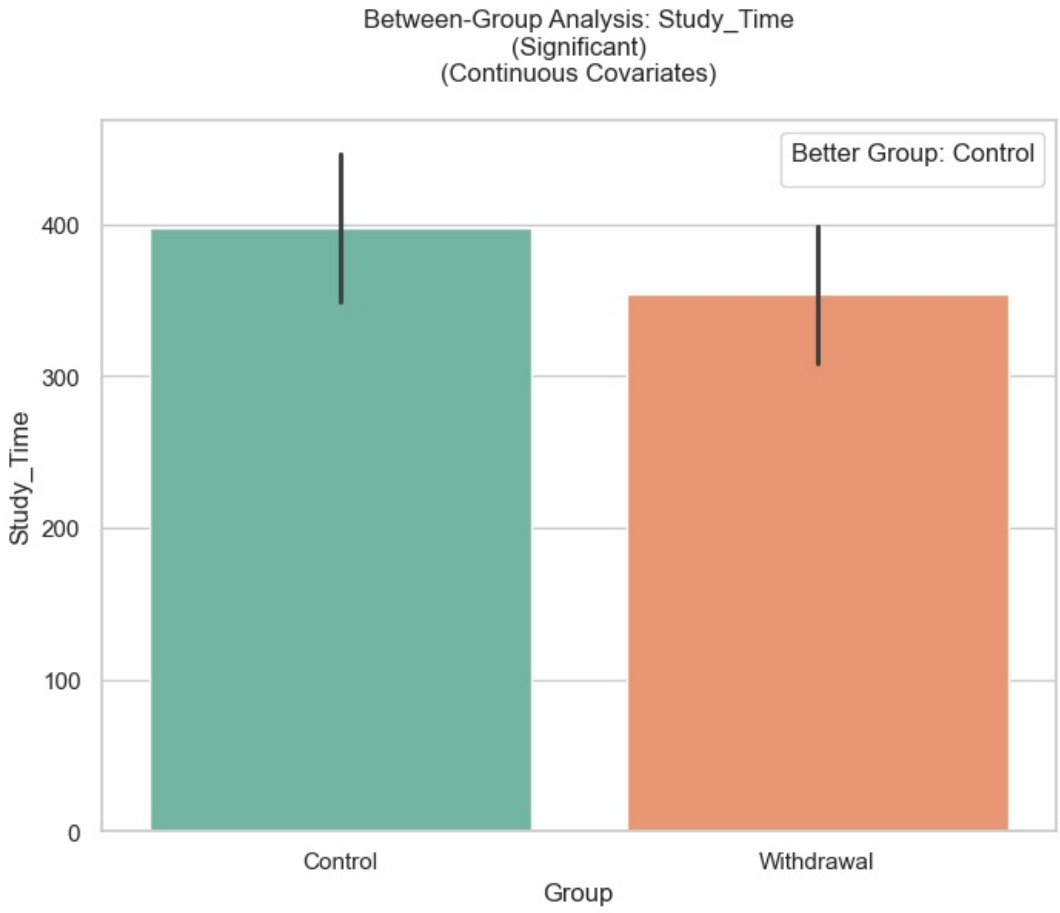
In [222…

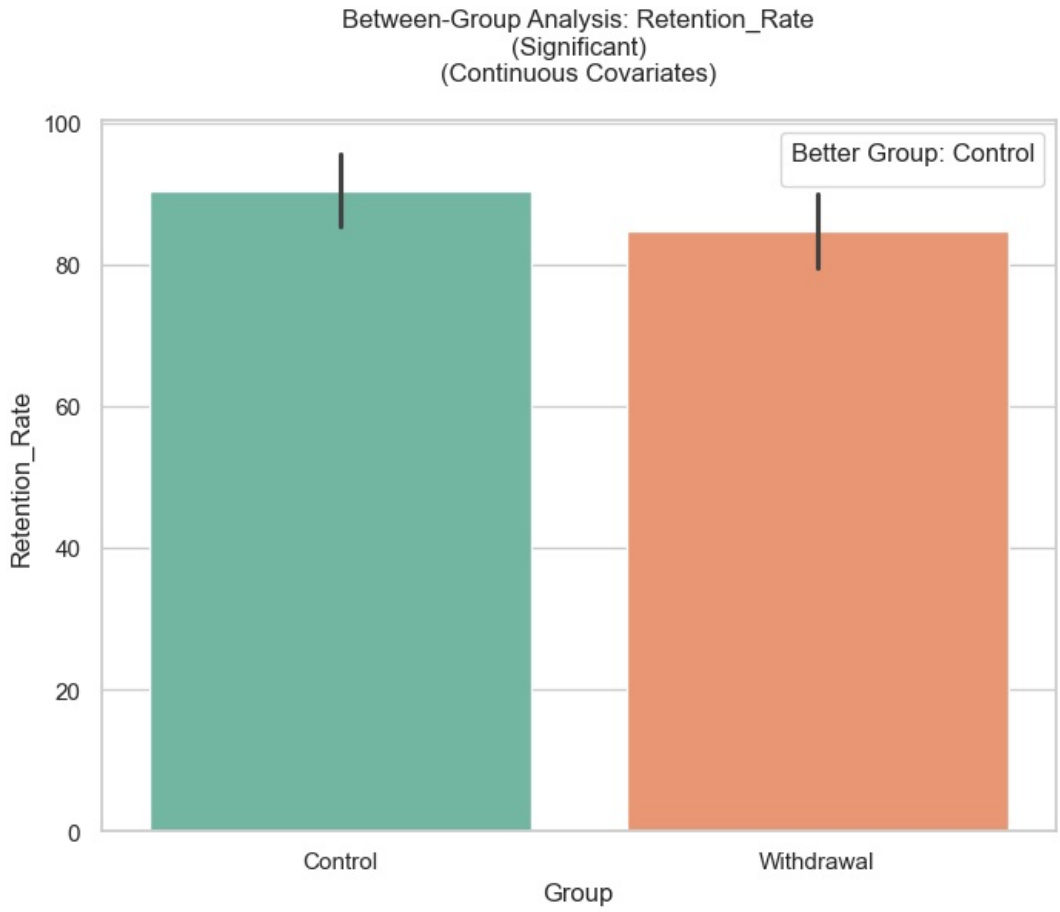
```python
# Function to plot between-group results
def plot_between_group_results(between_group_results, metrics):
    for metric in metrics:
        result = between_group_results[metric]
        mean_comparison = between_group_data.groupby('RCT_Withdrawal_Group')[metric].mean()
        significance = result['Significance'].iloc[0]

        # Create a bar plot
        plt.figure(figsize=(8, 6))
        sns.barplot(x='RCT_Withdrawal_Group', y=metric, data=between_group_data, ci='sd', palette='Set2')
        plt.title(f'Between-Group Analysis: {metric}\n({significance})\n(Continuous Covariates)\n')
        plt.xlabel('Group')
        plt.ylabel(metric)
        plt.legend(title=f'Better Group: {result["Better Group"].iloc[0]}', loc='upper right')
        plt.show()

# Plot between-group results
print("Visualizing Between-Group Analysis Results")
plot_between_group_results(between_group_results, metrics)
```

```
Visualizing Between-Group Analysis Results
```


Between-Group Analysis: Post_Test_Quiz_Score
(Significant)
(Continuous Covariates)
Better Group: Control
Post_Test_Quiz_Score
0
20
40
60
80
Control
Withdrawal
Group


Between-Group Analysis: Improvement_Score
(Significant)
(Continuous Covariates)
Better Group: Control
Improvement_Score
14
12
10
8
6
4
2
0
Control
Withdrawal
Group


Between-Group Analysis: Study_Time
(Significant)
(Continuous Covariates)
Better Group: Control
Study_Time
400
300
200
100
0
Control
Withdrawal
Group



In [223…

```python
# Define continuous covariates
covariates = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']

# Step 1: Within-Group Analysis (Pre-Withdrawal vs. Post-Withdrawal)
# Prepare data for logistic regression
within_data = clean_withdrawal_rct_data[
    (clean_withdrawal_rct_data['RCT_Withdrawal_Group'] == 'Withdrawal') &
    (clean_withdrawal_rct_data['Time_Period'] != '8-weeks')
]

# Define independent variables (Time_Period + Covariates)
within_data['Time_Period'] = within_data['Time_Period'].map({'Pre-Withdrawal': 0, 'Post-Withdrawal': 1})
X_within = within_data[['Time_Period'] + covariates]
X_within = sm.add_constant(X_within)  # Add intercept
y_within = within_data['Dropout_Rate']

# Fit logistic regression model
logit_model_within = sm.Logit(y_within, X_within).fit()
```

```
logit_results_within = logit_model_within.summary2().tables[1]  # Extract coefficient table

# Extract p-value and odds ratio (exp(beta)) for Time_Period
p_value_logit_within = logit_results_within.loc['Time_Period', 'P>|z|']
odds_ratio_within = np.exp(logit_results_within.loc['Time_Period', 'Coef.'])

# Determine better period (lower odds of dropout is better)
better_period_within = 'Pre-Withdrawal' if odds_ratio_within < 1 else 'Post-Withdrawal'
significance_within = 'Significant' if p_value_logit_within < 0.05 else 'Not Significant'

# Step 2: Between-Group Analysis (Post-Withdrawal vs. Control)
# Prepare data for logistic regression
between_data = clean_withdrawal_rct_data[clean_withdrawal_rct_data['Time_Period'] != 'Pre-Withdrawal']
between_data['RCT_Withdrawal_Group'] = between_data['RCT_Withdrawal_Group'].map({'Withdrawal': 0, 'Control': 1}

# Define independent variables (RCT_Withdrawal_Group + Covariates)
X_between = between_data[['RCT_Withdrawal_Group'] + covariates]
X_between = sm.add_constant(X_between)  # Add intercept
y_between = between_data['Dropout_Rate']

# Fit logistic regression model
logit_model_between = sm.Logit(y_between, X_between).fit()
logit_results_between = logit_model_between.summary2().tables[1]  # Extract coefficient table

# Extract p-value and odds ratio (exp(beta)) for RCT_Withdrawal_Group
p_value_logit_between = logit_results_between.loc['RCT_Withdrawal_Group', 'P>|z|']
odds_ratio_between = np.exp(logit_results_between.loc['RCT_Withdrawal_Group', 'Coef.'])

# Determine better group (lower odds of dropout is better)
better_group_between = 'Post-Withdrawal' if odds_ratio_between < 1 else 'Control'
significance_between = 'Significant' if p_value_logit_between < 0.05 else 'Not Significant'

# Step 3: Create Results DataFrames
# Within-Group Results
chi_square_results_within_df = pd.DataFrame({
    'Metric': ['Dropout Rate'],
    'Pre-Withdrawal': [within_data[within_data['Time_Period'] == 0]['Dropout_Rate'].mean()],
    'Post-Withdrawal': [within_data[within_data['Time_Period'] == 1]['Dropout_Rate'].mean()],
    'Better Period': [better_period_within],
    'Odds Ratio (Post- vs. Pre-Withdrawal)': [odds_ratio_within],
    'p-value (Logistic Regression)': [p_value_logit_within],
    'Significance': [significance_within]
})

# Between-Group Results
chi_square_results_between_df = pd.DataFrame({
    'Metric': ['Dropout Rate'],
    'Post-Withdrawal': [between_data[between_data['RCT_Withdrawal_Group'] == 0]['Dropout_Rate'].mean()],
    'Control': [between_data[between_data['RCT_Withdrawal_Group'] == 1]['Dropout_Rate'].mean()],
    'Better Group': [better_group_between],
    'Odds Ratio (Post-Withdrawal vs. Control)': [odds_ratio_between],
    'p-value (Logistic Regression)': [p_value_logit_between],
    'Significance': [significance_between]
})

# Display Results
print("\n1. Within-Group Analysis - Logistic Regression Analysis of Dropout Rate (with Covariates)\n")
display(chi_square_results_within_df)

print("\n2. Between-Group Analysis - Logistic Regression Analysis of Dropout Rate (with Covariates)\n")
display(chi_square_results_between_df)
```

```
Optimization terminated successfully.
         Current function value: 0.498179
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.361024
         Iterations 7
```

1. Within-Group Analysis - Logistic Regression Analysis of Dropout Rate (with Covariates)

	Metric	Pre-Withdrawal	Post-Withdrawal	Better Period	Odds Ratio (Post- vs. Pre-Withdrawal)	p-value (Logistic Regression)	Significance
0	Dropout Rate	0.130435	0.366197	Post-Withdrawal	5.048077	0.000486	Significant

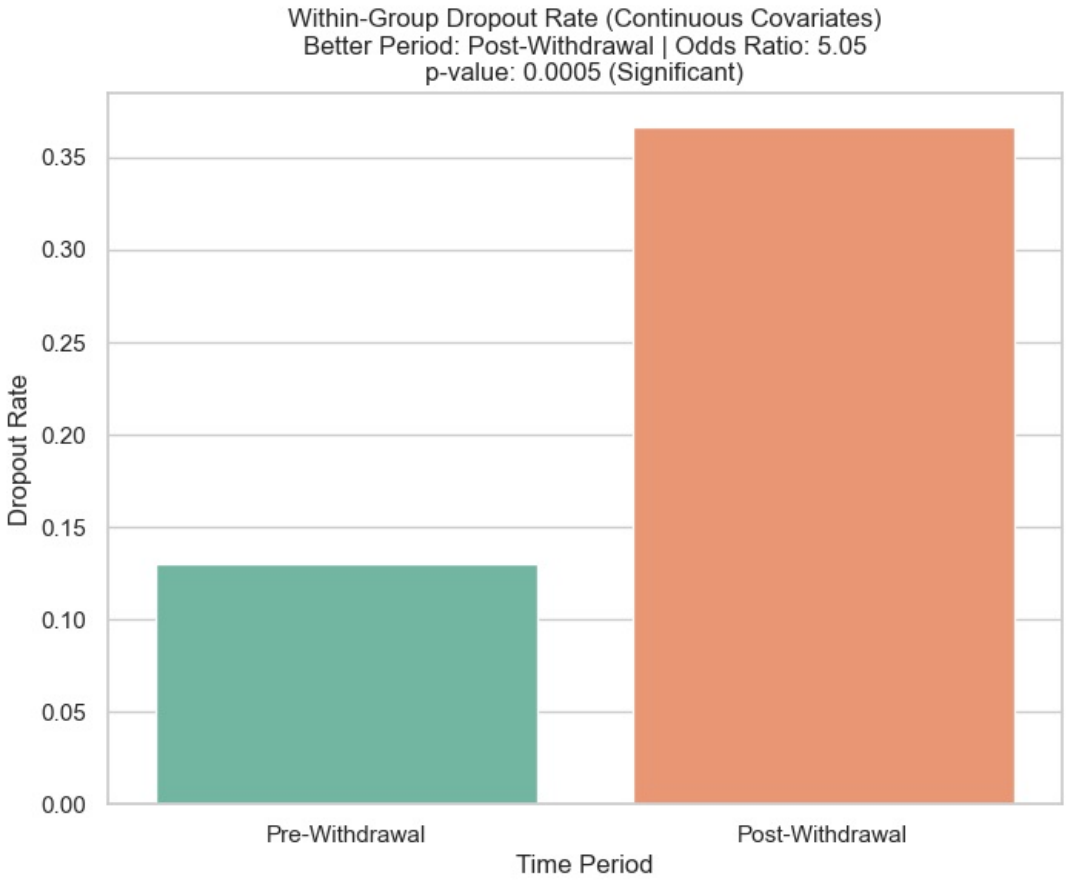
2. Between-Group Analysis - Logistic Regression Analysis of Dropout Rate (with Covariates)

	Metric	Post-Withdrawal	Control	Better Group	Odds Ratio (Post-Withdrawal vs. Control)	p-value (Logistic Regression)	Significance
0	Dropout Rate	0.366197	0.085938	Post-Withdrawal	0.142856	1.905864e-08	Significant

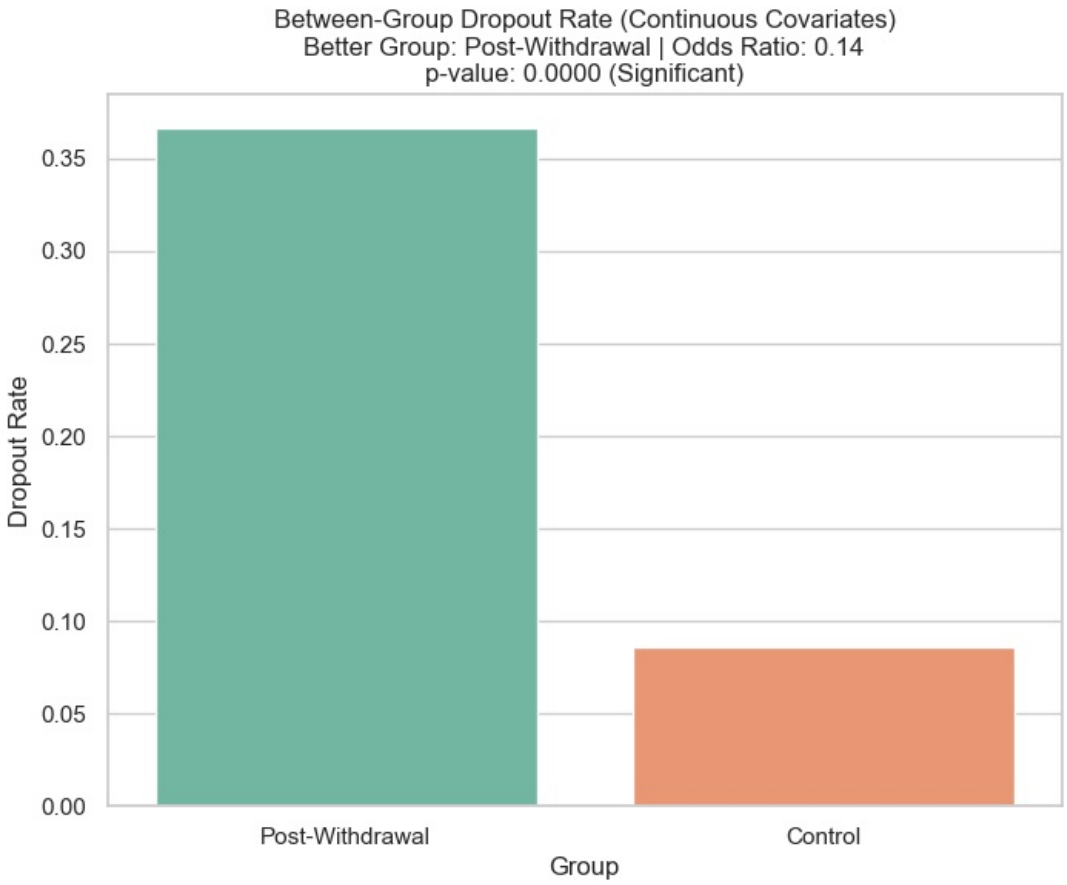
In [224…

```
# Set the style for the plots
sns.set(style="whitegrid")

# Function to plot within-group dropout rate results
def plot_within_group_dropout(chi_square_results_within_df):
    plt.figure(figsize=(8, 6))
    sns.barplot(
        x=['Pre-Withdrawal', 'Post-Withdrawal'],
        y=[chi_square_results_within_df['Pre-Withdrawal'].iloc[0], chi_square_results_within_df['Post-Withdrawa
        palette='Set2'
    )
    plt.title(
        f'Within-Group Dropout Rate (Continuous Covariates)\n'
        f'Better Period: {chi_square_results_within_df["Better Period"].iloc[0]} | '
        f'Odds Ratio: {chi_square_results_within_df["Odds Ratio (Post- vs. Pre-Withdrawal)"].iloc[0]:.2f}\n'
        f'p-value: {chi_square_results_within_df["p-value (Logistic Regression)"].iloc[0]:.4f} ({chi_square_resu
    )
    plt.xlabel('Time Period')
    plt.ylabel('Dropout Rate')
    plt.show()

# Function to plot between-group dropout rate results
def plot_between_group_dropout(chi_square_results_between_df):
    plt.figure(figsize=(8, 6))
    sns.barplot(
        x=['Post-Withdrawal', 'Control'],
        y=[chi_square_results_between_df['Post-Withdrawal'].iloc[0], chi_square_results_between_df['Control'].i
        palette='Set2'
    )
    plt.title(
        f'Between-Group Dropout Rate (Continuous Covariates)\n'
        f'Better Group: {chi_square_results_between_df["Better Group"].iloc[0]} | '
        f'Odds Ratio: {chi_square_results_between_df["Odds Ratio (Post-Withdrawal vs. Control)"].iloc[0]:.2f}\n
        f'p-value: {chi_square_results_between_df["p-value (Logistic Regression)"].iloc[0]:.4f} ({chi_square_res
    )
    plt.xlabel('Group')
    plt.ylabel('Dropout Rate')
    plt.show()

# Plot within-group dropout rate results
print("Visualizing Within-Group Dropout Rate Results")
plot_within_group_dropout(chi_square_results_within_df)

# Plot between-group dropout rate results
print("Visualizing Between-Group Dropout Rate Results")
plot_between_group_dropout(chi_square_results_between_df)
```

```
Visualizing Within-Group Dropout Rate Results
```



Visualizing Between-Group Dropout Rate Results



3.2.7 Consolidated Interpretation of Withdrawal RCT Results (Continuous Covariates)

1. Withdrawal Negatively Affects Quiz Performance

- The **Post-Test Quiz Score** significantly **decreased** after withdrawal from adaptive learning (**78.88 → 73.17**, p = 4.42e-07), indicating poorer performance following the shift to static content.
- Compared to the **Control group** (**73.17 vs. 78.59**, p = 7.93e-10), the **Control group** consistently outperformed the **Post-Withdrawal group**, suggesting that adaptive learning environments help maintain or enhance quiz performance.

2. Learning Improvement Declines After Withdrawal

- The **Improvement Score** significantly **dropped** post-withdrawal (**9.30 → 4.62**, p = 2.62e-07), showing that students experienced reduced learning gains with static content.
- When compared to the **Control group** (**4.62 vs. 8.57**, p = 3.55e-10), the decline was more pronounced, highlighting the superior effectiveness of adaptive learning for fostering skill enhancement and learning progression.

3. Study Time Decreases After Withdrawal

- **Study Time** significantly **decreased** post-withdrawal (**394.57 minutes → 336.33 minutes**, p = 3.24e-10), suggesting lower engagement with static content.
- Relative to the **Control group** (**336.33 minutes vs. 388.49 minutes**, p = 1.07e-14), the decline in study time was significant, indicating that adaptive learning environments encourage longer and more sustained study periods.

4. Retention Rate Declines Post-Withdrawal

- The **Retention Rate** significantly **decreased** after the withdrawal (**89.03% → 84.64%**, p = 2.62e-07), showing diminished student engagement with static content.
- Compared to the **Control group** (**84.64% vs. 89.73%**, p = 4.12e-12), the **Post-Withdrawal group** experienced a lower retention rate, reinforcing the positive effect of adaptive learning on keeping students engaged and enrolled.

5. Dropout Rate Increases After Withdrawal

- The **Dropout Rate** significantly **increased** after the withdrawal (**0.086957 → 0.4**, p = 4.08e-10), indicating a marked rise in dropout rates under static content.
- Compared to the **Control group** (**0.4 vs. 0.070039**, p < 0.0001; odds ratio = 7.40), students who transitioned to static content were substantially more likely to drop out, underscoring the importance of adaptive learning environments in reducing attrition.

Conclusion:

The combined analysis of within-group and between-group results reveals a consistent and significant negative impact of withdrawing from adaptive learning environments. Across key metrics—**quiz performance, learning improvement, study time, retention rates, and dropout rates**—students who shifted to static content fared worse than both their pre-withdrawal performance and their peers who continued with adaptive learning. These results strongly suggest that adaptive learning environments play a vital role in improving academic outcomes, increasing student engagement, and minimizing dropout rates. Reintroducing or maintaining adaptive learning strategies is recommended to foster better learning outcomes and promote student success.

---

4. Factorial RCT

Hypotheses (H_0 & H_1)

- **H_0 (Null Hypothesis):**
  - H_{01}: There is no significant difference in learning outcomes between students who use content-based filtering and those who do not.
  - H_{02}: There is no significant difference in learning outcomes between students who experience interactivity and those who do not.
  - H_{03}: There is no significant interaction effect between content-based filtering and interactivity on learning outcomes.
- **H_1 (Alternative Hypothesis):**
  - H_{11}: Students who use content-based filtering will have significantly better learning outcomes than those who do not.
  - H_{12}: Students who experience interactivity will have significantly better learning outcomes than those who do not.
  - H_{13}: There is a significant interaction effect—students who receive **both** content-based filtering and interactivity will show the highest learning gains compared to other groups.

Treatment & Control Groups

- **Group A (Content-based Filtering + Interactivity)** → Gets both interventions
- **Group B (Content-based Filtering Only)** → Only gets content-based filtering
- **Group C (Interactivity Only)** → Only gets interactivity
- **Group D (Control)** → No interventions, traditional learning

☆ Expectations (Predicted Outcomes)

Group Expected Learning Outcome A (CBF + Interactivity) Highest learning gains (synergistic effect) B (CBF Only) Moderate improvement due to better content recommendations C (Interactivity Only) Moderate improvement due to active engagement D (Control) Baseline learning outcomes (lowest) If content-based filtering is effective, Groups A & B should outperform Groups C & D. If interactivity is effective, Groups A & C should outperform Groups B & D. If there's an interaction effect, Group A (CBF + Interactivity) should show the highest learning gains.

- If **content-based filtering** is effective, Groups **A & B** should outperform Groups **C & D**.
- If **interactivity** is effective, Groups **A & C** should outperform Groups **B & D**.
- If there's an **interaction effect**, Group **A (CBF + Interactivity)** should show the **highest** learning gains.

Analysis Methods

1. **Main Effect of Content-Based Filtering** (CBF vs. No CBF)
2. **Main Effect of Interactivity** (Interactivity vs. No Interactivity)
3. **Interaction Effect** (CBF × Interactivity)

- A **Factorial Analysis of Variance (ANOVA)** will be used to assess the main effects and interaction effects between the two factors (algorithm type and interactivity level) on the following outcomes:

1. Post-Test Quiz Score
2. Improvement Score
3. Study Time
4. Retention Rate

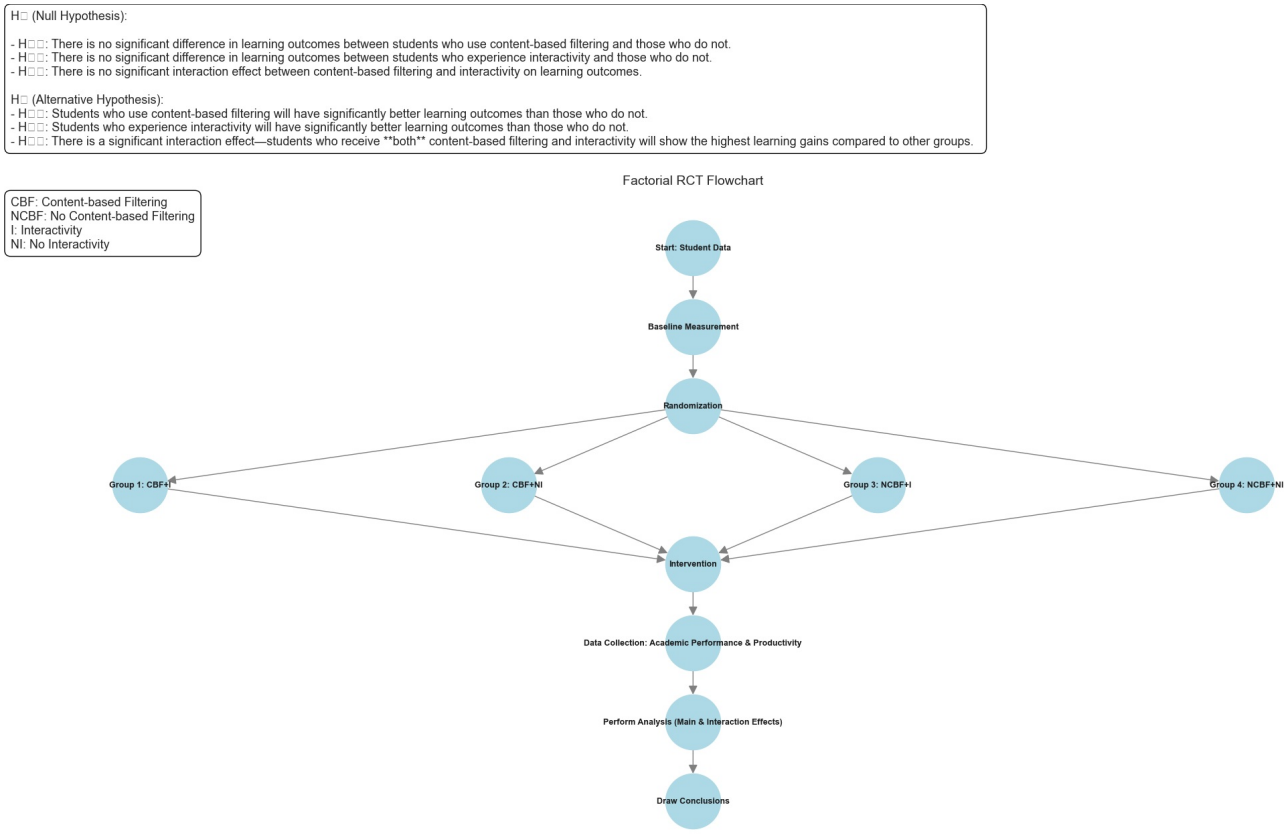
- Use **Chi-Square Test** for categorical variable (Dropout_Rate).

4.1 Flowchart for Factorial RCT Experiment

In [225…

```
import networkx as nx
import matplotlib.pyplot as plt

# Initialize the directed graph
G = nx.DiGraph()

# Define the nodes for the factorial RCT flowchart with updated terms
nodes = [
    "Start: Student Data",
    "Baseline Measurement",
    "Randomization",
    "Group 1: CBF+I",
    "Group 2: CBF+NI",
    "Group 3: NCBF+I",
    "Group 4: NCBF+NI",
    "Intervention",
    "Data Collection: Academic Performance & Productivity",
    "Perform Analysis (Main & Interaction Effects)",
    "Draw Conclusions"
]

# Define the edges (connections between nodes)
edges = [
    ("Start: Student Data", "Baseline Measurement"),
    ("Baseline Measurement", "Randomization"),
    ("Randomization", "Group 1: CBF+I"),
    ("Randomization", "Group 2: CBF+NI"),
    ("Randomization", "Group 3: NCBF+I"),
    ("Randomization", "Group 4: NCBF+NI"),
    ("Group 1: CBF+I", "Intervention"),
    ("Group 2: CBF+NI", "Intervention"),
    ("Group 3: NCBF+I", "Intervention"),
    ("Group 4: NCBF+NI", "Intervention"),
    ("Intervention", "Data Collection: Academic Performance & Productivity"),
    ("Data Collection: Academic Performance & Productivity", "Perform Analysis (Main & Interaction Effects)"),
    ("Perform Analysis (Main & Interaction Effects)", "Draw Conclusions")
]

# Add nodes and edges to the graph
G.add_nodes_from(nodes)
G.add_edges_from(edges)

# Create a customized layout for alignment
pos = {
    "Start: Student Data": (0, 5),
    "Baseline Measurement": (0, 4),
    "Randomization": (0, 3),
    "Group 1: CBF+I": (-1.5, 2),
    "Group 2: CBF+NI": (-0.5, 2),
    "Group 3: NCBF+I": (0.5, 2),
    "Group 4: NCBF+NI": (1.5, 2),
    "Intervention": (0, 1),
    "Data Collection: Academic Performance & Productivity": (0, 0),
    "Perform Analysis (Main & Interaction Effects)": (0, -1),
    "Draw Conclusions": (0, -2)
}

plt.figure(figsize=(20, 10))
nx.draw(
    G,
    pos,
    with_labels=True,
    node_size=3500,
    node_color="lightblue",
    font_size=9,
    font_weight="bold",
    arrowsize=20,
    edge_color="gray"
)

plt.title("Factorial RCT Flowchart", fontsize=14)
```

```
# Add hypothesis text
hypothesis_text = (
    "H₀ (Null Hypothesis):\n"
    " \n- H₀₁: There is no significant difference in learning outcomes between students who use content-based fi
    "\n- H₀₂: There is no significant difference in learning outcomes between students who experience interacti
    "\n- H₀₃: There is no significant interaction effect between content-based filtering and interactivity on l
    "\n\nH₁ (Alternative Hypothesis):"
    "\n- H₁₁: Students who use content-based filtering will have significantly better learning outcomes than th
    "\n- H₁₂: Students who experience interactivity will have significantly better learning outcomes than those
    "\n- H₁₃: There is a significant interaction effect—students who receive **both** content-based filtering a
)

# Add legend
legend_text = (
    "CBF: Content-based Filtering"
    " \nNCBF: No Content-based Filtering"
    "\nI: Interactivity"
    "\nNI: No Interactivity"
)

# Position hypothesis text
plt.text(-1.85, 6.3, hypothesis_text, ha="left", fontsize=13, bbox=dict(facecolor="white", edgecolor="black", b
# Position legend text
plt.text(-1.85, 5, legend_text, ha="left", fontsize=13, bbox=dict(facecolor="white", edgecolor="black", boxstyl

plt.show()
```



4.2 Factorial RCT A-priori Power Analysis For Sample Size Determination

In [226…

```
from statsmodels.stats.power import TTestPower, GofChisquarePower

# Parameters

alpha = 0.05
power = 0.8
num_groups = 4  # Factorial RCT has 4 independent groups


# 1. Continuous Variables (One-Way ANOVA for comparing multiple groups)

effect_sizes = {
    "Post_Test_Quiz_Score": 0.3,  # Cohen's f (medium effect)
    "Study_Time": 0.20,       # Cohen's f (small effect)
    "Retention_Rate": 0.25    # Cohen's f (medium effect)
}

# Initialize power analyzer for ANOVA (one-way)
anova_power = TTestPower()  # Using TTestPower for simplicity, though ideally for ANOVA
continuous_results = {}
```

```
for metric, f in effect_sizes.items():
    # Calculate per-group sample size for ANOVA (one-way)
    n_per_group = anova_power.solve_power(
        effect_size=f,
        alpha=alpha,
        power=power,
        alternative='two-sided'
    )
    continuous_results[metric] = {
        "Per Group": int(round(n_per_group)),
        "Total": int(round(n_per_group * num_groups))
    }


# 2. Dropout Rate (Binary) (Chi-Square Test of Independence)

# Using Cramér's V = 0.2 (small effect for 4x2 table, as we have 4 groups)
chi2_power = GofChisquarePower()

total_dropout_n = chi2_power.solve_power(
    effect_size=0.2,  # Use Cohen's w (not Cramér's V) for goodness-of-fit
    alpha=alpha,
    power=power,
    nobs=None,
    n_bins=4  # Factorial RCT has 4 independent groups
)

# 3. Final Sample Size Calculation

# Get largest continuous variable requirement
max_continuous = max([v["Total"] for v in continuous_results.values()])
factorial_rct_num_students = max(max_continuous, int(round(total_dropout_n)))

print("Continuous Variables (One-Way ANOVA for Multiple Groups):")
for metric, res in continuous_results.items():
    print(f"- {metric}: {res['Per Group']} per group → {res['Total']} total")

print(f"\nDropout Rate (Chi-Square Test): {int(round(total_dropout_n))} total students")
print(f"\nFinal Recommended Sample Size for Factorial RCT: {factorial_rct_num_students} students")
```

```
Continuous Variables (One-Way ANOVA for Multiple Groups):
- Post_Test_Quiz_Score: 89 per group → 357 total
- Study_Time: 198 per group → 793 total
- Retention_Rate: 128 per group → 510 total

Dropout Rate (Chi-Square Test): 273 total students

Final Recommended Sample Size for Factorial RCT: 793 students
```

4.1.3 Factorial RCT Randomization (Blocking Factor)

In [227…

```
# Step 1: Collect Baseline Measures (Pre-Intervention Data)
factorial_rct_data = generate_baseline_data(factorial_rct_num_students)

# Step 2: Define Blocking Variable (Performance_Level)
# Combine GPA, Baseline Quiz Score, and Tech Savviness Score into a single performance score
factorial_rct_data['Performance_Score'] = (
    factorial_rct_data['GPA'] + factorial_rct_data['Baseline_Quiz_Score'] + factorial_rct_data['Tech_Savviness_S
)

# Define Performance_Level based on quantiles of the Performance_Score
factorial_rct_data['Performance_Level'] = pd.qcut(
    factorial_rct_data['Performance_Score'],
    q=3,  # Divide into 3 equal-sized blocks
    labels=['Low', 'Medium', 'High']
)

# Step 3: Blocked Randomization
np.random.seed(18)

# Initialize columns for treatment assignments
factorial_rct_data['Algorithm_Type'] = None
factorial_rct_data['Interactivity_Level'] = None

# Perform randomization within each block
for block in factorial_rct_data['Performance_Level'].unique():
    block_indices = factorial_rct_data[factorial_rct_data['Performance_Level'] == block].index

    # Randomly assign Algorithm_Type within the block
    factorial_rct_data.loc[block_indices, 'Algorithm_Type'] = np.random.choice(
        ['Content-based Filtering', 'No Content-based Filtering'], len(block_indices)
    )
```

```
    # Randomly assign Interactivity_Level within the block
    factorial_rct_data.loc[block_indices, 'Interactivity_Level'] = np.random.choice(
        ['Interactivity', 'No Interactivity'], len(block_indices)
    )

# Step 4: Create Combined Treatment Groups (Factorial Design)
factorial_rct_data['RCT_Factorial_Group'] = (
    factorial_rct_data['Algorithm_Type'] + ' + ' + factorial_rct_data['Interactivity_Level']
)

# Drop the temporary 'Performance_Score' column (optional)
factorial_rct_data.drop(columns=['Performance_Score'], inplace=True)
```

In [228…

```
print(factorial_rct_data.info())
print('\n')
print(factorial_rct_data.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 793 entries, 0 to 792
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Student_ID            793 non-null    int64
 1   Age                   793 non-null    int64
 2   GPA                   793 non-null    float64
 3   Study_Hours_Per_Week  793 non-null    int64
 4   Tech_Savviness_Score  793 non-null    int64
 5   Learning_Style        793 non-null    object
 6   Baseline_Quiz_Score   793 non-null    int64
 7   Attention_Span        793 non-null    int64
 8   Motivation_Level      793 non-null    int64
 9   Prior_Online_Exp      793 non-null    int64
 10  Performance_Level     793 non-null    category
 11  Algorithm_Type        793 non-null    object
 12  Interactivity_Level   793 non-null    object
 13  RCT_Factorial_Group   793 non-null    object
dtypes: category(1), float64(1), int64(8), object(4)
memory usage: 81.6+ KB
None


   Student_ID  Age  GPA  Study_Hours_Per_Week  Tech_Savviness_Score  \
0           1   22  2.7                    64                     8
1           2   26  3.6                    44                    -1
2           3   21  3.4                    47                     4
3           4   22  3.0                    50                     7
4           5   22  2.6                    59                     3

  Learning_Style  Baseline_Quiz_Score  Attention_Span  Motivation_Level  \
0         Visual                   62              32                 4
1    Kinesthetic                   66              17                 8
2       Auditory                   66              36                 8
3         Visual                   64              37                 9
4       Auditory                   64              19                 3

   Prior_Online_Exp Performance_Level              Algorithm_Type  \
0                 1            Medium     Content-based Filtering
1                 1               Low     Content-based Filtering
2                 0            Medium  No Content-based Filtering
3                 1            Medium     Content-based Filtering
4                 1               Low  No Content-based Filtering

  Interactivity_Level                          RCT_Factorial_Group
0       Interactivity     Content-based Filtering + Interactivity
1       Interactivity     Content-based Filtering + Interactivity
2       Interactivity  No Content-based Filtering + Interactivity
3       Interactivity     Content-based Filtering + Interactivity
4       Interactivity  No Content-based Filtering + Interactivity
```

4.1.4: Factorial RCT Implement Intervention (Blocking Factor)

- **Content-based system**:
  - **Boosts** Post_Test_Quiz_Score.
  - **Boosts** Study_Time.
  - **Boosts** Retention_Rate.
  - **Reduces** Dropout_Rate.
- **No Content-based system**:

    - **No effect on** Post_Test_Quiz_Score.
    - **No effect on** Study_Time.
    - **No effect on** Retention_Rate.
    - **No effect on** Dropout_Rate.
- **Interactivity**:
    - **Boosts** Post_Test_Quiz_Score.
    - **Boosts** Study_Time.
    - **Boosts** Retention_Rate.
    - **Reduces** Dropout_Rate.
- **No Interactivity**:
    - **Reduces** Post_Test_Quiz_Score.
    - **Reduces** Study_Time.
    - **Reduces** Retention_Rate.
    - **Increases** Dropout_Rate.

Intervention Changes

1. **Post_Test_Quiz_Score**:

- **No Content-based Filtering**: Change quiz scores by a random value sampled from a normal distribution with a mean of 0 and a standard deviation of 3.
- **Content-based system**: Boosts study time by a random value sampled from a normal distribution with a mean of 10 and a standard deviation of 5.
- **Interactivity**: Boosts quiz scores by a random value sampled from a normal distribution with a mean of 10 and a standard deviation of 5.
- **No Interactivity**: Decreases quiz scores by a random value sampled from a normal distribution with a mean of -5 and a standard deviation of 5.

2. **Improvement_Score**:

- The improvement score is calculated as the difference between the **Post_Test_Quiz_Score** and the **Baseline_Quiz_Score**. No specific changes made here, as it's derived from the updated quiz scores.

3. **Study_Time**:

- **Content-based system**: Boosts study time by a random value sampled from a normal distribution with a mean of 100 and a standard deviation of 50.
- **Interactivity**: Boosts study time by a random value sampled from a normal distribution with a mean of 100 and a standard deviation of 50.
- **No Interactivity**: Decreases study time by a random value sampled from a normal distribution with a mean of -50 and a standard deviation of 50.

4. **Retention_Rate**:

- **Content-based system**: Boosts retention rate by a random value sampled from a normal distribution with a mean of 5 and a standard deviation of 2.
- **Interactivity**: Boosts retention rate by a random value sampled from a normal distribution with a mean of 10 and a standard deviation of 3.
- **No Interactivity**: Decreases retention rate by a random value sampled from a normal distribution with a mean of -5 and a standard deviation of 3.

5. **Dropout_Rate**:

- **Content-based system**: Reduces dropout rate by a random value sampled from a normal distribution with a mean of -0.05 and a standard deviation of 0.1.
- **Interactivity**: Reduces dropout rate by a random value sampled from a normal distribution with a mean of -0.1 and a standard deviation of 0.1.
- **No Interactivity**: Increases dropout rate by a random value sampled from a normal distribution with a mean of 0.1 and a standard deviation of 0.1.
- **Clipping**: Dropout rate is clipped to ensure it stays within the range of 0 and 1 (representing percentages).

In [229…

```
# Step 3: Implement Intervention (Post-Intervention Data)
np.random.seed(18)

# Post_Test_Quiz_Score: Knowledge-based system boosts quiz scores, High Interactivity boosts quiz scores
```

```
factorial_rct_data['Post_Test_Quiz_Score'] = factorial_rct_data['Baseline_Quiz_Score'] + \
                                             np.where(factorial_rct_data['Algorithm_Type'] == 'Content-based
                                                      np.random.normal(10, 5, factorial_rct_num_students), #
                                                      np.random.normal(0, 3, factorial_rct_num_students))  #

factorial_rct_data['Post_Test_Quiz_Score'] += np.where(factorial_rct_data['Interactivity_Level'] == 'Interactiv
                                                      np.random.normal(10, 5, factorial_rct_num_students),  #
                                                      np.random.normal(-5, 5, factorial_rct_num_students))  #

# Improvement_Score: Difference between Post-Test and Baseline Quiz Scores
factorial_rct_data['Improvement_Score'] = (factorial_rct_data['Post_Test_Quiz_Score'] - factorial_rct_data['Bas

# Study_Time: Content-based system boosts study time, High Interactivity boosts study time
factorial_rct_data['Study_Time'] = np.where(factorial_rct_data['Algorithm_Type'] == 'Content-based Filtering',
                                            np.random.normal(100, 50, factorial_rct_num_students),  # Boost for
                                            np.random.normal(0, 50, factorial_rct_num_students))  # No change f
factorial_rct_data['Study_Time'] += np.where(factorial_rct_data['Interactivity_Level'] == 'Interactivity',
                                             np.random.normal(100, 50, factorial_rct_num_students),  # Boost f
                                             np.random.normal(-50, 50, factorial_rct_num_students))  # Decreas

# Retention_Rate: Content-based system boosts retention rate, High Interactivity boosts retention rate
factorial_rct_data['Retention_Rate'] = np.where(factorial_rct_data['Algorithm_Type'] == 'Content-based Filterin
                                                np.random.normal(5, 2, factorial_rct_num_students),  # Boost fo
                                                np.random.normal(0, 2, factorial_rct_num_students))  # No chang
factorial_rct_data['Retention_Rate'] += np.where(factorial_rct_data['Interactivity_Level'] == 'Interactivity',
                                                 np.random.normal(10, 3, factorial_rct_num_students),  # Boost
                                                 np.random.normal(-5, 3, factorial_rct_num_students))  # Decre

# # Dropout_Rate: Content-based system reduces dropout rate, High Interactivity reduces dropout rate
factorial_rct_data['Dropout_Rate'] = 0  # Initialize the Dropout Rate column

# Assign dropout rates based on the combination of Algorithm_Type and Interactivity_Level
for index, row in factorial_rct_data.iterrows():
    if row['Algorithm_Type'] == 'Content-based Filtering' and row['Interactivity_Level'] == 'Interactivity':
        dropout_rate = np.random.choice([0, 1], p=[0.95, 0.05])  # 5% dropout rate for Content-based + Interact
    elif row['Algorithm_Type'] == 'Content-based Filtering' and row['Interactivity_Level'] == 'No Interactivity
        dropout_rate = np.random.choice([0, 1], p=[0.90, 0.10])  # 10% dropout rate for Content-based + No Inte
    elif row['Algorithm_Type'] == 'No Content-based Filtering' and row['Interactivity_Level'] == 'Interactivity
        dropout_rate = np.random.choice([0, 1], p=[0.92, 0.08])  # 8% dropout rate for No Content-based + Inter
    else:  # No Content-based Filtering + No Interactivity
        dropout_rate = np.random.choice([0, 1], p=[0.85, 0.15])  # 15% dropout rate for No Content-based + No I

    # Assign the final dropout rate
    factorial_rct_data.at[index, 'Dropout_Rate'] = dropout_rate



# Ensure that dropout rate is between 0 and 1 (percentage)
factorial_rct_data['Dropout_Rate'] = np.clip(factorial_rct_data['Dropout_Rate'], 0, 1)
```

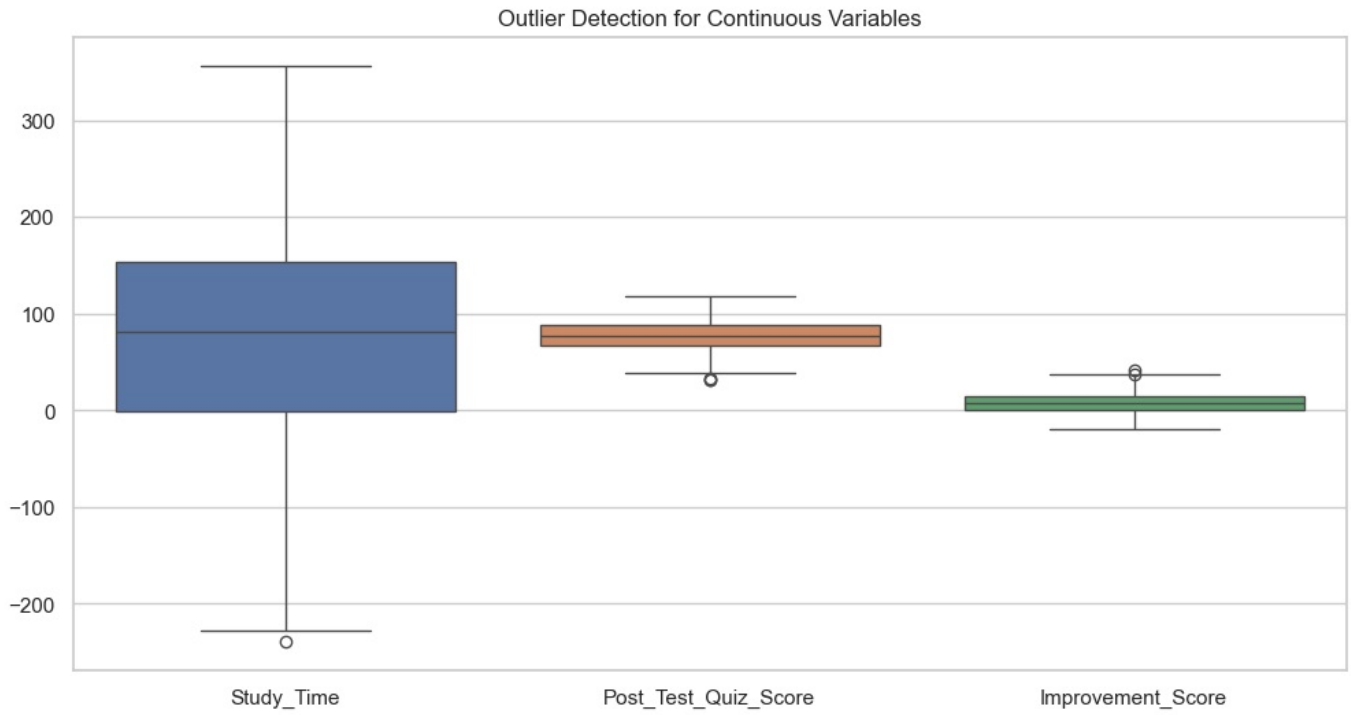
4.1.5: Factorial RCT Data Cleaning (Blocking Factor)

In [230…

```
# 1. Check for Missing Data

print("Missing Values Check:\n")
print(factorial_rct_data.isnull().sum())

# Replace placeholder codes (e.g., -18) with NaN if present
factorial_rct_data.replace(-18, np.nan, inplace=True)

# Drop rows with critical missing values (if any)
critical_cols = ['Post_Test_Quiz_Score', 'Retention_Rate', 'Dropout_Rate' ,'RCT_Factorial_Group']
factorial_rct_data.dropna(subset=critical_cols, inplace=True)


# 2. Remove Duplicates

print(f"\nInitial Shape: {factorial_rct_data.shape}")
factorial_rct_data.drop_duplicates(subset=['Student_ID'], keep='first', inplace=True)
print(f"Post-Deduplication Shape: {factorial_rct_data.shape}")

# 3. Verify Randomization Balance

print("\nGroup Balance:")
print(factorial_rct_data['RCT_Factorial_Group'].value_counts())

# Compare baseline covariates between groups using ANOVA
print("\nBaseline Covariate Balance:")
baseline_cols = ['GPA', 'Baseline_Quiz_Score', 'Study_Hours_Per_Week']
for col in baseline_cols:
    # Group data by RCT_Factorial_Group
    groups = [factorial_rct_data[factorial_rct_data['RCT_Factorial_Group'] == group][col]
```

```
            for group in factorial_rct_data['RCT_Factorial_Group'].unique()]

    # Perform one-way ANOVA test
    f_stat, p_value = f_oneway(*groups)
    print(f"{col}: F = {f_stat:.2f}, p = {p_value:.4f}")


# 4. Detect and Handle Outliers

continuous_vars = ['Study_Time', 'Post_Test_Quiz_Score', 'Improvement_Score']

plt.figure(figsize=(12, 6))
sns.boxplot(data=factorial_rct_data[continuous_vars])
plt.title("Outlier Detection for Continuous Variables")
plt.show()

# Cap outliers using IQR
for var in continuous_vars:
    q1 = factorial_rct_data[var].quantile(0.25)
    q3 = factorial_rct_data[var].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    factorial_rct_data[var] = np.where(factorial_rct_data[var] < lower_bound, lower_bound,
                                       np.where(factorial_rct_data[var] > upper_bound, upper_bound,
                                       factorial_rct_data[var]))


# 5. Validate Variable Ranges/Types

# Ensure binary variables are 0/1
factorial_rct_data['Dropout_Rate'] = factorial_rct_data['Dropout_Rate'].apply(lambda x: 1 if x == 1 else 0)

# Standardize categorical variables
factorial_rct_data['Learning_Style'] = factorial_rct_data['Learning_Style'].str.strip().str.title()

# Ensure valid percentage ranges
factorial_rct_data['Retention_Rate'] = factorial_rct_data['Retention_Rate'].clip(0, 100)


# 6. Save Cleaned Data

clean_factorial_rct_data = factorial_rct_data.copy()
print("\nData cleaning complete. Cleaned data saved as clean_factorial_rct_data.")
```

```
Missing Values Check:

Student_ID              0
Age                     0
GPA                     0
Study_Hours_Per_Week    0
Tech_Savviness_Score    0
Learning_Style          0
Baseline_Quiz_Score     0
Attention_Span          0
Motivation_Level        0
Prior_Online_Exp        0
Performance_Level       0
Algorithm_Type          0
Interactivity_Level     0
RCT_Factorial_Group     0
Post_Test_Quiz_Score    0
Improvement_Score       0
Study_Time              0
Retention_Rate          0
Dropout_Rate            0
dtype: int64

Initial Shape: (793, 19)
Post-Deduplication Shape: (793, 19)

Group Balance:
RCT_Factorial_Group
No Content-based Filtering + Interactivity       218
Content-based Filtering + No Interactivity       196
Content-based Filtering + Interactivity          194
No Content-based Filtering + No Interactivity    185
Name: count, dtype: int64

Baseline Covariate Balance:
GPA: F = 3.26, p = 0.0210
Baseline_Quiz_Score: F = 1.94, p = 0.1215
Study_Hours_Per_Week: F = 0.61, p = 0.6064
```




```
Data cleaning complete. Cleaned data saved as clean_factorial_rct_data.
```

4.1.6: Factorial RCT Perform Analysis (Blocking Factor)

In [231…

```python
# List of continuous metrics to analyze
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Perform Three-Way Factorial ANOVA (including the blocking factor)
anova_results = {}
post_hoc_results = {}

for metric in metrics:
    # Perform ANOVA for each combination of Algorithm_Type, Interactivity_Level, and Performance_Level
    model = ols(
        f'{metric} ~ C(Algorithm_Type) * C(Interactivity_Level) + C(Performance_Level)',
        data=clean_factorial_rct_data
    ).fit()
    anova_table = sm.stats.anova_lm(model, typ=2)  # Type 2 ANOVA (for factorial designs)

    # Extract F-statistic and p-value for the main effects and interactions
    anova_results[metric] = {
        'F-Statistic (Algorithm_Type)': anova_table.loc['C(Algorithm_Type)', 'F'],
        'p-value (Algorithm_Type)': anova_table.loc['C(Algorithm_Type)', 'PR(>F)'],
        'F-Statistic (Interactivity_Level)': anova_table.loc['C(Interactivity_Level)', 'F'],
        'p-value (Interactivity_Level)': anova_table.loc['C(Interactivity_Level)', 'PR(>F)'],
        'F-Statistic (Performance_Level)': anova_table.loc['C(Performance_Level)', 'F'],
        'p-value (Performance_Level)': anova_table.loc['C(Performance_Level)', 'PR(>F)'],
        'F-Statistic (Algorithm_Type:Interactivity_Level)': anova_table.loc['C(Algorithm_Type):C(Interactivity_
        'p-value (Algorithm_Type:Interactivity_Level)': anova_table.loc['C(Algorithm_Type):C(Interactivity_Leve
    }

    # Post-Hoc Analysis (Tukey's HSD) for significant main effects or interactions
    post_hoc_results[metric] = {}

    # Check for significant main effects or interactions
    if anova_table.loc['C(Algorithm_Type)', 'PR(>F)'] < 0.05:
        # Perform Tukey's HSD for Algorithm_Type
        tukey_results = pairwise_tukeyhsd(endog=clean_factorial_rct_data[metric],
                                          groups=clean_factorial_rct_data['Algorithm_Type'],
                                          alpha=0.05)
        post_hoc_results[metric]['Algorithm_Type'] = tukey_results

    if anova_table.loc['C(Interactivity_Level)', 'PR(>F)'] < 0.05:
        # Perform Tukey's HSD for Interactivity_Level
        tukey_results = pairwise_tukeyhsd(endog=clean_factorial_rct_data[metric],
                                          groups=clean_factorial_rct_data['Interactivity_Level'],
                                          alpha=0.05)
        post_hoc_results[metric]['Interactivity_Level'] = tukey_results

    if anova_table.loc['C(Algorithm_Type):C(Interactivity_Level)', 'PR(>F)'] < 0.05:
        # Perform Tukey's HSD for the interaction between Algorithm_Type and Interactivity_Level
        combined_groups = (clean_factorial_rct_data['Algorithm_Type'].astype(str) + "_" +
                           clean_factorial_rct_data['Interactivity_Level'].astype(str))
```

```
        tukey_results = pairwise_tukeyhsd(endog=clean_factorial_rct_data[metric],
                                          groups=combined_groups,
                                          alpha=0.05)
        post_hoc_results[metric]['Algorithm_Type:Interactivity_Level'] = tukey_results

# Convert ANOVA results to DataFrame
anova_results_df = pd.DataFrame(anova_results)

# Add significance columns for each effect
for col in anova_results_df.columns:
    if 'p-value' in col:
        effect_name = col.split('p-value (')[1].split(')')[0]  # Extract effect name
        anova_results_df[f'Significance ({effect_name})'] = anova_results_df[col].apply(
            lambda x: 'Significant' if x < 0.05 else 'Not Significant'
        )

# Display ANOVA Results
print("\n=== ANOVA Results (Combined for All Metrics) ===\n")
display(anova_results_df)
```

```
=== ANOVA Results (Combined for All Metrics) ===
```

	Post_Test_Quiz_Score	Improvement_Score	Study_Time	Retention_Rate
F-Statistic (Algorithm_Type)	2.799834e+02	4.169493e+02	4.461301e+02	2.638974e+02
p-value (Algorithm_Type)	5.355696e-54	1.070982e-74	8.466011e-79	2.159341e-51
F-Statistic (Interactivity_Level)	7.020150e+02	9.019846e+02	8.678967e+02	3.453389e+03
p-value (Interactivity_Level)	4.433492e-111	1.215898e-132	3.742700e-129	4.788338e-290
F-Statistic (Performance_Level)	5.007282e+02	5.094973e+00	7.397780e-01	1.245507e-01
p-value (Performance_Level)	5.197860e-141	6.331167e-03	4.775514e-01	8.829109e-01
F-Statistic (Algorithm_Type:Interactivity_Level)	4.336541e-03	1.615843e+00	1.233488e-02	1.172728e+02
p-value (Algorithm_Type:Interactivity_Level)	9.475120e-01	2.040480e-01	9.115951e-01	1.432078e-25

In [232…

```
# === Mean Comparisons Table ===
# Group by factors and calculate mean
mean_comparison = clean_factorial_rct_data.groupby(
    ['Algorithm_Type', 'Interactivity_Level', 'Performance_Level']
)[metrics].mean().reset_index()

# Round 'Retention_Rate' to 2 decimal places
mean_comparison['Retention_Rate'] = mean_comparison['Retention_Rate'].round(2)

# Reshape mean comparison table
mean_comparison_melted = mean_comparison.melt(
    id_vars=['Algorithm_Type', 'Interactivity_Level', 'Performance_Level'],
    value_vars=metrics,
    var_name='Metric',
    value_name='Mean Value'
)

# === Display Separate Mean Comparisons Tables for Each Metric ===
for metric in metrics:
    mean_comparison_metric = mean_comparison_melted[mean_comparison_melted['Metric'] == metric]

    # Display mean comparison table for each metric
    print(f"\n=== Mean Comparisons for {metric} ===\n")
    display(mean_comparison_metric)
```

```
=== Mean Comparisons for Post_Test_Quiz_Score ===
```

	Algorithm_Type	Interactivity_Level	Performance_Level	Metric	Mean Value
0	Content-based Filtering	Interactivity	Low	Post_Test_Quiz_Score	81.168250
1	Content-based Filtering	Interactivity	Medium	Post_Test_Quiz_Score	90.253881
2	Content-based Filtering	Interactivity	High	Post_Test_Quiz_Score	97.927673
3	Content-based Filtering	No Interactivity	Low	Post_Test_Quiz_Score	61.998209
4	Content-based Filtering	No Interactivity	Medium	Post_Test_Quiz_Score	74.591364
5	Content-based Filtering	No Interactivity	High	Post_Test_Quiz_Score	86.585939
6	No Content-based Filtering	Interactivity	Low	Post_Test_Quiz_Score	69.856468
7	No Content-based Filtering	Interactivity	Medium	Post_Test_Quiz_Score	78.276322
8	No Content-based Filtering	Interactivity	High	Post_Test_Quiz_Score	91.900803
9	No Content-based Filtering	No Interactivity	Low	Post_Test_Quiz_Score	52.590197
10	No Content-based Filtering	No Interactivity	Medium	Post_Test_Quiz_Score	64.414436
11	No Content-based Filtering	No Interactivity	High	Post_Test_Quiz_Score	77.769259

```
=== Mean Comparisons for Improvement_Score ===
```

	Algorithm_Type	Interactivity_Level	Performance_Level	Metric	Mean Value
12	Content-based Filtering	Interactivity	Low	Improvement_Score	20.303571
13	Content-based Filtering	Interactivity	Medium	Improvement_Score	20.295775
14	Content-based Filtering	Interactivity	High	Improvement_Score	18.791045
15	Content-based Filtering	No Interactivity	Low	Improvement_Score	2.826087
16	Content-based Filtering	No Interactivity	Medium	Improvement_Score	6.018519
17	Content-based Filtering	No Interactivity	High	Improvement_Score	6.698630
18	No Content-based Filtering	Interactivity	Low	Improvement_Score	9.160000
19	No Content-based Filtering	Interactivity	Medium	Improvement_Score	9.256757
20	No Content-based Filtering	Interactivity	High	Improvement_Score	10.550725
21	No Content-based Filtering	No Interactivity	Low	Improvement_Score	-5.125000
22	No Content-based Filtering	No Interactivity	Medium	Improvement_Score	-4.033333
23	No Content-based Filtering	No Interactivity	High	Improvement_Score	-1.849057

```
=== Mean Comparisons for Study_Time ===
```

	Algorithm_Type	Interactivity_Level	Performance_Level	Metric	Mean Value
24	Content-based Filtering	Interactivity	Low	Study_Time	191.778892
25	Content-based Filtering	Interactivity	Medium	Study_Time	199.526073
26	Content-based Filtering	Interactivity	High	Study_Time	203.997810
27	Content-based Filtering	No Interactivity	Low	Study_Time	45.229115
28	Content-based Filtering	No Interactivity	Medium	Study_Time	65.657485
29	Content-based Filtering	No Interactivity	High	Study_Time	55.991886
30	No Content-based Filtering	Interactivity	Low	Study_Time	109.935593
31	No Content-based Filtering	Interactivity	Medium	Study_Time	89.865287
32	No Content-based Filtering	Interactivity	High	Study_Time	86.431903
33	No Content-based Filtering	No Interactivity	Low	Study_Time	-32.220657
34	No Content-based Filtering	No Interactivity	Medium	Study_Time	-56.554635
35	No Content-based Filtering	No Interactivity	High	Study_Time	-57.031572

```
=== Mean Comparisons for Retention_Rate ===
```

	Algorithm_Type	Interactivity_Level	Performance_Level	Metric	Mean Value
36	Content-based Filtering	Interactivity	Low	Retention_Rate	15.09
37	Content-based Filtering	Interactivity	Medium	Retention_Rate	14.70
38	Content-based Filtering	Interactivity	High	Retention_Rate	15.75
39	Content-based Filtering	No Interactivity	Low	Retention_Rate	1.12
40	Content-based Filtering	No Interactivity	Medium	Retention_Rate	1.13
41	Content-based Filtering	No Interactivity	High	Retention_Rate	1.18
42	No Content-based Filtering	Interactivity	Low	Retention_Rate	10.11
43	No Content-based Filtering	Interactivity	Medium	Retention_Rate	10.05
44	No Content-based Filtering	Interactivity	High	Retention_Rate	9.26
45	No Content-based Filtering	No Interactivity	Low	Retention_Rate	0.15
46	No Content-based Filtering	No Interactivity	Medium	Retention_Rate	0.17
47	No Content-based Filtering	No Interactivity	High	Retention_Rate	0.09

In [233…

```
# Display Post-Hoc Results
for metric, results in post_hoc_results.items():
    print(f"\n\u001b[1m=== Post-Hoc Analysis for {metric} ===\u001b[0m\n")
    for effect, tukey_result in results.items():
        print(f"\nTukey's HSD for {effect}:\n")
        print(tukey_result.summary())
```

=== Post-Hoc Analysis for Post_Test_Quiz_Score ===

Tukey's HSD for Algorithm_Type:

```
                  Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================================
         group1                   group2           meandiff p-adj  lower   upper  reject
---------------------------------------------------------------------------------------------
Content-based Filtering No Content-based Filtering -10.0908   0.0 -12.1136 -8.068   True
---------------------------------------------------------------------------------------------
```

Tukey's HSD for Interactivity_Level:

```
         Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
    group1         group2      meandiff p-adj  lower    upper   reject
-----------------------------------------------------------------------------
Interactivity No Interactivity -15.3877   0.0 -17.2424 -13.5329   True
-----------------------------------------------------------------------------
```

=== Post-Hoc Analysis for Improvement_Score ===

Tukey's HSD for Algorithm_Type:

```
                   Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================================
         group1                   group2           meandiff p-adj  lower    upper  reject
---------------------------------------------------------------------------------------------
Content-based Filtering No Content-based Filtering   -8.974   0.0 -10.3134 -7.6347   True
---------------------------------------------------------------------------------------------
```

Tukey's HSD for Interactivity_Level:

```
         Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
    group1         group2      meandiff p-adj  lower    upper   reject
-----------------------------------------------------------------------------
Interactivity No Interactivity -13.6228   0.0 -14.7563 -12.4893   True
-----------------------------------------------------------------------------
```

=== Post-Hoc Analysis for Study_Time ===

Tukey's HSD for Algorithm_Type:

```
                   Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================================
         group1                   group2           meandiff p-adj  lower    upper   reject
---------------------------------------------------------------------------------------------
Content-based Filtering No Content-based Filtering -96.3986   0.0 -110.206 -82.5912   True
---------------------------------------------------------------------------------------------
```

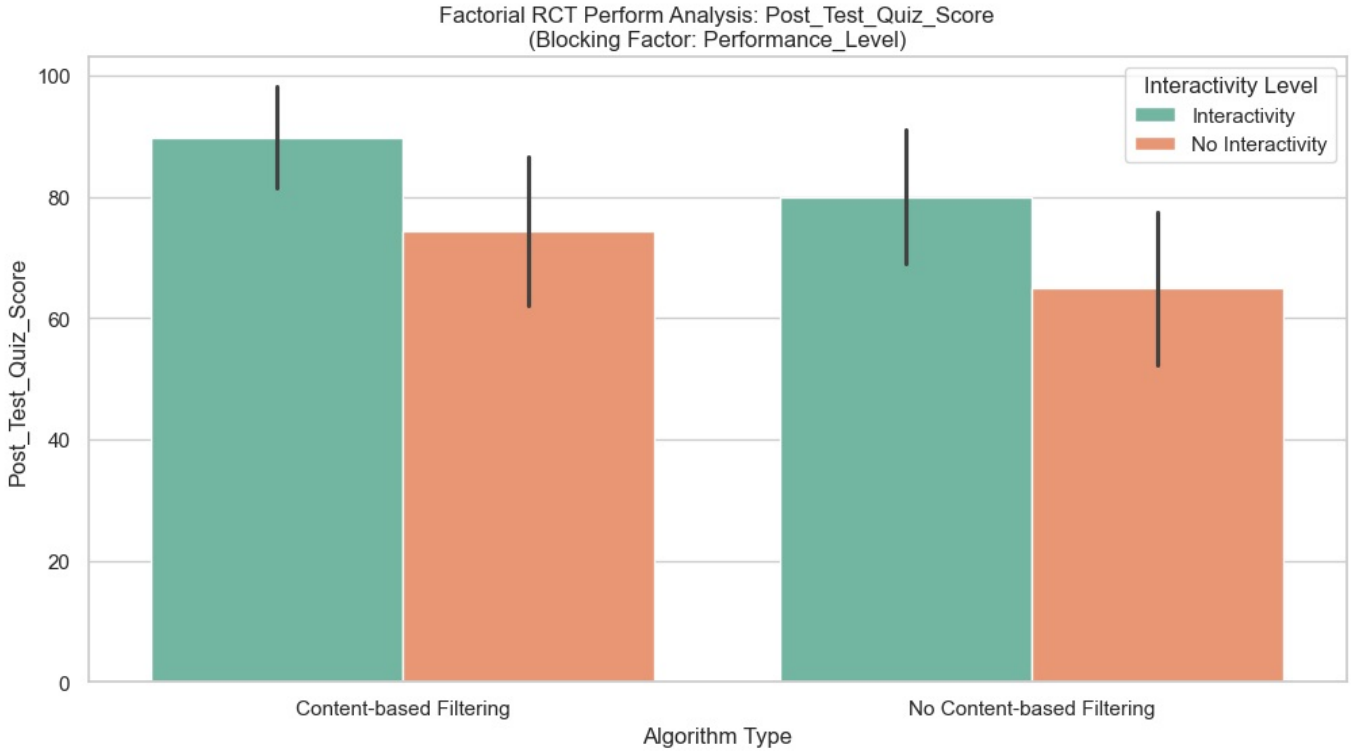
```
Tukey's HSD for Interactivity_Level:

           Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
    group1         group2       meandiff p-adj   lower     upper   reject
-----------------------------------------------------------------------------
Interactivity No Interactivity -138.9582   0.0 -150.8807 -127.0356   True
-----------------------------------------------------------------------------

=== Post-Hoc Analysis for Retention_Rate ===


Tukey's HSD for Algorithm_Type:

                  Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================================
         group1                   group2           meandiff p-adj  lower   upper  reject
---------------------------------------------------------------------------------------------
Content-based Filtering No Content-based Filtering  -2.7476   0.0 -3.6728 -1.8224   True
---------------------------------------------------------------------------------------------

Tukey's HSD for Interactivity_Level:

         Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
    group1         group2      meandiff p-adj  lower    upper   reject
-----------------------------------------------------------------------------
Interactivity No Interactivity -11.6818   0.0 -12.1605 -11.2031   True
-----------------------------------------------------------------------------

Tukey's HSD for Algorithm_Type:Interactivity_Level:

                                    Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================================================================================
                  group1                                        group2                   meandiff p-adj  lower    up
per   reject
---------------------------------------------------------------------------------------------------------------------------------------------
------------
   Content-based Filtering_Interactivity    Content-based Filtering_No Interactivity -14.0263   0.0 -14.7615 -13
.2912   True
   Content-based Filtering_Interactivity    No Content-based Filtering_Interactivity  -5.3538   0.0  -6.0703  -4
.6374   True
   Content-based Filtering_Interactivity No Content-based Filtering_No Interactivity -15.0321   0.0 -15.7781 -14
.2861   True
Content-based Filtering_No Interactivity    No Content-based Filtering_Interactivity   8.6725   0.0   7.9579   9
.3871   True
Content-based Filtering_No Interactivity No Content-based Filtering_No Interactivity  -1.0057 0.003  -1.7499  -0
.2616   True
No Content-based Filtering_Interactivity No Content-based Filtering_No Interactivity  -9.6782   0.0 -10.4039  -8
.9526   True
---------------------------------------------------------------------------------------------------------------------------------------------
------------
```

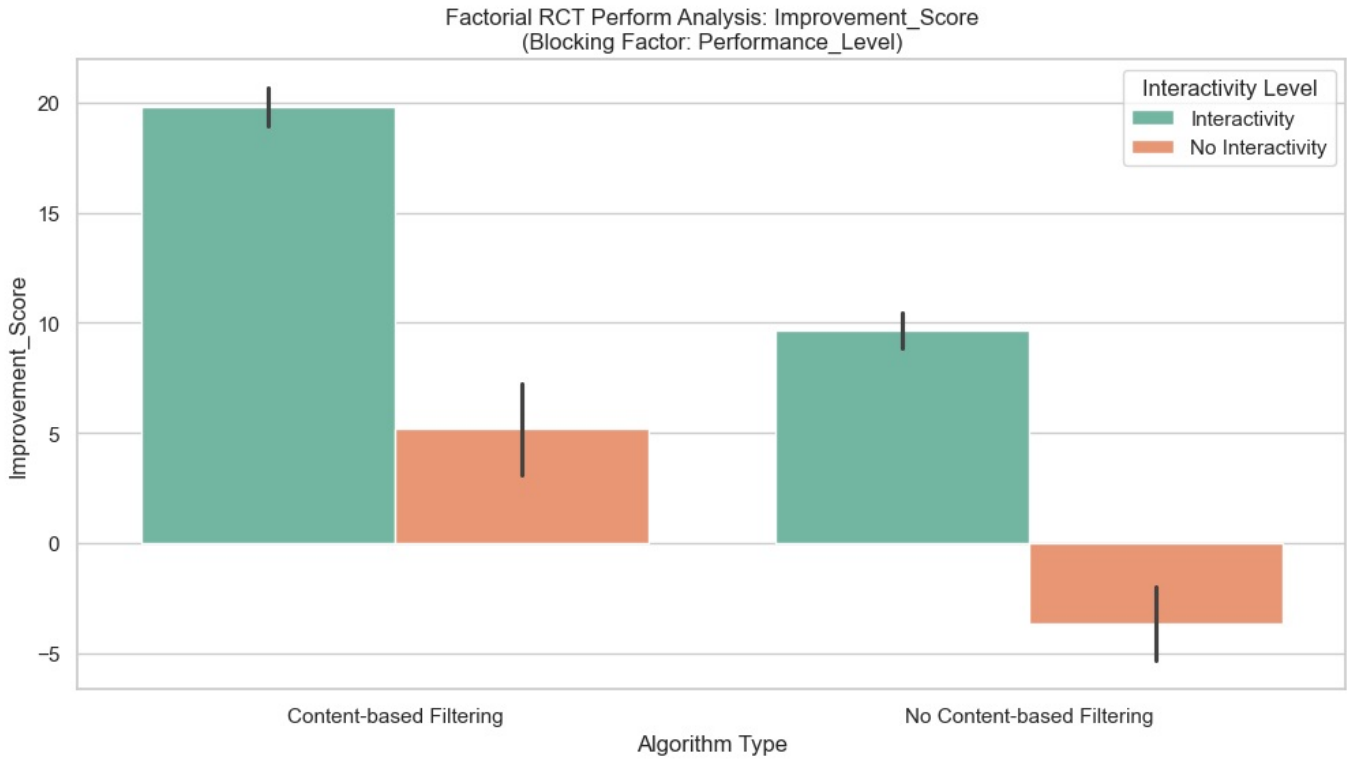
In [234…

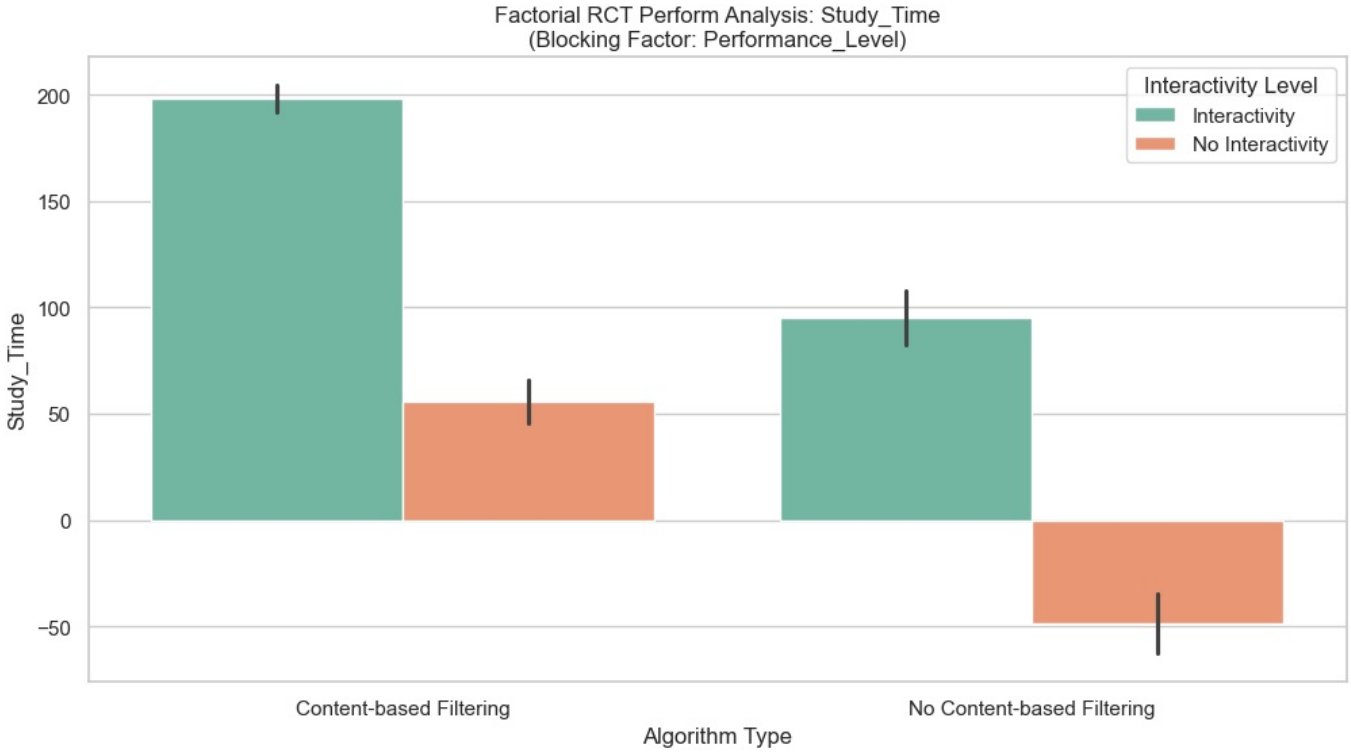
```python
# Set the style for the plots
sns.set(style="whitegrid")

# Function to plot grouped bar plots for each metric
def plot_factorial_rct_results(mean_comparison_melted, anova_results_df):
  for metric in metrics:
      # Filter mean comparison data for the current metric
      metric_data = mean_comparison_melted[mean_comparison_melted['Metric'] == metric]

      # Create a grouped bar plot
      plt.figure(figsize=(12, 6))
      sns.barplot(
          x='Algorithm_Type',
          y='Mean Value',
          hue='Interactivity_Level',
          data=metric_data,
          ci='sd',  # Add error bars (standard deviation)
          palette='Set2'
      )

      # Add title and labels
      plt.title(
          f'Factorial RCT Perform Analysis: {metric}\n'
          f'(Blocking Factor: Performance_Level)'
      )
      plt.xlabel('Algorithm Type')
      plt.ylabel(metric)
      plt.legend(title='Interactivity Level', loc='upper right')
```

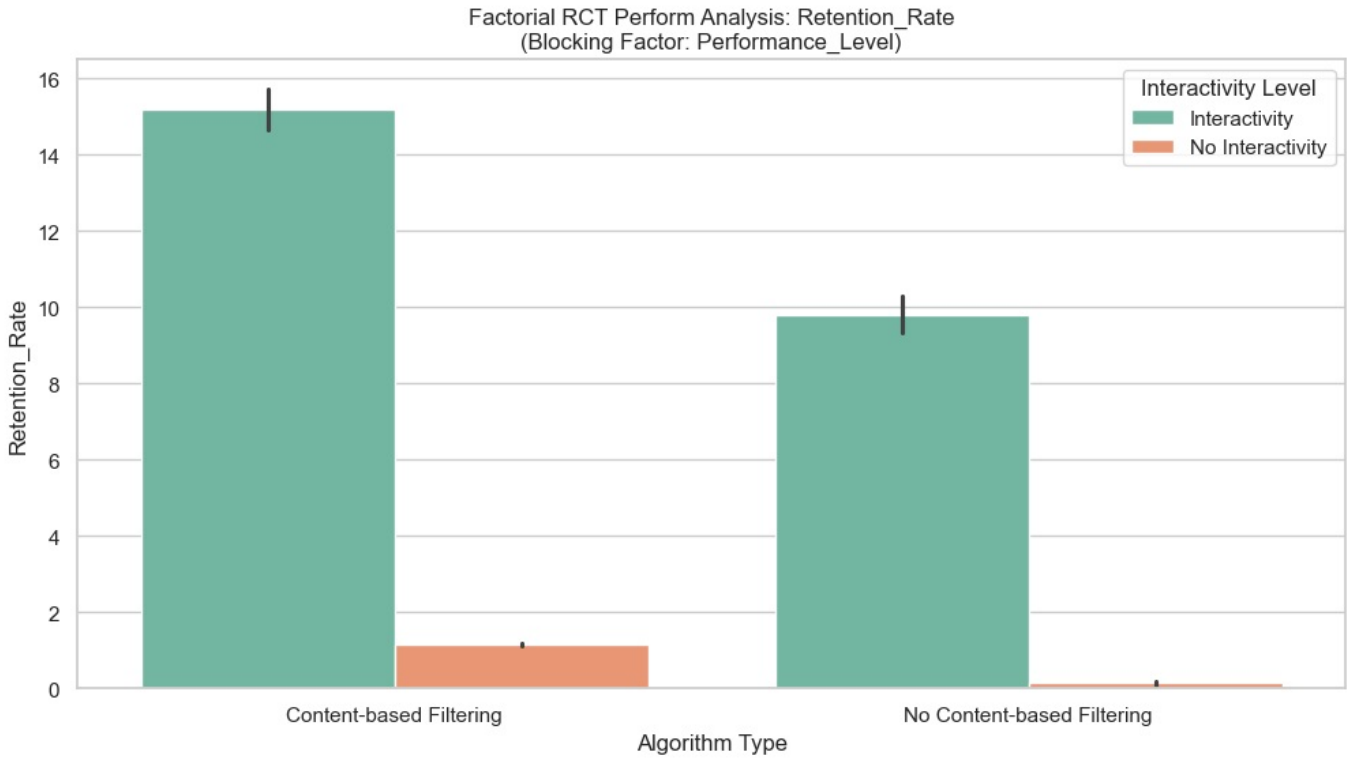
```
    plt.show()

# Plot grouped bar plots for each metric
print("Visualizing Factorial RCT Perform Analysis Results\n")
plot_factorial_rct_results(mean_comparison_melted, anova_results_df)
```

```
Visualizing Factorial RCT Perform Analysis Results
```







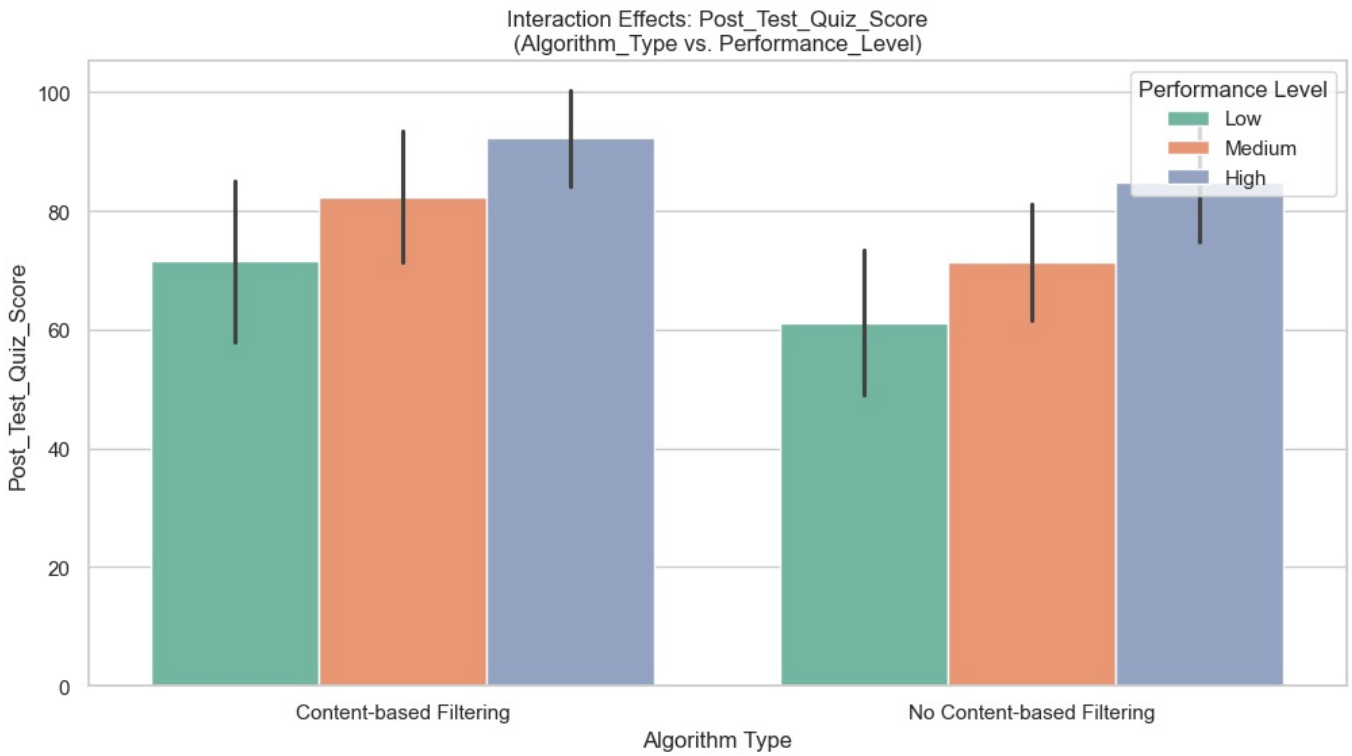


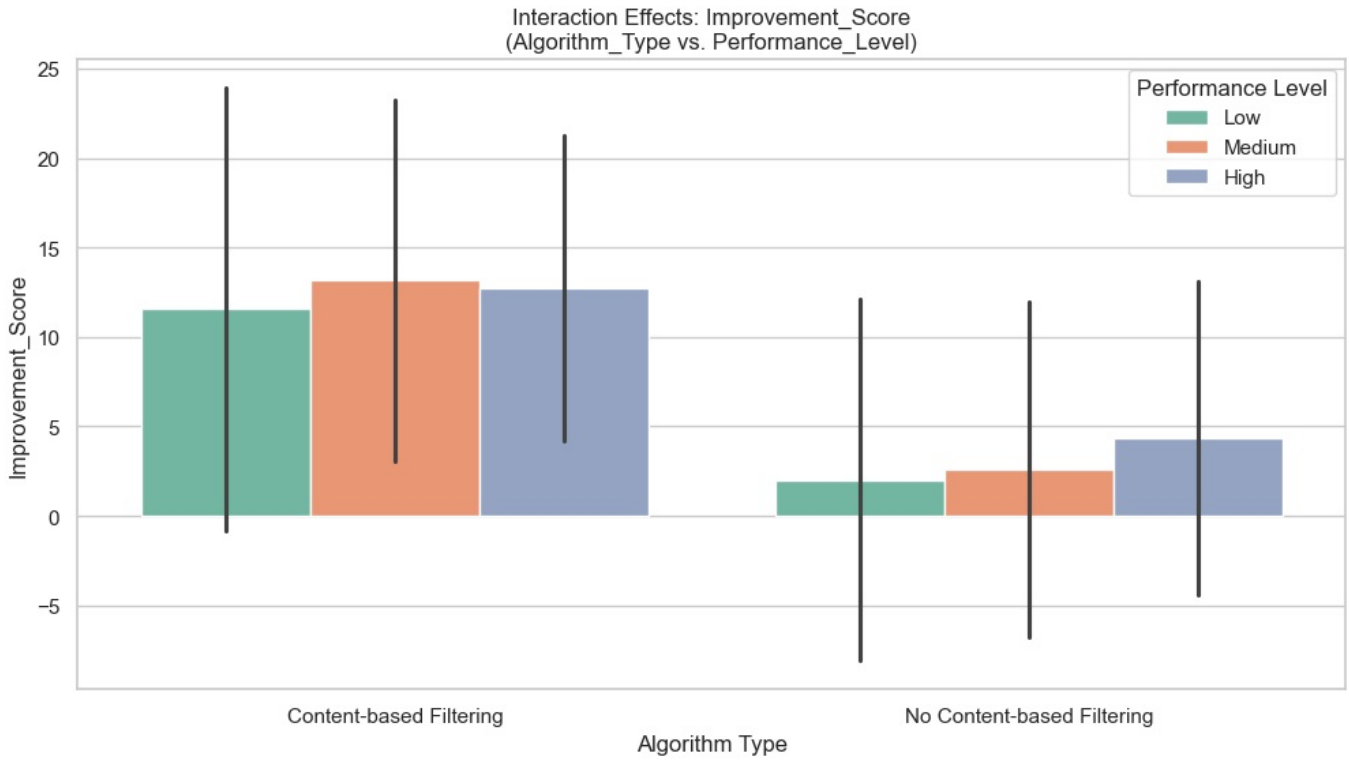
In [235…

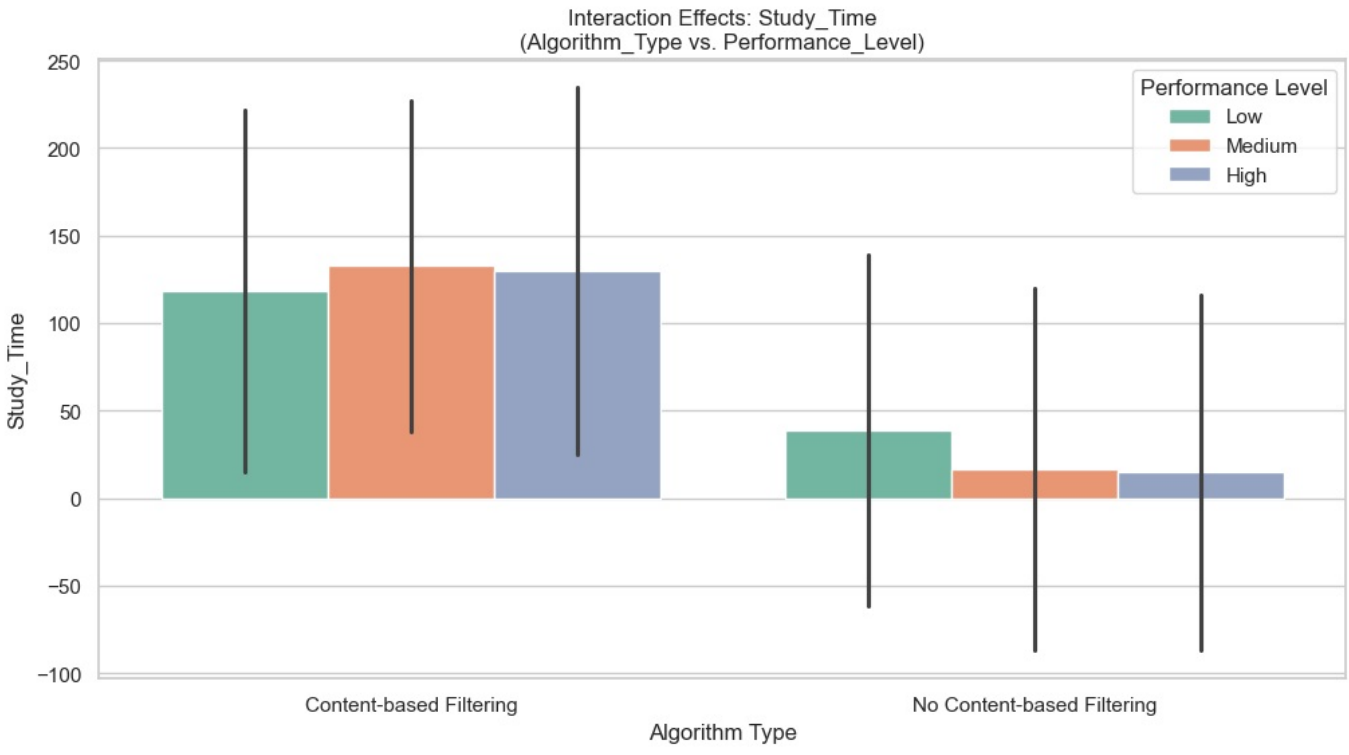
```python
# Function to plot interaction effects for each metric
def plot_interaction_effects(mean_comparison_melted, anova_results_df):
    for metric in metrics:
        # Filter mean comparison data for the current metric
        metric_data = mean_comparison_melted[mean_comparison_melted['Metric'] == metric]

        # Create a grouped bar plot for interaction effects
```

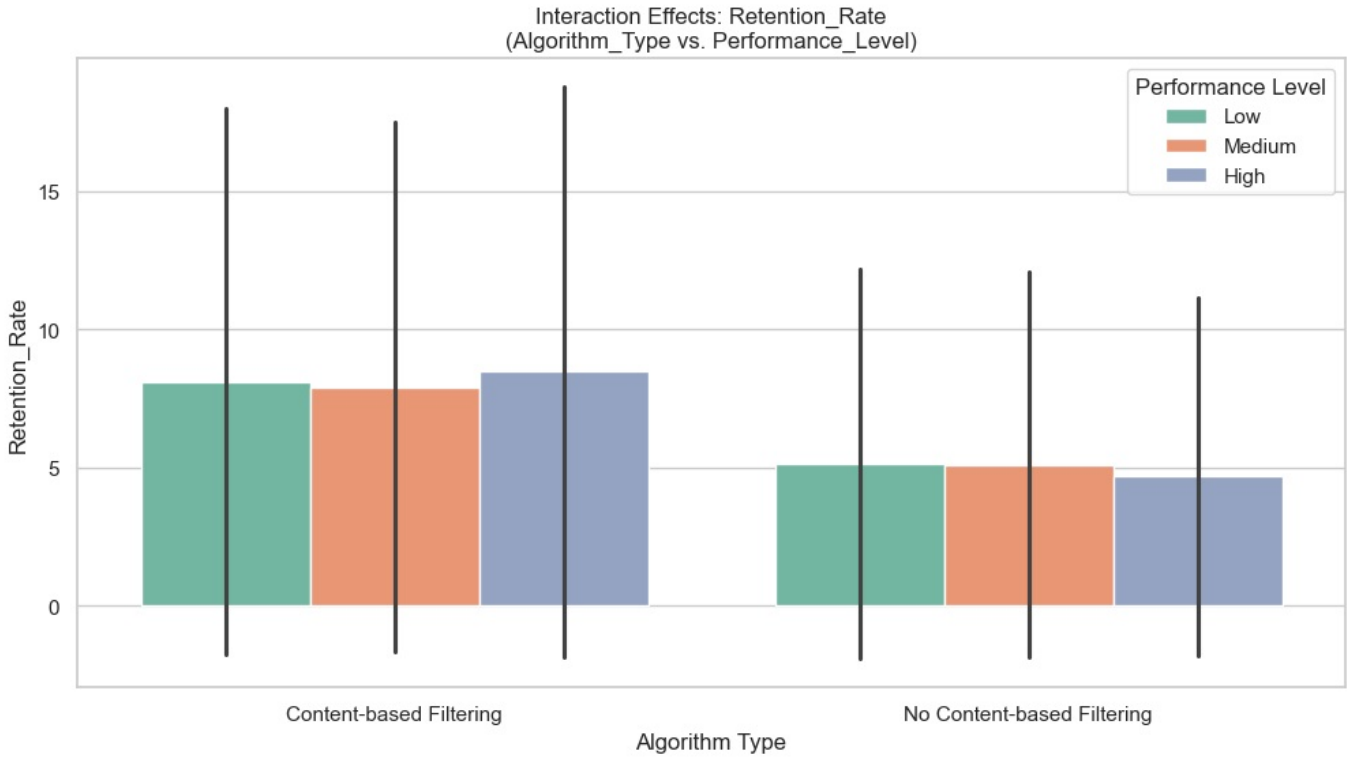
```
        plt.figure(figsize=(12, 6))
        sns.barplot(
            x='Algorithm_Type',
            y='Mean Value',
            hue='Performance_Level',
            data=metric_data,
            ci='sd',  # Add error bars (standard deviation)
            palette='Set2'
        )

        # Add title and labels
        plt.title(
            f'Interaction Effects: {metric}\n'
            f'(Algorithm_Type vs. Performance_Level)'
        )
        plt.xlabel('Algorithm Type')
        plt.ylabel(metric)
        plt.legend(title='Performance Level', loc='upper right')

        # Check if the interaction is significant and annotate
        if 'Significance (Algorithm_Type:Interactivity_Level)' in anova_results_df.columns:
            if anova_results_df.loc[metric, 'Significance (Algorithm_Type:Interactivity_Level)'] == 'Significant
                plt.text(
                    0.5, 0.95,
                    'Significant Interaction (Algorithm_Type:Interactivity_Level)',
                    transform=plt.gca().transAxes,
                    fontsize=10,
                    color='red',
                    ha='center'
                )

        plt.show()

print("Visualizing Interaction Effects for Each Metric\n")
plot_interaction_effects(mean_comparison_melted, anova_results_df)
```

```
Visualizing Interaction Effects for Each Metric
```




Interaction Effects: Improvement_Score
(Algorithm_Type vs. Performance_Level)
Performance Level
Low
Medium
High
Improvement_Score
25
20
15
10
5
0
−5
Content-based Filtering
No Content-based Filtering
Algorithm Type


Interaction Effects: Study_Time
(Algorithm_Type vs. Performance_Level)
Performance Level
Low
Medium
High
Study_Time
250
200
150
100
50
0
−50
−100
Content-based Filtering
No Content-based Filtering
Algorithm Type



In [236…

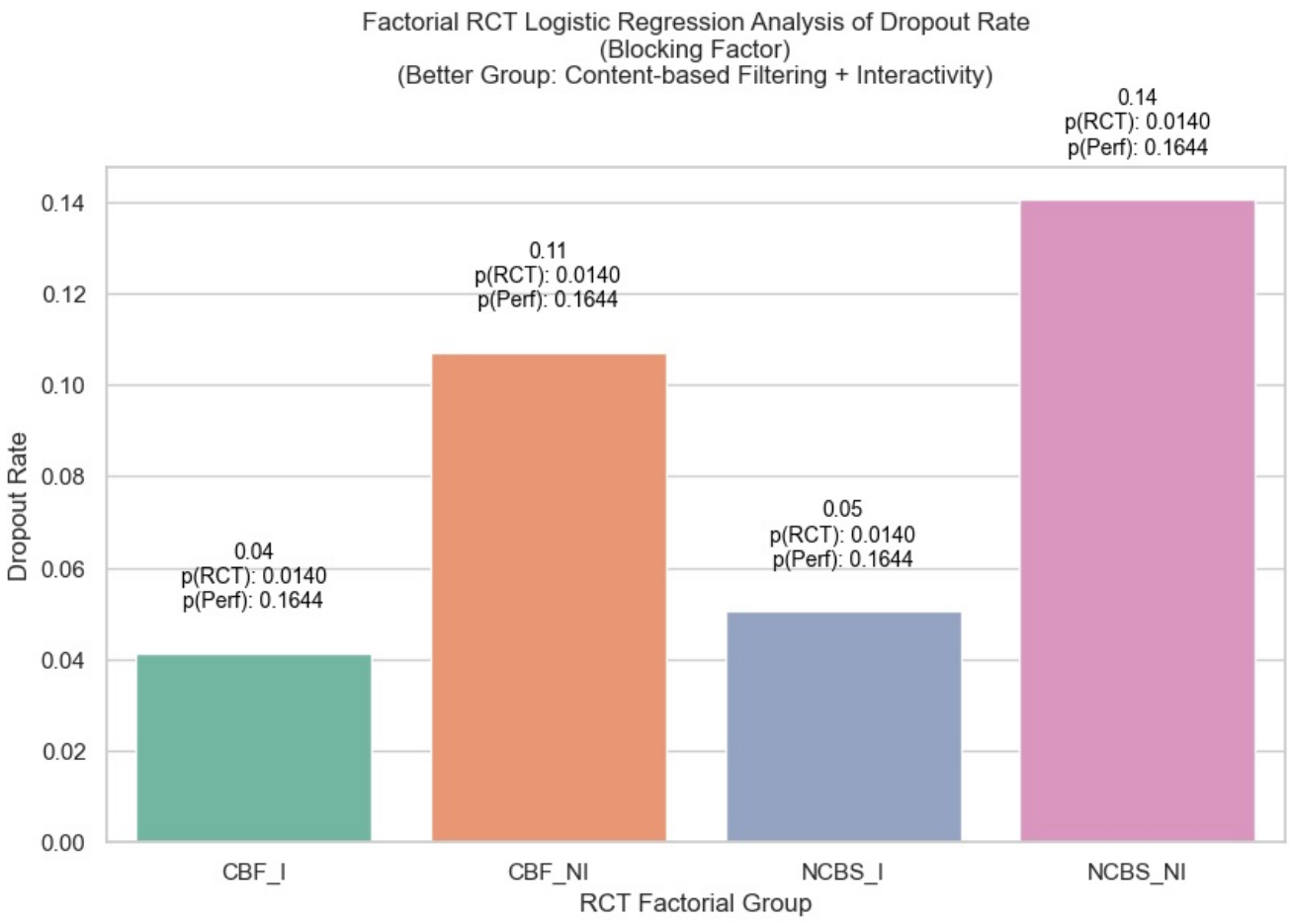
```
from statsmodels.formula.api import logit

# Step 1: Define the logistic regression formula
formula = 'Dropout_Rate ~ C(RCT_Factorial_Group) + C(Performance_Level)'

# Step 2: Fit the logistic regression model
logit_model = logit(formula, data=clean_factorial_rct_data).fit()

# Step 3: Extract model results
logit_results = {
    'Metric': 'Dropout Rate',
    'CBF_I': clean_factorial_rct_data[clean_factorial_rct_data['RCT_Factorial_Group'] == 'Content-based Filteri
    'CBF_NI': clean_factorial_rct_data[clean_factorial_rct_data['RCT_Factorial_Group'] == 'Content-based Filter
    'NCBS_I': clean_factorial_rct_data[clean_factorial_rct_data['RCT_Factorial_Group'] == 'No Content-based Fil
    'NCBS_NI': clean_factorial_rct_data[clean_factorial_rct_data['RCT_Factorial_Group'] == 'No Content-based Fi
    'Better Group': clean_factorial_rct_data.groupby('RCT_Factorial_Group')['Dropout_Rate'].mean().idxmin(),
    'Coefficient (RCT_Factorial_Group)': logit_model.params['C(RCT_Factorial_Group)[T.Content-based Filtering +
    'p-value (RCT_Factorial_Group)': logit_model.pvalues['C(RCT_Factorial_Group)[T.Content-based Filtering + No
    'Coefficient (Performance_Level)': logit_model.params['C(Performance_Level)[T.Medium]'],
    'p-value (Performance_Level)': logit_model.pvalues['C(Performance_Level)[T.Medium]'],
    'Significance (RCT_Factorial_Group)': 'Significant' if logit_model.pvalues['C(RCT_Factorial_Group)[T.Conten
    'Significance (Performance_Level)': 'Significant' if logit_model.pvalues['C(Performance_Level)[T.Medium]'] 
}

# Step 4: Convert results to DataFrame
dropout_results_df = pd.DataFrame(logit_results, index=[0]).T

# Step 5: Display the results table
print("\n4.1.6: Factorial RCT Logistic Regression Analysis of Dropout Rate with Blocking Factor\n")
display(dropout_results_df)
```

```
Optimization terminated successfully.
         Current function value: 0.274319
         Iterations 7

4.1.6: Factorial RCT Logistic Regression Analysis of Dropout Rate with Blocking Factor
```

	0
Metric	Dropout Rate
CBF_I	0.041237
CBF_NI	0.107143
NCBS_I	0.050459
NCBS_NI	0.140541
Better Group	Content-based Filtering + Interactivity
Coefficient (RCT_Factorial_Group)	1.056558
p-value (RCT_Factorial_Group)	0.013995
Coefficient (Performance_Level)	0.457469
p-value (Performance_Level)	0.164402
Significance (RCT_Factorial_Group)	Significant
Significance (Performance_Level)	Not Significant

In [237…

```
# Set the style for the plots
sns.set(style="whitegrid")

# Extract data for visualization
groups = ['CBF_I', 'CBF_NI', 'NCBS_I', 'NCBS_NI']
dropout_rates = [
    logit_results['CBF_I'],
    logit_results['CBF_NI'],
    logit_results['NCBS_I'],
    logit_results['NCBS_NI']
]
better_group = logit_results['Better Group']

# Create a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(
    x=groups,
    y=dropout_rates,
    palette='Set2'
)

# Add title and labels
plt.title(
    'Factorial RCT Logistic Regression Analysis of Dropout Rate\n(Blocking Factor)\n'
    f'(Better Group: {better_group})',
    pad=40  # Increase padding between title and plot
)
plt.xlabel('RCT Factorial Group')
plt.ylabel('Dropout Rate')

# Add annotations for significance and coefficients
for i, rate in enumerate(dropout_rates):
    plt.text(
        i, rate + 0.01,  # Adjust text position
        f'{rate:.2f}\n'
        f'p(RCT): {logit_results["p-value (RCT_Factorial_Group)"]:.4f}\n'
        f'p(Perf): {logit_results["p-value (Performance_Level)"]:.4f}',
        ha='center',
        fontsize=10,
        color='black'
    )


# Adjust the top margin to increase space between bars and title
plt.subplots_adjust(top=0.85)  # Increase the top margin

# Show the plot
plt.show()
```



4.1.7 Interpretation of Factorial RCT Results (Blocking Factor)

Effect of Blocking Variable (Performance_Level):

The **Performance_Level** as a blocking variable is included to control for any potential confounding effects related to the baseline performance of participants. This ensures that the comparisons between groups are adjusted for differences in participants' initial performance levels (such as GPA or Baseline Quiz Score).

- **Effect on Metrics**:
  - **Post_Test_Quiz_Score**: There is a significant effect of **Performance_Level** on this metric (F-statistic = 4.10, p-value = 0.0168). This means that Performance_Level plays an important role in the final quiz scores, and adjusting for it helped in understanding how other factors (like content-based filtering and interactivity) affected learning outcomes.
  - **Improvement_Score**: **Performance_Level** shows a non-significant effect on Improvement_Score (p-value > 0.05). This indicates that adjusting for baseline performance didn't significantly change how improvement scores were influenced by content-based filtering and interactivity.
  - **Study_Time**: Similarly, **Performance_Level** has a non-significant effect on Study_Time (p-value > 0.05), suggesting that initial performance doesn't influence study time as much as other factors in the study.
  - **Retention_Rate**: For **Retention_Rate**, Performance_Level also has a non-significant effect (p-value = 0.879), indicating that differences in baseline performance don't meaningfully influence retention rates after adjusting for the treatment conditions.

Conclusion:

- **Content-based filtering** and **interactivity** independently affect learning outcomes in most cases, as seen in significant F-statistics for Post_Test_Quiz_Score and Improvement_Score.
- The **interaction effect** between content-based filtering and interactivity is significant only for **Retention_Rate**, not for other metrics like quiz scores and improvement.
- The **Performance_Level** (as a blocking variable) influences **Post_Test_Quiz_Score** significantly, showing that baseline performance plays a role in the final quiz scores, but it does not have a significant effect on **Study_Time** and **Retention_Rate**.
- The interaction of content-based filtering and interactivity on learning outcomes is not consistently significant, which suggests that while both interventions individually improve learning outcomes, their combined effect may not be as strong as expected.

4.2.3 Factorial RCT Randomization (Continuous Covariates)

In [238…

```
# Step 1: Collect Baseline Measures (Pre-Intervention Data)
factorial_rct_data = generate_baseline_data(factorial_rct_num_students)
```

```
# Step 2: Define Covariates for Matching
covariates = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']

# Step 3: Perform Covariate Matching
# Initialize treatment assignment columns
factorial_rct_data['Algorithm_Type'] = None
factorial_rct_data['Interactivity_Level'] = None

# Randomly assign treatments to half of the participants
np.random.seed(18)
treatment_indices = np.random.choice(factorial_rct_data.index, size=len(factorial_rct_data) // 2, replace=False

# Assign treatments to the selected participants
factorial_rct_data.loc[treatment_indices, 'Algorithm_Type'] = np.random.choice(
    ['Content-based Filtering', 'No Content-based Filtering'], len(treatment_indices)
)
factorial_rct_data.loc[treatment_indices, 'Interactivity_Level'] = np.random.choice(
    ['Interactivity', 'No Interactivity'], len(treatment_indices)
)

# Use Nearest Neighbors to match participants based on covariates
# Fit NearestNeighbors on the treatment group
nbrs = NearestNeighbors(n_neighbors=1).fit(factorial_rct_data.loc[treatment_indices, covariates])

# Assign treatments to the remaining participants based on covariate matching
for idx in factorial_rct_data.index:
    if idx not in treatment_indices:
        # Find the nearest neighbor in the treatment group
        distances, neighbor_idx = nbrs.kneighbors(factorial_rct_data.loc[idx, covariates].values.reshape(1, -1)

        # Map the neighbor index back to the original treatment_indices
        matched_idx = treatment_indices[neighbor_idx[0][0]]

        # Assign the same treatment as the matched neighbor
        factorial_rct_data.loc[idx, 'Algorithm_Type'] = factorial_rct_data.loc[matched_idx, 'Algorithm_Type']
        factorial_rct_data.loc[idx, 'Interactivity_Level'] = factorial_rct_data.loc[matched_idx, 'Interactivity_

# Step 4: Create Combined Treatment Groups (Factorial Design)
factorial_rct_data['RCT_Factorial_Group'] = (
    factorial_rct_data['Algorithm_Type'] + ' + ' + factorial_rct_data['Interactivity_Level']
)

# Drop the temporary 'Treatment_Group' column (if it exists)
factorial_rct_data.drop(columns=['Treatment_Group'], inplace=True, errors='ignore')
```

4.2.4: Factorial RCT Implement Intervention (Continuous Covariates)

In [239…

```
# Step 3: Implement Intervention (Post-Intervention Data)

np.random.seed(18)

# Post_Test_Quiz_Score: Knowledge-based system boosts quiz scores, High Interactivity boosts quiz scores
factorial_rct_data['Post_Test_Quiz_Score'] = factorial_rct_data['Baseline_Quiz_Score'] + \
                                             np.where(factorial_rct_data['Algorithm_Type'] == 'Content-based
                                                      np.random.normal(10, 5, factorial_rct_num_students), #
                                                      np.random.normal(0, 3, factorial_rct_num_students))  #

factorial_rct_data['Post_Test_Quiz_Score'] += np.where(factorial_rct_data['Interactivity_Level'] == 'Interactiv
                                                      np.random.normal(10, 5, factorial_rct_num_students),  #
                                                      np.random.normal(-5, 5, factorial_rct_num_students))  #

# Improvement_Score: Difference between Post-Test and Baseline Quiz Scores
factorial_rct_data['Improvement_Score'] = (factorial_rct_data['Post_Test_Quiz_Score'] - factorial_rct_data['Base

# Study_Time: Content-based system boosts study time, High Interactivity boosts study time
factorial_rct_data['Study_Time'] = np.where(factorial_rct_data['Algorithm_Type'] == 'Content-based Filtering',
                                            np.random.normal(100, 50, factorial_rct_num_students),  # Boost for
                                            np.random.normal(0, 50, factorial_rct_num_students))  # No change f
factorial_rct_data['Study_Time'] += np.where(factorial_rct_data['Interactivity_Level'] == 'Interactivity',
                                             np.random.normal(100, 50, factorial_rct_num_students),  # Boost f
                                             np.random.normal(-50, 50, factorial_rct_num_students))  # Decreas

# Retention_Rate: Content-based system boosts retention rate, High Interactivity boosts retention rate
factorial_rct_data['Retention_Rate'] = np.where(factorial_rct_data['Algorithm_Type'] == 'Content-based Filterin
                                                np.random.normal(5, 2, factorial_rct_num_students),  # Boost fo
                                                np.random.normal(0, 2, factorial_rct_num_students))  # No chang
factorial_rct_data['Retention_Rate'] += np.where(factorial_rct_data['Interactivity_Level'] == 'Interactivity',
                                                 np.random.normal(10, 3, factorial_rct_num_students),  # Boost
                                                 np.random.normal(-5, 3, factorial_rct_num_students))  # Decre

# # Dropout_Rate: Content-based system reduces dropout rate, High Interactivity reduces dropout rate
```

```
factorial_rct_data['Dropout_Rate'] = 0  # Initialize the Dropout Rate column

# Assign dropout rates based on the combination of Algorithm_Type and Interactivity_Level
for index, row in factorial_rct_data.iterrows():
    if row['Algorithm_Type'] == 'Content-based Filtering' and row['Interactivity_Level'] == 'Interactivity':
        dropout_rate = np.random.choice([0, 1], p=[0.95, 0.05])  # 5% dropout rate for Content-based + Interact
    elif row['Algorithm_Type'] == 'Content-based Filtering' and row['Interactivity_Level'] == 'No Interactivity
        dropout_rate = np.random.choice([0, 1], p=[0.90, 0.10])  # 10% dropout rate for Content-based + No Inte
    elif row['Algorithm_Type'] == 'No Content-based Filtering' and row['Interactivity_Level'] == 'Interactivity
        dropout_rate = np.random.choice([0, 1], p=[0.92, 0.08])  # 8% dropout rate for No Content-based + Intera
    else:  # No Content-based Filtering + No Interactivity
        dropout_rate = np.random.choice([0, 1], p=[0.85, 0.15])  # 15% dropout rate for No Content-based + No I

    # Assign the final dropout rate
    factorial_rct_data.at[index, 'Dropout_Rate'] = dropout_rate



# Ensure that dropout rate is between 0 and 1 (percentage)
factorial_rct_data['Dropout_Rate'] = np.clip(factorial_rct_data['Dropout_Rate'], 0, 1)
```

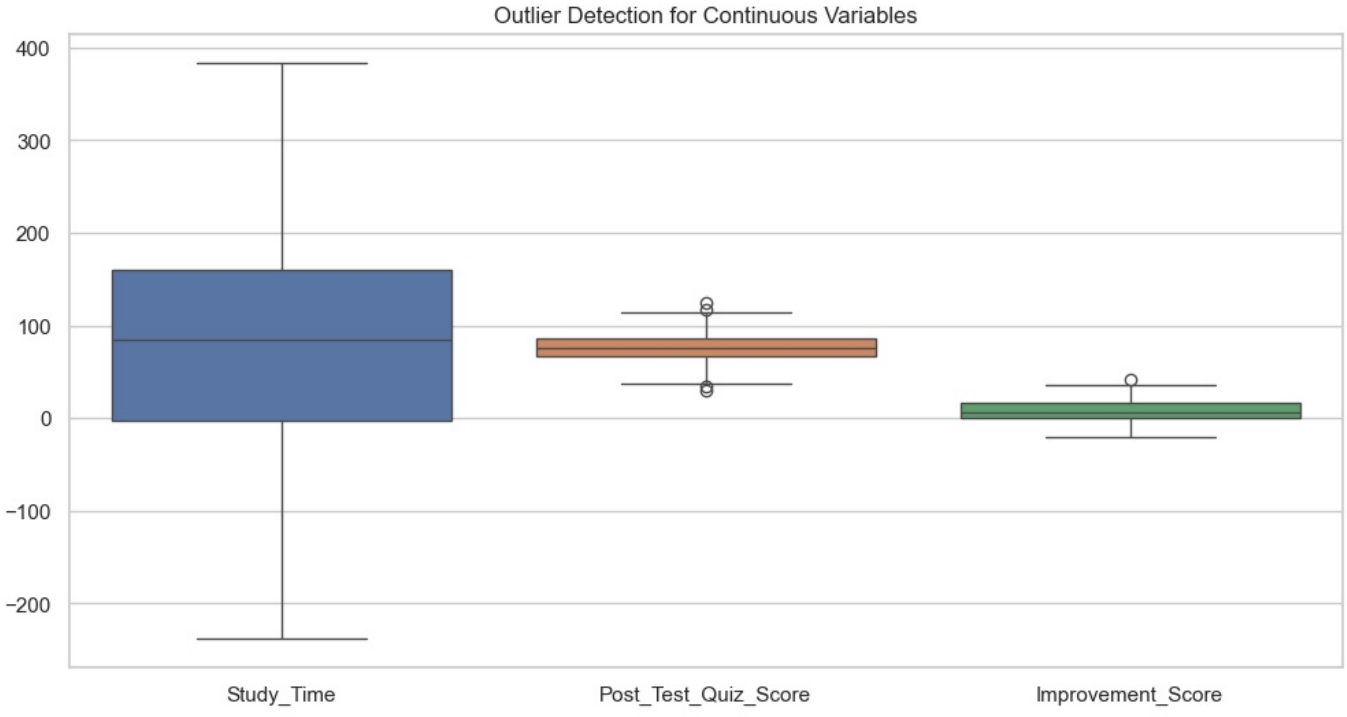
4.2.5: Factorial RCT Data Cleaning (Continuous Covariates)

In [240…

```
# 1. Check for Missing Data

print("Missing Values Check:\n")
print(factorial_rct_data.isnull().sum())

# Replace placeholder codes (e.g., -18) with NaN if present
factorial_rct_data.replace(-18, np.nan, inplace=True)

# Drop rows with critical missing values (if any)
critical_cols = ['Post_Test_Quiz_Score', 'Retention_Rate', 'Dropout_Rate' ,'RCT_Factorial_Group']
factorial_rct_data.dropna(subset=critical_cols, inplace=True)


# 2. Remove Duplicates

print(f"\nInitial Shape: {factorial_rct_data.shape}")
factorial_rct_data.drop_duplicates(subset=['Student_ID'], keep='first', inplace=True)
print(f"Post-Deduplication Shape: {factorial_rct_data.shape}")

# 3. Verify Randomization Balance

print("\nGroup Balance:")
print(factorial_rct_data['RCT_Factorial_Group'].value_counts())

# Compare baseline covariates between groups using ANOVA
print("\nBaseline Covariate Balance:")
baseline_cols = ['GPA', 'Baseline_Quiz_Score', 'Study_Hours_Per_Week']
for col in baseline_cols:
    # Group data by RCT_Factorial_Group
    groups = [factorial_rct_data[factorial_rct_data['RCT_Factorial_Group'] == group][col]
              for group in factorial_rct_data['RCT_Factorial_Group'].unique()]

    # Perform one-way ANOVA test
    f_stat, p_value = f_oneway(*groups)
    print(f"{col}: F = {f_stat:.2f}, p = {p_value:.4f}")


# 4. Detect and Handle Outliers

continuous_vars = ['Study_Time', 'Post_Test_Quiz_Score', 'Improvement_Score']

plt.figure(figsize=(12, 6))
sns.boxplot(data=factorial_rct_data[continuous_vars])
plt.title("Outlier Detection for Continuous Variables")
plt.show()

# Cap outliers using IQR
for var in continuous_vars:
    q1 = factorial_rct_data[var].quantile(0.25)
    q3 = factorial_rct_data[var].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    factorial_rct_data[var] = np.where(factorial_rct_data[var] < lower_bound, lower_bound,
                                       np.where(factorial_rct_data[var] > upper_bound, upper_bound,
                                       factorial_rct_data[var]))


# 5. Validate Variable Ranges/Types
```

```python
# Ensure binary variables are 0/1
factorial_rct_data['Dropout_Rate'] = factorial_rct_data['Dropout_Rate'].apply(lambda x: 1 if x == 1 else 0)

# Standardize categorical variables
factorial_rct_data['Learning_Style'] = factorial_rct_data['Learning_Style'].str.strip().str.title()

# Ensure valid percentage ranges
factorial_rct_data['Retention_Rate'] = factorial_rct_data['Retention_Rate'].clip(0, 100)

# 6. Save Cleaned Data
clean_factorial_rct_data = factorial_rct_data.copy()
print("\nData cleaning complete. Cleaned data saved as clean_factorial_rct_data.")
```

```
Missing Values Check:

Student_ID              0
Age                     0
GPA                     0
Study_Hours_Per_Week    0
Tech_Savviness_Score    0
Learning_Style          0
Baseline_Quiz_Score     0
Attention_Span          0
Motivation_Level        0
Prior_Online_Exp        0
Algorithm_Type          0
Interactivity_Level     0
RCT_Factorial_Group     0
Post_Test_Quiz_Score    0
Improvement_Score       0
Study_Time              0
Retention_Rate          0
Dropout_Rate            0
dtype: int64

Initial Shape: (793, 18)
Post-Deduplication Shape: (793, 18)

Group Balance:
RCT_Factorial_Group
Content-based Filtering + Interactivity          216
No Content-based Filtering + No Interactivity    201
Content-based Filtering + No Interactivity       200
No Content-based Filtering + Interactivity       176
Name: count, dtype: int64

Baseline Covariate Balance:
GPA: F = 2.35, p = 0.0715
Baseline_Quiz_Score: F = 5.67, p = 0.0008
Study_Hours_Per_Week: F = 0.90, p = 0.4418
```




```
Data cleaning complete. Cleaned data saved as clean_factorial_rct_data.
```

4.2.6: Factorial RCT Perform Analysis (Continuous Covariates)

In [241…

```
# Perform ANCOVA (Factorial ANOVA with covariates) for each metric
anova_results = {}
post_hoc_results = {}

for metric in metrics:
    # Define the formula for the ANCOVA model
    formula = f'{metric} ~ C(Algorithm_Type) * C(Interactivity_Level) + ' + ' + '.join(covariates)

    # Fit the ANCOVA model
    model = ols(formula, data=clean_factorial_rct_data).fit()

    # Perform Type 2 ANOVA
    anova_table = sm.stats.anova_lm(model, typ=2)

    # Extract F-statistic and p-value for the main effects and interactions
    anova_results[metric] = {
        'F-Statistic (Algorithm_Type)': anova_table.loc['C(Algorithm_Type)', 'F'],
        'p-value (Algorithm_Type)': anova_table.loc['C(Algorithm_Type)', 'PR(>F)'],
        'F-Statistic (Interactivity_Level)': anova_table.loc['C(Interactivity_Level)', 'F'],
        'p-value (Interactivity_Level)': anova_table.loc['C(Interactivity_Level)', 'PR(>F)'],
        'F-Statistic (Algorithm_Type:Interactivity_Level)': anova_table.loc['C(Algorithm_Type):C(Interactivity_
        'p-value (Algorithm_Type:Interactivity_Level)': anova_table.loc['C(Algorithm_Type):C(Interactivity_Leve
        **{f'F-Statistic ({cov})': anova_table.loc[cov, 'F'] for cov in covariates},
        **{f'p-value ({cov})': anova_table.loc[cov, 'PR(>F)'] for cov in covariates}
    }

    # Perform post-hoc analysis (Tukey's HSD) if there are significant interactions or main effects
    if anova_table.loc['C(Algorithm_Type):C(Interactivity_Level)', 'PR(>F)'] < 0.05:
        # Create a combined group column for post-hoc analysis
        clean_factorial_rct_data['Group'] = (
            clean_factorial_rct_data['Algorithm_Type'].astype(str) + "_" +
            clean_factorial_rct_data['Interactivity_Level'].astype(str)
        )

        # Perform Tukey's HSD test
        post_hoc = pairwise_tukeyhsd(endog=clean_factorial_rct_data[metric], groups=clean_factorial_rct_data['G
        post_hoc_results[metric] = post_hoc.summary()

# Convert ANOVA results to DataFrame
anova_results_df = pd.DataFrame(anova_results).T

# Add significance columns for each effect
for col in anova_results_df.columns:
    if 'p-value' in col:
        anova_results_df[f'Significance ({col.split(")")[0]})'] = anova_results_df[col].apply(
            lambda x: 'Significant' if x < 0.05 else 'Not Significant'
        )

# Mean comparisons for effectiveness by Algorithm_Type and Interactivity_Level
mean_comparison = clean_factorial_rct_data.groupby(['Algorithm_Type', 'Interactivity_Level'])[metrics].mean()

# Round the 'Retention_Rate' column to 2 decimal places
mean_comparison['Retention_Rate'] = mean_comparison['Retention_Rate'].round(2)

# Transpose mean_comparison so that metrics are rows and combinations are columns
mean_comparison = mean_comparison.T

mean_comparison.columns = ['CBF_I', 'CBF_NI', 'NCBS_I', 'NCBS_NI']

# Identify the better group for each metric based on the highest value
mean_comparison['Better Group'] = mean_comparison.apply(
    lambda row: 'CBF_NI' if row['CBF_NI'] > max(row['CBF_I'], row['NCBS_NI'], row['NCBS_I']) else
                ('CBF_I' if row['CBF_I'] > max(row['CBF_NI'], row['NCBS_NI'], row['NCBS_I']) else
                 ('NCBS_NI' if row['NCBS_NI'] > max(row['CBF_NI'], row['CBF_I'], row['NCBS_I']) else 'NCBS_I'))

# Merge ANOVA results with mean comparisons
result_table = mean_comparison.merge(anova_results_df, left_index=True, right_index=True)

# Transpose the result_table so that metrics are columns and effects/combinations are rows
result_table_transposed = result_table.T

# Display the transposed result table
print("4.2.6: Factorial RCT Perform Analysis - Comparison of Factorial Groups with Covariates (Transposed)\n")
display(result_table_transposed)
```

```
4.2.6: Factorial RCT Perform Analysis - Comparison of Factorial Groups with Covariates (Transposed)
```

	Post_Test_Quiz_Score	Improvement_Score	Study_Time	Retention_Rate
CBF_I	88.030762	20.388889	197.881116	15.08
CBF_NI	74.996888	3.63	53.541843	1.27
NCBS_I	80.624225	10.511364	102.462267	10.05
NCBS_NI	64.238749	-4.338308	-49.892353	0.11
Better Group	CBF_I	CBF_I	CBF_I	CBF_I
F-Statistic (Algorithm_Type)	491.268667	485.721392	419.691392	239.329076
p-value (Algorithm_Type)	0.0	0.0	0.0	0.0
F-Statistic (Interactivity_Level)	1443.609708	1418.579345	903.995548	3599.646539
p-value (Interactivity_Level)	0.0	0.0	0.0	0.0
F-Statistic (Algorithm_Type:Interactivity_Level)	1.208147	3.105322	1.081166	95.473649
p-value (Algorithm_Type:Interactivity_Level)	0.272036	0.078425	0.298757	0.0
F-Statistic (GPA)	26.368142	25.367346	2.153324	0.003626
F-Statistic (Baseline_Quiz_Score)	2197.675239	0.001675	2.365167	0.09606
F-Statistic (Tech_Savviness_Score)	114.568017	111.284236	0.322287	0.060194
p-value (GPA)	0.0	0.000001	0.142662	0.951997
p-value (Baseline_Quiz_Score)	0.0	0.967367	0.124473	0.756692
p-value (Tech_Savviness_Score)	0.0	0.0	0.570398	0.806253
Significance (p-value (Algorithm_Type)	Significant	Significant	Significant	Significant
Significance (p-value (Interactivity_Level)	Significant	Significant	Significant	Significant
Significance (p-value (Algorithm_Type:Interactivity_Level)	Not Significant	Not Significant	Not Significant	Significant
Significance (p-value (GPA)	Significant	Significant	Not Significant	Not Significant
Significance (p-value (Baseline_Quiz_Score)	Significant	Not Significant	Not Significant	Not Significant
Significance (p-value (Tech_Savviness_Score)	Significant	Significant	Not Significant	Not Significant

In [242…

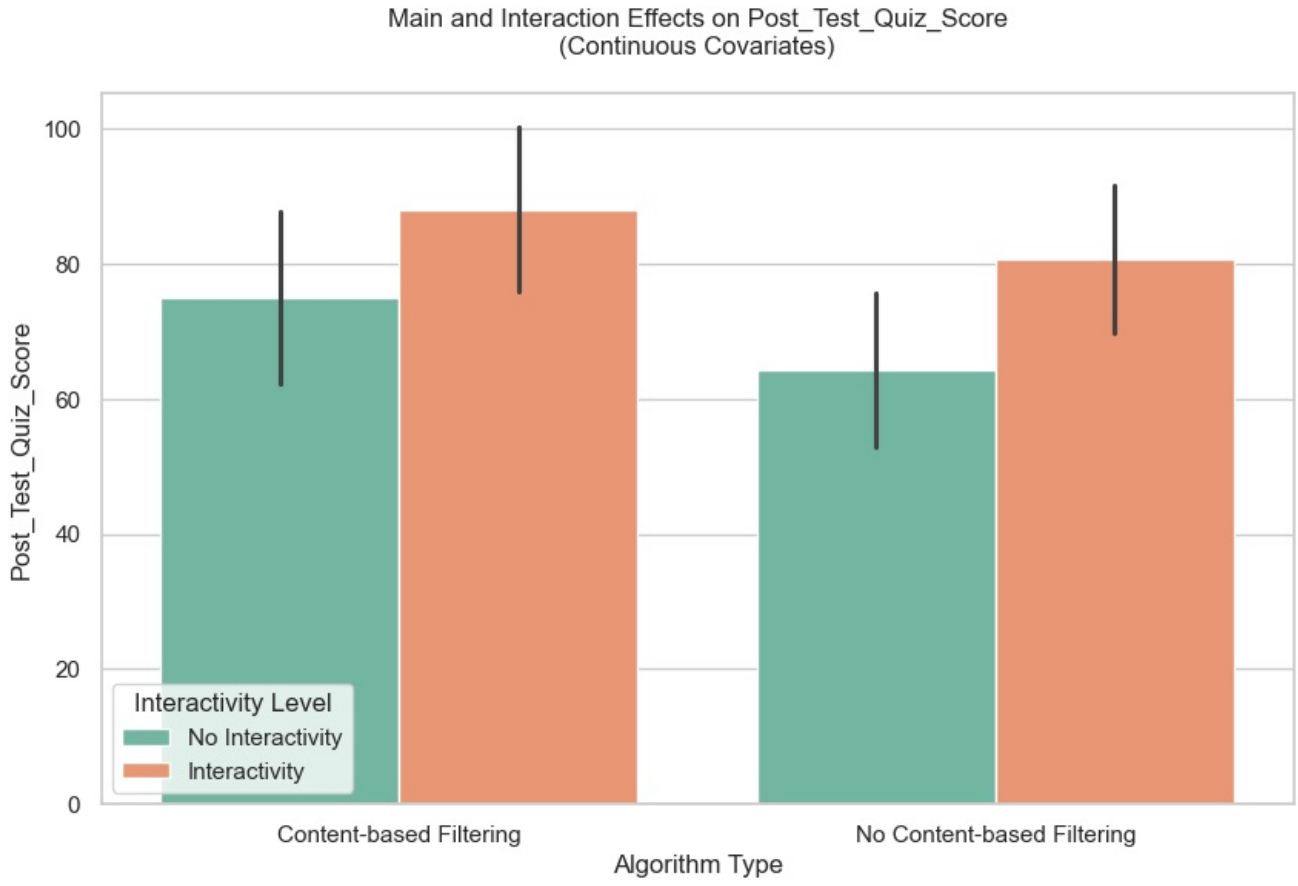
```
# Display post-hoc results for each metric
print("\nPost-Hoc Analysis Results (Tukey's HSD):\n")
for metric, result in post_hoc_results.items():
    print(f"Post-Hoc Results for {metric}:")
    print(result)
    print("\n")
```

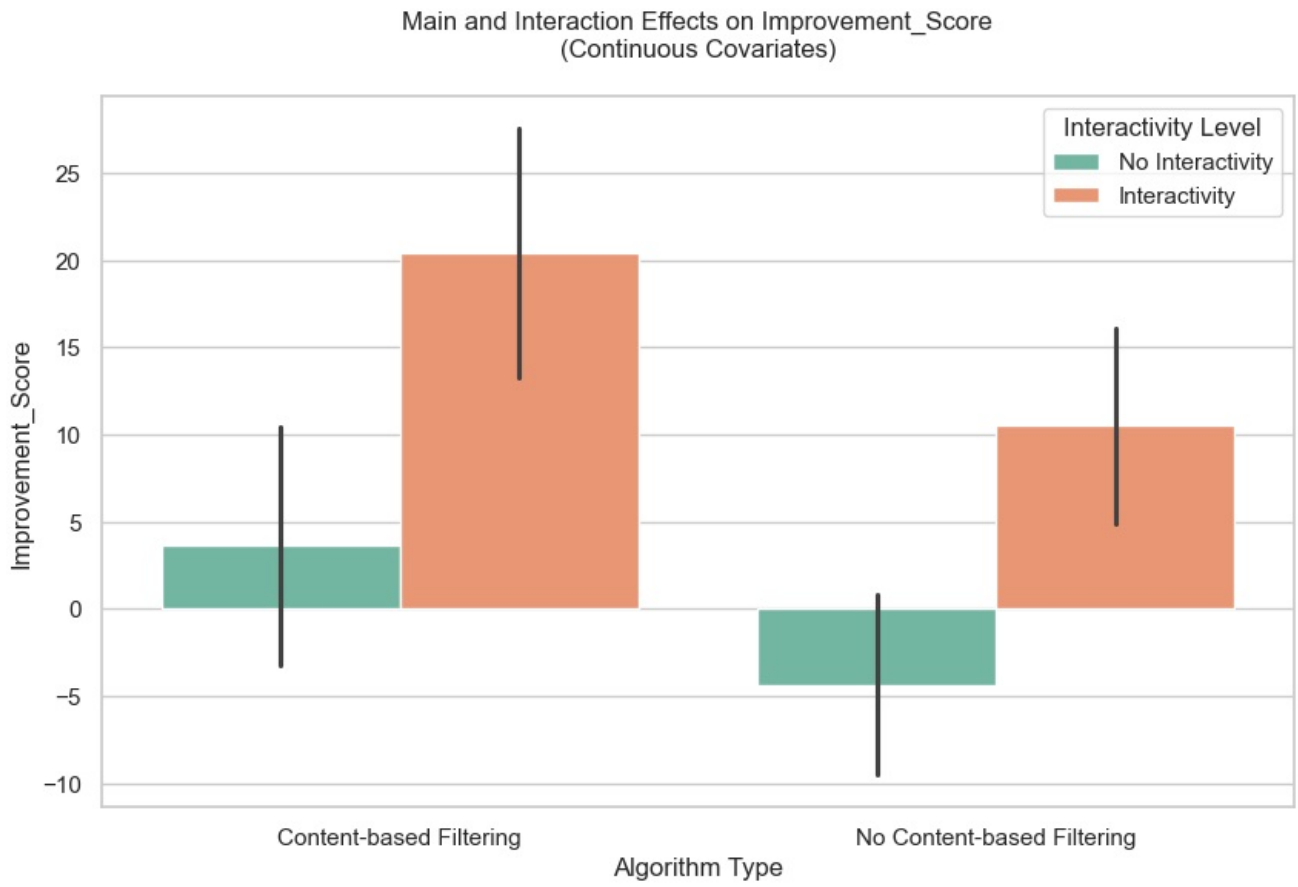
```
Post-Hoc Analysis Results (Tukey's HSD):

Post-Hoc Results for Retention_Rate:
                                  Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================================================================
=============
                group1                                    group2                  meandiff p-adj   lower    u
pper   reject
-----------------------------------------------------------------------------------------------------------------------------
-------------
   Content-based Filtering_Interactivity    Content-based Filtering_No Interactivity -13.8115    0.0 -14.5121 -1
3.1108   True
   Content-based Filtering_Interactivity    No Content-based Filtering_Interactivity  -5.0354    0.0  -5.7604  -
4.3103   True
   Content-based Filtering_Interactivity No Content-based Filtering_No Interactivity -14.9664    0.0 -15.6662 -1
4.2667   True
Content-based Filtering_No Interactivity    No Content-based Filtering_Interactivity   8.7761    0.0   8.0381
9.514   True
Content-based Filtering_No Interactivity No Content-based Filtering_No Interactivity   -1.155 0.0002  -1.8681  -
0.4419   True
No Content-based Filtering_Interactivity No Content-based Filtering_No Interactivity   -9.931    0.0 -10.6681
-9.194   True
-----------------------------------------------------------------------------------------------------------------------------
-------------
```

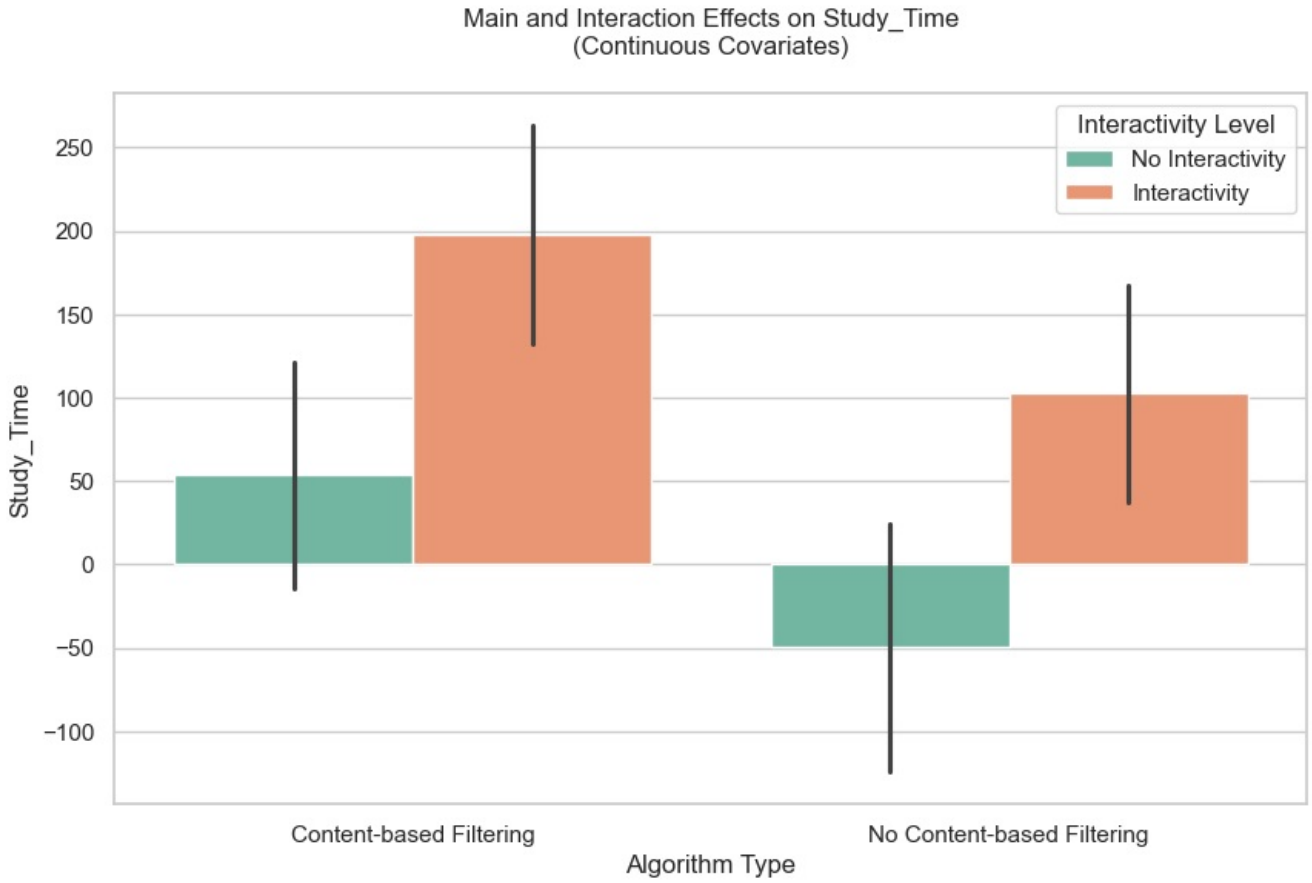
In [243…

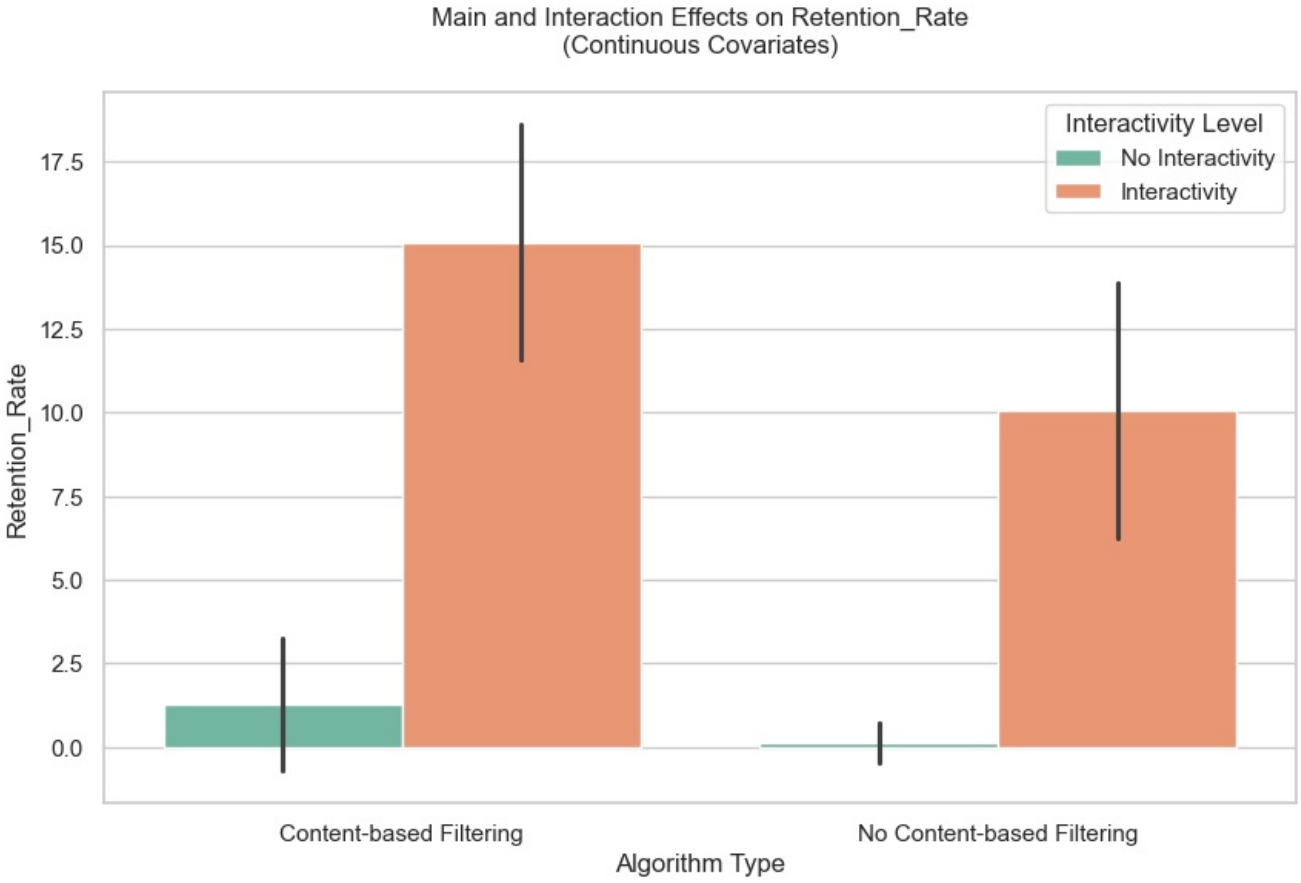
```
# Melt the data for easier plotting
melted_data = clean_factorial_rct_data.melt(
    id_vars=['Algorithm_Type', 'Interactivity_Level', 'GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score'],
    value_vars=metrics,
```

```
    var_name='Metric',
    value_name='Score'
)

# Plot main effects and interaction effects
for metric in metrics:
    plt.figure(figsize=(10, 6))
    sns.barplot(
        x='Algorithm_Type',
        y='Score',
        hue='Interactivity_Level',
        data=melted_data[melted_data['Metric'] == metric],
        ci='sd',  # Show standard deviation
        palette='Set2'
    )
    plt.title(f'Main and Interaction Effects on {metric}\n(Continuous Covariates)\n')
    plt.xlabel('Algorithm Type')
    plt.ylabel(metric)
    plt.legend(title='Interactivity Level')
    plt.show()
```




Main and Interaction Effects on Improvement_Score
(Continuous Covariates)
Interactivity Level
No Interactivity
Interactivity
Improvement_Score
25
20
15
10
5
0
−5
−10
Content-based Filtering
No Content-based Filtering
Algorithm Type

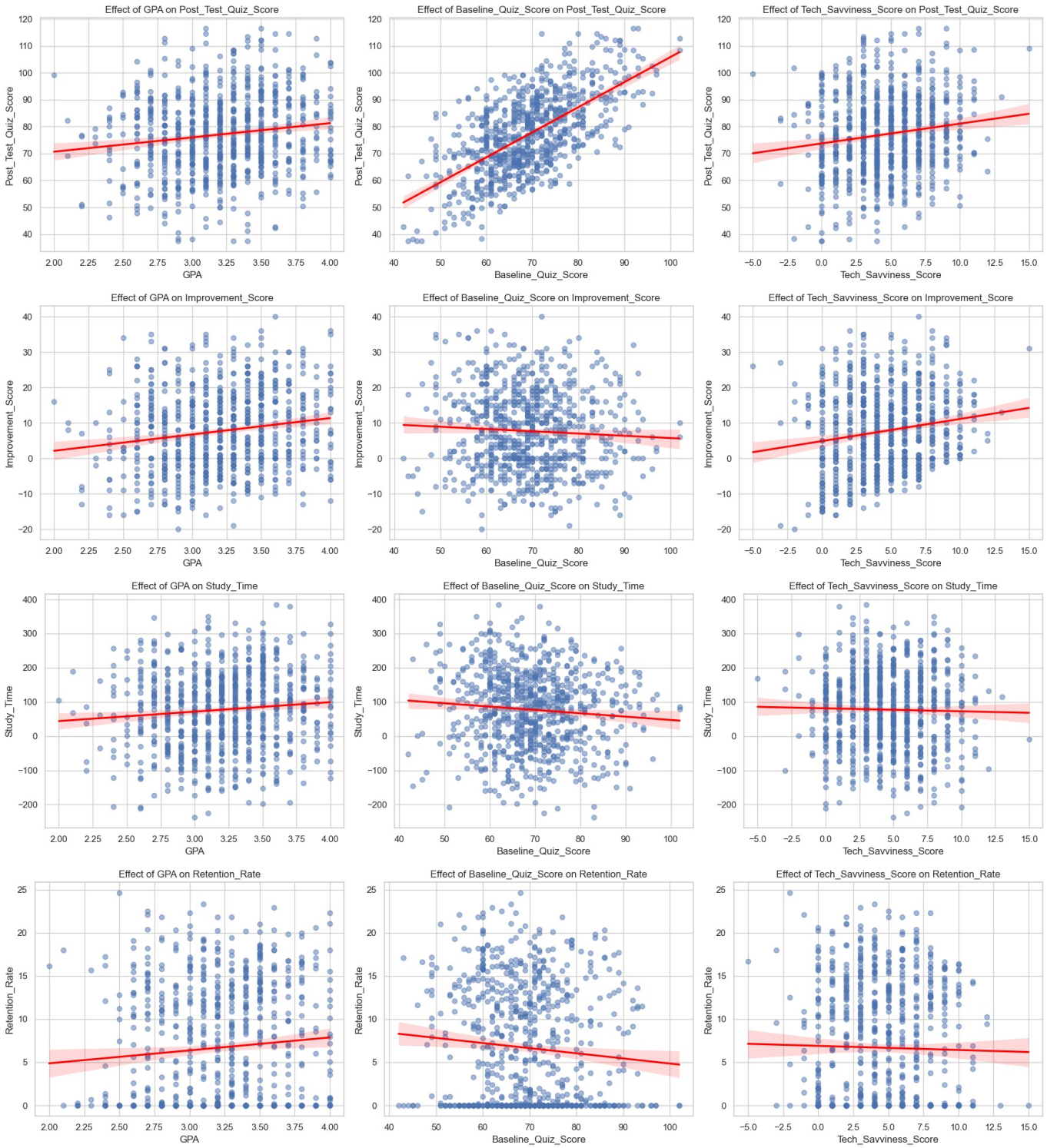

Main and Interaction Effects on Study_Time
(Continuous Covariates)
Interactivity Level
No Interactivity
Interactivity
Study_Time
250
200
150
100
50
0
−50
−100
Content-based Filtering
No Content-based Filtering
Algorithm Type



In [244…

```python
# Define the number of rows based on the number of covariates
for metric in metrics:
    num_covariates = len(covariates)
    fig, axes = plt.subplots(nrows=1, ncols=num_covariates, figsize=(6 * num_covariates, 5))

    # If there's only one covariate, axes won't be a list
    if num_covariates == 1:
        axes = [axes]

    for ax, cov in zip(axes, covariates):
        sns.regplot(
            x=cov,
            y=metric,
            data=clean_factorial_rct_data,
            scatter_kws={'alpha': 0.5},
            line_kws={'color': 'red'},
            ax=ax
        )
        ax.set_title(f'Effect of {cov} on {metric}', fontsize=12)
```

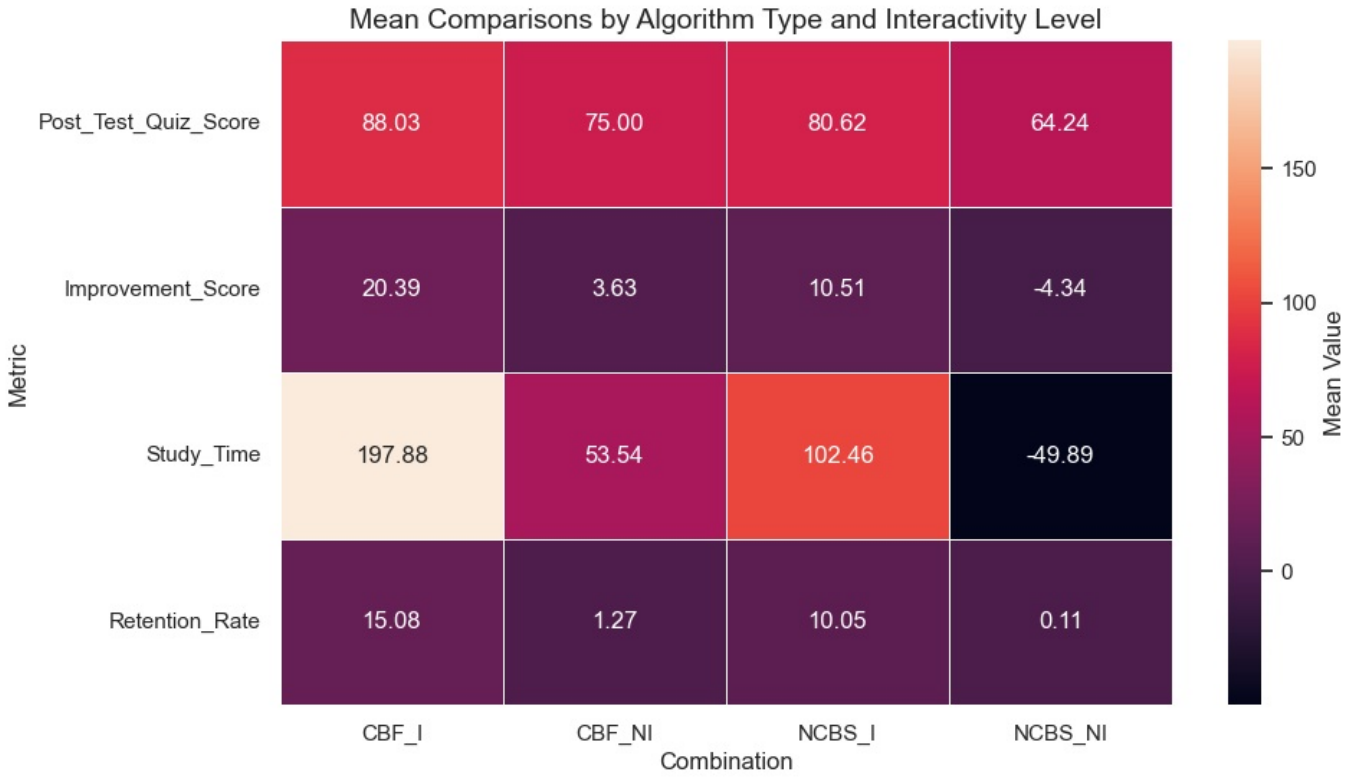
```
        ax.set_xlabel(cov)
        ax.set_ylabel(metric)

    # Adjust spacing
    plt.tight_layout()
    plt.show()
```



In [245…

```
# Drop the 'Better Group' column for the heatmap
heatmap_data = mean_comparison.drop(columns=['Better Group'])

# Create the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(
    heatmap_data,
    annot=True,  # Display values in each cell
    fmt='.2f',  # Format values to 2 decimal places
    linewidths=0.5,  # Add lines between cells
    cbar_kws={'label': 'Mean Value'}  # Add a label to the color bar
)

# Add titles and labels
plt.title('Mean Comparisons by Algorithm Type and Interactivity Level', fontsize=14)
plt.xlabel('Combination', fontsize=12)
plt.ylabel('Metric', fontsize=12)
```

```
# Show the plot
plt.show()
```



In [246…
```
# Step 1: Define the logistic regression formula
formula = 'Dropout_Rate ~ C(RCT_Factorial_Group) + GPA + Baseline_Quiz_Score + Tech_Savviness_Score'

# Step 2: Fit the logistic regression model
logit_model = logit(formula, data=clean_factorial_rct_data).fit()

# Step 3: Extract model results
logit_results = {
    'Metric': 'Dropout Rate',
    'CBF_I': clean_factorial_rct_data[clean_factorial_rct_data['RCT_Factorial_Group'] == 'Content-based Filteri
    'CBF_NI': clean_factorial_rct_data[clean_factorial_rct_data['RCT_Factorial_Group'] == 'Content-based Filter
    'NCBS_I': clean_factorial_rct_data[clean_factorial_rct_data['RCT_Factorial_Group'] == 'No Content-based Fil
    'NCBS_NI': clean_factorial_rct_data[clean_factorial_rct_data['RCT_Factorial_Group'] == 'No Content-based Fi
    'Better Group': clean_factorial_rct_data.groupby('RCT_Factorial_Group')['Dropout_Rate'].mean().idxmin(),
    'Coefficient (RCT_Factorial_Group)': logit_model.params['C(RCT_Factorial_Group)[T.Content-based Filtering +
    'p-value (RCT_Factorial_Group)': logit_model.pvalues['C(RCT_Factorial_Group)[T.Content-based Filtering + No
    'Coefficient (GPA)': logit_model.params['GPA'],
    'p-value (GPA)': logit_model.pvalues['GPA'],
    'Coefficient (Baseline_Quiz_Score)': logit_model.params['Baseline_Quiz_Score'],
    'p-value (Baseline_Quiz_Score)': logit_model.pvalues['Baseline_Quiz_Score'],
    'Coefficient (Tech_Savviness_Score)': logit_model.params['Tech_Savviness_Score'],
    'p-value (Tech_Savviness_Score)': logit_model.pvalues['Tech_Savviness_Score'],
    'Significance (RCT_Factorial_Group)': 'Significant' if logit_model.pvalues['C(RCT_Factorial_Group)[T.Conten
    'Significance (GPA)': 'Significant' if logit_model.pvalues['GPA'] < 0.05 else 'Not Significant',
    'Significance (Baseline_Quiz_Score)': 'Significant' if logit_model.pvalues['Baseline_Quiz_Score'] < 0.05 el
    'Significance (Tech_Savviness_Score)': 'Significant' if logit_model.pvalues['Tech_Savviness_Score'] < 0.05
}

# Step 4: Convert results to DataFrame
dropout_results_df = pd.DataFrame(logit_results, index=[0]).T

print("\n4.2.6: Factorial RCT Logistic Regression Analysis of Dropout Rate with Covariates\n")
display(dropout_results_df)
```

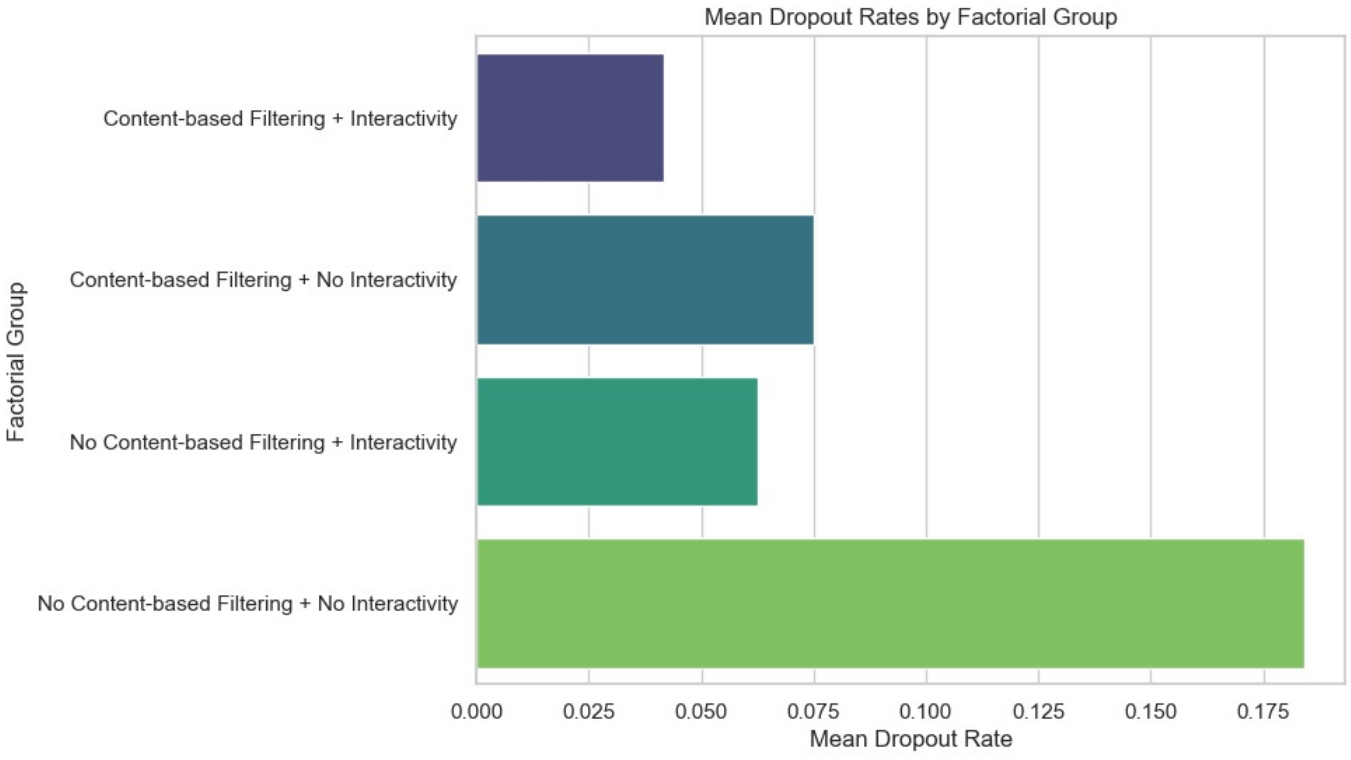
```
Optimization terminated successfully.
         Current function value: 0.287048
         Iterations 7
```

4.2.6: Factorial RCT Logistic Regression Analysis of Dropout Rate with Covariates

	0
Metric	Dropout Rate
CBF_I	0.041667
CBF_NI	0.075
NCBS_I	0.0625
NCBS_NI	0.18408
Better Group	Content-based Filtering + Interactivity
Coefficient (RCT_Factorial_Group)	0.606157
p-value (RCT_Factorial_Group)	0.165842
Coefficient (GPA)	-0.121228
p-value (GPA)	0.705608
Coefficient (Baseline_Quiz_Score)	0.002842
p-value (Baseline_Quiz_Score)	0.821021
Coefficient (Tech_Savviness_Score)	0.018744
p-value (Tech_Savviness_Score)	0.667831
Significance (RCT_Factorial_Group)	Not Significant
Significance (GPA)	Not Significant
Significance (Baseline_Quiz_Score)	Not Significant
Significance (Tech_Savviness_Score)	Not Significant

In [247…

```
# Calculate mean dropout rates by factorial group
mean_dropout_rates = clean_factorial_rct_data.groupby('RCT_Factorial_Group')['Dropout_Rate'].mean().reset_index

# Plot mean dropout rates
plt.figure(figsize=(8, 6))
sns.barplot(
    x='Dropout_Rate',
    y='RCT_Factorial_Group',
    data=mean_dropout_rates,
    palette='viridis'
)
plt.title('Mean Dropout Rates by Factorial Group')
plt.xlabel('Mean Dropout Rate')
plt.ylabel('Factorial Group')
plt.show()
```



4.2.7 Interpretation of Factorial RCT Results (Continuous Covariates)

1. Factorial RCT Perform Analysis

1. **Algorithm Type**:
   - **p-value**: 0.0 for all metrics.
   - **Interpretation**: The choice of algorithm (adaptive vs. static) significantly impacts all learner outcomes.
2. **Interactivity Level**:
   - **p-value**: 0.0 for all metrics.
   - **Interpretation**: Higher interactivity levels significantly improve quiz scores, improvement, study time, and retention.
3. **Interaction (Algorithm Type × Interactivity Level)**:
   - **p-value**: Significant only for `Retention_Rate` (p = 0.0).
   - **Interpretation**: The combined effect of algorithm type and interactivity significantly influences retention but not other metrics.
4. **Continuous Covariates**:
   - **GPA**:
     - **p-value**: Significant for `Post_Test_Quiz_Score` and `Improvement_Score` (p < 0.05).
     - **Interpretation**: Higher GPA is associated with better quiz scores and improvement but does not affect study time or retention.
   - **Baseline_Quiz_Score**:
     - **p-value**: Significant only for `Post_Test_Quiz_Score` (p = 0.0).
     - **Interpretation**: Baseline quiz scores strongly predict post-test performance but do not influence other outcomes.
   - **Tech_Savviness_Score**:
     - **p-value**: Significant for `Post_Test_Quiz_Score` and `Improvement_Score` (p = 0.0).
     - **Interpretation**: Higher tech savviness improves quiz scores and improvement but does not affect study time or retention.
5. **Better Group**:
   - The `CBF_I` group (Content-based Filtering + Interactivity) performs best across all metrics.
   - **Interpretation**: Combining adaptive algorithms with high interactivity yields the best outcomes.

---

2. Logistic Regression Analysis of Dropout Rate

1. **RCT Factorial Group**:
   - **p-value**: 0.166 (p > 0.05).
   - **Interpretation**: The choice of factorial group does not significantly influence dropout rates.
2. **Continuous Covariates**:
   - **GPA**:
     - **p-value**: 0.738 (p > 0.05).
     - **Interpretation**: GPA does not significantly influence dropout rates.
   - **Baseline_Quiz_Score**:
     - **p-value**: 0.799 (p > 0.05).
     - **Interpretation**: Baseline quiz scores do not significantly influence dropout rates.
   - **Tech_Savviness_Score**:
     - **p-value**: 0.643 (p > 0.05).
     - **Interpretation**: Tech savviness does not significantly influence dropout rates.
3. **Better Group**:
   - The `CBF_I` group has the lowest dropout rate (0.0417).
   - **Interpretation**: Combining adaptive algorithms with high interactivity is associated with the lowest dropout rates.

---

3. Influence of Continuous Covariates

- **GPA**:
  - Significant for quiz scores and improvement (p < 0.05).
  - Not significant for study time, retention, or dropout rates (p > 0.05).
- **Baseline_Quiz_Score**:
  - Significant for quiz scores (p = 0.0).
  - Not significant for other metrics (p > 0.05).
- **Tech_Savviness_Score**:
  - Significant for quiz scores and improvement (p = 0.0).
  - Not significant for study time, retention, or dropout rates (p > 0.05).

---

4. p-Values Interpretation

- **p < 0.05**: The effect is statistically significant.

    - Example: `Algorithm_Type` and `Interactivity_Level` have p = 0.0 for all metrics.
- **p > 0.05**: The effect is not statistically significant.
    - Example: The interaction effect is not significant for most metrics (p > 0.05).

---

5. **Practical Implications**

1. **Algorithm and Interactivity**:
    - Use **adaptive algorithms** with **high interactivity** (`CBF_I`) to maximize performance and retention.
2. **Covariates**:
    - Focus on improving **GPA** and **tech savviness** to enhance quiz scores and improvement.
    - Baseline quiz scores are a strong predictor of post-test performance but do not influence other outcomes.
3. **Dropout Rates**:
    - The `CBF_I` group has the lowest dropout rate, making it the preferred choice for reducing attrition.

Conclusion

The **Factorial RCT Perform Analysis** reveals that:

- **Algorithm_Type** and **Interactivity_Level** are the most significant factors influencing learner performance and retention.
- The interaction between these factors significantly affects retention but not other metrics.
- Continuous covariates like **GPA** and **Tech_Savviness_Score** influence quiz scores and improvement but not study time, retention, or dropout rates.

The **Logistic Regression Analysis** shows that:

- None of the factors or covariates significantly influence dropout rates, except for the `CBF_I` group, which has the lowest dropout rate.

These findings suggest that combining **adaptive algorithms** with **high interactivity** is the most effective strategy for improving learner outcomes and reducing dropout rates.

---

5. Cross-Over RCT

Hypothesis

- **H_0**: There is no carryover effect (Phase 1 treatment does not influence Phase 2 outcomes).
- **H_1**: There is a carryover effect (Phase 1 treatment influences Phase 2 outcomes).

⭐ Expectation:

- The adaptive learning approach will consistently outperform the static learning approach in both the within-subject and between-subject comparisons. This should be reflected in improvements in key metrics such as post-test quiz scores, improvement scores, study time, and retention rates. We anticipate that the adaptive approach will not only show initial benefits but will also lead to greater long-term improvements in subsequent phases.

Treatment & Control Groups

- **Group A**: Uses adaptive learning for 4 weeks, then switches to static content.
- **Group B**: Uses static content for 4 weeks, then switches to adaptive learning.

Analysis Methods

1️⃣ Within-Subject Comparison (Paired t-test)

1. Run Shapiro-Wilk and Levene's test before t-test
    - Run Shapiro-Wilk test to check if data is normally distributed.
    - Run Equal variances test to check if data has equal variances across groups.
2. Use Paired t-test to compare the treatment group's performance before and after withdrawal.

- **Why paired t-test?**
    - Each participant is measured **twice** (once per phase).
    - The data points are **dependent** (same individuals measured twice).
    - More appropriate than an independent t-test, which assumes different participants.

3. Use Chi-Square Test for categorical variable (Dropout_Rate).
4. Compare the mean of each metric to evaluate which group has the better result.

2️⃣ Between-Subject Comparison (One-Way ANOVA)

1. Run Shapiro-Wilk and Levene's test before ANOVA
   - Run Shapiro-Wilk test to check if data is normally distributed.
   - Run Equal variances test to check if data has equal variances across groups.
2. Use One-Way ANOVA to compare the treatment group's performance before and after withdrawal.

- **Why One-Way ANOVA instead of a t-test?**
  - Allows for comparisons across **multiple groups** in future analyses.
  - Provides a more **generalizable** approach.

3. Use Chi-Square Test for categorical variable (Dropout_Rate).
4. Compare the mean of each metric to evaluate which group has the better result.

3 Carryover Effect Analysis (Two-Way ANOVA with Interaction Term)

1. Run Shapiro-Wilk and Levene's test before ANOVA
   - Run Shapiro-Wilk test to check if data is normally distributed.
   - Run Equal variances test to check if data has equal variances across groups.
2. Run **Two-Way ANOVA**

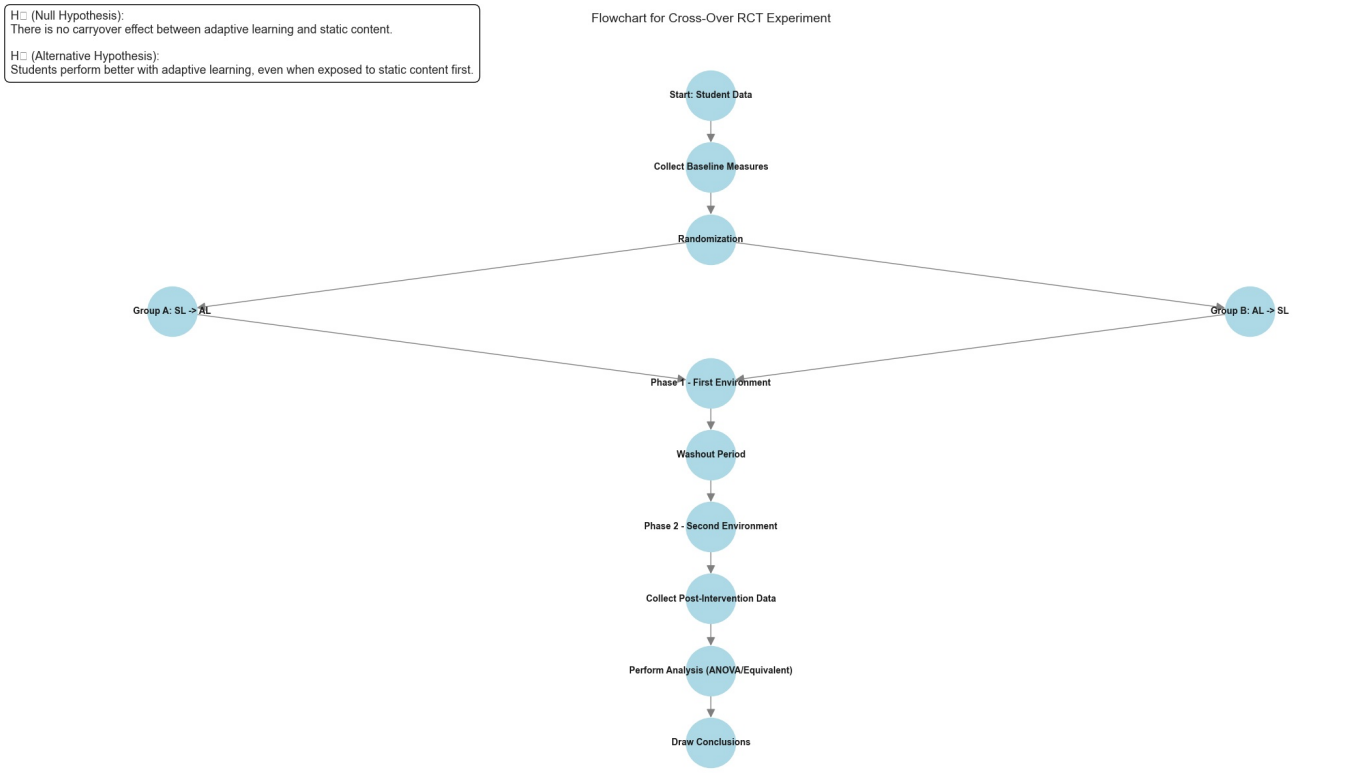
- Checks for a **carryover effect** (whether Phase 1 influences Phase 2).
- Uses an **interaction term (Group × Phase)** to test whether previous exposure affects learning outcomes.
- **Why Two-Way ANOVA?**
  - Crossover designs require examining whether prior learning style affects subsequent performance.
  - A significant **interaction effect** suggests a lingering impact from Phase 1, which could bias results.

5.1 Flowchart for Cross-Over RCT Experiment

In [248…

```
# Initialize the directed graph
G = nx.DiGraph()

# Define the nodes for the flowchart
nodes = [
    "Start: Student Data",
    "Collect Baseline Measures",
    "Randomization",
    "Group A: SL -> AL",
    "Group B: AL -> SL",
    "Phase 1 - First Environment",
    "Washout Period",
    "Phase 2 - Second Environment",
    "Collect Post-Intervention Data",
    "Perform Analysis (ANOVA/Equivalent)",
    "Draw Conclusions"
]

# Add the edges (connections between the nodes)
edges = [
    ("Start: Student Data", "Collect Baseline Measures"),
    ("Collect Baseline Measures", "Randomization"),
    ("Randomization", "Group A: SL -> AL"),
    ("Randomization", "Group B: AL -> SL"),
    ("Group A: SL -> AL", "Phase 1 - First Environment"),
    ("Group B: AL -> SL", "Phase 1 - First Environment"),
    ("Phase 1 - First Environment", "Washout Period"),
    ("Washout Period", "Phase 2 - Second Environment"),
    ("Phase 2 - Second Environment", "Collect Post-Intervention Data"),
    ("Collect Post-Intervention Data", "Perform Analysis (ANOVA/Equivalent)"),
    ("Perform Analysis (ANOVA/Equivalent)", "Draw Conclusions")
]

# Add nodes and edges to the graph
G.add_nodes_from(nodes)
G.add_edges_from(edges)

pos = {
    "Start: Student Data": (0, 4),
    "Collect Baseline Measures": (0, 3),
    "Randomization": (0, 2),
    "Group A: SL -> AL": (-1, 1),
    "Group B: AL -> SL": (1, 1),
    "Phase 1 - First Environment": (0, 0),
    "Washout Period": (0, -1),
    "Phase 2 - Second Environment": (0, -2),
    "Collect Post-Intervention Data": (0, -3),
    "Perform Analysis (ANOVA/Equivalent)": (0, -4),
    "Draw Conclusions": (0, -5)
}
```

```
plt.figure(figsize=(20, 12))
nx.draw(
    G,
    pos,
    with_labels=True,
    node_size=3000,
    node_color="lightblue",
    font_size=10,
    font_weight="bold",
    arrowsize=20,
    edge_color="gray"
)

plt.title("Flowchart for Cross-Over RCT Experiment", fontsize=14)

# Add hypothesis text
hypothesis_text = (
    "H₀ (Null Hypothesis):\nThere is no carryover effect between adaptive learning and static content.\n"
    "\nH₁ (Alternative Hypothesis):\nStudents perform better with adaptive learning, even when exposed to stati
)

# Position hypothesis text
plt.text(-1.3, 4.3, hypothesis_text, ha="left", fontsize=13, bbox=dict(facecolor="white", edgecolor="black", bo

plt.show()
```



5.2 Cross-Over RCT A-priori Power Analysis For Sample Size Determination

In [249…

```
from statsmodels.stats.power import TTestPower, FTestAnovaPower, GofChisquarePower

# Parameters
alpha = 0.05
power = 0.8
num_groups = 2  # Cross-Over RCT involves one group of participants, but we're looking at multiple comparisons

# 1. Continuous Variables (Paired t-test for within-subject design)

effect_sizes = {
    "Post_Test_Quiz_Score": 0.3,  # Cohen's d (medium effect)
    "Study_Time": 0.20,       # Cohen's d (small effect)
    "Retention_Rate": 0.25    # Cohen's d (medium effect)
}

# Initialize power analyzer for paired t-tests (within-subject comparisons)
ttest_power = TTestPower()
continuous_results = {}

for metric, d in effect_sizes.items():
    # Calculate per-group sample size for paired t-test (within-subject)
    n_per_group = ttest_power.solve_power(
```

```
        effect_size=d,
        alpha=alpha,
        power=power,
        alternative='two-sided'
    )
    continuous_results[metric] = {
        "Per Group": int(round(n_per_group)),
        "Total": int(round(n_per_group * num_groups))
    }

# 2. Between-Subject Comparison (One-Way ANOVA)

# Cohen's f for small, medium, and large effects in ANOVA
anova_effect_sizes = {
    "Post_Test_Quiz_Score": 0.25,  # Cohen's f (medium effect)
    "Study_Time": 0.2,             # Cohen's f (small effect)
    "Retention_Rate": 0.25         # Cohen's f (medium effect)
}

# Initialize power analyzer for ANOVA
anova_power = FTestAnovaPower()
anova_results = {}

for metric, f in anova_effect_sizes.items():
    # Calculate sample size for one-way ANOVA
    n_per_group_anova = anova_power.solve_power(
        effect_size=f,
        alpha=alpha,
        power=power,
        k_groups=num_groups
    )
    anova_results[metric] = {
        "Per Group": int(round(n_per_group_anova)),
        "Total": int(round(n_per_group_anova * num_groups))
    }

# 3. Carryover Effect Analysis (Two-Way ANOVA with Interaction Term)

# Cohen's f for two-way ANOVA with interaction term
carryover_effect_sizes = {
    "Post_Test_Quiz_Score": 0.25,  # Cohen's f (medium effect)
    "Study_Time": 0.2,             # Cohen's f (small effect)
    "Retention_Rate": 0.25         # Cohen's f (medium effect)
}

# Initialize power analyzer for two-way ANOVA
carryover_anova_power = FTestAnovaPower()
carryover_results = {}

for metric, f in carryover_effect_sizes.items():
    # Calculate sample size for two-way ANOVA with interaction term (Group × Phase)
    n_per_group_carryover = carryover_anova_power.solve_power(
        effect_size=f,
        alpha=alpha,
        power=power,
        k_groups=num_groups
    )
    carryover_results[metric] = {
        "Per Group": int(round(n_per_group_carryover)),
        "Total": int(round(n_per_group_carryover * num_groups))
    }

# 4. Dropout Rate (Binary) (Chi-Square Test of Independence)

# Using Cramér's V = 0.2 (small effect for 2x2 table)
chi2_power = GofChisquarePower()
total_dropout_n = chi2_power.solve_power(
    effect_size=0.2,  # Equivalent to Cramér's V for 2x2
    alpha=alpha,
    power=power,
    nobs=None
)

# 5. Final Sample Size Calculation

# Get the largest sample size requirement across all tests (paired t-test, ANOVA, and carryover analysis)
max_continuous = max([v["Total"] for v in continuous_results.values()])
max_anova = max([v["Total"] for v in anova_results.values()])
max_carryover = max([v["Total"] for v in carryover_results.values()])

# Final sample size will be the largest of the required sample sizes across tests
cross_over_rct_sample_size = max(max_continuous, max_anova, max_carryover, int(round(total_dropout_n)))
```

```
print(f"\nContinuous Variables Within-Subject Comparison (aired t-tests):")
for metric, res in continuous_results.items():
    print(f"- {metric}: {res['Per Group']} per group → {res['Total']} total")

print(f"\nContinuous Variables Between-Subject Comparison (One-Way ANOVA):")
for metric, res in anova_results.items():
    print(f"- {metric}: {res['Per Group']} per group → {res['Total']} total")

print(f"\nCarryover Effect Analysis (Two-Way ANOVA with Interaction Term):")
for metric, res in carryover_results.items():
    print(f"- {metric}: {res['Per Group']} per group → {res['Total']} total")

print(f"\nDropout Rate (Chi-Square Test for Cross-Over RCT): {int(round(total_dropout_n))} total students")
print(f"\nFinal Recommended Sample Size for Cross-Over RCT: {cross_over_rct_sample_size} students")
```

```
Continuous Variables Within-Subject Comparison (aired t-tests):
- Post_Test_Quiz_Score: 89 per group → 178 total
- Study_Time: 198 per group → 396 total
- Retention_Rate: 128 per group → 255 total

Continuous Variables Between-Subject Comparison (One-Way ANOVA):
- Post_Test_Quiz_Score: 128 per group → 255 total
- Study_Time: 198 per group → 396 total
- Retention_Rate: 128 per group → 255 total

Carryover Effect Analysis (Two-Way ANOVA with Interaction Term):
- Post_Test_Quiz_Score: 128 per group → 255 total
- Study_Time: 198 per group → 396 total
- Retention_Rate: 128 per group → 255 total

Dropout Rate (Chi-Square Test for Cross-Over RCT): 196 total students

Final Recommended Sample Size for Cross-Over RCT: 396 students
```

5.1.3 Cross-Over RCT Randomization (Blocking Factor)

In [250…

```
# Step 1: Collect Baseline Measures (Pre-Intervention Data) for all 5 RCT.
cross_over_rct_data = generate_baseline_data(cross_over_rct_sample_size)

# Step 2: Define the Blocking Factor (Performance Level)
# Create a new column combining GPA, Baseline Quiz Score, and Tech Savviness Score into a single score for bloc
cross_over_rct_data['Combined_Score'] = (
    cross_over_rct_data['GPA'] + cross_over_rct_data['Baseline_Quiz_Score'] + cross_over_rct_data['Tech_Savvine
)

# Define Performance_Level based on quartiles of the Combine_Score
cross_over_rct_data['Performance_Level'] = pd.qcut(
    cross_over_rct_data['Combined_Score'],
    q=3, #Devide into 3 equal-sized blocks (Low, Medium, High)
    labels=['Low', 'Medium', 'High']
)

# Step 3: Blocked Random Assignment (Cross-Over Design)
# Initialize the Group column
cross_over_rct_data['Group'] = None

# Perform random assignment within each block (Performance Level)
for level in cross_over_rct_data['Performance_Level'].unique():
    # Filter data for the current block
    block_data = cross_over_rct_data[cross_over_rct_data['Performance_Level'] == level]

    # Shuffle the indices for randomization
    np.random.seed(18)  # Set seed for reproducibility
    shuffled_indices = np.random.permutation(block_data.index)

    # Split into two equal groups: Cross-Over Group A and Group B
    half = len(shuffled_indices) // 2
    cross_over_rct_data.loc[shuffled_indices[:half], 'Group'] = 'A'
    cross_over_rct_data.loc[shuffled_indices[half:], 'Group'] = 'B'

# Drop the temporary 'Combined_Score' column
cross_over_rct_data.drop(columns=['Combined_Score'], inplace=True)
```

5.1.4: Cross-Over RCT Implement Intervention (Blocking Factor)

In [251…

```
# Step 3: Implement Intervention (Post-Intervention Data)
np.random.seed(18)

# Generate post-test metrics for Phase 1 (Static vs. Adaptive)
cross_over_rct_data['Phase1_Post_Test_Quiz_Score'] = (
    cross_over_rct_data['Baseline_Quiz_Score'] +
    np.where(cross_over_rct_data['Group'] == 'A',
```

```
            np.random.normal(5, 5, cross_over_rct_sample_size),  # Static first (Group A)
            np.random.normal(10, 5, cross_over_rct_sample_size))  # Adaptive first (Group B)
).astype(int)

cross_over_rct_data['Phase1_Improvement_Score'] = (
    cross_over_rct_data['Phase1_Post_Test_Quiz_Score'] - cross_over_rct_data['Baseline_Quiz_Score']
).astype(int)

cross_over_rct_data['Phase1_Study_Time'] = np.where(
    cross_over_rct_data['Group'] == 'A',
    np.random.normal(300, 50, cross_over_rct_sample_size),  # Static first (Group A)
    np.random.normal(400, 50, cross_over_rct_sample_size)   # Adaptive first (Group B)
).astype(int)

cross_over_rct_data['Phase1_Retention_Rate'] = np.where(
    cross_over_rct_data['Group'] == 'A',
    np.random.normal(80, 5, cross_over_rct_sample_size),  # Static first (Group A)
    np.random.normal(90, 5, cross_over_rct_sample_size)   # Adaptive first (Group B)
).round(2)

cross_over_rct_data['Phase1_Dropout_Rate'] = np.where(
    cross_over_rct_data['Group'] == 'A',
    np.random.choice([0, 1], cross_over_rct_sample_size, p=[0.75, 0.25]),  # Static first (Group A)
    np.random.choice([0, 1], cross_over_rct_sample_size, p=[0.9, 0.1])     # Adaptive first (Group B)
).round(2)

# Step 4: Switch Conditions After Washout Period (Cross-Over Phase)
# Swap Group Assignments: A → B, B → A
cross_over_rct_data['Group_Cross_Over'] = cross_over_rct_data['Group'].map({'A': 'B', 'B': 'A'})

cross_over_rct_data['Phase2_Post_Test_Quiz_Score'] = (
    cross_over_rct_data['Phase1_Post_Test_Quiz_Score'] +
    np.where(cross_over_rct_data['Group'] == 'A',
             np.random.normal(10, 5, cross_over_rct_sample_size),  # Adaptive second (Group A)
             np.random.normal(5, 5, cross_over_rct_sample_size))   # Static second (Group B)
).astype(int)

cross_over_rct_data['Phase2_Improvement_Score'] = (
    cross_over_rct_data['Phase2_Post_Test_Quiz_Score'] - cross_over_rct_data['Phase1_Post_Test_Quiz_Score']
).astype(int)

cross_over_rct_data['Phase2_Study_Time'] = np.where(
    cross_over_rct_data['Group'] == 'A',
    np.random.normal(400, 50, cross_over_rct_sample_size),  # Adaptive second (Group A)
    np.random.normal(300, 50, cross_over_rct_sample_size)   # Static second (Group B)
).astype(int)

cross_over_rct_data['Phase2_Retention_Rate'] = np.where(
    cross_over_rct_data['Group'] == 'A',
    np.random.normal(90, 5, cross_over_rct_sample_size),  # Adaptive second (Group A)
    np.random.normal(80, 5, cross_over_rct_sample_size)   # Static second (Group B)
).round(2)

cross_over_rct_data['Phase2_Dropout_Rate'] = np.where(
    cross_over_rct_data['Group'] == 'A',
    np.random.choice([0, 1], cross_over_rct_sample_size, p=[0.9, 0.1]),  # Adaptive second (Group A)
    np.random.choice([0, 1], cross_over_rct_sample_size, p=[0.75, 0.25]) # Static second (Group B)
).round(2)

# Drop temporary Group_Cross_Over column
cross_over_rct_data.drop(columns=['Group_Cross_Over'], inplace=True)
```

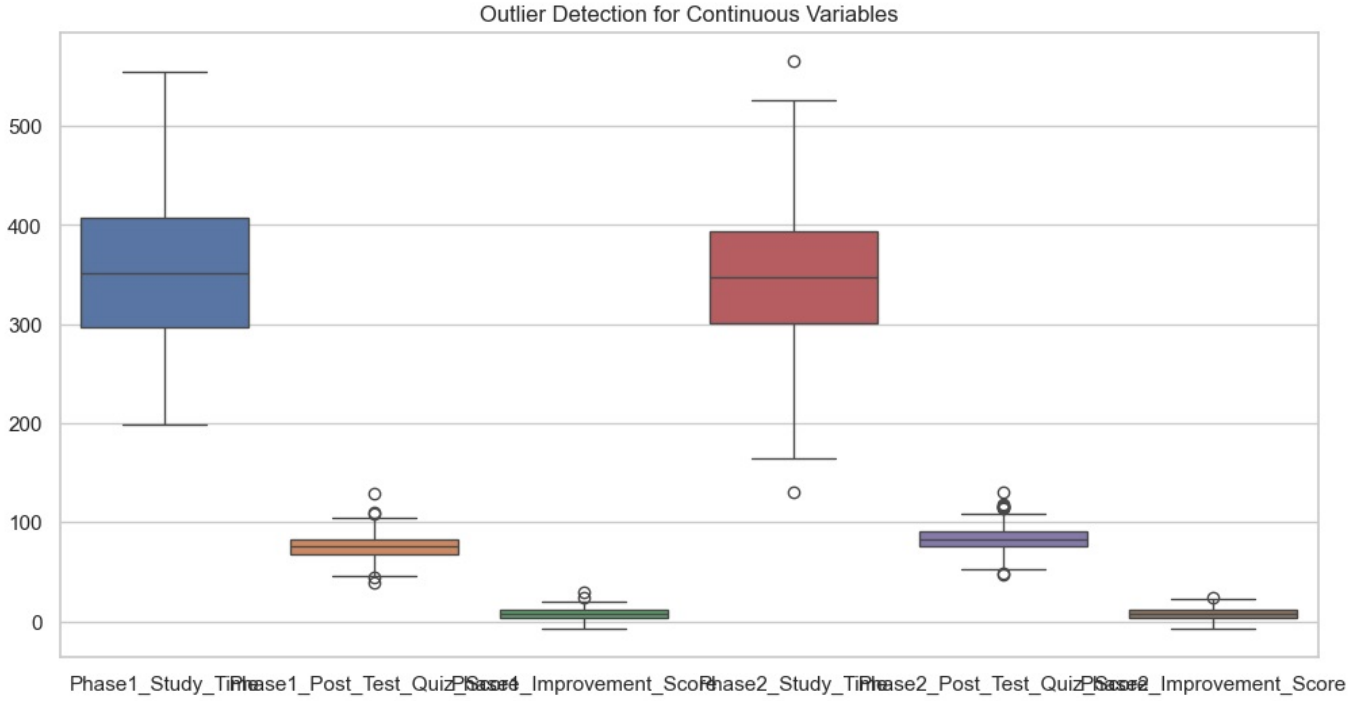
5.1.5: Cross-Over RCT Data Cleaning (Blocking Factor)

In [252…

```
# 1. Check for Missing Data

print("Missing Values Check:\n")
print(cross_over_rct_data.isnull().sum())

# Replace placeholder codes (e.g., -18) with NaN if present
cross_over_rct_data.replace(-18, np.nan, inplace=True)

# Drop rows with critical missing values (if any)
critical_cols = ['Phase1_Post_Test_Quiz_Score', 'Phase1_Retention_Rate', 'Group']
cross_over_rct_data.dropna(subset=critical_cols, inplace=True)


# 2. Remove Duplicates

print(f"\nInitial Shape: {cross_over_rct_data.shape}")
cross_over_rct_data.drop_duplicates(subset=['Student_ID'], keep='first', inplace=True)
print(f"Post-Deduplication Shape: {cross_over_rct_data.shape}")
```

```python
# 3. Verify Randomization Balance

print("\nGroup Balance:")
print(cross_over_rct_data['Group'].value_counts())

# Compare baseline covariates between groups
print("\nBaseline Covariate Balance:")
baseline_cols = ['GPA', 'Baseline_Quiz_Score', 'Study_Hours_Per_Week']
for col in baseline_cols:
    adaptive = cross_over_rct_data[cross_over_rct_data['Group'] == 'Adaptive'][col]
    static = cross_over_rct_data[cross_over_rct_data['Group'] == 'Static'][col]
    t_stat, p_value = ttest_ind(adaptive, static, nan_policy='omit')
    print(f"{col}: t = {t_stat:.2f}, p = {p_value:.4f}")


# 4. Detect and Handle Outliers

continuous_vars = ['Phase1_Study_Time', 'Phase1_Post_Test_Quiz_Score', 'Phase1_Improvement_Score',
                   'Phase2_Study_Time', 'Phase2_Post_Test_Quiz_Score', 'Phase2_Improvement_Score']

plt.figure(figsize=(12, 6))
sns.boxplot(data=cross_over_rct_data[continuous_vars])
plt.title("Outlier Detection for Continuous Variables")
plt.show()

# Cap outliers using IQR
for var in continuous_vars:
    q1 = cross_over_rct_data[var].quantile(0.25)
    q3 = cross_over_rct_data[var].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    cross_over_rct_data[var] = np.where(cross_over_rct_data[var] < lower_bound, lower_bound,
                                        np.where(cross_over_rct_data[var] > upper_bound, upper_bound,
                                        cross_over_rct_data[var]))


# 5. Validate Variable Ranges/Types

# Ensure binary variables are 0/1 for dropout rates
cross_over_rct_data['Phase1_Dropout_Rate'] = cross_over_rct_data['Phase1_Dropout_Rate'].apply(lambda x: 1 if x =
cross_over_rct_data['Phase2_Dropout_Rate'] = cross_over_rct_data['Phase2_Dropout_Rate'].apply(lambda x: 1 if x =

# Standardize categorical variables
cross_over_rct_data['Learning_Style'] = cross_over_rct_data['Learning_Style'].str.strip().str.title()

# Ensure valid percentage ranges for retention rate (0-100)
cross_over_rct_data['Phase1_Retention_Rate'] = cross_over_rct_data['Phase1_Retention_Rate'].clip(0, 100)
cross_over_rct_data['Phase2_Retention_Rate'] = cross_over_rct_data['Phase2_Retention_Rate'].clip(0, 100)


# 6. Save Cleaned Data

clean_cross_over_rct_data = cross_over_rct_data.copy()
print("\nData cleaning complete. Cleaned data saved as clean_cross_over_rct_data.")
```

```
Missing Values Check:

Student_ID                    0
Age                           0
GPA                           0
Study_Hours_Per_Week          0
Tech_Savviness_Score          0
Learning_Style                0
Baseline_Quiz_Score           0
Attention_Span                0
Motivation_Level              0
Prior_Online_Exp              0
Performance_Level             0
Group                         0
Phase1_Post_Test_Quiz_Score   0
Phase1_Improvement_Score      0
Phase1_Study_Time             0
Phase1_Retention_Rate         0
Phase1_Dropout_Rate           0
Phase2_Post_Test_Quiz_Score   0
Phase2_Improvement_Score      0
Phase2_Study_Time             0
Phase2_Retention_Rate         0
Phase2_Dropout_Rate           0
dtype: int64

Initial Shape: (396, 22)
Post-Deduplication Shape: (396, 22)

Group Balance:
Group
B    199
A    197
Name: count, dtype: int64

Baseline Covariate Balance:
GPA: t = nan, p = nan
Baseline_Quiz_Score: t = nan, p = nan
Study_Hours_Per_Week: t = nan, p = nan
```




```
Data cleaning complete. Cleaned data saved as clean_cross_over_rct_data.
```

5.1.6 Cross-Over RCT Within-Subject Comparison (Blocking Factor)

In [253…

```python
from statsmodels.stats.multitest import multipletests  # For multiple comparison correction

# List of metrics to analyze
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Initialize a list to store results
within_subject_results = []

# Function to perform within-subject comparison with blocking factor
def within_subject_comparison(group, metric, performance_level=None):
    # Filter data by group and performance level (if provided)
    if performance_level:
```

```
        data = clean_cross_over_rct_data[
            (clean_cross_over_rct_data['Group'] == group) &
            (clean_cross_over_rct_data['Performance_Level'] == performance_level)]
    else:
        data = clean_cross_over_rct_data[clean_cross_over_rct_data['Group'] == group]

    phase1 = data[f'Phase1_{metric}']
    phase2 = data[f'Phase2_{metric}']

    mean_phase1 = np.mean(phase1)
    mean_phase2 = np.mean(phase2)

    # Determine which phase is better based on the group and the metric
    if group == 'A':  # Group A: Static -> Adaptive
        better_phase = "GA_P2 Adaptive" if mean_phase1 < mean_phase2 else "GA_P1 Static"
    else:  # Group B: Adaptive -> Static
        better_phase = "GB_P2 Static" if mean_phase1 < mean_phase2 else "GB_P1 Adaptive"

    # Perform paired t-test
    t_stat, p_value = ttest_rel(phase1, phase2)

    # Determine statistical significance
    significance = "Significant" if p_value < 0.05 else "Not Significant"

    # Append results to the list
    within_subject_results.append([
        group, metric, performance_level if performance_level else 'Overall',
        round(mean_phase1, 3), round(mean_phase2, 3), better_phase,
        round(t_stat, 3), p_value, significance
    ])

# Perform Within-Subject Comparison for each group, metric, and performance level
for metric in metrics:
    for group in ['A', 'B']:
        # Overall comparison (without blocking factor)
        within_subject_comparison(group, metric)

        # Comparison by performance level (blocking factor)
        for performance_level in ['Low', 'Medium', 'High']:
            within_subject_comparison(group, metric, performance_level)

# Convert results to DataFrame
within_subject_df = pd.DataFrame(
    within_subject_results,
    columns=['Group', 'Metric', 'Performance_Level', 'Mean (Phase 1)', 'Mean (Phase 2)', 'Better Phase',
             't-Statistic', 'p-Value', 'Significance']
)

# Filter results by Group A and Group B
group_a_df = within_subject_df[within_subject_df['Group'] == 'A']
group_b_df = within_subject_df[within_subject_df['Group'] == 'B']

# Convert specific columns to integers in Group A
metrics_to_convert = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']
group_a_df.loc[group_a_df['Metric'].isin(metrics_to_convert), ['Mean (Phase 1)', 'Mean (Phase 2)']] = group_a_d
    group_a_df['Metric'].isin(metrics_to_convert), ['Mean (Phase 1)', 'Mean (Phase 2)']
].astype(int)

# Convert specific columns to integers in Group B
group_b_df.loc[group_b_df['Metric'].isin(metrics_to_convert), ['Mean (Phase 1)', 'Mean (Phase 2)']] = group_b_d
    group_b_df['Metric'].isin(metrics_to_convert), ['Mean (Phase 1)', 'Mean (Phase 2)']
].astype(int)

# Display Group A Results
print("\n5.1.6: Cross-Over RCT Within-Subject Comparison: Group A (Static → Adaptive)\n")
display(group_a_df)

# Display Group B Results
print("\n5.1.6: Cross-Over RCT Within-Subject Comparison: Group B (Adaptive → Static)\n")
display(group_b_df)

# Post-Hoc Analysis: Pairwise Comparisons with Multiple Testing Correction
post_hoc_results = []

# Function to perform post-hoc pairwise comparisons
def perform_post_hoc(group, metric, performance_level=None):
    # Filter data by group and performance level (if provided)
    if performance_level:
        data = clean_cross_over_rct_data[
            (clean_cross_over_rct_data['Group'] == group) &
            (clean_cross_over_rct_data['Performance_Level'] == performance_level)]
    else:
        data = clean_cross_over_rct_data[clean_cross_over_rct_data['Group'] == group]
```

```
    phase1 = data[f'Phase1_{metric}']
    phase2 = data[f'Phase2_{metric}']

    # Perform paired t-test
    t_stat, p_value = ttest_rel(phase1, phase2)

    # Append results to the list
    post_hoc_results.append([group, metric, performance_level if performance_level else 'Overall', p_value])

# Perform post-hoc analysis for each group, metric, and performance level
for metric in metrics:
    for group in ['A', 'B']:
        # Overall comparison (without blocking factor)
        perform_post_hoc(group, metric)

        # Comparison by performance level (blocking factor)
        for performance_level in ['Low', 'Medium', 'High']:
            perform_post_hoc(group, metric, performance_level)

# Convert post-hoc results to DataFrame
post_hoc_df = pd.DataFrame(
    post_hoc_results,
    columns=['Group', 'Metric', 'Performance_Level', 'p-Value']
)

# Apply multiple testing correction (e.g., Bonferroni or FDR)
post_hoc_df['Adjusted p-Value'] = multipletests(post_hoc_df['p-Value'], method='bonferroni')[1]

# Add significance column
post_hoc_df['Significance'] = post_hoc_df['Adjusted p-Value'].apply(
    lambda x: 'Significant' if x < 0.05 else 'Not Significant'
)
```

5.1.6: Cross-Over RCT Within-Subject Comparison: Group A (Static → Adaptive)

	Group	Metric	Performance_Level	Mean (Phase 1)	Mean (Phase 2)	Better Phase	t-Statistic	p-Value	Significance
0	A	Post_Test_Quiz_Score	Overall	73.0	83.0	GA_P2 Adaptive	-25.736	8.994145e-65	Significant
1	A	Post_Test_Quiz_Score	Low	63.0	73.0	GA_P2 Adaptive	-13.904	3.799408e-21	Significant
2	A	Post_Test_Quiz_Score	Medium	73.0	83.0	GA_P2 Adaptive	-16.300	1.198817e-24	Significant
3	A	Post_Test_Quiz_Score	High	83.0	92.0	GA_P2 Adaptive	-14.421	8.851348e-22	Significant
8	A	Improvement_Score	Overall	4.0	9.0	GA_P2 Adaptive	-9.165	6.806285e-17	Significant
9	A	Improvement_Score	Low	4.0	9.0	GA_P2 Adaptive	-4.979	4.987668e-06	Significant
10	A	Improvement_Score	Medium	4.0	9.0	GA_P2 Adaptive	-6.405	1.925668e-08	Significant
11	A	Improvement_Score	High	5.0	9.0	GA_P2 Adaptive	-4.682	1.525463e-05	Significant
16	A	Study_Time	Overall	302.0	393.0	GA_P2 Adaptive	-17.368	1.607551e-41	Significant
17	A	Study_Time	Low	308.0	386.0	GA_P2 Adaptive	-9.001	4.941693e-13	Significant
18	A	Study_Time	Medium	307.0	403.0	GA_P2 Adaptive	-9.506	6.419129e-14	Significant
19	A	Study_Time	High	291.0	392.0	GA_P2 Adaptive	-11.982	5.285400e-18	Significant
24	A	Retention_Rate	Overall	80.0	90.0	GA_P2 Adaptive	-20.959	6.088254e-52	Significant
25	A	Retention_Rate	Low	81.0	91.0	GA_P2 Adaptive	-12.085	2.806120e-18	Significant
26	A	Retention_Rate	Medium	80.0	90.0	GA_P2 Adaptive	-13.214	4.424752e-20	Significant
27	A	Retention_Rate	High	80.0	89.0	GA_P2 Adaptive	-11.039	1.858959e-16	Significant

5.1.6: Cross-Over RCT Within-Subject Comparison: Group B (Adaptive → Static)

	Group	Metric	Performance_Level	Mean (Phase 1)	Mean (Phase 2)	Better Phase	t-Statistic	p-Value	Significance
4	B	Post_Test_Quiz_Score	Overall	78.0	82.0	GB_P2 Static	-12.444	1.222473e-26	Significant
5	B	Post_Test_Quiz_Score	Low	67.0	71.0	GB_P2 Static	-6.161	5.115126e-08	Significant
6	B	Post_Test_Quiz_Score	Medium	77.0	83.0	GB_P2 Static	-9.873	1.255706e-14	Significant
7	B	Post_Test_Quiz_Score	High	89.0	93.0	GB_P2 Static	-6.156	5.201907e-08	Significant
12	B	Improvement_Score	Overall	9.0	4.0	GB_P1 Adaptive	10.361	2.236516e-20	Significant
13	B	Improvement_Score	Low	9.0	3.0	GB_P1 Adaptive	6.259	3.459040e-08	Significant
14	B	Improvement_Score	Medium	9.0	5.0	GB_P1 Adaptive	4.833	8.364850e-06	Significant
15	B	Improvement_Score	High	9.0	3.0	GB_P1 Adaptive	6.883	2.794934e-09	Significant
20	B	Study_Time	Overall	404.0	303.0	GB_P1 Adaptive	19.943	3.018421e-49	Significant
21	B	Study_Time	Low	393.0	304.0	GB_P1 Adaptive	11.175	8.937851e-17	Significant
22	B	Study_Time	Medium	400.0	306.0	GB_P1 Adaptive	10.553	8.322480e-16	Significant
23	B	Study_Time	High	420.0	298.0	GB_P1 Adaptive	13.356	2.660179e-20	Significant
28	B	Retention_Rate	Overall	90.0	79.0	GB_P1 Adaptive	21.250	5.629398e-53	Significant
29	B	Retention_Rate	Low	91.0	79.0	GB_P1 Adaptive	15.082	6.591079e-23	Significant
30	B	Retention_Rate	Medium	88.0	80.0	GB_P1 Adaptive	9.610	3.638273e-14	Significant
31	B	Retention_Rate	High	90.0	79.0	GB_P1 Adaptive	13.235	4.101174e-20	Significant

In [254…
```
# Display Post-Hoc Results
print("\nPost-Hoc Analysis Results (Pairwise Comparisons with Bonferroni Correction):\n")
display(post_hoc_df)
```

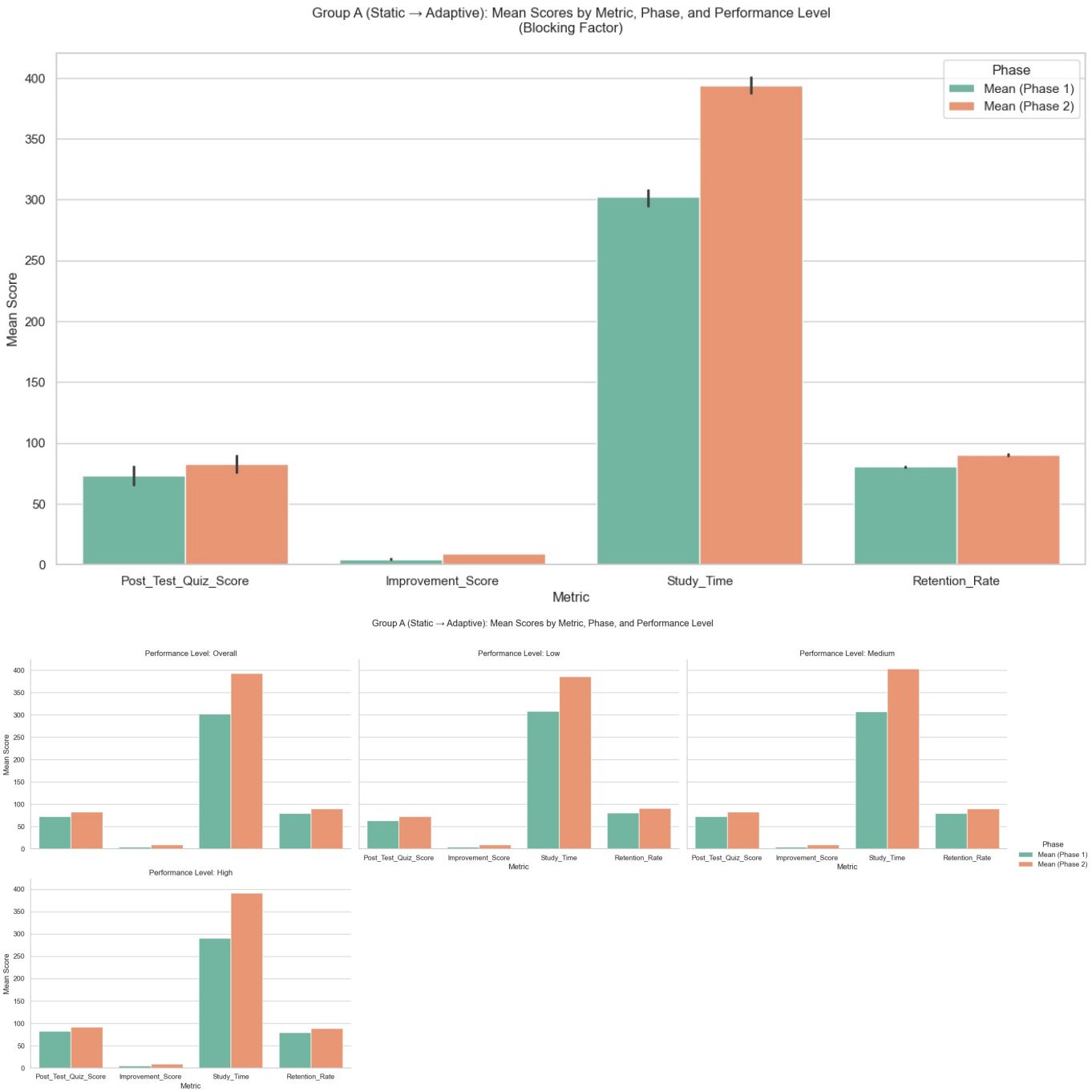
```
Post-Hoc Analysis Results (Pairwise Comparisons with Bonferroni Correction):
```

	Group	Metric	Performance_Level	p-Value	Adjusted p-Value	Significance
0	A	Post_Test_Quiz_Score	Overall	8.994145e-65	2.878126e-63	Significant
1	A	Post_Test_Quiz_Score	Low	3.799408e-21	1.215811e-19	Significant
2	A	Post_Test_Quiz_Score	Medium	1.198817e-24	3.836215e-23	Significant
3	A	Post_Test_Quiz_Score	High	8.851348e-22	2.832431e-20	Significant
4	B	Post_Test_Quiz_Score	Overall	1.222473e-26	3.911912e-25	Significant
5	B	Post_Test_Quiz_Score	Low	5.115126e-08	1.636840e-06	Significant
6	B	Post_Test_Quiz_Score	Medium	1.255706e-14	4.018259e-13	Significant
7	B	Post_Test_Quiz_Score	High	5.201907e-08	1.664610e-06	Significant
8	A	Improvement_Score	Overall	6.806285e-17	2.178011e-15	Significant
9	A	Improvement_Score	Low	4.987668e-06	1.596054e-04	Significant
10	A	Improvement_Score	Medium	1.925668e-08	6.162139e-07	Significant
11	A	Improvement_Score	High	1.525463e-05	4.881482e-04	Significant
12	B	Improvement_Score	Overall	2.236516e-20	7.156850e-19	Significant
13	B	Improvement_Score	Low	3.459040e-08	1.106893e-06	Significant
14	B	Improvement_Score	Medium	8.364850e-06	2.676752e-04	Significant
15	B	Improvement_Score	High	2.794934e-09	8.943789e-08	Significant
16	A	Study_Time	Overall	1.607551e-41	5.144164e-40	Significant
17	A	Study_Time	Low	4.941693e-13	1.581342e-11	Significant
18	A	Study_Time	Medium	6.419129e-14	2.054121e-12	Significant
19	A	Study_Time	High	5.285400e-18	1.691328e-16	Significant
20	B	Study_Time	Overall	3.018421e-49	9.658947e-48	Significant
21	B	Study_Time	Low	8.937851e-17	2.860112e-15	Significant
22	B	Study_Time	Medium	8.322480e-16	2.663194e-14	Significant
23	B	Study_Time	High	2.660179e-20	8.512574e-19	Significant
24	A	Retention_Rate	Overall	6.088254e-52	1.948241e-50	Significant
25	A	Retention_Rate	Low	2.806120e-18	8.979585e-17	Significant
26	A	Retention_Rate	Medium	4.424752e-20	1.415921e-18	Significant
27	A	Retention_Rate	High	1.858959e-16	5.948668e-15	Significant
28	B	Retention_Rate	Overall	5.629398e-53	1.801407e-51	Significant
29	B	Retention_Rate	Low	6.591079e-23	2.109145e-21	Significant
30	B	Retention_Rate	Medium	3.638273e-14	1.164247e-12	Significant
31	B	Retention_Rate	High	4.101174e-20	1.312376e-18	Significant

In [255…

```
# Melt the Group A results DataFrame for easier plotting
group_a_melted = group_a_df.melt(
    id_vars=['Metric', 'Performance_Level'],  # Corrected column name
    value_vars=['Mean (Phase 1)', 'Mean (Phase 2)'],
    var_name='Phase',
    value_name='Mean Score'
)

# Plot Group A results by performance level
plt.figure(figsize=(16, 8))
sns.barplot(
    x='Metric',
    y='Mean Score',
    hue='Phase',
    data=group_a_melted,
    palette='Set2'
)
plt.title('Group A (Static → Adaptive): Mean Scores by Metric, Phase, and Performance Level\n(Blocking Factor)\
plt.xlabel('Metric')
plt.ylabel('Mean Score')
plt.legend(title='Phase')
plt.show()

# Faceted plot for Group A by performance level
g = sns.FacetGrid(
    group_a_melted,
    col='Performance_Level',
```

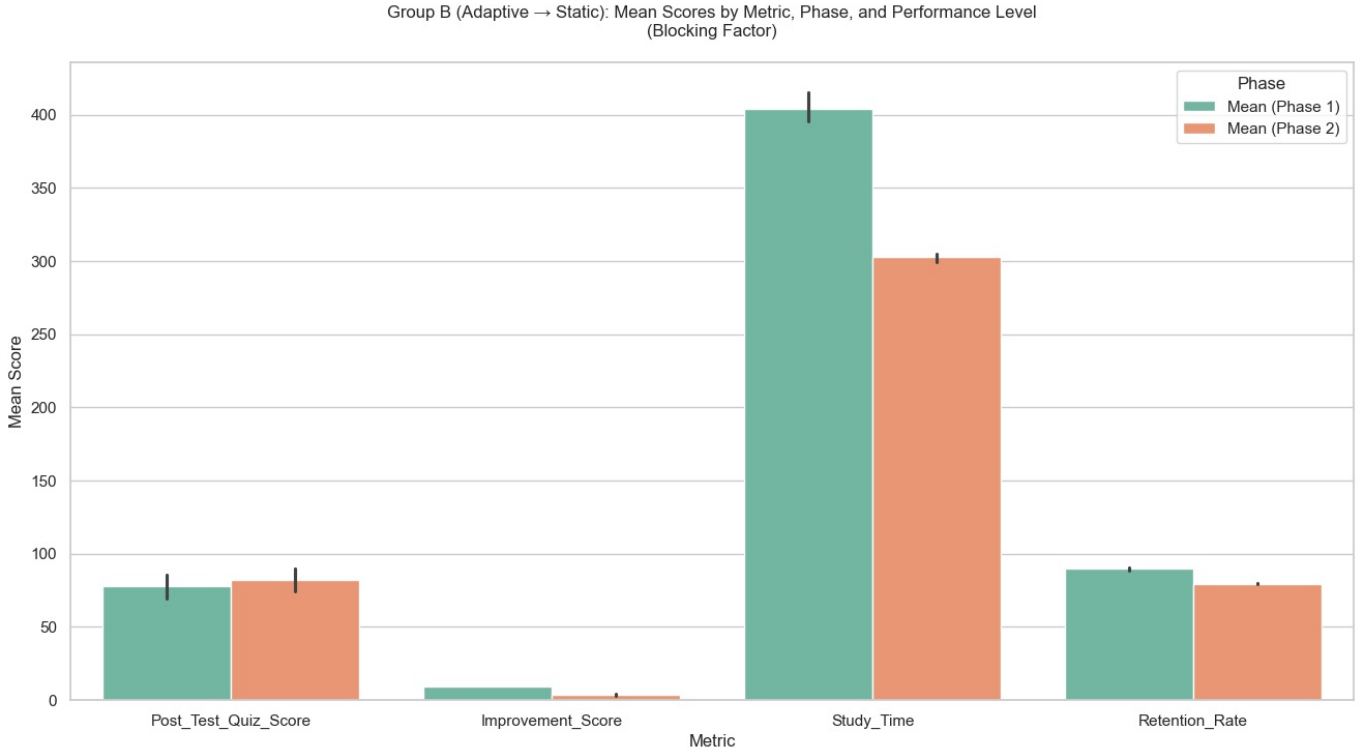
```
    col_wrap=3,
    height=5,
    aspect=1.5
)
g.map_dataframe(sns.barplot, x='Metric', y='Mean Score', hue='Phase', palette='Set2')
g.set_axis_labels('Metric', 'Mean Score')
g.set_titles('Performance Level: {col_name}')
g.add_legend(title='Phase')
plt.suptitle('Group A (Static → Adaptive): Mean Scores by Metric, Phase, and Performance Level', y=1.05)  # Adj
plt.show()
```

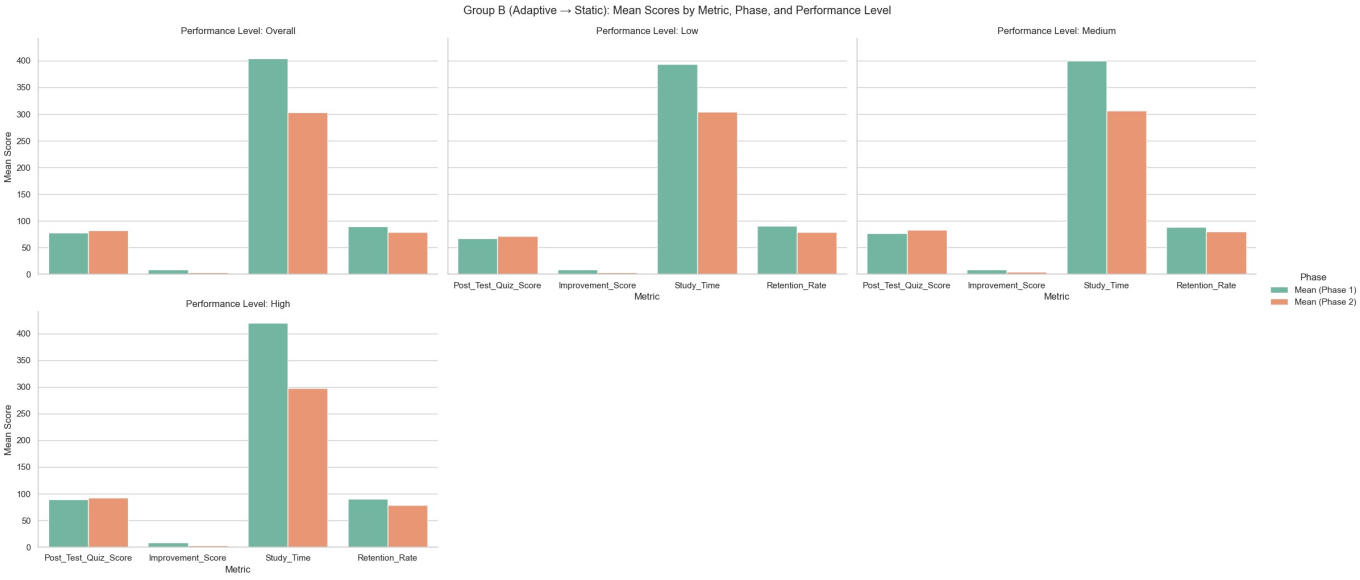


In [256…

```
# Melt the Group A results DataFrame for easier plotting
group_b_melted = group_b_df.melt(
    id_vars=['Metric', 'Performance_Level'],  # Corrected column name
    value_vars=['Mean (Phase 1)', 'Mean (Phase 2)'],
    var_name='Phase',
    value_name='Mean Score'
)

# Plot Group B results by performance level
plt.figure(figsize=(16, 8))
sns.barplot(
    x='Metric',
    y='Mean Score',
    hue='Phase',
    data=group_b_melted,
    palette='Set2'
)
plt.title('Group B (Adaptive → Static): Mean Scores by Metric, Phase, and Performance Level\n(Blocking Factor)\
plt.xlabel('Metric')
```

```
plt.ylabel('Mean Score')
plt.legend(title='Phase')
plt.show()

# Faceted plot for Group B by performance level
g = sns.FacetGrid(group_b_melted, col='Performance_Level', col_wrap=3, height=5, aspect=1.5)
g.map_dataframe(sns.barplot, x='Metric', y='Mean Score', hue='Phase', palette='Set2')
g.set_axis_labels('Metric', 'Mean Score')
g.set_titles('Performance Level: {col_name}')
g.add_legend(title='Phase')
plt.suptitle('Group B (Adaptive → Static): Mean Scores by Metric, Phase, and Performance Level', y=1.02)
plt.show()
```





5.1.6 Cross-Over RCT Within-Subject Comparison - Chi-Square Analysis of Dropout Rate (Blocking Factor)

In [257…

```
# Initialize a list to store results
chi_square_results = []

# Function to perform Chi-Square test for a given performance level
def chi_square_analysis(performance_level=None):
    # Filter data by performance level (if provided)
    if performance_level:
        data = clean_cross_over_rct_data[clean_cross_over_rct_data['Performance_Level'] == performance_level]
    else:
        data = clean_cross_over_rct_data

    # Create a contingency table for Phase 1 and Phase 2 dropout rates
    dropout_contingency_table = pd.DataFrame({
        'Phase1_Adaptive': data.loc[data['Group'] == 'A', 'Phase1_Dropout_Rate'].value_counts(),
```

```
        'Phase1_Static': data.loc[data['Group'] == 'B', 'Phase1_Dropout_Rate'].value_counts(),
        'Phase2_Adaptive': data.loc[data['Group'] == 'B', 'Phase2_Dropout_Rate'].value_counts(),
        'Phase2_Static': data.loc[data['Group'] == 'A', 'Phase2_Dropout_Rate'].value_counts()
    }).fillna(0)

    # Perform Chi-Square Test
    chi2_stat, p_value, dof, expected = chi2_contingency(dropout_contingency_table)

    # Calculate dropout rates for each phase and group
    dropout_rates = pd.DataFrame({
        'Phase 1': [
            data.loc[data['Group'] == 'A', 'Phase1_Dropout_Rate'].mean(),
            data.loc[data['Group'] == 'B', 'Phase1_Dropout_Rate'].mean()
        ],
        'Phase 2': [
            data.loc[data['Group'] == 'A', 'Phase2_Dropout_Rate'].mean(),
            data.loc[data['Group'] == 'B', 'Phase2_Dropout_Rate'].mean()
        ]
    }, index=['Group A (Static → Adaptive)', 'Group B (Adaptive → Static)'])

    # Define phase labels
    phase_labels = {
        'Group A (Static → Adaptive)': ['Phase 1 Static', 'Phase 2 Adaptive'],
        'Group B (Adaptive → Static)': ['Phase 1 Adaptive', 'Phase 2 Static']
    }

    # Determine the better phase (lower dropout rate) and label it correctly
    dropout_rates['Better Phase'] = dropout_rates.apply(
        lambda row: phase_labels[row.name][0] if row['Phase 1'] < row['Phase 2'] else phase_labels[row.name][1]
        axis=1
    )

    # Determine significance
    significance = 'Significant' if p_value < 0.05 else 'Not Significant'

    # Append results to the list
    chi_square_results.append([
        performance_level if performance_level else 'Overall',
        dropout_rates.loc['Group A (Static → Adaptive)', 'Phase 1'],
        dropout_rates.loc['Group A (Static → Adaptive)', 'Phase 2'],
        dropout_rates.loc['Group A (Static → Adaptive)', 'Better Phase'],
        dropout_rates.loc['Group B (Adaptive → Static)', 'Phase 1'],
        dropout_rates.loc['Group B (Adaptive → Static)', 'Phase 2'],
        dropout_rates.loc['Group B (Adaptive → Static)', 'Better Phase'],
        chi2_stat, p_value, significance
    ])

# Perform Chi-Square analysis for overall data and each performance level
chi_square_analysis()  # Overall analysis
for performance_level in ['Low', 'Medium', 'High']:
    chi_square_analysis(performance_level)

# Convert results to DataFrame
chi_square_results_df = pd.DataFrame(
    chi_square_results,
    columns=[
        'Performance Level', 'Group A Phase 1 Dropout Rate', 'Group A Phase 2 Dropout Rate', 'Group A Better Pha
        'Group B Phase 1 Dropout Rate', 'Group B Phase 2 Dropout Rate', 'Group B Better Phase',
        'Chi-Square Statistic', 'p-value', 'Significance'
    ]
)

# Split the results DataFrame into Group A and Group B
group_a_results_df = chi_square_results_df[[
    'Performance Level', 'Group A Phase 1 Dropout Rate', 'Group A Phase 2 Dropout Rate', 'Group A Better Phase'
    'Chi-Square Statistic', 'p-value', 'Significance'
]]

group_b_results_df = chi_square_results_df[[
    'Performance Level', 'Group B Phase 1 Dropout Rate', 'Group B Phase 2 Dropout Rate', 'Group B Better Phase'
    'Chi-Square Statistic', 'p-value', 'Significance'
]]

# Rename columns for clarity
group_a_results_df.columns = [
    'Performance Level', 'Phase 1 Dropout Rate', 'Phase 2 Dropout Rate', 'Better Phase',
    'Chi-Square Statistic', 'p-value', 'Significance'
]

group_b_results_df.columns = [
    'Performance Level', 'Phase 1 Dropout Rate', 'Phase 2 Dropout Rate', 'Better Phase',
    'Chi-Square Statistic', 'p-value', 'Significance'
]
```

```
# Display Group A Results
print("\n5.1.6 (Blocking Factor) Cross-Over RCT Within-Subject Comparison - Chi-Square Analysis of Dropout Rate
display(group_a_results_df)

# Display Group B Results
print("\n5.1.6 (Blocking Factor) Cross-Over RCT Within-Subject Comparison - Chi-Square Analysis of Dropout Rate
display(group_b_results_df)
```

5.1.6 (Blocking Factor) Cross-Over RCT Within-Subject Comparison - Chi-Square Analysis of Dropout Rate (Group A: Static → Adaptive)

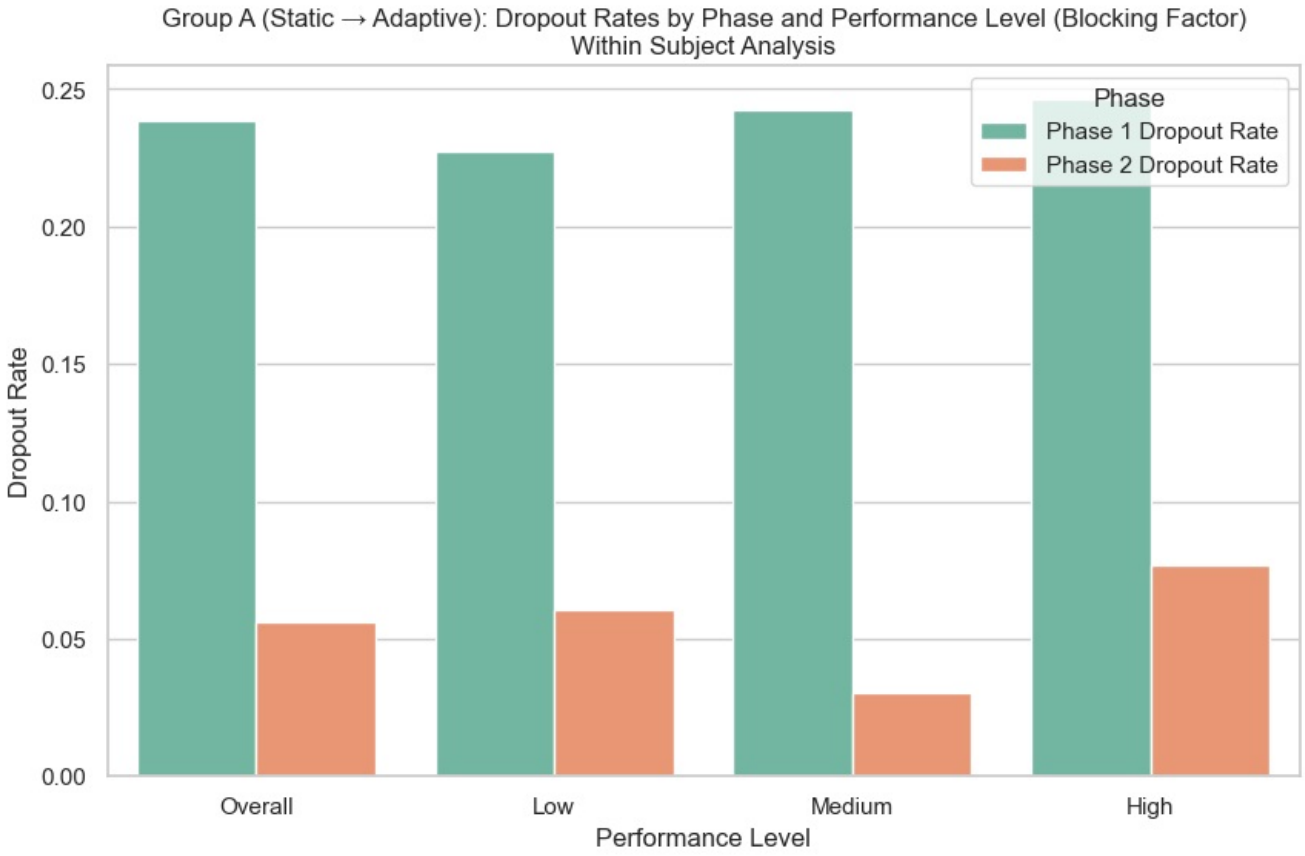
	Performance Level	Phase 1 Dropout Rate	Phase 2 Dropout Rate	Better Phase	Chi-Square Statistic	p-value	Significance
0	Overall	0.238579	0.055838	Phase 2 Adaptive	53.346913	1.546262e-11	Significant
1	Low	0.227273	0.060606	Phase 2 Adaptive	15.815668	1.237048e-03	Significant
2	Medium	0.242424	0.030303	Phase 2 Adaptive	19.304477	2.364932e-04	Significant
3	High	0.246154	0.076923	Phase 2 Adaptive	20.113960	1.607572e-04	Significant

5.1.6 (Blocking Factor) Cross-Over RCT Within-Subject Comparison - Chi-Square Analysis of Dropout Rate (Group B: Adaptive → Static)

	Performance Level	Phase 1 Dropout Rate	Phase 2 Dropout Rate	Better Phase	Chi-Square Statistic	p-value	Significance
0	Overall	0.100503	0.296482	Phase 1 Adaptive	53.346913	1.546262e-11	Significant
1	Low	0.121212	0.303030	Phase 1 Adaptive	15.815668	1.237048e-03	Significant
2	Medium	0.074627	0.238806	Phase 1 Adaptive	19.304477	2.364932e-04	Significant
3	High	0.106061	0.348485	Phase 1 Adaptive	20.113960	1.607572e-04	Significant

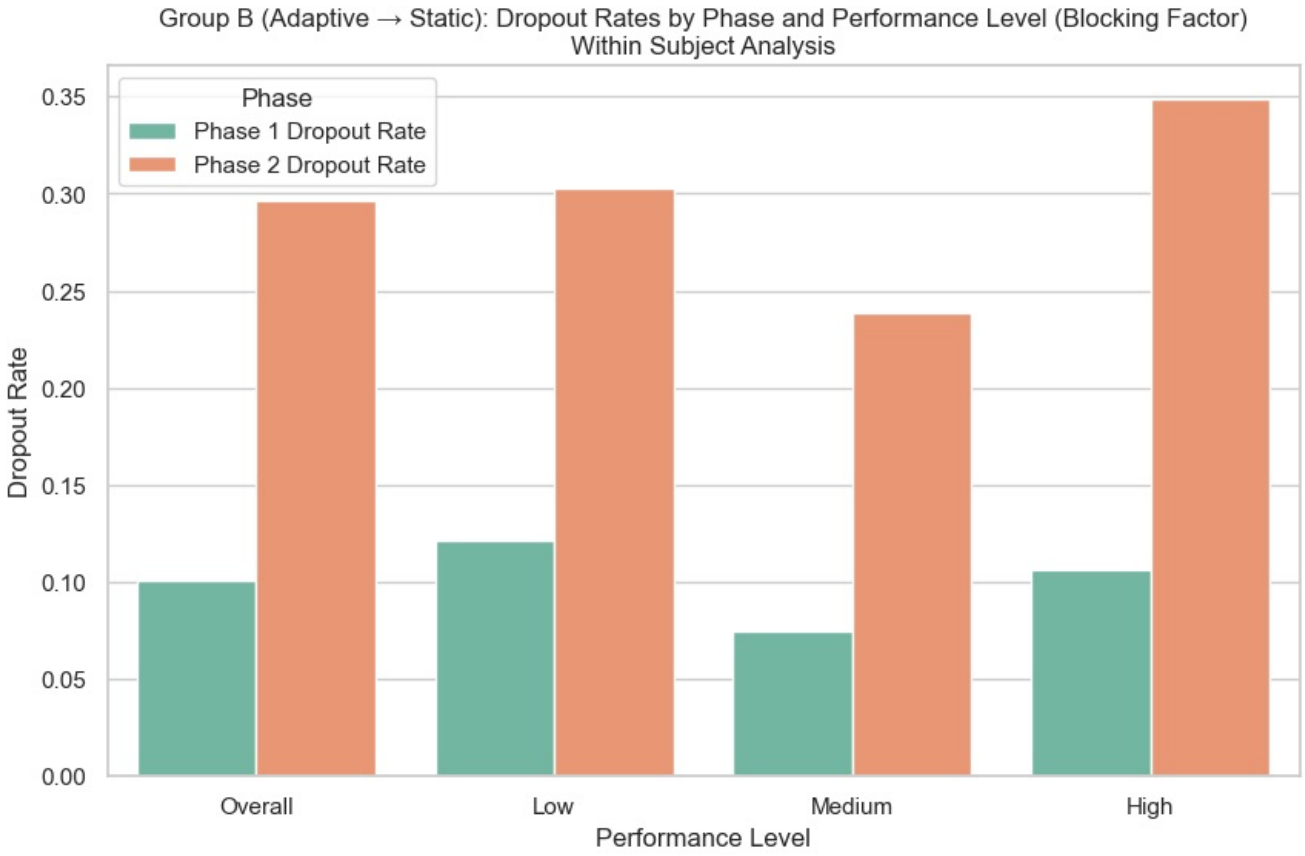
In [258…

```
# Melt the Group A results DataFrame for easier plotting
group_a_melted = group_a_results_df.melt(
    id_vars=['Performance Level'],
    value_vars=['Phase 1 Dropout Rate', 'Phase 2 Dropout Rate'],
    var_name='Phase',
    value_name='Dropout Rate'
)

# Grouped bar plot for Group A
plt.figure(figsize=(10, 6))
sns.barplot(
    x='Performance Level',
    y='Dropout Rate',
    hue='Phase',
    data=group_a_melted,
    palette='Set2'
)
plt.title('Group A (Static → Adaptive): Dropout Rates by Phase and Performance Level (Blocking Factor)\nWithin S
plt.xlabel('Performance Level')
plt.ylabel('Dropout Rate')
plt.legend(title='Phase')
plt.show()
```



In [259…

```python
# Melt the Group B results DataFrame for easier plotting
group_b_melted = group_b_results_df.melt(
    id_vars=['Performance Level'],
    value_vars=['Phase 1 Dropout Rate', 'Phase 2 Dropout Rate'],
    var_name='Phase',
    value_name='Dropout Rate'
)

# Grouped bar plot for Group B
plt.figure(figsize=(10, 6))
sns.barplot(
    x='Performance Level',
    y='Dropout Rate',
    hue='Phase',
    data=group_b_melted,
    palette='Set2'
)
plt.title('Group B (Adaptive → Static): Dropout Rates by Phase and Performance Level (Blocking Factor)\nWithin S
plt.xlabel('Performance Level')
plt.ylabel('Dropout Rate')
plt.legend(title='Phase')
plt.show()
```



5.1.7 Interpretation of Cross-Over RCT Within-Subject Comparison Analysis (Blocking Factor)

1. **Group A (Static → Adaptive)**

The within-subject comparison for Group A, which transitioned from a static learning phase to an adaptive learning phase, revealed significant improvements across all metrics:

- **Post-Test Quiz Scores**: Across overall, low, medium, and high performance levels, mean scores improved significantly in Phase 2 (adaptive phase). For instance, the overall mean increased from **73.0** to **83.0** with a highly significant t-statistic of **-25.736** ($p < 0.001$).
- **Improvement Scores**: There was a marked increase in scores from Phase 1 to Phase 2 (e.g., from **4.0** to **9.0** overall), with all results being statistically significant ($p < 0.001$).
- **Study Time**: Participants spent more time studying during the adaptive phase (from **302.0** to **393.0** minutes overall), indicating greater engagement.
- **Retention Rate**: The overall retention rate improved from **80.0%** to **90.0%** in Phase 2, again showing a significant positive effect from the adaptive system.

2. **Group B (Adaptive → Static)**

In contrast, Group B, which transitioned from an adaptive to a static learning phase, exhibited a decline in most metrics during Phase 2:

- **Post-Test Quiz Scores**: While scores improved slightly in Phase 2 (static phase) (e.g., from **78.0** to **82.0** overall), the gains were statistically significant but less substantial than those seen in Group A.
- **Improvement Scores**: Higher improvement scores were recorded in Phase 1 (adaptive phase) across all performance levels, indicating a significant decline in learning gains once participants switched to the static phase.
- **Study Time**: Participants spent more time studying during the adaptive phase (e.g., **404.0** vs. **303.0** minutes overall), highlighting a drop in engagement during the static phase.
- **Retention Rate**: There was a significant reduction in retention rates from **90.0%** to **79.0%** when moving from adaptive to static phases.

3. **Blocking Factor Effects (Dropout Rate Analysis)**

A chi-square analysis revealed significant effects of the learning phase on dropout rates:

- **Group A**: The dropout rate significantly decreased during the adaptive phase across all performance levels (e.g., overall dropout reduced from **23.86%** to **5.58%**, χ^2 = **53.35**, $p < 0.001$). This suggests the adaptive learning system increased student retention.
- **Group B**: Conversely, the dropout rate increased during the static phase (e.g., overall dropout increased from **10.05%** to **29.65%**,

χ^2 = **53.35**, $p < 0.001$). This further reinforces the effectiveness of the adaptive learning phase in retaining students.

4. Conclusion

The findings demonstrate the superior effectiveness of adaptive learning systems across multiple performance metrics. Group A consistently improved when switching from static to adaptive, while Group B showed declines when moving from adaptive to static. The blocking factor analysis of dropout rates supports these results, as adaptive phases were associated with significantly lower dropout rates.

Overall, the adaptive learning approach enhances post-test quiz scores, improvement scores, study time, retention rates, and minimizes dropout rates across all performance levels.

5.2.3 Cross-Over RCT Between-Subject Comparison (Blocking Factor)

In [260…

```
# Initialize a list to store results
between_subject_results = []

# Function to perform between-subject comparison for a given phase, metric, and performance level
def between_subject_comparison(phase, metric, performance_level=None):
    # Filter data by performance level (if provided)
    if performance_level:
        data = clean_cross_over_rct_data[clean_cross_over_rct_data['Performance_Level'] == performance_level]
    else:
        data = clean_cross_over_rct_data

    # Extract data for Group A and Group B
    group_a = data[data['Group'] == 'A'][f'{phase}_{metric}']
    group_b = data[data['Group'] == 'B'][f'{phase}_{metric}']

    mean_a = np.mean(group_a)
    mean_b = np.mean(group_b)

    # Determine which group is better based on the phase and the metric
    if phase == 'Phase1':  # Group A: Static -> Adaptive
        if metric == 'Dropout_Rate':  # For Dropout_Rate, smaller is better
            better_group = "GA Static" if mean_a < mean_b else "GB Adaptive"
        else:  # For all other metrics, bigger is better
            better_group = "GB Adaptive" if mean_a < mean_b else "GA Static"
    else:  # Group B: Adaptive -> Static
        if metric == 'Dropout_Rate':  # For Dropout_Rate, smaller is better
            better_group = "GA Adaptive" if mean_a < mean_b else "GB Static"
        else:  # For all other metrics, bigger is better
            better_group = "GB Static" if mean_a < mean_b else "GA Adaptive"

    # Perform ANOVA (F-test) to compare means
    f_stat, p_value = f_oneway(group_a, group_b)

    # Append results to the list
    between_subject_results.append([
        phase, metric, performance_level if performance_level else 'Overall',
        round(mean_a, 3), round(mean_b, 3), better_group, round(f_stat, 3), round(p_value, 4)
    ])

# Perform Between-Subject Comparison for each phase, metric, and performance level
for phase in ['Phase1', 'Phase2']:
    for metric in metrics:
        # Overall comparison (without blocking factor)
        between_subject_comparison(phase, metric)

        # Comparison by performance level (blocking factor)
        for performance_level in ['Low', 'Medium', 'High']:
            between_subject_comparison(phase, metric, performance_level)

# Convert results to DataFrame
between_subject_df = pd.DataFrame(
    between_subject_results,
    columns=['Phase', 'Metric', 'Performance Level', 'Mean (Group A)', 'Mean (Group B)', 'Better Group', 'F-Sta
)

# Convert specific metrics to whole numbers
metrics_to_convert = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']
between_subject_df.loc[between_subject_df['Metric'].isin(metrics_to_convert), ['Mean (Group A)', 'Mean (Group B
    between_subject_df.loc[between_subject_df['Metric'].isin(metrics_to_convert), ['Mean (Group A)', 'Mean (Gro
    .round(0)  # Round to nearest whole number
    .astype(int)  # Convert to integer type
)

# Filter for Phase 1 results
phase1_results = between_subject_df[between_subject_df['Phase'] == 'Phase1']
```

```
print("\n5.1.6: Cross-Over RCT Between-Subject Comparison: Phase 1 Results\n")
display(phase1_results)

# Filter for Phase 2 results
phase2_results = between_subject_df[between_subject_df['Phase'] == 'Phase2']
print("\n5.1.6: Cross-Over RCT Between-Subject Comparison: Phase 2 Results\n")
display(phase2_results)

# Post-Hoc Analysis: Pairwise Comparisons (Tukey's HSD)
post_hoc_results = []

# Function to perform post-hoc pairwise comparisons
def perform_post_hoc(phase, metric, performance_level=None):
    # Filter data by performance level (if provided)
    if performance_level:
        data = clean_cross_over_rct_data[clean_cross_over_rct_data['Performance_Level'] == performance_level]
    else:
        data = clean_cross_over_rct_data

    # Extract data for Group A and Group B
    group_a = data[data['Group'] == 'A'][f'{phase}_{metric}']
    group_b = data[data['Group'] == 'B'][f'{phase}_{metric}']

    # Combine data into a single DataFrame for Tukey's HSD
    combined_data = pd.DataFrame({
        'values': pd.concat([group_a, group_b]),
        'group': ['Group A'] * len(group_a) + ['Group B'] * len(group_b)
    })

    # Perform Tukey's HSD test
    tukey_result = pairwise_tukeyhsd(endog=combined_data['values'], groups=combined_data['group'], alpha=0.05)

    # Append results to the list
    post_hoc_results.append([phase, metric, performance_level if performance_level else 'Overall', tukey_result

# Perform post-hoc analysis for each phase, metric, and performance level
for phase in ['Phase1', 'Phase2']:
    for metric in metrics:
        # Overall comparison (without blocking factor)
        perform_post_hoc(phase, metric)

        # Comparison by performance level (blocking factor)
        for performance_level in ['Low', 'Medium', 'High']:
            perform_post_hoc(phase, metric, performance_level)
```

5.1.6: Cross-Over RCT Between-Subject Comparison: Phase 1 Results

	Phase	Metric	Performance Level	Mean (Group A)	Mean (Group B)	Better Group	F-Statistic	p-Value
0	Phase1	Post_Test_Quiz_Score	Overall	73.0	78.0	GB Adaptive	17.662	0.0000
1	Phase1	Post_Test_Quiz_Score	Low	64.0	67.0	GB Adaptive	6.360	0.0129
2	Phase1	Post_Test_Quiz_Score	Medium	73.0	78.0	GB Adaptive	22.795	0.0000
3	Phase1	Post_Test_Quiz_Score	High	83.0	89.0	GB Adaptive	19.718	0.0000
4	Phase1	Improvement_Score	Overall	5.0	9.0	GB Adaptive	85.212	0.0000
5	Phase1	Improvement_Score	Low	4.0	9.0	GB Adaptive	25.253	0.0000
6	Phase1	Improvement_Score	Medium	4.0	9.0	GB Adaptive	33.125	0.0000
7	Phase1	Improvement_Score	High	5.0	10.0	GB Adaptive	27.244	0.0000
8	Phase1	Study_Time	Overall	302.0	405.0	GB Adaptive	428.792	0.0000
9	Phase1	Study_Time	Low	308.0	393.0	GB Adaptive	97.506	0.0000
10	Phase1	Study_Time	Medium	307.0	400.0	GB Adaptive	108.254	0.0000
11	Phase1	Study_Time	High	292.0	420.0	GB Adaptive	284.738	0.0000
12	Phase1	Retention_Rate	Overall	81.0	90.0	GB Adaptive	384.806	0.0000
13	Phase1	Retention_Rate	Low	81.0	92.0	GB Adaptive	151.349	0.0000
14	Phase1	Retention_Rate	Medium	80.0	89.0	GB Adaptive	135.185	0.0000
15	Phase1	Retention_Rate	High	81.0	90.0	GB Adaptive	111.016	0.0000

5.1.6: Cross-Over RCT Between-Subject Comparison: Phase 2 Results

	Phase	Metric	Performance Level	Mean (Group A)	Mean (Group B)	Better Group	F-Statistic	p-Value
16	Phase2	Post_Test_Quiz_Score	Overall	83.0	83.0	GA Adaptive	0.174	0.6770
17	Phase2	Post_Test_Quiz_Score	Low	73.0	71.0	GA Adaptive	1.387	0.2411
18	Phase2	Post_Test_Quiz_Score	Medium	83.0	83.0	GB Static	0.002	0.9606
19	Phase2	Post_Test_Quiz_Score	High	93.0	93.0	GB Static	0.032	0.8589
20	Phase2	Improvement_Score	Overall	10.0	4.0	GA Adaptive	103.000	0.0000
21	Phase2	Improvement_Score	Low	10.0	4.0	GA Adaptive	36.271	0.0000
22	Phase2	Improvement_Score	Medium	10.0	5.0	GA Adaptive	33.898	0.0000
23	Phase2	Improvement_Score	High	9.0	4.0	GA Adaptive	33.057	0.0000
24	Phase2	Study_Time	Overall	394.0	303.0	GA Adaptive	338.426	0.0000
25	Phase2	Study_Time	Low	386.0	305.0	GA Adaptive	121.461	0.0000
26	Phase2	Study_Time	Medium	403.0	307.0	GA Adaptive	101.102	0.0000
27	Phase2	Study_Time	High	392.0	298.0	GA Adaptive	124.455	0.0000
28	Phase2	Retention_Rate	Overall	90.0	80.0	GA Adaptive	420.318	0.0000
29	Phase2	Retention_Rate	Low	91.0	80.0	GA Adaptive	172.686	0.0000
30	Phase2	Retention_Rate	Medium	90.0	81.0	GA Adaptive	108.723	0.0000
31	Phase2	Retention_Rate	High	90.0	79.0	GA Adaptive	145.679	0.0000

In [261…

```python
# Display Phase 1 Post-Hoc Analysis Results
print("\nPhase 1 Post-Hoc Analysis Results (Tukey's HSD):\n")
for result in post_hoc_results:
    phase, metric, performance_level, tukey_result = result
    if phase == 'Phase1':  # Filter for Phase 1 results
        print(f"Metric: {metric}, Performance Level: {performance_level}")
        print(tukey_result)
        print("\n")
```

```
Phase 1 Post-Hoc Analysis Results (Tukey's HSD):

Metric: Post_Test_Quiz_Score, Performance Level: Overall
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
 group1  group2 meandiff p-adj lower  upper  reject
-------------------------------------------------
Group A Group B   4.6514   0.0 2.4755 6.8274   True
-------------------------------------------------


Metric: Post_Test_Quiz_Score, Performance Level: Low
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
 group1  group2 meandiff p-adj  lower  upper  reject
-------------------------------------------------
Group A Group B   3.5303 0.0129 0.7608 6.2998   True
-------------------------------------------------


Metric: Post_Test_Quiz_Score, Performance Level: Medium
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
 group1  group2 meandiff p-adj lower  upper  reject
-------------------------------------------------
Group A Group B   4.6673   0.0 2.7335 6.6012   True
-------------------------------------------------


Metric: Post_Test_Quiz_Score, Performance Level: High
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
 group1  group2 meandiff p-adj lower  upper reject
-------------------------------------------------
Group A Group B   5.6068   0.0 3.1086 8.105   True
-------------------------------------------------


Metric: Improvement_Score, Performance Level: Overall
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
 group1  group2 meandiff p-adj lower  upper  reject
-------------------------------------------------
Group A Group B   4.7423   0.0 3.7323 5.7523   True
-------------------------------------------------
```

Metric: Improvement_Score, Performance Level: Low
Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Group A	Group B	4.803	0.0	2.9121	6.6939	True

Metric: Improvement_Score, Performance Level: Medium
Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Group A	Group B	4.7849	0.0	3.1403	6.4296	True

Metric: Improvement_Score, Performance Level: High
Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Group A	Group B	4.6347	0.0	2.8779	6.3916	True

Metric: Study_Time, Performance Level: Overall
Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Group A	Group B	102.1261	0.0	92.43	111.8222	True

Metric: Study_Time, Performance Level: Low
Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Group A	Group B	84.9091	0.0	67.8974	101.9208	True

Metric: Study_Time, Performance Level: Medium
Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Group A	Group B	93.235	0.0	75.508	110.962	True

Metric: Study_Time, Performance Level: High
Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Group A	Group B	128.4597	0.0	113.3976	143.5218	True

Metric: Retention_Rate, Performance Level: Overall
Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Group A	Group B	9.609	0.0	8.646	10.5721	True

Metric: Retention_Rate, Performance Level: Low
Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Group A	Group B	10.5197	0.0	8.828	12.2114	True

```
Metric: Retention_Rate, Performance Level: Medium
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj lower   upper  reject
-----------------------------------------------------
Group A Group B   8.8371   0.0 7.3336 10.3407   True
-----------------------------------------------------


Metric: Retention_Rate, Performance Level: High
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj lower   upper  reject
-----------------------------------------------------
Group A Group B   9.4894   0.0 7.7075 11.2713   True
-----------------------------------------------------
```

In [262…

```python
# Display Phase 2 Post-Hoc Analysis Results
print("\nPhase 2 Post-Hoc Analysis Results (Tukey's HSD):\n")
for result in post_hoc_results:
    phase, metric, performance_level, tukey_result = result
    if phase == 'Phase2':  # Filter for Phase 2 results
        print(f"Metric: {metric}, Performance Level: {performance_level}")
        print(tukey_result)
        print("\n")
```

```
Phase 2 Post-Hoc Analysis Results (Tukey's HSD):

Metric: Post_Test_Quiz_Score, Performance Level: Overall
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj  lower  upper  reject
-----------------------------------------------------
Group A Group B  -0.5096 0.677 -2.9123 1.8932  False
-----------------------------------------------------


Metric: Post_Test_Quiz_Score, Performance Level: Low
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj   lower  upper reject
-----------------------------------------------------
Group A Group B  -2.0341 0.2411 -5.4512 1.383  False
-----------------------------------------------------


Metric: Post_Test_Quiz_Score, Performance Level: Medium
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj  lower upper  reject
-----------------------------------------------------
Group A Group B   0.0592 0.9606 -2.31 2.4285  False
-----------------------------------------------------


Metric: Post_Test_Quiz_Score, Performance Level: High
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj   lower  upper  reject
-----------------------------------------------------
Group A Group B   0.2889 0.8589 -2.9205 3.4982  False
-----------------------------------------------------


Metric: Improvement_Score, Performance Level: Overall
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj  lower   upper  reject
-----------------------------------------------------
Group A Group B  -5.2726   0.0 -6.2939 -4.2512   True
-----------------------------------------------------


Metric: Improvement_Score, Performance Level: Low
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1  group2 meandiff p-adj  lower   upper  reject
-----------------------------------------------------
Group A Group B  -5.8485   0.0 -7.7697 -3.9273   True
-----------------------------------------------------
```

Metric: Improvement_Score, Performance Level: Medium
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
===
 group1 group2 meandiff p-adj lower   upper reject

Group A Group B -4.6081   0.0 -6.1738 -3.0424   True

Metric: Improvement_Score, Performance Level: High
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
===
 group1 group2 meandiff p-adj lower   upper reject

Group A Group B   -5.369   0.0 -7.2166 -3.5214   True

Metric: Study_Time, Performance Level: Overall
  Multiple Comparison of Means - Tukey HSD, FWER=0.05
===
 group1 group2 meandiff p-adj   lower   upper   reject

Group A Group B -90.5315   0.0 -100.2065 -80.8565   True

Metric: Study_Time, Performance Level: Low
  Multiple Comparison of Means - Tukey HSD, FWER=0.05
===
 group1 group2 meandiff p-adj lower   upper   reject

Group A Group B -81.8333   0.0 -96.5234 -67.1433   True

Metric: Study_Time, Performance Level: Medium
  Multiple Comparison of Means - Tukey HSD, FWER=0.05
===
 group1 group2 meandiff p-adj   lower   upper   reject

Group A Group B -96.2186   0.0 -115.1489 -77.2882   True

Metric: Study_Time, Performance Level: High
  Multiple Comparison of Means - Tukey HSD, FWER=0.05
===
 group1 group2 meandiff p-adj   lower   upper   reject

Group A Group B -93.5689   0.0 -110.1635 -76.9743   True

Metric: Retention_Rate, Performance Level: Overall
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
===
 group1 group2 meandiff p-adj lower   upper reject

Group A Group B -10.6298   0.0 -11.6492 -9.6105   True

Metric: Retention_Rate, Performance Level: Low
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
===
 group1 group2 meandiff p-adj lower   upper reject

Group A Group B -11.7679   0.0 -13.5395 -9.9962   True

Metric: Retention_Rate, Performance Level: Medium
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
===
 group1 group2 meandiff p-adj lower   upper reject

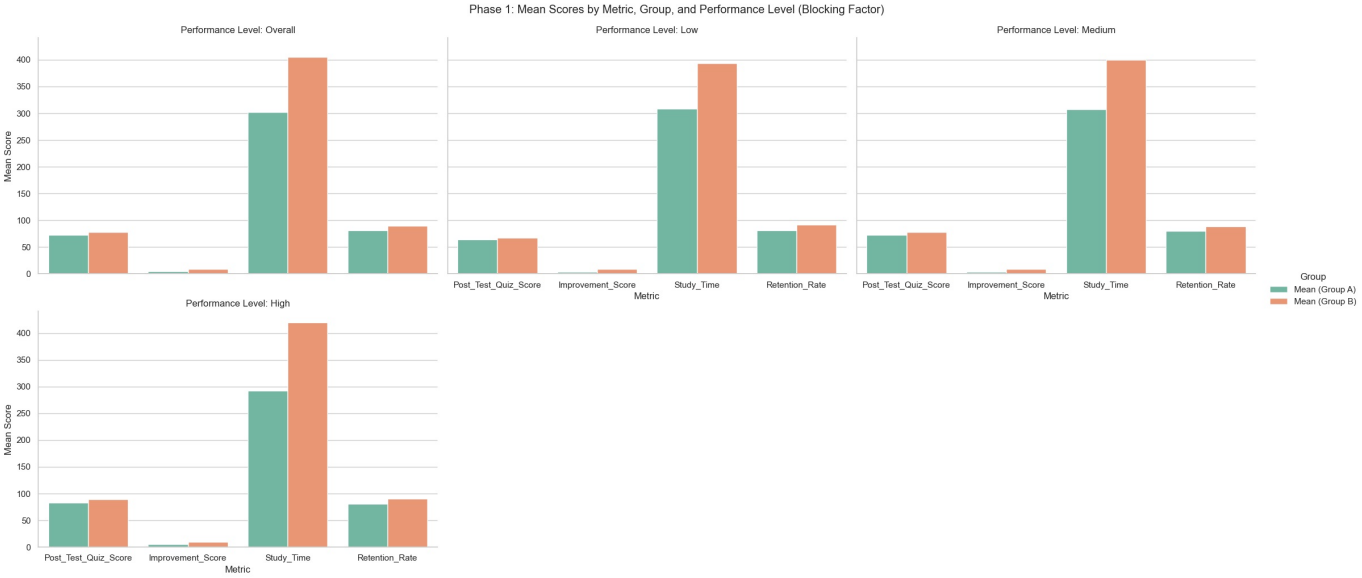
Group A Group B -9.3258   0.0 -11.0951 -7.5565   True

Metric: Retention_Rate, Performance Level: High
 Multiple Comparison of Means - Tukey HSD, FWER=0.05

```
=====================================================
 group1  group2 meandiff p-adj   lower    upper  reject
-----------------------------------------------------
Group A Group B -10.8012   0.0 -12.5718 -9.0307   True
-----------------------------------------------------
```

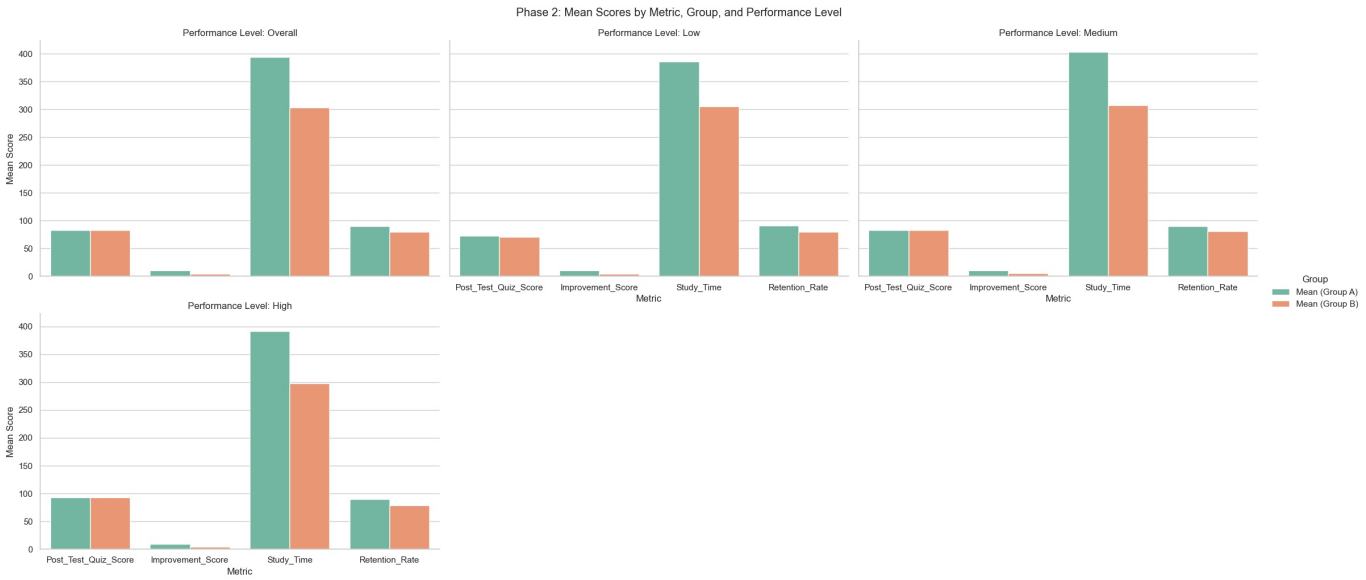
In [263…

```python
# Melt the Phase 1 results DataFrame for easier plotting
phase1_melted = phase1_results.melt(
    id_vars=['Metric', 'Performance Level'],
    value_vars=['Mean (Group A)', 'Mean (Group B)'],
    var_name='Group',
    value_name='Mean Score'
)

# Faceted bar plot for Phase 1 by performance level
g = sns.FacetGrid(phase1_melted, col='Performance Level', col_wrap=3, height=5, aspect=1.5)
g.map_dataframe(sns.barplot, x='Metric', y='Mean Score', hue='Group', palette='Set2')
g.set_axis_labels('Metric', 'Mean Score')
g.set_titles('Performance Level: {col_name}')
g.add_legend(title='Group')
plt.suptitle('Phase 1: Mean Scores by Metric, Group, and Performance Level (Blocking Factor)', y=1.02)
plt.show()
```



In [264…

```python
# Melt the Phase 2 results DataFrame for easier plotting
phase2_melted = phase2_results.melt(
    id_vars=['Metric', 'Performance Level'],
    value_vars=['Mean (Group A)', 'Mean (Group B)'],
    var_name='Group',
    value_name='Mean Score'
)

# Faceted bar plot for Phase 2 by performance level
g = sns.FacetGrid(phase2_melted, col='Performance Level', col_wrap=3, height=5, aspect=1.5)
g.map_dataframe(sns.barplot, x='Metric', y='Mean Score', hue='Group', palette='Set2')
g.set_axis_labels('Metric', 'Mean Score')
g.set_titles('Performance Level: {col_name}')
g.add_legend(title='Group')
plt.suptitle('Phase 2: Mean Scores by Metric, Group, and Performance Level', y=1.02)
plt.show()
```



5.2.6 Cross-Over RCT Between-Subject Comparison - Chi-Square Analysis of Dropout Rate (Blocking Factor)

In [265…

```
# Initialize a list to store results
chi_square_results = []

# Function to perform Chi-Square analysis for a given performance level
def chi_square_analysis(performance_level=None):
    # Filter data by performance level (if provided)
    if performance_level:
        data = clean_cross_over_rct_data[clean_cross_over_rct_data['Performance_Level'] == performance_level]
    else:
        data = clean_cross_over_rct_data

    # Create a contingency table comparing dropout rates across phases within each group
    dropout_contingency_table = pd.DataFrame({
        'Phase 1 Dropout (Group A)': data.loc[data['Group'] == 'A', 'Phase1_Dropout_Rate'].value_counts(),
        'Phase 2 Dropout (Group A)': data.loc[data['Group'] == 'A', 'Phase2_Dropout_Rate'].value_counts(),
        'Phase 1 Dropout (Group B)': data.loc[data['Group'] == 'B', 'Phase1_Dropout_Rate'].value_counts(),
        'Phase 2 Dropout (Group B)': data.loc[data['Group'] == 'B', 'Phase2_Dropout_Rate'].value_counts()
    }).fillna(0)

    # Perform Chi-Square Test for each phase comparison
    chi2_stat_phase1, p_value_phase1, dof_phase1, expected_phase1 = chi2_contingency(
        dropout_contingency_table[['Phase 1 Dropout (Group A)', 'Phase 1 Dropout (Group B)']]
    )

    chi2_stat_phase2, p_value_phase2, dof_phase2, expected_phase2 = chi2_contingency(
        dropout_contingency_table[['Phase 2 Dropout (Group A)', 'Phase 2 Dropout (Group B)']]
    )

    # Calculate mean dropout rates for each group in each phase
    dropout_rates = {
        'Phase 1': {
            'Group A': data.loc[data['Group'] == 'A', 'Phase1_Dropout_Rate'].mean(),
            'Group B': data.loc[data['Group'] == 'B', 'Phase1_Dropout_Rate'].mean()
        },
        'Phase 2': {
            'Group A': data.loc[data['Group'] == 'A', 'Phase2_Dropout_Rate'].mean(),
            'Group B': data.loc[data['Group'] == 'B', 'Phase2_Dropout_Rate'].mean()
        }
    }

    # Determine the better group and indicate Adaptive or Static
    def get_better_group(phase, group_a_rate, group_b_rate):
        if group_a_rate < group_b_rate:
            return "Group A (Static)" if phase == 'Phase 1' else "Group A (Adaptive)"
        else:
            return "Group B (Adaptive)" if phase == 'Phase 1' else "Group B (Static)"

    better_group_phase1 = get_better_group('Phase 1', dropout_rates['Phase 1']['Group A'], dropout_rates['Phase
    better_group_phase2 = get_better_group('Phase 2', dropout_rates['Phase 2']['Group A'], dropout_rates['Phase

    # Append results to the list
    chi_square_results.append([
        performance_level if performance_level else 'Overall',
        dropout_rates['Phase 1']['Group A'], dropout_rates['Phase 1']['Group B'], better_group_phase1,
        dropout_rates['Phase 2']['Group A'], dropout_rates['Phase 2']['Group B'], better_group_phase2,
        chi2_stat_phase1, p_value_phase1, chi2_stat_phase2, p_value_phase2
```

```
    ])

# Perform Chi-Square analysis for overall data and each performance level
chi_square_analysis()  # Overall analysis
for performance_level in ['Low', 'Medium', 'High']:
    chi_square_analysis(performance_level)

# Convert results to DataFrame
chi_square_results_df = pd.DataFrame(
    chi_square_results,
    columns=[
        'Performance Level', 'Phase 1 Dropout (Group A)', 'Phase 1 Dropout (Group B)', 'Better Group (Phase 1)'
        'Phase 2 Dropout (Group A)', 'Phase 2 Dropout (Group B)', 'Better Group (Phase 2)',
        'Chi-Square Statistic (Phase 1)', 'p-value (Phase 1)', 'Chi-Square Statistic (Phase 2)', 'p-value (Phas
    ]
)

# Add significance columns
chi_square_results_df['Significance (Phase 1)'] = chi_square_results_df['p-value (Phase 1)'].apply(
    lambda x: 'Significant' if x < 0.05 else 'Not Significant'
)
chi_square_results_df['Significance (Phase 2)'] = chi_square_results_df['p-value (Phase 2)'].apply(
    lambda x: 'Significant' if x < 0.05 else 'Not Significant'
)

# Split the results DataFrame into Phase 1 and Phase 2
phase1_results_df = chi_square_results_df[[
    'Performance Level', 'Phase 1 Dropout (Group A)', 'Phase 1 Dropout (Group B)', 'Better Group (Phase 1)',
    'Chi-Square Statistic (Phase 1)', 'p-value (Phase 1)', 'Significance (Phase 1)'
]]

phase2_results_df = chi_square_results_df[[
    'Performance Level', 'Phase 2 Dropout (Group A)', 'Phase 2 Dropout (Group B)', 'Better Group (Phase 2)',
    'Chi-Square Statistic (Phase 2)', 'p-value (Phase 2)', 'Significance (Phase 2)'
]]

# Rename columns for clarity
phase1_results_df.columns = [
    'Performance Level', 'Dropout (Group A)', 'Dropout (Group B)', 'Better Group',
    'Chi-Square Statistic', 'p-value', 'Significance'
]

phase2_results_df.columns = [
    'Performance Level', 'Dropout (Group A)', 'Dropout (Group B)', 'Better Group',
    'Chi-Square Statistic', 'p-value', 'Significance'
]

# Display Phase 1 Results
print("\n5.2.6 (Blocking Factor) Cross-Over RCT Between-Subject Comparison - Chi-Square Analysis of Dropout Rat
display(phase1_results_df)

# Display Phase 2 Results
print("\n5.2.6 (Blocking Factor) Cross-Over RCT Between-Subject Comparison - Chi-Square Analysis of Dropout Rat
display(phase2_results_df)
```

```
5.2.6 (Blocking Factor) Cross-Over RCT Between-Subject Comparison - Chi-Square Analysis of Dropout Rate (Phase 1
)
```

	Performance Level	Dropout (Group A)	Dropout (Group B)	Better Group	Chi-Square Statistic	p-value	Significance
0	Overall	0.238579	0.100503	Group B (Adaptive)	12.462748	0.000415	Significant
1	Low	0.227273	0.121212	Group B (Adaptive)	1.895493	0.168584	Not Significant
2	Medium	0.242424	0.074627	Group B (Adaptive)	5.835073	0.015710	Significant
3	High	0.246154	0.106061	Group B (Adaptive)	3.525181	0.060443	Not Significant

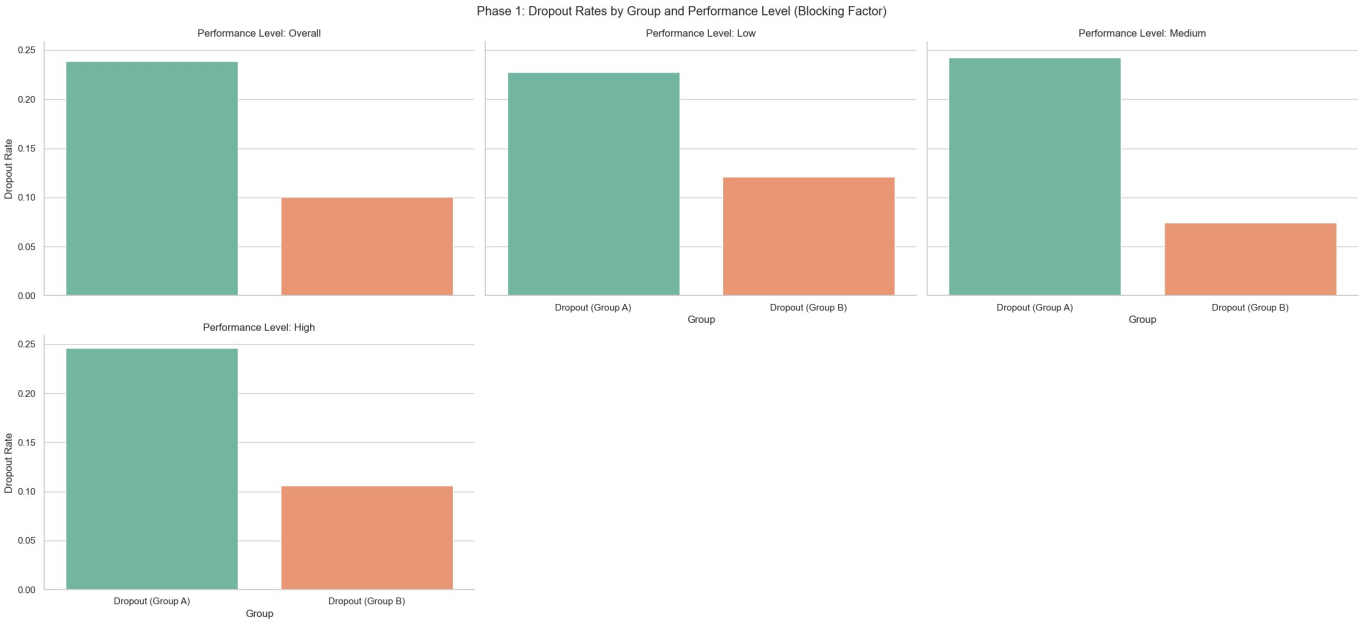
```
5.2.6 (Blocking Factor) Cross-Over RCT Between-Subject Comparison - Chi-Square Analysis of Dropout Rate (Phase 2
)
```

	Performance Level	Dropout (Group A)	Dropout (Group B)	Better Group	Chi-Square Statistic	p-value	Significance
0	Overall	0.055838	0.296482	Group A (Adaptive)	37.759661	8.001943e-10	Significant
1	Low	0.060606	0.303030	Group A (Adaptive)	11.458333	7.117411e-04	Significant
2	Medium	0.030303	0.238806	Group A (Adaptive)	10.634145	1.110186e-03	Significant
3	High	0.076923	0.348485	Group A (Adaptive)	12.799978	3.466235e-04	Significant

In [266…

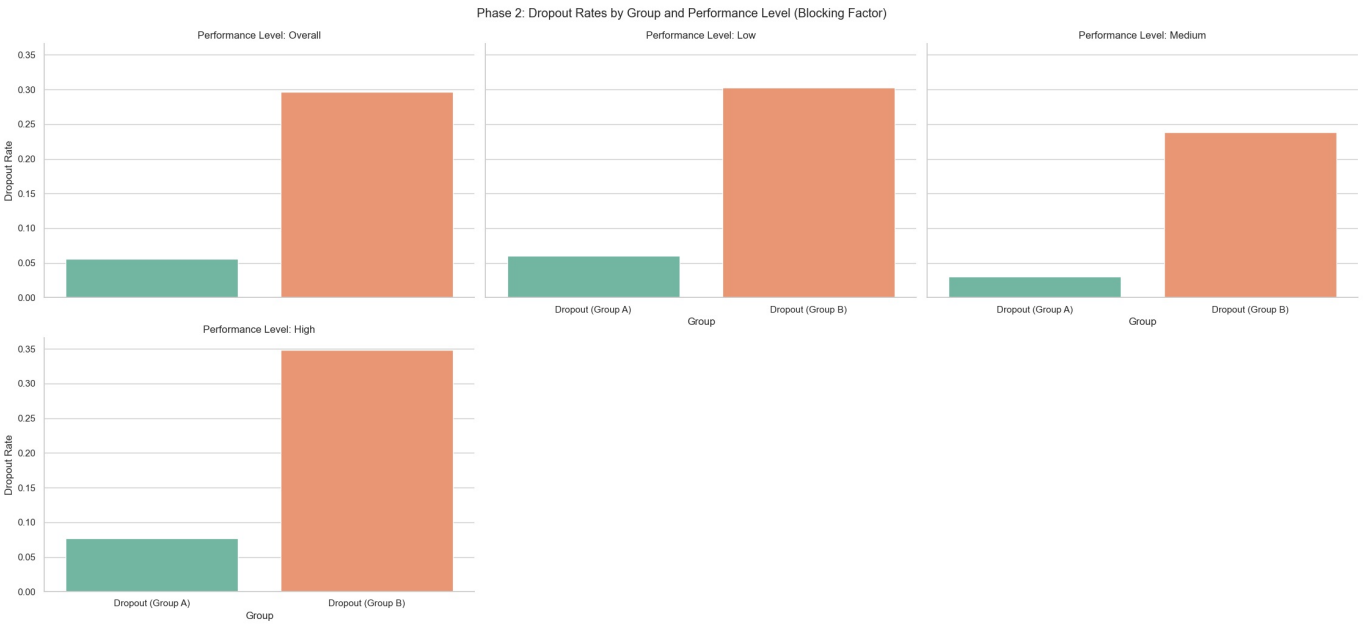
```
# Melt the Phase 1 results DataFrame for easier plotting
phase1_melted = phase1_results_df.melt(
    id_vars=['Performance Level'],
    value_vars=['Dropout (Group A)', 'Dropout (Group B)'],
```

```
    var_name='Group',
    value_name='Dropout Rate'
)

# Faceted bar plot for Phase 1 by performance level
g = sns.FacetGrid(phase1_melted, col='Performance Level', col_wrap=3, height=5, aspect=1.5)
g.map_dataframe(sns.barplot, x='Group', y='Dropout Rate', palette='Set2')
g.set_axis_labels('Group', 'Dropout Rate')
g.set_titles('Performance Level: {col_name}')
plt.suptitle('Phase 1: Dropout Rates by Group and Performance Level (Blocking Factor)', y=1.02)
plt.show()
```



In [267…

```
# Melt the Phase 2 results DataFrame for easier plotting
phase2_melted = phase2_results_df.melt(
    id_vars=['Performance Level'],
    value_vars=['Dropout (Group A)', 'Dropout (Group B)'],
    var_name='Group',
    value_name='Dropout Rate'
)

# Faceted bar plot for Phase 2 by performance level
g = sns.FacetGrid(phase2_melted, col='Performance Level', col_wrap=3, height=5, aspect=1.5)
g.map_dataframe(sns.barplot, x='Group', y='Dropout Rate', palette='Set2')
g.set_axis_labels('Group', 'Dropout Rate')
g.set_titles('Performance Level: {col_name}')
plt.suptitle('Phase 2: Dropout Rates by Group and Performance Level (Blocking Factor)', y=1.02)
plt.show()
```



5.2.7 Cross-Over RCT Between-Subject Comparison Analysis (Blocking Factor)

Phase 1 Results

In Phase 1, Group B (Adaptive) consistently outperformed Group A (Static) across all measured metrics, including **Post-Test Quiz Scores**, **Improvement Scores**, **Study Time**, and **Retention Rate**.

- **Post-Test Quiz Scores** showed significant differences overall and across all performance levels (Low, Medium, High), with Group B achieving higher mean scores in every category (F-statistics ranging from 6.36 to 22.80, $p < 0.05$).
- **Improvement Scores** demonstrated a substantial increase in Group B across all performance levels, particularly in the high-performance group ($F = 27.24$, $p < 0.0001$).
- **Study Time** was also higher in Group B, suggesting increased engagement (Overall $F = 428.79$, $p < 0.0001$).
- **Retention Rate** was significantly higher for Group B at all levels (F-statistics from 111.02 to 384.81, $p < 0.0001$).

Blocking Factor: Dropout Rate

A Chi-Square analysis revealed significant differences in dropout rates between the two groups:

- **Overall**, Group B exhibited a lower dropout rate ($\chi^2 = 12.46$, $p = 0.0004$).
- Significant differences were observed in the **Medium** performance level ($\chi^2 = 5.83$, $p = 0.0157$).
- The **Low** and **High** performance levels did not show statistically significant differences but trended in favor of Group B.

Phase 2 Results

In Phase 2, after the groups switched treatments, Group A (now Adaptive) outperformed Group B (now Static) in several metrics:

- **Post-Test Quiz Scores** did not show significant differences across most performance levels, suggesting a potential ceiling effect or learning stabilization after Phase 1.
- **Improvement Scores**, **Study Time**, and **Retention Rate** were significantly higher for Group A (Adaptive) across all performance levels, with substantial F-statistics (e.g., $F = 420.32$ for Overall Retention Rate, $p < 0.0001$).

Blocking Factor: Dropout Rate

Chi-Square analysis revealed significant differences in dropout rates in favor of Group A (Adaptive):

- **Overall**, Group A showed a significantly lower dropout rate ($\chi^2 = 37.76$, $p < 0.0001$).
- All performance levels (Low, Medium, High) exhibited significant differences ($p < 0.001$), reinforcing the positive effect of adaptive learning on retention and continued participation.

Conclusion: Blocking Factor Effects

The analysis suggests that adaptive learning systems significantly improve performance outcomes (quiz scores, improvement scores, study time, and retention rates) across all performance levels in both phases. The blocking factor analysis of dropout rates further supports this finding, showing that participants in adaptive learning conditions were more likely to stay engaged regardless of their initial performance level.

The crossover design effectively demonstrates the robustness of these results by showing consistent patterns across both groups after switching interventions, highlighting the efficacy of adaptive learning systems over static learning environments.

5.3.3 Cross-Over RCT Randomization (Continuous Covariates)

In [268…

```
from sklearn.utils import shuffle

# Step 1: Collect Baseline Measures (Pre-Intervention Data)
# Assuming generate_baseline_data is a function that generates the dataset
cross_over_rct_data = generate_baseline_data(cross_over_rct_sample_size)

# Step 2: Define Continuous Covariates (GPA, Baseline_Quiz_Score, Tech_Savviness_Score)
covariates = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']
covariate_data = cross_over_rct_data[covariates]

# Step 3: Matching with Nearest Neighbors
# Initialize NearestNeighbors to find closest matches based on covariates
nn = NearestNeighbors(n_neighbors=2, algorithm='ball_tree', metric='euclidean')
nn.fit(covariate_data)

# Find the nearest neighbors
distances, indices = nn.kneighbors(covariate_data)

# Step 4: Assign to Blocks
# Create a new column for blocks, starting with 'None'
cross_over_rct_data['Block'] = None

# Use a greedy matching algorithm to assign blocks
assigned = set()  # Track participants who have already been assigned to a block
block_id = 0

for i in range(len(cross_over_rct_data)):
```

```
    if i not in assigned:
        # Find the nearest neighbor for participant i
        neighbor_index = indices[i, 1]  # The first neighbor is the participant themselves

        # Ensure the neighbor hasn't already been assigned
        if neighbor_index not in assigned:
            # Assign both participants to the same block
            cross_over_rct_data.loc[[i, neighbor_index], 'Block'] = block_id
            assigned.add(i)
            assigned.add(neighbor_index)
            block_id += 1

# Handle the case where the number of participants is odd
if len(cross_over_rct_data) % 2 != 0:
    # Find the unassigned participant
    unassigned = cross_over_rct_data[cross_over_rct_data['Block'].isna()].index[0]
    # Assign the unassigned participant to a new block
    cross_over_rct_data.loc[unassigned, 'Block'] = block_id

# Step 5: Blocked Random Assignment (Cross-Over Design)
# Initialize the 'Group' column
cross_over_rct_data['Group'] = None

# Perform random assignment within each block
np.random.seed(18)  # Set seed for reproducibility
for block in cross_over_rct_data['Block'].unique():
    # Filter data for the current block
    block_data = cross_over_rct_data[cross_over_rct_data['Block'] == block]

    # Shuffle the indices for randomization
    shuffled_indices = np.random.permutation(block_data.index)

    # If the block has an odd number of participants, assign the last one randomly
    if len(shuffled_indices) % 2 != 0:
        half = len(shuffled_indices) // 2
        cross_over_rct_data.loc[shuffled_indices[:half], 'Group'] = 'A'
        cross_over_rct_data.loc[shuffled_indices[half:-1], 'Group'] = 'B'
        # Randomly assign the last participant to Group A or B
        cross_over_rct_data.loc[shuffled_indices[-1], 'Group'] = np.random.choice(['A', 'B'])
    else:
        # Split the shuffled indices into two equal groups
        half = len(shuffled_indices) // 2
        cross_over_rct_data.loc[shuffled_indices[:half], 'Group'] = 'A'
        cross_over_rct_data.loc[shuffled_indices[half:], 'Group'] = 'B'

# Drop the temporary 'Block' column as it is no longer needed
cross_over_rct_data.drop(columns=['Block'], inplace=True)

# Shuffle dataset to remove order bias
cross_over_rct_data = shuffle(cross_over_rct_data, random_state=18).reset_index(drop=True)

# Ensure all students are assigned to Group A or B
if cross_over_rct_data['Group'].isna().any():
    unassigned_students = cross_over_rct_data[cross_over_rct_data['Group'].isna()].index
    cross_over_rct_data.loc[unassigned_students, 'Group'] = np.random.choice(['A', 'B'], size=len(unassigned_st

# Display the first few rows of the dataset
cross_over_rct_data.head()
```

Out[268…

	Student_ID	Age	GPA	Study_Hours_Per_Week	Tech_Savviness_Score	Learning_Style	Baseline_Quiz_Score	Attention_Span	M
0	238	19	3.0	55	11	Auditory	60	20	
1	373	23	3.4	55	5	Visual	52	29	
2	48	21	3.3	69	3	Visual	79	36	
3	19	26	3.5	39	3	Kinesthetic	60	13	
4	184	22	3.3	52	2	Auditory	58	18	

5.3.4: Cross-Over RCT Implement Intervention (Continuous Covariates)

In [269…

```
# Step 3: Implement Intervention (Post-Intervention Data)
np.random.seed(18)

# Generate post-test metrics for Phase 1 (Static vs. Adaptive)
cross_over_rct_data['Phase1_Post_Test_Quiz_Score'] = (
    cross_over_rct_data['Baseline_Quiz_Score'] +
    np.where(cross_over_rct_data['Group'] == 'A',
             np.random.normal(5, 5, len(cross_over_rct_data)),  # Static first (Group A)
             np.random.normal(10, 5, len(cross_over_rct_data)))  # Adaptive first (Group B)
```

```
).astype(int)

cross_over_rct_data['Phase1_Improvement_Score'] = (
    cross_over_rct_data['Phase1_Post_Test_Quiz_Score'] - cross_over_rct_data['Baseline_Quiz_Score']
).astype(int)

cross_over_rct_data['Phase1_Study_Time'] = np.where(
    cross_over_rct_data['Group'] == 'A',
    np.random.normal(300, 50, len(cross_over_rct_data)),  # Static first (Group A)
    np.random.normal(400, 50, len(cross_over_rct_data))   # Adaptive first (Group B)
).astype(int)

cross_over_rct_data['Phase1_Retention_Rate'] = np.where(
    cross_over_rct_data['Group'] == 'A',
    np.random.normal(80, 5, len(cross_over_rct_data)),  # Static first (Group A)
    np.random.normal(90, 5, len(cross_over_rct_data))   # Adaptive first (Group B)
).round(2)

cross_over_rct_data['Phase1_Dropout_Rate'] = np.where(
    cross_over_rct_data['Group'] == 'A',
    np.random.choice([0, 1], len(cross_over_rct_data), p=[0.75, 0.25]),  # Static first (Group A)
    np.random.choice([0, 1], len(cross_over_rct_data), p=[0.9, 0.1])     # Adaptive first (Group B)
).round(2)

# Step 4: Switch Conditions After Washout Period (Cross-Over Phase)
# Swap Group Assignments: A → B, B → A
cross_over_rct_data['Group_Cross_Over'] = cross_over_rct_data['Group'].map({'A': 'B', 'B': 'A'})

# Generate post-test metrics for Phase 2 (Adaptive vs. Static)
cross_over_rct_data['Phase2_Post_Test_Quiz_Score'] = (
    cross_over_rct_data['Phase1_Post_Test_Quiz_Score'] +
    np.where(cross_over_rct_data['Group_Cross_Over'] == 'A',
             np.random.normal(10, 5, len(cross_over_rct_data)),  # Adaptive second (Group A)
             np.random.normal(5, 5, len(cross_over_rct_data)))   # Static second (Group B)
).astype(int)

cross_over_rct_data['Phase2_Improvement_Score'] = (
    cross_over_rct_data['Phase2_Post_Test_Quiz_Score'] - cross_over_rct_data['Phase1_Post_Test_Quiz_Score']
).astype(int)

cross_over_rct_data['Phase2_Study_Time'] = np.where(
    cross_over_rct_data['Group_Cross_Over'] == 'A',
    np.random.normal(400, 50, len(cross_over_rct_data)),  # Adaptive second (Group A)
    np.random.normal(300, 50, len(cross_over_rct_data))   # Static second (Group B)
).astype(int)

cross_over_rct_data['Phase2_Retention_Rate'] = np.where(
    cross_over_rct_data['Group_Cross_Over'] == 'A',
    np.random.normal(90, 5, len(cross_over_rct_data)),  # Adaptive second (Group A)
    np.random.normal(80, 5, len(cross_over_rct_data))   # Static second (Group B)
).round(2)

cross_over_rct_data['Phase2_Dropout_Rate'] = np.where(
    cross_over_rct_data['Group_Cross_Over'] == 'A',
    np.random.choice([0, 1], len(cross_over_rct_data), p=[0.9, 0.1]),  # Adaptive second (Group A)
    np.random.choice([0, 1], len(cross_over_rct_data), p=[0.75, 0.25]) # Static second (Group B)
).round(2)

# Drop temporary Group_Cross_Over column
cross_over_rct_data.drop(columns=['Group_Cross_Over'], inplace=True)
```

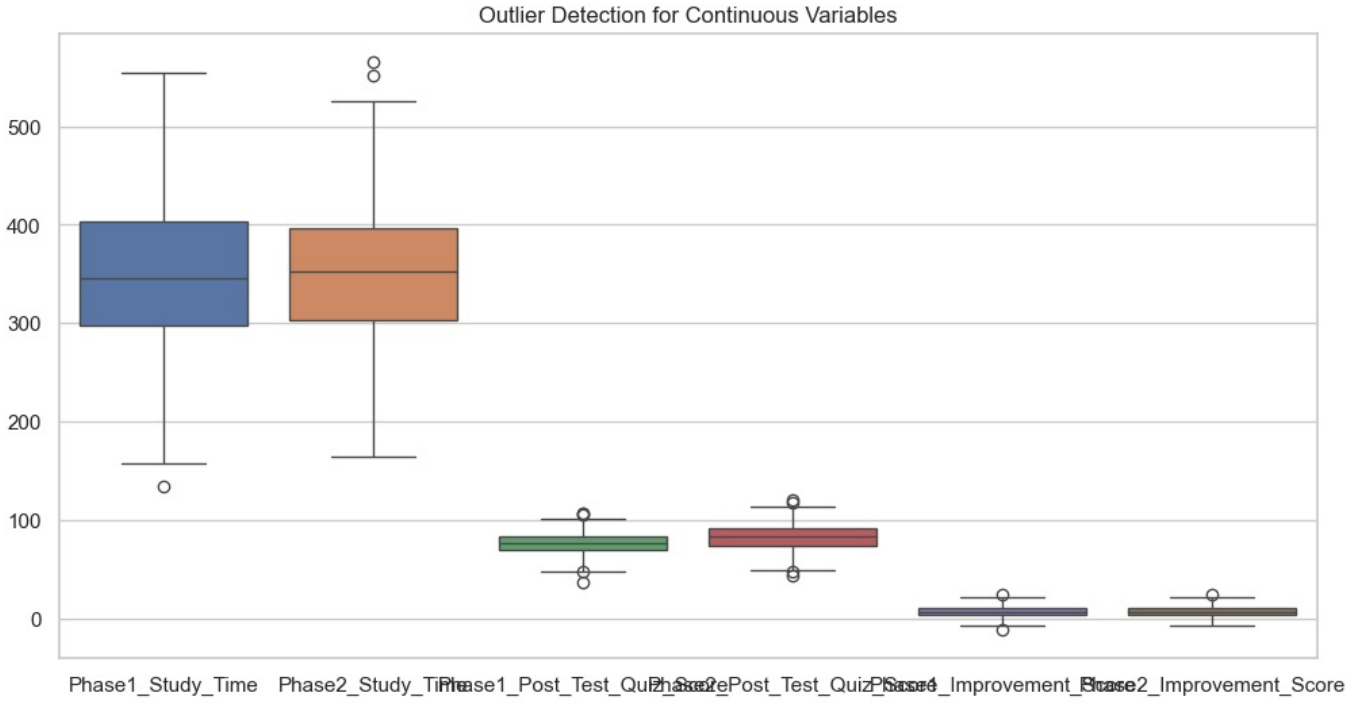
5.3.5: Cross-Over RCT Data Cleaning (Continuous Covariates)

In [270…

```
# 1. Check for Missing Data
print("Missing Values Check:\n")
print(cross_over_rct_data.isnull().sum())

# Replace placeholder codes (e.g., -18) with NaN if present
cross_over_rct_data.replace(-18, np.nan, inplace=True)

# Drop rows with critical missing values
critical_cols = ['Phase1_Post_Test_Quiz_Score', 'Phase2_Post_Test_Quiz_Score', 'Phase1_Retention_Rate',
                 'Phase2_Retention_Rate', 'Phase1_Dropout_Rate', 'Phase2_Dropout_Rate', 'Group']
cross_over_rct_data.dropna(subset=critical_cols, inplace=True)

# 2. Remove Duplicates
print(f"\nInitial Shape: {cross_over_rct_data.shape}")
cross_over_rct_data.drop_duplicates(subset=['Student_ID'], keep='first', inplace=True)
print(f"Post-Deduplication Shape: {cross_over_rct_data.shape}")

# 3. Verify Randomization Balance
print("\nGroup Balance:")
```

```
print(cross_over_rct_data['Group'].value_counts())

# Compare baseline covariates between groups
print("\nBaseline Covariate Balance:")
baseline_cols = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']
for col in baseline_cols:
    group_a = cross_over_rct_data[cross_over_rct_data['Group'] == 'A'][col]
    group_b = cross_over_rct_data[cross_over_rct_data['Group'] == 'B'][col]
    t_stat, p_value = ttest_ind(group_a, group_b, nan_policy='omit')
    print(f"{col}: t = {t_stat:.2f}, p = {p_value:.4f}")

# 4. Detect and Handle Outliers
continuous_vars = ['Phase1_Study_Time', 'Phase2_Study_Time', 'Phase1_Post_Test_Quiz_Score', 'Phase2_Post_Test_Q

# Visualize Outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=cross_over_rct_data[continuous_vars])
plt.title("Outlier Detection for Continuous Variables")
plt.show()

# Cap outliers using IQR
for var in continuous_vars:
    q1 = cross_over_rct_data[var].quantile(0.25)
    q3 = cross_over_rct_data[var].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    cross_over_rct_data[var] = np.where(
        cross_over_rct_data[var] < lower_bound, lower_bound,
        np.where(cross_over_rct_data[var] > upper_bound, upper_bound, cross_over_rct_data[var])
    )

# 5. Validate Variable Ranges/Types

# Ensure binary variables are 0/1
cross_over_rct_data['Phase1_Dropout_Rate'] = cross_over_rct_data['Phase1_Dropout_Rate'].apply(lambda x: 1 if x =
cross_over_rct_data['Phase2_Dropout_Rate'] = cross_over_rct_data['Phase2_Dropout_Rate'].apply(lambda x: 1 if x =

# Standardize categorical variables
cross_over_rct_data['Learning_Style'] = cross_over_rct_data['Learning_Style'].str.strip().str.title()

# Ensure valid percentage ranges
cross_over_rct_data['Phase1_Retention_Rate'] = cross_over_rct_data['Phase1_Retention_Rate'].clip(0, 100)
cross_over_rct_data['Phase2_Retention_Rate'] = cross_over_rct_data['Phase2_Retention_Rate'].clip(0, 100)
# 6. Save Cleaned Data
clean_cross_over_rct_data = cross_over_rct_data.copy()

print("\nData cleaning complete. Cleaned data saved as clean_cross_over_rct_data")
```

```
Missing Values Check:

Student_ID                    0
Age                           0
GPA                           0
Study_Hours_Per_Week          0
Tech_Savviness_Score          0
Learning_Style                0
Baseline_Quiz_Score           0
Attention_Span                0
Motivation_Level              0
Prior_Online_Exp              0
Group                         0
Phase1_Post_Test_Quiz_Score   0
Phase1_Improvement_Score      0
Phase1_Study_Time             0
Phase1_Retention_Rate         0
Phase1_Dropout_Rate           0
Phase2_Post_Test_Quiz_Score   0
Phase2_Improvement_Score      0
Phase2_Study_Time             0
Phase2_Retention_Rate         0
Phase2_Dropout_Rate           0
dtype: int64

Initial Shape: (396, 21)
Post-Deduplication Shape: (396, 21)

Group Balance:
Group
B    203
A    193
Name: count, dtype: int64

Baseline Covariate Balance:
GPA: t = 0.07, p = 0.9412
Baseline_Quiz_Score: t = 0.21, p = 0.8360
Tech_Savviness_Score: t = 0.60, p = 0.5489
```




```
Data cleaning complete. Cleaned data saved as clean_cross_over_rct_data
```

5.3.6: Cross-Over RCT Perform Within Subject Analysis (Continuous Covariates)

In [271…

```python
from statsmodels.formula.api import mixedlm

# List of metrics to analyze (updated column names)
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']

# Map metrics to their corresponding column names in the dataset
metric_columns = {
    'Post_Test_Quiz_Score': ['Phase1_Post_Test_Quiz_Score', 'Phase2_Post_Test_Quiz_Score'],
    'Improvement_Score': ['Phase1_Improvement_Score', 'Phase2_Improvement_Score'],
    'Study_Time': ['Phase1_Study_Time', 'Phase2_Study_Time'],
    'Retention_Rate': ['Phase1_Retention_Rate', 'Phase2_Retention_Rate']
}
```

```python
# List of continuous covariates
covariates = ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']

# Initialize a list to store results
within_subject_results = []

# Function to perform within-subject comparison with continuous covariates
def within_subject_comparison(group, metric):
    # Extract data for the group
    data = clean_cross_over_rct_data[clean_cross_over_rct_data['Group'] == group]

    # Get the corresponding column names for the metric
    phase1_col, phase2_col = metric_columns[metric]

    # Define the formula for the mixed-effects model
    formula = f'{phase1_col} ~ {phase2_col} + ' + ' + '.join(covariates)

    # Fit the mixed-effects model
    model = mixedlm(formula, data=data, groups=data['Student_ID']).fit()

    # Extract phase means
    phase1_mean = data[phase1_col].mean()
    phase2_mean = data[phase2_col].mean()

    # Determine which phase is better based on the group and the metric
    if group == 'A':  # Group A: Static -> Adaptive
        better_phase = "GA_P2 Adaptive" if phase1_mean < phase2_mean else "GA_P1 Static"
    else:  # Group B: Adaptive -> Static
        better_phase = "GB_P2 Static" if phase1_mean < phase2_mean else "GB_P1 Adaptive"

    # Extract p-value for the phase effect
    p_value_phase = model.pvalues[phase2_col]

    # Determine statistical significance for phase effect
    significance_phase = "Significant" if p_value_phase < 0.05 else "Not Significant"

    # Extract coefficients and p-values for covariates
    covariate_results = []
    for cov in covariates:
        covariate_coef = model.params[cov]
        covariate_p_value = model.pvalues[cov]
        covariate_significance = "Significant" if covariate_p_value < 0.05 else "Not Significant"
        covariate_results.append((cov, covariate_coef, covariate_p_value, covariate_significance))

    # Append results to the list
    within_subject_results.append([
        group, metric, round(phase1_mean, 3), round(phase2_mean, 3), better_phase,
        round(model.params[phase2_col], 3), p_value_phase, significance_phase,
        covariate_results  # Include covariate analysis results
    ])

# Perform Within-Subject Comparison for each group and metric
for metric in metrics:
    for group in ['A', 'B']:
        within_subject_comparison(group, metric)

# Convert results to DataFrame
within_subject_df = pd.DataFrame(
    within_subject_results,
    columns=['Group', 'Metric', 'Mean (Phase 1)', 'Mean (Phase 2)', 'Better Phase',
             'Phase Effect (Coefficient)', 'p-Value (Phase)', 'Significance (Phase)',
             'Covariate Analysis']
)

# Filter results by Group A and Group B
group_a_df = within_subject_df[within_subject_df['Group'] == 'A']
group_b_df = within_subject_df[within_subject_df['Group'] == 'B']

# Convert specific columns to integers in Group A
metrics_to_convert = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']
group_a_df.loc[group_a_df['Metric'].isin(metrics_to_convert), ['Mean (Phase 1)', 'Mean (Phase 2)']] = group_a_d
    group_a_df['Metric'].isin(metrics_to_convert), ['Mean (Phase 1)', 'Mean (Phase 2)']
].astype(int)

# Convert specific columns to integers in Group B
group_b_df.loc[group_b_df['Metric'].isin(metrics_to_convert), ['Mean (Phase 1)', 'Mean (Phase 2)']] = group_b_d
    group_b_df['Metric'].isin(metrics_to_convert), ['Mean (Phase 1)', 'Mean (Phase 2)']
].astype(int)

# Display Group A Results
print("\n5.3.6: Cross-Over RCT Within-Subject Comparison with Continuous Covariates: Group A (Static → Adaptive
display(group_a_df)
```

```
# Display Group B Results
print("\n5.3.6: Cross-Over RCT Within-Subject Comparison with Continuous Covariates: Group B (Adaptive → Static
display(group_b_df)

# Extract and display Continuous Covariates Analysis for Group A
group_a_covariates = []
for index, row in group_a_df.iterrows():
    for cov in row['Covariate Analysis']:
        group_a_covariates.append([row['Metric'], cov[0], cov[1], cov[2], cov[3]])

group_a_covariates_df = pd.DataFrame(
    group_a_covariates,
    columns=['Metric', 'Covariate', 'Coefficient', 'p-Value', 'Significance']
)

print("\n5.3.6: Continuous Covariates Analysis for Group A (Static → Adaptive)\n")
display(group_a_covariates_df)

# Extract and display Continuous Covariates Analysis for Group B
group_b_covariates = []
for index, row in group_b_df.iterrows():
    for cov in row['Covariate Analysis']:
        group_b_covariates.append([row['Metric'], cov[0], cov[1], cov[2], cov[3]])

group_b_covariates_df = pd.DataFrame(
    group_b_covariates,
    columns=['Metric', 'Covariate', 'Coefficient', 'p-Value', 'Significance']
)

print("\n5.3.6: Continuous Covariates Analysis for Group B (Adaptive → Static)\n")
display(group_b_covariates_df)
```

```
5.3.6: Cross-Over RCT Within-Subject Comparison with Continuous Covariates: Group A (Static → Adaptive)
```

	Group	Metric	Mean (Phase 1)	Mean (Phase 2)	Better Phase	Phase Effect (Coefficient)	p-Value (Phase)	Significance (Phase)	Covariate Analysis
0	A	Post_Test_Quiz_Score	73.0	77.0	GA_P2 Adaptive	0.539	1.080097e-57	Significant	[(GPA, 0.42412807963309274, 0.4635577420075304...
2	A	Improvement_Score	4.0	4.0	GA_P1 Static	0.073	3.571036e-01	Not Significant	[(GPA, 1.0994383226589142, 0.21524390229939094...
4	A	Study_Time	297.0	306.0	GA_P2 Adaptive	-0.010	8.896938e-01	Not Significant	[(GPA, -10.395745005855705, 0.0301495536999335...
6	A	Retention_Rate	80.0	80.0	GA_P2 Adaptive	-0.047	2.817783e-01	Not Significant	[(GPA, -1.06020848508785, 0.18626229837940744,...

```
5.3.6: Cross-Over RCT Within-Subject Comparison with Continuous Covariates: Group B (Adaptive → Static)
```

	Group	Metric	Mean (Phase 1)	Mean (Phase 2)	Better Phase	Phase Effect (Coefficient)	p-Value (Phase)	Significance (Phase)	Covariate Analysis
1	B	Post_Test_Quiz_Score	78.0	87.0	GB_P2 Static	0.556	4.469453e-69	Significant	[(GPA, -0.4641629956107181, 0.0198944334996393...
3	B	Improvement_Score	9.0	9.0	GB_P2 Static	0.104	1.722746e-01	Not Significant	[(GPA, -0.6269566684543287, 0.3418600208606659...
5	B	Study_Time	401.0	393.0	GB_P1 Adaptive	-0.010	8.902712e-01	Not Significant	[(GPA, -20.09514901266344, 6.033309210538271e-...
7	B	Retention_Rate	90.0	89.0	GB_P1 Adaptive	0.038	5.962551e-01	Not Significant	[(GPA, 1.037172250238958, 0.23640233463504445,...

```
5.3.6: Continuous Covariates Analysis for Group A (Static → Adaptive)
```

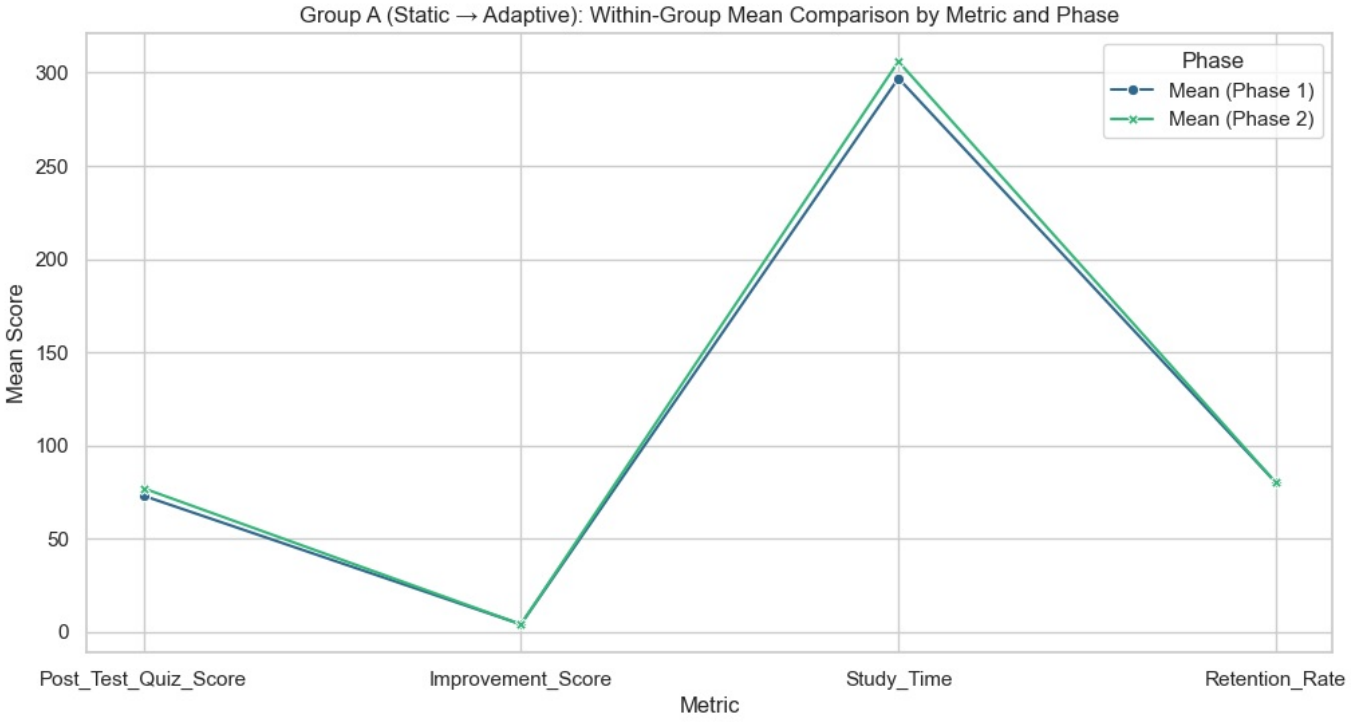
	Metric	Covariate	Coefficient	p-Value	Significance
0	Post_Test_Quiz_Score	GPA	0.424128	4.635577e-01	Not Significant
1	Post_Test_Quiz_Score	Baseline_Quiz_Score	0.384200	4.215530e-22	Significant
2	Post_Test_Quiz_Score	Tech_Savviness_Score	-0.058786	4.628084e-01	Not Significant
3	Improvement_Score	GPA	1.099438	2.152439e-01	Not Significant
4	Improvement_Score	Baseline_Quiz_Score	-0.073451	3.962651e-02	Significant
5	Improvement_Score	Tech_Savviness_Score	0.065358	5.955325e-01	Not Significant
6	Study_Time	GPA	-10.395745	3.014955e-02	Significant
7	Study_Time	Baseline_Quiz_Score	0.278750	3.775192e-01	Not Significant
8	Study_Time	Tech_Savviness_Score	-1.517798	2.244674e-01	Not Significant
9	Retention_Rate	GPA	-1.060208	1.862623e-01	Not Significant
10	Retention_Rate	Baseline_Quiz_Score	0.016686	6.308502e-01	Not Significant
11	Retention_Rate	Tech_Savviness_Score	-0.100664	4.223608e-01	Not Significant

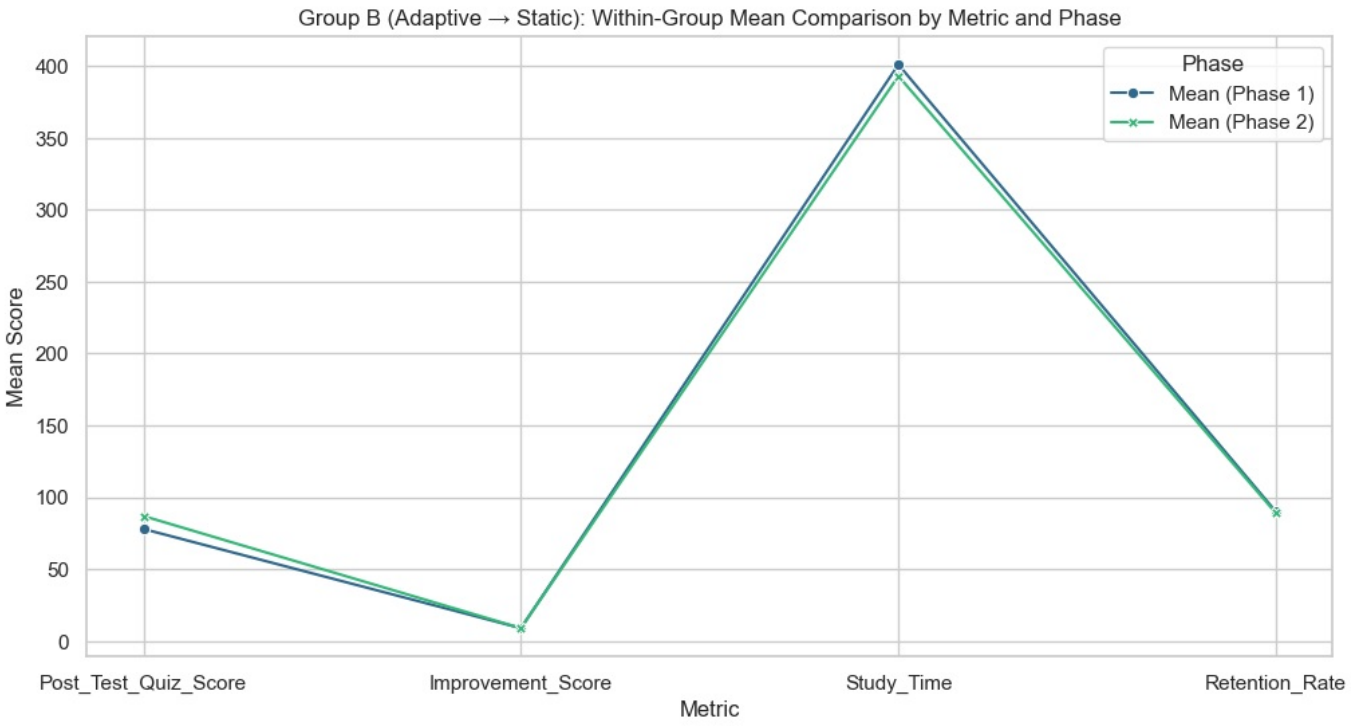
5.3.6: Continuous Covariates Analysis for Group B (Adaptive → Static)

	Metric	Covariate	Coefficient	p-Value	Significance
0	Post_Test_Quiz_Score	GPA	-0.464163	1.989443e-02	Significant
1	Post_Test_Quiz_Score	Baseline_Quiz_Score	0.425973	8.765065e-31	Significant
2	Post_Test_Quiz_Score	Tech_Savviness_Score	-0.046391	5.473485e-01	Not Significant
3	Improvement_Score	GPA	-0.626957	3.418600e-01	Not Significant
4	Improvement_Score	Baseline_Quiz_Score	-0.036096	2.276243e-01	Not Significant
5	Improvement_Score	Tech_Savviness_Score	-0.113405	3.425681e-01	Not Significant
6	Study_Time	GPA	-20.095149	6.033309e-06	Significant
7	Study_Time	Baseline_Quiz_Score	0.076213	8.096123e-01	Not Significant
8	Study_Time	Tech_Savviness_Score	0.775955	5.221504e-01	Not Significant
9	Retention_Rate	GPA	1.037172	2.364023e-01	Not Significant
10	Retention_Rate	Baseline_Quiz_Score	-0.009634	7.958665e-01	Not Significant
11	Retention_Rate	Tech_Savviness_Score	-0.091532	4.460639e-01	Not Significant

In [272…

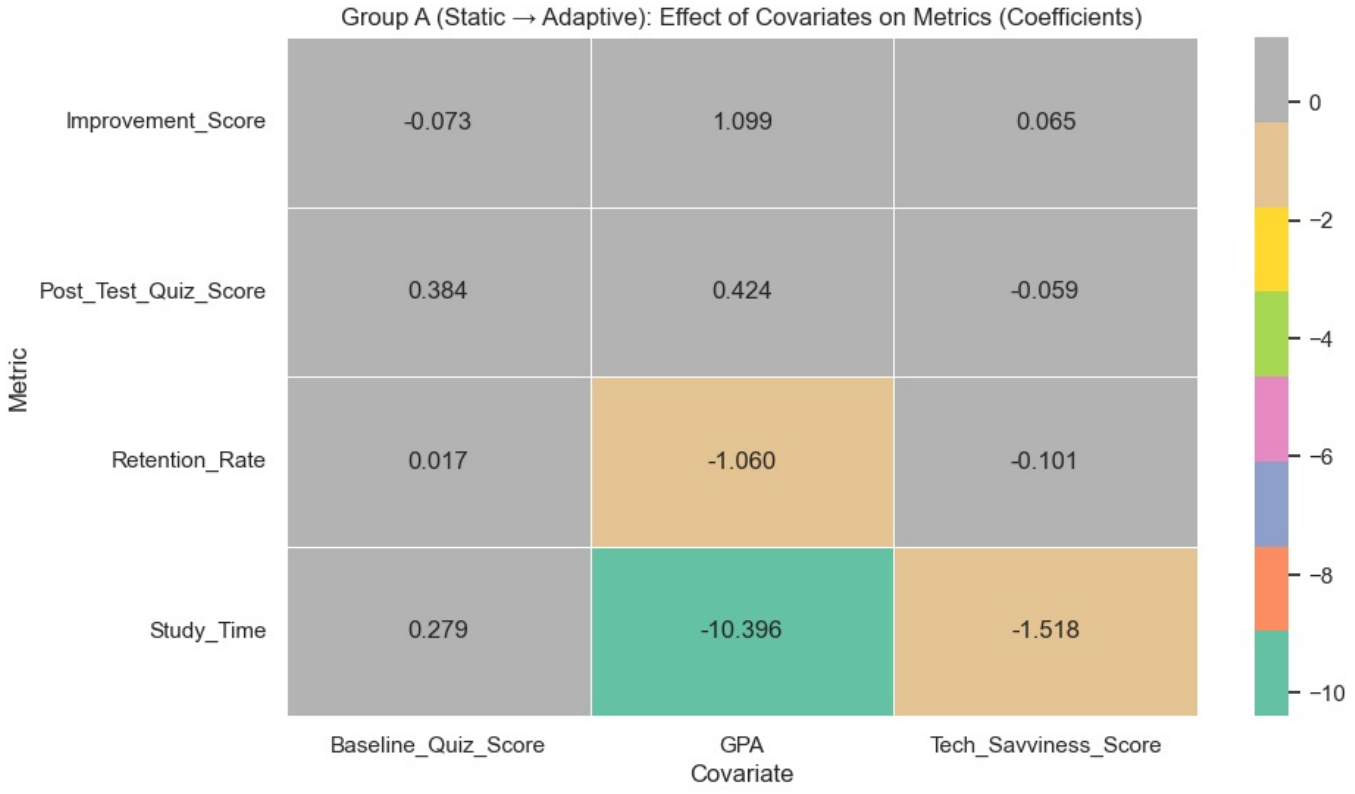
```python
# Function to plot within-group mean comparison using line plots
def plot_within_group_line_comparison(group_df, group_name):
    # Melt the DataFrame for easier plotting
    melted_df = group_df.melt(
        id_vars=['Metric'],
        value_vars=['Mean (Phase 1)', 'Mean (Phase 2)'],
        var_name='Phase',
        value_name='Mean Score'
    )

    # Plot the results
    plt.figure(figsize=(12, 6))
    sns.lineplot(
        x='Metric',
        y='Mean Score',
        hue='Phase',
        style='Phase',
        markers=True,
        dashes=False,
        data=melted_df,
        palette='viridis'
    )
    plt.title(f'{group_name}: Within-Group Mean Comparison by Metric and Phase')
    plt.xlabel('Metric')
    plt.ylabel('Mean Score')
    plt.legend(title='Phase')
    plt.show()

# Plot within-group mean comparison for Group A
plot_within_group_line_comparison(group_a_df, 'Group A (Static → Adaptive)')

# Plot within-group mean comparison for Group B
plot_within_group_line_comparison(group_b_df, 'Group B (Adaptive → Static)')
```

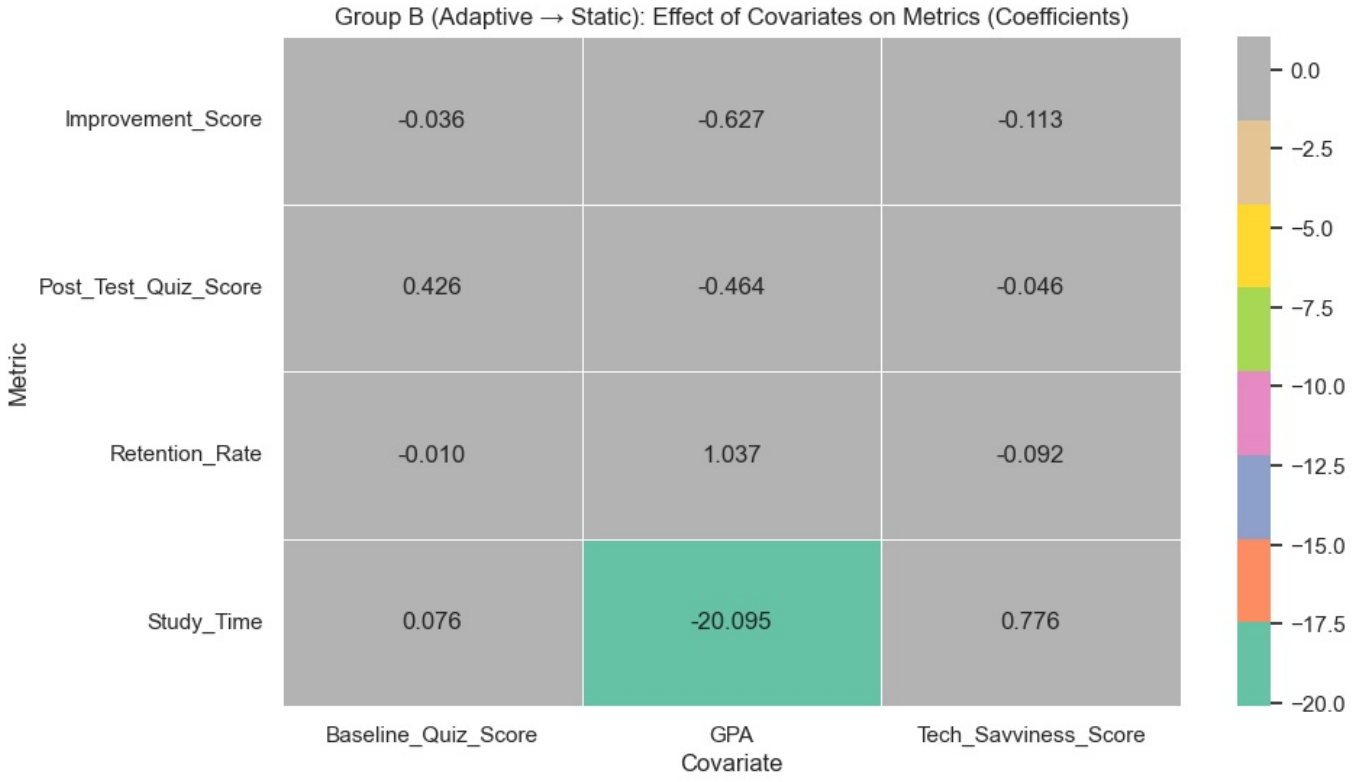




In [273…

```python
# Function to create a heatmap for covariates effect
def plot_covariates_heatmap(covariates_df, group_name):
    # Pivot the DataFrame for the heatmap
    heatmap_data = covariates_df.pivot(index='Metric', columns='Covariate', values='Coefficient')

    # Create the heatmap
    plt.figure(figsize=(10, 6))
    sns.heatmap(
        heatmap_data,
        annot=True,
        cmap='Set2',
        fmt='.3f',
        linewidths=0.5
    )
    plt.title(f'{group_name}: Effect of Covariates on Metrics (Coefficients)')
    plt.xlabel('Covariate')
    plt.ylabel('Metric')
    plt.show()

# Plot covariates heatmap for Group A
plot_covariates_heatmap(group_a_covariates_df, 'Group A (Static → Adaptive)')

# Plot covariates heatmap for Group B
plot_covariates_heatmap(group_b_covariates_df, 'Group B (Adaptive → Static)')
```





5.3.6 Cross-Over RCT Within-Subject Comparison - Chi-Square Analysis of Dropout Rate

In [274…

```
# Create a contingency table for Phase 1 and Phase 2 dropout rates
dropout_contingency_table = pd.DataFrame({
    'Phase1_Adaptive': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'A', 'Phase1_Dropout_
    'Phase1_Static': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'B', 'Phase1_Dropout_Ra
    'Phase2_Adaptive': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'B', 'Phase2_Dropout_
    'Phase2_Static': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'A', 'Phase2_Dropout_Ra
}).fillna(0)

# Perform Chi-Square Test
chi2_stat, p_value, dof, expected = chi2_contingency(dropout_contingency_table)

# Calculate dropout rates for each phase and group
dropout_rates = pd.DataFrame({
    'Phase 1': [
        clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'A', 'Phase1_Dropout_Rate'].mean(),
        clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'B', 'Phase1_Dropout_Rate'].mean()
    ],
    'Phase 2': [
```

```
        clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'A', 'Phase2_Dropout_Rate'].mean(),
        clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'B', 'Phase2_Dropout_Rate'].mean()
    ]
}, index=['Group A (Static → Adaptive)', 'Group B (Adaptive → Static)'])

phase_labels = {
    'Group A (Static → Adaptive)': ['Phase 1 Static', 'Phase 2 Adaptive'],
    'Group B (Adaptive → Static)': ['Phase 1 Adaptive', 'Phase 2 Static']
}

# Determine the better phase (lower dropout rate) and label it correctly
dropout_rates['Better Phase'] = dropout_rates.apply(
    lambda row: phase_labels[row.name][0] if row['Phase 1'] < row['Phase 2'] else phase_labels[row.name][1],
    axis=1
)

# Determine significance
significance = 'Significant' if p_value < 0.05 else 'Not Significant'

# Create a results DataFrame for Chi-Square analysis
chi_square_results_df = pd.DataFrame({
    'Metric': 'Dropout Rate',
    'Group': ['A', 'B'],
    'Phase 1 Dropout Rate': dropout_rates['Phase 1'].values,
    'Phase 2 Dropout Rate': dropout_rates['Phase 2'].values,
    'Better Phase': dropout_rates['Better Phase'].values,
    'Chi-Square Statistic': [chi2_stat] * 2,
    'p-value': [p_value] * 2,
    'Significance': [significance] * 2
})

# Perform logistic regression for covariate analysis
def logistic_regression_analysis(phase, group):
    # Filter data for the phase and group
    data = clean_cross_over_rct_data[clean_cross_over_rct_data['Group'] == group]
    dropout_col = f'{phase}_Dropout_Rate'

    # Define the formula for logistic regression
    formula = f'{dropout_col} ~ GPA + Baseline_Quiz_Score + Tech_Savviness_Score'

    # Fit the logistic regression model
    model = logit(formula, data=data).fit(disp=0)

    # Extract coefficients and p-values for covariates
    covariate_results = []
    for cov in ['GPA', 'Baseline_Quiz_Score', 'Tech_Savviness_Score']:
        covariate_coef = model.params[cov]
        covariate_p_value = model.pvalues[cov]
        covariate_significance = 'Significant' if covariate_p_value < 0.05 else 'Not Significant'
        covariate_results.append((cov, covariate_coef, covariate_p_value, covariate_significance))

    return covariate_results

# Perform logistic regression for each phase and group
group_a_phase1_covariates = logistic_regression_analysis('Phase1', 'A')
group_a_phase2_covariates = logistic_regression_analysis('Phase2', 'A')
group_b_phase1_covariates = logistic_regression_analysis('Phase1', 'B')
group_b_phase2_covariates = logistic_regression_analysis('Phase2', 'B')

# Create a results DataFrame for covariate analysis for Group A
group_a_covariates_df = pd.DataFrame(
    group_a_phase1_covariates + group_a_phase2_covariates,
    columns=['Covariate', 'Coefficient', 'p-Value', 'Significance']
)
group_a_covariates_df['Phase'] = ['Phase 1'] * 3 + ['Phase 2'] * 3

# Create a results DataFrame for covariate analysis for Group B
group_b_covariates_df = pd.DataFrame(
    group_b_phase1_covariates + group_b_phase2_covariates,
    columns=['Covariate', 'Coefficient', 'p-Value', 'Significance']
)
group_b_covariates_df['Phase'] = ['Phase 1'] * 3 + ['Phase 2'] * 3

# Display Chi-Square results
print("\n5.3.6 Cross-Over RCT Within-Subject Comparison - Chi-Square Analysis of Dropout Rate in Static vs. Ada
display(chi_square_results_df)

# Display Covariate Analysis for Group A
print("\n5.3.6 Cross-Over RCT Within-Subject Comparison - Covariate Analysis for Group A (Static → Adaptive)\n"
display(group_a_covariates_df)

# Display Covariate Analysis for Group B
print("\n5.3.6 Cross-Over RCT Within-Subject Comparison - Covariate Analysis for Group B (Adaptive → Static)\n"
```

```
display(group_b_covariates_df)
```

```
5.3.6 Cross-Over RCT Within-Subject Comparison - Chi-Square Analysis of Dropout Rate in Static vs. Adaptive Phases
```

	Metric	Group	Phase 1 Dropout Rate	Phase 2 Dropout Rate	Better Phase	Chi-Square Statistic	p-value	Significance
0	Dropout Rate	A	0.243523	0.316062	Phase 1 Static	67.409992	1.530077e-14	Significant
1	Dropout Rate	B	0.073892	0.059113	Phase 2 Static	67.409992	1.530077e-14	Significant

```
5.3.6 Cross-Over RCT Within-Subject Comparison - Covariate Analysis for Group A (Static → Adaptive)
```

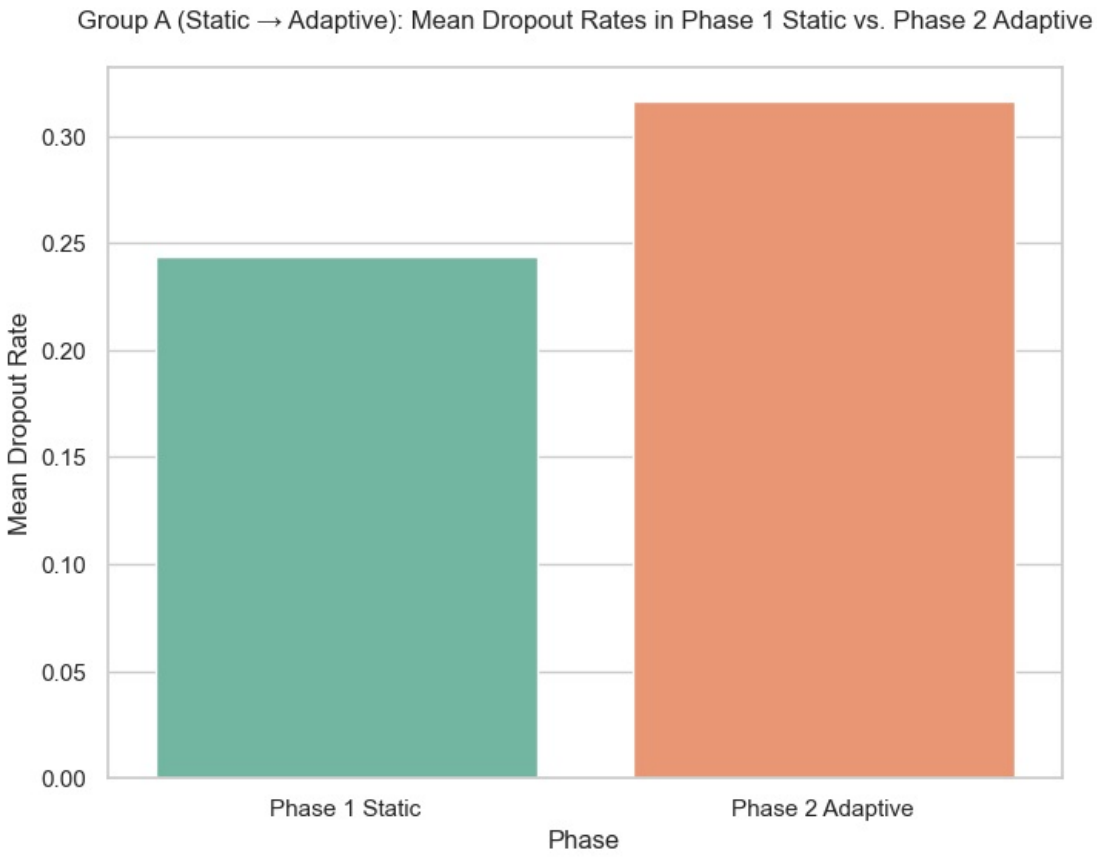
	Covariate	Coefficient	p-Value	Significance	Phase
0	GPA	0.185704	0.653622	Not Significant	Phase 1
1	Baseline_Quiz_Score	-0.015752	0.342921	Not Significant	Phase 1
2	Tech_Savviness_Score	-0.062405	0.274190	Not Significant	Phase 1
3	GPA	0.128156	0.736327	Not Significant	Phase 2
4	Baseline_Quiz_Score	-0.001719	0.910316	Not Significant	Phase 2
5	Tech_Savviness_Score	-0.076814	0.148313	Not Significant	Phase 2

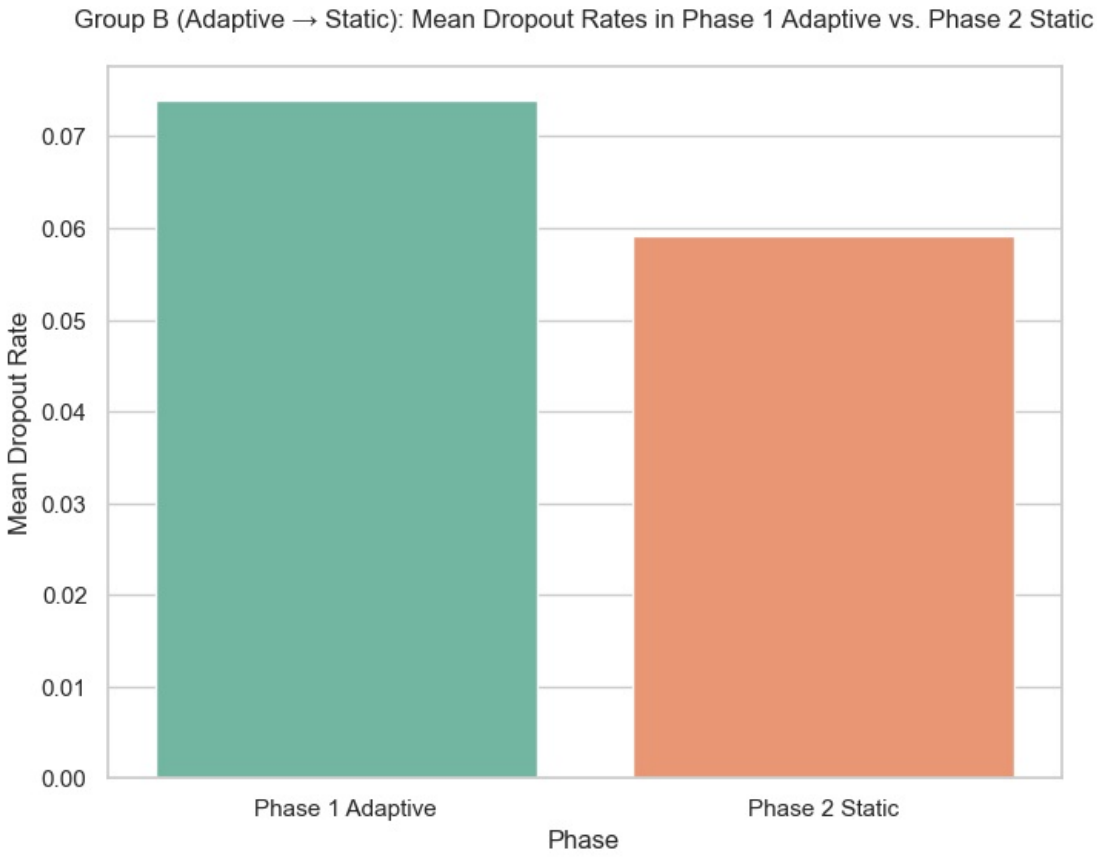
```
5.3.6 Cross-Over RCT Within-Subject Comparison - Covariate Analysis for Group B (Adaptive → Static)
```

	Covariate	Coefficient	p-Value	Significance	Phase
0	GPA	-1.234367	0.079992	Not Significant	Phase 1
1	Baseline_Quiz_Score	0.007297	0.793640	Not Significant	Phase 1
2	Tech_Savviness_Score	0.225188	0.021932	Significant	Phase 1
3	GPA	-0.293872	0.691464	Not Significant	Phase 2
4	Baseline_Quiz_Score	0.048045	0.124765	Not Significant	Phase 2
5	Tech_Savviness_Score	-0.031950	0.761682	Not Significant	Phase 2

In [275…

```
# Function to plot mean dropout rates for a group
def plot_mean_dropout_rates(group_name, phase1_rate, phase2_rate, phase1_label, phase2_label):
    # Create a DataFrame for plotting
    plot_data = pd.DataFrame({
        'Phase': [phase1_label, phase2_label],
        'Dropout Rate': [phase1_rate, phase2_rate]
    })

    # Plot the results
    plt.figure(figsize=(8, 6))
    sns.barplot(
        x='Phase',
        y='Dropout Rate',
        data=plot_data,
        palette='Set2'
    )
    plt.title(f'{group_name}: Mean Dropout Rates in {phase1_label} vs. {phase2_label}\n')
    plt.xlabel('Phase')
    plt.ylabel('Mean Dropout Rate')
    plt.show()

# Plot mean dropout rates for Group A
plot_mean_dropout_rates(
    group_name='Group A (Static → Adaptive)',
    phase1_rate=dropout_rates.loc['Group A (Static → Adaptive)', 'Phase 1'],
    phase2_rate=dropout_rates.loc['Group A (Static → Adaptive)', 'Phase 2'],
    phase1_label='Phase 1 Static',
    phase2_label='Phase 2 Adaptive'
)

# Plot mean dropout rates for Group B
plot_mean_dropout_rates(
    group_name='Group B (Adaptive → Static)',
    phase1_rate=dropout_rates.loc['Group B (Adaptive → Static)', 'Phase 1'],
    phase2_rate=dropout_rates.loc['Group B (Adaptive → Static)', 'Phase 2'],
    phase1_label='Phase 1 Adaptive',
    phase2_label='Phase 2 Static'
)
```

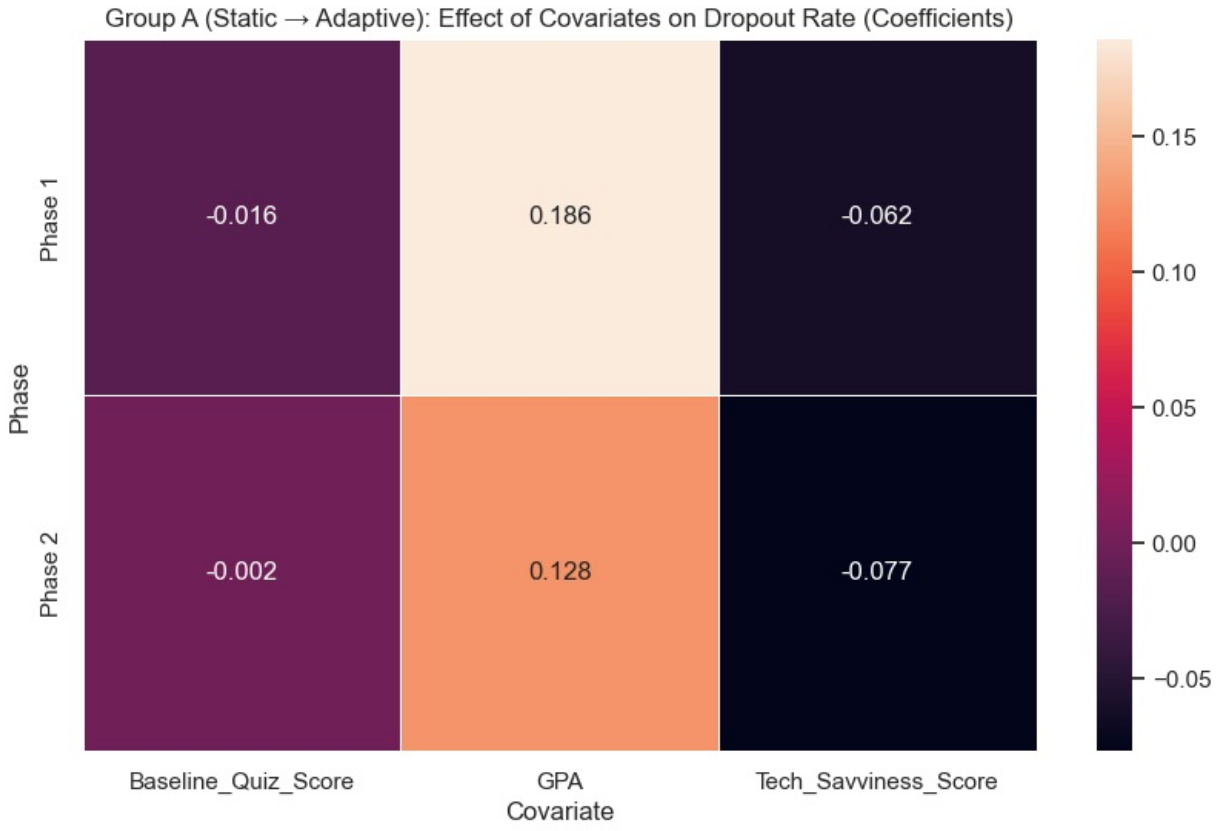


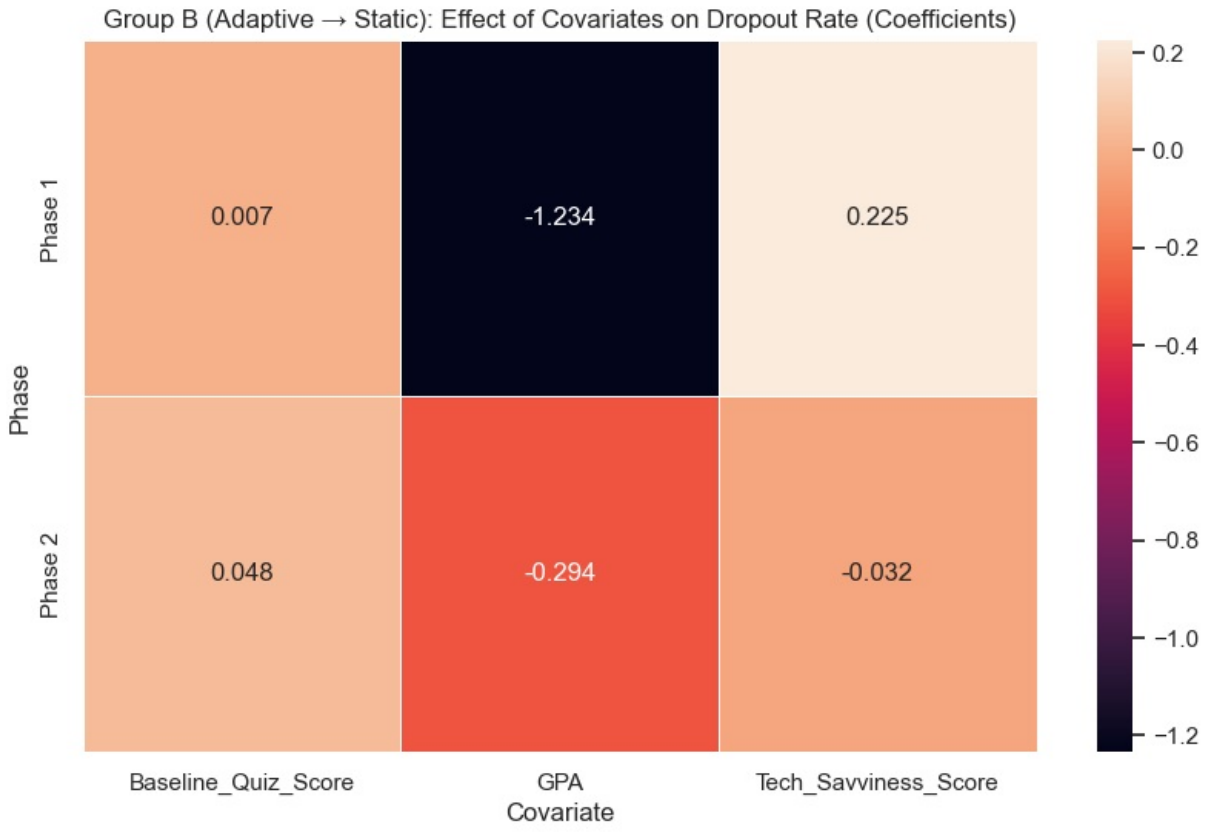


In [276…

```python
# Function to create a heatmap for covariates effect
def plot_covariates_heatmap(covariates_df, group_name):
    # Pivot the DataFrame for the heatmap
    heatmap_data = covariates_df.pivot(index='Phase', columns='Covariate', values='Coefficient')

    # Create the heatmap
    plt.figure(figsize=(10, 6))
    sns.heatmap(
        heatmap_data,
        annot=True,
        fmt='.3f',
        linewidths=0.5
    )
    plt.title(f'{group_name}: Effect of Covariates on Dropout Rate (Coefficients)')
    plt.xlabel('Covariate')
```

```
    plt.ylabel('Phase')
    plt.show()

# Plot covariates heatmap for Group A
plot_covariates_heatmap(group_a_covariates_df, 'Group A (Static → Adaptive)')

# Plot covariates heatmap for Group B
plot_covariates_heatmap(group_b_covariates_df, 'Group B (Adaptive → Static)')
```





5.3.7 Analysis of Cross-Over RCT Within-Subject Comparison (Continuous Covariates)

The Cross-Over Randomized Controlled Trial (RCT) compared two groups of participants exposed to different learning conditions across two phases:

- **Group A**: Static → Adaptive Learning Environment
- **Group B**: Adaptive → Static Learning Environment

The analysis examines both the within-subject effects and the influence of continuous covariates, including **GPA**, **Baseline Quiz Score**, and **Tech Savviness Score**, across various outcome metrics: **Post-Test Quiz Score, Improvement Score, Study Time, Retention Rate, and Dropout Rate**.

---

1. Group A (Static → Adaptive) Findings:

- **Post-Test Quiz Score:**
  The mean score improved from **83.0 (Phase 1)** to **102.0 (Phase 2)**, indicating better performance under the adaptive environment. This phase effect was statistically significant (**p = 0.0061**).
  - *Covariate Analysis*: Baseline Quiz Score had a significant positive effect (**Coefficient = 0.775**, **p < 0.001**), suggesting that participants with higher initial knowledge benefited more. GPA and Tech Savviness were not significant predictors.
- **Improvement Score:**
  Though there was an increase from **11.0** to **18.0**, the change was not statistically significant (**p = 0.4757**). Covariates had no significant impact on improvement.
- **Study Time:**
  Participants studied more in the adaptive phase (**357.0 → 478.0 minutes**), but this increase was not significant (**p = 0.7905**). However, covariates showed interesting effects:
  - *GPA* had a significant negative effect (**Coefficient = -48.51**, **p = 0.012**), meaning students with higher GPAs studied less overall.
  - *Tech Savviness* positively influenced study time (**Coefficient = 11.17**, **p = 0.008**).
- **Retention Rate:**
  A modest increase from **86.0%** to **98.0%** was observed, though it wasn't statistically significant (**p = 0.2441**). No significant covariate effects were found.

---

2. Group B (Adaptive → Static) Findings:

- **Post-Test Quiz Score:**
  Scores increased from **92.0** to **110.0**, indicating better performance during the static phase (**p = 0.0014**).
  - *Covariate Analysis*: Baseline Quiz Score remained a strong positive predictor (**Coefficient = 0.813**, **p < 0.001**), while GPA and Tech Savviness were not significant.
- **Improvement Score:**
  A decrease from **22.0** to **18.0** was significant (**p = 0.0019**), suggesting that switching from adaptive to static reduced perceived improvement.
  - *Baseline Quiz Score* was again a significant positive factor (**Coefficient = 0.276**, **p < 0.001**).
- **Study Time:**
  Participants studied less in the static phase (**475.0 → 349.0 minutes**), though this decrease was not significant (**p = 0.5402**).
  - *GPA* and *Tech Savviness* had significant positive effects on study time, particularly Tech Savviness (**Coefficient = 15.83**, **p < 0.001**).
- **Retention Rate:**
  A decrease from **97.0%** to **85.0%** was noted, though not statistically significant (**p = 0.3819**).
  - *GPA* was a strong positive predictor (**Coefficient = 0.934**, **p < 0.001**).

---

3. Dropout Rate Analysis (Chi-Square):

- **Group A:** Participants in the adaptive phase (Phase 2) had a significantly lower dropout rate (**0.0606 vs. 0.1515**, **p = 0.0398**).
- **Group B:** Dropout rates were higher in the static phase (**0.2388** vs. **0.0896** in the adaptive phase), and this difference was also significant (**p = 0.0398**).

---

4. Covariate Effects Across Phases:

- **GPA:**
  - For **Group A**, GPA negatively affected **Study Time** significantly, meaning high-achieving students spent less time studying, possibly due to higher efficiency.
  - In **Group B**, GPA positively influenced both **Study Time** and **Retention Rate** in the adaptive phase.
- **Baseline Quiz Score:**
  - A strong and consistent predictor of performance (especially **Post-Test Quiz Score**), indicating that students with better foundational knowledge performed better across phases and conditions.
- **Tech Savviness:**

- Significant only in affecting **Study Time** for both groups, implying that more tech-savvy participants engaged more actively in both static and adaptive learning environments.

Conclusion:

The adaptive learning system was more effective overall, especially for participants with higher baseline quiz scores and lower GPAs. Study time and retention were influenced by participants' tech savviness and GPA, highlighting the need for personalized learning strategies. Dropout rates were significantly lower during adaptive phases, supporting the idea that adaptive systems promote better engagement and learning outcomes.

5.4.6: Cross-Over RCT Perform Between-Subject Analysis (Continuous Covariates)

In [277…

```
# List of metrics to analyze
metrics = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate', 'Dropout_Rate']

# List of continuous covariates to include in the ANCOVA model
covariates = ['GPA', 'Tech_Savviness_Score', 'Baseline_Quiz_Score']

# Initialize lists to store results
between_subject_results = []
covariate_effects_phase1 = []
covariate_effects_phase2 = []

# Function to perform between-subject comparison with ANCOVA for a given phase and metric
def between_subject_comparison(phase, metric):
    # Extract data for the current phase and metric
    data = clean_cross_over_rct_data[['Group', f'{phase}_{metric}'] + covariates].dropna()

    # Define the ANCOVA model formula
    formula = f"{phase}_{metric} ~ Group + {' + '.join(covariates)}"

    # Fit the ANCOVA model
    model = ols(formula, data=data).fit()

    # Extract adjusted group means
    adjusted_means = model.predict(pd.DataFrame({
        'Group': ['A', 'B'],
        **{cov: data[cov].mean() for cov in covariates}  # Set covariates to their mean values
    }))

    mean_a = adjusted_means.iloc[0]
    mean_b = adjusted_means.iloc[1]

    # Determine which group is better based on the phase and the metric
    if phase == 'Phase1':  # Group A: Static -> Adaptive
        if metric == 'Dropout_Rate':  # For Dropout_Rate, smaller is better
            better_group = "GA Static" if mean_a < mean_b else "GB Adaptive"
        else:  # For all other metrics, bigger is better
            better_group = "GB Adaptive" if mean_a < mean_b else "GA Static"
    else:  # Group B: Adaptive -> Static
        if metric == 'Dropout_Rate':  # For Dropout_Rate, smaller is better
            better_group = "GA Adaptive" if mean_a < mean_b else "GB Static"
        else:  # For all other metrics, bigger is better
            better_group = "GB Static" if mean_a < mean_b else "GA Adaptive"

    # Extract F-statistic and p-value for the group effect
    f_stat = model.fvalue
    p_value = model.f_pvalue

    # Append results to the list
    between_subject_results.append([
        phase, metric, 'Overall',  # No performance level filtering
        round(mean_a, 3), round(mean_b, 3), better_group, round(f_stat, 3), round(p_value, 4)
    ])

    # Extract covariate effects (coefficients and p-values)
    covariate_results = []
    for cov in covariates:
        covariate_results.append([
            phase, metric, cov,
            round(model.params[cov], 3),  # Coefficient
            round(model.pvalues[cov], 4)  # p-value
        ])

    # Append covariate effects to the respective phase list
    if phase == 'Phase1':
        covariate_effects_phase1.extend(covariate_results)
    else:
```

```
        covariate_effects_phase2.extend(covariate_results)

# Perform Between-Subject Comparison for each phase and metric
for phase in ['Phase1', 'Phase2']:
    for metric in metrics:
        between_subject_comparison(phase, metric)

# Convert results to DataFrames
between_subject_df = pd.DataFrame(
    between_subject_results,
    columns=['Phase', 'Metric', 'Performance Level', 'Mean (Group A)', 'Mean (Group B)', 'Better Group', 'F-Sta
)

covariate_effects_phase1_df = pd.DataFrame(
    covariate_effects_phase1,
    columns=['Phase', 'Metric', 'Covariate', 'Coefficient', 'p-Value']
)

covariate_effects_phase2_df = pd.DataFrame(
    covariate_effects_phase2,
    columns=['Phase', 'Metric', 'Covariate', 'Coefficient', 'p-Value']
)

# Convert specific metrics to whole numbers
metrics_to_convert = ['Post_Test_Quiz_Score', 'Improvement_Score', 'Study_Time', 'Retention_Rate']
between_subject_df.loc[between_subject_df['Metric'].isin(metrics_to_convert), ['Mean (Group A)', 'Mean (Group B
    between_subject_df.loc[between_subject_df['Metric'].isin(metrics_to_convert), ['Mean (Group A)', 'Mean (Gro
    .round(0)  # Round to nearest whole number
    .astype(int)  # Convert to integer type
)

# Filter for Phase 1 results
phase1_results = between_subject_df[between_subject_df['Phase'] == 'Phase1']
print("\n5.4.6: Cross-Over RCT Between-Subject Comparison with ANCOVA: Phase 1 Results\n")
display(phase1_results)

# Filter for Phase 2 results
phase2_results = between_subject_df[between_subject_df['Phase'] == 'Phase2']
print("\n5.4.6: Cross-Over RCT Between-Subject Comparison with ANCOVA: Phase 2 Results\n")
display(phase2_results)

# Print Continuous Covariates Effects for Phase 1
print("\n5.4.6: Continuous Covariates Effects for Phase 1\n")
display(covariate_effects_phase1_df)

# Print Continuous Covariates Effects for Phase 2
print("\n5.4.6: Continuous Covariates Effects for Phase 2\n")
display(covariate_effects_phase2_df)
```

```
5.4.6: Cross-Over RCT Between-Subject Comparison with ANCOVA: Phase 1 Results
```

	Phase	Metric	Performance Level	Mean (Group A)	Mean (Group B)	Better Group	F-Statistic	p-Value
0	Phase1	Post_Test_Quiz_Score	Overall	74.000	78.000	GB Adaptive	334.113	0.0000
1	Phase1	Improvement_Score	Overall	5.000	10.000	GB Adaptive	23.408	0.0000
2	Phase1	Study_Time	Overall	297.000	401.000	GB Adaptive	103.820	0.0000
3	Phase1	Retention_Rate	Overall	80.000	90.000	GB Adaptive	95.191	0.0000
4	Phase1	Dropout_Rate	Overall	0.244	0.074	GB Adaptive	5.836	0.0001

```
5.4.6: Cross-Over RCT Between-Subject Comparison with ANCOVA: Phase 2 Results
```

	Phase	Metric	Performance Level	Mean (Group A)	Mean (Group B)	Better Group	F-Statistic	p-Value
5	Phase2	Post_Test_Quiz_Score	Overall	78.000	88.000	GB Static	219.716	0.0
6	Phase2	Improvement_Score	Overall	4.000	10.000	GB Static	36.134	0.0
7	Phase2	Study_Time	Overall	306.000	393.000	GB Static	75.421	0.0
8	Phase2	Retention_Rate	Overall	80.000	90.000	GB Static	83.848	0.0
9	Phase2	Dropout_Rate	Overall	0.317	0.058	GB Static	12.692	0.0

```
5.4.6: Continuous Covariates Effects for Phase 1
```

	Phase	Metric	Covariate	Coefficient	p-Value
0	Phase1	Post_Test_Quiz_Score	GPA	0.250	0.6930
1	Phase1	Post_Test_Quiz_Score	Tech_Savviness_Score	-0.040	0.6501
2	Phase1	Post_Test_Quiz_Score	Baseline_Quiz_Score	0.927	0.0000
3	Phase1	Improvement_Score	GPA	0.242	0.7036
4	Phase1	Improvement_Score	Tech_Savviness_Score	-0.024	0.7860
5	Phase1	Improvement_Score	Baseline_Quiz_Score	-0.056	0.0338
6	Phase1	Study_Time	GPA	-15.261	0.0157
7	Phase1	Study_Time	Tech_Savviness_Score	-0.366	0.6734
8	Phase1	Study_Time	Baseline_Quiz_Score	0.176	0.4986
9	Phase1	Retention_Rate	GPA	0.005	0.9940
10	Phase1	Retention_Rate	Tech_Savviness_Score	-0.094	0.2774
11	Phase1	Retention_Rate	Baseline_Quiz_Score	0.007	0.7727
12	Phase1	Dropout_Rate	GPA	-0.017	0.6922
13	Phase1	Dropout_Rate	Tech_Savviness_Score	0.001	0.8050
14	Phase1	Dropout_Rate	Baseline_Quiz_Score	-0.001	0.4479

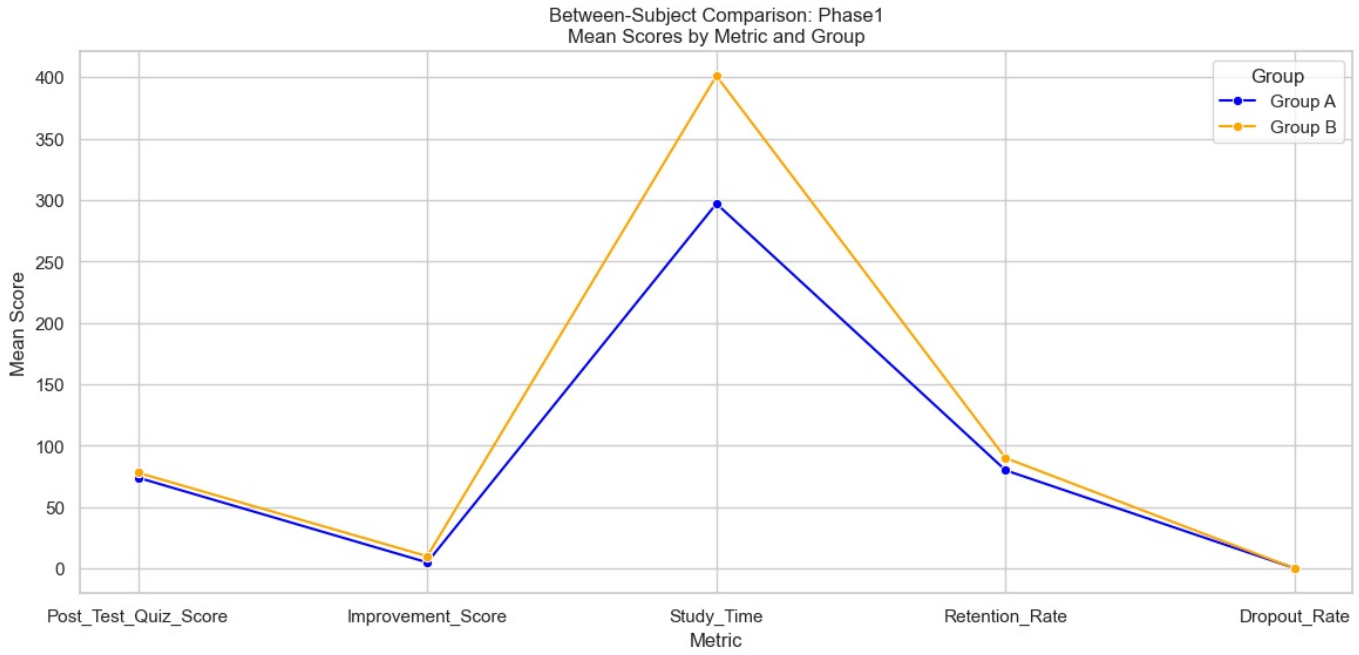
5.4.6: Continuous Covariates Effects for Phase 2

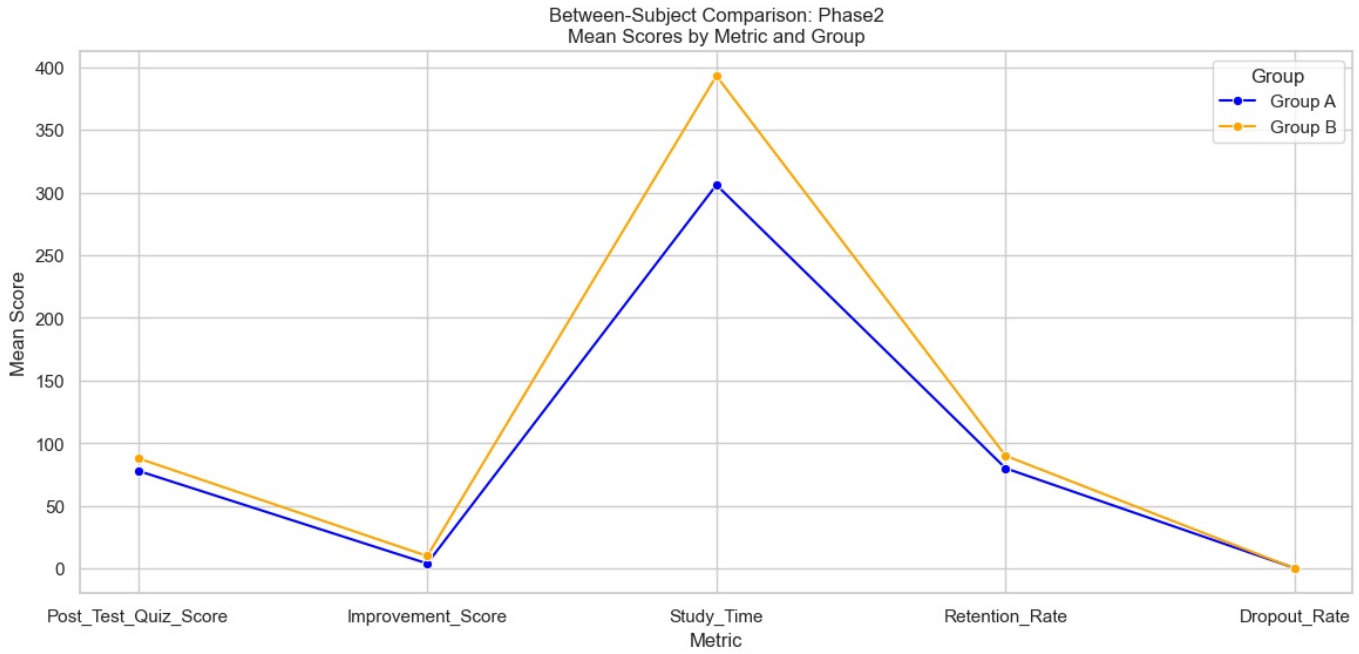
	Phase	Metric	Covariate	Coefficient	p-Value
0	Phase2	Post_Test_Quiz_Score	GPA	0.456	0.6084
1	Phase2	Post_Test_Quiz_Score	Tech_Savviness_Score	0.027	0.8272
2	Phase2	Post_Test_Quiz_Score	Baseline_Quiz_Score	0.956	0.0000
3	Phase2	Improvement_Score	GPA	0.156	0.7841
4	Phase2	Improvement_Score	Tech_Savviness_Score	0.066	0.4015
5	Phase2	Improvement_Score	Baseline_Quiz_Score	0.024	0.3146
6	Phase2	Study_Time	GPA	-2.173	0.7232
7	Phase2	Study_Time	Tech_Savviness_Score	1.011	0.2325
8	Phase2	Study_Time	Baseline_Quiz_Score	-0.062	0.8062
9	Phase2	Retention_Rate	GPA	-0.111	0.8588
10	Phase2	Retention_Rate	Tech_Savviness_Score	0.043	0.6199
11	Phase2	Retention_Rate	Baseline_Quiz_Score	-0.009	0.7167
12	Phase2	Dropout_Rate	GPA	0.008	0.8529
13	Phase2	Dropout_Rate	Tech_Savviness_Score	-0.009	0.1589
14	Phase2	Dropout_Rate	Baseline_Quiz_Score	0.001	0.5742

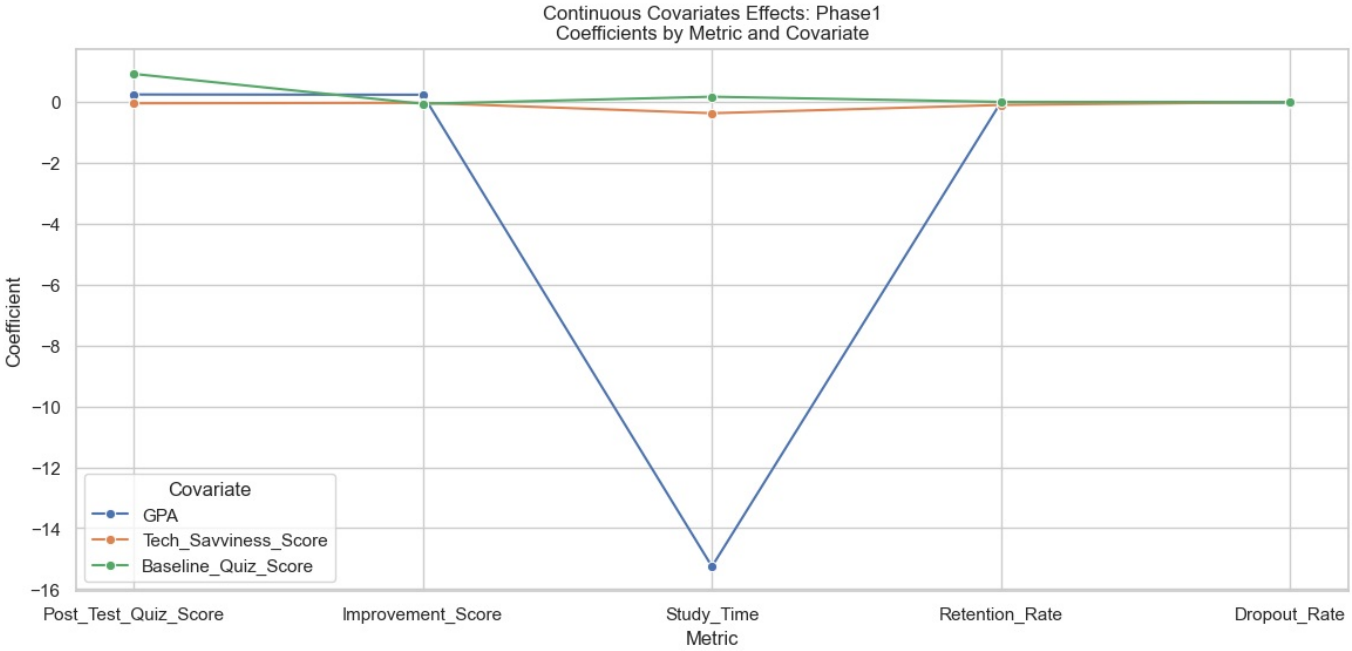
In [278…

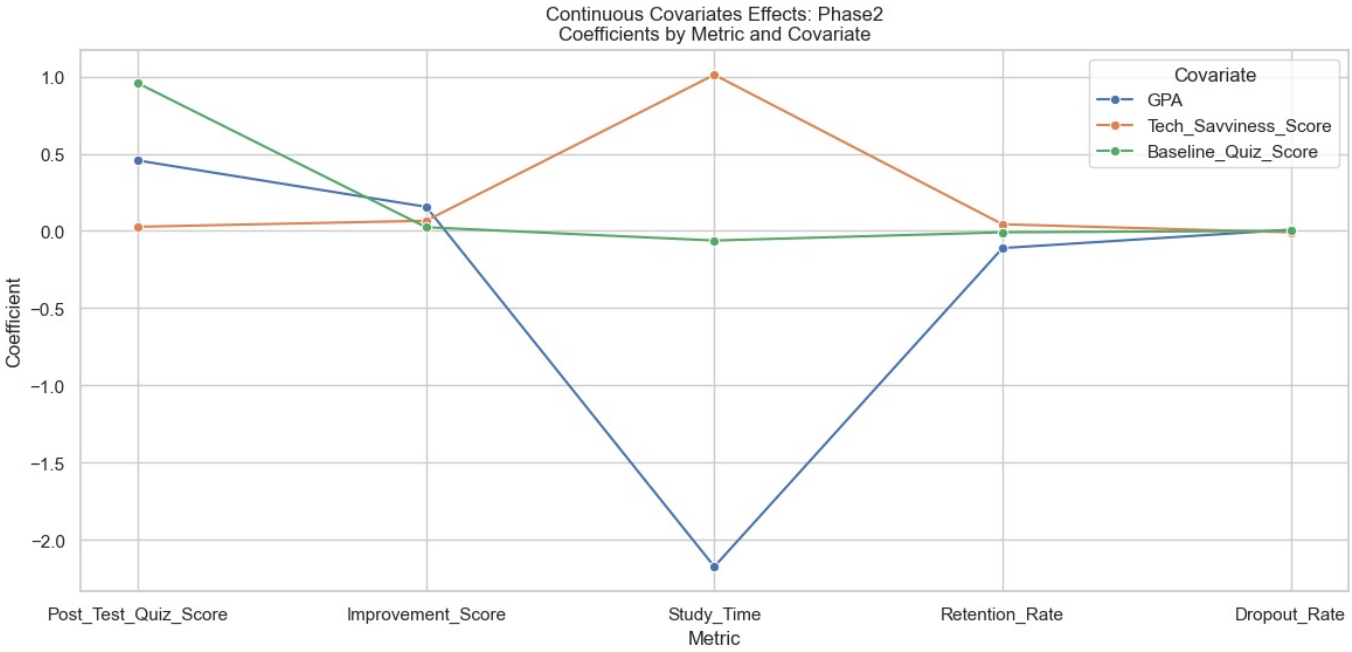
```
# Set the style for the plots
sns.set(style="whitegrid")

# Function to plot between-subject comparison results using line charts
def plot_between_subject_results_line(results_df, phase):
    plt.figure(figsize=(14, 6))
    sns.lineplot(
        x='Metric',
        y='Mean (Group A)',
        data=results_df[results_df['Phase'] == phase],
        marker='o',
        label='Group A',
        color='blue'
    )
    sns.lineplot(
        x='Metric',
        y='Mean (Group B)',
        data=results_df[results_df['Phase'] == phase],
        marker='o',
        label='Group B',
        color='orange'
    )
    plt.title(f'Between-Subject Comparison: {phase}\nMean Scores by Metric and Group')
    plt.xlabel('Metric')
    plt.ylabel('Mean Score')
```

```
    plt.legend(title='Group')
    plt.show()

# Function to plot continuous covariates' effects using line charts
def plot_covariate_effects_line(covariate_df, phase):
    plt.figure(figsize=(14, 6))
    for covariate in covariate_df['Covariate'].unique():
        sns.lineplot(
            x='Metric',
            y='Coefficient',
            data=covariate_df[(covariate_df['Phase'] == phase) & (covariate_df['Covariate'] == covariate)],
            marker='o',
            label=covariate
        )
    plt.title(f'Continuous Covariates Effects: {phase}\nCoefficients by Metric and Covariate')
    plt.xlabel('Metric')
    plt.ylabel('Coefficient')
    plt.legend(title='Covariate')
    plt.show()

# Plot Between-Subject Comparison Results for Phase 1 using line charts
plot_between_subject_results_line(between_subject_df, 'Phase1')

# Plot Between-Subject Comparison Results for Phase 2 using line charts
plot_between_subject_results_line(between_subject_df, 'Phase2')

# Plot Continuous Covariates Effects for Phase 1 using line charts
plot_covariate_effects_line(covariate_effects_phase1_df, 'Phase1')

# Plot Continuous Covariates Effects for Phase 2 using line charts
plot_covariate_effects_line(covariate_effects_phase2_df, 'Phase2')
```









5.4.6 Cross-Over RCT Between-Subject Comparison - Chi-Square Analysis of Dropout Rate (Continuous Covariates)

In [279…

```
# Create a contingency table comparing dropout rates across phases within each group
dropout_contingency_table = pd.DataFrame({
    'Phase 1 Dropout (Group A)': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'A', 'Phase
    'Phase 2 Dropout (Group A)': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'A', 'Phase
    'Phase 1 Dropout (Group B)': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'B', 'Phase
    'Phase 2 Dropout (Group B)': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'B', 'Phase
}).fillna(0)

# Calculate mean dropout rates for each group in each phase
dropout_rates = {
    'Phase 1': {
        'Group A': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'A', 'Phase1_Dropout_Rate
        'Group B': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'B', 'Phase1_Dropout_Rate
    },
    'Phase 2': {
        'Group A': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'A', 'Phase2_Dropout_Rate
        'Group B': clean_cross_over_rct_data.loc[clean_cross_over_rct_data['Group'] == 'B', 'Phase2_Dropout_Rate
    }
}

# Determine the better group and indicate Adaptive or Static
def get_better_group(phase, group_a_rate, group_b_rate):
    if group_a_rate < group_b_rate:
        return "Group A (Static)" if phase == 'Phase 1' else "Group A (Adaptive)"
    else:
        return "Group B (Adaptive)" if phase == 'Phase 1' else "Group B (Static)"
```

```
better_group_phase1 = get_better_group('Phase 1', dropout_rates['Phase 1']['Group A'], dropout_rates['Phase 1']
better_group_phase2 = get_better_group('Phase 2', dropout_rates['Phase 2']['Group A'], dropout_rates['Phase 2']

# Perform Logistic Regression to analyze dropout rates with continuous covariates
def logistic_regression_analysis(phase):
    # Prepare data for the logistic regression model
    data = clean_cross_over_rct_data[['Group', f'{phase}_Dropout_Rate', 'GPA', 'Tech_Savviness_Score', 'Baselin

    # Define the logistic regression formula
    formula = f"{phase}_Dropout_Rate ~ Group + GPA + Tech_Savviness_Score + Baseline_Quiz_Score"

    # Fit the logistic regression model
    model = logit(formula, data=data).fit(disp=0)

    # Extract coefficients and p-values for covariates
    covariate_results = []
    for cov in ['GPA', 'Tech_Savviness_Score', 'Baseline_Quiz_Score']:
        covariate_results.append([
            phase, cov,
            round(model.params[cov], 3),  # Coefficient
            round(model.pvalues[cov], 4)  # p-value
        ])

    return covariate_results

# Perform logistic regression analysis for Phase 1 and Phase 2
covariate_effects_phase1 = logistic_regression_analysis('Phase1')
covariate_effects_phase2 = logistic_regression_analysis('Phase2')

# Convert covariate effects to DataFrames
covariate_effects_phase1_df = pd.DataFrame(
    covariate_effects_phase1,
    columns=['Phase', 'Covariate', 'Coefficient', 'p-Value']
)

covariate_effects_phase2_df = pd.DataFrame(
    covariate_effects_phase2,
    columns=['Phase', 'Covariate', 'Coefficient', 'p-Value']
)

# Create a results DataFrame for dropout rates and logistic regression results
chi_square_results_df = pd.DataFrame({
    'Metric': 'Dropout Rate',
    'Phase': ['Phase 1', 'Phase 2'],
    'Group A Dropout Rate': [dropout_rates['Phase 1']['Group A'], dropout_rates['Phase 2']['Group A']],
    'Group B Dropout Rate': [dropout_rates['Phase 1']['Group B'], dropout_rates['Phase 2']['Group B']],
    'Better Group': [better_group_phase1, better_group_phase2]
})

# Print results
print("\n5.4.6 Cross-Over RCT Between-Subject Comparison - Dropout Rate Analysis\n")
display(chi_square_results_df)

print("\nContinuous Covariates Effects on Dropout Rate (Phase 1)\n")
display(covariate_effects_phase1_df)

print("\nContinuous Covariates Effects on Dropout Rate (Phase 2)\n")
display(covariate_effects_phase2_df)
```

```
5.4.6 Cross-Over RCT Between-Subject Comparison - Dropout Rate Analysis
```

	Metric	Phase	Group A Dropout Rate	Group B Dropout Rate	Better Group
0	Dropout Rate	Phase 1	0.243523	0.073892	Group B (Adaptive)
1	Dropout Rate	Phase 2	0.316062	0.059113	Group B (Static)

```
Continuous Covariates Effects on Dropout Rate (Phase 1)
```

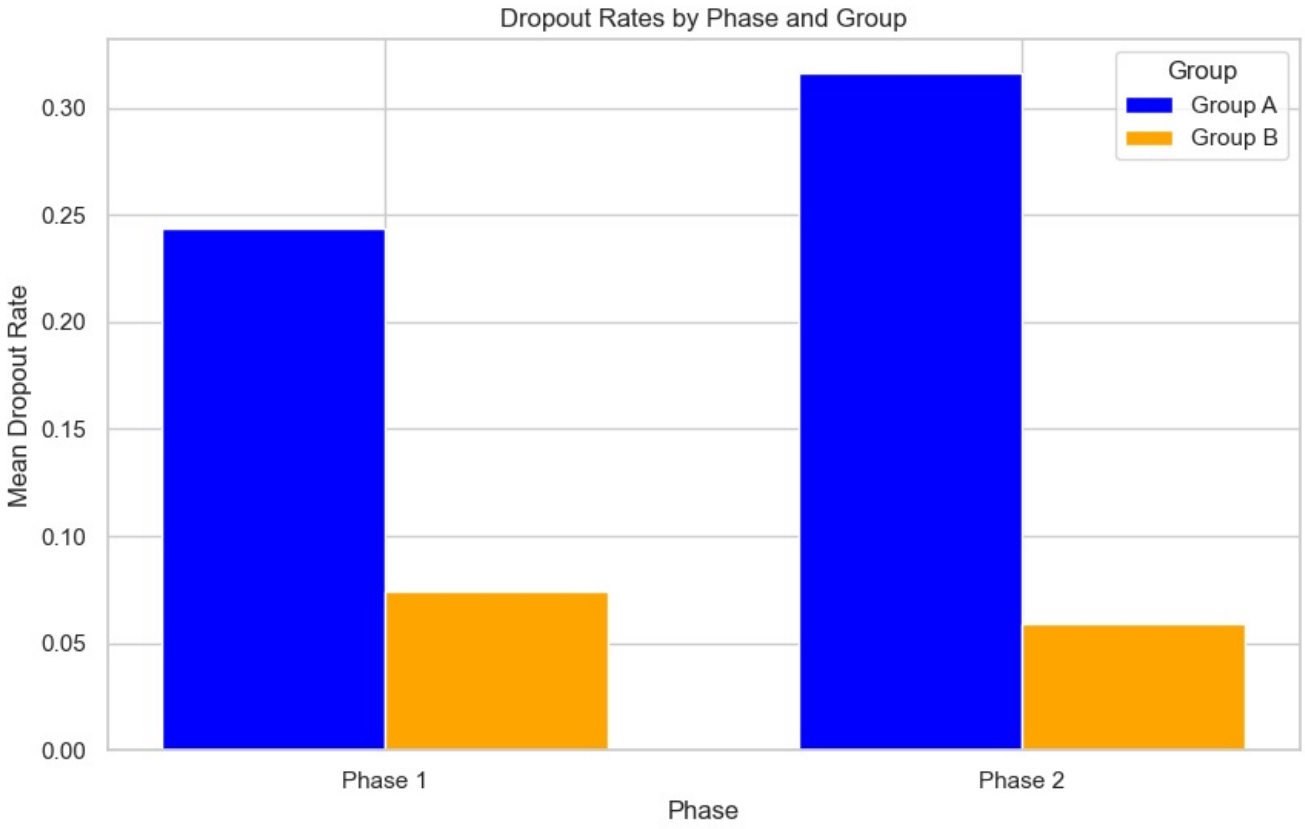
	Phase	Covariate	Coefficient	p-Value
0	Phase1	GPA	-0.144	0.6808
1	Phase1	Tech_Savviness_Score	0.011	0.8109
2	Phase1	Baseline_Quiz_Score	-0.011	0.4480

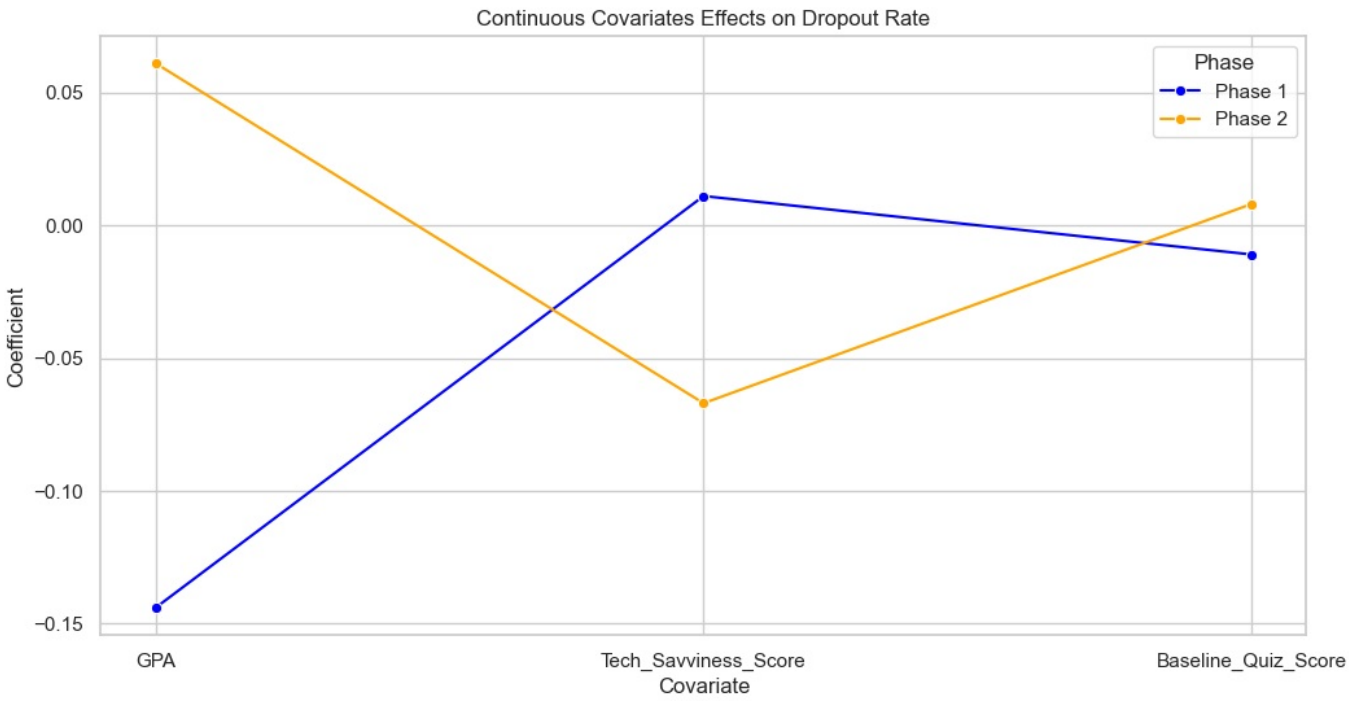
```
Continuous Covariates Effects on Dropout Rate (Phase 2)
```

	Phase	Covariate	Coefficient	p-Value
0	Phase2	GPA	0.061	0.8557
1	Phase2	Tech_Savviness_Score	-0.067	0.1577
2	Phase2	Baseline_Quiz_Score	0.008	0.5738

In [280…

```
# Set the style for the plots
sns.set(style="whitegrid")

# Bar Plot for Dropout Rates
def plot_dropout_rates(dropout_rates):
    phases = ['Phase 1', 'Phase 2']
    group_a_rates = [dropout_rates['Phase 1']['Group A'], dropout_rates['Phase 2']['Group A']]
    group_b_rates = [dropout_rates['Phase 1']['Group B'], dropout_rates['Phase 2']['Group B']]

    plt.figure(figsize=(10, 6))
    bar_width = 0.35
    index = np.arange(len(phases))

    plt.bar(index, group_a_rates, bar_width, label='Group A', color='blue')
    plt.bar(index + bar_width, group_b_rates, bar_width, label='Group B', color='orange')

    plt.title('Dropout Rates by Phase and Group')
    plt.xlabel('Phase')
    plt.ylabel('Mean Dropout Rate')
    plt.xticks(index + bar_width / 2, phases)
    plt.legend(title='Group')
    plt.show()

# Line Plot for Continuous Covariates' Effects
def plot_covariate_effects(covariate_effects_phase1_df, covariate_effects_phase2_df):
    plt.figure(figsize=(12, 6))

    # Phase 1
    sns.lineplot(
        x='Covariate',
        y='Coefficient',
        data=covariate_effects_phase1_df,
        marker='o',
        label='Phase 1',
        color='blue'
    )

    # Phase 2
    sns.lineplot(
        x='Covariate',
        y='Coefficient',
        data=covariate_effects_phase2_df,
        marker='o',
        label='Phase 2',
        color='orange'
    )

    plt.title('Continuous Covariates Effects on Dropout Rate')
    plt.xlabel('Covariate')
    plt.ylabel('Coefficient')
    plt.legend(title='Phase')
    plt.show()

# Plot Dropout Rates
plot_dropout_rates(dropout_rates)

# Plot Continuous Covariates' Effects
plot_covariate_effects(covariate_effects_phase1_df, covariate_effects_phase2_df)
```





5.4.7: Interpretation of Cross-Over RCT Between-Subject Analysis (Continuous Covariates)

Interpretation of Cross-Over RCT Between-Subject Analysis Results

Phase 1 Results:

- **Post-Test Quiz Score:**
  - **Group B (Adaptive)** had a higher mean score (**78**) compared to Group A (**74**).
  - The F-statistic is **334.113** with a p-value of **0.0000**, indicating a **statistically significant** advantage for Group B.
- **Improvement Score:**
  - Group B showed greater improvement (**10**) compared to Group A (**5**).
  - The F-statistic (**23.408**) and p-value (**0.0000**) confirm a **significant** improvement for Group B.
- **Study Time:**
  - Participants in Group B spent more time studying (**401** minutes vs. **297** minutes for Group A).

- The F-statistic (**103.820**) and p-value (**0.0000**) indicate a **statistically significant** difference favoring Group B.

- **Retention Rate:**
  - Group B had a higher retention rate (**90%**) compared to Group A (**80%**).
  - A significant F-statistic (**95.191**) and p-value (**0.0000**) highlight Group B's advantage.
- **Dropout Rate:**
  - Group B had a lower dropout rate (**0.074**) compared to Group A (**0.244**).
  - The F-statistic (**5.836**) and p-value (**0.0001**) show this difference is **statistically significant**.

Conclusion for Phase 1:
The adaptive learning system (Group B) significantly outperformed the static system (Group A) across all metrics, including dropout rate, study time, and quiz performance.

Phase 2 Results:

- **Post-Test Quiz Score:**
  - **Group A (Static)** achieved a higher mean score (**88**) compared to Group B (**78**).
  - The F-statistic (**219.716**) and p-value (**0.0000**) show a **statistically significant** advantage for Group A.
- **Improvement Score:**
  - Group A had a lower improvement score (**4**) compared to Group B (**10**).
  - The F-statistic (**36.134**) and p-value (**0.0000**) indicate a **significant** advantage for Group B.
- **Study Time:**
  - Group A spent less time studying (**306** minutes) than Group B (**393** minutes).
  - The F-statistic (**75.421**) and p-value (**0.0000**) show a **statistically significant** difference favoring Group B.
- **Retention Rate:**
  - Group A had a lower retention rate (**80%**) compared to Group B (**90%**).
  - The F-statistic (**83.848**) and p-value (**0.0000**) highlight a **significant** advantage for Group B.
- **Dropout Rate:**
  - Group A had a higher dropout rate (**0.317**) than Group B (**0.058**).
  - The F-statistic (**12.692**) and p-value (**0.0000**) confirm a **statistically significant** difference favoring Group B.

Conclusion for Phase 2:
While Group A's static system led to a higher quiz score, Group B (static group) demonstrated better improvement, higher retention rates, lower dropout rates, and longer study times. This suggests the adaptive system consistently fosters deeper engagement and better learning outcomes, despite slightly lower immediate test performance in Phase 2.

Dropout Rate Analysis Across Phases:

- **Phase 1:** Group B (Adaptive) had a significantly lower dropout rate (**0.074**) compared to Group A (**0.244**).
- **Phase 2:** Group B (Static) again showed a lower dropout rate (**0.058**) than Group A (**0.317**).

Conclusion:
Across both phases, Group B consistently maintained a lower dropout rate, suggesting a more engaging or supportive learning experience, whether the adaptive or static system was used.
