

## Project Report

# **Optimizing Ad Placements in Mobile Applications: A Dynamic, User-Centric Approach**

### Team members

1. Madi Zhaksylyk
2. Waseem Khan
3. Uzma Rauf
4. Zeinab Soumah

Decision analytics for business and policy

12/08/2023

## Executive Summary

The project develops a dynamic programming model for optimizing ad placements in Android mobile applications, integrating an innovative tool named OptiGuide. This model is specifically designed to adapt ad placements based on real-time user interactions and preferences, aiming to strike a balance between maximizing revenue for app developers and ensuring a positive user experience. By leveraging advanced data analytics, the model employs dynamic programming algorithms to deeply understand user behavior, enabling more personalized and less intrusive ad placements.

A key aspect of this approach is its departure from traditional static ad placements, which often lead to user dissatisfaction and churn. The project proposes a user-centric ad strategy, where real-time user data analysis allows for adjustments in ad frequency, type, and location, making them less disruptive and more relevant to the user experience. This adaptability is crucial in ensuring minimal intrusion while maintaining user engagement with the application.

Additionally, the project's potential to boost revenue for app developers is significant. By optimizing ad placements to be more acceptable and less annoying to users, it increases the likelihood of user engagement with ads. This could lead to higher click-through rates and ad revenues. Importantly, the enhanced user experience can also drive increased app usage and retention, contributing indirectly to revenue growth.

The project includes a comprehensive examination of user feedback mechanisms. These mechanisms ensure that the ad optimization model remains responsive to changing user preferences and behaviors. This ongoing feedback loop is essential for continuously refining the ad placement strategy to align with user expectations.

In summary, this project presents a sophisticated and forward-thinking solution to the complex challenge of balancing ad revenue generation with user satisfaction in Android applications. Its dynamic, data-driven approach, powered by OptiGuide and dynamic programming algorithms, promises to significantly enhance user experience while providing sustainable revenue streams for developers.

## Project Overview

In the fiercely competitive mobile app market, maintaining a positive user experience is extremely important. However, for app developers, generating revenue through advertisements is essential for sustaining operations and fostering growth. The challenge lies in integrating ads in a manner that contributes to revenue without undermining the user experience.

Users often perceive in-app advertisements as a necessary evil; a trade-off for free or low-cost access to app functionality. However, their patience is not limitless. Intrusive or excessive advertising can lead to app abandonment, while too few ads may fail to capitalize on potential earnings. Finding the right balance is therefore not just a matter of financial gain, but also of user retention and satisfaction.

Our project intends to construct an optimization model that identifies the sweet spot for ad placements within Android applications. The model aims to:

- **Maximize Revenue:** By ensuring ads are placed to capture the highest possible engagement without prompting user churn.
- **Enhance User Experience:** Through strategic ad placement that respects the user's in-app journey and minimizes disruption.
- **Dynamic Adjustment:** The model will not be static; it will adapt to the evolving patterns of user interaction and preferences, ensuring that the ad placement strategy remains optimal over time.

## Problem Statement

The core problem is the dichotomy between ad revenue and user experience. The current ad placement strategies often neglect the subtle balance required, leading to diminished user satisfaction or suboptimal revenue generation. This project proposes a dynamic, data-driven approach to ad placement that aligns the financial goals of developers with the user-centric design ethos that modern app users demand.

## Analytical Formulation:

We formulate a dynamic programming solution which addresses our problem by adjusting ad placements based on ongoing user engagement metrics and revenue data, ensuring a harmonious balance between them.

Following is the mathematical formulation of our proposed dynamic programming algorithm.

## Mathematical Formulation:

### Objective Function:

The goal is to maximize the total revenue over the campaign period while considering user engagement. The objective function can be formulated as:

$$\text{Maximize } Z = \sum_{t=1}^T \sum_{i=1}^N R_i \times I_{it}$$

### Variables and Parameters:

1.  $N$ : Number of ad types (e.g., Banner, Interstitial, App Open).
2.  $T$ : Total number of days for the ad campaign.
3.  $I_{it}$ : Number of impressions for ad type  $i$  on day  $t$ .
4.  $R_i$ : Revenue per impression for ad type  $i$ , calculated from historical data.
5.  $D_i$ : User disengagement score for ad type  $i$ .
6.  $B_i$ : Base number of impressions for ad type  $i$ , determined from historical averages.
7.  $\lambda$ : Disengagement decrease rate, a factor by which disengagement score reduces if an ad type is not shown.
8.  $M$ : Minimum revenue threshold for showing an ad.

### Constraints:

1. *Impression Adjustment Constraint*: Ensure that the number of impressions for each ad type is adjusted based on the disengagement score and does not exceed the base number of impressions significantly:

$$I_{it} \leq B_i \times (1 - D_i) \quad \forall i, \forall t$$

2. *User Engagement Constraint*: The cumulative disengagement score for each ad type should be updated based on the number of impressions and the disengagement decrease rate:

$$D_{i,t+1} = D_{it} + (D_i \times I_{it}) - (\lambda \times D_{it}) \quad \forall i, \forall t$$

3. *Revenue Threshold Constraint*: Ensure that the revenue for each ad type exceeds the minimum threshold:

$$R_i \times I_{it} \geq M \quad \forall i, \forall t$$

### Solving Using Dynamic Programming

The dynamic programming approach to solve this problem involves breaking it down into smaller subproblems, each representing the optimization of ad impressions for a single day, and then combining these solutions to form an optimal strategy over the entire period.

#### Steps:

1. *Initialization*: Set the initial values of  $I_{it}$  and  $D_{it}$  based on historical data.
2. *Daily Optimization*:
  - a. For each day  $t$ , calculate the adjusted number of impressions  $I_{it}$  for each ad type  $i$  using the impression adjustment constraint.
  - b. Calculate the revenue for each ad type and ensure it meets the revenue threshold constraint.
  - c. Update the user disengagement scores  $D_{it}$  for the next day using the user engagement constraint.
3. *Iterative Process*:

- a. Repeat the daily optimization process for each day  $t$  in the campaign period.
  - b. Adjust the ad impressions dynamically based on the updated disengagement scores and historical revenue data.
4. *Cumulative Optimization:*
  - a. After iterating through all days, evaluate the total revenue generated and the final user engagement scores.
  - b. Adjust the model parameters if necessary (e.g.,  $\lambda$ ,  $M$ ) and rerun the optimization if the results are not satisfactory.

### Calculating the User Disengagement Score (UDS):

1. Define Changes in Metrics:  $\Delta\text{Metric} = \text{Metric}_{\text{current}} - \text{Metric}_{\text{baseline}}$
2. Assign Weights to Each Metric:
  - a. Let  $W_{\text{Metric}}$  be the weight for each metric.
  - b. User Disengagement Score Formula: 
$$\text{UDS} = \sum (W_{\text{Metric}} \times \Delta\text{Metric})$$

This formulation captures the essential components of the UDS calculation: the change in key user engagement metrics due to ad placements and their respective weights, culminating in a weighted sum to yield the disengagement score.

## Data Summary

We used 3 datasets from 2 different sources: historical revenue data from Google Admob account and user engagement metrics from Google Analytics.

The data csv files can be accessed at:

<https://drive.google.com/drive/folders/1u2bqTeSSbmuNhpy-SHHScUbSGPy6CaQz?usp=sharing>

From Google Admob account we downloaded the report for our application for the 2-month period between October 1 and November 30, 2023. The data includes statistics on the ad type, estimated earnings, and the number of impressions for each ad type on that particular day, among other variables.

Two datasets from Google Analytics represent data on the number of installs and uninstalls for the specified time period.

After cleaning the three datasets (removing unnecessary columns, renaming columns, adjusting data types), we merged them based on the common column – date. The final, ‘ready-to-work’ dataset looked like following:

	format	earnings	impressions	clicks	cpm	new_installs	new_uninstalls	year	month	day	day_of_week
0	Banner	4.27	1482	63	2.88	325	336	2023	10	1	6
1	Banner	5.24	2148	77	2.44	405	344	2023	10	2	0
2	Banner	6.01	2491	87	2.41	438	355	2023	10	3	1
3	Banner	5.78	2558	63	2.26	446	312	2023	10	4	2
4	Banner	6.76	2588	85	2.61	460	405	2023	10	5	3
...	...	...	...	...	...	...	...	...	...	...	...
136	Banner	14.98	6561	163	2.28	665	480	2023	11	29	2
137	Interstitial	2.68	826	37	3.24	665	480	2023	11	29	2
138	App open	10.70	2287	174	4.68	644	459	2023	11	30	3
139	Banner	16.76	6488	164	2.58	644	459	2023	11	30	3
140	Interstitial	3.28	863	42	3.80	644	459	2023	11	30	3

Given this historical data, we were able to initialize the base parameters for our dynamic programming model. For example, base impressions for each ad type depended on the average number of impressions for that ad type in the last two months. Similarly, revenue per impression for each ad type was based on historical averages.

Then our model employs dynamic programming to maximize ad revenue over a series of days. Each day, it calculates potential revenue from different ad types based on impressions and adjusts the revenue value by accounting for user disengagement (measure of user's dissatisfaction with ads). The model then updates the base impressions for each ad type to reflect the current day's performance. This iterative process ensures that each day's ad placement strategy is optimized based on the evolving engagement levels and revenue potential, at the same time aiming to exceed the minimum revenue threshold before an ad is placed.

As an output, the model produces a table with the optimal strategy for every day of interest with ad types to place, estimated net revenue, and the number of impressions for each ad type.

## Analysis with the Language Wrapper. Model questions and their outputs:

### **Question 1: Compare ad placement strategies for the next 5 days with current revenue per impression for each ad type against the one with revenue per impression for each ad type increased by 20%?**

The goal of this question was twofold: to understand how revenue per impression may change the optimal strategy and whether OptiGuide is able to generate two tables to compare.

```
--- Code ---
# Increase revenue per impression by 20% for each ad type
num_days = 5
increased_revenue = [revenue * 1.2 for revenue in revenue_per_impression]
increased_optimizer = AdPlacementOptimizer(num_days, ad_types, increased_revenue, disengagement_score,
base_impressions, disengagement_decrease_rate, min_revenue)
print("Increased Revenue Strategy:")
increased_optimizer.print_optimal_strategy()

# Compare ad placement strategies with current revenue per impression with given number of days
num_days = 5
current_optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_score,
base_impressions, disengagement_decrease_rate, min_revenue)
print("Current Strategy:")
current_optimizer.print_optimal_strategy()
```

Increased Revenue Strategy:				Current Strategy:			
Day	Ad Type	Net Revenue	Impressions	Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	\$6.71	2954.0	Day 1	Banner	\$3.01	1742.0
	Interstitial	\$1.53	547.0		OpenAd	\$2.93	766.0
	OpenAd	\$6.49	1301.0	Day 2	Banner	\$2.71	1567.0
Day 2	Banner	\$6.04	2658.0		OpenAd	\$2.64	689.0
	Interstitial	\$1.38	492.0	Day 3	Banner	\$2.44	1410.0
	OpenAd	\$5.84	1170.0		OpenAd	\$2.37	620.0
Day 3	Banner	\$5.44	2392.0	Day 4	Banner	\$2.19	1269.0
	Interstitial	\$1.24	442.0		OpenAd	\$2.13	558.0
	OpenAd	\$5.25	1053.0	Day 5	Banner	\$1.97	1142.0
Day 4	Banner	\$4.89	2152.0		OpenAd	\$1.92	502.0
	Interstitial	\$1.11	397.0	Total Revenue over 5 days: \$24.31			
	OpenAd	\$4.73	947.0	'optimizer'			
Day 5	Banner	\$4.40	1936.0	Ad Optimization (to user):			
	OpenAd	\$4.25	852.0	Sorry. I cannot answer your question.			
Total Revenue over 5 days: \$59.30							

Although it was able to identify the variable to change the code for (revenue per impression), at first, OptiGuide struggled to generate two similar tables: they could differ



in the number of days the strategy is designed for. Only after guiding by adding the code snippets into example\_qa helper, the output was correctly generated. However, even after implementing those changes we still received 'Sorry. I cannot answer your question.' message.

**Question 2: What would be the ad placement strategy over the next 7 days if I increase the base impression by 10% ?**

In this inquiry, we investigated the potential ad placement strategy when increasing the frequency of ad displays.

```
--- Code ---
num_days = 7 # Number of days for the ad campaign
new_base_impressions = [impression * 1.1 for impression in base_impressions] # Increase base impressions by 10%

optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, new_base_impressions, disengagement_decrease_rate, min_revenue)
optimizer.print_optimal_strategy()

--- One-Word Answer: SAFE or DANGER ---
```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	\$5.61	3250.0
	Interstitial	\$1.10	602.0
	OpenAd	\$5.47	1431.0
Day 2	Banner	\$5.05	2925.0
	OpenAd	\$4.92	1287.0
Day 3	Banner	\$4.55	2632.0
	OpenAd	\$4.43	1158.0
Day 4	Banner	\$4.09	2368.0
	OpenAd	\$3.99	1042.0
Day 5	Banner	\$3.68	2131.0
	OpenAd	\$3.58	937.0
Day 6	Banner	\$3.31	1917.0
	OpenAd	\$3.22	843.0
Day 7	Banner	\$2.98	1725.0
	OpenAd	\$2.90	758.0
Total Revenue over 7 days:		\$58.90	

**Question 3: What is the best ad placement strategy for the next 5 days if the minimum revenue threshold is set to 3 instead of 1?**

In this analysis, we explored the potential impact of elevating the revenue threshold on our ad placement strategy. By raising this threshold, we aimed to refine our approach, filtering out underperforming ads. This shift prioritizes quality over quantity, enabling a more targeted strategy geared towards placements with a higher probability of meeting or exceeding our revenue goals.

Optiguide adeptly implemented the updated parameters based on the new variables provided.

```
-----  
writer (to Ad Optimization):  
  
num_days = 5 # Number of days for the ad campaign  
min_revenue = 3 # New minimum revenue threshold for showing an ad  
  
optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impressions, disengagement_decrease_rate, min_revenue)  
  
optimizer.print_optimal_strategy()  
-----
```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	\$5.10	2954.0
	OpenAd	\$4.98	1301.0
Day 2	Banner	\$4.59	2658.0
	OpenAd	\$4.48	1170.0
Day 3	Banner	\$4.13	2392.0
	OpenAd	\$4.03	1053.0
Day 4	Banner	\$3.72	2152.0
	OpenAd	\$3.62	947.0
Day 5	Banner	\$3.34	1936.0
	OpenAd	\$3.26	852.0
Total Revenue over 5 days: \$41.25			

#### Question 4: What would the ad placement for next 7 days look like if we take out interstitial ads?

In this inquiry, our goal is to analyze the revised ad strategy following the removal of the interstitial ad from the list. Interstitial ads can disrupt the user experience by interrupting the flow of content or navigation.

Optiguide effectively executed the removal and presented the updated ad strategy accordingly.

```
# Removing interstitial ads from the ad_types, revenue_per_impression, disengagement_scores, and base_impressions lists
ad_types.remove('Interstitial')
revenue_per_impression.pop(1)
disengagement_scores.pop(1)
base_impressions.pop(1)

# Creating an instance of AdPlacementOptimizer with the updated parameters
num_days = 7
optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impressions, disengagement_decrease_rate, min_revenue)

optimizer.print_optimal_strategy() # Executing the ad placement strategy to get the optimal ad placement strategy and total revenue
```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	\$5.10	2954.0
	OpenAd	\$4.98	1301.0
Day 2	Banner	\$4.59	2658.0
	OpenAd	\$4.48	1170.0
Day 3	Banner	\$4.13	2392.0
	OpenAd	\$4.03	1053.0
Day 4	Banner	\$3.72	2152.0
	OpenAd	\$3.62	947.0
Day 5	Banner	\$3.34	1936.0
	OpenAd	\$3.26	852.0
Day 6	Banner	\$3.01	1742.0
	OpenAd	\$2.93	766.0
Day 7	Banner	\$2.71	1567.0
	OpenAd	\$2.64	689.0

Total Revenue over 7 days: \$52.53

## Question 5: “What is the best ad placement strategy for the next 10 days if I want only the OpenAd?”

Through this question, we wanted to see how an ad will perform in a time bound. It was similar to a scenario analysis which is conducted for almost every business strategy before implementation.

```
--- Code ---
# Creating an instance of AdPlacementOptimizer with specified parameters
num_days = 10 # Number of days for the ad campaign
ad_types = ["OpenAd"] # Only using the OpenAd type
revenue_per_impression = [app_open_revenue_avg_per_imp] # Revenue per impression for OpenAd
disengagement_scores = [0.002] # Disengagement score for OpenAd
base_impressions = [app_open_impressions_avg] # Base impressions for OpenAd
disengagement_decrease_rate = 0.1 # Rate at which disengagement decreases over time
min_revenue = 1.0 # Minimum revenue threshold for showing an ad

optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impressions, disengagement_decrease_rate, min_revenue)
optimizer.print_optimal_strategy() # Executing the ad placement strategy to get the optimal ad placement strategy and total revenue
```

Day	Ad Type	Net Revenue	Impressions
Day 1	OpenAd	\$4.98	1301.0
Day 2	OpenAd	\$4.48	1170.0
Day 3	OpenAd	\$4.03	1053.0
Day 4	OpenAd	\$3.62	947.0
Day 5	OpenAd	\$3.26	852.0
Day 6	OpenAd	\$2.93	766.0
Day 7	OpenAd	\$2.64	689.0
Day 8	OpenAd	\$2.37	620.0
Day 9	OpenAd	\$2.13	558.0
Day 10	OpenAd	\$1.92	502.0

Total Revenue over 10 days: \$32.35

**Question 6: What is the best ad placement strategy for the next 5 days if the initial disengagement score for interstitial ad type was 0.006 instead of 0.003?**

The intention of this question is to understand the impact of doubling the initial disengagement score for interstitial ads on the overall ad placement strategy. This aims to see how sensitive the strategy is to user engagement levels specifically for interstitial ads.

```
--- Code ---
num_days = 5 # Number of days for the ad campaign
ad_types = ["Banner", "Interstitial", "OpenAd"] # Types of ads
revenue_per_impression = [banner_revenue_avg_per_imp, interstitial_revenue_avg_per_imp, app_open_revenue_avg_per_imp] # Average revenue per impression for each ad type
disengagement_scores = [0.001, 0.006, 0.002] # Disengagement scores for each ad type
base_impressions = [banner_impressions_avg, interstitial_impressions_avg, app_open_impressions_avg] # Average base impressions for each ad type
disengagement_decrease_rate = 0.1 # Rate at which disengagement decreases over time
min_revenue = 1.0 # Minimum revenue threshold for showing an ad

optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impressions, disengagement_decrease_rate, min_revenue)
optimizer.print_optimal_strategy()

--- One-Word Answer: SAFE or DANGER ---
```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	\$5.10	2954.0
	OpenAd	\$4.98	1301.0
Day 2	Banner	\$4.59	2658.0
	OpenAd	\$4.48	1170.0
Day 3	Banner	\$4.13	2392.0
	OpenAd	\$4.03	1053.0
Day 4	Banner	\$3.72	2152.0
	OpenAd	\$3.62	947.0
Day 5	Banner	\$3.34	1936.0
	OpenAd	\$3.26	852.0
Total Revenue over 5 days: \$41.25			

As we have learned from our experience with OptiGuide, questions should be direct and parameter-specific to be effectively processed by OptiGuide. This question clearly indicates the variable to be altered, so Optiguide was able to incorporate changes into the code and output a reasonable answer.

**Question 7: How should we change the initial (base) parameters to achieve diversity (every ad type appears at least 2 times, on every day there are at least 2 different ad types) in ad types in the next 7 days?**

This time we wanted to check if OptiGuide can handle more complex questions, requiring more comprehensive analysis and modification of the original code by adding constraints.

```
--- Code ---
num_days = 7 # Number of days for the ad campaign
ad_types = ["Banner", "Interstitial", "OpenAd"] # Types of ads
# Average revenue per impression for each ad type
revenue_per_impression = [banner_revenue_avg_per_imp, interstitial_revenue_avg_per_imp, app_open_revenue_avg_per_imp]
disengagement_scores = [0.001, 0.003, 0.002] # Disengagement scores for each ad type
# Average base impressions for each ad type
base_impressions = [banner_impressions_avg, interstitial_impressions_avg, app_open_impressions_avg]
disengagement_decrease_rate = 0.1 # Rate at which disengagement decreases over time
min_revenue = 1.0 # Minimum revenue threshold for showing an ad

# Update the base impressions to ensure diversity
for i in range(len(ad_types)):
    base_impressions[i] = 2 * base_impressions[i]

# Creating an instance of AdPlacementOptimizer with updated parameters
optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impressions, disengagement_decrease_rate, min_revenue)

optimizer.print_optimal_strategy() # Executing the ad placement strategy to get the optimal ad placement strategy and total revenue

--- One-Word Answer: SAFE or DANGER ---
```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	\$10.21	5909.0
	Interstitial	\$2.00	1094.0
	OpenAd	\$9.95	2602.0
Day 2	Banner	\$9.19	5318.0
	Interstitial	\$1.80	984.0
	OpenAd	\$8.95	2341.0
Day 3	Banner	\$8.27	4786.0
	Interstitial	\$1.62	885.0
	OpenAd	\$8.06	2106.0
Day 4	Banner	\$7.44	4307.0
	Interstitial	\$1.46	796.0
	OpenAd	\$7.25	1895.0
Day 5	Banner	\$6.70	3876.0
	Interstitial	\$1.31	716.0
	OpenAd	\$6.52	1705.0
Day 6	Banner	\$6.02	3488.0
	Interstitial	\$1.18	644.0
	OpenAd	\$5.87	1534.0
Day 7	Banner	\$5.42	3139.0
	Interstitial	\$1.06	579.0
	OpenAd	\$5.28	1380.0
Total Revenue over 7 days:		\$115.55	

As can be seen from the screenshots, the python code was modified, but the language wrapper interpreted the condition as 'double the base impressions' instead of adding constraint-type variables. We should not rely on the output table since OptiGuide was not able to deal with this question correctly in the first place.

**Question 8: What is the ad placement strategy if we want a total revenue of more than \$100 in the next 6 days by introducing a new ad type while keeping the minimum revenue threshold the same?**

Next we wanted to check if OptiGuide can deal with prompts that basically set new (untrained) objectives.

```

--- Code ---
# OPTIGUIDE OVERRIDE INPUT VALUES CODE GOES HERE
# If any parameter is not specified in question, the default value will be used from the above code

# Creating an instance of AdPlacementOptimizer with specified parameters
num_days = 6 # Number of days for the ad campaign
ad_types = ["Banner", "Interstitial", "OpenAd", "NewAd"] # Types of ads including the new ad type
# Revenue per impression for each ad type, including the new ad type
revenue_per_impression = [banner_revenue_avg_per_imp, interstitial_revenue_avg_per_imp, app_open_revenue_avg_per_imp, 0.005]
disengagement_scores = [0.001, 0.003, 0.002, 0.001] # Disengagement scores for each ad type, including the new ad type
# Base impressions for each ad type, including the new ad type
base_impressions = [banner_impressions_avg, interstitial_impressions_avg, app_open_impressions_avg, 1000]
disengagement_decrease_rate = 0.1 # Rate at which disengagement decreases over time
min_revenue = 1.0 # Minimum revenue threshold for showing an ad

# Creating an instance of AdPlacementOptimizer with specified parameters
optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impressions, disengagement_decrease_rate, min_revenue)

optimizer.print_optimal_strategy() # Executing the ad placement strategy to get the optimal ad placement strategy and total revenue

```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	\$5.10	2954.0
	Interstitial	\$1.00	547.0
	OpenAd	\$4.98	1301.0
	NewAd	\$3.60	900.0
Day 2	Banner	\$4.59	2658.0
	OpenAd	\$4.48	1170.0
	NewAd	\$3.24	810.0
Day 3	Banner	\$4.13	2392.0
	OpenAd	\$4.03	1053.0
	NewAd	\$2.92	729.0
Day 4	Banner	\$3.72	2152.0
	OpenAd	\$3.62	947.0
	NewAd	\$2.62	656.0
Day 5	Banner	\$3.34	1936.0
	OpenAd	\$3.26	852.0
	NewAd	\$2.36	590.0
Day 6	Banner	\$3.01	1742.0
	OpenAd	\$2.93	766.0
	NewAd	\$2.12	531.0
Total Revenue over 6 days: \$65.05			

Although the part of the code that introduces a new ad type was correctly created and run, nothing in the original code has been changed to ensure the minimal amount of total revenue we targeted. Most likely that is due to the nature of our dynamic programming model that was designed to maximize revenue iteratively on a daily basis , rather than targeting maximization or specific values by the end of a period.

**Question 9: What is the best ad placement strategy for the next 5 days if the revenues per impression for each ad type were based on the maximum historical values of revenue for that ad type instead of mean values?**

As a manager or an app owner we might want to see what our revenues and optimal strategy would be if we could always stick to the historically best performance. So in this question we changed the value of one of the initial parameters, revenue per impression for each ad type, from the mean of its historical values to the maximum recorded so far.

```
--- Code ---
num_days = 5 # Number of days for the ad campaign
revenue_per_impression = []

for ad_type, impressions in zip(ad_types, base_impressions):
    ad_type_data = merged_data[merged_data['format'] == ad_type]
    if not ad_type_data.empty:
        max_revenue = ad_type_data['earnings'].max()
        revenue_per_impression.append(max_revenue / impressions)
    else:
        revenue_per_impression.append(0)

optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disen-
gagement_scores, base_impressions, disengagement_decrease_rate, min_revenue)
optimizer.print_optimal_strategy()
```

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	\$12.13	2954.0
	Interstitial	\$4.27	547.0
Day 2	Banner	\$10.91	2658.0
	Interstitial	\$3.84	492.0
Day 3	Banner	\$9.82	2392.0
	Interstitial	\$3.45	442.0
Day 4	Banner	\$8.83	2152.0
	Interstitial	\$3.10	397.0
Day 5	Banner	\$7.95	1936.0
	Interstitial	\$2.79	357.0
Total Revenue over 5 days: \$67.07			

The code was changed correctly from `.mean()` to `.max`, which is exactly what we wanted. Correspondingly, the optimal ad placement strategy is different from the one with initial parameter values.



**Question 10: What is the best ad placement strategy for the next 7 days if you divide each banner's revenue by 2?**

Every manager is interested in maximizing the revenue, and would only set the price that would bring the maximum profit, but a manager also has to plan for the worst case scenario. These type of questions are crucial for building a stress testing model for the product.

```
--- Code ---
num_days = 7 # Number of days for the ad campaign
revenue_per_impression[0] = banner_revenue_avg_per_imp / 2 # Divide banner's revenue by 2
ad_placement_optimizer = AdPlacementOptimizer(num_days, ad_types, revenue_per_impression, disengagement_scores, base_impressions, disengagement_decrease_rate, min_revenue)
ad_placement_optimizer.print_optimal_strategy()
```

--- One-Word Answer: SAFE or DANGER ---

Day	Ad Type	Net Revenue	Impressions
Day 1	Banner	\$1.07	2954.0
	Interstitial	\$1.00	547.0
	OpenAd	\$4.98	1301.0
Day 2	OpenAd	\$4.48	1170.0
Day 3	OpenAd	\$4.03	1053.0
Day 4	OpenAd	\$3.62	947.0
Day 5	OpenAd	\$3.26	852.0
Day 6	OpenAd	\$2.93	766.0
Day 7	OpenAd	\$2.64	689.0
Total Revenue over 7 days:		\$28.00	

## Analyses with Language Wrapper. Challenges

To adapt the OptiGuide framework for the specific challenges and requirements of ad placement optimization using dynamic programming, rather than supply chain optimization with Gurobi, we implemented several modifications. These changes were crucial to tailor the tool to the unique complexities and solution strategies of the ad placement domain. A more comprehensive comparison, including examples of code changes, is as follows:

### Shift in Application Domain:

- **Why We Modified:** The original framework was tailored for supply chain issues solvable with Gurobi, an optimization solver. Ad placement optimization, on the other hand, often involves dynamic programming techniques, which require a different approach.
- **Example Change:** System messages were changed to reflect this new context.
  - **Original:** Messages were about Gurobi/Python solver solutions.
  - **Our Version:** Messages now guide towards dynamic programming solutions. For example, the `WRITER_SYSTEM_MSG` was changed from "...explain solutions from a Gurobi/Python solver" to "...explain solutions from a Python-based dynamic programming approach."

### Code Insertion Points:

- **Why We Modified:** Different types of optimization problems necessitate different types of code modifications.
- **Example Change:** We introduced new constants like `ALGORITHM_CODE_STR`.
  - **Original:** Used `CONSTRAINT_CODE_STR` for Gurobi model constraints.
  - **Our Version:** `ALGORITHM_CODE_STR` is used to mark where the core algorithm of the dynamic programming approach should be inserted.

### Initialization Process:

- **Why We Modified:** Ensuring the right placeholders for code insertion are present is crucial for the framework to function correctly.
- **Example Change:** Assertions in the `__init__` method were updated.
  - **Original:** Checked for placeholders relevant to Gurobi optimization.

- Our Version: Checks for DATA\_CODE\_STR and ALGORITHM\_CODE\_STR, confirming the setup is right for dynamic programming.

#### Helper Functions for Execution and Code Manipulation:

- Why We Modified: The execution and manipulation of code differ significantly between optimization solvers and dynamic programming algorithms.
- Example Change: The \_run\_with\_exec and \_insert\_code functions were adapted.
  - Original: Functions were designed to execute Gurobi models.
  - Our Version: \_run\_with\_exec now potentially interacts with an 'optimizer' object specific to dynamic programming, and \_insert\_code is tailored to insert dynamic programming logic.

#### Debugging and Interaction Prompts:

- Why We Modified: Different problem domains require different types of debugging and user interaction.
- Example Change: Updated prompts for code, debugging, and interpretation.
  - Original: Prompts were oriented towards Gurobi optimization.
  - Our Version: Prompts are now framed for debugging and interpreting results from dynamic programming algorithms.

In essence, our modifications were driven by the need to align the OptiGuide framework with the intricacies of ad placement optimization, which diverges significantly from supply chain optimization in terms of problem structure, solution techniques, and code requirements. By updating system messages, altering code insertion points, adjusting the initialization process, revamping helper functions, and refining debugging and interaction prompts, we made the framework suitable for the new application domain. The modified OptiGuide class can be accessed at:

[https://drive.google.com/file/d/1rSSe0KsVIYqczFvZNcQQXxoPIH\\_k52My/view?usp=sharing](https://drive.google.com/file/d/1rSSe0KsVIYqczFvZNcQQXxoPIH_k52My/view?usp=sharing)

## APPENDIX

Links to our question notebook outputs containing the code, questions, and complete chat transcripts:

- **Question 1:**  
<https://drive.google.com/file/d/1PvW9VsYV3aoYibOqMYyZGTRImqR-L7eM/view?usp=sharing>
- **Question 2:**  
<https://drive.google.com/file/d/1LBCQD-8Pi2LCUXg5Qx0ohdFqm0a4hAhk/view?usp=sharing>
- **Question 3:**  
[https://drive.google.com/file/d/13ePv\\_UDyKey9d26\\_lfkuQTIzpktBPk4Y/view?usp=sharing](https://drive.google.com/file/d/13ePv_UDyKey9d26_lfkuQTIzpktBPk4Y/view?usp=sharing)
- **Question 4:**  
[https://drive.google.com/file/d/1PUBrE7ucpbheWByRmJeUPTHaEntko2Ou/view?usp=drive\\_link](https://drive.google.com/file/d/1PUBrE7ucpbheWByRmJeUPTHaEntko2Ou/view?usp=drive_link)
- **Question 5:**  
[https://drive.google.com/file/d/1hkOyIDkzWhPKq7SCLjIL9iRm3q2Tg\\_hg/view?usp=drive\\_link](https://drive.google.com/file/d/1hkOyIDkzWhPKq7SCLjIL9iRm3q2Tg_hg/view?usp=drive_link)
- **Question 6:**  
[https://drive.google.com/file/d/1nB04xk8-7vl\\_P4C77Is6GaopKMhYYSXG/view?usp=drive\\_link](https://drive.google.com/file/d/1nB04xk8-7vl_P4C77Is6GaopKMhYYSXG/view?usp=drive_link)
- **Question 7:**  
[https://drive.google.com/file/d/16iiY6dYf3Sy4Nq4Mi1easKA6oDtuYPC3/view?usp=drive\\_link](https://drive.google.com/file/d/16iiY6dYf3Sy4Nq4Mi1easKA6oDtuYPC3/view?usp=drive_link)
- **Question 8:**  
[https://drive.google.com/file/d/16TXkZtcwBK3xkvfU1FbAZG0wiBlj4iMX/view?usp=drive\\_link](https://drive.google.com/file/d/16TXkZtcwBK3xkvfU1FbAZG0wiBlj4iMX/view?usp=drive_link)
- **Question 9:**  
[https://drive.google.com/file/d/11IQBwqtWM0s-ZEDnIHtta4QBz3L3bSqS/view?usp=drive\\_link](https://drive.google.com/file/d/11IQBwqtWM0s-ZEDnIHtta4QBz3L3bSqS/view?usp=drive_link)
- **Question 10:**  
[https://drive.google.com/file/d/1WPsBk40C84J65wBTHE1AnqI9jttBkiAV/view?usp=drive\\_link](https://drive.google.com/file/d/1WPsBk40C84J65wBTHE1AnqI9jttBkiAV/view?usp=drive_link)