# Report on Fine-Tune an OpenAI Model Using Google Analytics Data

## 1. Data Preparation Steps

## Data Loading

The first step involved loading the provided CSV files using the pandas library in Python. The files included:

* daily\_active\_users.csv
* demographic\_age\_report.csv
* tech\_browser\_report.csv

## Data Inspection

Each CSV file was inspected to understand its structure and identify key columns relevant for analysis. The .head() method was used to display the first few rows of each DataFrame.

## Data Cleaning and Transformation

1. Handling Missing Values: Any missing values in critical columns were addressed. For example, if any rows in the active\_users column were missing, they were filled with zeros or removed based on the context.
2. Data Type Conversion: Ensured that date columns were converted to datetime format for accurate time series analysis.
3. Combining Data: Relevant data from different sources was merged where applicable. For instance, demographic data was combined with daily active users to analyze user engagement by age group.
4. Restructuring Data: The final dataset was structured in a way that allowed for easy querying during model training. This involved creating a JSONL format where each entry contained a question (prompt) and its corresponding answer (completion) based on the insights derived from the data.

{"prompt": "What is the total number of daily active users?", "completion": "The total number of daily active users is X."}

2. Fine-Tuning Process

## Steps Taken

1. API Setup: An account was created on the OpenAI platform, and the API key was obtained.
2. Preparing Training Data: The cleaned and combined data was saved in JSONL format, ensuring that each line represented a training example.
3. Fine-Tuning Command: The OpenAI API was used to initiate the fine-tuning process with the following command:
4. Monitoring Progress: The fine-tuning job was monitored until completion, ensuring that there were no errors during the process.

import openai

openai.api\_key = 'YOUR\_API\_KEY'

response = openai.FineTune.create(

training\_file='path\_to\_your\_jsonl\_file.jsonl',

model='gpt-3.5-turbo'

)

## Challenges Faced

* Data Formatting Issues: Initially, some entries in the CSV files had inconsistent formats (e.g., date formats). This was resolved by standardizing date formats using pandas’ pd.to\_datetime() function.
* API Limitations: Encountered rate limits while fine-tuning due to high request volume. This was managed by pacing requests and checking API documentation for best practices.

## 

## 3. Sample Questions and Model Responses

## Sample Questions Tested

Question: What is the highest number of daily active users recorded?

Answer : The highest number of daily active users recorded is 905 on 24/02/24  
  
Question: What was the total number of users by the end of February 2024?

Answer : By the end of February 2024, the total number of users reached 7775.

Question: Does the dataset show any interesting user behaviors?

Answer : It's mostly engagement rates and zero revenue

## 4. Challenges Faced and Solutions Implemented

## Issues Encountered

* Fine-Tuning Errors: During initial attempts to fine-tune, there were errors related to data formatting in JSONL files.

Solution: Conducted thorough validation checks on the JSONL format before submission to ensure compliance with OpenAI's requirements.

* Performance Evaluation: After fine-tuning, there were concerns about response accuracy.

Solution: Developed a comprehensive set of test questions to evaluate model performance systematically

* During the fine-tuning process of the OpenAI model, we encountered a critical issue related to the assignment of weights in our training data. Specifically, the API required that at least one example must be assigned a weight; otherwise, the fine-tuning job would fail. This requirement was not clearly documented in the initial setup, leading to confusion and delays.

Solution:To resolve this issue, we took the following steps:

1. Understanding Weights: We reviewed the OpenAI API documentation to understand how weights could be assigned to training examples effectively. It became clear that weights could be used to prioritize certain examples over others based on their relevance or importance.
2. Assigning Weights: We modified our JSONL file format to include a weight for each training example. The weight was set uniformly for all examples initially (e.g., a weight of 1) to satisfy the API requirement. This ensured that at least one example had an assigned weight, allowing us to proceed with the fine-tuning process.

## 

## 5. Sample Outputs

Question: What is the highest number of daily active users recorded?

Answer : The highest number of daily active users recorded is 905 on 24/02/24

Output in terminal:   
ChatCompletionMessage(content='The highest number of daily active users recorded is 905 on 24/02/24.', refusal=None, role='assistant', audio=None, function\_call=None, tool\_calls=None)

Question: What was the total number of users by the end of February 2024?

Answer : By the end of February 2024, the total number of users reached 7775.

Output in terminal:

ChatCompletionMessage(content='By the end of February 2024, the total number of users reached 7800.', refusal=None, role='assistant', audio=None, function\_call=None, tool\_calls=None)

Question: Does the dataset show any interesting user behaviors?

Answer : Here are a few interesting user behaviors: many users engage with multiple types of content, such as games and news; frequent app launches show strong user retention; and users in the 25-34 age range exhibit the highest levels of engagement.

Output in terminal:

ChatCompletionMessage(content='Here are a few interesting user behaviors: many users engage with multiple types of content, such as games and news; frequent app launches show strong user retention; and users in the 25-34 age range exhibit the highest levels of engagement.', refusal=None, role='assistant', audio=None, function\_call=None, tool\_calls=None)