# EXP 3: Evaluation of Prompting Tools Across Diverse AI Platforms: ChatGPT, Claude, Bard, Cohere Command, and Meta

**Date : 18/10/2024**

---

## Aim:

To compare the performance, user experience, and response quality of different AI platforms (ChatGPT, Claude, Bard, Cohere Command, and Meta) within a specific use case, such as summarising text or answering technical questions.

## Procedure:

1. **Specified task:**
   ➔ Generation of Python code for a given question.

2. **Set of Prompts:**
   Employees of a company are planning a trip for all its team so that they can spend time together and get to know each other. The teams ul.com are represented by an array A where A[i] is the number of members present in the ith team. Each team has one particular member who has a car, and the number of seats of the car is represented by an array C where C[i] indicates the number of seats in the ith team member's car. The company trip organiser wants to use the minimum number of cars so that all the members who are willing to go on the trip can join. Each team who is ready to go for the trip is also ready to share their Sdaniecar with members from other teams. Your task is to find and return an integer value representing the minimum number of cars required for all the team members to go on the trip.

**Generate me a python code for the above question with better time and space complexity.**

| ChatGPT | Gemini | BlackBox.AI |
|---|---|---|
| def min_cars_for_trip(A, C):<br>  # Filter out teams with less than or equal to 2 members<br>  filtered_teams = [(members, seats) for members, seats in zip(A, C) if members > 2]<br><br>  # Sort teams based on car seating capacity in descending order<br>  filtered_teams.sort(key=lambda x: x[1], reverse=True)<br><br>  total_members = sum(members for members, _ in filtered_teams)<br>  total_seats = 0<br>  cars_used = 0 | from collections import deque<br><br>def min_cars_for_trip(teams, capacities):<br>  sorted_data = sorted(zip(teams, capacities), key=lambda x: (-x[1], x[0]))<br>  teams, capacities = zip(*sorted_data)<br><br>  remaining_seats = deque(capacities)<br><br>  total_members = sum(teams)<br><br>  min_cars = 0<br><br>  for team_size in teams:<br>  while team_size > 0 and | class UserMainCode(object):<br><br>  @classmethod<br>  def findCars(cls, input1, input2, input3):<br>  # Filter out teams with less than or equal to 2 members as they will not go<br>  teams_going = [(members, seats) for members, seats in zip(input2, input3) if members > 2]<br><br>  # If no teams are going, return -1<br>  if not teams_going:<br>  return -1<br><br>  # Calculate the total members |

| | | |
|---|---|---|
| for members, seats in filtered_teams:<br>if total_members <= 0:<br>break<br>cars_used += 1<br>total_seats += seats<br>total_members -= min(seats, members)<br><br>if total_members > 0:<br>return -1 # Not enough seats for all members<br>return cars_used<br><br># Example usage<br>input1 = 3<br>A = [4, 5, 7] # Team members<br>C = [7, 6, 7] # Car seating capacities<br><br>result = min_cars_for_trip(A, C)<br>print(result)<br><br>**Output:**<br><br>Output is not satisfied and the execution time is more**.** | remaining_seats:<br>current_car = remaining_seats[0]<br>if team_size <= current_car:<br>current_car -= team_size<br>team_size = 0<br>else:<br>team_size -= current_car<br>remaining_seats.popleft()<br>current_car = 0<br>if team_size > 0:<br>min_cars += 1<br>remaining_seats.append(4) # Assuming maximum car capacity is 4<br><br>return min_cars<br><br># Example usage<br>teams = [3, 5, 2]<br>capacities = [1, 4, 1]<br>min_cars = min_cars_for_trip(teams, capacities)<br>print(f"Minimum number of cars required: {min_cars}")<br><br>**Output:**<br><br>Generates error. | who are going<br>total_members = sum([team[0] for team in teams_going])<br><br># Sort the teams by seats in descending order<br>seats_available = sorted([team[1] for team in teams_going], reverse=True)<br><br># Keep a count of the number of cars used<br>cars_used = 0<br>members_remaining = total_members<br><br># Assign members to cars based on the seat capacity of each car<br>for seats in seats_available:<br>if members_remaining <= 0:<br>break<br>members_remaining -= min(members_remaining, seats)<br>cars_used += 1<br><br># If there are still members remaining, return -1 (not enough seats)<br>if members_remaining > 0:<br>return -1<br><br>return cars_used<br><br>**Output:**<br><br>Output satisfied and the execution time is efficient**.** |

## Conclusion:

By analysing the results provided by ChatGPT, Gemini, BlackBox.ai for the given question, we can infer that the code provided by **BlackBox.ai** tool is more **Clarity, Accurate and Efficient** in execution.