Prisoners Dilemma in the Business World, IOS vs Android

Introduction

My project is based on Prisoners Dilemma, a concept of Game Theory, a paradox that highlights the way rational individuals may not cooperate with one another even when it is in their best interests to do so. In most cases, two suspects must decide whether to cooperate or betray each other. If they both collaborate, they get a mild punishment; however, if one of them betrays the other, they will receive a very light penalty while the other will receive a harsh sentence.

The problem occurs because each suspect must choose between trusting the other to remain silent and betraying the other by confessing. If one suspect fears the other will betray them, it is reasonable for them to confess as well, even if both suspects would have been better off if they had both remained silent.

Subject Case Study

In real world, the study of prisoners dilemma between business competitors can be used to understand their strategic decisions.

For example, Both companies may gain from a larger market share and higher earnings if they collaborate by not using aggressive pricing or marketing techniques. However, if one company chooses to use aggressive techniques, such as decreasing prices or running negative advertising campaigns, they may acquire a competitive advantage and take a larger share of the market.

In this case, the other company is in a difficult position, if they chose to react with similar techniques, both companies may wind up losing money owing to greater competition. Alternatively, companies risk losing market share to the aggressive opponent if they choose to collaborate rather than engage in such actions.

In my project and report we are going to discuss and study similar tensions between two leaders of smartphone across world, namely; IOS and Android.

Study of the game (IOS vs Android)

I am going to consider the two players of my game as IOS and the other as Android. We are going to execute the prisoners dilemma by constructing a payoff matrix. The payoff matrix will be concretely constructed based on a data set taken from internet which includes the percentage of share between both the players.

The values of payoff matrix will always be taken from that dataset in each and every simulation. However, the dataset I am considering for my project really does not have huge variations in their values.

The link is the reference of the dataset. https://gs.statcounter.com/os-market-share/mobile/worldwide/2022

The main payoff matrix based on the dataset that I am going to consider on which the strategies of the game would be simulated is going to be,

	Android Cooperate	Android Defects
Ios Cooperate	5,5	7,3
Ios Defects	3,7	3,3

Strategies of the game:

In the game we are going to design the payoff matrix for unique simulation only. However, the based on the respective matrix we shall consider different types of strategies that shall be taken into account to play our game between both the players.

We are going to construct the payoff matrices for each strategy based on the following information, here the payoff could be considered as the profits based on the market share.

- If both companies cooperate (C,C), they both receive a payoff of 3.
- If both companies defect (D,D), they both receive a payoff of 1.
- If one company cooperates and the other defects (C,D or D,C), the defector receives a payoff of 4 and the cooperator receives a payoff of 0.

Strategy 1 : Tit for Tat

In this method, each company begins by cooperating in the first round and then just replicates the previous move of the other company in each successive round. This strategy is a form of reciprocal altruism, where each company rewards the other for cooperating and punishes the other for defecting.

	Android Cooperate	Android Defects
Ios Cooperate	3,3	0,4
Ios Defects	4,0	1,1

Strategy 2 : Grim Trigger

In this strategy, each company starts by cooperating in the first round, but switches to defecting if the other company ever defects. Once a company has defected, it continues to defect for the rest of the game, leading to a "grim" outcome for both companies.

	Android Cooperate	Android Defects
Ios Cooperate	3,3	0,4
Ios Defects	0,4	1,1

Strategy 3 : Random

In this strategy, each company randomly chooses to cooperate or defect in each round. This strategy is unpredictable and doesn't rely on any specific pattern, but can still lead to interesting outcomes.

	Android Cooperate	Android Defects
Ios Cooperate	3,3	0,4
Ios Defects	4,0	1,1

Strategy 4 : Always Cooperate

In this strategy, regardless of what the other firm does, each company always decides to work together in every round. This is a type of altruism in which each corporation points out the collective advantage of collaboration over its own benefit.

	Android Cooperate	Android Defects
Ios Cooperate	3,3	0,4
Ios Defects	3,0	1,1

Strategy 5 : Always Defect

In this approach, regardless of what the other firm does, each company always chooses to defect in every round. This is a selfish approach in which each corporation puts its individual advantage over the collective benefit of cooperation.

	Android Cooperate	Android Defects
Ios Cooperate	1,4	0,1
Ios Defects	4,1	1,1

Working of Code

My project is based on python programming language. Below is the explanation of how the above mentioned strategies and its respective payoff matrices are written and converted into code which when simulated gives the output in the form of a mathematical plot.
All the code which has to be executed is attached and explained below in different parts.

First, the code imports the required Python libraries, numpy and matplotlib. Numpy is used for performing numerical computations and matplotlib is used for data visualization.

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

The code then defines the game's payoff matrices. The game contains two players, Android and Ios, and each has two options: cooperate or defect. The payoffs are represented in a 2x2 matrix for each player, where the rows indicate the other player's approach and the columns represent the player's own strategy.

```
android_payoffs = np.array([[5, 5], [7, 3]])
```

```
ios_payoffs = np.array([[3, 7], [3, 3]])
```

The code then defines five different strategies that can be used by the players in the game, represented as numpy arrays.

```
always_android = np.array([1, 0])
```

```
always_ios = np.array([0, 1])
```

```
tit_for_tat = np.array([1, 1])
```

```
random_choice = np.array([0.5, 0.5])
```

```
grim_trigger = np.array([1, 0])
```

The function get_total_payoff() is defined next. This function takes two-player strategies, payoff matrices for each player, and calculates the total reward for each player based on their chosen strategies and payoffs for each possible outcome in the game. The total payoff for each participant is returned by the function.

```
def get_total_payoff(strategy1, strategy2, payoffs1, payoffs2):
    payoff1 = np.dot(strategy1, payoffs1.dot(strategy2))
    payoff2 = np.dot(strategy2, payoffs2.dot(strategy1))
    return payoff1, payoff2
```

The simulate_game() function is then defined. This function accepts two-player strategies, payoff matrices for each player, and the number of rounds to be simulated. The function simulates the game for the specified number of rounds, recording the total payoffs for each participant at the end of each round. It returns two lists: one with the total payoffs for the first player after each round and another with the total payoffs for the second player after each round.

```
def simulate_game(strategy1, strategy2, payoffs1, payoffs2, num_rounds):
    total_payoffs1 = []
    total_payoffs2 = []
    for i in range(num_rounds):
        payoff1, payoff2 = get_total_payoff(strategy1, strategy2, payoffs1, payoffs2)
        total_payoffs1.append(payoff1)
        total_payoffs2.append(payoff2)
        # Update the strategies for the next round
        if i > 0:
            last_play1 = strategy1[-1]
            last_play2 = strategy2[-1]
            strategy1 = np.append(strategy1[1:], last_play2)
            strategy2 = np.append(strategy2[1:], last_play1)
    return total_payoffs1, total_payoffs2
```

Finally, the code defines the plot_results() function. This function takes in the total payoffs for the two players and plots them on a graph using matplotlib.

```
def plot_results(total_payoffs1, total_payoffs2):
    plt.plot(total_payoffs1, label='android')
    plt.plot(total_payoffs2, label='ios')
    plt.xlabel('Round')
    plt.ylabel('Total Payoff')
    plt.legend()
    plt.show()
```

After defining the plot_results function, the code defines the main simulation loop. It initializes a list total_payoffs to store the total payoffs for each strategy, and then loops over the different strategies. The total payoff for the strategy is computed as the sum of the payoffs received by both players in each round of the game, and is added to the total_payoffs list.

Inside the loop, the plot_results function is called with the payoffs for the current simulation, in order to generate a plot of the payoffs over time for the two players.
After the loop is completed, the total payoffs for each strategy are shown in a bar chart using matplotlib, with the x-axis labeled with the strategy names and the y-axis labeled "Total Payoff." This figure visualizes the performance of each strategy over the course of the simulation.

```
num_rounds = 10

total_payoffs = []

strategies = [always_android, always_ios, tit_for_tat, random_choice, grim_trigger]

for strategy in strategies:

   total_payoffs1, total_payoffs2 = simulate_game(strategy, tit_for_tat, android_payoffs, ios_payoffs, num_rounds)

   total_payoff = sum(total_payoffs1 + total_payoffs2)

   total_payoffs.append(total_payoff)

   plot_results(total_payoffs1, total_payoffs2)
```

The final part depicts of how the results are plotted

```
plt.bar(range(len(strategies)), total_payoffs)

plt.xticks(range(len(strategies)), ['Always Cooperate', 'Always Defect', 'Tit-for-Tat', 'Random Choice', 'Grim Trigger'])

plt.xlabel('Strategy')

plt.ylabel('Total Payoff')

plt.show()
```

Conclusion and Result

Below is the explanation of how the dot product works, lets consider the given matrices,
strategy1 = [1, 0]

strategy2 = [1, 1]

payoffs1 = [[5, 5], [7, 3]]

payoffs2 = [[3, 7], [3, 3]]

The first step is to multiply the **payoffs1** matrix with the **strategy2** vector using the **dot** method:
payoffs1.dot(strategy2) = [10, 16]

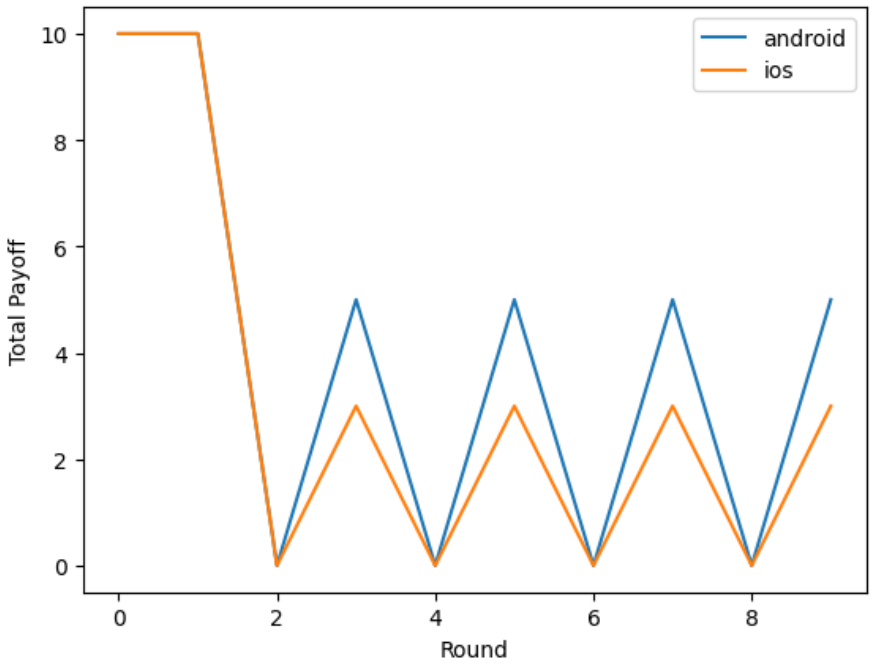
This result represents the payoffs player 1 would receive based on the actions of both players.

Then, take the dot product of the resulting vector with the strategy1 vector
np.dot(strategy1, payoffs1.dot(strategy2)) = np.dot([1, 0], [10, 16]) = ***10***

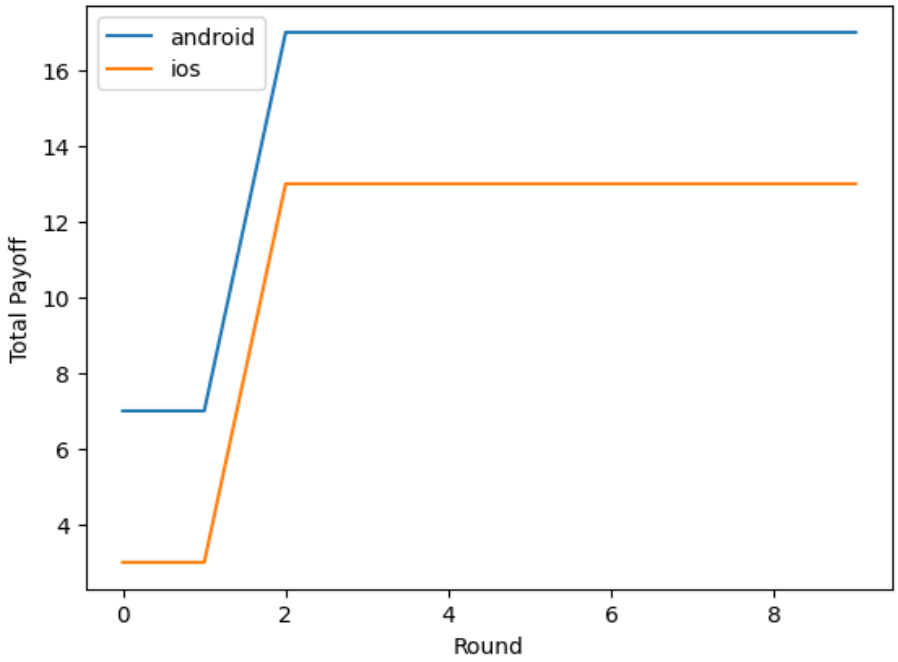
This result represents the total payoff for player 1.
Similarly, Using this way we can calculate the dot product of matrices as required from the code. This is why we can see, in the first graph, the value ranges to **10**

Conclusion and Result

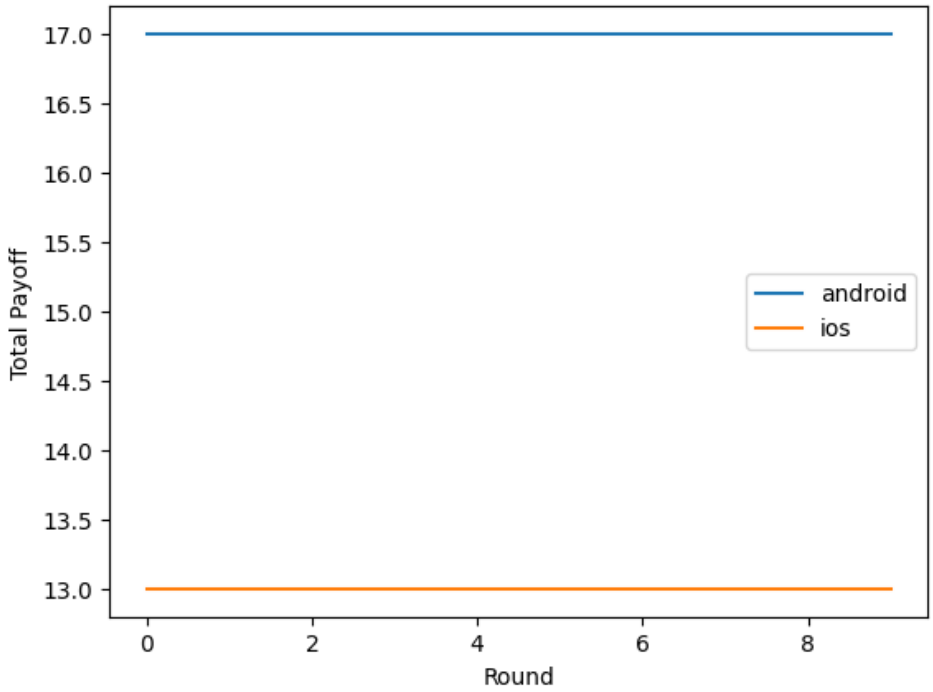
After the code is run, the following conclusion can be depicted,
Graph 1 : The following plot is obtained when the simulation is run for certain rounds VS the payoffs obtained. Here the strategy followed is “Always Android”.



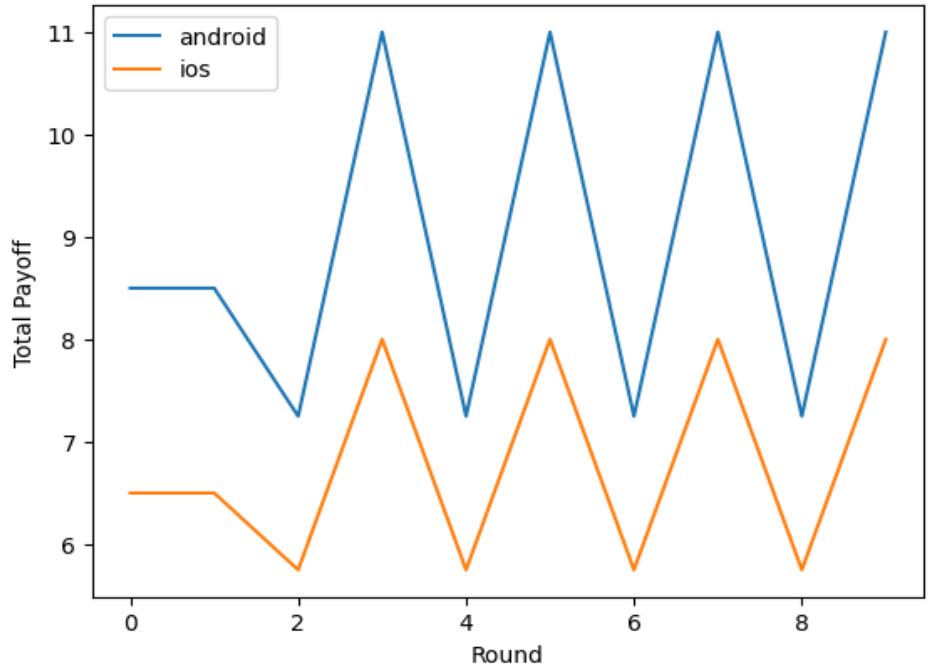
Graph 2: The following plot is obtained when the simulation is run for certain rounds VS the payoffs obtained. Here the strategy followed is “Always Ios”.



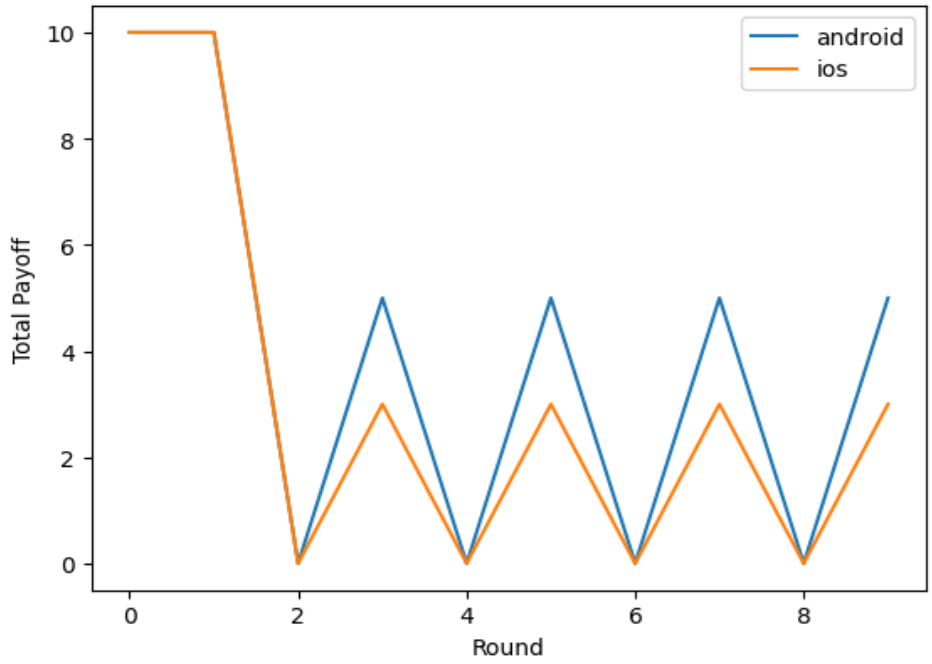
Graph 3: The following plot is obtained when the simulation is run for certain rounds VS the payoffs obtained. Here the strategy followed is “Tit for Tat”.



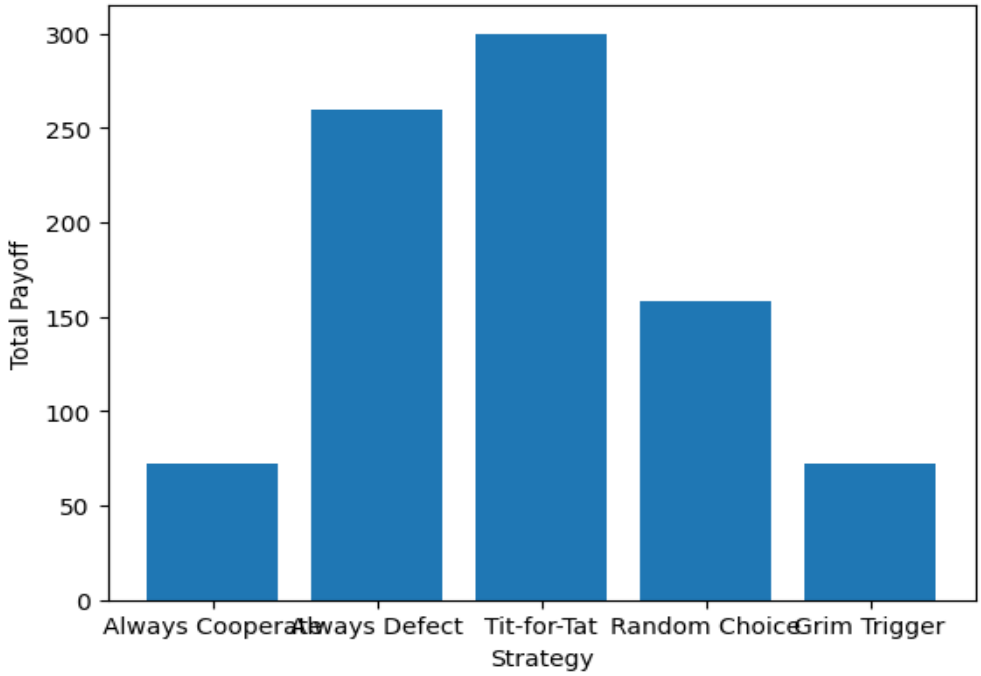
Graph 4: The following plot is obtained when the simulation is run for certain rounds VS the payoffs obtained. Here the strategy followed is “Random Choice”.



Graph 5: The following plot is obtained when the simulation is run for certain rounds VS the payoffs obtained. Here the strategy followed is “Grim Trigger”.



Final Plot : Considering the data from Total Payoffs vs Rounds Played of all the plots, the below plot shows the most significant strategy which can be obtained from performing the simulation.



Based on the experiment's outcomes, it appears that the "Tit-for-Tat" method performs the best overall, having the largest total payoff of all five strategies. "Always Cooperate" and "Random Choice" also do well, whereas "Always Defect" and "Grim Trigger" do not. This implies that in this case, cooperation is often preferable than defection. However, it is important to note that the simulation results may vary based on the specific situations and initial conditions used, and this is only one possible outcome.

Resources

- *https://blogs.cornell.edu/info2040/2019/09/25/coca-cola-pepsico-and-the-prisoners-dilemma/*
- *https://www.investopedia.com/articles/investing/110513/utilizing-prisoners-dilemma-business-and-economy.asp*
- *https://plato.stanford.edu/entries/prisoner-dilemma/strategy-table.html*
- *https://gs.statcounter.com/os-market-share/mobile/worldwide/2022*