Project Report

Table of Contents

About Data set

The data set has entries 499 Among Us games. Let's look at one document (one entry in the data) representing one game. There are four high-level fields:

1. game
2. Game_Feed
3. player_data
4. voting_data

The game is a unique identifier for each game. The other three fields are divided into nested documents with more granular-level data.

A game feed is an array of objects where each object is an event in the game. Each game in Among Us has multiple events.

Task1. Read the data (10%)

1.1 Read the data and examine the collections.

mongoimport -d dbAmongUS -c Among_Us_data --type json --file AmongUs.json --jsonArray

db.Among_Us_data.find().limit(5)

```
C:\mongosh-2.1.1-win32-x64\mongosh-2.1.1-win32-x64\bin>mongoimport -d dbAmongUS -c Among_Us_data --ty
pe json --file AmongUs.json --jsonArray
2023-12-20T13:28:56.660-0500    connected to: mongodb://localhost/
2023-12-20T13:28:56.796-0500    499 document(s) imported successfully. 0 document(s) failed to import
.
```

```
dbAmongUS> db.Among_Us_data.find().limit(5)
[
  {
    _id: ObjectId('658332685e46d1fdb63db778'),
    game: '2',
    Game_Feed: [
      {
        Event: 1,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Dreamy',
        Action: 'kills',,
        Player: 'Mani',Skipped Vote.',
        Role: '(Crew).',
        'Game Feed': 'Dreamy (Impostor) kills Mani (Crew).',
        Day: 1,'7-2'
        'Votes Off Code': '',
        'Vote ID': '',
        'Day 1 vote': '',
        'Crew Alive': 8,
        'Impostors Alive': 2,
        Score: '8-2'': 'Crew',
      },Action: 'skips voting.',
      { Player: '',
        Event: 2,
        Map: 'Polus','Crew skips voting.',
        Outcome: '',
        'Player/Team': 'dokomoy',
        Action: 'finds body of',
        Player: 'Mani',',
        Role: '(Crew).',
        'Game Feed': 'dokomoy (Impostor) finds body of Mani (Crew).',
```

1.2 Display data for matches where the "game" field equals "3".

db.Among_Us_data.find({'game':{$eq:'3'}}).pretty()

```
dbAmongUS> db.Among_Us_data.find({'game':{$eq:'3'}}).pretty()
[
  {
    _id: ObjectId('658332685e46d1fdb63db779'),
    game: '3',
    Game_Feed: [
      {
        Event: 1,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'kills',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) kills BK (Crew).',
        Day: 1,
        'Votes Off Code': '',
        'Vote ID': '',
        'Day 1 vote': '',
        'Crew Alive': 8,
        'Impostors Alive': 2,
        Score: '8-2'
      },
      {
        Event: 2,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'finds body of',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) finds body of BK (Crew).',
        Day: 1,
        'Votes Off Code': 0,
        'Vote ID': '3-1',
        'Day 1 vote': 'Skipped Vote.',
        'Crew Alive': 7,
        'Impostors Alive': 2,
        Score: '7-2'
      },
      {
        Event: 3,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Crew',
        Action: 'skips voting.',
        Player: '',
        Role: '',
        'Game Feed': 'Crew skips voting.',
        Day: 1,
        'Votes Off Code': '',
        'Vote ID': '',
        'Day 1 vote': '',
        'Crew Alive': 7,
        'Impostors Alive': 2,
        Score: '7-2'
```

Task2. Explore the data for game 3 (40%)

In this subtask, you are expected to create a new collection with only the document relating to game 3.

var game3Document = db.Among_Us_data.findOne({'game': '3'})

db.Game3Collection.insertOne(game3Document)

```
dbAmongUS> var game3Document = db.Among_Us_data.findOne({'game': '3'})

dbAmongUS> db.Game3Collection.insertOne(game3Document)
{
  acknowledged: true,
  insertedId: ObjectId('658332685e46d1fdb63db779')
}
dbAmongUS> db.Game3Collection.find().pretty()
[
  {
    _id: ObjectId('658332685e46d1fdb63db779'),
    game: '3',
    Game_Feed: [
      {
        Event: 1,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'kills',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) kills BK (Crew).',
        Day: 1,
        'Votes Off Code': '',
        'Vote ID': '',
        'Day 1 vote': '',
        'Crew Alive': 8,
        'Impostors Alive': 2,
        Score: '8-2'
      },
      {
        Event: 2,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'finds body of',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) finds body of BK (Crew).',
        Day: 1,
        'Votes Off Code': 0,
```

2.1. Show the Game Feed data specifically for game 3 in the newly created collection.

db.Game3Collection.find({}, { Game_Feed: 1, _id: 0 }).pretty()

```
dbAmongUS> db.Game3Collection.find({}, { Game_Feed: 1, _id: 0 }).pretty()
[
  {
    Game_Feed: [
      {
        Event: 1,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'kills',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) kills BK (Crew).',
        Day: 1,
        'Votes Off Code': '',
        'Vote ID': '',
        'Day 1 vote': '',
        'Crew Alive': 8,
        'Impostors Alive': 2,
        Score: '8-2'
      },
      {
        Event: 2,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Keaton',
        Action: 'finds body of',
        Player: 'BK',
        Role: '(Crew).',
        'Game Feed': 'Keaton (Impostor) finds body of BK (Crew).',
        Day: 1,
        'Votes Off Code': 0,
        'Vote ID': '3-1',
        'Day 1 vote': 'Skipped Vote.',
        'Crew Alive': 7,
        'Impostors Alive': 2,
        Score: '7-2'
      },
      {
        Event: 3,
        Map: 'Polus',
        Outcome: '',
        'Player/Team': 'Crew',
        Action: 'skips voting.',
        Player: '',
        Role: '',
        'Game Feed': 'Crew skips voting.',
        Day: 1,
        'Votes Off Code': '',
        'Vote ID': '',
        'Day 1 vote': '',
        'Crew Alive': 7,
        'Impostors Alive': 2,
        Score: '7-2'
```

2.2. Show the most recent event that occurred in game 3.

db.Game3Collection.find({}, { "Game_Feed": { $slice: -1 }, _id: 0 })

The most recent event in Game 3 is the last Event in Game_Feed, so I use $slice: -1 select the last element in the Game_Feed array, which is the Event10, and exclude the document's _id field

2.3. Determine the winner of game 3 - imposters or crew.

Answer: Crew

db.Game3Collection.find({"Game_Feed.Outcome":{$regex:"End"}},

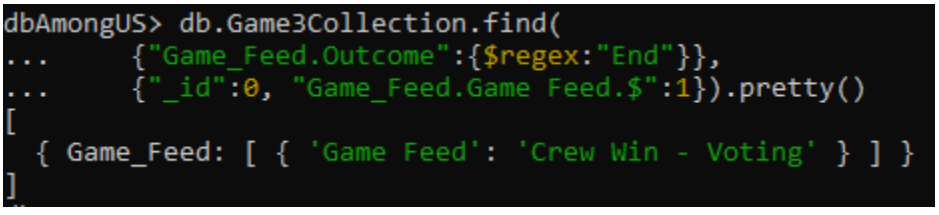
{"_id":0, "Game_Feed.Game Feed.$":1}).pretty()

{"Game_Feed.Outcome":{$regex:"End"}}: Specifies to search for documents whose Outcome field contains the word "End" in the elements of the Game_Feed array.

{"_id":0, "Game_Feed.Game Feed.$":1}: Specify which fields should be included in the query results.

"_id":0 means not to include the document's _id field in the results.

"Game_Feed.Game Feed.$":1 is a special projection operator that returns the Game Feed field matching the first element in the Game_Feed array that satisfies the $regex:"End" condition. The $ operator only extracts the first array element that matches the query criteria.



2.4. Identify the player who chose the black color in game 3 and whether they were a crew member or imposter.

Answer: Name: Keaton, Role: Impostor

db.Game3Collection.findOne({"player_data.Color":"Black"},{'player_data.$':1,_id:0})

Query the player_data field of the collection and return the player_data field (excluding the _id field).

The player_data array is searched for players with the colour "black", returning documents for players who chose black in match 3, including whether they were crewmates or imposters.

```
dbAmongUS> db.Game3Collection.findOne({"player_data.Color":"Black"},{'player_data.$':1,_id:0})
{
  player_data: [ { Player: 9, name: 'Keaton', Role: '(Impostor)', Color: 'Black' } ]
}
```

2.5. Count the number of voting events that took place in game 3.

Answer: 3

db.Game3Collection.aggregate([

{ $match: { 'game': '3' } },

{ $unwind: '$voting_data' },

{ $group: { _id: '$voting_data.Vote_Event' } },

{ $count: 'total_voting_events' }])

$unwind: '$voting_data': Expand the voting_data array so that each element in the array is converted into an independent document.

$group: { _id: '$voting_data.Vote_Event' }: Group documents according to the Vote_Event field, so that each different Vote_Event will be counted as a group.

$count: 'total_voting_events': Calculate how many different Vote_Events there are in total, that is, the total number of voting events.

```
dbAmongUS> db.Game3Collection.aggregate([
...     { $match: { 'game': '3' } },
...     { $unwind: '$voting_data' },
...     { $group: { _id: '$voting_data.Vote_Event' } },
...     { $count: 'total_voting_events' }
... ])
[ { total_voting_events: 3 } ]
```

2.6. If you were redesigning this database for easier querying, describe the changes you would make and explain your reasoning.

Normalize Data Structures:

Split the data into related collections instead of a single collection with nested arrays. This would reduce data redundancy and make updates easier. For instance, player_data, Game_Feed, and voting_data can be separate collections, linked by a game_id.

Use consistent naming conventions and data types for fields across collections to make queries more intuitive.

Timestamps for Events:

Include timestamp fields for game events and voting events. This allows for more straightforward chronological sorting and querying of events.

Separate Collection for Player Profile:

Instead of repeating player information in each game's player_data, have a separate collection for player profiles. Link these profiles to games through player IDs. This reduces redundancy and makes updating player information easier.

Flatten Document Structures:

Reduce the depth of nested documents. Deeply nested structures can make queries complex and slow. Flattening the structure, while maintaining relationships through references, can simplify queries.

Include Aggregation Data in the Document:

For frequently calculated data (like the total number of voting events), consider storing this aggregated data directly in the game document if it doesn't change often. This reduces the need for aggregation queries.

Task3. Overall aggregation (40%)

3.1. Calculate the total number of events recorded in this collection across all games.

Answer: 5889

db.Among_Us_data.aggregate([

{ $unwind: "$Game_Feed" }, { $group: { _id: null, totalEvents: { $sum: 1 } } }])

$unwind: "$Game_Feed": This operation deconstructs the Game_Feed array fields in each document and outputs a document for each element of the arrays.

$group: { _id: null, totalEvents: { $sum: 1 } }: This groups all the unwound documents together (since _id is set to null, which means no specific grouping field) and sums up the count of documents, which corresponds to the total number of events.

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $unwind: "$Game_Feed" },
...     { $group: { _id: null, totalEvents: { $sum: 1 } } }])
[ { _id: null, totalEvents: 5889 } ]
```

3.2. Compare the crew's wins to the impostors' wins and provide the counts.

Answer: Crew wins 323 times, impostors win 176 times

db.Among_Us_data.aggregate([

{ $project: { lastEvent: { $arrayElemAt: ["$Game_Feed", -1] } } },

{ $project: { outcome: "$lastEvent.Game Feed" } },

{ $group: { _id: { $cond: { if: { $regexMatch: { input: "$outcome", regex: "Crew Win" } }, then: "Crew", else: "Impostor" } }, count: { $sum: 1 } }}])

Get the last event first: Use the $project operator combined with $arrayElemAt to extract the last event in the Game_Feed array of each document.

Then extract the victory type: Use the $project operator to extract the Game Feed field in the last event of each game, which contains the victory type information.

Finally, group and count: use the $group operator to group by victory type and use the $cond operator combined with $regexMatch to determine whether "Crew" wins or "Impostor" wins. Then use the $sum operator to calculate the number of games in each group

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $project: { lastEvent: { $arrayElemAt: ["$Game_Feed", -1] } } },
...     { $project: { outcome: "$lastEvent.Game Feed" } },
...     { $group: {
...         _id: { $cond: { if: { $regexMatch: { input: "$outcome", regex: "Crew Win" } },
...                         then: "Crew", else: "Impostor" } },
...         count: { $sum: 1 } }}])
[ { _id: 'Crew', count: 323 }, { _id: 'Impostor', count: 176 } ]
```

3.3. List the maps played and the total number of games on each map.

Answer: "MIRA HQ": 7, "The Skeld": 88, "Polus": 404

```
db.Among_Us_data.aggregate([
  { $unwind: "$Game_Feed" },
  { $match: {"Game_Feed.Event":1}},
  { $group: { _id: "$Game_Feed.Map", count: { $sum:1}}}])
```

First expand the Game_Feed array. A new document is created for each array element.

Then use $match to filter those documents whose Game_Feed.Event field value is 1. Because the map values of all events in each game are the same.

Finally, use $group to group the documents by the Game_Feed.Map field. For each distinct Map value, it counts the number of times that value occurs and stores this count in the count field. This is accomplished using "$sum":1, which increments it by one each time it encounters a document.

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $unwind: "$Game_Feed" },
...     { $match: {"Game_Feed.Event":1}},
...     { $group: { _id: "$Game_Feed.Map", count: { $sum:1}}}])
[
  { _id: 'MIRA HQ', count: 7 },
  { _id: 'Polus', count: 404 },
  { _id: 'The Skeld', count: 88 }
]
```

3.4. Determine the total instances of crew members skipping a vote across all games.

Answer: 693

```
db.Among_Us_data.aggregate([
  { $unwind: "$Game_Feed" },
  { $match: { "Game_Feed.Votes Off Code": 0,}},
  { $count: "skippedVotes" } ])
```

First, expand the Game_Feed array: use the $unwind operator to expand the Game_Feed array in each document.

Then filter the events that skip voting: Use the $match operator to filter out those events whose Votes Off Code field is 0.

Finally count: Use the $count operator to calculate the total number of events that meet the criteria.

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $unwind: "$Game_Feed" },
...     { $match: { "Game_Feed.Votes Off Code": 0,}},
...     { $count: "skippedVotes" } ])
[ { skippedVotes: 693 } ]
```

3.5. Calculate the total occurrences of crew members voting against imposters across all matches.

Answer: 639

```
db.Among_Us_data.aggregate([
  { $unwind: "$Game_Feed" },
  { $match: { "Game_Feed.Votes Off Code": 2,}},
  { $count: "skippedVotes" } ])
```

First unwind the Game_Feed Array: use the $unwind operator to expand the Game_Feed array in each document.

Then match the Specific Voting Events: Use the $match operator to filter out those events whose Votes Off Code field is 2.

Finally count: Use the $count operator to calculate the total number of events that meet the criteria.

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $unwind: "$Game_Feed" },
...     { $match: { "Game_Feed.Votes Off Code": 2,}},
...     { $count: "skippedVotes" } ])
[ { skippedVotes: 639 } ]
```

3.6. Share your opinion on whether the game is more or less challenging for impostors, supported by insights from the data.

Win Rates: Crew members have won 323 times, while impostors have won 176 times. This significant difference in win rates suggests that the game might be more challenging for impostors. A higher win rate for the crew indicates that they are more effective, on average, at completing their tasks and identifying impostors.

Skipping Votes: There are 693 instances of crew members skipping votes. Skipped votes can imply uncertainty or a lack of consensus among crew members. This situation could be advantageous for impostors as it might indicate missed opportunities to identify and vote off impostors. However, the fact that the crew still wins more often suggests they overcome this challenge effectively.

Voting Against Impostors: There are 639 instances of crew members voting against impostors. This high number, combined with the crew's higher win rate, might indicate that crew members are generally successful in identifying and voting off impostors, adding to the challenge for impostors to remain undetected.

In conclusion, the data suggests that the game poses a greater challenge for impostors. The higher win rate for crew members, combined with a substantial number of events where the crew votes against impostors, indicates that impostors might find it more difficult to deceive the crew and achieve their objectives.

Task4. Player-level aggregation (10%)

4.1. Find the count of unique players in the dataset.

Answer: 108

db.Among_Us_data.distinct("player_data.name").length

```
dbAmongUS> db.Among_Us_data.distinct("player_data.name").length
108
```

4.2. Identify the player considered the best crew member.

Answer: "BK"

```
db.Among_Us_data.aggregate([
  { $unwind: "$voting_data"},
  { $match: {"voting_data.Vote": { $regex:"Impostor voted off"}} },
  { $group: { _id:"$voting_data.name", Count: { $sum:1}}},
  { $sort: { Count: -1 } },
  { $limit: 1 }])
```

Analyze specific patterns in the voting data in the Among_Us_data collection to find the players who were most successful in identifying and voting to expel the Impostor from the game.

Use **$unwind** to unwind the voting_data array so that each array element becomes a separate document.

Use **$match** to filter documents whose voting_data.The vote field value matches the regular expression "Impostor voted off". This is the event where the Impostor was successfully cast in the Find All Polls.

Use **$group** to group documents by the voting_data.name field (i.e. the name of the voting player). For each different player, it counts the number of times they successfully cast the Impostor and stores this count in the Count field.

Use **$sort** and **$limit** to sort the results in descending order based on the value of the Count field. The highest Count value will be ranked first, and the player with the most successful Impostor rolls out of all players will be returned.

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $unwind: "$voting_data"},
...     { $match: {"voting_data.Vote": { $regex:"Impostor voted off"}} },
...     { $group: { _id:"$voting_data.name", Count: { $sum:1}}},
...     { $sort: { Count: -1 } },
...     { $limit: 1 }])
[ { _id: 'BK', Count: 178 } ]
```

4.3. Identify the player regarded as the least effective crew member.

Answer: "Sam"

```
db.Among_Us_data.aggregate([
  { $unwind: "$voting_data"},
  { $match: {"voting_data.Vote": { $regex:"Crew voted off"}} },
  { $group: { _id:"$voting_data.name", Count: { $sum:1}}},
  { $sort: { Count: -1 } },
  { $limit: 1 }])
```

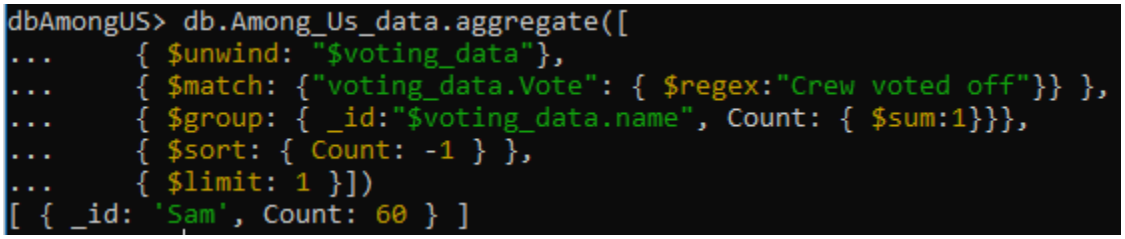
Analyze specific patterns in the voting data in the Among_Us_data collection to identify players who are most likely to misjudge the game and result in the expulsion of Crew members.

Use **$unwind** to unwind the voting_data array so that each array element becomes a separate document.

Use **$match** to filter documents whose voting_data. The vote field value matches the regular expression "Crew voted off". Only keep those cases where the vote shows that a Crew member was wrongfully evicted.

Use **$group** to group documents by the voting_data.name field (i.e. the name of the voting player). For each different player, it counts the number of times they mistakenly evicted a Crew member and stores this count in the Count field.

Use **$sort** and $limit to sort the results in descending order based on the value of the Count field. Only the player with the most mistakenly expelled Crew members out of all players is returned.



4.4. Compute the win percentage for each player. (Optional)

```
db.Among_Us_data.aggregate([
  { $unwind: "$player_data"},
  { $unwind: "$Game_Feed"},
  { $match: { "Game_Feed.Outcome": { $regex:"End"}} },
  { $project: { "player_data.name":1,
                Role : { $substr: ["$player_data.Role", 1, 4 ]} ,
                Result : { $substr: ["$Game_Feed.Action", 0, 4 ]} }},
  { $project: { "player_data.name":1, Win: { $cond: [{ $eq: ["$Result", "$Role"]},1,0]} }},
  { $group: { _id:"$player_data.name", Played:{ $sum:1}, Wins: { $sum : "$Win" }}},
  { $project: { _id:1, "Played":1, Wins:1,
              WinRate:{ $multiply: [{ $divide: ["$Wins", "$Played"]} ,100]}}},
  { $sort: { WinRate: -1, Wins: -1 } }])
```

Analyze a player's win rate by counting the number of games and wins for each player

Use **$unwind** to expand the player_data array so that a new document is created for each array element. Use $unwind to expand the Game_Feed array again so that each Game_Feed element in the original document becomes a separate document.

Use **$match** to filter out documents whose Game_Feed.Outcome field contains "End".

The **first $project**: projection operation, extracts the player_data.name and Role fields. The Role is obtained by intercepting the first 4 characters of the player_data.Role field. Similarly, the Result is obtained by intercepting the first 4 characters of the Game_Feed.Action field.

The **second $project**: Another projection operation, calculates the Win field. If Result and Role are equal, the value is assigned to 1 (representing victory), otherwise it is 0.

Use **$group** to group the documents by player_data.name and calculate the total number of games played (Played) and the number of games won (Wins) by each player.

Project again with **$project**, this time calculating each player's win rate (WinRate) by dividing the number of games won by the total number of games played and multiplying by 100.

Finally, use $sort to sort in descending order based on winning rate and number of games won.

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $unwind: "$player_data"},
...     { $unwind: "$Game_Feed"},
...     { $match: { "Game_Feed.Outcome": { $regex:"End"}} },
...     { $project: {
...         "player_data.name":1,
...         Role : { $substr: ["$player_data.Role", 1, 4 ]} ,
...         Result : { $substr: ["$Game_Feed.Action", 0, 4 ]} }},
...     { $project: {
...         "player_data.name":1,
...         Win: { $cond: [{ $eq: ["$Result", "$Role"]},1,0]} }},
...     { $group: {
...         _id:"$player_data.name",
...         Played:{ $sum:1},
...         Wins: { $sum : "$Win" }}},
...     { $project: {
...         _id:1, "Played":1,
...         Wins:1,
...         WinRate:{ $multiply: [{ $divide: ["$Wins", "$Played"]} ,100]}}},
...     { $sort: { WinRate: -1, Wins: -1 } }])
[
  { _id: 'ZJ', Played: 7, Wins: 7, WinRate: 100 },
  { _id: 'synthe', Played: 5, Wins: 5, WinRate: 100 },
  { _id: 'julie', Played: 4, Wins: 4, WinRate: 100 },
  { _id: 'Isaac', Played: 3, Wins: 3, WinRate: 100 },
  { _id: 'CALC', Played: 2, Wins: 2, WinRate: 100 },
  {
    _id: 'ScaldingHotSoup',
    Played: 15,
    Wins: 13,
    WinRate: 86.66666666666667
  },
  { _id: 'Sup', Played: 17, Wins: 14, WinRate: 82.35294117647058 },
  { _id: 'flutter', Played: 35, Wins: 28, WinRate: 80 },
  { _id: 'Squirrel_Loot', Played: 15, Wins: 12, WinRate: 80 },
  { _id: 'GGards', Played: 5, Wins: 4, WinRate: 80 },
  { _id: 'Dix', Played: 17, Wins: 13, WinRate: 76.47058823529412 },
  { _id: 'Rafon', Played: 8, Wins: 6, WinRate: 75 },
  { _id: 'Aberdasher', Played: 4, Wins: 3, WinRate: 75 },
  { _id: 'Zyla', Played: 30, Wins: 22, WinRate: 73.33333333333333 },
  { _id: 'Nairbly', Played: 37, Wins: 27, WinRate: 72.97297297297297 },
  { _id: 'Ben P', Played: 7, Wins: 5, WinRate: 71.42857142857143 },
  { _id: 'TomM', Played: 59, Wins: 42, WinRate: 71.1864406779661 },
  { _id: 'Iroknight', Played: 10, Wins: 7, WinRate: 70 },
  { _id: 'FoxBox', Played: 16, Wins: 11, WinRate: 68.75 },
  {
    _id: 'Chosaucer',
    Played: 121,
    Wins: 83,
    WinRate: 68.59504132231406
  }
]
Type "it" for more
```

4.5. Determine the color preferences chosen by all players. (Optional)

```
db.Among_Us_data.aggregate([
  { $unwind: "$player_data"},
  { $group: {
    _id: { Player: "$player_data.name", Color: "$player_data.Color"},
    Total: { $sum: 1 }}},
  { $group: { _id: "$_id.Player", Colors: { $push: { Color: "$_id.Color", Count:"$Total"}}}},
  { $unwind: "$Colors"},
  { $sort: { "Colors.Count": -1 }},
  { $group: { _id: "$_id", details: { $push: "$Colors" }}},
  { $project: { _id:1 , colors: { $first: "$details"}}},
  { $project: { _id:1 , Preference: "$colors.Color" }}])
```

Counting how many times each colour is used by each player, then sort the colours to find the most used ones.

Use **$unwind** to expand the player_data array so that each array element creates a new document.

The **first $group**: Groups the document by player name and colour, counting the number of times each colour is used by each player.

The **second $group**: groups the document by player name and pushes each color and its count as an object into the Colors array.

Use **$unwind** to expand the Colors array so that each array element creates a new document.

Use **$sort** to sort colours in descending order based on the number of times they are used.

The **third $group**: groups the document by player name and pushes the sorted Colors array into the details field.

The **first $project**: projection operation holds a details field for each player, which contains a list of colours sorted by the number of uses.

The **second $project**: projects again, extracts each player's favourite colour (the most used colour) from the details field, and names its Preference.

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $unwind: "$player_data"},
...     { $group: {
...         _id: { Player: "$player_data.name", Color: "$player_data.Color"},
...         Total: { $sum: 1 }}},
...     { $group: {
...         _id: "$_id.Player",
...         Colors: { $push: { Color: "$_id.Color", Count:"$Total"}}}},
...     { $unwind: "$Colors"},
...     { $sort: { "Colors.Count": -1 }},
...     { $group: { _id: "$_id", details: { $push: "$Colors" }}},
...     { $project: { _id:1 , colors: { $first: "$details"}}},
...     { $project: { _id:1 , Preference: "$colors.Color" }}])
[
  { _id: 'Andrew', Preference: 'Cyan' },
  { _id: 'Kibler', Preference: 'Orange' },
  { _id: 'wrapter', Preference: 'Cyan' },
  { _id: 'Seabats', Preference: 'Orange' },
  { _id: 'Fader', Preference: 'Purple' },
  { _id: 'Pheylop', Preference: 'Red' },
  { _id: 'nucleosynth', Preference: 'Green' },
  { _id: 'Bim', Preference: 'Yellow' },
  { _id: 'Dix', Preference: 'Lime' },
  { _id: 'Adrien', Preference: 'Red' },
  { _id: 'dokomoy', Preference: 'Cyan' },
  { _id: 'Eric', Preference: 'Lime' },
  { _id: 'Squirrel_Loot', Preference: 'Blue' },
  { _id: 'Ben P', Preference: 'Pink' },
  { _id: 'Corey', Preference: 'Cyan' },
  { _id: 'Voxy', Preference: 'Cyan' },
  { _id: 'jorbs', Preference: 'Pink' },
  { _id: 'LegenVD', Preference: 'Yellow' },
  { _id: 'sponsz', Preference: 'Brown' },
  { _id: 'DBatterskull', Preference: 'Blue' }
]
Type "it" for more
```

Task5. Export from MongoDB (Optional)

Create an export from MongoDB in the form given below

Player name	Games won as imposter	Games won as crew	Win percentage (overall)	Voted Against Imposter	Voted against Crew members	Color preference	Voting rate

Save the results of win, voting and color to collections respectively, then use "_id" as the key value to merge the collections, rename the fields and finally use mongoexport to export the final results as a CSV file

Save the results of "Color preference" to a collection called 'Color'.

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $unwind: "$player_data"},
...     { $group: {
...         _id: { Player: "$player_data.name", Color: "$player_data.Color"},
...         Total: { $sum: 1 }}},
...     { $group: {
...         _id : "$_id.Player",
...         Colors: { $push: { Color: "$_id.Color", Count:"$Total"}} }},
...     { $unwind: "$Colors" },
...     { $sort: { "Colors.Count": -1 } },
...     { $group: { _id: "$_id", details: { $push : "$Colors" }}},
...     { $project: { _id:1 , colors: { $first: "$details"}}},
...     { $project: { _id:1 , Preference: "$colors.Color" }},
...     { $out: "Color"}])

dbAmongUS>
```

Save the results of "Games won as imposter", "Games won as crew", "Win percentage(overall)" to a collection called 'Win'.

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $unwind: "$player_data"},
...     { $unwind: "$Game_Feed"},
...     { $match: { "Game_Feed.Outcome": { $regex:"End"}} },
...     { $project: {
...         "player_data.name": 1,
...         Role: { $substr: [ "$player_data.Role", 1, 4 ]} ,
...         Result: { $substr: [ "$Game_Feed.Action", 0, 4 ]}}},
...     { $project: {
...         "player_data.name": 1,
...         Win: { $cond: [{ $eq:["$Result","$Role"]}, 1, 0]},
...         Crew_Win: { $cond: [{ $and:[{ $eq:["$Result","Crew"]},{ $eq:["$Role","Crew"]}]},1,0]},
...         Imposter_Win: { $cond: [{ $and:[{ $eq:["$Result","Impo"]},{ $eq:["$Role","Impo"]}]},1,0]}}},
...     { $group: {
...         _id: "$player_data.name",
...         Played: { $sum:1 },
...         Wins: { $sum: "$Win" },
...         Win_As_Crew: { $sum: "$Crew_Win" },
...         Win_As_Imposter: { $sum: "$Imposter_Win" }}},
...     { $project: {
...         _id: 1,
...         Win_As_Crew: 1,
...         Win_As_Imposter: 1,
...         Win_Rate:{$multiply:[{ $divide:["$Wins", "$Played"]}, 100]}}},
...     { $out: "Win"}])

dbAmongUS>
```

Save the results of "Voted Against Imposter", "Voted against Crew members", “Voting rate" to a collection called ‘Voting’.

```
dbAmongUS> db.Among_Us_data.aggregate([
...     { $unwind: "$voting_data"},
...     { $match: { "voting_data.Is_alive": { $regex:"Yes"}} },
...     { $project: {
...         "voting_data.name": 1,
...         "AgainstCrew":{ $cond:[{ $regexMatch:{ "input":"$voting_data.Vote", "regex": "Crew"}},1,0]},
...         "AgainstImpo":{ $cond:[{ $regexMatch:{ "input":"$voting_data.Vote", "regex": "Impostor"}},1,0]} }},
...     { $group: {
...         _id: "$voting_data.name",
...         Voted_Against_Crew: { $sum : "$AgainstCrew" },
...         Voted_Against_Impo: { $sum : "$AgainstImpo" },
...         Voting_Opportunities: { $sum: 1} }},
...     { $project: {
...         _id: 1,
...         Voted_Against_Crew: 1,
...         Voted_Against_Impo: 1,
...         Voting_rate:
...         { $multiply:[{$divide:[{$add:["$Voted_Against_Crew","$Voted_Against_Impo"]},"$Voting_Opportunities"]} ,100]} }},
...     { $out: "Voting"}])

dbAmongUS>
```

Use “_id” as the key value to merge the collections. Merge the collection ‘Win’ and the collection ‘Voting’ into the temporary collection ‘temp’, and then merge’ temp’ with the collection ‘Color’.

```
dbAmongUS> db.Win.aggregate([
...     { $lookup: {
...         from: "Voting",
...         localField: "_id",
...         foreignField: "_id",
...         as: "fromVoting" }},
...     { $unwind: { path: "$fromVoting", preserveNullAndEmptyArrays: true }},
...     { $merge: {
...         into: "temp",
...         whenMatched: "merge",
...         whenNotMatched: "insert" }}])

dbAmongUS>
```

```
dbAmongUS> db.Color.aggregate([
...     { $lookup: {
...         from: "temp",
...         localField: "_id",
...         foreignField: "_id",
...         as: "fromWin_Voting" }},
...     { $unwind: { path: "$fromWin_Voting", preserveNullAndEmptyArrays: true }},
...     { $merge: {
...         into: "Final_data",
...         whenMatched: "merge",
...         whenNotMatched: "insert" }}])

dbAmongUS>
```

Rename the fields

```
dbAmongUS> db.Final_data.aggregate([
...     { $project: {
...         _id:0,
...         "Player name": "$_id",
...         "Games won as imposter": "$fromWin_Voting.Win_As_Imposter",
...         "Games won as crew": "$fromWin_Voting.Win_As_Crew",
...         "Win percentage(overall)": "$fromWin_Voting.Win_Rate",
...         "Voted Against Imposter": "$fromWin_Voting.fromVoting.Voted_Against_Impo",
...         "Voted against Crew members": "$fromWin_Voting.fromVoting.Voted_Against_Crew",
...         "Color preference": "$Preference",
...         "Voting rate": "$fromWin_Voting.fromVoting.Voting_rate" }},
...     { $out: "AmongUs_Export"}])

dbAmongUS>
```

Finally, use mongoexport to export the final results as a CSV file.

```
C:\mongosh-2.1.1-win32-x64\mongosh-2.1.1-win32-x64\bin>mongoexport --db=dbAmongUS --collection=AmongUs_Export
 --type=csv --fields="Player name","Games won as imposter","Games won as crew","Win percentage(overall)","Vot
ed Against Imposter","Voted against Crew members","Color preference","Voting rate" --out=D:/DataScience/AdvSQ
L/AmongUs_Export.csv
2023-12-30T19:53:52.932-0500    connected to: mongodb://localhost/
2023-12-30T19:53:52.939-0500    exported 108 records
```

	A	B	C	D	E	F	G	H
1	Player name	Games won as imposter	Games won as crew	Win percentage(overall)	Voted Against Imposter	Voted against Crew members	Color preference	Voting rate
2	bsweitz	11	82	59.61538462	101	50	White	52.7972028
3	Stranjak	1	1	66.66666667	2	0	White	100
4	CALC	1	1	100			Blue	
5	Memes	5	40	57.69230769	46	24	Green	55.11811024
6	Ian	9	71	66.11570248	66	36	Yellow	50
7	DoubleFried	3	43	56.79012346	54	15	Pink	56.09756098
8	zlubars	9	71	52.98013245	91	31	Green	52.36051502
9	Nairbly	4	23	72.97297297	23	4	Lime	50
10	Riley	0	3	37.5	1	2	Blue	18.75
11	GrahamLRR	0	4	40	5	5	Cyan	45.45454545
12	Barnabus	6	45	63.75	44	31	Brown	48.38709677
13	MurkLurker	0	0	0	0	1	Brown	20
14	SamSherman	0	0	0	0	2	Blue	66.66666667
15	Dreamy	0	21	52.5	32	14	Pink	56.79012346
16	Jason	1	34	46.66666667	40	14	Yellow	43.2
17	DaveWilliams	0	9	39.13043478	7	7	Black	41.17647059
18	Ambiance	1	3	57.14285714	4	1	White	55.55555556
19	dcsports8	0	26	56.52173913	30	11	Red	57.74647887
20	Nate	0	1	50	2	0	Purple	100

Task6. Discuss additional statistics (Optional)

In tasks 4 and 5, you create individual player statistics. Discuss additional statistics that might be worth exploring (aside from those already calculated) while selecting players. (There's no requirement to write code for them.)

First Vote Accuracy: The accuracy of a player's first vote in identifying an impostor. This metric can reveal how quickly a player can deduce the impostor's identity.

Survival Rate: The proportion of games in which a player survives until the end. This statistic is particularly relevant for crew members and indicates their ability to avoid being eliminated.

Kill-to-Death Ratio (for Impostors): For players acting as impostors, this ratio compares the number of kills to the number of times they were voted off or killed. It reflects the effectiveness of an impostor in eliminating crew members while evading detection.

Average Game Duration: The average length of games in which the player participates. This can indicate how quickly the player tends to resolve games, either through efficient problem-solving or rapid elimination strategies.