Abstract

This project looks into the competitive balance on the de_mirage map in Counter-Strike: Global Offensive, or CS:GO for short. In esports, keeping things fair means the map layout should not give one side a clear edge in stats. With a huge set of match logs to work from, the goal here was to figure out if the map setup tilts results toward the Terrorist side or the Counter-Terrorist side. To deal with all that data, a Python setup using stream processing helped sort through millions of damage moments. The end result came as a sharp geospatial heatmap on top of an interactive dashboard showing control over areas. The findings point to almost even balance, where Counter-Terrorists handle 50.4 percent of the damage and Terrorists take 49.6 percent. That puts an end to the usual talk in the community about unfairness, since even though the sides hold different amounts of space, the total damage stays balanced in the math.

1. Datasets

Data Retrieval

The primary dataset was retrieved from Kaggle, titled "CS:GO Matchmaking Damage".

- **Source URL:** https://www.kaggle.com/datasets/skihikingkevin/csgo-matchmaking-damage

Data Description

The starting data came from several CSV files that added up to about 4GB. Those files held millions of entries for single damage events in competitive games.

- **Key Attributes:** Key details included the file for match ID, att_side for the attacking team, hp_dmg for damage given, att_pos_x and att_pos_y for position coordinates, and wp for the weapon picked.
- **Data Types:** The data mixed categories like team names and weapon names with numbers for damage and spots, plus some metadata on file IDs.

Big Data Aspects

- **Volume:** Handling the size proved the biggest hurdle at 4GB, which went beyond what regular spreadsheet tools like Excel could manage in rows. That meant turning to Python for the processing work.
- **Variety:** To add more layers, the raw game logs got combined with a structured file called weapon_stats.csv that was put together by hand. This mix brought in different kinds from live play data to fixed info on weapon costs.

2. Data Exploration, Processing, Cleaning and Integration

Preparation Pipeline

Since the data pile was too big for full load into RAM, Python with Pandas used a chunking approach to set up a solid ETL pipeline for extracting, transforming, and loading.

1. **Stream Processing:** Stream processing started with a loop reading the CSVs in batches of 100,000 rows.
2. **Filtering:** Right away, filters cut out anything not needed, like matches on maps besides de_mirage or events with no damage where hp_dmg equaled zero.
3. **Coordinate Transformation:** For coordinates, the original game used a Cartesian system from minus 3000 to plus 3000. To fit that onto a 1024 by 1024 radar image, a formula scaled the points, flipped the Y-axis to fit image rules, and mirrored the X-axis.
4. **Formula used:** The formula went like this, X_new equals 1024 minus ((X_game minus Offset) divided by Scale).

Attribute Selection

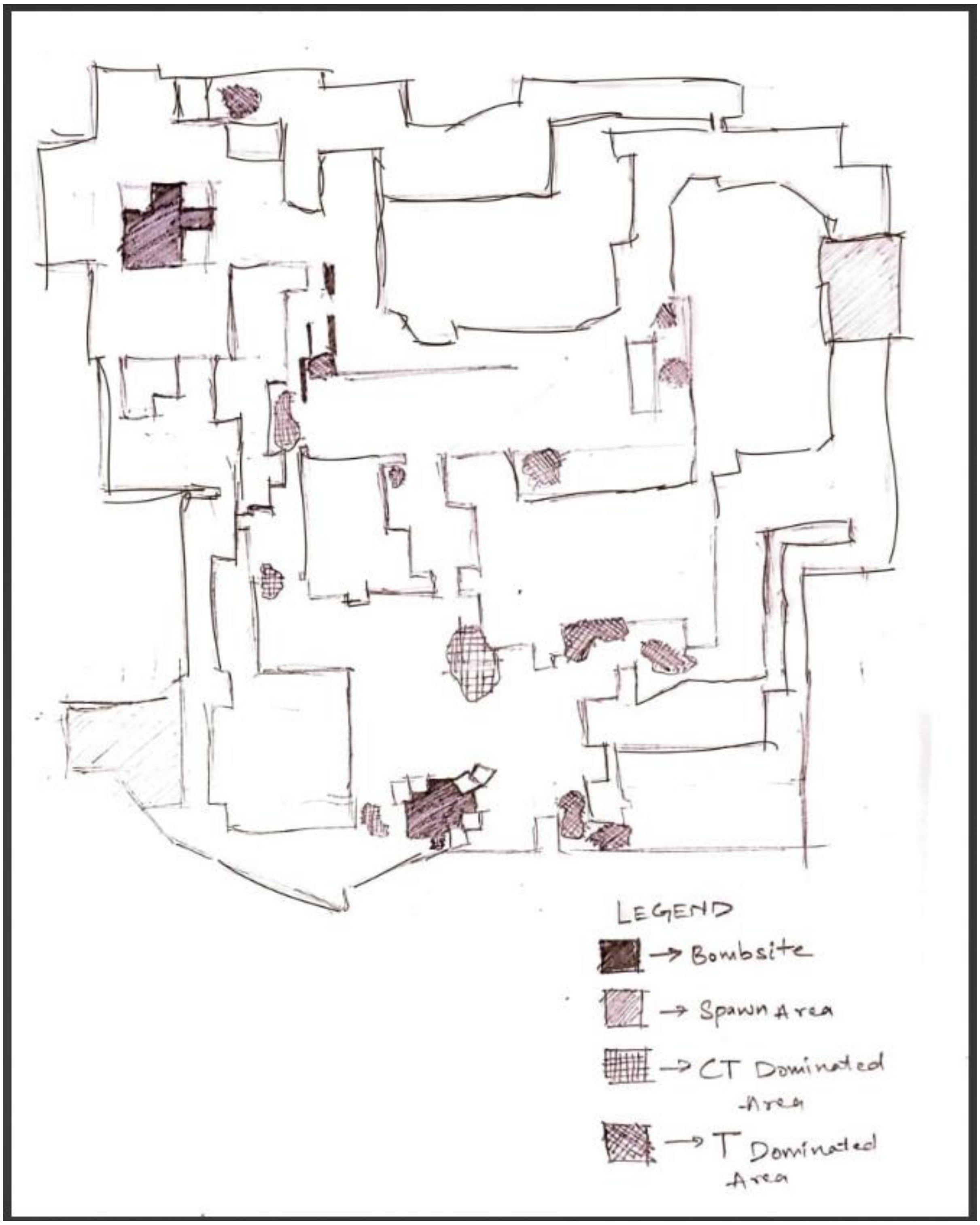
In picking attributes, att_pos_x, att_pos_y, and att_side stood out for visualization.

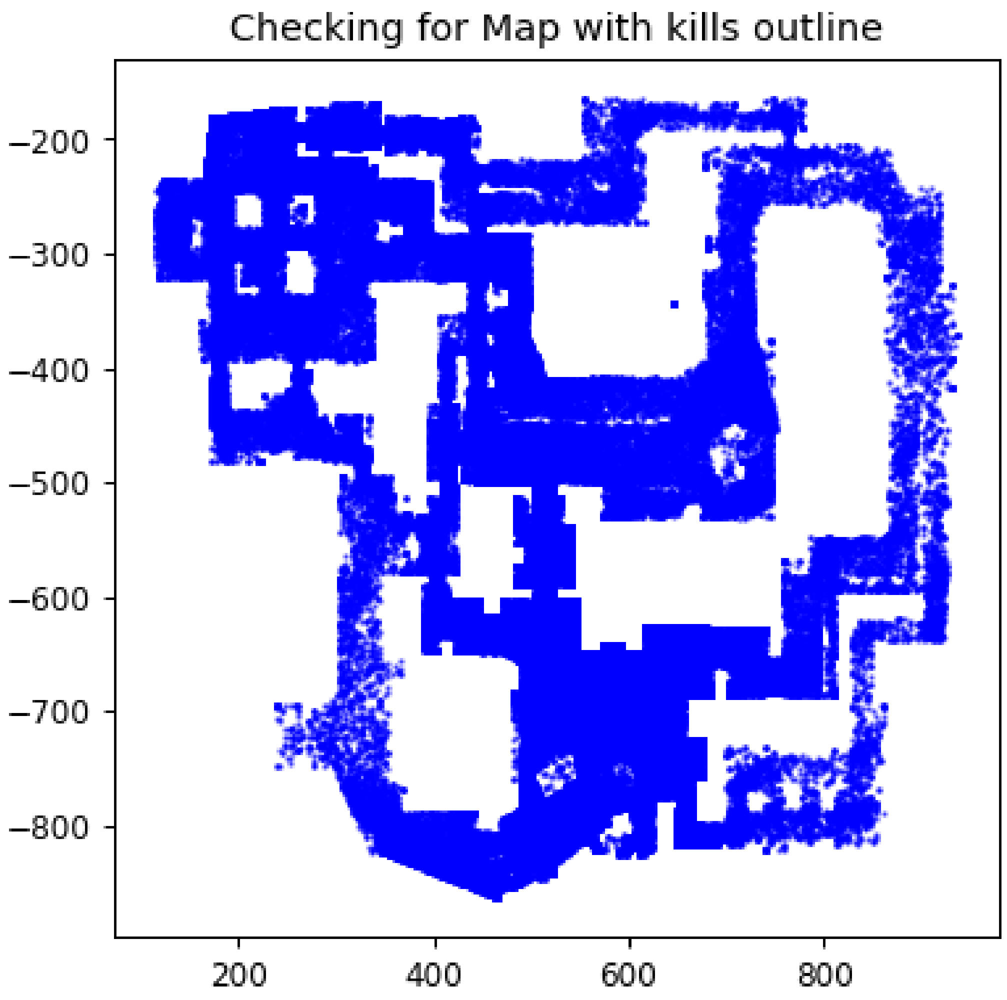
- **Why:** The main question focused on space and control, so showing attacker spots captured active hold on the map instead of just where deaths happened.
- **Integration:** A left join merged in the weapon stats to check engagement styles, and missing spots for weapon names got filled, like making USP and USP-S the same.

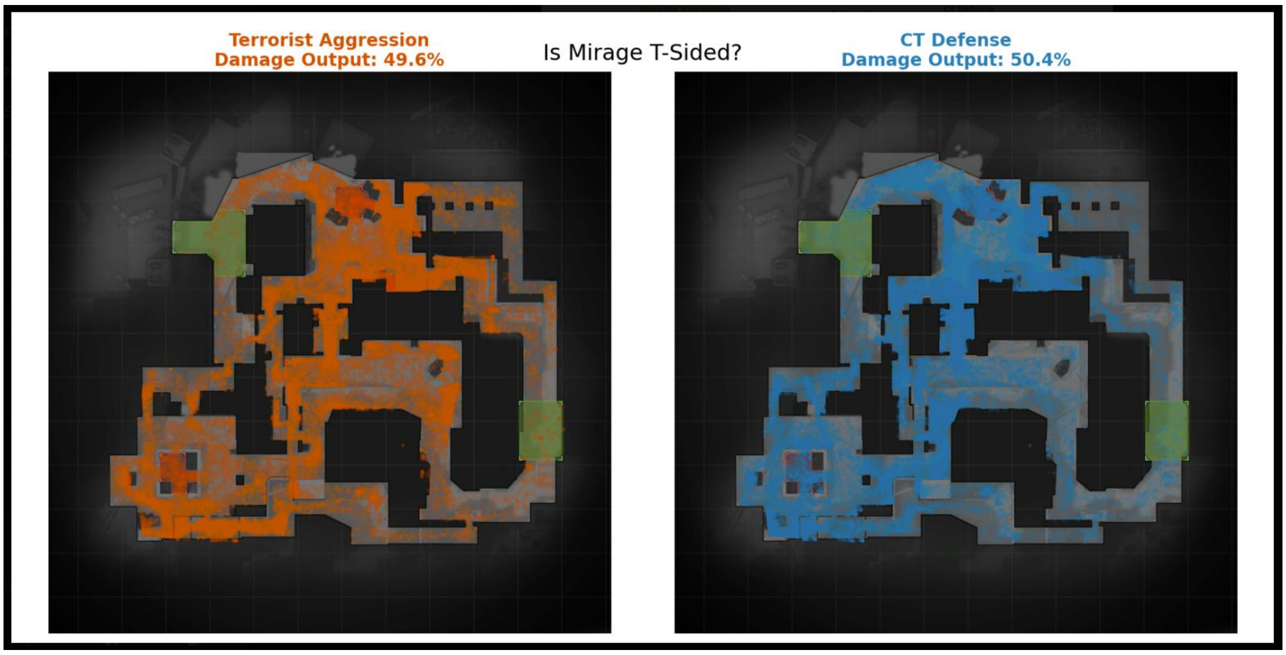
3. Visualisation

Design Process

The design kicked off with hand sketches for a territory map idea. Simple scatter plots just made messy noise, so the shift went to a differential heatmap showing pressure differences between teams.


LEGEND
Bombsite
Spawn Area
CT Dominated Area
T Dominated Area


Checking for Map with kills outline
−200
−300
−400
−500
−600
−700
−800
200
400
600
800


Terrorist Aggression
Damage Output: 49.6%
Is Mirage T-Sided?
CT Defense
Damage Output: 50.4%

Final Visualisation

Below is the High-Definition Heatmap generated by the Python pipeline, featuring the static labels generated for the final report.

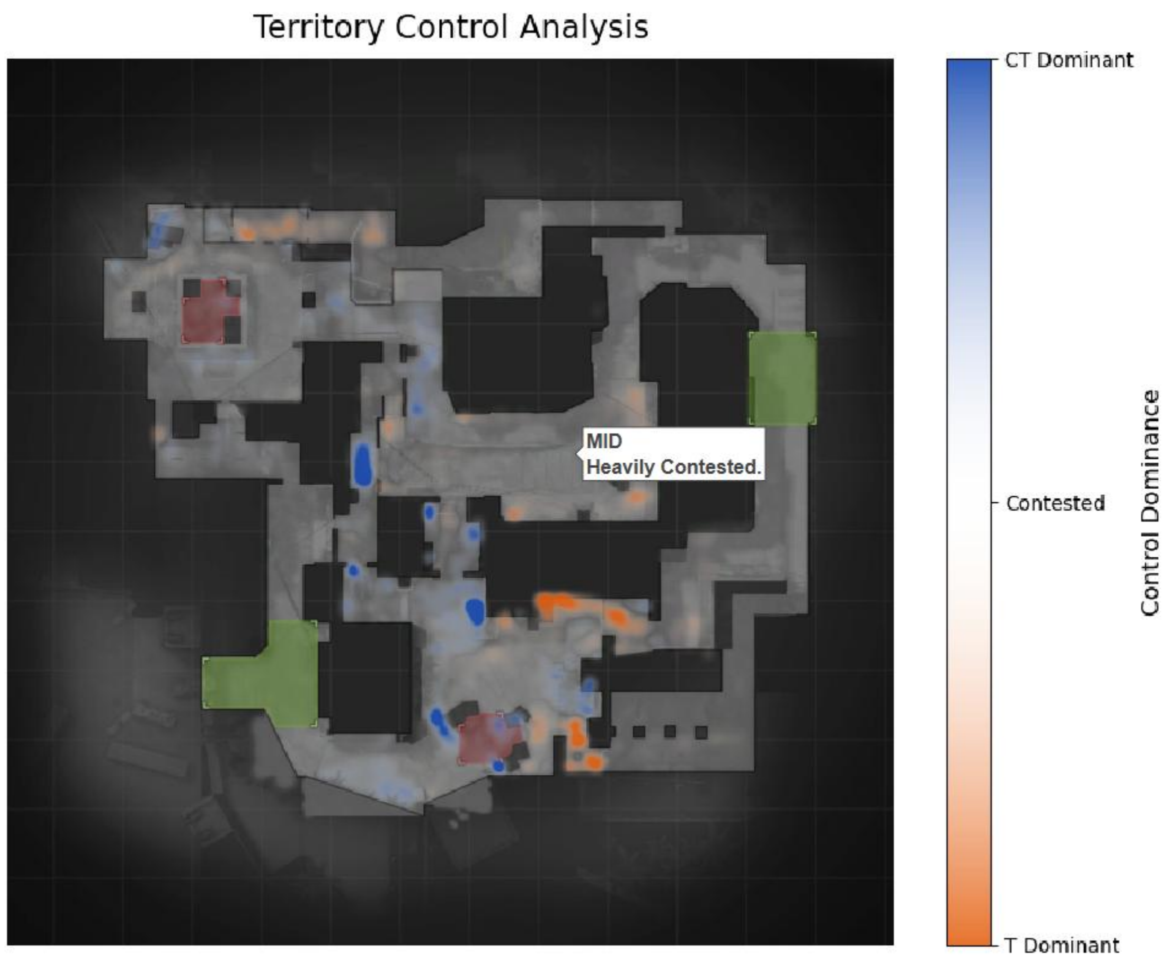


Chart Type & Justification

A geospatial heatmap over the map layout fit the bill for the chart type.

- **Relationship:** It captured spatial spread, and while a bar chart might tally total damage, it would miss where the action took place.
- **Colour Encoding:** We utilized a diverging colormap:
  - **Orange:** Represents Terrorist dominance.
  - **Blue:** Represents Counter-Terrorist dominance.
  - **White:** For even ground where damage is matched up.

  - **Justification:** Orange and blue contrast well against the grey background, so zones pop right away and stay easy to read. Also, these are the original colours in the game.

Marks & Layout

- **Density Calculation:** Density built on a 1024 by 1024 grid instead of single points that overlap too much, calculating delta as sum of CT minus sum of T.
- **Opacity:** Opacity varied to keep the map walls and shapes visible underneath, helping viewers tie data to the layout.

Interactivity

Plotly handled the interactive HTML dashboard for the deliverable.

- **Hitboxes:** Invisible zones sat over key spots like Mid or A Site.
- **Function:** Hovering brought up tooltips on tactical status, such as heavily contested areas.
- **Communication Value:** This kept text from crowding the map, staying clean while digging into details without paragraphs everywhere.

4. Conclusion

Tools Used

- **Pandas:** Pandas managed the chunking and data tweaks.
- **Matplotlib:** Matplotlib built the high-res static heatmap.
- **Plotly:** Plotly set up the web dashboard with interactivity.
- **Scikit-Image:** Scikit-Image dealt with image tweaks at the pixel level.

Critical Analysis

The visualization nails the research question. A white zone runs down the map middle as the front line for that even split. It shows damage stays equal, but patterns differ, with Terrorists holding big open spaces in orange and Counter-Terrorists gripping tight points in blue.

Improvements & Limitations

- **Velocity:** One limit stayed on velocity, missing the time flow. An upgrade could animate the heatmap across a round from zero to 120 seconds, tracking control shifts as time ticks.

- **3D Visualization:** Since CS:GO plays in 3D, the 2D top view loses height differences, like underpass against window. A 3D scatter might bring back that layer.

Collaboration

We worked as a pair through a GitLab repo for code sharing. Aditya took the Python backend for data processing and Matplotlib output, while Ritik handled the frontend with Plotly and hitboxes. We teamed up on design picks and the report write-up.

5. References

1. **Dataset:** Kevin, S. (2023). CS:GO Matchmaking Damage. Kaggle. Available at: https://www.kaggle.com/datasets/skihikingkevin/csgo-matchmaking-damage
2. **Tool:** Pandas Documentation - Read CSV Chunksize. Available at: https://pandas.pydata.org/docs/
3. **Tool:** Plotly Graphing Libraries. Available at: https://plotly.com/python/
4. **AI Declaration:** Generative AI from Google Gemini helped with grammar corrections and references.