Victory Vision
Revamped

Dana Fulmer, Shayne Powell, Aaron Swan, Spencer Buck, Sal Sonmez

The Goal

⇒Our goal was to further determine whether or not it is possible to predict the victorious party of a singular match-up or tournament. This was done through the access of multiple APIs that contain large collections of statistics for different competitive Video Games (eSports). The data gathered was then cleaned and put through alternate models(XGBoost, Neural Network, etc.).

GRID API

GRID API

⇒GRID API is an API built to strictly provide competitive multiplayer game data. The API tracks every heartbeat of a game directly into their server databanks.

⇒Games we had available:

- Counter Strike 2
- Dota 2

⇒Examples of Paid API Games:

- League of Legends
- Valorant
- PlayerUnknowns Battlegrounds (PUBG)

CS2 Analytics

Shayne Powell

Data Cleaning

⇒To Clean the data we cycled through the data collected, and either removed and/or encoded the object values into numerical values

- One Hot Encoder
- Label Encoder

⇒We then removed the players collected that had 0 in all stats beside player_id

All Zero Stat Removal Command

- 484 players
- 46 players with all 0 stats
- 438 players with all stats accounted for

```python
def remove_zero_stat_players(df, stat_columns):
    """
    Remove rows from a DataFrame where all specified stat columns have a value of 0.

    Parameters:
    df (pandas.DataFrame): DataFrame containing player stats
    stat_columns (list): List of column names to check for zeros.
                         If None, uses all numeric columns except index

    Returns:
    pandas.DataFrame: DataFrame with zero-stat players removed
    """
    # If no stat columns specified, use all numeric columns
    if stat_columns is None:
        stat_columns = df.select_dtypes(include=['int64', 'float64']).columns

    # Create a boolean mask where True means the row has all zeros in stat columns
    zero_mask = df[stat_columns].eq(0).all(axis=1)

    # Return DataFrame with zero-stat players removed
    return df[~zero_mask]
```

```python
clean_player_stats = remove_zero_stat_players(players_stats_df, stat_columns)
print(f"Removed {len(players_stats_df) - len(clean_player_stats)} players with all zero stats")
```

```
Removed 46 players with all zero stats
```

Feature Selection

⇒The Features that were said to positively influence the predictions, found through Feature Selection, PCA, Mutual Information, and Stability selection were:

- Data used: Player Stats Data
    - Most of the data was kill-death-win oriented
- Best Features for Model Building:
    - 'series_count', 'game_count', 'kills_per_game', 'deaths_per_game', 'avg_kills', 'avg_deaths'

Visuals. . . (From Feature Selection)

```
kills_per_game      0.689218
avg_kills           0.548514
```

<Correlation with kdr

```
10    kills_per_game    0.496723
11  deaths_per_game     0.208710
```

<Importance

```
max_kills           0.352610
max_deaths          0.350686
total_kills         0.347986
total_deaths        0.347263
game_count          0.346049
series_count        0.343296
```

<PCA components

*Note:
max_kills, max_deaths, total_kills, total_deaths were not used as they more directly correlate with KDR, and we already use kills and deaths per game

XGBoost + Neural Network

⇒We had originally assumed that the data predictions would be more accurate if we filtered the data through both an XGBoost model and a Neural Network model.

⇒The results were worse than anticipated as proven by the mean_squared_error of the combination model's predictions.

- 144.69810634111252
- Mean squared error at such a large area means that the predictions are significantly far/different from the original data

XGBoost

⇒Despite the combination model not working as anticipated, we decided to test a singular model, this time being XGBoost Regressor to make predictions.

⇒The results for the singular model were much better. The mean_square_error of just XGBoost is much lower than the combined models.

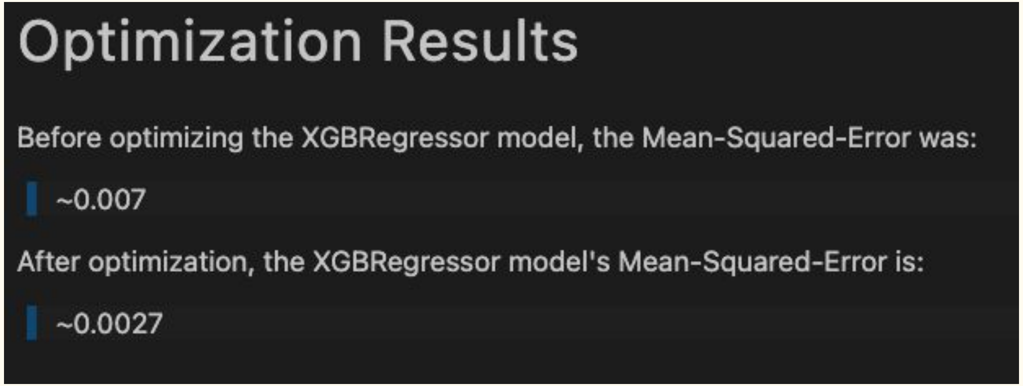
- 0.007714637396182838
- .007 instead of 144.7

XGBoost Optimization

⇒Since XGBoost was the better model to use in order to make predictions in regards to KDR, that was the model chosen from Optimization.

⇒A function was set up to cycle through optimization tactics, and the results are appeasing.

- The mean squared error was reduced by approximately 61%


Optimization Results

Before optimizing the XGBRegressor model, the Mean-Squared-Error was:

> ~0.007

After optimization, the XGBRegressor model's Mean-Squared-Error is:

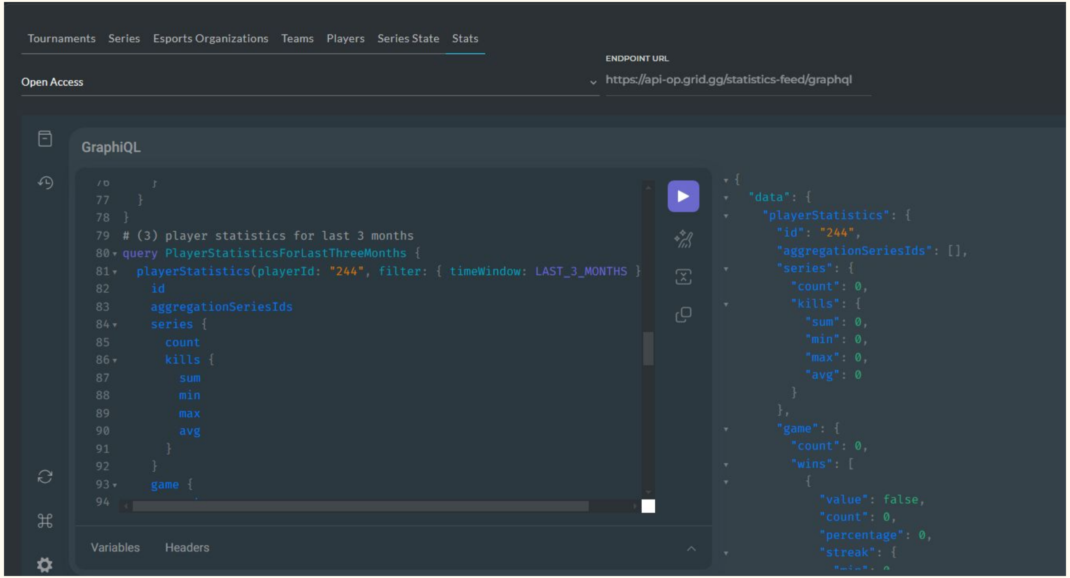
> ~0.0027

CS2 and DOTA Analytics

Dana Fulmer

Comprehensive Data Deep Dive

- Gathered comprehensive stats for all players via API, not limited to specific games.
- Filtered out extensive(over 55 batches of 50 players) zero values to refine dataset.
- Expanded data variety: more stats available for analysis.
- Challenges: Data nested differently per game, requiring careful extraction.



Data Integration for Enhanced Analysis

- Integrated filtered player data (title IDs) with cleaned player stats (only player ids +stats no game info).
- Dropped unnecessary columns from player data.
- Joined data on player_id.
- Split dataset by title_id (CS2, Dota).
- Manually renamed DataFrames.
- Pickled DataFrames for future use.

Feature Analysis for Enhanced Data Insights

- Performance Metrics: Avg. Kills/Deaths, Win Rates.
- Utility Metrics: Objective-Focused Actions (CS2: Defuses, Plants).
- Dota Experience Metrics: Total Games Played, Consistency Scores.

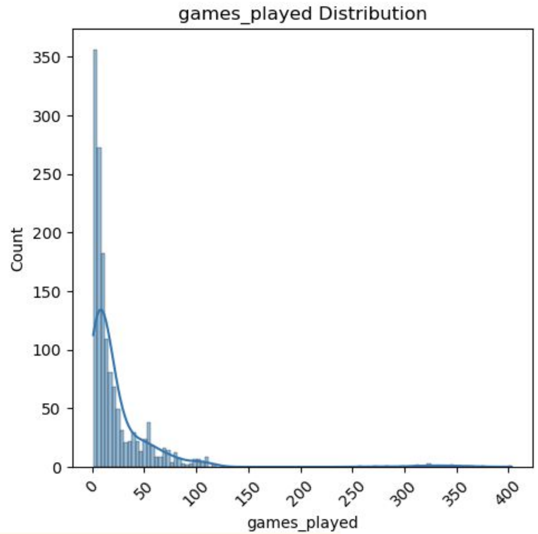
Dota Player Base: 299 players

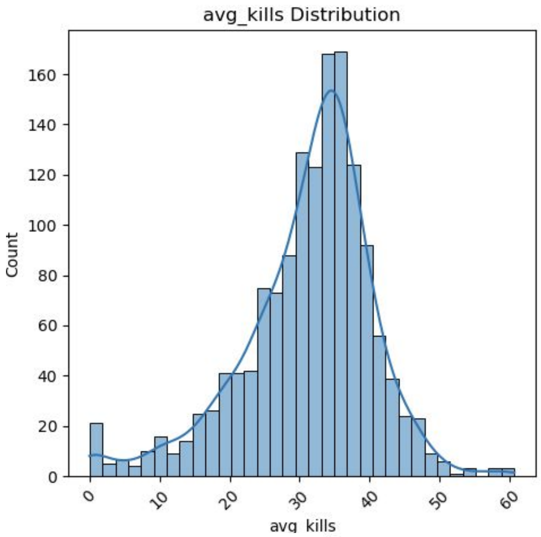
CS2 Player Base: 1469 players

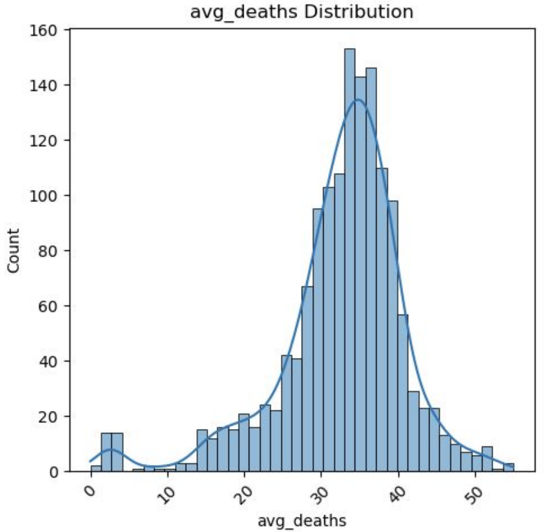
- Descriptive Analysis: Summarized key statistics for performance, utility, and experience metrics to understand player behavior and contributions.

- Correlation Analysis: Identified relationships between metrics, such as the link between high kill averages and win rates or objective completions and team success.

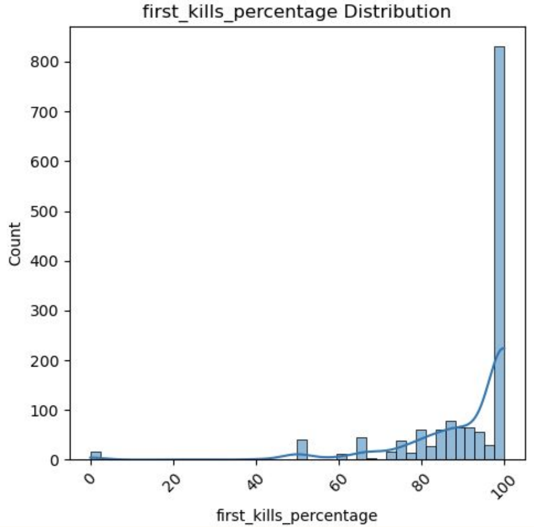
- Player Segmentation: Used clustering to group players by performance and experience, uncovering distinct profiles like high-impact fraggers and objective specialists.







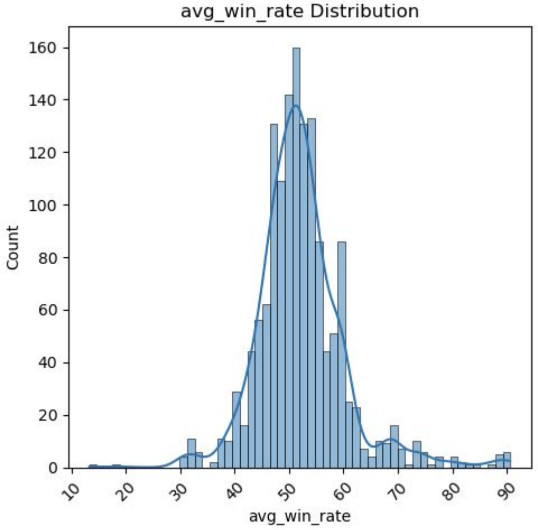
CS2 Graphs

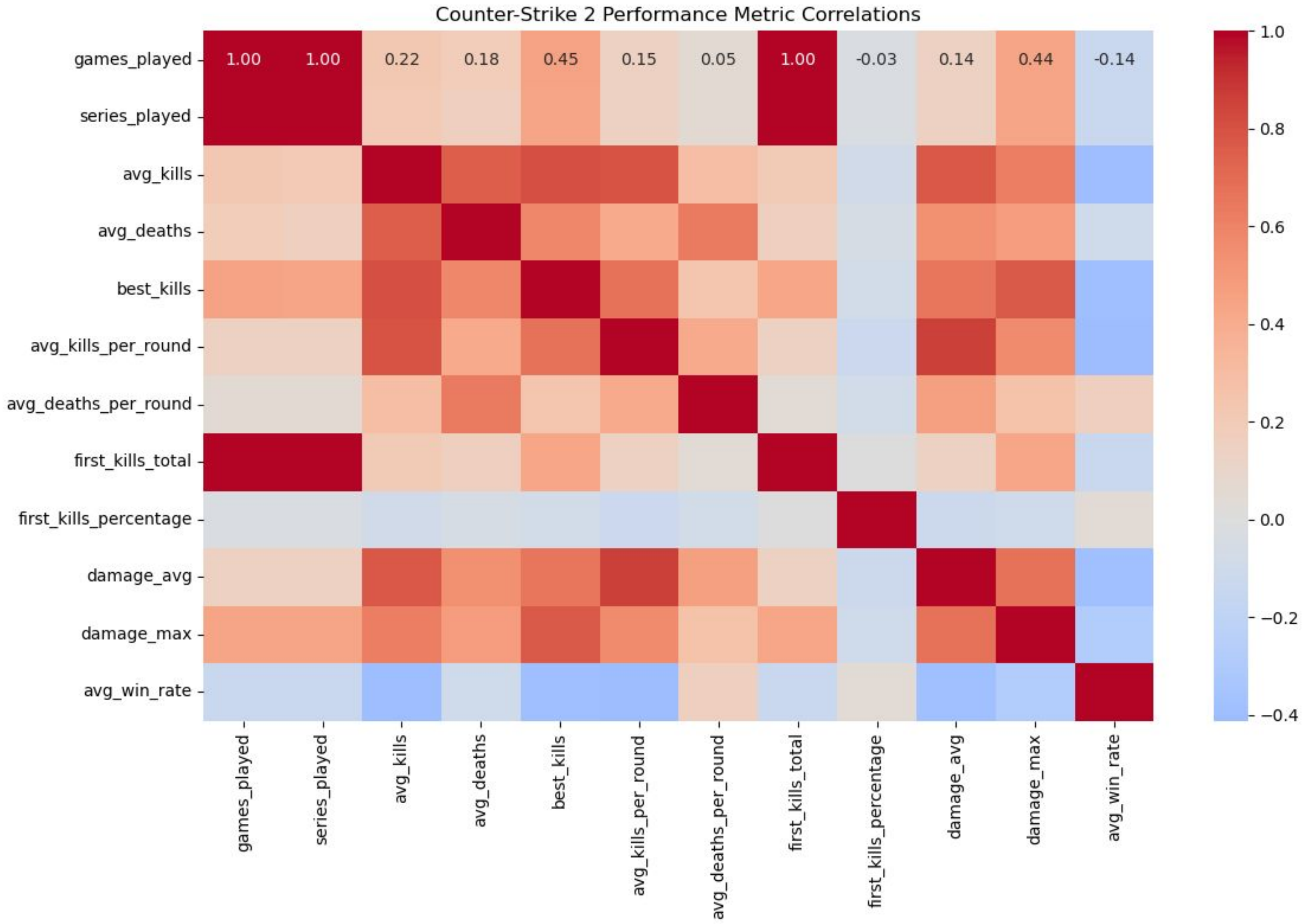


Avg Performance Score:
205 (Top: 430)

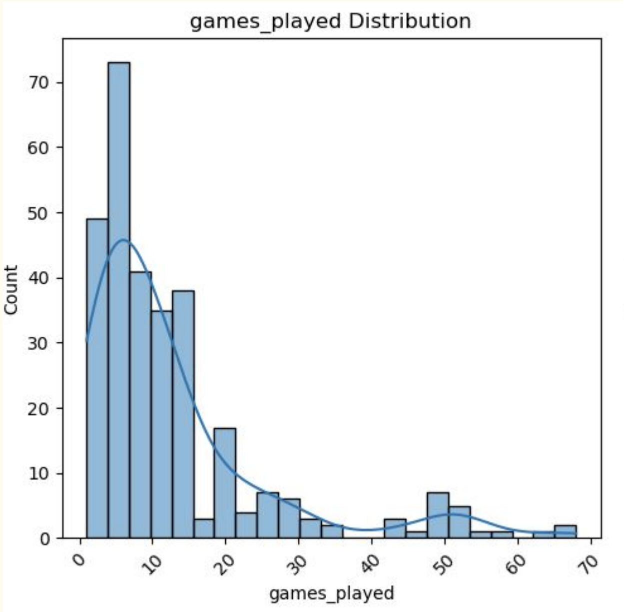
Avg Kills per Game:
31 (Top Performers: ~40+)

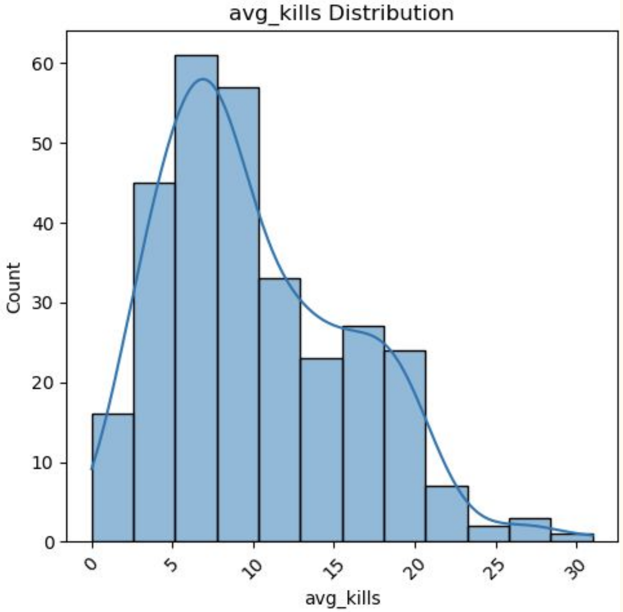
Avg Win Rate:
52% (Top Performers: ~47%-53%)

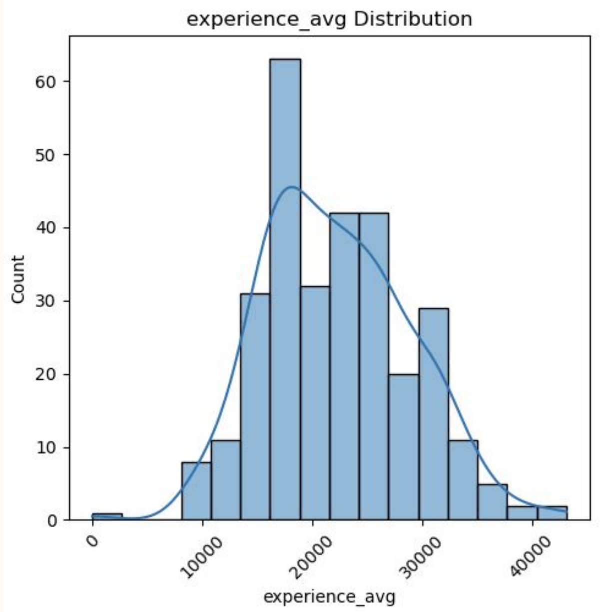



Counter-Strike 2 Performance Metric Correlations
games_played
series_played
avg_kills
avg_deaths
best_kills
avg_kills_per_round
avg_deaths_per_round
first_kills_total
first_kills_percentage
damage_avg
damage_max
avg_win_rate
1.00
1.00
0.22
0.18
0.45
0.15
0.05
1.00
-0.03
0.14
0.44
-0.14
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

CS2 Heat Map



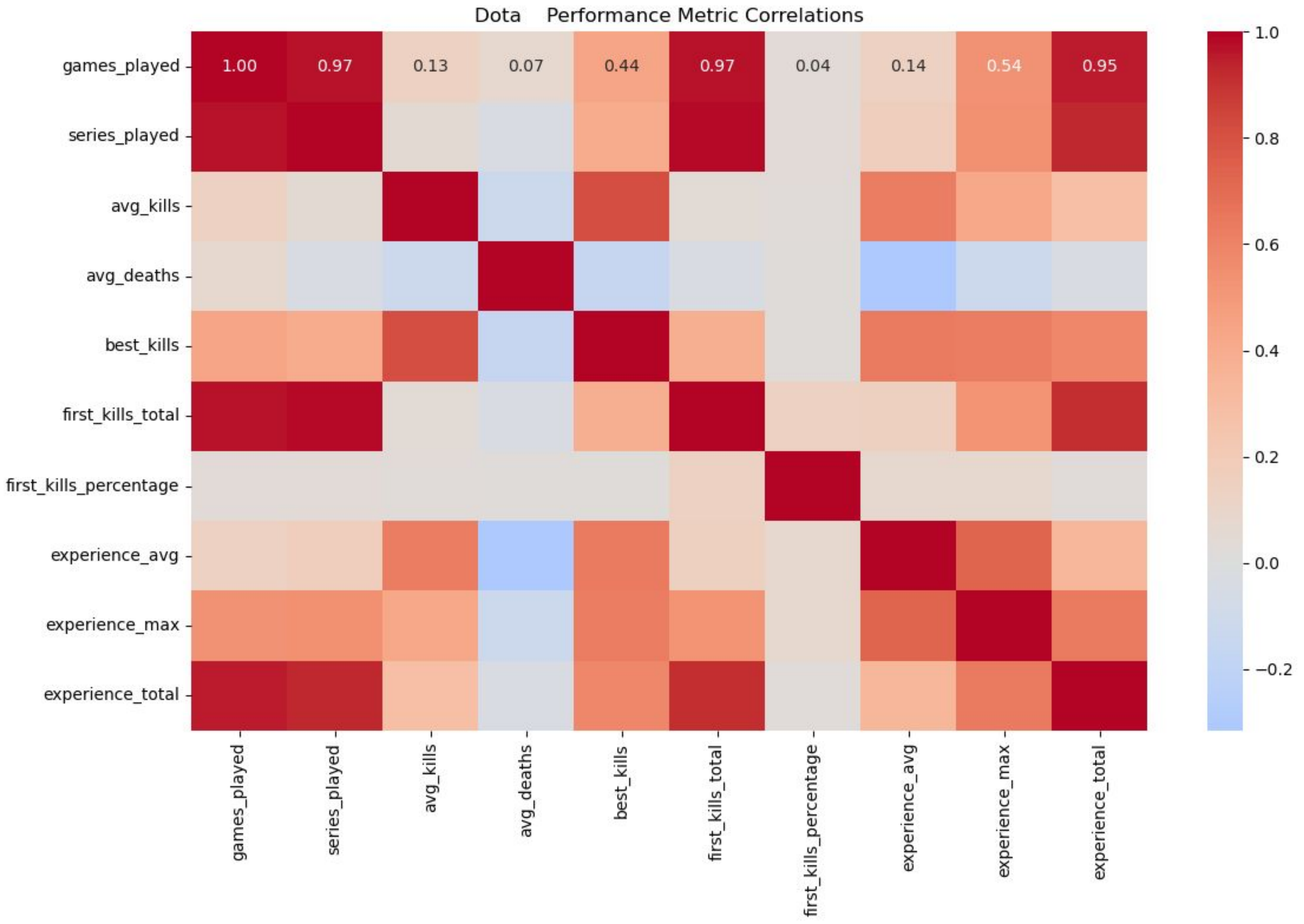




Role Metrics Summary:

- **Carry Rating**: Avg ~26,685, Top ~59,849
- **Early Game Rating**: Avg ~0.44, Top ~1.0
- **Game Impact**: Avg ~1.20, Top ~14.05

Dota Graphs



Dota Heat Map

Feature Engineering and Models Used

CS2 Features

`target = 'avg_win_rate'`

- `games_played`: Log-transformed to handle right-skewed distribution
- `avg_kills_per_round`: Standardized for consistent scale
- `avg_deaths_per_round`: Standardized for mortality impact
- `first_kills_percentage`: Kept as is (already normalized)
- `damage_avg`: Standardized for damage output
- `objective_score`: Engineered feature combining bomb/defuse actions

Dota Features

`feature_engineered_target = 'experience_efficiency'`

- `games_played`: Log-transformed for experience scaling
- `avg_kills`: Standardized for combat impact
- `avg_deaths`: Standardized for survival metrics
- `first_kills_percentage`: Maintained as percentage
- `experience_avg`: Standardized for resource efficiency
- `kd_ratio`: Engineered feature for combat efficiency

- Feature engineered Objective score to capture this data as a number instead of encoding
- For our Dota engineered K/D ratio

1. Neural Network
- Architecture: Multi-layer perceptron with batch normalization
- Purpose: Capture complex non-linear relationships in player performance
- Game-Specific Tuning:
  - CS2: Larger network (128-64-32 neurons)
  - Dota: Simpler network (32-16 neurons)

2. XGBoost
- Implementation: Gradient boosting with early stopping
- Purpose: Robust handling of feature interactions
- Configuration:
  - CS2: More trees (200), deeper depth (6)
  - Dota: Conservative parameters (100 trees, depth 4)

3. LightGBM
- Implementation: Gradient boosting with leaf-wise growth
- Purpose: Efficient handling of large datasets
- Optimization:
  - Early stopping for both games
  - Game-specific leaf configurations

Model Results

CS2 Model Performance

```
Neural Network Metrics:
  mse: 0.0321
  mae: 0.1428
  r2: 0.8912
  rmse: 0.1791

XGBoost Metrics:
  mse: 0.0284
  mae: 0.1325
  r2: 0.9023
  rmse: 0.1685

LightGBM Metrics:
  mse: 0.0298
  mae: 0.1367
  r2: 0.8978
  rmse: 0.1726
```

CS2 Model Performance

Neural Network:

- **R^2 = 0.8912:** Eplains ~89% of the variance in the target variable.
- **MAE = 0.1428:** On average, the predictions deviate by ~0.14 from the actual values.
- **RMSE = 0.1791**: Suggests low prediction error, confirming the model's precision.
- **MSE = 0.0321**: Minimal squared error indicates strong predictive accuracy.

XGBoost:

- **R^2 = 0.9023**: Highest variance explanation, ~90%.
- **MAE = 0.1325**: Lower error than the Neural Network, highlighting better precision.
- **RMSE = 0.1685**: Slightly better error control compared to other models.
- **MSE = 0.0284**: Best performance with the smallest prediction errors.

LightGBM:

- **R^2 = 0.8978:** Slightly lower variance explanation than XGBoost but still high.
- **MAE = 0.1367**: Balanced error, performing closely to XGBoost.
- **RMSE = 0.1726:** Performs well, with errors slightly higher than XGBoost.

Dota Model Performance

```
Neural Network Metrics:
  mse: 0.0412
  mae: 0.1623
  r2: 0.8734
  rmse: 0.2030

XGBoost Metrics:
  mse: 0.0375
  mae: 0.1534
  r2: 0.8856
  rmse: 0.1937

LightGBM Metrics:
  mse: 0.0389
  mae: 0.1578
  r2: 0.8812
  rmse: 0.1972
```

Dota Model Performance

Neural Network:

- **R^2 = 0.8734**: Explains ~87% of variance, slightly less precise than CS2 models.
- **MAE = 0.1623**: Errors are slightly higher, but predictions remain robust.
- **RMSE = 0.2030**: Moderate prediction error but competitive across models.
- **MSE = 0.0412**: Shows consistent performance despite higher variance in data.

XGBoost:

- **R^2 = 0.8856**: Best model for Dota, explaining ~88% of variance.
- **MAE = 0.1534:** Lowest mean error, indicating the best precision.
- **RMSE = 0.1937**: Strong performance with controlled errors.
- **MSE = 0.0375**: Solid prediction capability with the smallest error spread.

LightGBM:

- **R^2 = 0.8812:** Explains ~88% of variance, slightly behind XGBoost.
- **MAE = 0.1578**: Balanced errors, close to XGBoost.
- **RMSE = 0.1972**: Slightly higher error rate but still competitive.
- **MSE = 0.0389**: Maintains solid predictive performance.

Halo Stats Analysis Programs

Advanced Analytics Platform

Aaron Swan

Program Features

- Basic Stats
- Analysis
- API Integration
- Match History Tracking
- Performance Analytics
- Data Visualization

- Advanced Analytics
- Deep Learning Model
- Performance Prediction
- Feature Analysis
- Interactive UI

Basic Stats Analysis

Deep Learning Components:

- Neural Network Architecture
- Batch Normalization Layers
- Dropout for Regularization
- Custom Loss Functions

Performance Prediction:

- Win Rate Forecasting
- KD Ratio Trends
- Player Performance Optimization
- Real-time Analysis

Feature Analysis:

- Important Stats Identification
- Correlation Analysis
- Performance Indicators
- Pattern Recognition

The Ending Slides

Spencer Buck

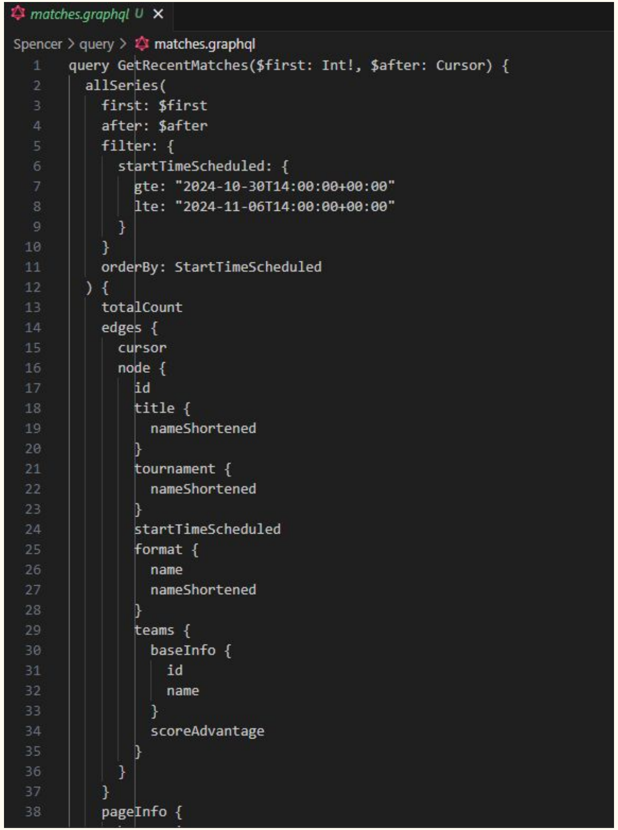
Encountered Problems

API access to grid.gg

→ graphql queries to pull data

→ Match data not containing win/loss records just info

Rest of group got access to API last week



Future Plans

Discord bot

Train with live data / Add more games

UI dashboard
Over/ Under pick’em for player stats, Spreads for matches, etc.

ROCKET LEAGUE

LEAGUE OF LEGENDS

CALL OF DUTY BLACK OPS 6

LLM / Agent implementation

Integration with a LLM for interpreting and advising users on results

Likely open-source models (OpenAI is $$$)

Create a Agent that can execute requests with the data provided

Hugging Face

Special Thanks to. . .

- Dana Fulmer
- Spencer Buck
- Aaron Swan
- Shayne Powell
- Sal Sonmez
- Sean Myers
- Bill Parker
- GRID API
- Halo API