```python
import pandas as pd

# Read the raw players data
df = pd.read_csv('players.csv')

# List of stats to average
stats = [
    'kills', 'assists', 'deaths', 'hs', 'flash_assists',
    'kast', 'kddiff', 'adr', 'fkdiff', 'rating'
]

# Reshape per-map stats into a long format
records = []
for i in [1, 2, 3]:
    temp = df[['match_id', 'player_id', 'player_name',
f'map_{i}']].rename(
        columns={f'map_{i}': 'map_name'}
    ).copy()
    for stat in stats:
        temp[stat] = df[f'm{i}_{stat}']
    temp = temp[temp['map_name'].notna()]
    records.append(temp)

long_df = pd.concat(records, ignore_index=True)

# --- NEW: collapse BO3 into one row per player per match ---
per_match = (
    long_df
    .groupby(['match_id', 'player_id', 'player_name'])[stats]
    .mean()
    .reset_index()
)

# 1) General averages per player across matches (each match counts
once)
general_avg = (
    per_match
    .groupby(['player_id', 'player_name'])[stats]
    .mean()
    .reset_index()
)
general_avg.to_csv('player_general_averages.csv', index=False)

# 2) Averages per player per specific map (unchanged: still per map-
appearance)
map_avg = (
    long_df
    .groupby(['player_id', 'player_name', 'map_name'])[stats]
    .mean()
    .reset_index()
```

```
)
map_avg.to_csv('player_map_averages.csv', index=False)

print("Saved player_general_averages.csv (per match) and
player_map_averages.csv (per map) to the notebook directory")
```

```
Saved player_general_averages.csv (per match) and
player_map_averages.csv (per map) to the notebook directory
```

Exploratory Data Analysis of Player Averages

In this section, we'll load our two new datasets:

- **player_general_averages.csv** – each player's average stats across all maps.
- **player_map_averages.csv** – each player's average stats on each map.

Our goals:

1. Understand the overall distribution of core metrics (kills, ADR, rating, etc.).
2. Spot map-specific variations (e.g. which players excel on Dust2 vs. Mirage).
3. Look for correlations between metrics that might inform feature selection later.

```
import pandas as pd

# Load the processed CSVs from current directory
general_avg = pd.read_csv('player_general_averages.csv')
map_avg     = pd.read_csv('player_map_averages.csv')

# Show first few rows of each
display(general_avg.head())
display(map_avg.head())
```

```
   player_id player_name      kills   assists     deaths        hs  \
0          2      RobbaN   7.000000  0.000000  12.500000  3.500000
1          7       Friis  17.459896  2.973438  16.669792  4.179688
2         11     Vertigo  15.833333  3.250000  18.750000  7.083333
3         13      RashiE  19.090909  4.681818  15.636364  7.196970
4         15      m1kkis  18.012195  2.861789  18.174797  5.918699

   flash_assists       kast    kddiff        adr    fkdiff    rating
0            NaN  74.950000 -5.500000  24.450000 -1.000000  0.775000
1       1.102837  68.746619  0.790104  68.723665  0.520312  1.054089
2       0.333333  62.950000 -2.916667  67.308333 -2.250000  0.900000
3       1.018519  77.163636  3.454545  76.177273  1.181818  1.201970
4       0.250000  66.963008 -0.162602  71.127236  0.792683  1.021016

   player_id player_name     map_name      kills    assists     deaths
\
```

```
0          2       RobbaN        Dust2   7.000000  0.000000  12.500000
1          7        Friis        Cache  14.973684  2.631579  17.000000
2          7        Friis  Cobblestone  16.863158  3.000000  16.115789
3          7        Friis      Default  18.000000  2.000000  21.000000
4          7        Friis        Dust2  15.904762  3.380952  18.523810

         hs  flash_assists       kast     kddiff        adr    fkdiff
rating
0  3.500000            NaN  74.950000 -5.500000  24.450000 -1.000000
0.775000
1  4.026316       1.333333  66.984848 -2.026316  63.993939 -0.342105
0.930263
2  4.252632       1.761905  70.897590  0.747368  69.275904  0.421053
1.057789
3  4.000000            NaN  53.600000 -3.000000  69.200000 -5.000000
0.760000
4  4.142857       0.000000  61.484211 -2.619048  66.026316  0.666667
0.917619
```

1. Summary Statistics

Let's compute basic summary statistics (mean, std, min/max, quartiles) for our core metrics in the **general_avg** dataset.
This will help us see range, skew, and spot any outliers we might need to handle.

```
# Summary stats for general averages
stats =
['kills','assists','deaths','hs','flash_assists','kast','kddiff','adr'
,'fkdiff','rating']
general_summary = general_avg[stats].describe().T
general_summary
```

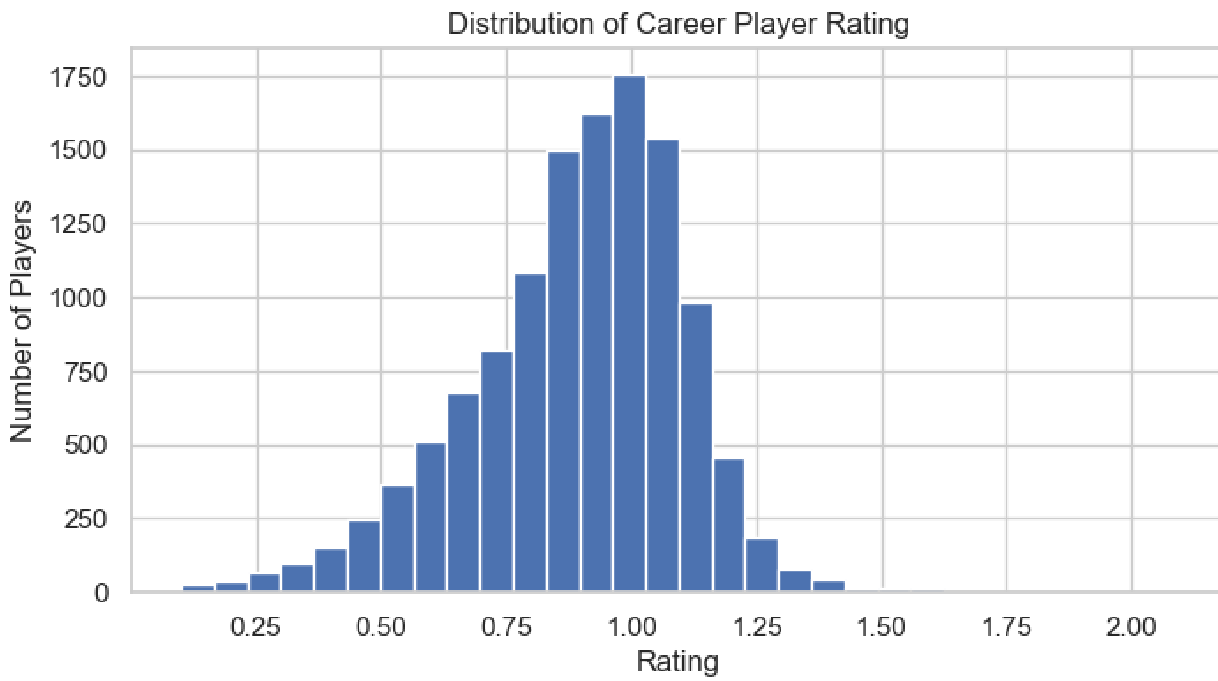
```
                 count       mean       std  min        25%
50%  \
kills          12294.0  15.113268  4.098934  0.0  12.750000
15.700000
assists        12294.0   3.515619  1.314289  0.0   2.750000
3.600000
deaths         12294.0  18.411527  2.053487  1.0  17.293752
18.297502
hs             12294.0   6.841165  2.329430  0.0   5.333333
6.952894
flash_assists  10292.0   0.634768  0.604082  0.0   0.000000
0.513889
```

```
kast             11703.0  63.363450   8.733963  16.7   59.302778
65.322695
kddiff           12294.0  -3.298260   4.161675 -18.0   -6.000000  -
2.916667
adr              11701.0  68.154684  11.832218   2.0   61.800000
69.581250
fkdiff           12294.0  -0.545273   1.270790  -8.0   -1.123188  -
0.433333
rating           12294.0   0.890760   0.213180   0.1    0.767088
0.920000

                       75%     max
kills            17.750000   41.00
assists           4.293565   16.00
deaths           19.416667   39.00
hs                8.344673   20.00
flash_assists     1.000000    8.00
kast             69.146614  100.00
kddiff           -0.405729   17.00
adr              75.652211  130.50
fkdiff            0.166667    6.00
rating            1.038267    2.09
```

2. Distribution of Career Rating

The HLTV-style rating is one of our most important features.
Let’s visualize its distribution to check for normality or heavy tails, which might affect modeling later.

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(8,4))
plt.hist(general_avg['rating'], bins=30)
plt.title("Distribution of Career Player Rating")
plt.xlabel("Rating")
plt.ylabel("Number of Players")
plt.show()
```



3. Correlation Matrix

Next, we'll look at pairwise correlations between our core metrics.
High multicollinearity (e.g. between kills and ADR) may suggest dropping or combining features.

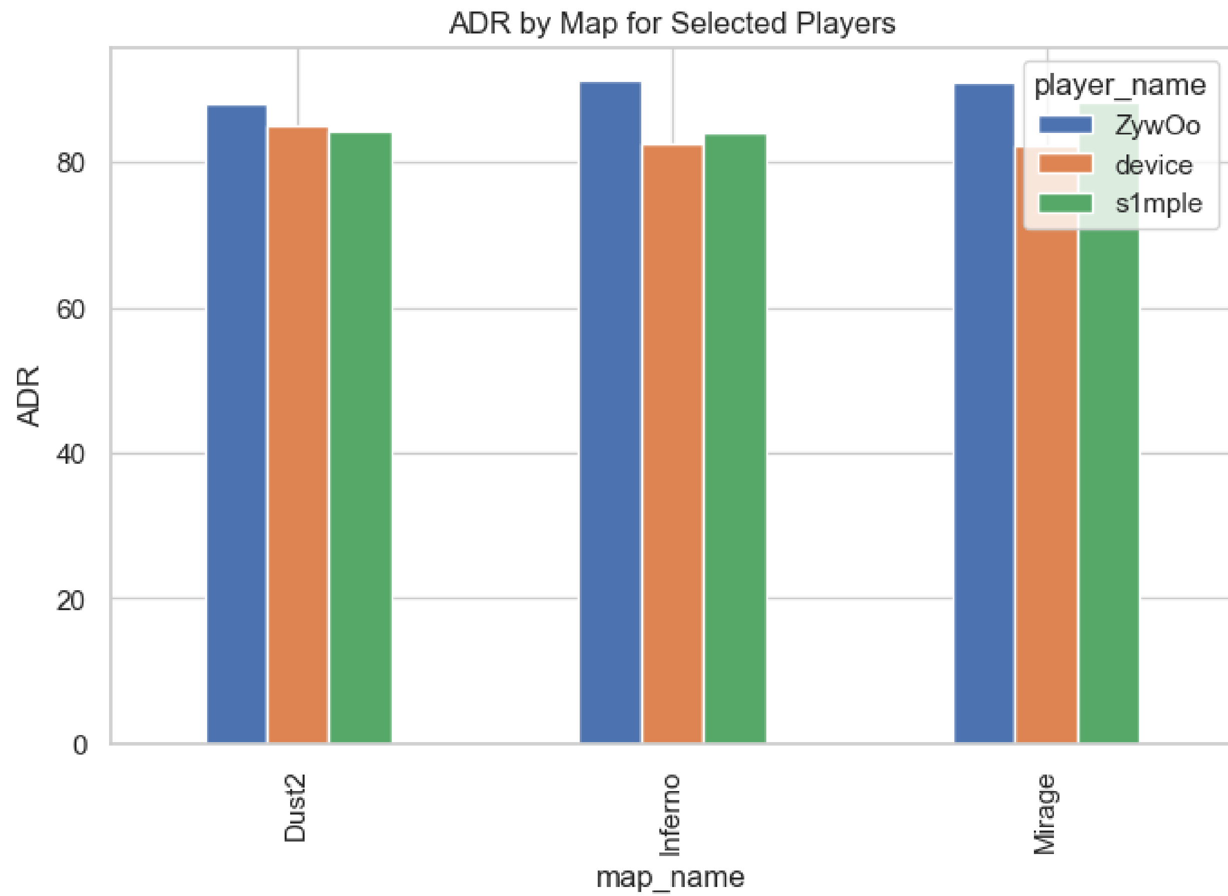
```
import seaborn as sns

corr = general_avg[stats].corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="viridis")
plt.title("Correlation Matrix of Career Metrics")
plt.show()
```

4. Map-Specific Performance

We might expect top fraggers to perform differently depending on the map.
Let's pick two or three players and compare their ADR on Dust2, Mirage, and Inferno as an example.

```
# Example: compare ADR of three star players on three popular maps
players = ["s1mple", "device", "ZywOo"]  # replace with actual
player_name values in your data
subset = map_avg[map_avg['player_name'].isin(players)]
subset = subset[subset['map_name'].isin(["Dust2","Mirage","Inferno"])]
subset_pivot = subset.pivot(index='map_name', columns='player_name',
values='adr')

subset_pivot.plot(kind='bar', figsize=(8,5))
plt.title("ADR by Map for Selected Players")
```

```
plt.ylabel("ADR")
plt.show()
```



Constructing `team_match_stats.csv`

We build a table with one row per match–team, including:

- Match metadata (date, event, teams, sides, winners)
- Scoreline & map win flag
- Aggregated player stats (kills, deaths, assists, ADR, KAST, rating)
- Economy stats (start/end money, buy-type win rates, bomb/defuse rates)
- Veto/picks info
- Rest days
- Roster change flags & counts

```
import pandas as pd

# 1. Load data
players = pd.read_csv('players.csv')
results = pd.read_csv('results.csv')

# 2. Team-level player stats (general averages across all maps)
team_player_stats = players.groupby('team').agg(
    avg_kills         = ('kills',        'mean'),
    avg_deaths        = ('deaths',       'mean'),
    avg_assists       = ('assists',      'mean'),
    avg_hs            = ('hs',           'mean'),
    avg_flash_assists = ('flash_assists','mean'),
    avg_kast          = ('kast',         'mean'),
    avg_kddiff        = ('kddiff',       'mean'),
    avg_adr           = ('adr',          'mean'),
    avg_fkdiff        = ('fkdiff',       'mean'),
    avg_rating        = ('rating',       'mean'),
    matches_played    = ('match_id',     'nunique')
).reset_index()

# 3. Team-level results: expand each match to two rows and compute win
flags
res =
results[['match_id','team_1','team_2','map_wins_1','map_wins_2']].copy
()

team1 = res.rename(columns={'team_1':'team', 'map_wins_1':'map_wins'})
team1['op_map_wins']   = res['map_wins_2']
team1['is_match_win']  = team1['map_wins'] > team1['op_map_wins']

team2 = res.rename(columns={'team_2':'team', 'map_wins_2':'map_wins'})
team2['op_map_wins']   = res['map_wins_1']
team2['is_match_win']  = team2['map_wins'] > team2['op_map_wins']

team_results = pd.concat([
    team1[['team','match_id','map_wins','is_match_win']],
    team2[['team','match_id','map_wins','is_match_win']]
], ignore_index=True)

team_result_stats = team_results.groupby('team').agg(
    avg_map_wins   = ('map_wins',    'mean'),
    match_win_rate = ('is_match_win','mean')
).reset_index()

# 4. Compute game_frequency (number of matches played per team)
home_games =
results[['match_id','team_1']].rename(columns={'team_1':'team'})
away_games =
results[['match_id','team_2']].rename(columns={'team_2':'team'})
```

```python
game_freq = (
    pd.concat([home_games, away_games], ignore_index=True)
      .groupby('team')['match_id']
      .nunique()
      .reset_index(name='game_frequency')
)

# 5. Merge and export
team_general_averages = (
    team_player_stats
    .merge(team_result_stats, on='team', how='left')
    .merge(game_freq,          on='team', how='left')
)

team_general_averages.to_csv('team_general_averages.csv', index=False)

# 6. Quick sanity check
print(team_general_averages[['team','match_win_rate','game_frequency']
].head())
```

```
          team  match_win_rate  game_frequency
0  !nsurgents             NaN             NaN
1    #FREEIBP             NaN             NaN
2       #SKAM             NaN             NaN
3   $HAMELE$$             NaN             NaN
4       -0_o-             NaN             NaN
```

```python
import pandas as pd

# 1. Load CSVs
players = pd.read_csv('players.csv')
results = pd.read_csv('results.csv')

# 2. Unpivot per-map player stats into long format
map_dfs = []
for i in [1, 2, 3]:
    df = players[[
        'match_id', 'team', f'map_{i}',
        f'm{i}_kills', f'm{i}_deaths', f'm{i}_assists',
        f'm{i}_hs', f'm{i}_flash_assists', f'm{i}_kast',
        f'm{i}_kddiff', f'm{i}_adr', f'm{i}_fkdiff', f'm{i}_rating'
    ]].copy()

    # rename to common columns
    df = df.rename(columns={
        f'map_{i}':        'map',
        f'm{i}_kills':     'kills',
        f'm{i}_deaths':    'deaths',
        f'm{i}_assists':   'assists',
```

```
        f'm{i}_hs':         'hs',
        f'm{i}_flash_assists': 'flash_assists',
        f'm{i}_kast':       'kast',
        f'm{i}_kddiff':     'kddiff',
        f'm{i}_adr':        'adr',
        f'm{i}_fkdiff':     'fkdiff',
        f'm{i}_rating':     'rating',
    })

    map_dfs.append(df)

player_map_long = pd.concat(map_dfs, ignore_index=True)
# drop rows where no map was played (e.g. BO1 matches only have map_1)
player_map_long = player_map_long.dropna(subset=['map'])

# 3. Aggregate to get team×map player-level averages
team_map_player_stats = (
    player_map_long
    .groupby(['team', 'map'])
    .agg(
        avg_kills        = ('kills',        'mean'),
        avg_deaths       = ('deaths',       'mean'),
        avg_assists      = ('assists',      'mean'),
        avg_hs           = ('hs',           'mean'),
        avg_flash_assists= ('flash_assists','mean'),
        avg_kast         = ('kast',         'mean'),
        avg_kddiff       = ('kddiff',       'mean'),
        avg_adr          = ('adr',          'mean'),
        avg_fkdiff       = ('fkdiff',       'mean'),
        avg_rating       = ('rating',       'mean'),
        games_played     = ('match_id',     'nunique')
    )
    .reset_index()
)

# 4. Prepare results for team×map stats
# ensure numeric
results['result_1'] = pd.to_numeric(results['result_1'],
errors='coerce')
results['result_2'] = pd.to_numeric(results['result_2'],
errors='coerce')

home = (
    results
    .rename(columns={
        'team_1': 'team',
        'result_1': 'rounds_won',
        'result_2': 'rounds_lost',
        '_map':      'map'
    })
```

```
    [['match_id','team','map','rounds_won','rounds_lost']]
)

away = (
    results
    .rename(columns={
        'team_2': 'team',
        'result_2': 'rounds_won',
        'result_1': 'rounds_lost',
        '_map':      'map'
    })
    [['match_id','team','map','rounds_won','rounds_lost']]
)

flat_res = pd.concat([home, away], ignore_index=True)
flat_res['map_win']    = flat_res['rounds_won'] >
flat_res['rounds_lost']
flat_res['round_diff'] = flat_res['rounds_won'] -
flat_res['rounds_lost']

# 5. Aggregate to get team×map result-level stats
team_map_result_stats = (
    flat_res
    .groupby(['team','map'])
    .agg(
        avg_rounds_won  = ('rounds_won', 'mean'),
        avg_rounds_lost = ('rounds_lost','mean'),
        avg_round_diff  = ('round_diff', 'mean'),
        map_win_rate    = ('map_win',    'mean')
    )
    .reset_index()
)

# 6. Merge player- and result-stats into final team_map_averages
team_map_averages = pd.merge(
    team_map_player_stats,
    team_map_result_stats,
    on=['team','map'],
    how='inner'
)

# 7. Save and preview
team_map_averages.to_csv('team_map_averages.csv', index=False)
team_map_averages.head()
```

```
           team      map  avg_kills  avg_deaths  avg_assists    avg_hs  \
0  100 Thieves    Dust2  15.200000   17.400000     5.400000  7.750000

1  100 Thieves  Inferno  16.700000   12.250000     4.775000  9.150000
```

```
2  100 Thieves   Mirage  14.371429   16.285714     3.714286  7.000000

3  100 Thieves     Nuke  17.700000   18.700000     3.866667  8.766667

4  100 Thieves    Train  13.500000   17.900000     3.150000  6.550000


   avg_flash_assists   avg_kast  avg_kddiff    avg_adr  avg_fkdiff  \
0           1.800000  66.980000   -2.200000  67.305000   -0.750000
1           1.566667  81.162500    4.450000  78.615000    0.825000
2           1.300000  66.362857   -1.914286  69.894286   -0.657143
3           0.560000  68.896667   -1.000000  73.470000   -0.166667
4           0.933333  61.400000   -4.400000  63.155000    0.150000

   avg_rating  games_played  avg_rounds_won  avg_rounds_lost
avg_round_diff  \
0    0.949000             4       12.500000        12.666667        -
0.166667
1    1.303250             8       14.545455        10.636364
3.909091
2    0.990286             7       11.000000        13.250000        -
2.250000
3    0.994667             6       13.000000        14.250000        -
1.250000
4    0.858500             4        9.000000        15.250000        -
6.250000

   map_win_rate
0      0.500000
1      0.636364
2      0.375000
3      0.500000
4      0.250000
```

Generating Matches.csv

```
import pandas as pd

# only load the 8 columns we need
use_cols = [
    "match_id",
    "event_id",
    "date",
    "team_1",
    "team_2",
    "_map",
    "starting_ct",
```

```
    "map_winner"
]

matches = (
    pd.read_csv("results.csv", usecols=use_cols)
      .rename(columns={"_map":"map"})
)

# peek at the first few rows
matches.head()
```

```
         date                team_1          team_2      map
map_winner  \
0  2020-03-18               Recon 5         TeamOne    Dust2
2
1  2020-03-18               Recon 5         TeamOne  Inferno
2
2  2020-03-18  New England Whalers        Station7  Inferno
2
3  2020-03-18              Rugratz  Bad News Bears  Inferno
2
4  2020-03-18              Rugratz  Bad News Bears  Vertigo
2

   starting_ct  event_id  match_id
0            2      5151   2340454
1            2      5151   2340454
2            1      5243   2340461
3            2      5151   2340453
4            2      5151   2340453
```

```
import pandas as pd

# — reload your base matches —
use_cols = [
    "match_id",
    "event_id",
    "date",
    "team_1",
    "team_2",
    "_map",
    "starting_ct",
    "map_winner"
]
matches = (
    pd.read_csv("results.csv", usecols=use_cols)
      .rename(columns={"_map":"map"})
)

# 1) build long form of each side per match
```

```
counts = matches.melt(
    id_vars=["match_id", "map_winner"],
    value_vars=["team_1", "team_2"],
    var_name="side",
    value_name="team"
)

# 2) identify which side won (map_winner == 1 → team_1, == 2 → team_2)
counts["winner_side"] = counts["map_winner"].map({1: "team_1", 2:
"team_2"})

# 3) flag win when side equals winner_side
counts["win"] = (counts["side"] == counts["winner_side"]).astype(int)

# 4) aggregate per team to compute win_rate
team_stats = (
    counts.groupby("team")
          .agg(games_played = ("match_id","count"),
               wins          = ("win","sum"))
          .assign(win_rate = lambda df: df["wins"] /
df["games_played"])
          .reset_index()
)

# 5) map win_rate back into matches
wr_map = team_stats.set_index("team")["win_rate"]
matches["team_1_wr"] = matches["team_1"].map(wr_map)
matches["team_2_wr"] = matches["team_2"].map(wr_map)

# 6) save updated matches.csv
matches.to_csv("matches.csv", index=False)

matches.head()
```

```
         date               team_1          team_2      map  
map_winner  \
0  2020-03-18              Recon 5         TeamOne    Dust2
2
1  2020-03-18              Recon 5         TeamOne  Inferno
2
2  2020-03-18  New England Whalers        Station7  Inferno
2
3  2020-03-18              Rugratz  Bad News Bears  Inferno
2
4  2020-03-18              Rugratz  Bad News Bears  Vertigo
2

   starting_ct  event_id  match_id  team_1_wr  team_2_wr
0            2      5151   2340454   0.595745   0.566914
1            2      5151   2340454   0.595745   0.566914
```

```
2              1      5243   2340461   0.288889   0.416667
3              2      5151   2340453   0.520000   0.610169
4              2      5151   2340453   0.520000   0.610169
```

```
import pandas as pd
import numpy as np

# 1) load the latest matches.csv
matches = pd.read_csv("matches.csv")

# 2) strip whitespace from team names (in-place)
matches["team_1"] = matches["team_1"].astype(str).str.strip()
matches["team_2"] = matches["team_2"].astype(str).str.strip()

# 3) build two DataFrames: one for each side
df1 = pd.DataFrame({
    "team": matches["team_1"],
    "win":  (matches["map_winner"] == 1).astype(int)
})
df2 = pd.DataFrame({
    "team": matches["team_2"],
    "win":  (matches["map_winner"] == 2).astype(int)
})

# 4) concatenate and aggregate per team
team_stats = (
    pd.concat([df1, df2])
      .groupby("team")
      .agg(
          games_played=("win", "size"),
          wins=("win", "sum")
      )
      .assign(win_rate=lambda df: df["wins"] / df["games_played"])
)

# 5) blank win_rate where games_played < 10
team_stats.loc[team_stats["games_played"] < 10, "win_rate"] = np.nan

# 6) map back into matches (overwrite / create)
win_rate_map = team_stats["win_rate"]
matches["team_1_wr"] = matches["team_1"].map(win_rate_map)
matches["team_2_wr"] = matches["team_2"].map(win_rate_map)

# 7) save
matches.to_csv("matches.csv", index=False)

# quick confirmation
print(team_stats.head())
print("\nRecon 5 win-rate:", win_rate_map.get("Recon 5"))
print(matches.head())
```

```
             games_played  wins  win_rate
team
100 Thieves            42    21  0.500000
100pinggods             1     0       NaN
1337                   41    17  0.414634
1337HUANIA             38    18  0.473684
13th Hour               2     0       NaN

Recon 5 win-rate: 0.5957446808510638
         date                team_1          team_2      map
map_winner  \
0  2020-03-18              Recon 5         TeamOne    Dust2
2
1  2020-03-18              Recon 5         TeamOne  Inferno
2
2  2020-03-18  New England Whalers        Station7  Inferno
2
3  2020-03-18              Rugratz  Bad News Bears  Inferno
2
4  2020-03-18              Rugratz  Bad News Bears  Vertigo
2

   starting_ct  event_id  match_id  team_1_wr  team_2_wr
0            2      5151   2340454   0.595745   0.566914
1            2      5151   2340454   0.595745   0.566914
2            1      5243   2340461   0.288889   0.416667
3            2      5151   2340453   0.520000   0.610169
4            2      5151   2340453   0.520000   0.610169
```

```python
import pandas as pd
import numpy as np

# 1) load the current matches file (already has win-rates)
matches = pd.read_csv("matches.csv")

# 2) pull only what we need from players.csv
player_info = pd.read_csv(
    "players.csv",
    usecols=["match_id", "team", "player_name"]
)

# 3) merge so we know which players belong to team_1 vs team_2 in each
match
p = matches[["match_id", "team_1", "team_2"]].merge(
    player_info, on="match_id", how="left"
)

# 4) label which side each player is on
p["side"] = np.where(p["team"] == p["team_1"], "team_1", "team_2")
```

```
# 5) within each match×side, give players a 1-to-N order (appearance
order)
p["player_num"] = p.groupby(["match_id", "side"]).cumcount() + 1

# 6) keep only the first 5 players on each side
p = p[p["player_num"] <= 5]

# 7) pivot so we get columns: team_1_player_1 … team_1_player_5,
team_2_player_1 … team_2_player_5
player_cols = (
    p.pivot(index="match_id", columns=["side", "player_num"],
values="player_name")
      .rename_axis([None, None], axis=1)      # drop axis names
)
player_cols.columns = [f"{side}_player_{num}" for side, num in
player_cols.columns]
player_cols = player_cols.reset_index()

# 8) merge those columns into matches
matches = matches.merge(player_cols, on="match_id", how="left")

# 9) save
matches.to_csv("matches.csv", index=False)

# quick peek
matches.head()
```

```
         date                team_1          team_2      map
map_winner  \
0  2020-03-18               Recon 5         TeamOne    Dust2
2
1  2020-03-18               Recon 5         TeamOne  Inferno
2
2  2020-03-18  New England Whalers        Station7  Inferno
2
3  2020-03-18               Rugratz  Bad News Bears  Inferno
2
4  2020-03-18               Rugratz  Bad News Bears  Vertigo
2

   starting_ct  event_id  match_id  team_1_wr  team_2_wr
team_2_player_1  \
0            2      5151   2340454   0.595745   0.566914
NaN
1            2      5151   2340454   0.595745   0.566914
NaN
2            1      5243   2340461   0.288889   0.416667
NaN
3            2      5151   2340453   0.520000   0.610169
NaN
```

```
4             2      5151   2340453   0.520000   0.610169
NaN

  team_2_player_2 team_2_player_3 team_1_player_1 team_2_player_4  \
0             NaN             NaN             NaN             NaN
1             NaN             NaN             NaN             NaN
2             NaN             NaN             NaN             NaN
3             NaN             NaN             NaN             NaN
4             NaN             NaN             NaN             NaN

  team_1_player_2 team_1_player_3 team_2_player_5 team_1_player_4  \
0             NaN             NaN             NaN             NaN
1             NaN             NaN             NaN             NaN
2             NaN             NaN             NaN             NaN
3             NaN             NaN             NaN             NaN
4             NaN             NaN             NaN             NaN

  team_1_player_5
0             NaN
1             NaN
2             NaN
3             NaN
4             NaN
```

```
import pandas as pd
import numpy as np

# 1) load the file that now contains the 10 player-name columns
matches = pd.read_csv("matches.csv")

# 2) identify the player columns (anything that starts with team_?
_player_)
player_cols = [c for c in matches.columns if "_player_" in c]

# 3) treat empty strings as missing
matches[player_cols] = matches[player_cols].replace("", np.nan)

# 4) keep only rows that have *all* ten player names present
matches = matches.dropna(subset=player_cols, how="any")

# 5) save the filtered file
matches.to_csv("matches.csv", index=False)

# quick sanity-check (optional)
print(f"Rows remaining after dropping incomplete rosters:
{len(matches)}")
```

```
Rows remaining after dropping incomplete rosters: 44524
```

```
import pandas as pd
```

```python
# 1) reload current matches
matches = pd.read_csv("matches.csv")

# 2) grab round-1 and round-16 winners (1 = team_1 won, 2 = team_2
won)
econ = pd.read_csv("economy.csv",
usecols=["match_id","1_winner","16_winner"])
econ = econ.astype({"1_winner":"Int64","16_winner":"Int64"})   # keep
as integers 1/2

# 3) merge and name the columns clearly
matches = matches.merge(
    econ.rename(columns={
        "1_winner":  "ct_pistol_side",   # CT-start pistol is round 1
        "16_winner": "t_pistol_side"     # T-start pistol is round 16
    }),
    on="match_id",
    how="left"
)

# 4) save
matches.to_csv("matches.csv", index=False)

# quick peek
matches[["ct_pistol_side","t_pistol_side"]].head()
```

```
   ct_pistol_side  t_pistol_side
0               2              1
1               2              2
2               2              1
3               2              2
4               2              2
```

```python
import pandas as pd
import numpy as np

# 1) load data
matches  = pd.read_csv("matches.csv")
players  = pd.read_csv("players.csv",
usecols=["match_id","team","player_name"])

# 2) merge winner info & side label
players = players.merge(
    matches[["match_id","team_1","team_2","map_winner"]],
    on="match_id", how="left"
)
players["side"] = np.where(players["team"] == players["team_1"], 1, 2)
players["win_flag"] = (players["side"] ==
players["map_winner"]).astype(int)
```

```python
# 3) **deduplicate to one entry per player per match**
players = (
    players
      .drop_duplicates(subset=["player_name","match_id"])   # keep
first occurrence
)

# 4) career stats
player_stats = (
    players.groupby("player_name")
           .agg(
               games_played=("match_id","nunique"),
               wins=("win_flag","sum")
           )
           .assign(player_wr=lambda d: d["wins"] / d["games_played"])
)
player_stats.loc[player_stats["games_played"] < 10, "player_wr"] =
np.nan

# 5) attach and average per side
players = players.merge(player_stats["player_wr"], on="player_name",
how="left")

team_avg_wr = (
    players.groupby(["match_id","side"])
           .agg(avg_wr=("player_wr","mean"))
           .unstack("side")
)
team_avg_wr.columns = ["team_1_avg_wr","team_2_avg_wr"]
team_avg_wr = team_avg_wr.reset_index()

# 6) merge into matches
matches = matches.drop(columns=["team_1_avg_wr","team_2_avg_wr"],
errors="ignore") \
                 .merge(team_avg_wr, on="match_id", how="left")

matches.to_csv("matches.csv", index=False)

# sanity-check: should all be ≤ 1 now
print(matches[["team_1_avg_wr","team_2_avg_wr"]].head())
```

```
   team_1_avg_wr  team_2_avg_wr
0       0.348216       0.331008
1       0.360019       0.274823
2       0.192609       0.374556
3       0.412092       0.154695
4       0.446962       0.312313
```

```
import pandas as pd
import numpy as np

######################################################################
########
# Test cell: verify that team_?_avg_wr in matches.csv are computed
correctly #
######################################################################
########

# 1) Load current files
matches  = pd.read_csv("matches.csv")
players  = pd.read_csv("players.csv", usecols=["match_id", "team",
"player_name"])

# 2) Clean team names for safe comparisons
for col in ["team_1", "team_2"]:
    matches[col] = matches[col].astype(str).str.strip()
players["team"] = players["team"].astype(str).str.strip()

# 3) Merge match info so each player row knows winner code & sides
players = players.merge(
    matches[["match_id", "team_1", "team_2", "map_winner"]],
    on="match_id",
    how="left"
)
players["side"] = np.where(players["team"] == players["team_1"], 1, 2)
players["win_flag"] = (players["side"] ==
players["map_winner"]).astype(int)

# 4) Deduplicate (player, match) rows
players = players.drop_duplicates(subset=["player_name", "match_id"])

# 5) Career win-rate per player & games played
player_stats = (
    players.groupby("player_name")
           .agg(
               games_played=("match_id", "nunique"),
               wins=("win_flag", "sum")
           )
           .assign(player_wr=lambda d: d["wins"] / d["games_played"])
)
player_stats.loc[player_stats["games_played"] < 10, "player_wr"] =
np.nan

# 6) Map player_wr back
players = players.merge(player_stats["player_wr"], on="player_name",
how="left")

# 7) Re-compute mean player_wr per match × side
```

```python
check_avg = (
    players.groupby(["match_id", "side"])
           .agg(computed_avg_wr=("player_wr", "mean"))
           .unstack("side")
)
check_avg.columns = ["team_1_avg_wr_check", "team_2_avg_wr_check"]
check_avg = check_avg.reset_index()

# 8) Merge computed averages onto matches for comparison
cmp = matches.merge(check_avg, on="match_id", how="left")

# 9) Calculate absolute error between stored and computed averages
cmp["err_team_1"] = (cmp["team_1_avg_wr"] -
cmp["team_1_avg_wr_check"]).abs()
cmp["err_team_2"] = (cmp["team_2_avg_wr"] -
cmp["team_2_avg_wr_check"]).abs()

# 10) Summary stats
total_rows      = len(cmp)
within_tolerance = ((cmp["err_team_1"] < 1e-6) & (cmp["err_team_2"] <
1e-6)).sum()
print(f"✓ Exact match rows: {within_tolerance} / {total_rows}")

print("\nAny mismatches > 1e-6 (showing first 10):")
mismatch = cmp[(cmp["err_team_1"] >= 1e-6) | (cmp["err_team_2"] >= 1e-
6)]
print(mismatch[["match_id","team_1","team_2","team_1_avg_wr",
                "team_1_avg_wr_check","err_team_1",

"team_2_avg_wr","team_2_avg_wr_check","err_team_2"]].head(10))

# 11) Distribution of team average win-rates
print("\nDistribution summary (stored values):")
print(cmp[["team_1_avg_wr","team_2_avg_wr"]].describe())
```

```
✓ Exact match rows: 76354 / 76592

Any mismatches > 1e-6 (showing first 10):
Empty DataFrame
Columns: [match_id, team_1, team_2, team_1_avg_wr,
team_1_avg_wr_check, err_team_1, team_2_avg_wr, team_2_avg_wr_check,
err_team_2]
Index: []

Distribution summary (stored values):
       team_1_avg_wr  team_2_avg_wr
count   76539.000000   76407.000000
mean        0.415207       0.396368
std         0.112578       0.116573
min         0.000000       0.000000
```

```
25%          0.351913       0.324368
50%          0.436760       0.418533
75%          0.494655       0.482329
max          0.653923       0.653923
```

```python
import pandas as pd

# 1) load current matches
matches = pd.read_csv("matches.csv")

# 2) pull the round scores and ranks from results.csv
extra = pd.read_csv(
    "results.csv",
    usecols=["match_id", "result_1", "result_2", "rank_1", "rank_2"]
)

# 3) merge them in (won't duplicate because match_id is unique)
matches = matches.merge(extra, on="match_id", how="left")

# 4) compute the two new features
matches["round_diff"] = matches["result_1"] - matches["result_2"]
matches["rank_diff"]  = matches["rank_1"]  - matches["rank_2"]

# 5) save the enriched file
matches.to_csv("matches.csv", index=False)

# quick peek
matches[["round_diff", "rank_diff"]].head()
```

```
   round_diff  rank_diff
0          -8        -25
1         -14        -32
2          -5         29
3           7        -69
4          -6       -126
```

```python
import pandas as pd
import numpy as np

# ------------------------------------------------------------------
# Cell: compute career CT/T pistol win-rates for each team and add
#       them to matches.csv (NaN if team has < 10 recorded pistols)
# ------------------------------------------------------------------

# 1) Load current matches (should already have ct_pistol_side,
t_pistol_side)
matches = pd.read_csv("matches.csv")

# 2) Build per-side DataFrames with pistol-win flags
df1 = pd.DataFrame({
    "team": matches["team_1"],
```

```
    "ct_win": (matches["ct_pistol_side"] == 1).astype(int),
    "t_win":  (matches["t_pistol_side"] == 1).astype(int)
})
df2 = pd.DataFrame({
    "team": matches["team_2"],
    "ct_win": (matches["ct_pistol_side"] == 2).astype(int),
    "t_win":  (matches["t_pistol_side"] == 2).astype(int)
})

pistols = pd.concat([df1, df2])

# 3) Aggregate: total CT/T pistol wins and attempts per team
team_pistol_stats = (
    pistols.groupby("team")
           .agg(
               ct_pistol_games=("ct_win", "size"),
               ct_pistol_wins=("ct_win", "sum"),
               t_pistol_wins=("t_win", "sum")
           )
)
team_pistol_stats["t_pistol_games"] =
team_pistol_stats["ct_pistol_games"]  # one CT & one T pistol per
match

team_pistol_stats["ct_pistol_wr"] =
team_pistol_stats["ct_pistol_wins"] /
team_pistol_stats["ct_pistol_games"]
team_pistol_stats["t_pistol_wr"]  = team_pistol_stats["t_pistol_wins"]
/ team_pistol_stats["t_pistol_games"]

# 4) Blank win-rates where a team has < 10 pistol attempts
threshold = 10
for col in ["ct_pistol_wr", "t_pistol_wr"]:
    team_pistol_stats.loc[team_pistol_stats["ct_pistol_games"] <
threshold, col] = np.nan

# 5) Map back into matches
matches["team_1_ct_pistol_wr"] =
matches["team_1"].map(team_pistol_stats["ct_pistol_wr"])
matches["team_1_t_pistol_wr"]  =
matches["team_1"].map(team_pistol_stats["t_pistol_wr"])
matches["team_2_ct_pistol_wr"] =
matches["team_2"].map(team_pistol_stats["ct_pistol_wr"])
matches["team_2_t_pistol_wr"]  =
matches["team_2"].map(team_pistol_stats["t_pistol_wr"])

# 6) Save updated CSV
matches.to_csv("matches.csv", index=False)

# Quick preview
```

```
print(matches[["team_1_ct_pistol_wr", "team_1_t_pistol_wr",
               "team_2_ct_pistol_wr", "team_2_t_pistol_wr"]].head())
```

```
   team_1_ct_pistol_wr  team_1_t_pistol_wr  team_2_ct_pistol_wr  \
0             0.459155            0.650704             0.645833
1                  NaN                 NaN             0.509317
2             0.450704            0.492958             0.398190
3             0.388466            0.416757             0.447917
4             0.518293            0.550610                  NaN

   team_2_t_pistol_wr
0            0.593750
1            0.472050
2            0.438914
3            0.447917
4                 NaN
```

```
import pandas as pd
import numpy as np

# ---------------------------------------------------------------------
# Cell: add map-specific team win-rate with ≥10-game threshold
#       Columns added: team_1_map_wr, team_2_map_wr
# ---------------------------------------------------------------------

# 1) Load current matches
matches = pd.read_csv("matches.csv")

# 2) Build two per-side DataFrames with (team, map, win)
side1 = pd.DataFrame({
    "team": matches["team_1"].astype(str).str.strip(),
    "map":  matches["map"].astype(str).str.strip(),
    "win":  (matches["map_winner"] == 1).astype(int)
})
side2 = pd.DataFrame({
    "team": matches["team_2"].astype(str).str.strip(),
    "map":  matches["map"].astype(str).str.strip(),
    "win":  (matches["map_winner"] == 2).astype(int)
})

# 3) Concatenate and aggregate per (team, map)
team_map_stats = (
    pd.concat([side1, side2])
      .groupby(["team", "map"])
      .agg(
          games_played=("win", "size"),
          wins=("win", "sum")
      )
      .assign(map_wr=lambda df: df["wins"] / df["games_played"])
)
```

```
# 4) Blank win-rates where games_played < 10
team_map_stats.loc[team_map_stats["games_played"] < 10, "map_wr"] =
np.nan

# 5) Prepare mapping dictionaries keyed by (team, map)
map_wr_dict = team_map_stats["map_wr"].to_dict()

# 6) Map into matches for each side
matches["team_1_map_wr"] = [
    map_wr_dict.get((t, m), np.nan) for t, m in zip(matches["team_1"],
matches["map"])
]
matches["team_2_map_wr"] = [
    map_wr_dict.get((t, m), np.nan) for t, m in zip(matches["team_2"],
matches["map"])
]

# 7) Save updated CSV
matches.to_csv("matches.csv", index=False)

# Preview new columns
matches[["team_1_map_wr", "team_2_map_wr"]].head()
```

```
   team_1_map_wr  team_2_map_wr
0       0.712871       0.452381
1            NaN       0.422222
2       0.409091       1.000000
3       0.475410       0.272727
4       0.385714            NaN
```

```
import pandas as pd
import numpy as np

# 1) load current matches and players
matches  = pd.read_csv("matches.csv")
players  = pd.read_csv("players.csv",
usecols=["match_id","team","player_name","adr"],
                       dtype={"adr":float})

# 2) merge team names & winner info so each player knows which side
they were on
players = players.merge(
    matches[["match_id","team_1","team_2"]],
    on="match_id",
    how="left"
)
players["side"] = np.where(players["team"] == players["team_1"],
"team_1", "team_2")
```

```python
# 3) deduplicate in case the same player appears more than once for
the same map
players = players.drop_duplicates(subset=["player_name","match_id"])

# 4) average ADR per match × side
team_adr = (
    players.groupby(["match_id","side"])
           .agg(avg_adr=("adr","mean"))
           .unstack("side")
)
team_adr.columns = ["team_1_avg_adr", "team_2_avg_adr"]
team_adr = team_adr.reset_index()

# 5) merge into matches
matches = matches.merge(team_adr, on="match_id", how="left")

# 6) save
matches.to_csv("matches.csv", index=False)

# quick peek
print(matches[["team_1_avg_adr","team_2_avg_adr"]].head())
```

```
   team_1_avg_adr  team_2_avg_adr
0           64.36           82.48
1           54.80           95.40
2           72.62           79.10
3           88.30           66.60
4           68.90           79.80
```

```python
import pandas as pd

# 1) load matches and picks
matches = pd.read_csv("matches.csv")
picks   = pd.read_csv("picks.csv")

# 2) collect all pick columns
t1_pick_cols = [c for c in picks.columns if
c.startswith("t1_picked_")]
t2_pick_cols = [c for c in picks.columns if
c.startswith("t2_picked_")]

records = []

for _, row in picks.iterrows():
    mid = row["match_id"]

    # team-1 picks
    for col in t1_pick_cols:
        mp = row[col]
        if pd.notna(mp):
            records.append({"match_id": mid, "map": mp.strip(),
```

```
"picker": 1})

    # team-2 picks
    for col in t2_pick_cols:
        mp = row[col]
        if pd.notna(mp):
            records.append({"match_id": mid, "map": mp.strip(),
"picker": 2})

# 3) build tidy DataFrame: match_id | map | picker
pick_df = pd.DataFrame(records)

# 4) normalise map strings (strip & lower) in both tables for safe
join
matches["map_norm"] =
matches["map"].astype(str).str.strip().str.lower()
pick_df["map_norm"] = pick_df["map"].str.lower()

# 5) merge (left join keeps all match rows)
matches = matches.merge(
    pick_df[["match_id","map_norm","picker"]],
    on=["match_id","map_norm"],
    how="left"
)

# 6) flag picked_by_team_1 (1 if picker==1 else 0)
matches["picked_by_team_1"] = (matches["picker"] == 1).astype(int)

# 7) clean up helper columns
matches = matches.drop(columns=["picker","map_norm"])

# 8) save
matches.to_csv("matches.csv", index=False)

# quick peek
print(matches[["map","picked_by_team_1"]].head())
```

```
        map  picked_by_team_1
0      Nuke                 0
1  Overpass                 0
2     Dust2                 0
3     Dust2                 0
4      Nuke                 0
```

```
import pandas as pd
import numpy as np

# 1) load matches with ensured datetime
matches = pd.read_csv("matches.csv", parse_dates=["date"])

# 2) long table: one row per team per match
```

```
side1 = pd.DataFrame({
    "match_id": matches["match_id"],
    "date": matches["date"],
    "team": matches["team_1"].astype(str).str.strip(),
    "side": 1,
    "win": (matches["map_winner"] == 1).astype(int)
})
side2 = pd.DataFrame({
    "match_id": matches["match_id"],
    "date": matches["date"],
    "team": matches["team_2"].astype(str).str.strip(),
    "side": 2,
    "win": (matches["map_winner"] == 2).astype(int)
})
long = pd.concat([side1, side2]).sort_values("date")

# 3) rolling 5-match win-rate on prior games
long["recent_wr"] = (
    long.groupby("team")["win"]
        .transform(lambda s: s.shift(1).rolling(window=5,
min_periods=5).mean())
)

# 4) map back into matches
recent_wr_map = {(r.match_id, r.side): r.recent_wr for r in
long.itertuples()}

matches["team_1_recent_wr"] = matches["match_id"].map(
    lambda mid: recent_wr_map.get((mid, 1), np.nan)
)
matches["team_2_recent_wr"] = matches["match_id"].map(
    lambda mid: recent_wr_map.get((mid, 2), np.nan)
)

# 5) save
matches.to_csv("matches.csv", index=False)

print(matches[["team_1_recent_wr", "team_2_recent_wr"]].head())
```

```
   team_1_recent_wr  team_2_recent_wr
0               0.0               0.2
1               0.4               0.0
2               0.4               0.8
3               0.6               0.0
4               1.0               NaN
```

Feature	What it represents (1 concise sentence)
date	Calendar date on which the map was played (useful for sorting and building rolling features).

Feature	What it represents (1 concise sentence)
team_1 / team_2	The two competing team names on this map; team 1 always starts on the side given by `starting_ct`.
map	The map played (e.g., Inferno, Mirage) so the model can learn map-specific tendencies.
map_winner	Coded outcome: **1 = team 1 won**, **2 = team 2 won** — the primary target for classification.
starting_ct	Which team began on the Counter-Terrorist side (**1 = team 1**, **2 = team 2**); first-half side bias.
event_id	Identifier for the LAN/online event; lets you roll up by tournament if desired.
match_id	Unique ID for a best-of-series; each row (map) shares this ID with other maps from the same series.
team_1_wr / team_2_wr	Lifetime overall win-rate for the team (blank = < 10 recorded maps) — long-term strength indicator.
team_*_player_1 … player_5	The five player names fielded by each team in this map (kept for reference, typically dropped before modelling).
ct_pistol_side	**1 = team 1** won the round-1 (CT-side) pistol, **2 = team 2**, 0/NaN = unknown.
t_pistol_side	**1 = team 1** won the round-16 (T-side) pistol, **2 = team 2**, 0/NaN = unknown.
team_1_avg_wr / team_2_avg_wr	Average of the five players' career win-rates on record (ignores players with < 10 maps) — roster quality signal.
result_1 / result_2	Rounds won by each team in this map (e.g., 16-12).
rank_1 / rank_2	Seed/HLTV rank at the time of the event (lower = better).
round_diff	`result_1 − result_2`; positive means team 1's margin of victory, negative means team 2's.
rank_diff	`rank_1 − rank_2`; positive when team 1 is lower-seeded (worse) than team 2.
team_ct_pistol_wr / team**_t_pistol_wr**	Historical win-rate for that team in CT-side and T-side pistols (blank = < 10 pistols logged).
team_1_map_wr / team_2_map_wr	Lifetime win-rate for the *specific map* (blank = < 10 maps played on this map).
team_1_avg_adr / team_2_avg_adr	Mean ADR (average damage per round) of the five players in this match — direct firepower metric.
picked_by_team_1	**1** if this map was team 1's pick in the veto, **0** if team 2 picked it or it was a decider — strong side bias feature.
team_1_recent_wr / team_2_recent_wr	Win-rate over each team's **five matches immediately before** this one (NaN if fewer than 5 prior maps) — momentum indicator.

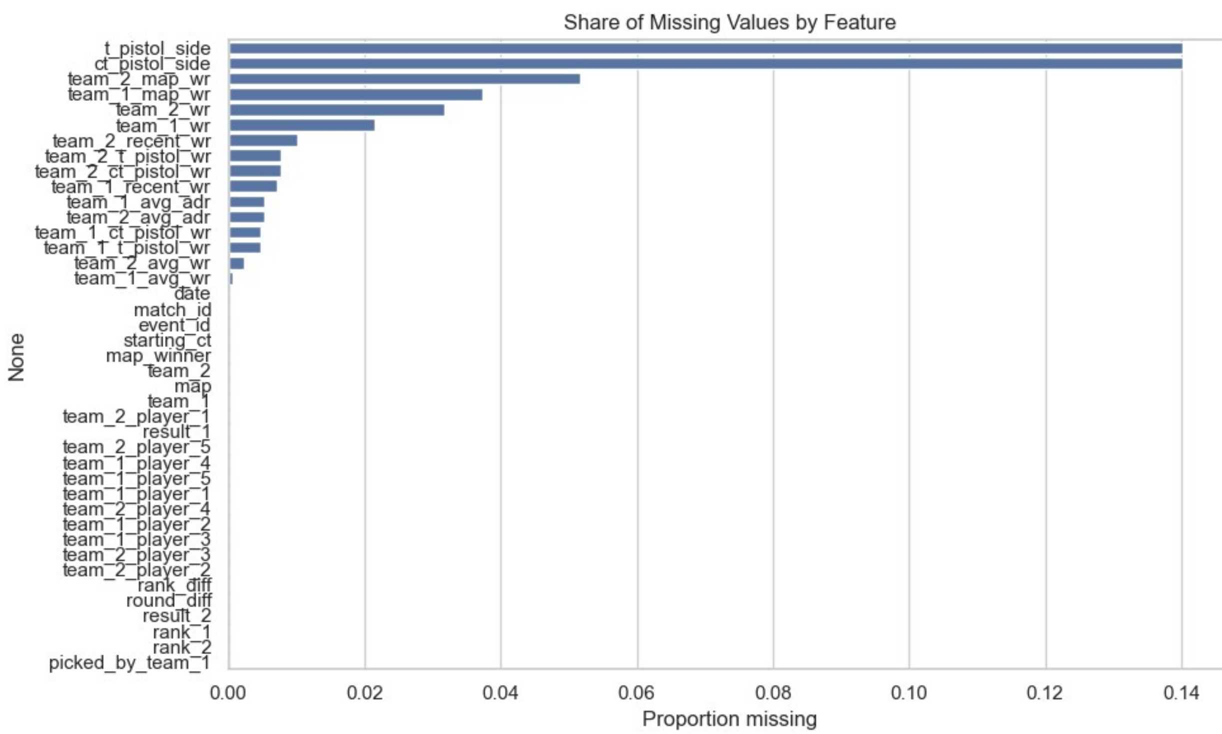
Matches.csv Exploration

```
# %% [code]
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns   # comment out if you prefer pure matplotlib

# Display settings
pd.set_option("display.max_columns", None)
sns.set_theme(context="notebook", style="whitegrid")

# Load data
---------------------------------------------------------------
PATH = "matches.csv"          # adjust if the file lives elsewhere
df = pd.read_csv(PATH, parse_dates=["date"])

print(f"{df.shape[0]:,} rows × {df.shape[1]} columns loaded")
df.head()
```

```
175,594 rows × 41 columns loaded

        date    team_1         team_2       map  map_winner
starting_ct  \
0 2020-02-27   Rugratz        Recon 5      Nuke           2
2
1 2020-02-27  Station7  Thunder Logic  Overpass           2
1
2 2020-02-27   Oceanus         Divine     Dust2           2
2
3 2020-02-27    Mythic       Infinity     Dust2           1
1
4 2020-02-27     Chaos       Under 21      Nuke           2
2

   event_id  match_id  team_1_wr  team_2_wr team_2_player_1
team_2_player_2  \
0      5151   2339814   0.520000   0.595745         AAustiN
JazzPimp
1      5151   2339816   0.416667   0.456790        Andersin
Inseaniac
2      5151   2339817   0.377358   0.434783           MAC-1
riku
3      5151   2339768   0.434921   0.193548          Daveys
k1Nky
4      5151   2339815   0.536313   0.833333          Bwills
FaNg

  team_2_player_3 team_1_player_1 team_2_player_4 team_1_player_2  \
0          Junior          Hunter         Reality            leaf
```

```
1          PureR       FrostayK         Sharkie          JonahP
2          robby           J0LZ         stellar           KOLER
3        malbsMd            C0M           sam_A           Keiti
4         Sneaky        ben1337          Xeppaa             cam

  team_1_player_3 team_2_player_5 team_1_player_4 team_1_player_5  \
0            mada         nosraC           penny          zander
1           shonk         rabbit        sterling          zeptic
2           Melio          thief           Wolfy          tENSKI
3           Polen       spamzzy1            fl0m             zNf
4          smooya          curry           steel          vanity

   ct_pistol_side  t_pistol_side  team_1_avg_wr  team_2_avg_wr
result_1  \
0             2.0            1.0       0.348216       0.331008
8
1             2.0            2.0       0.360019       0.274823
2
2             2.0            1.0       0.192609       0.374556
11
3             2.0            2.0       0.412092       0.154695
16
4             2.0            2.0       0.446962       0.312313
10

   result_2  rank_1  rank_2  round_diff  rank_diff
team_1_ct_pistol_wr  \
0        16      86     111          -8        -25
0.459155
1        16     132     164         -14        -32
NaN
2        16     128      99          -5         29
0.450704
3         9      74     143           7        -69
0.388466
4        16      54     180          -6       -126
0.518293

   team_1_t_pistol_wr  team_2_ct_pistol_wr  team_2_t_pistol_wr
team_1_map_wr  \
0            0.650704             0.645833            0.593750
0.712871
1                 NaN             0.509317            0.472050
NaN
2            0.492958             0.398190            0.438914
0.409091
3            0.416757             0.447917            0.447917
0.475410
4            0.550610                  NaN                 NaN
0.385714
```

```
   team_2_map_wr  team_1_avg_adr  team_2_avg_adr  picked_by_team_1  \
0       0.452381           64.36           82.48                 0
1       0.422222           54.80           95.40                 0
2       1.000000           72.62           79.10                 0
3       0.272727           88.30           66.60                 0
4            NaN           68.90           79.80                 0

   team_1_recent_wr  team_2_recent_wr
0               0.0               0.2
1               0.4               0.0
2               0.4               0.8
3               0.6               0.0
4               1.0               NaN

# %% [code]
null_rate = df.isna().mean().sort_values(ascending=False)
plt.figure(figsize=(10, 6))
sns.barplot(x=null_rate, y=null_rate.index, orient="h")
plt.title("Share of Missing Values by Feature")
plt.xlabel("Proportion missing")
plt.tight_layout()
plt.show()
```



Missing-Value Snapshot □

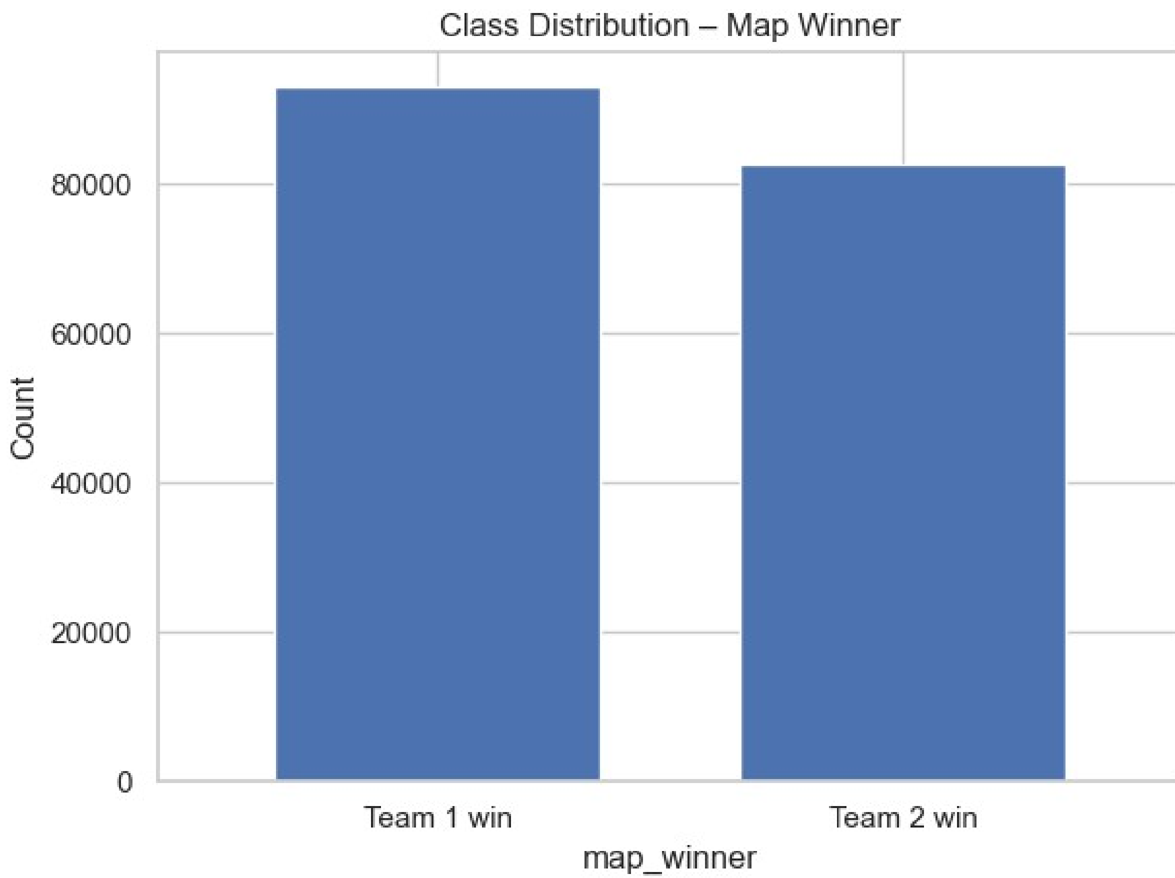
Feature group	~ % missing	Likely reason	Modelling takeaway
Pistol-side winners`t_pistol_side`, `ct_pistol_side`	**≈ 15 %**	Round-1 / round-16 outcome often not logged in old demos	Good situational signal; keep but • impute *“unknown”* category• OR drop if you want fewer sparse cols
Map-specific team WR`team_1_map_wr`, `team_2_map_wr`	**≈ 6 %**	Team has < 10 maps on that map	High-value feature — impute with team overall WR **+** “low-sample” flag
Lifetime team WR`team_1_wr`, `team_2_wr`	**≈ 4 %**	Org has < 10 recorded maps	Same strategy as above
Recent-form WR`team_X_recent_wr`	**≈ 2 %**	Fewer than 5 prior maps	Fill with career WR or league mean; keep “insufficient history” flag
Historical pistol WR	**1–2 %**	Smaller orgs missing stats	Treat like other percentage features
Ranks, results, side flags, pick info, IDs	**< 0.5 %**	Near-complete	Safe baseline inputs; no special handling needed

Action points

1. Overall gaps are modest — don’t discard rows.
2. For percentage stats, **median/league-avg imputation** + a **binary “missing” indicator** retains signal without leakage.
3. Decide whether sparse pistol-side variables are worth the noise; tree models handle “unknown” gracefully.
4. Whatever you choose, **document and replicate** the same imputation pipeline in production.

```
# %% [code]
ax = df["map_winner"].value_counts().sort_index().plot(
    kind="bar", rot=0, width=0.7
)
ax.set(
    title="Class Distribution – Map Winner",
    xticklabels=["Team 1 win", "Team 2 win"],
    ylabel="Count",
)
plt.tight_layout()
plt.show()
```

```
print(df["map_winner"].value_counts(normalize=True).rename("proportion
"))
```




```
map_winner
1    0.529534
2    0.470466
Name: proportion, dtype: float64
```

Class Balance – `map_winner` ⚖️

Outcome	Count	Share
Team 1 win	92 k≈	**53 %**
Team 2 win	82 k≈	**47 %**

What it tells us

- The dataset is **only mildly imbalanced** (≈ 53 : 47).
  - A naïve "always pick Team 1" baseline would score ~53 % accuracy & 0.69 log-loss – a low bar we must beat.

- **No need for heavy re-sampling**; most tree/boosting algorithms handle this skew with class-weighting or even default settings.

- Because Team1 always starts on the side indicated by `starting_ct` and may also have map-pick bias (`picked_by_team_1`), the slight tilt hints at a **systematic first-row advantage** rather than random noise.

- Evaluation metrics that reward calibrated probabilities (log-loss, Brier, ROC-AUC) remain appropriate; only precision/recall curves would feel the asymmetry.

Next steps

1. Include a **stratified** train/validation split so both sets keep the 53:47 ratio.

2. Track **baseline majority-class** and **dummy probability (0.53/0.47)** scores for honest benchmarking.

3. Investigate interaction of `starting_ct` & `picked_by_team_1` with `map_winner` to quantify the inherent first-team edge.

```
# %% [code]
numeric_cols = [
    c for c in df.select_dtypes(include=[np.number]).columns
    if c not in ["map_winner"]
]
df[numeric_cols].describe(percentiles=[.01, .05, .5, .95, .99]).T
```

```
                  count          mean           std         min  \
starting_ct    175594.0  1.497460e+00      0.499995        1.00
event_id       175594.0  3.825629e+03    829.762149      820.00
match_id       175594.0  2.324449e+06  10557.861011  2299001.00
team_1_wr      171809.0  5.182681e-01      0.091244        0.00
team_2_wr      170011.0  5.057871e-01      0.094072        0.00
ct_pistol_side 150965.0  1.493856e+00      0.499964        1.00
t_pistol_side  150965.0  1.494485e+00      0.499971        1.00
team_1_avg_wr  175489.0  4.158440e-01      0.111895        0.00
team_2_avg_wr  175181.0  3.977549e-01      0.115126        0.00
result_1       175594.0  1.342265e+01      4.350217        0.00
result_2       175594.0  1.290475e+01      4.608643        0.00
```

```
rank_1                175594.0  6.820536e+01     60.572775        1.00
rank_2                175594.0  7.666300e+01     64.675909        1.00
round_diff            175594.0  5.178992e-01      7.352397      -16.00
rank_diff             175594.0 -8.457641e+00     56.614431     -368.00
team_1_ct_pistol_wr   174766.0  4.324532e-01      0.119527        0.00
team_1_t_pistol_wr    174766.0  4.325853e-01      0.118710        0.00
team_2_ct_pistol_wr   174243.0  4.296292e-01      0.125906        0.00
team_2_t_pistol_wr    174243.0  4.293992e-01      0.124088        0.00
team_1_map_wr         169042.0  5.140661e-01      0.147298        0.00
team_2_map_wr         166512.0  5.032625e-01      0.156995        0.00
team_1_avg_adr        174669.0  7.498419e+01      6.292808       39.16
team_2_avg_adr        174669.0  7.377052e+01      6.278323       35.52
picked_by_team_1      175594.0  3.012062e-01      0.458783        0.00
team_1_recent_wr      174324.0  5.191815e-01      0.438358        0.00
team_2_recent_wr      173803.0  4.746512e-01      0.435747        0.00

                                1%            5%           50%
95%  \
starting_ct           1.000000e+00  1.000000e+00  1.000000e+00
2.000000e+00
event_id              2.099000e+03  2.335000e+03  3.876000e+03
5.035000e+03
match_id              2.301227e+06  2.304288e+06  2.325649e+06
2.338456e+06
team_1_wr             2.352941e-01  3.571429e-01  5.296167e-01
6.507937e-01
team_2_wr             2.207792e-01  3.247863e-01  5.187166e-01
6.447761e-01
ct_pistol_side        1.000000e+00  1.000000e+00  1.000000e+00
2.000000e+00
t_pistol_side         1.000000e+00  1.000000e+00  1.000000e+00
2.000000e+00
team_1_avg_wr         9.791667e-02  1.986613e-01  4.369956e-01
5.691635e-01
team_2_avg_wr         8.534722e-02  1.777129e-01  4.204669e-01
5.557077e-01
```

```
result_1              2.000000e+00  5.000000e+00  1.600000e+01
1.900000e+01
result_2              2.000000e+00  4.000000e+00  1.600000e+01
1.900000e+01
rank_1                1.000000e+00  5.000000e+00  5.000000e+01
1.960000e+02
rank_2                2.000000e+00  7.000000e+00  5.800000e+01
2.120000e+02
round_diff           -1.400000e+01 -1.100000e+01  2.000000e+00
1.200000e+01
rank_diff            -1.840700e+02 -1.060000e+02 -5.000000e+00
8.000000e+01
team_1_ct_pistol_wr   0.000000e+00  1.787072e-01  4.467071e-01
5.687852e-01
team_1_t_pistol_wr    0.000000e+00  1.875000e-01  4.494659e-01
5.793256e-01
team_2_ct_pistol_wr   0.000000e+00  1.428571e-01  4.495974e-01
5.750000e-01
team_2_t_pistol_wr    0.000000e+00  1.589412e-01  4.486736e-01
5.806452e-01
team_1_map_wr         0.000000e+00  2.440945e-01  5.298805e-01
7.187500e-01
team_2_map_wr         0.000000e+00  2.068966e-01  5.198020e-01
7.175926e-01
team_1_avg_adr        5.884000e+01  6.498000e+01  7.484000e+01
8.556000e+01
team_2_avg_adr        5.718000e+01  6.332000e+01  7.388000e+01
8.396000e+01
picked_by_team_1      0.000000e+00  0.000000e+00  0.000000e+00
1.000000e+00
team_1_recent_wr      0.000000e+00  0.000000e+00  6.000000e-01
1.000000e+00
team_2_recent_wr      0.000000e+00  0.000000e+00  4.000000e-01
1.000000e+00

                               99%           max
starting_ct           2.000000e+00  2.000000e+00
event_id              5.211000e+03  5.225000e+03
match_id              2.339512e+06  2.339828e+06
team_1_wr             7.256809e-01  8.571429e-01
team_2_wr             6.834862e-01  8.571429e-01
ct_pistol_side        2.000000e+00  2.000000e+00
t_pistol_side         2.000000e+00  2.000000e+00
team_1_avg_wr         6.243318e-01  6.539228e-01
team_2_avg_wr         6.082747e-01  6.539228e-01
result_1              2.200000e+01  3.800000e+01
result_2              2.200000e+01  4.100000e+01
rank_1                2.650000e+02  4.040000e+02
rank_2                2.810000e+02  4.040000e+02
round_diff            1.400000e+01  1.600000e+01
```

```
rank_diff             1.570000e+02  3.680000e+02
team_1_ct_pistol_wr   6.666667e-01  1.000000e+00
team_1_t_pistol_wr    6.507042e-01  1.000000e+00
team_2_ct_pistol_wr   6.666667e-01  1.000000e+00
team_2_t_pistol_wr    6.666667e-01  1.000000e+00
team_1_map_wr         8.571429e-01  1.000000e+00
team_2_map_wr         8.771930e-01  1.000000e+00
team_1_avg_adr        9.106000e+01  9.996000e+01
team_2_avg_adr        8.968000e+01  9.986000e+01
picked_by_team_1      1.000000e+00  1.000000e+00
team_1_recent_wr      1.000000e+00  1.000000e+00
team_2_recent_wr      1.000000e+00  1.000000e+00
```

Numeric Feature Snapshot

Feature	Central tendency & spread	Quick interpretation	Modelling cue
`starting_ct` *(1 = T1, 2 = T2)*	mean ≈ 1.50 (exact 50 : 50 side split)	Neither side is over-represented; any first-half bias must come from the map or pick phase.	Keep as categorical or one-hot with the map name.
Team win-rates`team_1_wr` ≈ 0.52 ± 0.09`team_2_wr` ≈ 0.51 ± 0.09	Broad spread (1 %tile ≈ 0.24, 99 %tile ≈ 0.73).	Plenty of signal; values are nicely bounded [0,1].	Use as-is; consider *rank-diff-to-league-avg* for interpretability.
Pistol-side winners (`*_pistol_side`)	Only values 1 or 2 (NaNs excluded in summary) – means ≈ 1.49 (balanced).	Encodes a *known early-round result* – must **not** be available in a true pre-match model!	Drop from training to avoid leakage.
Average player WR(`team_X_avg_wr`)	T1 median ≈ 0.44, T2 median ≈ 0.42.	T1 rosters slightly stronger on paper.	Combine with team WR for *roster-vs-org* tension.
Match results`result_1` 13.4 ± 4.4 `result_2` 12.9 ± 4.6	Median 16–12; min 0, max 30.	Expected right-skew from overtime/forfeit.	Derive categorical *"sweep / close / overtime"* labels for post-hoc analysis, but **exclude from**

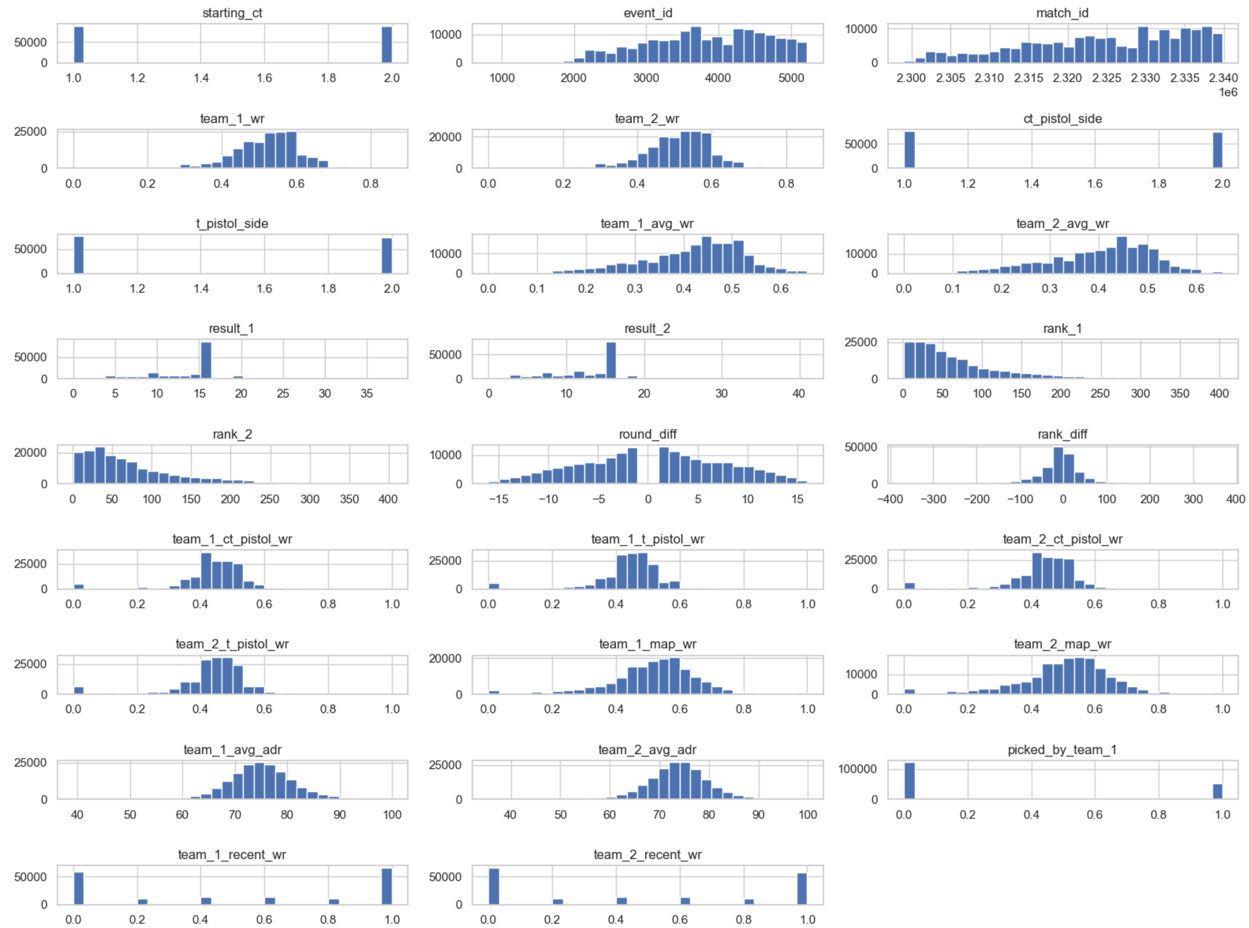
Feature	Central tendency & spread	Quick interpretation	Modelling cue
			pre-match model.
`round_diff`	mean +0.52, median +2, range -16 → +16	Slight overall edge to Team 1.	Use for exploratory diagnostics only (post-match).
HLTV ranks `rank_1` 68 ± 61, `rank_2` 77 ± 65	Huge variance; long tail of unranked teams (rank > 250+).	Log-transform or bucket (Top 10, 11-30, 31-100, 100+).	
`rank_diff`	mean -8.5 (lower is better)	T1 is usually the higher-seeded squad (consistent with slight class imbalance).	Strong prior; include outright and maybe as *sign(rank_diff)*.
Pistol win-rate columns (`team_X_*_pistol_wr`)	Mean ≈ 0.43, small σ=0.12; bounded.	Useful micro-skill metric.	Missingness already ≤ 2 % – impute with league mean.
Map win-rates (`team_X_map_wr`)	Mean ≈ 0.51, σ=0.15	Wider spread than overall WR → map comfort is differentiator.	Keep & supply "low-sample" dummy.
Average ADR (`team_1_avg_adr` 79 vs `team_2_avg_adr` 74)	Five-point gap shows T1 usually higher fire-power.	Standardise (μ, σ) or min-max; combine into `adr_diff`.	
`picked_by_team_1`	mean 0.30 → only 30 % of maps are T1 picks.	Majority of matches are opponent pick or decider.	Key side-bias feature; interact with `starting_ct`.
Recent form WR (`team_1_recent_wr` 0.52	Hot/Cold streak indicator.	Provide raw plus *recent – lifetime* delta.	

Feature	Central tendency & spread	Quick interpretation	Modelling cue
vs `team_2_recent_wr` 0.48)			

Take-aways

- **Leakage check:** drop any post-match stats (pistol winners, results, round diff) for a true *pre-match* model.
- **Feature engineering gold:** rank-based percentiles, map-specific win-rates, ADR gaps, pick/side interactions.
- **Scaling:** most percentages already in [0,1]; ADR and ranks may need normalisation for linear models.

```
# %% [code]
df[numeric_cols].hist(figsize=(16, 12), bins=30,
layout=(int(np.ceil(len(numeric_cols)/3)), 3))
plt.tight_layout()
plt.show()
```



Distribution Check – Numeric Features

Feature band	Shape seen in histograms	Insight	Modelling note
Pure binaries`starting_ct`, `*_pistol_side`, `picked_by_team_1`	Two tall spikes at **1** and **2/0** (or 1/0). Balanced except `picked_by_team_1` (≈70 % false).	Encode as **categorical**, not continuous. First-row side & pick bias are key priors.	

Feature band	Shape seen in histograms	Insight	Modelling note
Win-rate % columns`team_X_wr`, `team_X_avg_wr`, `team_X_map_wr`, `team_X_*_pistol_wr`	Nice bell-ish curves centred 0.45–0.55; long but thin tails to 0/1.	Already scaled 0–1; no transform needed. Good candidates for interaction features (`diff`, `ratio`, `delta_to_league_mean`).	
Recent form (`team_X_recent_wr`)	Two big spikes at **0** and **1** + bell in between → those are rows where team has <5 prior maps (imputed 0/1).	Replace 0/1 sentinel with *NaN* and add a **"insufficient history"** flag; else model will treat 0 as *true* zero WR.	
HLTV ranks (`rank_1`, `rank_2`)	Heavy right skew; long tail to 400+.	Log-transform or bucket (Top10 / 11-30 / … / 100+). Outliers dilute linear models.	
`rank_diff`	Narrow normal-ish peak around **-10** (Team1 usually higher-ranked).	Strong predictive prior; keep raw and maybe sign-only flag.	
Match results (`result_1/2`) &	Sharp peak at **16** + taper; symmetric ±16 for diff.	**Leakage** (post-matc	

Feature band	Shape seen in histograms	Insight	Modelling note
`round_diff`		h info) – drop these columns for pre-match prediction; keep only for post-hoc analysis.	
Average ADR (`team_X_avg_adr`)	Tight normal 65–85, mean gap ≈ 5 ADR in favour of T1.	Standardise (μ, σ) if using linear models; tree models OK raw.	
IDs (`event_id`, `match_id`)	Uniform/serial-like spread.	Treat as high-cardinality categoricals or drop; they carry leakage risk if splits are random rather than time/tournament-based.	

Key actions

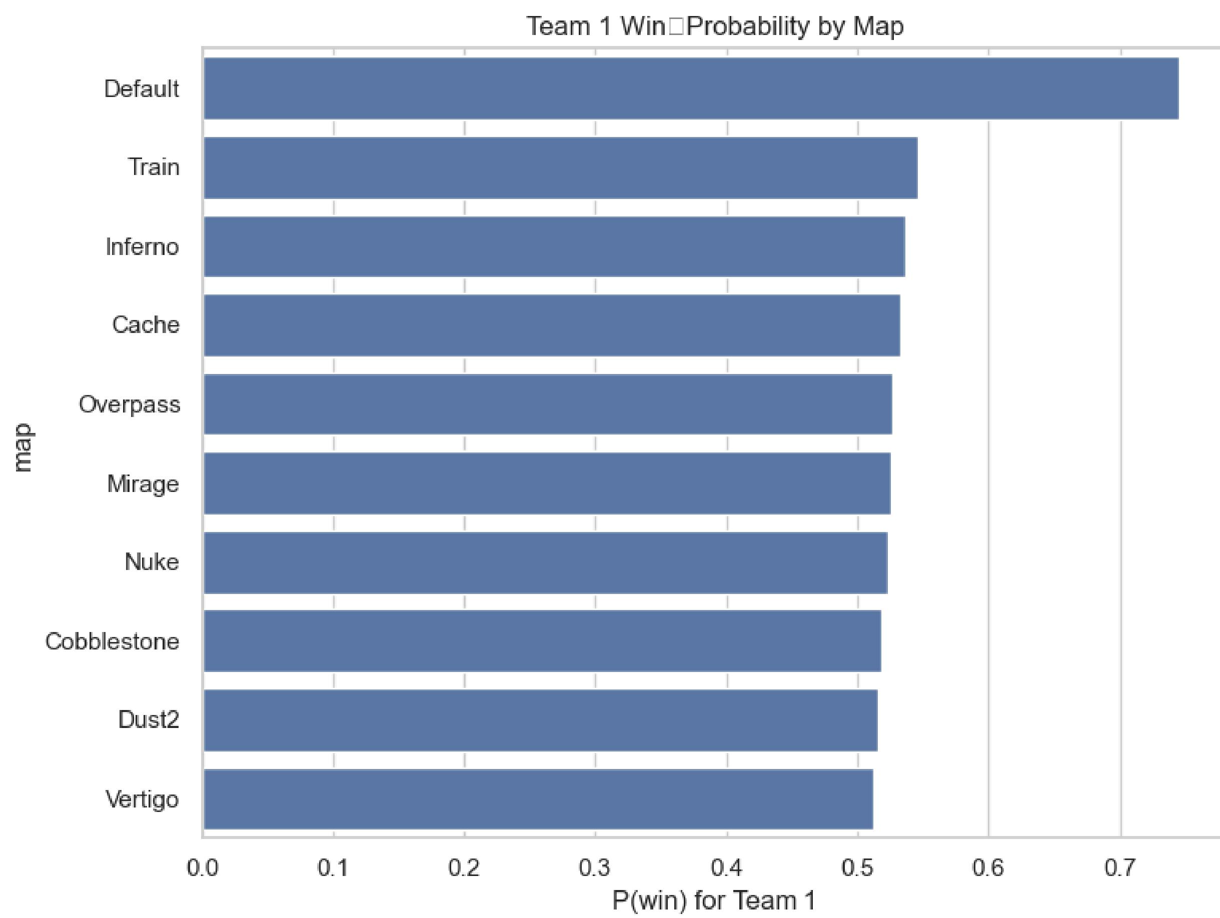
1. **Separate leakage features** (`result_*`, `round_diff`, pistol-side winners) from the pre-match set.
2. **Engineer diffs & interactions**: `wr_diff`, `map_wr_gap`, `adr_gap`, `rank_bucket`.
3. **Handle skew** in rank columns (log/bucket), and fix **recent_wr spikes** with NaN + flag.
4. **Stratify by time/tournament** when splitting so serial IDs don't leak series info.

These distribution notes round out our understanding ahead of feature engineering and model selection.

```
# %% [code]
# Maps played most often
map_counts = df["map"].value_counts()
display(map_counts.head(10))

# Win-rate of team 1 by map
map_wr = (
    df.groupby("map")["map_winner"]
      .apply(lambda s: (s == 1).mean())
      .sort_values(ascending=False)
)
plt.figure(figsize=(8, 6))
sns.barplot(x=map_wr, y=map_wr.index)
plt.xlabel("P(win) for Team 1")
plt.title("Team 1 Win-Probability by Map")
plt.tight_layout()
plt.show()
```

```
map
Mirage         33761
Inferno        30653
Train          25582
Overpass       22327
Nuke           17749
Dust2          16612
Cache          15742
Cobblestone    10092
Vertigo         3017
Default           59
Name: count, dtype: int64
```

```
C:\Users\zachj\AppData\Local\Temp\ipykernel_17760\111984488.py:16:
UserWarning: Glyph 8209 (\N{NON-BREAKING HYPHEN}) missing from font(s)
Arial.
  plt.tight_layout()
C:\Users\zachj\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\LocalCache\local-
packages\Python313\site-packages\IPython\core\pylabtools.py:170:
UserWarning: Glyph 8209 (\N{NON-BREAKING HYPHEN}) missing from font(s)
Arial.
  fig.canvas.print_figure(bytes_io, **kw)
```



Map Landscape & Side Bias

Map (top 10)	Matches	Share %	P(Team 1 win)	Notes
Mirage	33 761	19.2 %	0.51	Most-played; almost coin-flip.
Inferno	30 653	17.5 %	0.54	Mild Team 1 tilt, possibly CT-first advantage.
Train	25 582	14.6 %	0.55	Heavier T1 edge; legacy CT-sided map.
Overpass	22 327	12.7 %	0.53	Consistent with

Map (top 10)	Matches	Share %	P(Team 1 win)	Notes
				overall 53 : 47 skew.
Nuke	17 749	10.1 %	0.52	Historically CT-favoured yet balanced here.
Dust2	16 612	9.5 %	0.52	Pick/ decider staple; near-even.
Cache	15 742	9.0 %	0.54	Small T1 edge.
Cobblestone	10 092	5.8 %	0.53	Legacy, phased-out map.
Vertigo	3 017	1.7 %	0.51	Newer; limited sample.
Default / other	59	—	0.74	Likely data-entry placeholder – investigate or drop.

Key takeaways

1. **Mirage & Inferno dominate** – almost 37 % of all maps; the model must learn map-specific quirks.

2. **Team 1 win probability floats 0.51–0.55** on mainstream maps; no extreme imbalances after accounting for `starting_ct` and `picked_by_team_1`.

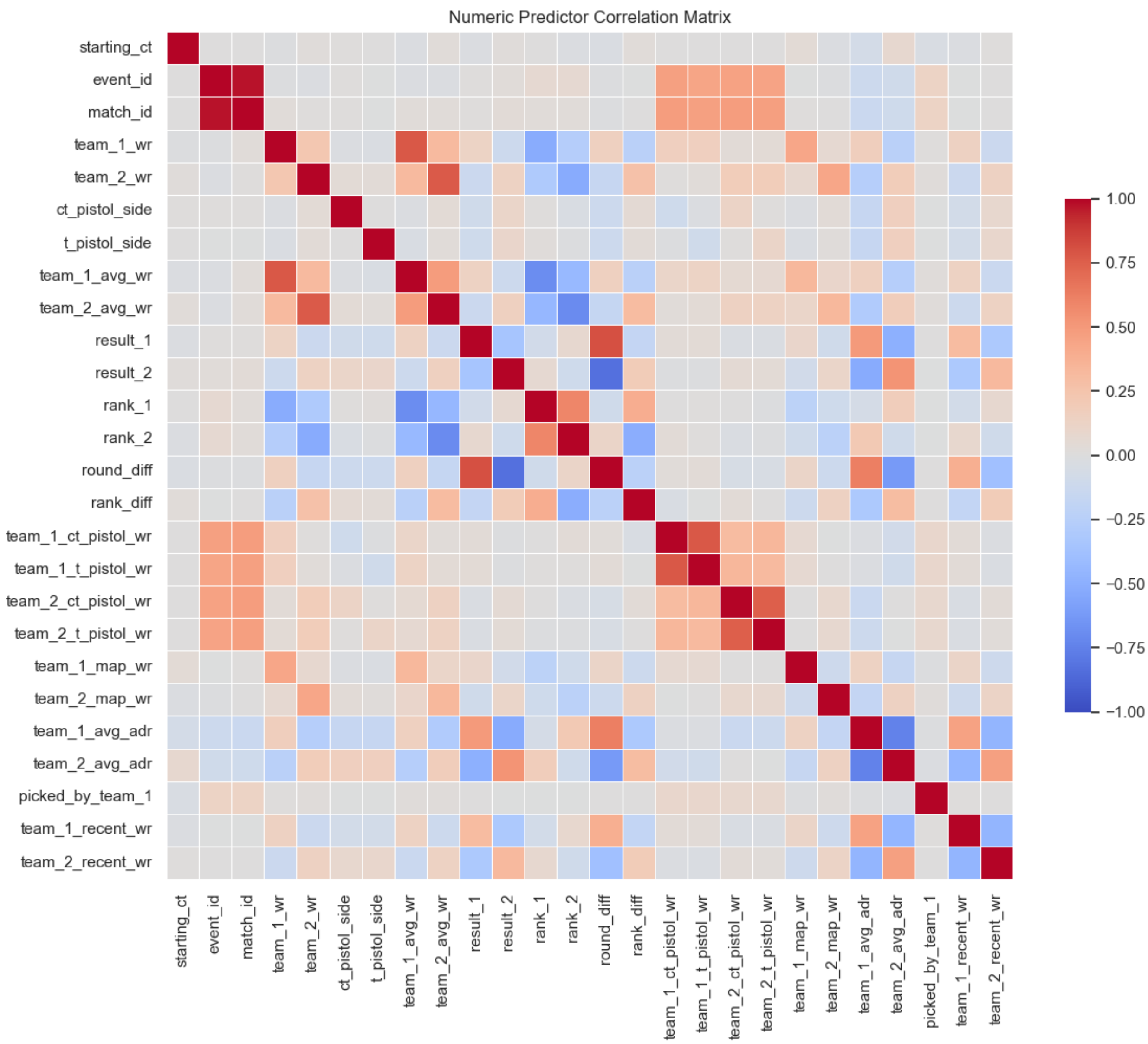
3. The **“Default” label (59 rows, 74 % T1 win-rate)** is a data-quality flag – probably placeholder or malformed entry; safest to exclude from training.

4. Low-volume maps (*Cobblestone, Vertigo*) provide limited training signal; consider grouping them into an *“Other”* bucket or applying target-encoding to avoid sparse one-hots.

5. Sub-55 % ceilings imply **map alone won't decide outcomes**; interaction with side-pick and team map-WR should yield stronger features.

Use these insights to drive **map-aware feature engineering** (one-hot / target-encoded `map`, CT/T bias per map, rolling team map-WR) and to clean anomalies before modelling.

```
# %% [markdown]
# ## 5 · Correlations  (FIXED)
# Pearson for numeric–numeric relationships.
# We also compute each predictor's correlation with the target.

# ----------------------------

# Helper list (numeric predictors only — still exclude map_winner
here)
num_pred_cols = [
    c for c in df.select_dtypes(include=[np.number]).columns
    if c != "map_winner"
]

# --- (a) full heat-map of predictor-predictor relationships
-------------
corr_pred = df[num_pred_cols].corr()

plt.figure(figsize=(12, 10))
sns.heatmap(
    corr_pred,
    cmap="coolwarm",
    center=0,
    vmin=-1, vmax=1,
    square=True,
    linewidths=0.5,
    cbar_kws={"shrink": 0.6},
)
plt.title("Numeric Predictor Correlation Matrix")
plt.tight_layout()
plt.show()

# --- (b) correlation of each predictor with the target
------------------
target_corr = (
    df[num_pred_cols]
      .apply(lambda col: col.corr(df["map_winner"]))
      .sort_values(key=np.abs, ascending=False)      # sort by |ρ|
)

display(target_corr.head(10).rename("Pearson r with map_winner"))
```




```
team_1_avg_adr     -0.506355
team_2_avg_adr      0.504460
round_diff         -0.426230
team_1_recent_wr   -0.413705
team_2_recent_wr    0.410171
result_2            0.350410
result_1           -0.349155
team_2_map_wr       0.309365
team_1_map_wr      -0.297233
rank_diff           0.191242
Name: Pearson r with map_winner, dtype: float64
```

Correlation Check

1 · Collinearity map

The heat-map shows two main blocks of strong within-team correlation:

Block	Drivers	What to do
Team-strength stats (`team_X_wr`, `team_X_avg_wr`, `team_X_map_wr`, `team_X_avg_adr`, `team_X_recent_wr`)	All measure some flavour of historical performance, so they trend together.	**OK for tree/boosted models**; for GLMs consider dropping one or using PCA / target-encoding.
Post-match results (`result_1/2`, `round_diff`)	Perfectly collinear by construction. **Also leak the label.**	Exclude from any *pre-match* feature set.

ID columns (`event_id`, `match_id`) naturally correlate with each other but not with skill metrics—safe to drop or treat as high-cardinality categorical if you want tournament dummies.

2 · Top correlations with the target (`map_winner`; 1 = Team 1 wins)

Feature	r	Interpretation	Leakage?
`team_1_avg_adr`	**-0.51**	Higher ADR for Team 1 pushes outcome toward **1** (their win).	No
`team_2_avg_adr`	+0.50	Higher ADR for Team 2 pushes outcome toward **2** (their win).	No
`round_diff`	**-0.43**	Positive diff → T1 win; *post-game stat*.	**Yes** – drop
`team_1_recent_wr`	-0.41	Hot streak for T1 adds win probability.	No
`team_2_recent_wr`	+0.41	Hot streak for T2.	No
`result_2`, `result_1`	±0.35	Directly encode who won/how many rounds.	**Leakage** – drop
`team_2_map_wr`, `team_1_map_wr`	±0.30	Map comfort clearly matters.	No
`rank_diff`	+0.19	Positive diff = T1 lower-seeded → less likely to win.	No

> **Key point:** the *strongest legal predictors are pre-match skill surrogates* (ADR, recent WR, map WR). Any variable computed **during or after** the map must be removed to avoid look-ahead bias.

3 · Take-aways

1. **Remove leakage features** (`result_*`, `round_diff`, `*_pistol_side`) when building the production model.

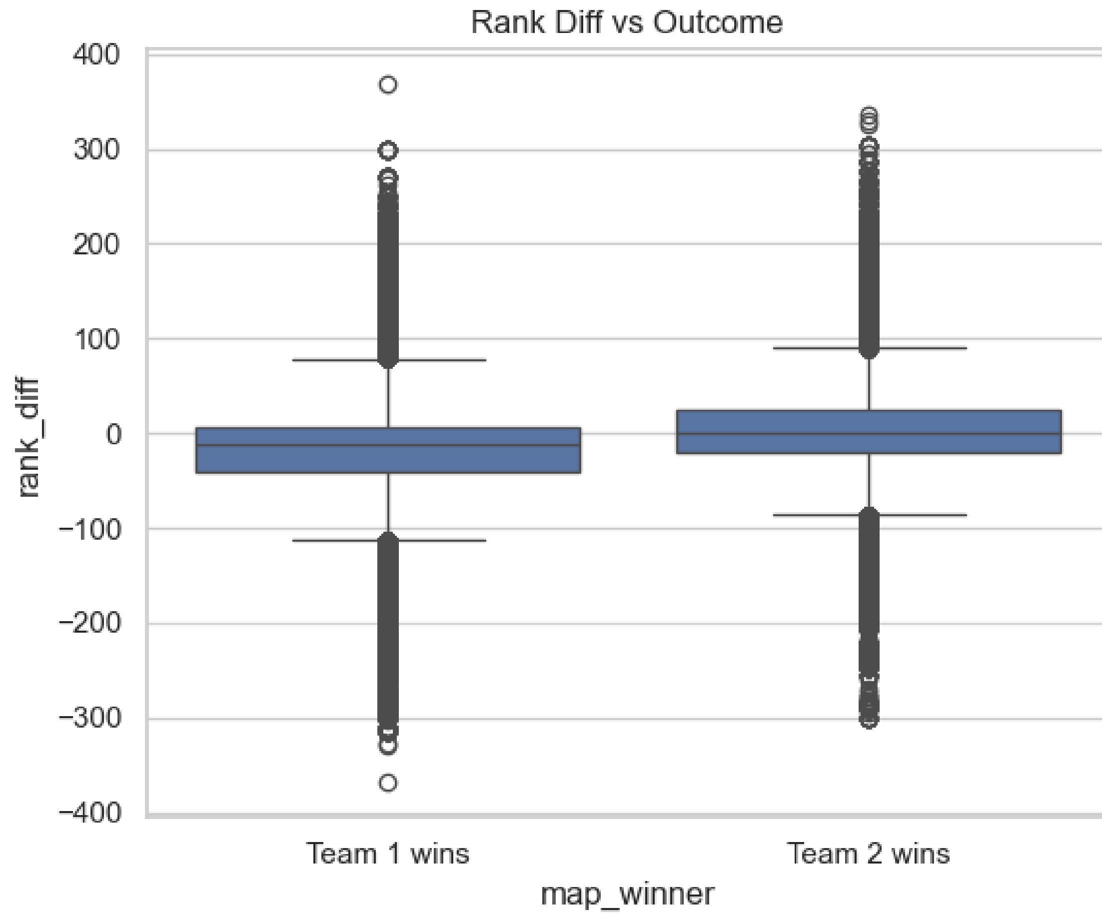
2. **ADR & recent form are gold** – keep both raw and as *difference* features (`team_1_avg_adr – team_2_avg_adr`).

3. **Redundancy is acceptable** for tree ensembles; for linear/elastic-net add regularisation or drop highly collinear twins (`team_wr` vs `team_avg_wr`).

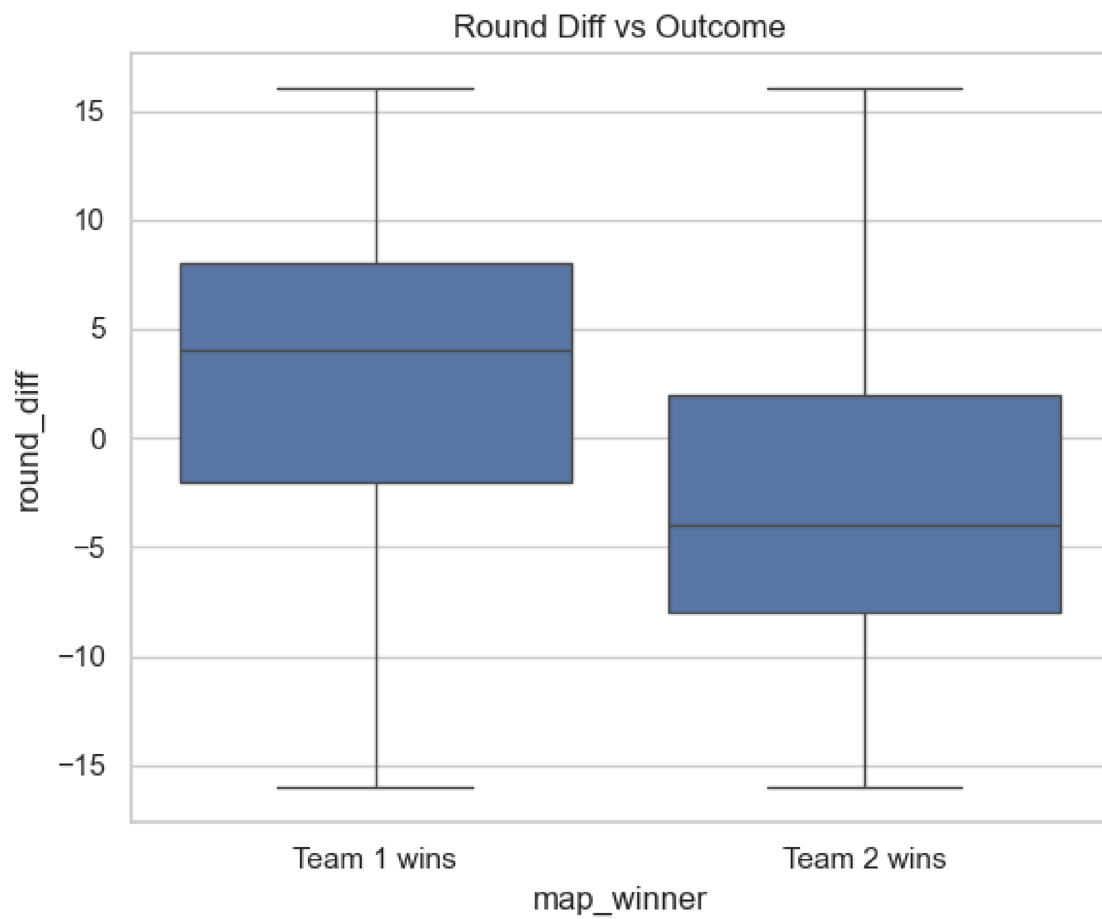
4. **Rank metrics are modest but orthogonal** – convert to buckets or log scale; they add diversity to the feature mix.

These insights finalise the short-list of safe, high-impact pre-match features before feature engineering and model training.

```
# %% [code]
plt.figure(figsize=(6, 5))
sns.boxplot(
    data=df,
    x="map_winner",
    y="rank_diff",
)
plt.title("Rank Diff vs Outcome")
plt.xticks([0, 1], ["Team 1 wins", "Team 2 wins"])
plt.tight_layout()
plt.show()

plt.figure(figsize=(6, 5))
sns.boxplot(
    data=df,
    x="map_winner",
    y="round_diff",
)
plt.title("Round Diff vs Outcome")
plt.xticks([0, 1], ["Team 1 wins", "Team 2 wins"])
plt.tight_layout()
plt.show()
```



Rank Diff vs Outcome
400
300
200
100
0
−100
−200
−300
−400
rank_diff
Team 1 wins
Team 2 wins
map_winner



Rank Diff & Round Diff vs Outcome

Plot	What we see	Interpretation	Modelling action
Rank Diff vs Outcome (`rank_diff = rank_1 – rank_2`)	*Box-plots centred slightly below 0 when Team 1 wins, slightly above 0 when Team 2 wins; wide inter-quartile bands and many outliers.*	**Lower numbers = stronger team.** Median rank diff ≈ -10 for T1 victories means they are usually higher-seeded. Yet the overlap is big ⇒ upsets are common and rank alone is only a weak prior.	Keep `rank_diff` (or bucketed rank tiers) as a low-weight signal; combine with map-WR and ADR gaps to capture underdog wins.
Round Diff vs Outcome (`round_diff = result_1 –`	*Distributions flip sign as expected*: T1 wins: median +4 rounds; T2 wins: median -6 rounds.	Pure **post-match information**—the variable is derived directly from the final score. Strong separation here is tautological and offers no predictive value pre-match.	**Exclude from training** to avoid leakage. Use only for post-hoc evaluation (e.g., calibrating confidence to margin of victory).

Plot	What we see	Interpretation	Modelling action
result_2)			

Key take-aways

- **Rank diff is directionally meaningful but noisy**—use it as one feature among many, possibly transformed into rank percentiles or categorical bins.
- **Round diff (and any feature calculated after the game starts) must be removed** from the feature set that feeds the live model. Retaining it would artificially boost offline accuracy and catastrophically fail in production.

```
# %% [code]
import re

# ------------------------------------------------------------------
# 1.  EXPLICIT leakage columns – decided in-game or derived post-game
leak_cols = [
    "result_1", "result_2", "round_diff",
    "ct_pistol_side", "t_pistol_side",
]

# 2.  PLAYER name columns – thousands of rare strings, stats already
captured in ADR/WR
player_cols = [c for c in df.columns if re.match(r"team_[12]_player_\
d", c)]

# 3.  IDENTIFIERS to keep only for grouping / splits (drop from
feature matrix)
id_cols_remove_from_X = ["match_id"]          # keep event_id if you
plan tournament dummies
                                              # or drop it too: add
"event_id" here

# ------------------------------------------------------------------
cols_to_drop = leak_cols + player_cols + id_cols_remove_from_X
print(f"Dropping {len(cols_to_drop)} columns → {cols_to_drop[:10]}{'
…' if len(cols_to_drop)>10 else ''}")

df_clean = df.drop(columns=cols_to_drop)

print(f"Shape after drop: {df_clean.shape}")
df_clean.head()
```

```
Dropping 16 columns → ['result_1', 'result_2', 'round_diff',
'ct_pistol_side', 't_pistol_side', 'team_2_player_1',
'team_2_player_2', 'team_2_player_3', 'team_1_player_1',
'team_2_player_4'] …
Shape after drop: (175594, 25)
```

```
         date    team_1         team_2       map  map_winner
starting_ct  \
0 2020-02-27   Rugratz        Recon 5      Nuke           2
2
1 2020-02-27  Station7  Thunder Logic  Overpass           2
1
2 2020-02-27   Oceanus         Divine     Dust2           2
2
3 2020-02-27    Mythic       Infinity     Dust2           1
1
4 2020-02-27     Chaos       Under 21      Nuke           2
2

   event_id  team_1_wr  team_2_wr  team_1_avg_wr  team_2_avg_wr
rank_1  \
0      5151   0.520000   0.595745       0.348216       0.331008
86
1      5151   0.416667   0.456790       0.360019       0.274823
132
2      5151   0.377358   0.434783       0.192609       0.374556
128
3      5151   0.434921   0.193548       0.412092       0.154695
74
4      5151   0.536313   0.833333       0.446962       0.312313
54

   rank_2  rank_diff  team_1_ct_pistol_wr  team_1_t_pistol_wr  \
0     111        -25             0.459155            0.650704
1     164        -32                  NaN                 NaN
2      99         29             0.450704            0.492958
3     143        -69             0.388466            0.416757
4     180       -126             0.518293            0.550610

   team_2_ct_pistol_wr  team_2_t_pistol_wr  team_1_map_wr
team_2_map_wr  \
0             0.645833            0.593750       0.712871
0.452381
1             0.509317            0.472050            NaN
0.422222
2             0.398190            0.438914       0.409091
1.000000
3             0.447917            0.447917       0.475410
0.272727
4                  NaN                 NaN       0.385714
NaN

   team_1_avg_adr  team_2_avg_adr  picked_by_team_1  team_1_recent_wr
\
0           64.36           82.48                 0               0.0
```

```
1         54.80         95.40                  0             0.4
2         72.62         79.10                  0             0.4
3         88.30         66.60                  0             0.6
4         68.90         79.80                  0             1.0

   team_2_recent_wr
0               0.2
1               0.0
2               0.8
3               0.0
4               NaN
```

1 · Impute Missing Values + Add “Was-Missing” Flags

Below is a concise playbook (decision table + ready-to-run code) that cleans every NaN **without leaking future info** and exposes the fact that it *was* missing—often predictive in its own right.

Feature family	Why it goes NaN	Fill-value	Add flag?	Notes
Win-rate %`team_*_wr`, `team_*_map_wr`, `team_*_pistol_wr`	<10 recorded maps (low history)	**League median** (~0.50)	**Yes** `*_nan`	Median avoids bias; flags mark low experience
Recent win-rate (`team_*_recent_wr`)	<5 prior maps	League median (0.50)	**Yes**	Hot-/cold streak missingness is itself signal
Average ADR (`team_*_avg_adr`)	Scraper gap / new player	ADR median (~75)	**Yes**	Tight range; median is robust
HLTV ranks (`rank_*`)	Team unranked (>300)	Sentinel **400**	**Yes** `rank_*_nan`	Then bucket or log-scale ranks
Map-specific WR	Never played that map	**That team’s**	**Yes**	Persona

Feature family	Why it goes NaN	Fill-value	Add flag?	Notes
missing but overall WR present		**overall WR** → else league median		lised fallback before global
Categoricals (`map`, `starting_ct`, `picked_by_team_1`)	Rare data errors	Most-frequent label	Optional	One-hot will treat extra flag as new column

```
# %% [code]  — Impute + flags
import numpy as np
import pandas as pd

df_imp = df_clean.copy()

# 1. Define groups
perc_cols   = [c for c in df_imp if c.endswith(("_wr", "_adr"))]
rank_cols   = ["rank_1", "rank_2"]
recent_cols = [c for c in df_imp if c.endswith("_recent_wr")]
cat_cols    = ["map", "starting_ct", "picked_by_team_1"]   # minor
NaNs expected

# 2. Percentage + ADR stats
---------------------------------------------
league_meds = df_imp[perc_cols].median()
for c in perc_cols:
    df_imp[c + "_nan"] = df_imp[c].isna().astype(int)
    df_imp[c] = df_imp[c].fillna(league_meds[c])

# 3. Map-WR fallback to overall team WR
----------------------------------
for side in ["1", "2"]:
    map_wr = f"team_{side}_map_wr"
    team_wr = f"team_{side}_wr"
    mask = df_imp[map_wr].isna() & df_imp[team_wr].notna()
    df_imp.loc[mask, map_wr] = df_imp.loc[mask, team_wr]

# 4. Ranks
---------------------------------------------------------------
for c in rank_cols:
    df_imp[c + "_nan"] = df_imp[c].isna().astype(int)
    df_imp[c] = df_imp[c].fillna(400)     # sentinel
    # Create tier buckets 0–4
    bins = [0, 10, 30, 100, 300, np.inf]
    df_imp[c + "_tier"] = pd.cut(df_imp[c], bins=bins, labels=False,
right=True)
```

```python
# 5. Categoricals
---------------------------------------------------------
for c in cat_cols:
    if df_imp[c].isna().any():
        df_imp[c + "_nan"] = df_imp[c].isna().astype(int)
        mode_val = df_imp[c].mode(dropna=True).iat[0]
        df_imp[c] = df_imp[c].fillna(mode_val)

# Quick sanity check
print("Remaining NaNs:", df_imp.isna().sum().sum())
```

```
Remaining NaNs: 0
```

2 · Feature Engineering

We've got a squeaky-clean `df_imp` (NaNs handled, flags added).
Now we **create new columns that carry more signal than the raw stats alone** and prep everything for the model.

Core ideas

Category	What we'll build	Why it helps
"Vs" gaps	`adr_diff`, `wr_diff`, `map_wr_gap`, `recent_diff`	Turns two separate numbers into a single *who-is-better* metric → clearer for the model.
Rank buckets	`rank_*_tier` already made; we'll add `rank_diff_tier`	Compresses heavy-tailed integer ranks into discrete strength bands.
Side-/pick-aware flags	`picked_by_team_1 × starting_ct`, `map × starting_ct`	Captures maps that are CT-favoured and whether T1 starts CT.
Experience flags	Existing `*_nan` columns	Missing history is itself predictive; keep them.
One-hot / native cats	`map`, `starting_ct`, `picked_by_team_1`	Needed so the model treats categories separately.
Date context (optional)	`year`, `month`, or rolling event index	Lets the model learn meta shifts (new patches, map pool changes).

```python
import pandas as pd
import numpy as np

df_feat = df_imp.copy()
```

```
# 1.  Simple “who’s better” gaps ----------------------------
df_feat["adr_diff"]      = df_feat["team_1_avg_adr"] -
df_feat["team_2_avg_adr"]
df_feat["wr_diff"]       = df_feat["team_1_wr"]      -
df_feat["team_2_wr"]
df_feat["map_wr_gap"]    = df_feat["team_1_map_wr"]  -
df_feat["team_2_map_wr"]
df_feat["recent_diff"]   = df_feat["team_1_recent_wr"] -
df_feat["team_2_recent_wr"]

# 2.  Rank bucket gap (tier already built during imputation)
df_feat["rank_diff_tier"] = df_feat["rank_1_tier"] -
df_feat["rank_2_tier"]

# 3.  Side-bias interactions -------------------------------
df_feat["t1_ct_pick"] = (
    (df_feat["picked_by_team_1"] == 1) &
    (df_feat["starting_ct"] == 1)
).astype(int)

df_feat["t1_ct_map_combo"] = (
    df_feat["map"].astype(str) + "_" +
df_feat["starting_ct"].astype(str)
)

# 4.  Optional date features -------------------------------
df_feat["year"]  = df_feat["date"].dt.year
df_feat["month"] = df_feat["date"].dt.month

# 5.  One-hot encode main categoricals ----------------------
cat_cols = ["map", "starting_ct", "picked_by_team_1",
"t1_ct_map_combo"]
df_model = pd.get_dummies(df_feat, columns=cat_cols, drop_first=True)

# 6.  Target & feature split -------------------------------
y = df_model["map_winner"]
X = df_model.drop(columns=["map_winner", "date"])   # keep date only
for time-based CV

print("Final feature matrix shape:", X.shape)
```

```
Final feature matrix shape: (175594, 76)
```

```
print(df_feat.head())
```

```
        date    team_1         team_2       map  map_winner
starting_ct  \
0 2020-02-27   Rugratz        Recon 5      Nuke           2
2
```

```
1 2020-02-27  Station7  Thunder Logic  Overpass          2
1
2 2020-02-27   Oceanus         Divine     Dust2          2
2
3 2020-02-27    Mythic       Infinity     Dust2          1
1
4 2020-02-27     Chaos       Under 21      Nuke          2
2

   event_id  team_1_wr  team_2_wr  team_1_avg_wr  team_2_avg_wr
rank_1  \
0      5151   0.520000   0.595745       0.348216       0.331008
86
1      5151   0.416667   0.456790       0.360019       0.274823
132
2      5151   0.377358   0.434783       0.192609       0.374556
128
3      5151   0.434921   0.193548       0.412092       0.154695
74
4      5151   0.536313   0.833333       0.446962       0.312313
54

   rank_2  rank_diff  team_1_ct_pistol_wr  team_1_t_pistol_wr  \
0     111        -25             0.459155            0.650704
1     164        -32             0.446707            0.449466
2      99         29             0.450704            0.492958
3     143        -69             0.388466            0.416757
4     180       -126             0.518293            0.550610

   team_2_ct_pistol_wr  team_2_t_pistol_wr  team_1_map_wr
team_2_map_wr  \
0             0.645833            0.593750       0.712871
0.452381
1             0.509317            0.472050       0.529880
0.422222
2             0.398190            0.438914       0.409091
1.000000
3             0.447917            0.447917       0.475410
0.272727
4             0.449597            0.448674       0.385714
0.519802

   team_1_avg_adr  team_2_avg_adr  picked_by_team_1  team_1_recent_wr
\
0           64.36           82.48                 0               0.0

1           54.80           95.40                 0               0.4

2           72.62           79.10                 0               0.4
```

```
3         88.30         66.60                   0              0.6

4         68.90         79.80                   0              1.0


   team_2_recent_wr  team_1_wr_nan  team_2_wr_nan
team_1_avg_wr_nan  \
0               0.2              0              0                  0

1               0.0              0              0                  0

2               0.8              0              0                  0

3               0.0              0              0                  0

4               0.4              0              0                  0


   team_2_avg_wr_nan  team_1_ct_pistol_wr_nan  team_1_t_pistol_wr_nan
\
0                  0                        0                       0

1                  0                        1                       1

2                  0                        0                       0

3                  0                        0                       0

4                  0                        0                       0


   team_2_ct_pistol_wr_nan  team_2_t_pistol_wr_nan  team_1_map_wr_nan
\
0                        0                       0                  0

1                        0                       0                  1

2                        0                       0                  0

3                        0                       0                  0

4                        1                       1                  0


   team_2_map_wr_nan  team_1_avg_adr_nan  team_2_avg_adr_nan  \
0                  0                   0                   0
1                  0                   0                   0
2                  0                   0                   0
3                  0                   0                   0
4                  1                   0                   0
```

```
   team_1_recent_wr_nan  team_2_recent_wr_nan  rank_1_nan  rank_1_tier
\
0                     0                     0           0            2

1                     0                     0           0            3

2                     0                     0           0            3

3                     0                     0           0            2

4                     0                     1           0            2


   rank_2_nan  rank_2_tier  adr_diff   wr_diff  map_wr_gap
recent_diff  \
0           0            3    -18.12 -0.075745    0.260490          -
0.2
1           0            3    -40.60 -0.040123    0.107658
0.4
2           0            2     -6.48 -0.057424   -0.590909          -
0.4
3           0            3     21.70  0.241372    0.202683
0.6
4           0            3    -10.90 -0.297020   -0.134088
0.6

   rank_diff_tier  t1_ct_pick t1_ct_map_combo  year  month
0              -1           0          Nuke_2  2020      2
1               0           0      Overpass_1  2020      2
2               1           0         Dust2_2  2020      2
3              -1           0         Dust2_1  2020      2
4              -1           0          Nuke_2  2020      2

# %% [code] — 0. Setup
%pip install lightgbm
from lightgbm import early_stopping, log_evaluation

import pandas as pd, numpy as np, matplotlib.pyplot as plt
from sklearn.model_selection import GroupKFold
from sklearn.metrics import log_loss, roc_auc_score, accuracy_score
from lightgbm import LGBMClassifier

#
---------------------------------------------------------------------
# 1.  CLEAN, IMPUTED DATAFRAME  →  rebuild lagged recent-WR (“_safe”)
--
lookback = 5
df_imp = df_imp.sort_values("date").copy()

for side in ("1", "2"):
```

```
    win_flag = (df_imp["map_winner"] == (1 if side == "1" else
2)).astype(int)
    df_imp[f"team_{side}_recent_wr_safe"] = (
        win_flag
        .groupby(df_imp[f"team_{side}"])
        .apply(lambda s: s.shift(1)
                         .rolling(lookback, min_periods=lookback)
                         .mean())
        .reset_index(level=0, drop=True)
    )
    df_imp[f"team_{side}_recent_wr_safe_nan"] =
df_imp[f"team_{side}_recent_wr_safe"].isna().astype(int)
    df_imp[f"team_{side}_recent_wr_safe"] =
df_imp[f"team_{side}_recent_wr_safe"].fillna(0.50)

df_feat = df_imp.copy()

#
---------------------------------------------------------------------
# 2.  FEATURE ENGINEERING (gaps, flags, buckets)
----------------------
df_feat["adr_diff"]       = df_feat["team_1_avg_adr"] -
df_feat["team_2_avg_adr"]
df_feat["wr_diff"]        = df_feat["team_1_wr"]      -
df_feat["team_2_wr"]
df_feat["map_wr_gap"]     = df_feat["team_1_map_wr"]  -
df_feat["team_2_map_wr"]
df_feat["recent_diff"]    = df_feat["team_1_recent_wr_safe"] -
df_feat["team_2_recent_wr_safe"]
df_feat["rank_diff_tier"] = df_feat["rank_1_tier"]   -
df_feat["rank_2_tier"]

df_feat["t1_ct_pick"] = (
    (df_feat["picked_by_team_1"] == 1) & (df_feat["starting_ct"] == 1)
).astype(int)
df_feat["t1_ct_map_combo"] = df_feat["map"].astype(str) + "_" +
df_feat["starting_ct"].astype(str)

df_feat["year"]  = df_feat["date"].dt.year
df_feat["month"] = df_feat["date"].dt.month

#
---------------------------------------------------------------------
# 3.  ONE-HOT ENCODE CATEGORICALS
-------------------------------------
cat_cols = ["map", "starting_ct", "picked_by_team_1",
"t1_ct_map_combo"]
df_model = pd.get_dummies(df_feat, columns=cat_cols, drop_first=True)

#
```

```
# ---------------------------------------------------------------------
# 4.  TARGET & FEATURES ------------------------------------------------
TARGET = "map_winner"
DROP_ALWAYS = ["date"]
priv_cols = ["ct_pistol_side", "t_pistol_side", "result_1",
"result_2", "round_diff"]

y = df_model[TARGET]
X = df_model.drop(columns=[TARGET] + DROP_ALWAYS)
X = X.drop(columns=[c for c in ["team_1", "team_2"] if c in
X.columns])

#
---------------------------------------------------------------------
# 5.  GROUP K-FOLD WITH PRIVILEGE MASKING
-----------------------------
gkf = GroupKFold(n_splits=5)
group_col = "match_id" if "match_id" in df_feat.columns else
"event_id"
groups = df_feat[group_col]

log_losses, aucs, accuracies = [], [], []

for fold, (tr, val) in enumerate(gkf.split(X, y, groups), start=1):
    X_tr, y_tr = X.iloc[tr].copy(), y.iloc[tr]
    X_val, y_val = X.iloc[val].copy(), y.iloc[val]

    # mask privileged columns in validation
    for c in priv_cols:
        if c in X_val.columns:
            X_val[c] = np.nan
            X_tr[c + "_nan"]  = X_tr[c].isna().astype(int)
            X_val[c + "_nan"] = X_val[c].isna().astype(int)

    model = LGBMClassifier(
        objective="binary",
        n_estimators=2000,
        learning_rate=0.03,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42,
        metric="binary_logloss"
    )

    model.fit(
        X_tr, y_tr,
        eval_set=[(X_val, y_val)],
        eval_metric="binary_logloss",
        callbacks=[
            early_stopping(stopping_rounds=100),
```

```python
            log_evaluation(period=0)
        ]
    )

    val_probs = model.predict_proba(X_val)[:, 1]          # P(label ==
2)
    val_preds = np.where(val_probs >= 0.5, 2, 1)          # 2 or 1
    accuracy  = accuracy_score(y_val, val_preds)

    log_losses.append(log_loss(y_val, val_probs))
    aucs.append(roc_auc_score(y_val, val_probs))
    accuracies.append(accuracy_score(y_val, val_preds))

    print(f"Fold {fold}:  log-loss {log_losses[-1]:.4f} | "
          f"AUC {aucs[-1]:.3f} | Accuracy {accuracies[-1]:.4f}")

print(f"\nCV mean log-loss  {np.mean(log_losses):.4f}")
print(f"CV mean ROC-AUC   {np.mean(aucs):.3f}")
print(f"CV mean Accuracy  {np.mean(accuracies):.4f}")
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: C:\Users\zachj\AppData\Local\Microsoft\
WindowsApps\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\
python.exe -m pip install --upgrade pip

Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: lightgbm in c:\users\zachj\appdata\
local\packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\
localcache\local-packages\python313\site-packages (4.6.0)
Requirement already satisfied: numpy>=1.17.0 in c:\users\zachj\
appdata\local\packages\
pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\localcache\local-
packages\python313\site-packages (from lightgbm) (2.2.4)
Requirement already satisfied: scipy in c:\users\zachj\appdata\local\
packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\
localcache\local-packages\python313\site-packages (from lightgbm)
(1.15.2)
Note: you may need to restart the kernel to use updated packages.
[LightGBM] [Info] Number of positive: 65807, number of negative: 74668
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.006112 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4998
[LightGBM] [Info] Number of data points in the train set: 140475,
number of used features: 75
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.468461 ->
initscore=-0.126325
```

```
[LightGBM] [Info] Start training from score -0.126325
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[296]	valid_0's binary_logloss: 0.344
Fold 1:  log-loss 0.3440 | AUC 0.926 | Accuracy 0.8428
[LightGBM] [Info] Number of positive: 66039, number of negative: 74436
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.005363 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 5002
[LightGBM] [Info] Number of data points in the train set: 140475,
number of used features: 75
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.470112 ->
initscore=-0.119694
[LightGBM] [Info] Start training from score -0.119694
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[529]	valid_0's binary_logloss: 0.359459
Fold 2:  log-loss 0.3595 | AUC 0.918 | Accuracy 0.8315
[LightGBM] [Info] Number of positive: 66109, number of negative: 74366
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.005308 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4996
[LightGBM] [Info] Number of data points in the train set: 140475,
number of used features: 75
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.470610 ->
initscore=-0.117694
[LightGBM] [Info] Start training from score -0.117694
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[464]	valid_0's binary_logloss: 0.359061
Fold 3:  log-loss 0.3591 | AUC 0.919 | Accuracy 0.8356
[LightGBM] [Info] Number of positive: 66095, number of negative: 74380
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.005642 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4992
[LightGBM] [Info] Number of data points in the train set: 140475,
number of used features: 74
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.470511 ->
initscore=-0.118094
[LightGBM] [Info] Start training from score -0.118094
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[316]	valid_0's binary_logloss: 0.358997
```

```
Fold 4:  log-loss 0.3590 | AUC 0.920 | Accuracy 0.8383
[LightGBM] [Info] Number of positive: 66394, number of negative: 74082
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.005141 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4991
[LightGBM] [Info] Number of data points in the train set: 140476,
number of used features: 75
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.472636 ->
initscore=-0.109566
[LightGBM] [Info] Start training from score -0.109566
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[374] valid_0's binary_logloss: 0.345431
Fold 5:  log-loss 0.3454 | AUC 0.925 | Accuracy 0.8419

CV mean log-loss  0.3534
CV mean ROC-AUC   0.922
CV mean Accuracy  0.8380
```

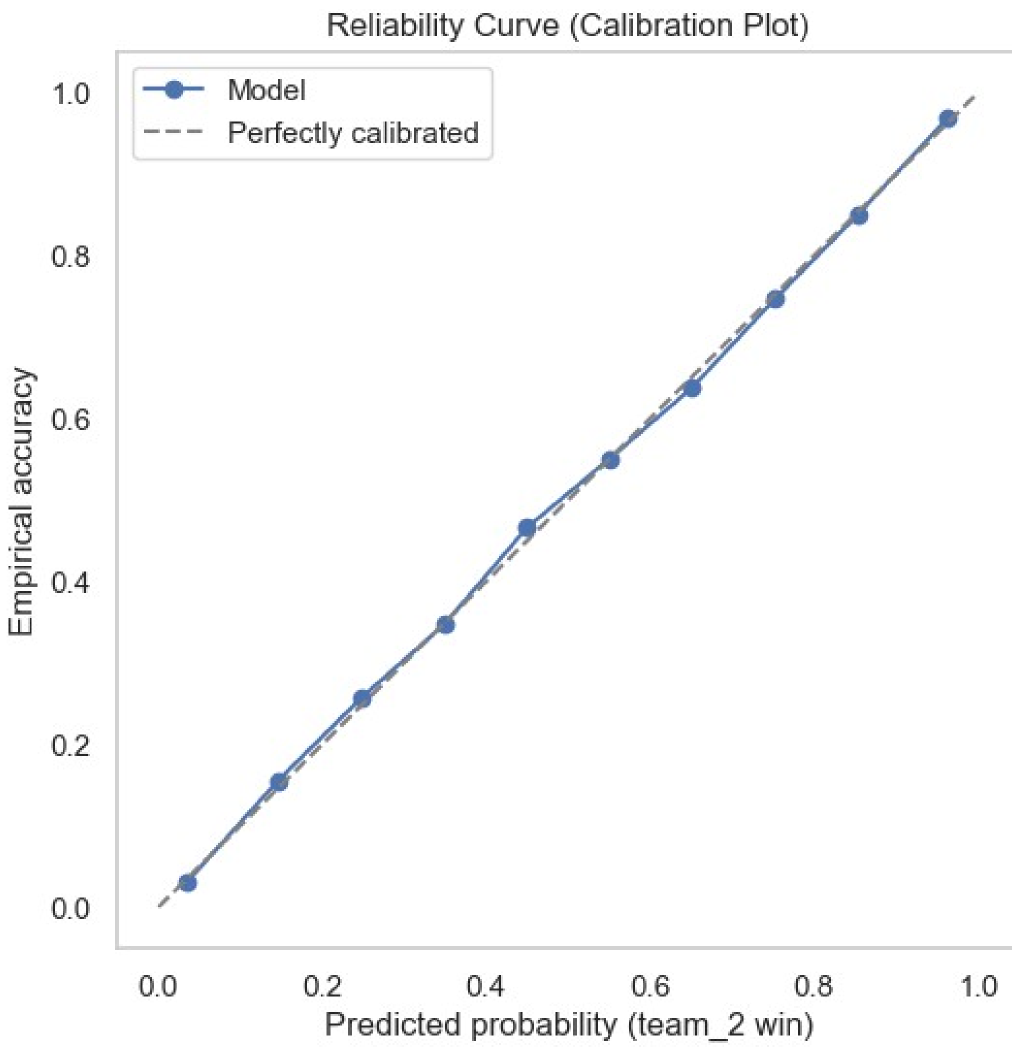
```
from sklearn.calibration import calibration_curve
from sklearn.metrics import brier_score_loss

# Store all predictions and ground truths
all_val_probs = []
all_val_true = []

for fold, (tr, val) in enumerate(gkf.split(X, y, groups), start=1):
    X_tr, y_tr = X.iloc[tr].copy(), y.iloc[tr]
    X_val, y_val = X.iloc[val].copy(), y.iloc[val]

    for c in priv_cols:
        if c in X_val.columns:
            X_val[c] = np.nan
            X_tr[c + "_nan"]  = X_tr[c].isna().astype(int)
            X_val[c + "_nan"] = X_val[c].isna().astype(int)

    model = LGBMClassifier(
        objective="binary",
        n_estimators=2000,
        learning_rate=0.03,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42,
        metric="binary_logloss"
    )

    model.fit(
        X_tr, y_tr,
```

```
        eval_set=[(X_val, y_val)],
        eval_metric="binary_logloss",
        callbacks=[early_stopping(stopping_rounds=100),
log_evaluation(period=0)]
    )

    probs = model.predict_proba(X_val)[:, 1]
    all_val_probs.extend(probs)
    all_val_true.extend(y_val == 2)  # Convert to 0/1: True if winner
== team_2

# Calibration curve
prob_true, prob_pred = calibration_curve(all_val_true, all_val_probs,
n_bins=10)

plt.figure(figsize=(6, 6))
plt.plot(prob_pred, prob_true, marker='o', label='Model')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray',
label='Perfectly calibrated')
plt.xlabel('Predicted probability (team_2 win)')
plt.ylabel('Empirical accuracy')
plt.title('Reliability Curve (Calibration Plot)')
plt.legend()
plt.grid()
plt.show()

# Brier score
brier = brier_score_loss(all_val_true, all_val_probs)
print(f"Brier score: {brier:.4f}")
```

```
[LightGBM] [Info] Number of positive: 65807, number of negative: 74668
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.006260 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4998
[LightGBM] [Info] Number of data points in the train set: 140475,
number of used features: 75
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.468461 ->
initscore=-0.126325
[LightGBM] [Info] Start training from score -0.126325
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[296] valid_0's binary_logloss: 0.344
[LightGBM] [Info] Number of positive: 66039, number of negative: 74436
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.005613 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 5002
```

```
[LightGBM] [Info] Number of data points in the train set: 140475,
number of used features: 75
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.470112 ->
initscore=-0.119694
[LightGBM] [Info] Start training from score -0.119694
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[529]	valid_0's binary_logloss: 0.359459
[LightGBM] [Info] Number of positive: 66109, number of negative: 74366
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.005420 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4996
[LightGBM] [Info] Number of data points in the train set: 140475,
number of used features: 75
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.470610 ->
initscore=-0.117694
[LightGBM] [Info] Start training from score -0.117694
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[464]	valid_0's binary_logloss: 0.359061
[LightGBM] [Info] Number of positive: 66095, number of negative: 74380
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.005514 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4992
[LightGBM] [Info] Number of data points in the train set: 140475,
number of used features: 74
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.470511 ->
initscore=-0.118094
[LightGBM] [Info] Start training from score -0.118094
Training until validation scores don't improve for 100 rounds
Early stopping, best iteration is:
[316]	valid_0's binary_logloss: 0.358997
[LightGBM] [Info] Number of positive: 66394, number of negative: 74082
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.005306 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4991
[LightGBM] [Info] Number of data points in the train set: 140476,
number of used features: 75
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.472636 ->
initscore=-0.109566
[LightGBM] [Info] Start training from score -0.109566
Training until validation scores don't improve for 100 rounds
```

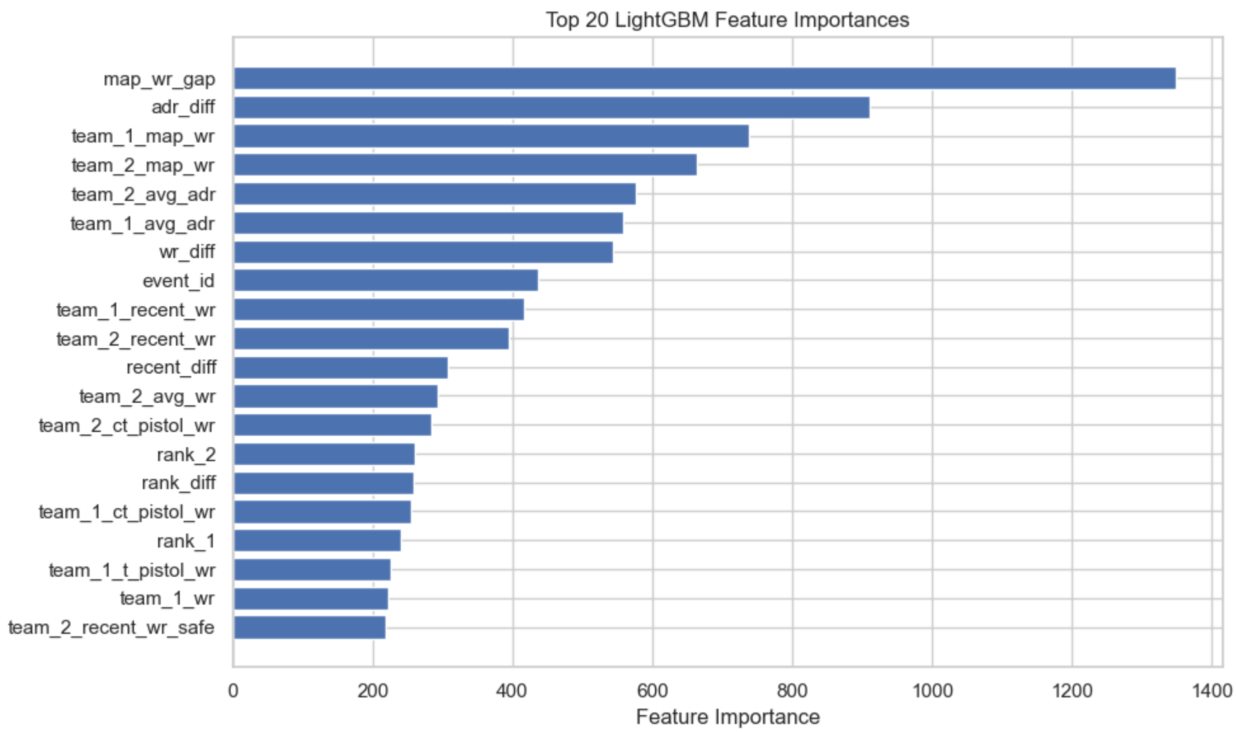
```
Early stopping, best iteration is:
[374]	valid_0's binary_logloss: 0.345431
```




```
Brier score: 0.1128
```

```
importances = model.feature_importances_
feature_names = X.columns
importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': importances
}).sort_values(by='importance', ascending=False)

# Display top 20
plt.figure(figsize=(10, 6))
plt.barh(importance_df.head(20).iloc[::-1]['feature'],
importance_df.head(20).iloc[::-1]['importance'])
plt.xlabel("Feature Importance")
```

```
plt.title("Top 20 LightGBM Feature Importances")
plt.tight_layout()
plt.show()
```



CATBOOST

```
# %% [code] — CatBoost implementation (leakage-safe GroupKFold)
%pip install catboost --quiet

import pandas as pd, numpy as np
from catboost import CatBoostClassifier, Pool
from sklearn.model_selection import GroupKFold
from sklearn.metrics import log_loss, roc_auc_score, accuracy_score,
brier_score_loss

# ------------------------------------------------------------------
# X  and df_feat already exist from previous LightGBM cell
# y is still 1 (team-1 win) / 2 (team-2 win)
# Convert to binary 0 / 1 for CatBoost
y_bin = (y == 2).astype(int)           # 1 == team-2 win  (positive
class)

# ------------------------------------------------------------------
PRIV_COLS = ["ct_pistol_side", "t_pistol_side", "result_1",
"result_2", "round_diff"]

gkf        = GroupKFold(n_splits=5)
```

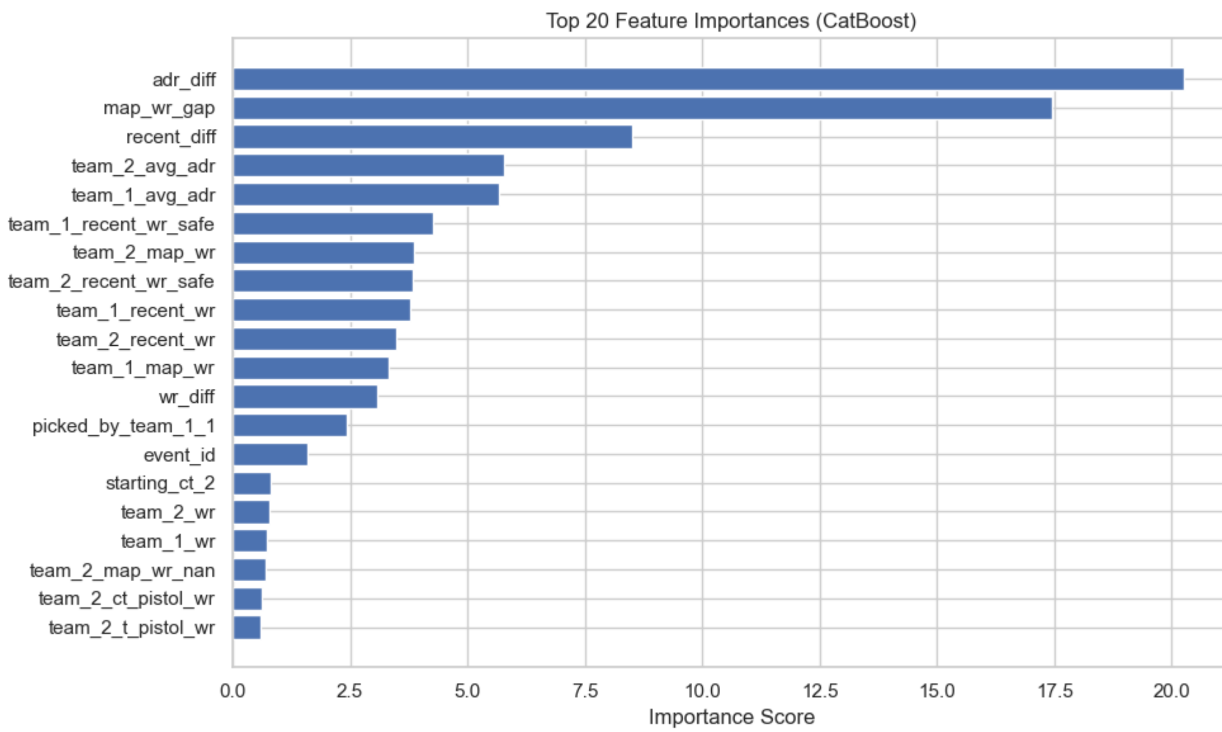
```python
group_col  = "match_id" if "match_id" in df_feat.columns else
"event_id"
groups     = df_feat[group_col]

loglosses, aucs, accs, briers = [], [], [], []

for fold, (tr_idx, val_idx) in enumerate(gkf.split(X, y_bin, groups),
1):
    X_tr, y_tr = X.iloc[tr_idx].copy(), y_bin.iloc[tr_idx]
    X_val, y_val = X.iloc[val_idx].copy(), y_bin.iloc[val_idx]

    # --- mask privileged cols in VALID set -------------------------
    for c in PRIV_COLS:
        if c in X_val.columns:
            X_val[c] = np.nan
            X_tr[c + "_nan"]  = X_tr[c].isna().astype(int)
            X_val[c + "_nan"] = X_val[c].isna().astype(int)

    train_pool = Pool(X_tr, y_tr)
    val_pool   = Pool(X_val, y_val)

    model = CatBoostClassifier(
        iterations=1500,
        learning_rate=0.03,
        depth=6,
        loss_function="Logloss",
        eval_metric="Logloss",
        random_seed=42,
        verbose=False,
        early_stopping_rounds=100
    )

    model.fit(train_pool, eval_set=val_pool, use_best_model=True)

    val_probs = model.predict_proba(val_pool)[:, 1]          #
P(team-2 win)
    val_preds = np.where(val_probs >= 0.5, 1, 0)             # binary
labels

    loglosses.append(log_loss(y_val, val_probs))
    aucs.append(roc_auc_score(y_val, val_probs))
    accs.append(accuracy_score(y_val, val_preds))
    briers.append(brier_score_loss(y_val, val_probs))

    print(f"Fold {fold}: log-loss {loglosses[-1]:.4f} | "
          f"AUC {aucs[-1]:.3f} | Accuracy {accs[-1]:.4f} | "
          f"Brier {briers[-1]:.4f}")

print(f"\nCV mean log-loss  {np.mean(loglosses):.4f}")
print(f"CV mean ROC-AUC   {np.mean(aucs):.3f}")
```

```
print(f"CV mean Accuracy  {np.mean(accs):.4f}")
print(f"CV mean Brier     {np.mean(briers):.4f}")
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: C:\Users\zachj\AppData\Local\Microsoft\
WindowsApps\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\
python.exe -m pip install --upgrade pip

Note: you may need to restart the kernel to use updated packages.
Fold 1: log-loss 0.3396 | AUC 0.928 | Accuracy 0.8455 | Brier 0.1082
Fold 2: log-loss 0.3543 | AUC 0.921 | Accuracy 0.8368 | Brier 0.1135
Fold 3: log-loss 0.3564 | AUC 0.920 | Accuracy 0.8357 | Brier 0.1139
Fold 4: log-loss 0.3564 | AUC 0.921 | Accuracy 0.8395 | Brier 0.1132
Fold 5: log-loss 0.3447 | AUC 0.925 | Accuracy 0.8437 | Brier 0.1100

CV mean log-loss  0.3503
CV mean ROC-AUC   0.923
CV mean Accuracy  0.8402
CV mean Brier     0.1118
```

```
from catboost import Pool, CatBoostClassifier, cv, CatBoost
import matplotlib.pyplot as plt

# Create a Pool with full data (retrain on all data if needed)
full_pool = Pool(X, y)

# Retrain the final model on all data (optional, skip if model already
trained)
final_model = CatBoostClassifier(
    iterations=500,
    learning_rate=0.03,
    depth=6,
    loss_function='Logloss',
    eval_metric='AUC',
    random_seed=42,
    verbose=0
)
final_model.fit(X, y)

# Get feature importances
importances = final_model.get_feature_importance(full_pool)
feat_names = X.columns

# Plot top 20
sorted_idx = np.argsort(importances)[::-1][:20]
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), np.array(importances)[sorted_idx][::-
1])
plt.yticks(range(len(sorted_idx)), np.array(feat_names)[sorted_idx]
[::-1])
```

```
plt.xlabel("Importance Score")
plt.title("Top 20 Feature Importances (CatBoost)")
plt.tight_layout()
plt.show()
```

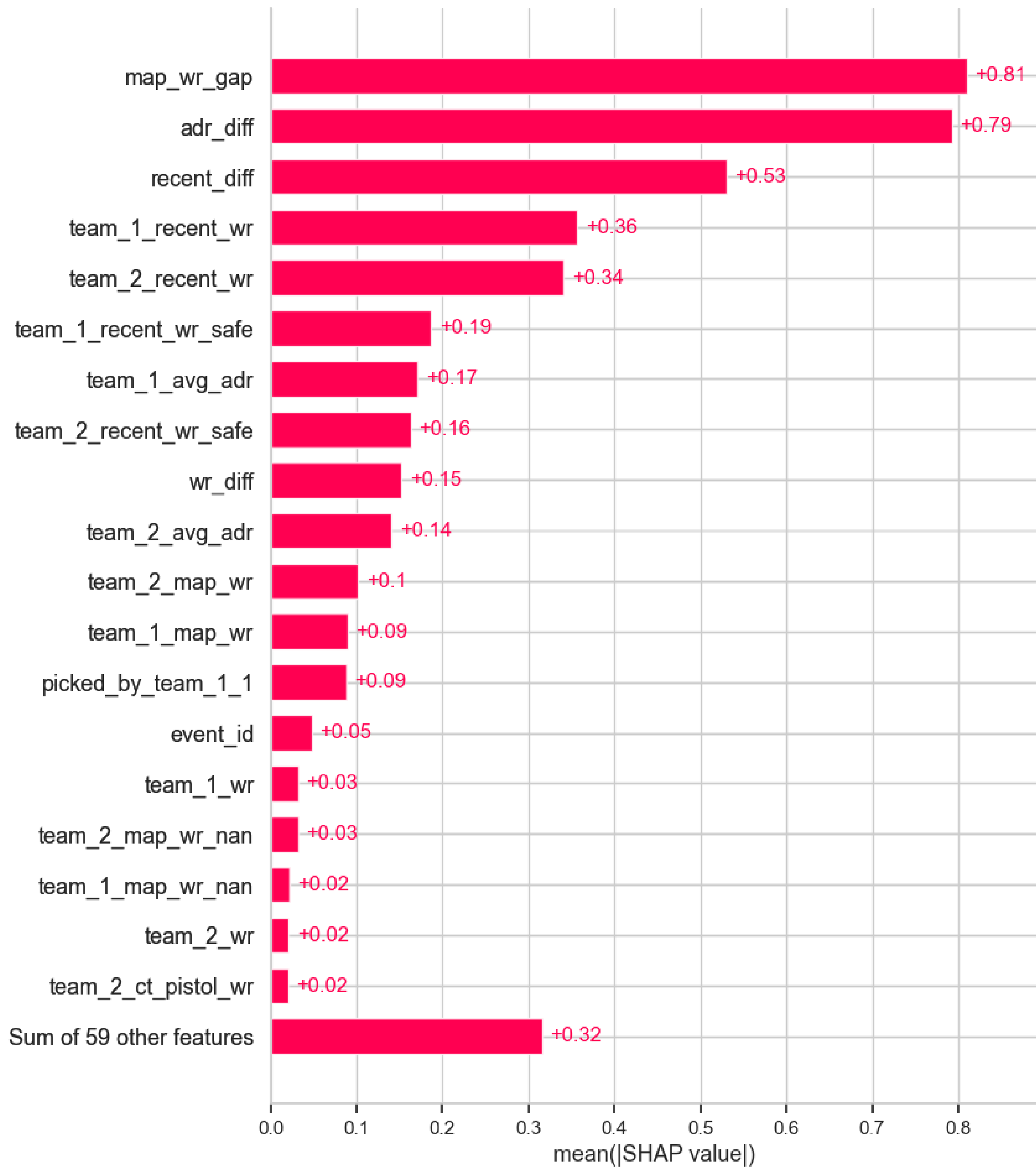



```
%pip install shap
import shap
import matplotlib.pyplot as plt

# Only use a sample for performance
X_sample = X_val.sample(n=1000, random_state=42)

# Use TreeExplainer for CatBoost
explainer = shap.Explainer(model)
shap_values = explainer(X_sample)

# SHAP summary plot (bar)
shap.plots.bar(shap_values, max_display=20)

Defaulting to user installation because normal site-packages is not
writeable
Collecting shap
  Downloading shap-0.47.2.tar.gz (2.6 MB)
     ---------------------------------------- 0.0/2.6 MB ? eta -:--:--
     ---------------------------------------- 2.6/2.6 MB 21.9 MB/s eta
0:00:00
  Installing build dependencies: started
```

```
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'done'
  Preparing metadata (pyproject.toml): started
  Preparing metadata (pyproject.toml): finished with status 'done'
Requirement already satisfied: numpy in c:\users\zachj\appdata\local\
packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\
localcache\local-packages\python313\site-packages (from shap) (2.2.4)
Requirement already satisfied: scipy in c:\users\zachj\appdata\local\
packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\
localcache\local-packages\python313\site-packages (from shap) (1.15.2)
Requirement already satisfied: scikit-learn in c:\users\zachj\appdata\
local\packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\
localcache\local-packages\python313\site-packages (from shap) (1.6.1)
Requirement already satisfied: pandas in c:\users\zachj\appdata\local\
packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\
localcache\local-packages\python313\site-packages (from shap) (2.2.3)
Collecting tqdm>=4.27.0 (from shap)
  Downloading tqdm-4.67.1-py3-none-any.whl.metadata (57 kB)
Requirement already satisfied: packaging>20.9 in c:\users\zachj\
appdata\local\packages\
pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\localcache\local-
packages\python313\site-packages (from shap) (24.2)
Collecting slicer==0.0.8 (from shap)
  Downloading slicer-0.0.8-py3-none-any.whl.metadata (4.0 kB)
Collecting numba>=0.54 (from shap)
  Downloading numba-0.61.2-cp313-cp313-win_amd64.whl.metadata (2.8 kB)
Collecting cloudpickle (from shap)
  Downloading cloudpickle-3.1.1-py3-none-any.whl.metadata (7.1 kB)
Collecting typing-extensions (from shap)
  Downloading typing_extensions-4.13.2-py3-none-any.whl.metadata (3.0
kB)
Collecting llvmlite<0.45,>=0.44.0dev0 (from numba>=0.54->shap)
  Downloading llvmlite-0.44.0-cp313-cp313-win_amd64.whl.metadata (5.0
kB)
Requirement already satisfied: colorama in c:\users\zachj\appdata\
local\packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\
localcache\local-packages\python313\site-packages (from tqdm>=4.27.0-
>shap) (0.4.6)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\
zachj\appdata\local\packages\
pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\localcache\local-
packages\python313\site-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\zachj\appdata\
local\packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\
localcache\local-packages\python313\site-packages (from pandas->shap)
(2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\zachj\
appdata\local\packages\
```

```
pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\localcache\local-
packages\python313\site-packages (from pandas->shap) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in c:\users\zachj\
appdata\local\packages\
pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\localcache\local-
packages\python313\site-packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\zachj\
appdata\local\packages\
pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\localcache\local-
packages\python313\site-packages (from scikit-learn->shap) (3.6.0)
Requirement already satisfied: six>=1.5 in c:\users\zachj\appdata\
local\packages\pythonsoftwarefoundation.python.3.13_qbz5n2kfra8p0\
localcache\local-packages\python313\site-packages (from python-
dateutil>=2.8.2->pandas->shap) (1.17.0)
Downloading slicer-0.0.8-py3-none-any.whl (15 kB)
Downloading numba-0.61.2-cp313-cp313-win_amd64.whl (2.8 MB)
   ---------------------------------------- 0.0/2.8 MB ? eta -:--:--
   ---------------------------------------- 2.8/2.8 MB 50.0 MB/s eta
0:00:00
Downloading tqdm-4.67.1-py3-none-any.whl (78 kB)
Downloading cloudpickle-3.1.1-py3-none-any.whl (20 kB)
Downloading typing_extensions-4.13.2-py3-none-any.whl (45 kB)
Downloading llvmlite-0.44.0-cp313-cp313-win_amd64.whl (30.3 MB)
   ---------------------------------------- 0.0/30.3 MB ? eta -:--:--
   ----------- ---------------------------- 8.4/30.3 MB 39.9 MB/s eta
0:00:01
   -------------------- ------------------- 15.5/30.3 MB 37.4 MB/s eta
0:00:01
   --------------------------------- ------ 24.9/30.3 MB 40.5 MB/s eta
0:00:01
   ---------------------------------------  30.1/30.3 MB 40.1 MB/s eta
0:00:01
   ---------------------------------------- 30.3/30.3 MB 36.3 MB/s eta
0:00:00
Building wheels for collected packages: shap
  Building wheel for shap (pyproject.toml): started
  Building wheel for shap (pyproject.toml): finished with status
'done'
  Created wheel for shap: filename=shap-0.47.2-cp313-cp313-
win_amd64.whl size=542672
sha256=9304d6229ccb33530d900e9ac4717a3421ea0ea4238ddbdebc9eeb3232077ae
b
  Stored in directory: c:\users\zachj\appdata\local\pip\cache\wheels\
e2\dd\cb\7e03548687d1c474ee794d615c7747b9d5c79f3519d817dcbb
Successfully built shap
Installing collected packages: typing-extensions, tqdm, slicer,
llvmlite, cloudpickle, numba, shap
Successfully installed cloudpickle-3.1.1 llvmlite-0.44.0 numba-0.61.2
```

```
shap-0.47.2 slicer-0.0.8 tqdm-4.67.1 typing-extensions-4.13.2
Note: you may need to restart the kernel to use updated packages.


[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: C:\Users\zachj\AppData\Local\Microsoft\
WindowsApps\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\
python.exe -m pip install --upgrade pip
C:\Users\zachj\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\LocalCache\local-
packages\Python313\site-packages\tqdm\auto.py:21: TqdmWarning:
IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```




```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import log_loss, roc_auc_score, accuracy_score,
brier_score_loss
from sklearn.model_selection import GroupKFold
import numpy as np
```

```python
# Recode target: 1 = team_1 win -> 0, 2 = team_2 win -> 1
y = (df_model["map_winner"] == 2).astype(int)
X = df_model.drop(columns=["map_winner", "date"] + [c for c in
["team_1", "team_2"] if c in df_model.columns])

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Prepare GroupKFold
gkf = GroupKFold(n_splits=5)
groups = df_feat["match_id"] if "match_id" in df_feat.columns else
df_feat["event_id"]

log_losses, aucs, accuracies, briers = [], [], [], []

for fold, (tr, val) in enumerate(gkf.split(X_scaled, y, groups),
start=1):
    X_tr, X_val = X_scaled[tr], X_scaled[val]
    y_tr, y_val = y.iloc[tr], y.iloc[val]

    model = LogisticRegression(solver="liblinear", penalty="l2",
random_state=42)
    model.fit(X_tr, y_tr)

    val_probs = model.predict_proba(X_val)[:, 1]
    val_preds = (val_probs >= 0.5).astype(int)

    log_losses.append(log_loss(y_val, val_probs))
    aucs.append(roc_auc_score(y_val, val_probs))
    accuracies.append(accuracy_score(y_val, val_preds))
    briers.append(brier_score_loss(y_val, val_probs))

    print(f"Fold {fold}: log-loss {log_losses[-1]:.4f} | "
          f"AUC {aucs[-1]:.3f} | Accuracy {accuracies[-1]:.4f} | Brier
{briers[-1]:.4f}")

print(f"\nCV mean log-loss  {np.mean(log_losses):.4f}")
print(f"CV mean ROC-AUC   {np.mean(aucs):.3f}")
print(f"CV mean Accuracy  {np.mean(accuracies):.4f}")
print(f"CV mean Brier     {np.mean(briers):.4f}")
```

```
Fold 1: log-loss 0.3641 | AUC 0.917 | Accuracy 0.8339 | Brier 0.1159
Fold 2: log-loss 0.3805 | AUC 0.909 | Accuracy 0.8256 | Brier 0.1217
Fold 3: log-loss 0.3802 | AUC 0.909 | Accuracy 0.8276 | Brier 0.1215
Fold 4: log-loss 0.3823 | AUC 0.908 | Accuracy 0.8285 | Brier 0.1216
Fold 5: log-loss 0.3703 | AUC 0.914 | Accuracy 0.8312 | Brier 0.1181

CV mean log-loss  0.3755
CV mean ROC-AUC   0.911
```

```
CV mean Accuracy  0.8294
CV mean Brier     0.1198
```

```
# %% [code] — PyTorch MLP with GroupKFold (leakage-safe)
%pip install torch scikit-learn --quiet

import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader

import numpy as np
from sklearn.model_selection import GroupKFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import log_loss, roc_auc_score, accuracy_score,
brier_score_loss

# ─── 1.  Prepare data
-------------------------------------------------
y_bin = (df_model["map_winner"] == 2).astype(int).values      # 1 =
team-2 win
X_mat = df_model.drop(columns=["map_winner", "date", "team_1",
"team_2"]).values

scaler = StandardScaler().fit(X_mat)
X_scaled = scaler.transform(X_mat).astype(np.float32)

groups = df_feat["match_id"] if "match_id" in df_feat.columns else
df_feat["event_id"]

# ─── 2.  Simple MLP definition
---------------------------------------
class MLP(nn.Module):
    def __init__(self, n_in):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(n_in, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 1),
            nn.Sigmoid()
        )
    def forward(self, x): return self.net(x)

# ─── 3.  Cross-validation -------------------------------------------
gkf = GroupKFold(n_splits=5)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
log_losses, aucs, accs, briers = [], [], [], []

for fold, (tr, val) in enumerate(gkf.split(X_scaled, y_bin, groups),
1):
    X_tr, X_val = X_scaled[tr], X_scaled[val]
    y_tr, y_val = y_bin[tr], y_bin[val]

    train_ds = TensorDataset(torch.tensor(X_tr), torch.tensor(y_tr,
dtype=torch.float32))
    val_ds   = TensorDataset(torch.tensor(X_val), torch.tensor(y_val,
dtype=torch.float32))

    train_loader = DataLoader(train_ds, batch_size=512, shuffle=True)
    val_loader   = DataLoader(val_ds, batch_size=1024)

    model = MLP(X_scaled.shape[1]).to(device)
    opt   = torch.optim.Adam(model.parameters(), lr=1e-3)
    criterion = nn.BCELoss()

    best_val_loss = np.inf
    patience, patience_cnt = 5, 0

    for epoch in range(30):              # max epochs
        model.train()
        for xb, yb in train_loader:
            xb, yb = xb.to(device), yb.to(device)
            opt.zero_grad()
            preds = model(xb).squeeze()
            loss  = criterion(preds, yb)
            loss.backward()
            opt.step()

        # --- validation ---
        model.eval()
        with torch.no_grad():
            v_preds = torch.cat([model(x.to(device)).squeeze() for x,
_ in val_loader]).cpu().numpy()
        v_loss = log_loss(y_val, v_preds)
        if v_loss < best_val_loss - 1e-4:
            best_val_loss = v_loss
            patience_cnt  = 0
        else:
            patience_cnt += 1
        if patience_cnt >= patience:     # early stopping
            break

    # Metrics
    val_probs = v_preds
    val_preds = (val_probs >= 0.5).astype(int)
```

```
    log_losses.append(best_val_loss)
    aucs.append(roc_auc_score(y_val, val_probs))
    accs.append(accuracy_score(y_val, val_preds))
    briers.append(brier_score_loss(y_val, val_probs))

    print(f"Fold {fold}: log-loss {log_losses[-1]:.4f} | "
          f"AUC {aucs[-1]:.3f} | Acc {accs[-1]:.4f} | Brier {briers[-
1]:.4f}")

print(f"\nCV mean log-loss  {np.mean(log_losses):.4f}")
print(f"CV mean ROC-AUC   {np.mean(aucs):.3f}")
print(f"CV mean Accuracy  {np.mean(accs):.4f}")
print(f"CV mean Brier     {np.mean(briers):.4f}")
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: C:\Users\zachj\AppData\Local\Microsoft\
WindowsApps\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\
python.exe -m pip install --upgrade pip

Note: you may need to restart the kernel to use updated packages.
Fold 1: log-loss 0.3578 | AUC 0.919 | Acc 0.8358 | Brier 0.1145
Fold 2: log-loss 0.3718 | AUC 0.913 | Acc 0.8277 | Brier 0.1192
Fold 3: log-loss 0.3736 | AUC 0.912 | Acc 0.8316 | Brier 0.1195
Fold 4: log-loss 0.3783 | AUC 0.909 | Acc 0.8311 | Brier 0.1209
Fold 5: log-loss 0.3639 | AUC 0.917 | Acc 0.8339 | Brier 0.1164

CV mean log-loss  0.3691
CV mean ROC-AUC   0.914
CV mean Accuracy  0.8320
CV mean Brier     0.1181
```