Machine Learning Project

Counter-Strike Pro Match Data Analysis

Import

We start our project by **importing the needed modules.**

If we have new import to make we can just add them here to make the code lighter to read.

In [1]:
```
import re
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import LinearSVC
import xgboost as xgb
from xgboost import XGBClassifier

print("Modules imported.")
```

```
Modules imported.
```

We now import the data from the different csv files,

We merge them in a single dataset thanks to the data_link variables that is a **unique ID** for each match,

We then proceed to drop duplicate if they exist, and we sort the values by the time of the matchs, this will be useful later on to get the performances of a given team.

In [2]:
```
df_games = pd.read_csv('./database/game_data_rh.csv')
df_results = pd.read_csv('./database/historic_games_list.csv')

df = df_results.merge(df_games, on='game_link', how='inner')
df = df.drop_duplicates(subset='game_link', keep='first')
df = df.sort_values('date_unix').reset_index(drop=True)

print(f'Total matches: {len(df)}')
```

```
Total matches: 126872
```

Map Encoding

Instead of one-hot encoding maps (which doesn't capture team-specific performance), we track each team's **historical win rate on each map**.

This makes the map feature predictive: some teams perform better on certain maps.

In [ ]:
```
print(f"Unique maps: {df['map_name_short'].nunique()}")
print(f"\nMap distribution:\n{df['map_name_short'].value_counts()}")

team_map_history = {}

def get_team_map_winrate(team_name, map_name):
    """Get a team's historical win rate on a specific map"""
    if team_name not in team_map_history:
        return 0.5  #default
    if map_name not in team_map_history[team_name]:
        return 0.5  #default
    stats = team_map_history[team_name][map_name]
    if stats['total'] == 0:
        return 0.5
    return stats['wins'] / stats['total']

def update_team_map_history(team_name, map_name, won):
    """Update a team's map history after a match"""
    if team_name not in team_map_history:
        team_map_history[team_name] = {}
    if map_name not in team_map_history[team_name]:
        team_map_history[team_name][map_name] = {'wins': 0, 'total': 0}

    team_map_history[team_name][map_name]['total'] += 1
    if won:
        team_map_history[team_name][map_name]['wins'] += 1

print("\nMap win rate tracking functions defined.")
```

```
Unique maps: 15

Map distribution:
mrg        23758
inf        22121
ovp        14940
nuke       14107
d2         13804
trn        11993
cch         8885
vtg         6391
cbl         5751
anc         4086
anb          981
ssn           46
tcn            4
mill_ce        3
dust_se        2
Name: map_name_short, dtype: int64

Map win rate tracking functions defined.
```

The extract the rounds variables that is into parenthese and we look at the number of win for both team1 and team2

In [4]:
```
def extract_rounds(x):
    return int(str(x).strip().replace('(', '').replace(')', ''))
```

```
df['team1_rounds'] = df['team1_rounds'].apply(extract_rounds)
df['team2_rounds'] = df['team2_rounds'].apply(extract_rounds)
df['team1_won'] = (df['team1_rounds'] > df['team2_rounds']).astype(int)

print(f"Team1 wins: {df['team1_won'].sum()}, Team2 wins: {(1-df['team1_won']).su
```

```
Team1 wins: 63196, Team2 wins: 63676
```

It's approximatevely the same so we don't need to do any class balancing

We can now proceed to get the data of the rest of the variables, using similar techniques to get the value when inside parenthse, with a "%" sign...

In [ ]:
```
def parse_khs(s):
    m = re.findall(r'\d+', str(s))
    if len(m) >= 2:
        return int(m[0]), int(m[1])
    return np.nan, np.nan

def parse_stat(s):
    s = str(s).replace('%', '').replace('(', '').replace(')', '')
    try:
        return float(s)
    except:
        return np.nan


# Create a copy of the dataframe because we had fragmentation issues
df_processed = df.copy()

# Parse player stats
for t in [1, 2]:
    for p in range(1, 6):
        khs_col = f'team{t}_p{p}_khs'
        if khs_col in df_processed.columns:
            # Parse kills and headshots
            kills, hs = zip(*df_processed[khs_col].apply(parse_khs))
            df_processed[f'team{t}_p{p}_kills'] = kills
            df_processed[f'team{t}_p{p}_hs'] = hs

        for stat in ['deaths', 'kast', 'kddiff', 'adr', 'fkdiff', 'game_rating']
            col = f'team{t}_p{p}_{stat}'
            if col in df_processed.columns:
                df_processed[f'team{t}_p{p}_{stat}_num'] = df_processed[col].app


print('Player stats processed.')
```

```
Player stats processed.
```

Now that we have got all players variables parsed we will calculate the average for each team.

In [6]:
```
team_history = {}
features = []
HISTORY_SIZE = 10

def calculate_team_features(team_name):
```

```
    if team_name not in team_history or len(team_history[team_name]) == 0:
        return { #average of 25th percentile values
            'win_rate': 0.5,
            'avg_rating': 1.06,
            'avg_kills': 16.8,
            'avg_deaths': 16.8,
            'avg_kast': 69.3,
            'avg_adr': 74.2,
            'avg_kddiff': 0.07,
            'avg_fkdiff': 0.03,
            'rounds_won': 13.1
        }

    recent_games = team_history[team_name][-HISTORY_SIZE:]

    return {
        'win_rate': np.mean([g['won'] for g in recent_games]),
        'avg_rating': np.mean([g['avg_rating'] for g in recent_games]),
        'avg_kills': np.mean([g['avg_kills'] for g in recent_games]),
        'avg_deaths': np.mean([g['avg_deaths'] for g in recent_games]),
        'avg_kast': np.mean([g['avg_kast'] for g in recent_games]),
        'avg_adr': np.mean([g['avg_adr'] for g in recent_games]),
        'avg_kddiff': np.mean([g['avg_kddiff'] for g in recent_games]),
        'avg_fkdiff': np.mean([g['avg_fkdiff'] for g in recent_games]),
        'rounds_won': np.mean([g['rounds'] for g in recent_games])
    }

def extract_team_avg_stats(match, team_prefix):
    """Extract team average stats (average across 5 players) for one match"""
    ratings, kills, deaths, kasts, adrs, kddiffs, fkdiffs = [], [], [], [], [],

    for player_num in range(1, 6):
        r = match.get(f'{team_prefix}_p{player_num}_game_rating_num')
        if pd.notna(r): ratings.append(r)

        k = match.get(f'{team_prefix}_p{player_num}_kills')
        if pd.notna(k): kills.append(k)

        d = match.get(f'{team_prefix}_p{player_num}_deaths_num')
        if pd.notna(d): deaths.append(d)

        kast = match.get(f'{team_prefix}_p{player_num}_kast_num')
        if pd.notna(kast): kasts.append(kast)

        adr = match.get(f'{team_prefix}_p{player_num}_adr_num')
        if pd.notna(adr): adrs.append(adr)

        kd = match.get(f'{team_prefix}_p{player_num}_kddiff_num')
        if pd.notna(kd): kddiffs.append(kd)

        fk = match.get(f'{team_prefix}_p{player_num}_fkdiff_num')
        if pd.notna(fk): fkdiffs.append(fk)


    return {
        'avg_rating': np.mean(ratings) if ratings else 1.06,
        'avg_kills': np.mean(kills) if kills else 16.8,
        'avg_deaths': np.mean(deaths) if deaths else 16.8,
        'avg_kast': np.mean(kasts) if kasts else 69.3,
        'avg_adr': np.mean(adrs) if adrs else 74.2,
```

```
        'avg_kddiff': np.mean(kddiffs) if kddiffs else 0.07,
        'avg_fkdiff': np.mean(fkdiffs) if fkdiffs else 0.03,
        'rounds': match.get(f'{team_prefix}_rounds', 13.1)
    }

# Process each match
for idx, match in df.iterrows():
    team1_name = match['team1']
    team2_name = match['team2']
    map_name = match['map_name_short']

    team1_features = calculate_team_features(team1_name)
    team2_features = calculate_team_features(team2_name)

    # Save features for this match
    match_features = {'game_link': match['game_link']}
    for stat_name, stat_value in team1_features.items():
        match_features[f'team1_hist_{stat_name}'] = stat_value
    for stat_name, stat_value in team2_features.items():
        match_features[f'team2_hist_{stat_name}'] = stat_value

    # Add map-specific win rate
    match_features['team1_map_winrate'] = get_team_map_winrate(team1_name, map_n
    match_features['team2_map_winrate'] = get_team_map_winrate(team2_name, map_n

    match_features['target'] = match['team1_won']
    features.append(match_features)

    if team1_name not in team_history:
        team_history[team1_name] = []
    team1_stats = extract_team_avg_stats(match, 'team1')
    team1_stats['won'] = 1 if match['team1_won'] == 1 else 0
    team_history[team1_name].append(team1_stats)

    if team2_name not in team_history:
        team_history[team2_name] = []
    team2_stats = extract_team_avg_stats(match, 'team2')
    team2_stats['won'] = 1 if match['team1_won'] == 0 else 0
    team_history[team2_name].append(team2_stats)

    update_team_map_history(team1_name, map_name, match['team1_won'] == 1)
    update_team_map_history(team2_name, map_name, match['team1_won'] == 0)

    if idx % 10000 == 0:
        print(f'Processed {idx}/{len(df)} matches')

features_df = pd.DataFrame(features)
print(f'Built features: {features_df.shape}')
```

```
Processed 0/126872 matches
Processed 10000/126872 matches
Processed 20000/126872 matches
Processed 30000/126872 matches
Processed 40000/126872 matches
Processed 50000/126872 matches
Processed 60000/126872 matches
Processed 70000/126872 matches
Processed 80000/126872 matches
Processed 90000/126872 matches
Processed 100000/126872 matches
Processed 110000/126872 matches
Processed 120000/126872 matches
Built features: (126872, 22)
```

In [7]:
```python
# create difference features
for stat in ['win_rate', 'avg_rating', 'avg_kills', 'avg_deaths', 'avg_kast', 'a
    features_df[f'{stat}_diff'] = features_df[f'team1_hist_{stat}'] - features_d

# Add map-specific win rate difference
features_df['map_winrate_diff'] = features_df['team1_map_winrate'] - features_df

print('Difference features created (including map win rate)')
```

```
Difference features created (including map win rate)
```

In [8]:
```python
X = features_df.drop(columns=['game_link', 'target'])
y = features_df['target']

# chronological split
split_idx = int(len(X) * 0.8)
X_train, X_test = X.iloc[:split_idx], X.iloc[split_idx:]
y_train, y_test = y.iloc[:split_idx], y.iloc[split_idx:]

print(f'Train: {X_train.shape}, Test: {X_test.shape}')
print(f'Train win rate: {y_train.mean():.3f}, Test win rate: {y_test.mean():.3f}
```

```
Train: (101497, 30), Test: (25375, 30)
Train win rate: 0.503, Test win rate: 0.478
```

In [15]:
```python
#We scale data
X_train = X_train.apply(pd.to_numeric).fillna(0)
X_test = X_test.apply(pd.to_numeric).fillna(0)

print(f'\nFinal feature count: {X_train.shape[1]}')
print(f'Features: {list(X_train.columns)}')

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print('\nData scaled successfully!')
```

```
Final feature count: 30
Features: ['team1_hist_win_rate', 'team1_hist_avg_rating', 'team1_hist_avg_kill
s', 'team1_hist_avg_deaths', 'team1_hist_avg_kast', 'team1_hist_avg_adr', 'team1_
hist_avg_kddiff', 'team1_hist_avg_fkdiff', 'team1_hist_rounds_won', 'team2_hist_w
in_rate', 'team2_hist_avg_rating', 'team2_hist_avg_kills', 'team2_hist_avg_death
s', 'team2_hist_avg_kast', 'team2_hist_avg_adr', 'team2_hist_avg_kddiff', 'team2_
hist_avg_fkdiff', 'team2_hist_rounds_won', 'team1_map_winrate', 'team2_map_winrat
e', 'win_rate_diff', 'avg_rating_diff', 'avg_kills_diff', 'avg_deaths_diff', 'avg
_kast_diff', 'avg_adr_diff', 'avg_kddiff_diff', 'avg_fkdiff_diff', 'rounds_won_di
ff', 'map_winrate_diff']

Data scaled successfully!
```

In [16]:
```
X_train
```

Out[16]:

	team1_hist_win_rate	team1_hist_avg_rating	team1_hist_avg_kills	team1_hist_avg
0	0.5	1.06	16.8	
1	0.5	1.06	16.8	
2	0.5	1.06	16.8	
3	0.5	1.06	16.8	
4	0.5	1.06	16.8	
...	...	...	...	
101492	0.6	1.06	16.8	
101493	0.8	1.06	16.8	
101494	0.7	1.06	16.8	
101495	0.0	1.06	16.8	
101496	0.6	1.06	16.8	

101497 rows × 30 columns

We can now proceed to train and run different model, we will start with a random forest classifier

In [17]:
```
model = RandomForestClassifier(n_estimators=100,max_depth=10, random_state=42)

model.fit(X_train_scaled, y_train)

y_pred_train = model.predict(X_train_scaled)
y_pred_test = model.predict(X_test_scaled)

train_acc = accuracy_score(y_train, y_pred_train)
test_acc = accuracy_score(y_test, y_pred_test)
train_f1 = f1_score(y_train, y_pred_train)
test_f1 = f1_score(y_test, y_pred_test)

print(f'Train Accuracy: {train_acc:.4f}, F1: {train_f1:.4f}')
print(f'Test  Accuracy: {test_acc:.4f}, F1: {test_f1:.4f}')
print("\n"+classification_report(y_test, y_pred_test))
```

```
Train Accuracy: 0.7776, F1: 0.7629
Test  Accuracy: 0.7395, F1: 0.6987

              precision    recall  f1-score   support

           0       0.71      0.84      0.77     13258
           1       0.78      0.63      0.70     12117

    accuracy                           0.74     25375
   macro avg       0.75      0.73      0.73     25375
weighted avg       0.75      0.74      0.74     25375
```

In [18]:
```python
importances = pd.DataFrame({
    'feature': X_train.columns,
    'importance': model.feature_importances_
}).sort_values('importance', ascending=False)

print('Top 15 most important features:')
print(importances.head(15))
```

```
Top 15 most important features:
                 feature  importance
28       rounds_won_diff    0.217949
8   team1_hist_rounds_won    0.151117
17  team2_hist_rounds_won    0.150114
0     team1_hist_win_rate    0.108340
20          win_rate_diff    0.099762
9     team2_hist_win_rate    0.097218
29       map_winrate_diff    0.054373
19      team2_map_winrate    0.034783
18      team1_map_winrate    0.032516
27        avg_fkdiff_diff    0.009389
23        avg_deaths_diff    0.006471
24          avg_kast_diff    0.006421
26        avg_kddiff_diff    0.006019
22         avg_kills_diff    0.004147
25           avg_adr_diff    0.004121
```

We will try the same thing with a simpler model like regression

In [19]:
```python
model = LinearSVC(dual=False, max_iter=1000, tol=0.127)
model.fit(X_train_scaled, y_train)
y_pred_train = model.predict(X_train_scaled)
y_pred_test = model.predict(X_test_scaled)

train_f1 = f1_score(y_train, y_pred_train)
test_f1 = f1_score(y_test, y_pred_test)
train_acc = accuracy_score(y_train, y_pred_train)
test_acc = accuracy_score(y_test, y_pred_test)

print(f'Train Accuracy: {train_acc:.4f}, F1: {train_f1:.4f}')
print(f'Test  Accuracy: {test_acc:.4f}, F1: {test_f1:.4f}')
print(classification_report(y_test, y_pred_test))
```

```
Train Accuracy: 0.7543, F1: 0.7492
Test  Accuracy: 0.7353, F1: 0.7133
              precision    recall  f1-score   support

           0       0.73      0.78      0.75     13258
           1       0.74      0.69      0.71     12117

    accuracy                           0.74     25375
   macro avg       0.74      0.73      0.73     25375
weighted avg       0.74      0.74      0.73     25375
```

In [20]:
```python
importances = pd.DataFrame({
    'feature': X_train.columns,
    'importance': np.abs(model.coef_[0])  # Use absolute value of coefficients
}).sort_values('importance', ascending=False)

print('Top 15 most important features:')
print(importances.head(15))
```

```
Top 15 most important features:
                  feature  importance
28        rounds_won_diff    0.110718
20          win_rate_diff    0.097842
17  team2_hist_rounds_won    0.091889
8   team1_hist_rounds_won    0.085420
29       map_winrate_diff    0.084944
9     team2_hist_win_rate    0.080679
0     team1_hist_win_rate    0.074020
19      team2_map_winrate    0.069492
18      team1_map_winrate    0.062256
27        avg_fkdiff_diff    0.016875
26        avg_kddiff_diff    0.015898
24          avg_kast_diff    0.015898
23        avg_deaths_diff    0.015898
22         avg_kills_diff    0.015898
25           avg_adr_diff    0.015898
```

In [21]:
```python
model = XGBClassifier(
    n_estimators=90,
    learning_rate=0.14,
    objective='binary:logistic',
    random_state=42,
    eval_metric='logloss'
)

model.fit(X_train_scaled, y_train)

y_pred_train = model.predict(X_train_scaled)
y_pred_test = model.predict(X_test_scaled)

train_acc = accuracy_score(y_train, y_pred_train)
test_acc = accuracy_score(y_test, y_pred_test)
train_f1 = f1_score(y_train, y_pred_train)
test_f1 = f1_score(y_test, y_pred_test)

print(f'Train Accuracy: {train_acc:.4f}, F1: {train_f1:.4f}')
print(f'Test  Accuracy: {test_acc:.4f}, F1: {test_f1:.4f}')
print("\n"+classification_report(y_test, y_pred_test))
```

```
Train Accuracy: 0.7807, F1: 0.7724
Test  Accuracy: 0.7409, F1: 0.7092

              precision    recall  f1-score   support

           0       0.72      0.81      0.77     13258
           1       0.76      0.66      0.71     12117

    accuracy                           0.74     25375
   macro avg       0.74      0.74      0.74     25375
weighted avg       0.74      0.74      0.74     25375
```

In [22]:
```python
importances = pd.DataFrame({
    'feature': X_train.columns,
    'importance': model.feature_importances_
}).sort_values('importance', ascending=False)

print('Top 15 most important features:')
print(importances.head(15))
```

```
Top 15 most important features:
                  feature  importance
28        rounds_won_diff    0.366016
8   team1_hist_rounds_won    0.225963
17  team2_hist_rounds_won    0.176966
27        avg_fkdiff_diff    0.108605
16  team2_hist_avg_fkdiff    0.033991
19      team2_map_winrate    0.010237
9     team2_hist_win_rate    0.009876
20          win_rate_diff    0.009561
0     team1_hist_win_rate    0.008578
18      team1_map_winrate    0.008530
11   team2_hist_avg_kills    0.007582
7   team1_hist_avg_fkdiff    0.006718
29       map_winrate_diff    0.005600
10  team2_hist_avg_rating    0.004936
1   team1_hist_avg_rating    0.004514
```

Baseline Benchmark

To verify our models actually learn something useful, we compare against a simple **win rate baseline**.

This baseline predicts the team with the higher historical win rate will win.

In [28]:
```python
y_pred_baseline = (features_df['team1_hist_win_rate'] > features_df['team2_hist_

# For test set only (using same chronological split)
y_pred_baseline_test = y_pred_baseline.iloc[split_idx:]

baseline_acc = accuracy_score(y_test, y_pred_baseline_test)
baseline_f1 = f1_score(y_test, y_pred_baseline_test)

print(f'Baseline Accuracy: {baseline_acc:.4f}, F1: {baseline_f1:.4f}')
print(f'Model Test Accuracy: {test_acc:.4f}, F1: {test_f1:.4f}')
print(f'Improvement over baseline: {((test_acc/baseline_acc)-1)*100:.2f}% accura
```

```
Baseline Accuracy: 0.7187, F1: 0.6968
Model Test Accuracy: 0.7409, F1: 0.7092
Improvement over baseline: 3.09% accuracy
```