DATA_PREPROCESSING_FEATURE_ENGINEERING

This folder contains all the scripts responsible for transforming raw Hudle StatsBomb data into a clean, structured, and feature-enriched dataset ready for modeling and analysis. These scripts form the data preprocessing and feature engineering stage of the pipeline. This stage ensures that all relevant match, event, and frame-level information from the Hudle StatsBomb API is extracted, validated, and reformatted into a processable DataFrame that later feeds the analytical and machine learning components of the project.

Main Pipeline Overview

The main() function automates the complete corner analysis workflow. It retrieves match data from all available competitions and seasons, computes temporal and contextual metrics, and produces a consolidated analytical dataset.

- Data retrieval: Downloads and caches StatsBomb data (competitions, matches, events, and freeze-frames).
- Preprocessing: Calculates game state, 20-second effectiveness, and corner execution time.
- Corner analysis: Runs CornerAnalyzerP0 (corner moment) and CornerAnalyzerP1 (immediate post-corner moment).
- Output: Exports a consolidated CSV file containing all P0 and P1 tactical, temporal, and outcome metrics for every analyzed match.

Class DataDownloader

The DataDownloader class handles secure data extraction from Hudle StatsBomb API. It is the first step in the data pipeline, designed to automate the process of retrieving competitions, matches, events, and frames.

1. *get_competitions()*: retrieves all available competitions in the Hudle StatsBomb database. Returns a DataFrame listing competition IDs, names, and available seasons.
2. *get_matches(competition_id, season_id)*: retrieves all matches belonging to a given competition and season. Returns a DataFrame.
3. *get_match_events(match_id)*: retrieves all events belonging to a given match. Returns a DataFrame.
4. *get_match_frames(match_id)*: downloads freeze-frame data corresponding to a given match. Returns a DataFrame.

Class CornerP0

The Corner class processes each corner kick event from the Hudle StatsBomb dataset, organizing player positions, spatial distributions, and tactical information into predefined pitch zones. It enables detailed quantitative analysis of attacking and defending structures during corner situations.
All the computations correspond to the P0 moment: the exact instant when the corner kick is taken.

1. count_defenders_in_18yd_box( ): sums all defenders positioned inside the 18-yard box at P0.
2. count_attackers_in_6yd_box( ): counts all attacking players (teammates) positioned within the 6-yard box at P0.
3. count_attackers_out_6yd_box( ): counts attackers positioned outside the 6-yard box but within the 18-yard box
4. determine_corner_side( ): determines whether the corner was executed from the right or left side of the field.
5. iter_freeze_players( ): iterates through the freeze_frame data to access all player dictionaries (attackers, defenders, and goalkeeper) at the exact moment of the corner (P0).
6. count_players_in_zones( ): generic utility to count players within any specified rectangular region at P0. It allows filtering by teammate status (attackers or defenders) and excludes goalkeepers by default.
7. zones_to_dict ( ): Converts the structured zone count data into a flat dictionary, ready for DataFrame integration. Each zone generates: P0_n_att_zone_X → number of attackers; P0_n_def_zone_X → number of defenders; P0_total_n_zone_X → total players; zone_X_name → zone label
8. get_goalkeeper_coordinates( ): extracts the goalkeeper's coordinates (x,y) from the freeze-frame at P0.
9. to_dict( ): converts the entire corner object into a single flat dictionary containing:
    - Event metadata (match, player, time, technique)
    - Positional data in P0 (locations, zones, and box counts)
    - Goalkeeper position

Class CornerAnalyzerP0

The CornerAnalyzerP0 class compiles a detailed P0 corner dataset by taking already-processed events, fetching freeze-frames, building CornerP0 objects, and returning a tidy DataFrame ready for analysis or modeling.

1. analyze_corners_from_processed_data(): generates the P0 corners table: filters corner events from the processed DataFrame (with game state and timing), retrieves freeze-frames, constructs CornerP0 analyses, and merges selected contextual fields (e.g., game_state, corner execution time, xG/goal 20s, teams/season). Returns a single consolidated DataFrame.
2. save_to_csv(): exports the current corners analysis DataFrame to a CSV file.

Class CornerAnalyzerP1

The CornerAnalyzerP1 class identifies and analyzes the first post-corner event (P1), normalizing perspective to the corner-taking team and producing a tidy DataFrame of P1 tactical features.

1. iter_freeze_players(): yields player dicts from a freeze frame, handling nested list shapes.
2. coerce_bool(): standardizes truthy/falsey values (bool/num/str) into a boolean.
3. debug_p1_analysis(): prints quick diagnostics comparing original vs adjusted freeze-frame roles/coordinates.

4. normalize_coordinates(): keeps all coordinates in the attacking (P0) perspective; flips when possession changes.
5. adjust_freeze_frame_for_p0_perspective(): produces an adjusted freeze frame aligned to P0: normalizes coordinates and fixes teammate flags (goalkeeper-aware, with fallback).
6. find_first_p1_with_freeze_frame(): finds the first event after P0 that has a freeze frame to use as P1.
7. analyze_p1_event(): builds a CornerP1 object from the chosen P1 event and returns its flat metrics dict.
8. analyze_p1_events(): runs the full P1 pipeline for all P0 corners and returns a DataFrame of P1 analyses.
9. merge_p0_p1_data(): joins P1 metrics back onto the P0 corners table, producing a combined dataset.
10. save_to_csv(): exports the collected P1 analysis to a CSV file.

Class CornerP1

The CornerP1 class analyzes the P1 moment, which represents the first event occurring after the corner kick (P0). It evaluates player positioning, team distribution, and tactical setup at that post-corner instant. The analysis uses the same predefined zones as CornerP0, but always keeps a normalized "Right side" perspective for consistency.

1. coerce_bool(): converts any value type (boolean, numeric, or string) into a standardized boolean to ensure reliable teammate and goalkeeper identification.
2. iter_freeze_players(): iterates over all player dictionaries from the freeze-frame data to access their positions and attributes.
3. count_defenders_in_18yd_box(): counts the total number of defenders located inside the 18-yard box at P1.
4. count_attackers_in_6yd_box(): counts attackers positioned inside the 6-yard box at P1.
5. count_attackers_out_6yd_box(): counts attackers who are outside the 6-yard box but still within the 18-yard box.
6. count_players_in_zones(): calculates how many attackers and defenders are present in each of the predefined tactical zones based on player coordinates at P1.
7. get_goalkeeper_coordinates(): extracts the goalkeeper's position (x, y) from the freeze-frame data at P1.
8. zones_to_dict(): converts the zone-based counts into a flat dictionary for easier integration into DataFrames, prefixed with "P1_".
9. to_dict(): generates a complete dictionary summarizing the P1 analysis, including zone data, aggregated box counts, goalkeeper coordinates, and whether coordinates were normalized relative to the attacking team's perspective.

Class GameStateCalculator

The GameStateCalculator class computes the game state for each event and performs minimal enrichment of pass attributes.

1. calculate_game_state(events-df, matches_df): generates a complete event dataset that includes match metadata, chronological ordering, the score difference before each event (game_state), and consistent pass technique labels.
2. sort_events_chronologically(DataFrame): ensures all events are properly ordered in time within each match for accurate sequential analysis.
3. calculate_game_state_simple(DataFrame): generates the game_state variable, which reflects the running score (home minus away) before each event.
4. fill_pass_technique_na(DataFrame): fills missing pass technique values to create a uniform and interpretable categorical variable across all events (Ground, Short, Cutback)

Class TimeToCorner

The TimeToCorner class calculates the corner execution time and categorizes it into defined time ranges. It operates on a pre-processed DataFrame that has already been sorted chronologically by the GameStateCalculator.

The computed time refers to the P0 moment — the exact instant when the corner kick is executed. This allows the analysis of how quickly corners are taken after the preceding event in the same match and period.

1. calculate_corner_execution_time( ): generates two new features (corner execution time raw and corner execution time label) that describe how quickly a corner kick (P0) is executed after the previous event
2. ts_to_seconds( ): converts event timestamps into total seconds to enable precise time difference calculations.
3. categorize_execution_time( ): generates a descriptive label (e.g., “0–5 seconds”, “5–10 seconds”) based on the execution time, facilitating categorical analysis of corner tempo.

Class TwentySecondMetrics

The TwentySecondMetrics class calculates performance indicators within the 20 seconds following each corner kick (P0). It measures offensive and defensive effectiveness immediately after the set piece.

1. calculate_20s_metrics( ): generates new metrics for every corner event, including:
    a. xg_20s: total expected goals (xG) created by the attacking team within 20 seconds after the corner.
    b. xg_20s_def: xG conceded (opponent’s xG) in the same 20-second window.
    c. goal_20s: whether the attacking team scored within 20 seconds.
    d. goal_20s_def: whether the defending team scored within 20 seconds.
2. identify_corner_events( ): detects all events corresponding to corner executions (play_pattern = "From Corner", pass_type = "Corner", type = "Pass").

3. calculate_metrics_for_corner(): calculates the attacking and defending outcomes in the 20-second window after each corner, summarizing xG and goal occurrences for both teams.