

# Configuration Manual

MSc Research Project  
MSCDA\_JANOL24\_O

Coenraad Lukas Odendaal  
Student ID: X23130512

School of Computing  
National College of Ireland

Supervisor: Mohammed Hasanuzzaman

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Coenraad Lukas Odendaal  
**Student ID:** x23130512  
**Programme:** MSc Data Analytics **Year:** 2025  
**Module:** Research Project  
**Lecturer:** Mohammed Hasanuzzaman  
**Submission Due Date:** 11/12/2025  
**Project Title:** Decision-Making Under Pressure: A Multi-Level Analysis of Tactical Adaptation and Passing Intelligence in Football  
**Word Count:** 1092 **Page Count:** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Lukas Odendaal

**Date:** 11/12/2025

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

|                                                                                                                                                                                           |                          |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| Attach a completed copy of this sheet to each project (including multiple copies)                                                                                                         | <input type="checkbox"/> |
| <b>Attach a Moodle submission receipt of the online project submission,</b> to each project (including multiple copies).                                                                  | <input type="checkbox"/> |
| <b>You must ensure that you retain a HARD COPY of the project,</b> both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | <input type="checkbox"/> |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

|                                  |  |
|----------------------------------|--|
| <b>Office Use Only</b>           |  |
| Signature:                       |  |
| Date:                            |  |
| Penalty Applied (if applicable): |  |

# Configuration Manual

Coenraad Lukas Odendaal  
Student ID: x23130512

## 1 Introduction

This configuration manual helps setting up the environment to successfully execute the code/analysis for the research project. It includes the hardware and software requirements, all the python libraries used, and all tools used in the creation of this project.

## 2 Hardware Requirement

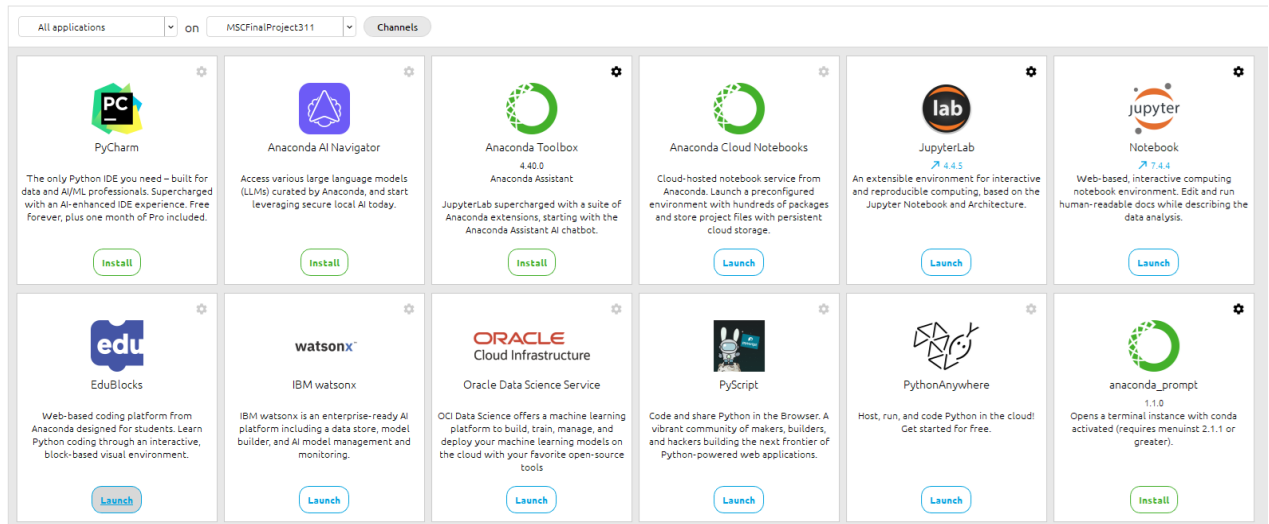
This project used a personal laptop, with the below hardware:

| Device specifications |                                                     |
|-----------------------|-----------------------------------------------------|
| Device name           | Lukas-Odendaal                                      |
| Processor             | 13th Gen Intel(R) Core(TM) i7-13700H (2.40 GHz)     |
| Installed RAM         | 40.0 GB (39.6 GB usable)                            |
| Device ID             | DFE3E830-257F-403E-8DF8-BFA04B54D18D                |
| Product ID            | 00342-21117-13581-AAOEM                             |
| System type           | 64-bit operating system, x64-based processor        |
| Pen and touch         | No pen or touch input is available for this display |

**Figure 1. Hardware Requirement**

## 3 Software requirement

The environment used for this project is Jupyter notebook in the anaconda environment and listed below are the software used in this project as well as the libraries and tools used for this project. All these tools need to be installed in the environment, before the executing of notebooks.



**Figure 2. Anaconda Environment**

**Table1. Software requirements**

| Software                                 | Metric                                                                                                                                                                                                                                                                   |
|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Operating System</b>                  | Windows 10 (64-bit)                                                                                                                                                                                                                                                      |
| <b>Programming Language:</b>             | Python 3.10                                                                                                                                                                                                                                                              |
| <b>Development Environment:</b>          | Jupyter Notebook                                                                                                                                                                                                                                                         |
| <b>Core Data Processing Libraries:</b>   | <ul style="list-style-type: none"> <li>• NumPy for numerical computation</li> <li>• Pandas for data manipulation and transformation</li> <li>• SciPy for statistical operations</li> </ul>                                                                               |
| <b>Machine Learning &amp; Modelling:</b> | <ul style="list-style-type: none"> <li>• Scikit-learn for feature scaling, model evaluation and probability estimation</li> <li>• XGBoost for gradient-boosted classification for pass success prediction</li> <li>• Joblib for model persistence and loading</li> </ul> |
| <b>Visualisation Libraries:</b>          | <ul style="list-style-type: none"> <li>• Matplotlib for primary plotting and time-series visualisation</li> <li>• Seaborn for statistical visualisations and distribution plots</li> </ul>                                                                               |
| <b>File Handling &amp; Storage:</b>      | <ul style="list-style-type: none"> <li>• CSV and NumPy binary formats for structured data outputs</li> <li>• JSON parsing utilities for StatsBomb raw data extraction</li> </ul>                                                                                         |
| <b>Data Source:</b>                      | StatsBomb Open Event Data (JSON format)                                                                                                                                                                                                                                  |



```

#Master_file
import os
import time

#0_Json_download
import os
import asyncio
import aiohttp
from tqdm.asyncio import tqdm
from pathlib import Path
import requests

#1_Extraction_final
import os, json, re
import numpy as np
import pandas as pd
from tqdm import tqdm

#2_Grid_final
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#3_GOI_model_final
import os
import joblib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    roc_auc_score, accuracy_score, roc_curve,
    precision_recall_curve, average_precision_score,
    brier_score_loss
)
from sklearn.calibration import calibration_curve
from xgboost import XGBClassifier

#4_Compute_GOI_final
import os
import numpy as np
import pandas as pd
import joblib
import matplotlib.pyplot as plt
from tqdm import tqdm
import seaborn as sns

#5_Formation_drift_final
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#6_HPEI_final
import os
import numpy as np
import pandas as pd
import joblib
import matplotlib.pyplot as plt
import seaborn as sns

#7_Sub_recommendation_sys
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

```

Figure 3. Libraries used

## 4 File Structure

Below gives the file structure and sub file structure for the project to be successfully executed. The libraries file contains all the libraries used in the project; it is not necessary for the project to be complete. The raw file contains to sub folders named “events” and “raw” containing the JSON files for events and lineups. All notebook files and the folders need to be stored in the root of the project as well as the “raw”, “processed”, “outputs” and “models” folders. There will be two folders inside the “raw” folder named “events” and “lineups” as seen in figure 4.

There is a total of 8 Notebooks that is needed for this project to be successfully re-created. They are numbers in order from 0 to 7 with the Masterfile needed to run all notebooks.

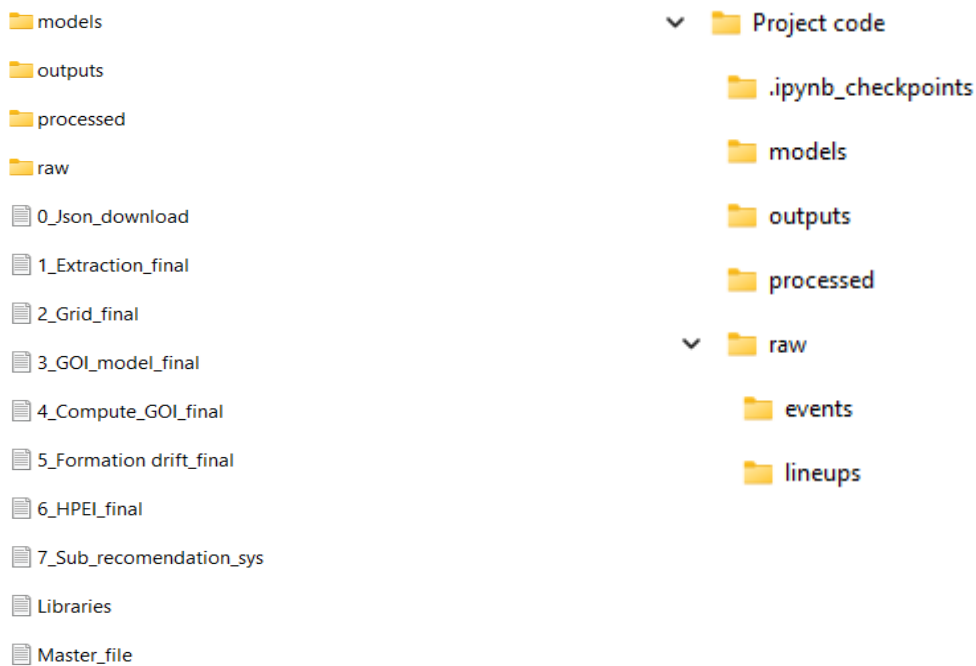


Figure 4. folder structure

## 5 Implementation

### Step 1 – Data download

File 0\_Json\_download contains the path to where the original data for StatsBomb is located and the paths where the data needs to be extracted to.

```
[1]: import os
import asyncio
import aiohttp
from tqdm.asyncio import tqdm
from pathlib import Path
import requests

# --- GitHub base paths ---
BASE_URL = "https://raw.githubusercontent.com/statsbomb/open-data/master/data/"
MATCHES_INDEX = BASE_URL + "matches/"
EVENTS_URL = BASE_URL + "events/"
LINEUPS_URL = BASE_URL + "lineups/"

# --- Output folders ---
EVENTS_DIR = Path("raw/events")
LINEUPS_DIR = Path("raw/lineups")
```

### Step 2 – Extraction and preprocessing

The raw JSON files are extracted and turned into a CSV File named “compact\_events.csv”. The code finds the lists in the JSON structure, then iterate over each JSON file in “raw/events” folder and extract the data to get it in one row per event.

```
[1]: import os, json, re
import numpy as np
import pandas as pd
from tqdm import tqdm

# Directory where JSON event files are stored
RAW_DIR = "raw/events"

# Output CSV
OUT_CSV = "processed/events_compact.csv"
os.makedirs("processed", exist_ok=True)
```

For the model derivations, there are some extra flags required, this gets added to the dataset after the extraction. Dataset columns after the extra flags have been created and/or converted. As well as conversion of datatypes to contain 0,1 values and added columns.

```
# Type conversions / cleaning
df["minute"] = pd.to_numeric(df["minute"], errors="coerce").fillna(0).astype(int)
df["second"] = pd.to_numeric(df["second"], errors="coerce").fillna(0).astype(int)
df["under_pressure"] = df["under_pressure"].fillna(False).astype(int)

# Convenience flags
df["pass_flag"] = df["event_type"].eq("Pass")
df["carry_flag"] = df["event_type"].eq("Carry")
df["shot_flag"] = df["event_type"].eq("Shot")
df["press_flag"] = df["event_type"].eq("Pressure")
df["duel_flag"] = df["event_type"].eq("Duel")
df["ball_receipt_flag"] = df["event_type"].eq("Ball Receipt")

# Pass completion flag (StatsBomb: pass_outcome NaN => completed)
df["pass_success"] = np.where(df["pass_flag"] & df["pass_outcome"].isna(), 1, 0)
```

### Step 3 - Expected Threat (xT) Grid

Dataset loaded from the “compact\_event.csv” file

```
[1]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Output Locations
IN_CSV = "processed/events_compact.csv"
OUT_GRID_NPY = "models/xT_grid.npy"
OUT_GRID_CSV = "processed/xT_grid.csv"

os.makedirs("models", exist_ok=True)
os.makedirs("processed", exist_ok=True)
```

Converts the raw coordinates to standardized pitch dimensions and breaks the pitch into  $X \times Y$  zones. Create the start and end zones of passes to derive the distance traveled and angle of passes

```
[3]: # Grid definitions
X_BINS = 12
Y_BINS = 8

PITCH_X = 120.0
PITCH_Y = 80.0

def bin_xy(x, y):

    bx = int(np.clip((x / PITCH_X) * X_BINS, 0, X_BINS - 1))
    by = int(np.clip((y / PITCH_Y) * Y_BINS, 0, Y_BINS - 1))
    return bx, by

# Apply binning
passes["start_bin_x"], passes["start_bin_y"] = zip(*passes.apply(lambda r: bin_xy(r["start_x"], r["start_y"]), axis=1))
passes["end_bin_x"], passes["end_bin_y"] = zip(*passes.apply(lambda r: bin_xy(r["end_x"], r["end_y"]), axis=1))
```

Compute the xT by doing value iteration, and save the model and csv files to the output locations.

```
[7]: # Initialize xT with empirical goal probabilities
xT = goal_probs.copy()

gamma = 0.9 # future threat discount

for iteration in range(50): # iterate to convergence
    new_xT = goal_probs + gamma * np.sum(trans_probs * xT[None, None, :, :], axis=(2, 3))

    if np.max(np.abs(new_xT - xT)) < 1e-5:
        print(f"Converged after {iteration} iterations.")
        break

    xT = new_xT

xT
```

## Step 4 - Ghost Opportunity Index (GOI) Modelling

Dataset loaded:

```
[2]: IN_CSV = "processed/events_compact.csv"
```

Feature Selection to create Start grid, end grid, pass distance and pass angle

```
[3]: passes["dx"] = passes["end_x"] - passes["start_x"]
passes["dy"] = passes["end_y"] - passes["start_y"]

passes["distance"] = np.sqrt(passes["dx"]**2 + passes["dy"]**2)
passes["angle_to_goal"] = np.arctan2(40 - passes["end_y"], 120 - passes["end_x"])
passes["minute_norm"] = passes["minute"].fillna(0) / 90.0
passes["under_pressure"] = passes["under_pressure"].fillna(0).astype(int)

# target label: successful passes
passes["pass_success"] = passes["pass_outcome"].isna().astype(int)

# Add pass height categorical feature
passes["pass_height_cat"] = passes["pass_height"].astype("category").cat.codes

feature_cols = ["distance", "angle_to_goal", "under_pressure", "minute_norm", "pass_height_cat"]

X = passes[feature_cols].fillna(0).values
y = passes["pass_success"].values
```

The data gets split into 80% training and 20% for validation. The XGBoost model gets trained and uses the AUC and accuracy as evaluation.

```
[6]: if HAS_XGB:
    print("\nTraining XGBoost model...")
    xgb_model = XGBClassifier(
        n_estimators=300,
        learning_rate=0.05,
        max_depth=4,
        subsample=0.8,
        colsample_bytree=0.8,
        eval_metric="logloss",
        random_state=42,
        n_jobs=-1
    )
    xgb_model.fit(X_train, y_train)

    yhat_xgb = xgb_model.predict_proba(X_val)[:,-1]
    auc_xgb = roc_auc_score(y_val, yhat_xgb)
    acc_xgb = accuracy_score(y_val, (yhat_xgb > 0.5))
```

Saves trained model and scaler.

```
joblib.dump(scaler, "models/pass_scaler.joblib")
joblib.dump(best_model, "models/pass_model.joblib")
```

## Step 5 - Compute Ghost Opportunity Index (GOI)

Applies trained model to compute GOI values per action/player. Below are the important directories. Loads pretrained GOI model and scaler.

```
[1]: import os
import numpy as np
import pandas as pd
import joblib
import matplotlib.pyplot as plt
from tqdm import tqdm
import seaborn as sns

IN_CSV = "processed/events_compact.csv"
XT_NPY = "models/xT_grid.npy"
SCALER_PKL = "models/pass_scaler.joblib"
MODEL_PKL = "models/pass_model.joblib"

OUT_PASSES = "processed/passes_with_goi.csv"
OUT_PLAYER = "outputs/player_goi.csv"
OUT_TEAM = "processed/team_goi.csv"
```

Player Aggregation to sum GOI values per player. And save the values to the outputs/player\_goi

```
[6]: # Correct calculation of minutes played across matches
minutes_by_match = passes.groupby(["player", "match_id"])["minute"].nunique()
minutes_played = minutes_by_match.groupby("player").sum().reset_index(name="minutes_played")

# Summed GOI
goi_total = passes.groupby("player")["GOI_pass"].sum().reset_index(name="total_goi")

# Merge
player_goi = goi_total.merge(minutes_played, on="player", how="left")

# Avoid division by zero
player_goi["minutes_played"] = player_goi["minutes_played"].replace(0, 1)

# GOI per 90
player_goi["GOI_per_90"] = player_goi["total_goi"] / player_goi["minutes_played"] * 90

player_goi.to_csv(OUT_PLAYER, index=False)
player_goi.tail()
```

## Step 6 - Formation Drift Index (FDI)

Important directories. Input and output directories

```
[1]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

IN_CSV = "processed/events_compact.csv"
OUT_PLAYER = "processed/player_fdi.csv"
OUT_TEAM = "processed/team_fdi.csv"
FIG_DIR = "outputs"
os.makedirs(FIG_DIR, exist_ok=True)
```

Calculate the baseline values for the formation drift using only the first 10 mins of each player per match.

```
[6]: # Limit to first 10 minutes after entry
baseline = (
    df[df["minute_since_entry"].between(0, 10)]
    .groupby(["match_id", "team", "player"])
    [["start_x", "start_y"]]
    .mean()
    .reset_index()
    .rename(columns={"start_x": "base_x", "start_y": "base_y"})
)

print("Baseline formation players:", len(baseline))
```

After deriving 5-minute rolling windows per player, calculate the drift.

```
[8]: player_fdi = (
    merged.groupby(["match_id", "team", "player"])
    .agg(mean_drift=("drift", "mean"),
         max_drift=("drift", "max"))
    .reset_index()
)
```

Save output to directories.

```
[10]: player_fdi.to_csv(OUT_PLAYER, index=False)
team_fdi.to_csv(OUT_TEAM, index=False)
```

## Step 7 - High pressure expected index (HPEI)

Important directories. Input and output directories. These where the Grid model and scaler are saved.

```
[1]: import os
import numpy as np
import pandas as pd
import joblib
import matplotlib.pyplot as plt
import seaborn as sns

# -----
# Paths
# -----
IN_CSV = "processed/events_compact.csv"
XT_NPY = "models/xT_grid.npy" # from Grid_final_FIXED
MODEL_PKL = "models/pass_model.joblib" # from GOI_model_final_FIXED
SCALER_PKL = "models/pass_scaler.joblib"

OUT_PASSES = "processed/passes_with_hpei.csv"
OUT_PLAYER = "outputs/player_hpei.csv"
OUT_TEAM = "outputs/team_hpei.csv"
```

This is the derivation of the xT gain and the high pressure expected index showing the most important derivations.

```
[7]: # xT at start and end
passes["xT_start"] = passes.apply(lambda r: xt_value(r["start_x"], r["start_y"]), axis=1)
passes["xT_end"] = passes.apply(lambda r: xt_value(r["end_x"], r["end_y"]), axis=1)

# xT gain
passes["xT_gain"] = passes["xT_end"] - passes["xT_start"]

# Expected contribution
passes["expected_impact"] = passes["success_prob"] * passes["xT_gain"]

# High-Pressure weighting
ALPHA = 0.75
passes["HPEI"] = passes["expected_impact"] * (1 + ALPHA * passes["under_pressure"])

passes[["xT_start", "xT_end", "xT_gain", "success_prob", "expected_impact", "HPEI"]].head()
```

Once the derivation is done, it then gets used to cascade across players and team. This then get saved:

```
•[9]: passes.to_csv(OUT_PASSES, index=False)
player_hpei.to_csv(OUT_PLAYER, index=False)
team_hpei.to_csv(OUT_TEAM, index=False)

print(" Saved HPEI outputs:")
print(f" - Per-pass: {OUT_PASSES}")
print(f" - Player summary: {OUT_PLAYER}")
print(f" - Team summary: {OUT_TEAM}")
```

## Step 8 - Substitution recommendation engine

Important directories. Input and output directories. These will be the GOI, FDI and HPEI values as input.

```
•[2]: # -----  
# Load Input Data  
# -----  
  
subs = pd.read_csv("processed/player_substitution_summary.csv")  
pos = pd.read_csv("processed/player_positions.csv")  
goi = pd.read_csv("outputs/player_goi.csv")  
fdi = pd.read_csv("processed/player_fdi.csv")  
hpei = pd.read_csv("outputs/player_hpei.csv")
```

We flag players to be substituted when they fall in any of these categories:

```
[9]: goi_mean, goi_sd = df.GOI.mean(), df.GOI.std()  
fdi_mean, fdi_sd = df.FDI.mean(), df.FDI.std()  
hpei_mean = df.HPEI.mean()  
  
def flag_player(row):  
    reasons = []  
  
    if row.GOI > goi_mean + 1.5 * goi_sd:  
        reasons.append("High GOI")  
  
    if row.FDI > fdi_mean + 1.0 * fdi_sd:  
        reasons.append("High FDI")  
  
    if row.HPEI < hpei_mean * 0.60:  
        reasons.append("Low HPEI")  
  
    return "; ".join(reasons)  
  
df["flag_reason"] = df.apply(flag_player, axis=1)  
df_sub_flags = df[df.flag_reason != ""].copy()  
  
df_sub_flags.head()
```

The substitution model then gets saved to:

```
[15]: rec_df.to_csv("processed/substitution_recommendations_barcelona.csv", index=False)
```

## Step 9 - Master file

Due to the number of notebooks, running them individually can take a lot of time, so there is a Masterfile that can be executed for all the notebooks to be refreshed in order and outputs generated.



```
[1]: import os
import time
import subprocess
import sys

notebooks = [
    "0_Json_download.ipynb",
    "1_Extraction_final.ipynb",
    "2_Grid_final.ipynb",
    "3_GOI_model_final.ipynb",
    "4_Compute_GOI_final.ipynb",
    "5_Formation drift_final.ipynb",
    "6_HPEI_final.ipynb",
    "7_Sub_recomendation_sys.ipynb"
]
```

Built in failure stop should there be an error in one of the Notebooks:

```
# SUCCESS
if result.returncode == 0:
    print(f" {nb} finished in {format_time(elapsed)}")

# FAILURE - STOP IMMEDIATELY
else:
    print(f"\n ERROR: Notebook FAILED → {nb}")
    print(f" Failed after: {format_time(elapsed)}")
    print("\n Error Output:\n")
    print(result.stderr)
    print("\n PIPELINE STOPPED - FIX THIS NOTEBOOK BEFORE CONTINUING.")
    sys.exit(1) # <-- HARD STOP
```

Once the notebook is executed, it gives the running time it took for the notebooks to be refreshed at the end. Just open each notebook in Jupyter and it will give all output created, without the need to re-run.

```
Running 0_Json_download.ipynb...
0_Json_download.ipynb finished in 16.60 seconds

Running 1_Extraction_final.ipynb...
1_Extraction_final.ipynb finished in 6.70 minutes

Running 2_Grid_final.ipynb...
2_Grid_final.ipynb finished in 4.05 minutes

Running 3_GOI_model_final.ipynb...
3_GOI_model_final.ipynb finished in 52.37 seconds

Running 4_Compute_GOI_final.ipynb...
4_Compute_GOI_final.ipynb finished in 2.39 minutes

Running 5_Formation drift_final.ipynb...
5_Formation drift_final.ipynb finished in 47.37 seconds

Running 6_HPEI_final.ipynb...
6_HPEI_final.ipynb finished in 3.40 minutes

Running 7_Sub_recomendation_sys.ipynb...
7_Sub_recomendation_sys.ipynb finished in 2.44 minutes

All notebooks completed successfully in 20.91 minutes
```

The Below references are the inspiration behind the code used to create all the notebooks.

# References

Jakeyk11 (n.d.) *Football data analytics: Expected Threat and zone-based models* [online]. Available at: <https://github.com/jakeyk11/football-data-analytics> (Accessed: 5 December 2025).

Minimav (n.d.) *Basic football pitch with heatmap (Plotly grid visualisation)* [online]. Available at: [https://github.com/minimav/plotly-football-pitch/blob/main/images/basic\\_pitch\\_with\\_heatmap.png](https://github.com/minimav/plotly-football-pitch/blob/main/images/basic_pitch_with_heatmap.png) (Accessed: 27 October 2025).

ML-KU Leuven (n.d.) *socceraction: Valuing actions in soccer using VAEP and xT* [online]. Available at: <https://github.com/ML-KULeuven/socceraction> (Accessed: 9 December 2025).

Pleuler, D. (n.d.) *Soccer analytics handbook* [online]. Available at: [https://github.com/devinpleuler/analytics-handbook/blob/master/soccer\\_analytics\\_handbook.ipynb](https://github.com/devinpleuler/analytics-handbook/blob/master/soccer_analytics_handbook.ipynb) (Accessed: 10 December 2025).

Rowlinson, A. (n.d.) *mplsoccer: Python library for football pitch plotting and spatial analysis* [online]. Available at: <https://github.com/andrewRowlinson/mplsoccer> (Accessed: 13 November 2025).

Soccermatics (n.d.) *Expected Threat (xT) model tutorial* [online]. Available at: [https://soccermatics.readthedocs.io/en/latest/gallery/lesson4/plot\\_ExpectedThreat.html](https://soccermatics.readthedocs.io/en/latest/gallery/lesson4/plot_ExpectedThreat.html) (Accessed: 21 November 2025).

Soccermatics (n.d.) *Soccermatics: Football analytics and modelling tutorials* [online]. Available at: <https://github.com/soccermatics/Soccermatics> (Accessed: 10 December 2025).

Szeyeung (n.d.) *Football prediction by XGBoost* [online]. Available at: <https://www.kaggle.com/code/szeyeung/football-prediction-by-xgboost> (Accessed: 14 November 2025).

Webster, E. (n.d.) *Application of the VAEP framework with Wyscout event data* [online]. Available at: [https://github.com/eddwebster/football\\_analytics/blob/master/notebooks/5\\_data\\_analysis\\_and\\_projects/action\\_value\\_frameworks/vaep/Application%20of%20VAEP%20Framework%20with%20Wyscout%20Event%20Data.ipynb](https://github.com/eddwebster/football_analytics/blob/master/notebooks/5_data_analysis_and_projects/action_value_frameworks/vaep/Application%20of%20VAEP%20Framework%20with%20Wyscout%20Event%20Data.ipynb) (Accessed: 9 December 2025).

XGBoost Developers (n.d.) *XGBoost documentation: Scalable machine learning system* [online]. Available at: <https://xgboost.readthedocs.io> (Accessed: 13 November 2025).