National College of Ireland

Hunger Hotspots Prediction: Evaluating the Effectiveness of Data-Driven Prediction for Early Crisis Identification.

MSc Research Project
Msc in Data Analytics

Dnyanesh Mohan Bhonde
Student ID: X23270802

School of Computing
National College of Ireland

Supervisor: Prof.Rejwanul Haque

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Dnyanesh Mohan Bhonde
Student ID:	x23270802
Programme:	MSc in Data Analytics
Year:	2025
Module:	MSc Research Project
Supervisor:	Rejwanul Haque
Submission Due Date:	11/08/2025
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	10

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:	Dnyanesh Mohan Bhonde
Date:	13th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Dnyanesh Mohan Bhonde
x23270802

1 Introduction

This Configuration manual helps the user to replicate the research project "Hunger Hotspots Prediction: Evaluating the Effectiveness of Data-Driven Prediction for Early Crisis Identification." The manual explain the storage, hardware and software requirements, programming languages, tools and system setup used in the development of the research project.

2 System Configuration

2.1 Storage

The research project dataset is stored in an online Google Drive as shown in the figure. There is a folder "Thesis dataset" that contains Wfp-hungermap.CSV and clean data "cleaned_dataset.csv. The data is stored on cloud-based storage to open it easily, share, and update. It also helps to keep the data safe, access it from other devices and work well with other tools for data analysis.

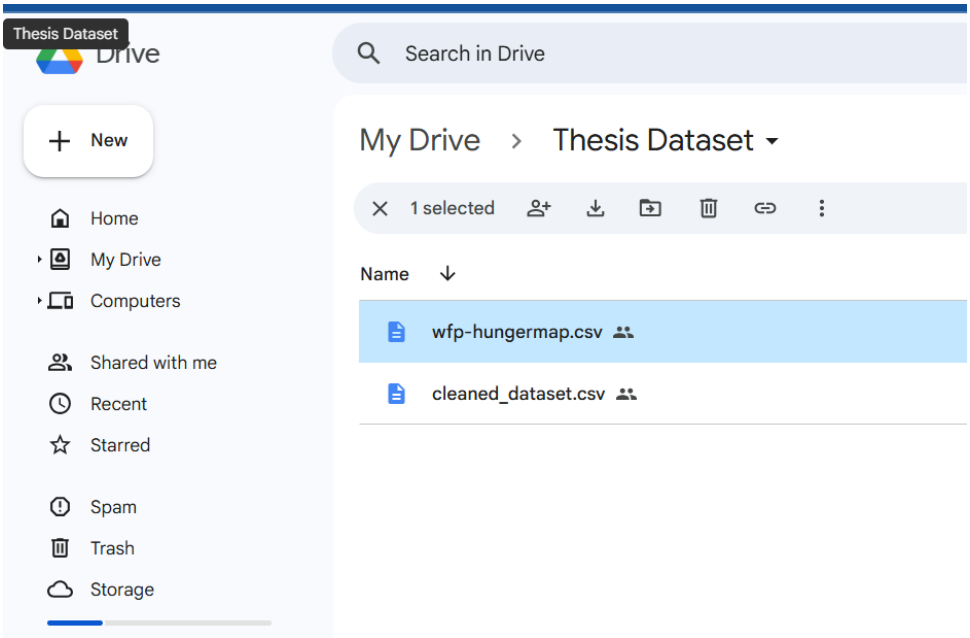

Figure 1: Cloud storage folder containing project datasets

2.2 Hardware Specification

The research project is developed on a computer with the following specifications.

Table 1: Device Specifications

Specification	Details
Device Name	LAPTOP-NP0TRVVQ
Processor	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (2.40 GHz)
Installed RAM	8.00 GB (7.85 GB usable)
System Type	64-bit operating system, x64-based processor

The specification in table 1 gives an efficient performance balance that is suitable for data processing and analytical applications requiring compute.

2.3 Software Specification

2.3.1 Operating System

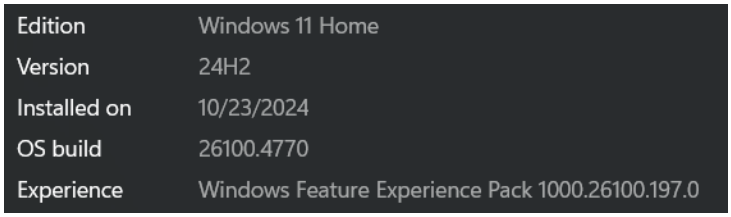

Figure 2: Operating system information of the device

2.3.2 Tools

Tool	Purpose
`Jupyter Notebook`	Interactive code execution
`Python`	Core programming language
`VSCode`	Code development environment
`Google Drive`	Cloud file storage
`Tableau`	Advanced data visualization
`Miniconda`	Environment and dependency management

Table 2: Software Tools Used in the Project

Table 2 shows various tools are used to provide specific purpose in the project. Jupyter Notebook is used to for code execution, while python is used as the core programming language. VS Code provides the development, and Google Drive is used for cloud file storage. Tableau gives advanced data visualization, and Miniconda handles the environment and dependency management

2.3.3 Python Environment Setup

Figure 3 shows that the project is running in the base environment of miniconda using python version 3.12.9. This environment helps to manage all necessary libraries and dependencies needed for execution of the code.

Figure 3: Base environment using Python 3.12.9 in Miniconda

3 Implementation Setup

This section gives an overview of the implementation setup such as collecting data, storing in Google Drive, code implementation, and libraries.

3.1 Data Collection

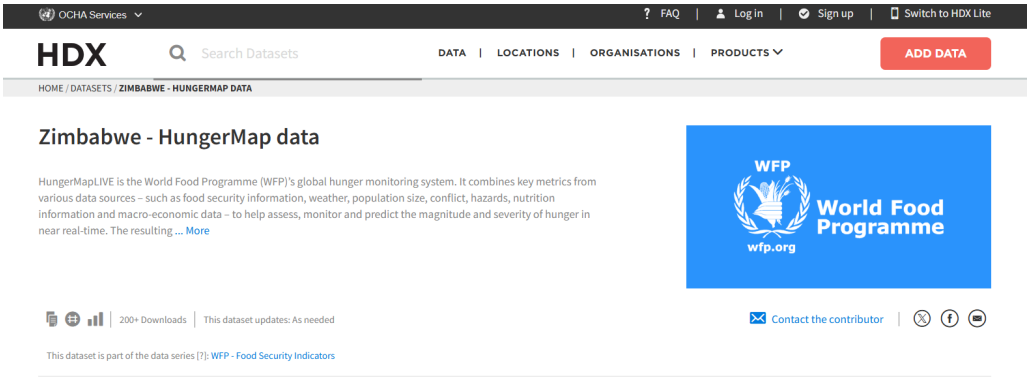

Figure 4: Dataset Source from the HDX platform

Figure 4 shows dataset use in the research is collected from the World Food Programme's HungerMapLive platform from the HDX(Humanitarian Data Exchange) Humdata.org (2024). It gives details about the hunger-related indicators from Zimbabwe, prevalence, population, and food insecurity metrics. This helps to monitor and predict the hunger hotspot across the regions.

3.2 Libraries

Library	Use (3 Words)
`pandas`	Data manipulation, analysis
`numpy`	Numerical array operations
`matplotlib`	Static data visualization
`seaborn`	Statistical plot styling
`datetime`	Date-time calculations
`sklearn`	Machine learning tools
`xgboost`	Gradient boosting regression
`tensorflow`	Deep learning models
`random`	Random number generation

Table 3: Libraries Used and Their Main Purpose

Table 3 gives the key information of python libraries used in the project. Each row of lists of library name alongside their primary function is summarized. It contains the libraries for data manipulation, visualization and machine learning. The table help to clarify the role of each library in the data science pipeline. This format improves the readability and supports quick reference during documentation or presentations Sundaram et al. (2023). Figure 5 shows the libraries that are imported.

```
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import matplotlib.pyplot as plt
import tensorflow as tf
from datetime import timedelta
import random
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Figure 5: Loading a CSV file from Google Drive using Pandas

3.3 Load Dataset

This section shows the dataset load from the Google Drive using the pandas library and it read CSV file into data frame for anlysis and preprocessing.

```python
import pandas as pd
url = 'https://drive.google.com/uc?id=1k4xEnuNFp8J6UYBNxnAi9RWtEg55oKnj'
df = pd.read_csv(url)
```

Figure 6: Loading a CSV file from Google Drive using Pandas

Figure 7 shows the first five rows of the dataset, which shows that first row is a metarow and has columns like Countrycode, adminzone, prevalence, etc.

	countrycode	countryname	adminone	adminlevel	date	datatype	indicator name	population	prevalence
0	#country+code	#country+name	#adm1+name	#meta+adminlevel	#date	#data+type	#indicator+name	#population	#indicator+prevalence
1	ZWE	Zimbabwe	NaN	national	2025-01-15	SURVEY	fcs	4501062	0.2965325928707062
2	ZWE	Zimbabwe	NaN	national	2025-01-15	SURVEY	rcsi	8885207	0.5853627040942525
3	ZWE	Zimbabwe	Bulawayo	subnational	2025-01-15	SURVEY	total	665940	NaN
4	ZWE	Zimbabwe	Bulawayo	subnational	2025-01-15	SURVEY	fcs	94628	0.1420968856053098

Figure 7: Dataset Preview

3.4 Data Preprocessing

Figure 8 shows the data preprocessing on the dataframe. It clean the data by converting types, handling missing values, and normalizing column names and it finally cleaned data to csv file.

```python
def data_preprocessing(df):
    df= df.iloc[1:].reset_index(drop=True)# remove the first metarow
    df['date'] = pd.to_datetime(df['date'], errors='coerce') # convert the date into datetime format
    df['population'] = pd.to_numeric(df['population'], errors='coerce')# convert the population column datatype string to numeric
    df['prevalence'] = pd.to_numeric(df['prevalence'], errors='coerce')# convert the prevalence column datatype string to numeric
    df['adminone'] = df['adminone'].fillna('Unknown')# fill null value with Unknown
    df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_').str.replace(r'[^\w\s]', '', regex=True)# Normalize column
    df['prevalence'] = df['prevalence'].fillna(df['prevalence'].mean())# fill missing value using mean
    print(df.head(5))
    print(df.isnull().sum())
    df.to_csv(r"C:\Users\dnyan\Desktop\clean data\cleandata.csv",index=False)
data_preprocessing(df)
```

Figure 8: Data preprocessing

3.5 Feature Engineering

```python
def FeatureEngineering(df):
    df = df.copy()
# label-encode
    for col in ['adminone', 'indicator_name']:
        if col in df.columns:
            df[col] = LabelEncoder().fit_transform(df[col].astype(str))
    # numeric conversions
    df['population'] = pd.to_numeric(df.get('population'), errors='coerce')
    df['adminlevel'] = pd.to_numeric(df.get('adminlevel'), errors='coerce')
    df['adminone']   = pd.to_numeric(df.get('adminone'),   errors='coerce')
    # date features
    if 'date' in df.columns:
        df['date']    = pd.to_datetime(df['date'], errors='coerce')
        df['year']    = df['date'].dt.year
        df['month']   = df['date'].dt.month
        df['quarter'] = df['date'].dt.quarter
    # population scaling & log
    if 'population' in df.columns:
        df['population_scaled'] = StandardScaler().fit_transform(df[['population']])
        df['log_population']    = np.log1p(df['population'])
    # one-hot encode if present
    dummy_cols = [c for c in ['countrycode', 'datatype'] if c in df.columns]
    if dummy_cols:
        df = pd.get_dummies(df, columns=dummy_cols, drop_first=True)
    # drop rows required for interaction terms
    df = df.dropna(subset=['population', 'adminlevel'])
    # interaction features
    df['pop_x_adminlevel'] = df['population'] * df['adminlevel']
    if {'adminone', 'indicator_name'}.issubset(df.columns):
        df['zone_x_indicator'] = df['adminone'] * df['indicator_name']
    return df
df = FeatureEngineering(df)
```

Figure 9: Feature Engineering

Figure 9 shows a python function that prepares the dataset for modelling. It convert date columns, encodes categorical features, scales numerical data and creates interaction terms. The transformed data is ready for input into machine learning approaches.

3.6 Model Implementation

3.6.1 XGBoost Model

Figure 10 code train the XGBoost model regression to predict hunger prevalence using feature and split the data, fit the model,make predictions, and evaluate the performance with MSE,RMSE,and R2.

```python
# features and target variable
features_xgb = ['adminone', 'indicator_name', 'population']
X_xgb = df[features_xgb]
y_xgb = df['prevalence']

# Split the data
X_train_xgb, X_test_xgb, y_train_xgb, y_test_xgb = train_test_split(X_xgb, y_xgb, test_size=0.2, random_state=42)

# fit the model
model_xgb = XGBRegressor(n_estimators=100,learning_rate=0.05,max_depth=3,reg_alpha=1,reg_lambda=1,subsample=0.7,
                         colsample_bytree=0.7,
                         random_state=42
)
model_xgb.fit(X_train_xgb, y_train_xgb)
# prediction to make
y_pred_xgb = model_xgb.predict(X_test_xgb)
# metrics
mse = mean_squared_error(y_test_xgb, y_pred_xgb)
rmse = np.sqrt(mse)
r2 = r2_score(y_test_xgb, y_pred_xgb)
print("XGBoost  Evaluation:")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"R²: {r2:.4f}")
```

Figure 10: XGBoost model

3.6.2 Random Forest Regression Model

The figure 11 shows the Random Forest Regression to predict the hunger crisis using selected features. It includes steps like data transformation and one hot encoding before model training.

```python
# feature selection and target variable
features = ['countryname', 'adminlevel', 'population', 'year', 'month']
X = df[features]
y = df['prevalence']
# spliting for testing and training
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X, y, test_size=0.2, random_state=42)
# model initialization and training
model_rf = RandomForestRegressor(n_estimators=100, random_state=42)
model_rf.fit(X_train_rf, y_train_rf)
# prediction
y_pred_rf = model_rf.predict(X_test_rf)
# evaluation metrics
mse = mean_squared_error(y_test_rf, y_pred_rf)
r2 = r2_score(y_test_rf, y_pred_rf)
rmse = np.sqrt(mse)
print(" Random Forest Evaluation:")
print(f"Mean Squared Error: {mse:.4f}")
print(f"R² Score: {r2:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
```

Figure 11: Random Forest model

3.6.3 LSTM Model

Figure 13 shows the LSTM code to predict the hunger prevalence trends in Zimbabwe. It uses the sequence-based data from normalized prevalence and population values for time series forecasting. This model was also used to forecast prevalence for 5 next five days using the last observed data.

```
df_zim = df[df['countrycode'] == 'ZWE'].copy()
df_zim = df_zim.groupby('date').agg({
    'prevalence': 'mean',
    'population': 'sum'
}).reset_index().sort_values('date')

scaler = MinMaxScaler()
df_zim[['prevalence', 'population']] = scaler.fit_transform(df_zim[['prevalence', 'population']])
def create_sequences(data, seq_len):
    X, y = [], []
    for i in range(len(data) - seq_len):
        X.append(data[i:i+seq_len, :])
        y.append(data[i+seq_len, 0])
    return np.array(X), np.array(y)

X, y = create_sequences(df_zim[['prevalence', 'population']].values, sequence_length=12)
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, epochs=30, batch_size=8, verbose=1)
y_pred_lstm = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred_lstm)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred_lstm)
future_steps = 5
last_sequence = data[-sequence_length:]
for _ in range(future_steps):
    pred = model.predict(last_sequence.reshape(1, sequence_length, 2), verbose=0)[0][0]
    next_step = np.array([pred, last_sequence[-1, 1]])
    last_sequence = np.vstack([last_sequence[1:], next_step])
```

Figure 12: LSTM model

3.6.4 Linear Regression Model

Figure 13 the linear regression code prepares the data by creating new date and population-related features. It changes text columns into numbers and scales the data for better model performance. Finally model was trained to predict hunger hotspot prevalence.

```
feature_cols = [
    'countryname', 'adminone', 'adminlevel', 'indicator_name',
    'log_population', 'year', 'month', 'quarter',
    'pop_x_adminlevel', 'zone_x_indicator'
] + [col for col in df.columns if col.startswith('countrycode_') or col.startswith('datatype_')]
#  Drop rows with missing target or features
df_model = df.dropna(subset=feature_cols + ['prevalence'])
#  Define input and target
X = df_model[feature_cols]
y = df_model['prevalence']
# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
#  Split data
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
#  Train model
model = LinearRegression()
model.fit(X_train, y_train)
# Evaluate on test set
y_pred_lr = model.predict(X_test)
mse_test = mean_squared_error(y_test, y_pred_lr)
rmse_test = np.sqrt(mse_test)
r2_test = r2_score(y_test, y_pred_lr)
# Print results
print(" Linear Performance:")
print(f"Test MSE: {mse_test:.4f}")
print(f"Test RMSE: {rmse_test:.4f}")
print(f"Test R²: {r2_test:.4f}")
```

Figure 13: Linear Regression Model

3.6.5 Decision Tree Regression Model

Figure 14 shows the model was trained using a decision tree regressor on labeled numerical data. Data was split into training and testing, and the model learned from the training portion. it was evaluated on the test set using MSE, RMSE, and R2 metrics.

```
# features and target
X = df.drop(columns=['prevalence']).select_dtypes(include=[np.number])
y = pd.to_numeric(df['prevalence'], errors='coerce')
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# train the Decision Tree Regressor
regressor = DecisionTreeRegressor(max_depth=5, random_state=42)
regressor.fit(X_train, y_train)
# Make predictions
y_pred_dt = regressor.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred_dt)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred_dt)
# Print evaluation metrics
print(f"Decision Tree evaluation:")
print(f" Mean Squared Error (MSE): {mse:.4f}")
print(f" Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f" R² Score: {r2:.2f}")
```

Figure 14: Decision Tree Model

3.6.6 Hybrid Stacked Model

Figure 15 shows code to build a hybrid stacked model using predictions from the five models: XGBoost, RandomForest, LSTM, Linear Regression, and Decision Tree. It frist align in the prediction lengths and use them as input feature for new meta model. Linear regression was trained on the predictions to learn how model performs after combining them. The meta model output is evaluated using metrics like MSE and RMSE and R2.

```python
# Ensure all predictions are the same length
min_len = min(
    len(y_pred_xgb),
    len(y_pred_rf),
    len(y_pred_lstm),
    len(y_pred_lr),
    len(y_pred_dt),
    len(y_test)
)
# Truncate predictions and true labels to common length
y_true = y_test[:min_len]
pred_xgb = np.array(y_pred_xgb).flatten()[:min_len]
pred_rf = np.array(y_pred_rf).flatten()[:min_len]
pred_lstm = np.array(y_pred_lstm).flatten()[:min_len]
pred_lr = np.array(y_pred_lr).flatten()[:min_len]
pred_dt = np.array(y_pred_dt).flatten()[:min_len]
#Create the meta-model input
X_meta = np.column_stack([pred_xgb, pred_rf, pred_lstm, pred_lr, pred_dt])
#  Train the meta-model
meta_model = LinearRegression()
meta_model.fit(X_meta, y_true)
# Predict and evaluate
meta_pred = meta_model.predict(X_meta)
mse = mean_squared_error(y_true, meta_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_true, meta_pred)
print("\n Hybrid Model Performance:")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"R² Score: {r2:.4f}")
```

Figure 15: Hybrid (Stacked) Model

3.7 Tableau Visualization

Set up for Tableau Visualization:
1 Installed Tableau.
2 Connecting data Open the tableau and select Connect to file"Text". choose the data CSV file output predicted by the model. Ensure the proper data type of the column.
3 Create the Visualization.

References

Humdata.org (2024). Zimbabwe - HungerMap data — Humanitarian Dataset — HDX, `https://data.humdata.org/dataset/wfp-hungermap-data-for-zwe`.

Sundaram, J., Gowri, K., Devaraju, S., Gokuldev, S., Jayaprakash, S., Anandaram, H., Manivasagan, C. and Thenmozhi, M. (2023). An exploration of python libraries in machine learning models for data science, *Handbook of Research on Advanced Practical Approaches to Deepfake Detection and Applications*, IGI Global.
URL: *https://www.igi-global.com/chapter/an-exploration-of-python-libraries-in-machine-learning-models-for-datascience/330569*