National College of Ireland

# Configuration Manual

MSc Research Project
Master of Science in AI for Business

Felipe Rojas
Student ID: X23438941

School of Computing
National College of Ireland

Supervisor: Faithful Onwuegbuche

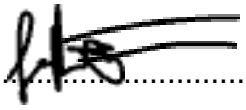
**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Felipe Rojas |
| **Student ID:** | X23438941 |
| **Programme:** | Msc in AI for Business **Year:** 2025 |
| **Module:** | Practicum |
| **Supervisor:** | Faithful Onwuegbuche |
| **Submission Due Date:** | 11/08/2025 |
| **Project Title:** | Predicting Dietary B12 Deficiency in Vegetarians Using Interpretable Models |
| **Word Count:** | 1398 **Page Count:** 11 |

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:** 

**Date:** 08/08/2025

**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

Felipe Rojas
X23438941

# 1 Introduction

The configuration manual delivers a brief technical summary of the data pipeline and machine learning experiments used in the MSc Research Project. The project analysed NHANES data from 2017 to 2023 to develop predictions for vitamin B12 deficiency through dietary intake and demographic variables and custom vegetarian flag indicators.

# 2 System Configuration

## 2.1 Hardware Requirements

1. OS: Windows 10 / macOS / Linux
2. RAM: 8 GB minimum
3. Processor: i5 or equivalent

## 2.2 Software Requirements

- Python ≥ 3.10
- Jupyter Notebook
- pandas
- numpy
- scikit-learn
- imbalanced-learn
- matplotlib / seaborn
- shap
- lime

Install via Anaconda (recommended) or pip. To install core libraries:

```
pip install pandas numpy scikit-learn imbalanced-learn matplotlib seaborn shap lime
```

It is advised to run the code using Jupyter Notebook, either locally or in Google Colab.

# 3 Folder Structure

```
project-root/
│
├── notebooks/
│   ├── STEP 1. BUILD INTAKES AND FLAG.ipynb
│   ├── STEP 2. RENAME AND ROUND.ipynb
│   ├── STEP 3. EDA IMBALANCED DATA.ipynb
│   ├── STEP 4. ML MODEL IMBALANCED DATA.ipynb
│   ├── STEP 5. DOWNSAMPLE DATA AND SMOTE.ipynb
│   ├── STEP 6. EDA BALANCED DATA.ipynb
│   └── STEP 7. ML MODEL BALANCED DATA.ipynb
│
├── data/
│   ├── vegetarian_flags_merged.csv
│   ├── b12_clean_final_with_age_rda.csv
│   ├── b12_nutrients_cleaned_rounded.csv
│   ├── b12_smote_tagged_cleaned.csv
│   └── NHANES/
│       └── CYCLES
│
│
```

# 4 Implementation

## 4.1 Step 1, Build Intakes and Flags

### 4.1.1 Loading NHANES Intake Files

Integrates Day 1 and Day 2 dietary intake CSV files for each NHANES cycle (2017–2020, 2021–2023) and then tags them with the corresponding cycle. This ensures the dataset includes all intake records required for downstream processing.

```python
import pandas as pd

### STEP 1 – Load Day 1 and Day 2 Intake Files for Each Cycle ###

# Cycle 1: 2017–2020
intake_day1_2017 = pd.read_csv("/Users/feliperojas/Documents/Class/Practicum/Final Project/Datasets/NHANES/CONVERTED/2017-2020/P_DR1IFF.csv")
intake_day2_2017 = pd.read_csv("/Users/feliperojas/Documents/Class/Practicum/Final Project/Datasets/NHANES/CONVERTED/2017-2020/P_DR2IFF.csv")

# Cycle 2: 2021–2023
intake_day1_2021 = pd.read_csv("/Users/feliperojas/Documents/Class/Practicum/Final Project/Datasets/NHANES/CONVERTED/DR1IFF_L.csv")
intake_day2_2021 = pd.read_csv("/Users/feliperojas/Documents/Class/Practicum/Final Project/Datasets/NHANES/CONVERTED/DR2IFF_L.csv")

# Add cycle for traceability
intake_day1_2017["cycle"] = "2017-2020"
intake_day2_2017["cycle"] = "2017-2020"
intake_day1_2021["cycle"] = "2021-2023"
intake_day2_2021["cycle"] = "2021-2023"
```

Figure 1: Data Import – NHANES Intake Files

### 4.1.2 Merging Day 1 and Day 2 Data

The script combines Day 1 intake data from all cycles with Day 2 intake data from all cycles by disregarding the original indexing to generate two combined datasets. The process prepares the data for a two-day merging operations while keeping all participant records intact.

```
# Merge all Day 1 and Day 2 data
merged_day1 = pd.concat([intake_day1_2017, intake_day1_2021], ignore_index=True)
merged_day2 = pd.concat([intake_day2_2017, intake_day2_2021], ignore_index=True)

# Merge Day 1 and Day 2 on SEQN and cycle (no suffixes)
merged = pd.merge(merged_day1, merged_day2, on=["SEQN", "cycle"])

# Nutrients of interest with proper NHANES column names
nutrients = {
    "vb12": ("DR1IVB12", "DR2IVB12"),
    "kcal": ("DR1IKCAL", "DR2IKCAL"),
    "foldfe": ("DR1IFDFE", "DR2IFDFE"),
    "iron": ("DR1IIRON", "DR2IIRON"),
    "phos": ("DR1IPHOS", "DR2IPHOS"),
    "choln": ("DR1ICHL", "DR2ICHL")
}
```

Figure 2: Data Consolidation – Merge Day 1 and Day 2

### 4.1.3 Merging Vegetarian Status

The pre-processed vegetarian flag dataset gets integrated into the combined NHANES intake data. The classification serves as an important tool for dividing the population into segments so the analysis can focus on vegetarian dietary patterns.

```
vegetarian_df = pd.read_csv("/Users/feliperojas/vegetarian_flags_merged.csv")  # pre-processed
merged = pd.merge(merged, vegetarian_df[["SEQN", "vegetarian_flag"]], on="SEQN", how="left")
merged["vegetarian_flag"] = merged["vegetarian_flag"].fillna(0).astype(int)
```

Figure 3: Metadata Merge – Vegetarian Flag Integration

### 4.1.4 Feature Engineering: Assigning RDA and Deficiency Flags

The National Institutes of Health (2022) defines Recommended Dietary Allowance (RDA) values for Vitamin B12 according to participant age groups. This step establishes the reference point for determining deficiency.

```
# RDA by age
def get_b12_rda(age):
    if age < 1:
        return 0.5
    elif age < 4:
        return 0.9
    elif age < 9:
        return 1.2
    elif age < 14:
        return 1.8
    else:
        return 2.4

merged["rda_b12"] = merged["Age"].apply(get_b12_rda)
```

Figure 4: Feature Engineering – Assign RDA for B12

### 4.1.5 Creating B12 Deficiency Flag

The model generates a binary indicator for deficiency through participant B12 intake averages that fall below their age-specific RDA. The model training depends on this variable as its main target.

```
merged["b12_deficiency"] = (merged["avg_vb12"] < merged["rda_b12"]).astype(int)
```

Figure 5: Label Generation – B12 Deficiency Classification

## 4.2 Step 2, Rename and Round

The dataset receives standardization through column name renaming and nutrient value rounding while validated participant IDs (SEQNs) become the only included data points. The dataset becomes ready for dependable analysis through this process.

```
# Load legacy SEQNs to match
legacy_df = pd.read_csv("/Users/feliperojas/Documents/Class/Practicum/Final Project/Datasets/NHANES/CONVERTED/b12_nutrients_cleaned_rounded.csv")
legacy_seqns = legacy_df["SEQN"].unique()
```

Figure 6, Step 2: Rename and Round

## 4.3 Step 3, EDA on Imbalanced Data

### 4.3.1 Class Distribution

Shows the distribution between vegetarian and non-vegetarian participants. The model bias may be affected by the dietary population imbalances which become apparent through this analysis.

```
import matplotlib.pyplot as plt

# Count vegetarians vs non-vegetarians
counts = df['vegetarian_flag'].map({0: "Non-Vegetarian", 1: "Vegetarian"}).value_counts()
colors = ['#1f77b4', '#ff7f0e']

# Calculate percentages
percentages = counts / counts.sum() * 100

# Create bar plot
plt.figure(figsize=(6, 4))
bars = counts.plot(kind='bar', color=colors)
plt.title("Class Distribution: Vegetarian vs. Non-Vegetarian", fontsize=14)
plt.ylabel("Number of Participants", fontsize=12)
plt.xlabel("Diet Type", fontsize=12)
plt.xticks(rotation=0, ha='center', fontsize=12)
plt.yticks(fontsize=10)

# Add labels above bars with counts + percentages
for i, (value, pct) in enumerate(zip(counts, percentages)):
    plt.text(i, value + 0.01 * max(counts),
             f"{value} ({pct:.1f}%)",
             ha='center', fontsize=10)

plt.tight_layout(pad=2.0)
plt.show()
```

Figure 7, EDA: Vegetarian vs Non-Vegetarian Distribution

### 4.3.2 Class Distribution Plots

The chart displays the distribution of non-vegetarians against vegetarians across various features. The data requires resampling techniques because the classes show substantial imbalance.

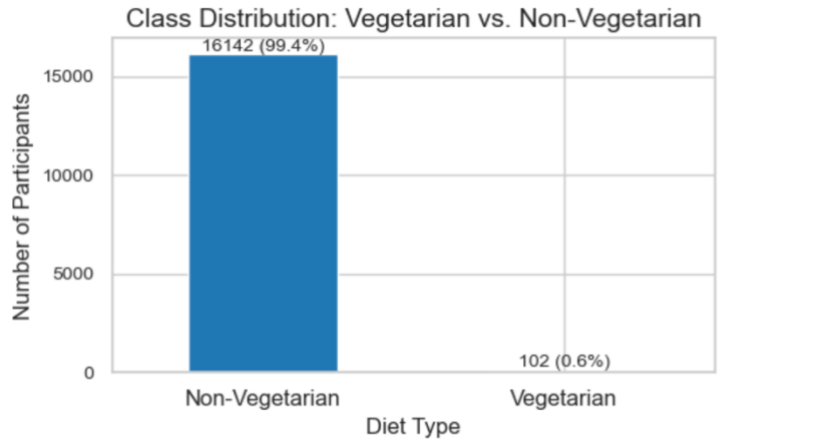



Figure 8, EDA: Class Distribution Imbalance

### 4.3.3 Nutrient Distribution by Diet

The figure shows how B12, folate, iron, phosphorus and choline levels distribute between different dietary groups. The analysis reveals patterns of nutrient distribution that relate to vegetarian dietary choices.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Ensure vegetarian_label is mapped
df['vegetarian_label'] = df['vegetarian_flag'].map({0: "Non-Vegetarian", 1: "Vegetarian"})

# Define nutrients and styling
nutrients = ['avg_b12', 'b12_density', 'folate_avg', 'iron_avg', 'phosphorus_avg', 'choline_avg']
titles = [
    "Average B12 by Diet Type",
    "B12 Density by Diet Type",
    "Folate by Diet Type",
    "Average Iron by Diet Type",
    "Average Phosphorus by Diet Type",
    "Average Choline by Diet Type"
]
y_labels = [
    "Average B12 (mcg)",
    "B12 per 1000 kcal",
    "Folate (mcg)",
    "Iron (mg)",
    "Phosphorus (mg)",
    "Choline (mg)"
]
palette = {'Non-Vegetarian': '#1f77b4', 'Vegetarian': '#ff7f0e'}
x_label = "Diet Type"

# Loop through plots
for nutrient, title, y_label in zip(nutrients, titles, y_labels):
    plt.figure(figsize=(7, 5))
    sns.boxplot(
        data=df,
        x="vegetarian_label",
        y=nutrient,
        hue="vegetarian_label",
        palette=palette,
        legend=False
    )
    plt.title(title, fontsize=14)
    plt.xlabel(x_label, fontsize=12)
    plt.ylabel(y_label, fontsize=12)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.tight_layout()
    plt.show()
```

Figure 9, EDA: Nutrient Distribution by Diet Type

## 4.4 Step 4, Random Forest on Imbalanced Data

### 4.4.1 Model Training

The model evaluates a Random Forest model without resampling to determine baseline performance metrics.

```
# 1. Load data
df = pd.read_csv("/Users/feliperojas/Documents/Class/Practicum/Final Project/Datasets/NHANES/CONVERTED/b12_nutrients_cleaned_rounded.csv")

# 2. Filter only vegetarian participants
df_veg = df[df["vegetarian_flag"] == 1]

# 3. Separate features and target
X = df_veg.drop(columns=["b12_deficiency", "vegetarian_flag", "rda_b12", "SEQN"])
X = X.select_dtypes(include=np.number)  # Keep only numeric columns
y = df_veg["b12_deficiency"]


# Optional: check shape and column names
print("Features shape:", X.shape)
print("Target distribution:\n", y.value_counts())

# 4. Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, test_size=0.2, random_state=42
)

# 5. Train Random Forest model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

Figure 10, Random Forest Classifier on Original Data

### 4.4.2 Evaluation Metrics

The model presents AUC together with accuracy and precision and recall and F1 scores. The results will serve as a reference point to evaluate the performance of the balanced data set.

```
Features shape: (102, 9)
Target distribution:
 b12_deficiency
1    68
0    34
Name: count, dtype: int64
📊 Evaluation Metrics (All Original Features):
AUC: 0.9897959183673469
Balanced Accuracy: 0.9642857142857143
Accuracy: 0.9523809523809523
Precision: 1.0
Recall: 0.9285714285714286
F1 Score: 0.9629629629629629

Classification Report:
              precision    recall  f1-score   support

           0       0.88      1.00      0.93         7
           1       1.00      0.93      0.96        14

    accuracy                           0.95        21
   macro avg       0.94      0.96      0.95        21
weighted avg       0.96      0.95      0.95        21
```

Figure 11, Performance – AUC, Precision, Recall, F1

## 4.5 Step 5, SMOTE Resampling

### 4.5.1 Downsampling

The dataset size reduction occurs before applying SMOTE to enhance computational efficiency. The class distribution remains unaltered throughout the process.

```
# === Step 1: Reduce to 5,000 randomly for tractability ===
df_sampled = df.sample(n=5000, random_state=42).reset_index(drop=True)
```

Figure 12, Preprocessing: Downsampling Before SMOTE

### 4.5.2 SMOTE Balancing

The SMOTE algorithm generates synthetic data to balance the deficient and non-deficient cases (Chawla et al., 2002). The model becomes better at detecting the minority class after this step.

```
# === Step 3: Apply SMOTE ===
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X_original, y_original)
```

Figure 13, Preprocessing, Apply Smote

## 4.6 Step 6, EDA on Balanced Data

The process in this step duplicates the EDA analysis from Step 3. The process uses b12_smote_tagged_cleaned.csv as its input file. Run the same code cells again with this new balanced dataset to check class balance and nutrient distributions.

## 4.7 Step 7, Random Forest on Balanced Data

The Random Forest model receives retraining and evaluation using the balanced dataset to achieve better minority class prediction sensitivity.

```
📊 Evaluation Metrics (All Original Features):
AUC: 0.9971770334928229
Balanced Accuracy: 0.9877158310796754
Accuracy: 0.9869083585095669
Precision: 0.9764705882352941
Recall: 0.992822966507177
F1 Score: 0.9845788849347569

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       575
           1       0.98      0.99      0.98       418

    accuracy                           0.99       993
   macro avg       0.99      0.99      0.99       993
weighted avg       0.99      0.99      0.99       993
```

Figure 14, RF final model results

# 5 Explainability

## 5.1 SHAP

SHAP values (Lundberg & Lee, 2017) enable interpretability by showing which features most affect model predictions through ranking.

```
import shap
import matplotlib.pyplot as plt

# Step 1: Rename columns
X_test_renamed = X_test.copy()
X_test_renamed.columns = [label_dict.get(col, col) for col in X_test.columns]

# Step 2: SHAP explainer
explainer = shap.Explainer(model, X_train)
shap_values = explainer(X_test)

# Step 3: Get SHAP values for class 1 (Deficient)
class1_shap_values = shap_values.values[:, :, 1]

# ✅ Step 4: Bar plot with correct size — set *before* shap.summary_plot
shap.summary_plot(class1_shap_values, X_test_renamed, plot_type="bar", max_display=10, show=False)

# Now modify the current figure
plt.gcf().set_size_inches(10, 6)
plt.tight_layout()
plt.subplots_adjust(left=0.35)  # space for long feature names
plt.show()

# ✅ Optional: Violin-style summary
shap.summary_plot(class1_shap_values, X_test_renamed, max_display=10)
```

Figure 15, SHAP: Global Feature Importance

## 5.2 LIME

The model provides individual predictions for selected participants by using local approximations (Lundberg & Lee, 2017) to explain its decision-making process for each case.

```
import lime
import lime.lime_tabular
import numpy as np

# ✅ Step 1: Prepare readable feature names
label_dict = {
    'avg_b12': 'Average Vitamin B12 (µg/day)',
    'b12_density': 'Vitamin B12 per 1000 kcal',
    'avg_kcal': 'Average Energy Intake (kcal)',
    'iron_avg': 'Average Iron (mg)',
    'phosphorus_avg': 'Average Phosphorus (mg)',
    'choline_avg': 'Average Choline (mg)',
    'folate_avg': 'Average Folate (µg)',
    'Age': 'Age',
    'Gender': 'Gender',
    'vegetarian_label': 'Diet Type (Vegetarian/Non-Vegetarian)',
    'vegetarian_flag': 'Diet Type (Vegetarian/Non-Vegetarian)'
}

# ✅ Rename columns for display
X_train_renamed = X_train.copy()
X_test_renamed = X_test.copy()

X_train_renamed.columns = [label_dict.get(col, col) for col in X_train.columns]
X_test_renamed.columns = [label_dict.get(col, col) for col in X_test.columns]

# ✅ Step 2: Initialize LIME explainer
lime_explainer = lime.lime_tabular.LimeTabularExplainer(
    training_data=X_train_renamed.values,
    feature_names=X_train_renamed.columns.tolist(),
    class_names=['Non-Deficient', 'Deficient'],
    mode='classification',
    discretize_continuous=True
)

# ✅ Step 3: Define label mapping
target_labels = {0: "Non-Deficient", 1: "Deficient"}

# ✅ Step 4: Explain selected samples
for i in [7, 30, 40]:
    true_class = y_test.iloc[i]
    print(f"\n🔍 LIME Explanation for Test Sample {i} (True Label: {target_labels[true_class]})")

    lime_exp = lime_explainer.explain_instance(
        data_row=X_test_renamed.iloc[i].values,
        predict_fn=model.predict_proba,
        num_features=10
    )
    lime_exp.show_in_notebook(show_table=True)
```

Figure 16, LIME: Local Explanations

# 6. Notebook and Output Flow

The notebooks use previous output to create a reproducible pipeline. The following section presents a detailed explanation of data movement and step dependencies throughout the process:

- **Step 1** → Generates: **b12_clean_final_with_age_rda.csv**
  Raw NHANES intake files are merged with vegetarian status and B12 RDA assignments.
- **Step 2** → Input: **b12_clean_final_with_age_rda.csv**
  → Output: **b12_nutrients_cleaned_rounded.csv**
  This includes column standardization, rounding of nutrient values, and SEQN filtering.
- **Steps 3, 4** → Input: **b12_nutrients_cleaned_rounded.csv**
  → Output: exploratory plots and Random Forest model (imbalanced).
  These notebooks perform initial EDA and train a model using the original class imbalance.
- **Step 5** → Input: **b12_nutrients_cleaned_rounded.csv**
  → Output: **b12_smote_tagged_cleaned.csv**
  Downsamples the dataset and applies SMOTE to balance the classes.
- **Steps 6, 7 → Input: b12_smote_tagged_cleaned.csv**
  → Output: plots and balanced Random Forest model.
  These notebooks run updated EDA and train the final model on balanced data (vegetarian subset only).

# References

National Institutes of Health, 2022. Vitamin B12 – health professional fact sheet. *Office of Dietary Supplements, U.S. Department of Health & Human Services*. [online] Available at: https://ods.od.nih.gov/factsheets/VitaminB12-HealthProfessional/

Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P., 2002. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, pp.321–357. https://doi.org/10.1613/jair.953

Lundberg, S.M. and Lee, S.-I., 2017. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, pp.4765–4774. https://doi.org/10.48550/arXiv.1705.07874