National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Preeti Sharma
Student ID: 23379031

School of Computing
National College of Ireland

Supervisor: Dr Hicham Rifai

10th December 2025

National College of Ireland

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Preeti Sharma
Student ID:	23379031
Programme:	Data Analytics
Year:	2025
Module:	MSc Research Project
Supervisor:	Dr Hicham Rifai
Submission Due Date:	10th December 2025
Project Title:	Socio-Economic and Demographic Determinants of Unmet Need for Family Planning Among Women of Reproductive Age in India
Word Count:	1004
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:	10th December 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
Socio-Economic and Demographic Determinants of Unmet Need for Family Planning Among Women of Reproductive Age in India

Preeti Sharma
23379031

1 Overview

This configuration manual provides the technical setup required to run the complete analytical workflow for the research project titled **"Socio-Economic and Demographic Determinants of Unmet Need for Family Planning Among Women of Reproductive Age in India"**. It details the hardware specifications, software installations, Python environment, dependencies, and the step-by-step execution of data extraction, transformation, modelling, and evaluation procedures 1.

2 Hardware and Software Requirements

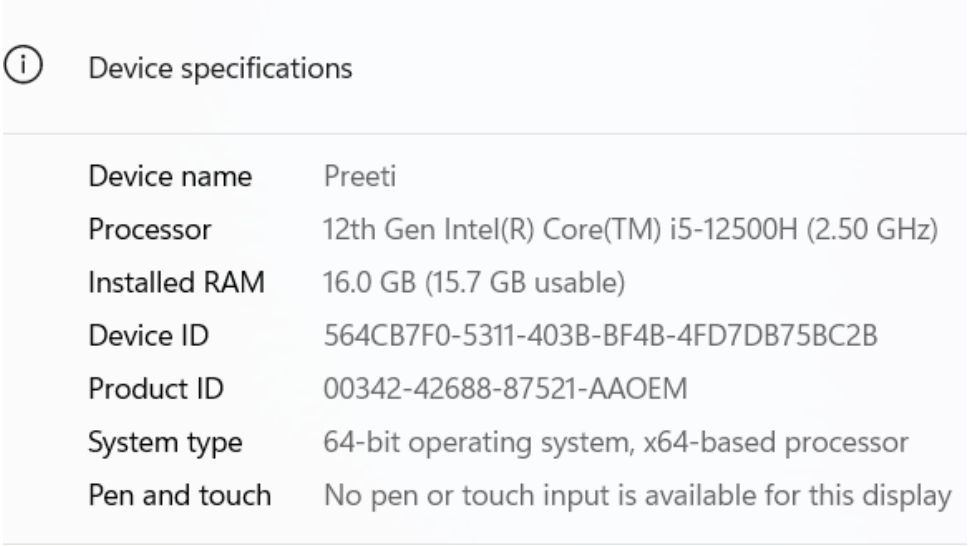

Figure 1: Hardware configuration

2.1 Hardware Requirements

The system configuration used to develop, test, and execute the analytical models for this project is as follows:

- **Operating System:** Windows 11 Home, Version 24H2
- **Processor:** 12th Gen Intel® Core™ i5-12500H (2.50 GHz)
- **Installed RAM:** 16.0 GB (15.7 GB usable)
- **System Type:** 64-bit operating system, x64-based processor
- **Pen and Touch:** No pen or touch input available

2.2 Software Requirements

- **Programming Language:** Python 3.10+
- **IDE / Notebook:** Jupyter Notebook (Anaconda)
- **Data storage:** local server
- **Big Data Libraries:** pandas, numpy, pyreadstat
- **Modelling Libraries:** scikit-learn, statsmodels, xgboost
- **Visualisation Tools:** matplotlib, seaborn
- **Additional Tools:** SPSS/CSV

3 Environment Setup

3.1 Installing Anaconda

Jupyter Notebook setup begins with installing the Anaconda distribution. Since Jupyter is packaged within Anaconda, it becomes available for use immediately after installation. 2:

1. Download Anaconda from `https://www.anaconda.com/products/distribution`
2. Install with default settings.
3. Launch **Jupyter Notebook** from Anaconda.

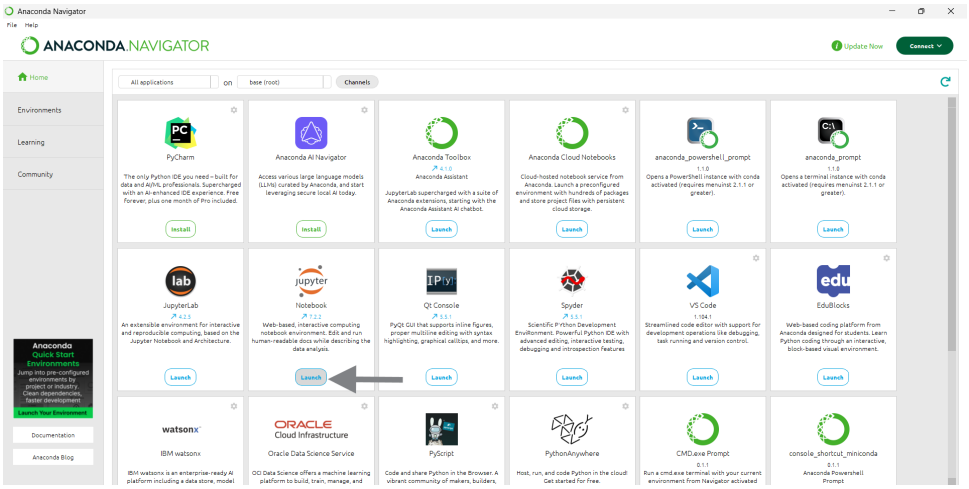

Figure 2: Anaconda application

4 Implementation

1 Data Acquisition

After logging into the IIPS Data Portal `https://iipsdata.ac.in`, I accessed approved project (ID: RQ00003453) and downloaded the required NFHS-5 datasets from the "My Subscribed Datasets" section. Using the download option provided for each file, I saved the ZIP folders locally, extracted them, and placed all recode files into the project's _extracted directory for use in the analysis.

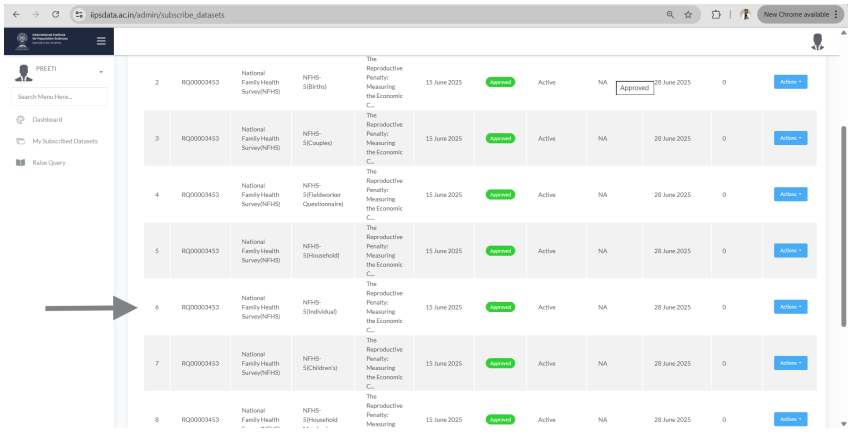

Figure 3: National Family Health Data 5

- Individual Recode (IR)

Files were placed in a dedicated project directory:

`BASE_DIR / _extracted / NFHS-5`

2. Python Script Execution

The following stages were implemented in Python:

3. Importing Required Python Libraries

The following Python libraries were used for data loading, cleaning, statistical modelling, machine-learning implementation, and visualisation throughout this project:

- **Core utilities:** os, glob, zipfile, warnings, re
- **Data manipulation:** pandas, numpy
- **Statistical modelling:** statsmodels, statsmodels.formula.api
- **Survey data reading:** pyreadstat
- **Visualisation:** matplotlib.pyplot, matplotlib.ticker, seaborn
- **Density estimation:** scipy.stats (gaussian_kde)
- **Machine learning models:** scikit-learn (LogisticRegression, RandomForestClassifier), XGBoost (XGBClassifier)
- **ML evaluation metrics:** accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, precision_recall_curve, average_precision_score, roc_curve, auc

4. Folder Setup and Library Configuration

To organise the project files, a base directory was created on the local system, and several sub-folders were generated automatically using Python's `os` library 4. These folders store the extracted NFHS-5 datasets, generated tables, plots, and the exported codebooks.

```
import os
BASE_DIR = r"D:\NCI_COLLEGE PROJECTS\RESEARCH_PROJECT(Aug 2025)"

# Derived folders
EXTRACT_DIR = os.path.join(BASE_DIR, "_extracted")
OUT_DIR     = os.path.join(BASE_DIR, "_outputs")
CODEBOOK_DIR= os.path.join(OUT_DIR, "codebook")
TABS_DIR    = os.path.join(OUT_DIR, "tables")
PLOTS_DIR   = os.path.join(OUT_DIR, "plots")

for d in [EXTRACT_DIR, OUT_DIR, CODEBOOK_DIR, TABS_DIR, PLOTS_DIR]:
    os.makedirs(d, exist_ok=True)

print("Base:", BASE_DIR)
print("Extracted files will be in:", EXTRACT_DIR)
print("Outputs will be in:", OUT_DIR)
```

```
Base: D:\NCI_COLLEGE PROJECTS\RESEARCH_PROJECT(Aug 2025)
Extracted files will be in: D:\NCI_COLLEGE PROJECTS\RESEARCH_PROJECT(Aug 2025)\_extracted
Outputs will be in: D:\NCI_COLLEGE PROJECTS\RESEARCH_PROJECT(Aug 2025)\_outputs
```

Figure 4: Project directory setup

5. NFHS Dataset Setup

The `pyreadstat` library 5 was used to read the NFHS-5 Stata/SAV files along with their metadata, while `pandas` was used to create structured codebooks for variables and value labels. This setup ensures that all input and output files are stored systematically and can be easily accessed throughout the analysis workflow.

```
# Read metadata for IR
_, ir_meta = pyreadstat.read_sav(IR_SAV, metadataonly=True)
# _, cr_meta = pyreadstat.read_sav(CR_SAV, metadataonly=True)
# Robust codebook exporter
def export_codebook(meta, prefix):
    # 1) Variable names & labels
    pd.DataFrame(
        {"name": meta.column_names, "label": meta.column_labels}
    ).to_csv(os.path.join(CODEBOOK_DIR, f"{prefix}_variables.csv"), index=False)

    # 2) Value labels tied to variables
    rows = []
    var_to_labelset = getattr(meta, "variable_value_labels", {}) or {}
    labelsets = getattr(meta, "value_labels", {}) or {}

    for var, ls in var_to_labelset.items():
        # Case A: ls is already a dict of {code: text}
        if isinstance(ls, dict):
            mapping = ls
        else:
            # Case B: ls is a labelset name (str) -> look up in meta.value_labels
            mapping = labelsets.get(ls, {}) if isinstance(ls, str) else {}

        for code, text in (mapping or {}).items():
            rows.append((var, code, text))

    #If no labels are captured for any variable, the code exports the complete label sets as a fallback.
    if not rows and labelsets:If no labels are captured for any variable, the code exports the complete label sets as a fallback.
        for ls_name, mapping in labelsets.items():
            for code, text in mapping.items():
                rows.append((f"[labelset:{ls_name}]", code, text))

    pd.DataFrame(rows, columns=["variable", "code", "label"]) \
      .to_csv(os.path.join(CODEBOOK_DIR, f"{prefix}_value_labels.csv"), index=False)

# Export all
export_codebook(ir_meta, "IR")
print("Exported IR codebooks to:", CODEBOOK_DIR)
```

```
Exported IR codebooks to: D:\NCI_COLLEGE PROJECTS\RESEARCH_PROJECT(Aug 2025)\_outputs\codebook
```

Figure 5: Codebook generation from NFHS-5 metadata

6. SAV Import Helper

The helper function load_sav_cols() reads a .sav file and identifies which of the requested columns actually exist in the dataset. It then loads only those valid columns and returns both the filtered DataFrame and its metadata 6.

```
# 0) Helpers to read/select
def load_sav_cols(path, usecols=None):
    # Read a SAV keeping only requested columns that exist; return (df, meta).
    _, meta = pyreadstat.read_sav(path, metadataonly=True)
    wanted = list(usecols) if usecols else list(meta.column_names)
    present = [c for c in wanted if c in meta.column_names]
    if len(present) == 0:
        raise KeyError(f"No requested columns are present in {os.path.basename(path)}")
    df, meta = pyreadstat.read_sav(path, apply_value_formats=False, usecols=present)
    return df, meta
```

Figure 6: SAV import with column filtering

7. Varible Selection & Data Cleaning

A predefined list of NFHS-5 variables was selected for analysis, covering survey design identifiers, fertility preferences, parity, sociodemographic factors, reproductive history, media exposure, and autonomy indicators. The variables were then filtered from the IR file using the helper loader 7.

```
# 1) Load IR columns
IR_VARS = [
    # design/keys
    "V005","V021","V023","V024","V025","V001","V002",
    # denominator + target  / Marital status
    "V501","V626A",
    # Number of living children
    "V218",
    # Desire for more children
    "V605",
    # core sociodemographics
    "V013","V106","V133","V190",
    # reproductive history
    "V201","V511",
    # knowledge & exposure
    "V301",
    "V157","V158","V159","V384A","V384B","V384C","V364",
    # autonomy & barriers (if present)
    "V467B","V467C","V467D", "V467E", "V467F", "V467G", "V467H", "V467I"
    # social (Caste and religion)
    "V130","V131",
    # convenience
    "V012",
]

# deduplicate while preserving order
IR_VARS = list(dict.fromkeys(IR_VARS))

IR, ir_meta = load_sav_cols(IR_SAV, IR_VARS)
print("IR shape:", IR.shape)
```

Figure 7: Selected Variable

Helper code defines 8 a set of DHS special missing codes (8, 9, 96–99) and provides two functions for cleaning numeric variables. clean_int() converts values to integers and masks these missing codes as NaN, while clean_num() safely converts general numeric fields.

```python
# Data cleaning & handling missing values
MISS = {8, 9, 96, 97, 98, 99}

def clean_int(series):
    x = pd.to_numeric(series, errors="coerce").astype("Int64")
    return x.mask(x.isin(MISS))

def clean_num(series):
    return pd.to_numeric(series, errors="coerce")
```

Figure 8: Data Cleaning Helper Functions

8. Variable Label Mapping

Manual recoding was important because NFHS-5 variables rely on DHS-specific numeric codes that are not analytically interpretable in their raw form.

```python
# LABEL MAPS: verified for NFHS-5 India
V025_MAP  = {1:"Urban", 2:"Rural"}
V013_MAP  = {1:"15-19",2:"20-24",3:"25-29",4:"30-34",5:"35-39",6:"40-44",7:"45-49"}
V106_MAP  = {0:"No education",1:"Primary",2:"Secondary",3:"Higher"}
V190_MAP  = {1:"Poorest",2:"Poorer",3:"Middle",4:"Richer",5:"Richest"}
V301_MAP  = {0:"Knows no method",1:"Knows only folkloric",2:"Knows only traditional",3:"Knows modern"}
```

Figure 9: Category mapping for selected NFHS-5 variables

9. Logistic Regression Output Saving

The logistic regression model, fitted using the statsmodels library, produces both Odds Ratios (OR) and Average Marginal Effects (AME) 10. These outputs are stored as pandas DataFrames and automatically exported to the project's tables directory using pandas and os, ensuring the results are saved in a structured and reproducible format.

```python
# Show non-intercept first
or_tab = pd.concat([or_tab[or_tab["term"]!="Intercept"], or_tab[or_tab["term"]=="Intercept"]], ignore_index=True)
os.makedirs(TABS_DIR, exist_ok=True)
or_path = os.path.join(TABS_DIR, "logit_unmet_determinants_final_OR.csv")
or_tab.to_csv(or_path, index=False)
print(f"\nSaved OR table -> {or_path}")

# 5) Average Marginal Effects (AME)
mfx = res.get_margeff(at="overall", method="dydx")
mfx_df = mfx.summary_frame()
ame_path = os.path.join(TABS_DIR, "logit_unmet_determinants_final_AME.csv")
mfx_df.to_csv(ame_path, index=False)
print(f"Saved AME table -> {ame_path}")
```

Figure 10: Saving OR and AME outputs

10. Elastic Net Logistic Regression

A scikit-learn pipeline was created to preprocess variables with one-hot encoding and fit an Elastic Net logistic regression model. A predefined hyperparameter grid was used to configure the strength and mixture of regularisation.

```python
# Preprocessing: OHE for cats
preprocess = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(drop="first", handle_unknown="ignore"), cat_vars),
        ("num_bin", "passthrough", num_vars + bin_vars),
    ]
)

# Base classifier: Elastic Net logistic regression
logit_en = LogisticRegression(
    penalty="elasticnet",
    solver="saga",
    max_iter=200,
    n_jobs=-1
)

pipe = Pipeline(steps=[
    ("preprocess", preprocess),
    ("clf", logit_en),
])
```

Figure 11: Preprocessing + Pipeline Setup

```python
# Hyperparameter grid + CV
param_grid = {
    "clf__C": [0.01, 0.1, 1.0, 10.0],            # regularisation strength
    "clf__l1_ratio": [0.0, 0.25, 0.5, 0.75, 1.0] # 0=L2, 1=L1, between=Elastic Net
}
```

Figure 12: Hyperparameter Grid

11. Random Forest Model

The Random Forest classifier was configured using a predefined hyperparameter grid and tuned through stratified 5-fold cross-validation using GridSearchCV. This setup ensures that the model selects the optimal parameter combination based on weighted ROC–AUC performance. The configuration block defines the model, search space, and cross-validation strategy.

```python
# 3) Base RF + hyperparameter grid

rf_base = RandomForestClassifier(
    random_state=42,
    n_jobs=-1
)

param_grid = {
    "n_estimators":      [200, 300],
    "max_depth":         [None, 15],
    "min_samples_split": [20, 50],
    "min_samples_leaf":  [5, 10],
    "max_features":      ["sqrt"]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid_rf = GridSearchCV(
    estimator=rf_base,
    param_grid=param_grid,
    scoring="roc_auc",
    cv=cv,
    n_jobs=-1,
    verbose=2
)
```

Figure 13: Random Forest configuration and hyperparameter grid

12. XGBoost Model

A gradient-boosted tree model (XGBoost) 14 was configured for binary classification using a predefined hyperparameter grid 15 and optimised with 5-fold Stratified GridSearchCV 17. Only the model setup and tuning configuration were included to define how the system is prepared for training.

```python
# 3) Base XGBoost model
xgb_base = XGBClassifier(
    objective="binary:logistic",
    eval_metric="logloss",
    n_jobs=-1,
    random_state=42,
    tree_method="hist"
)
```

Figure 14: Base XGBoost Model

```
# Hyperparameter grid
param_grid = {
    "n_estimators":      [200, 400],
    "learning_rate":     [0.05, 0.1],
    "max_depth":         [3, 4, 5],
    "subsample":         [0.8, 0.9],
    "colsample_bytree":  [0.7, 0.9],
    "min_child_weight":  [1, 5]
}
```

Figure 15: Hyperparameter Grid

```
# Cross-Validation Setup
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

Figure 16: Cross-Validation Setup

```
# Hyperparameter grid
param_grid = {
    "n_estimators":      [200, 400],
    "learning_rate":     [0.05, 0.1],
    "max_depth":         [3, 4, 5],
    "subsample":         [0.8, 0.9],
    "colsample_bytree":  [0.7, 0.9],
    "min_child_weight":  [1, 5]
}
```

Figure 17: GridSearchCV Configuration

5 Replication Steps

1. Download NFHS-5 datasets and place in project directory.
2. Open Anaconda → Launch Jupyter Notebook.
3. Activate project environment:

```
conda activate unmet_env
```

4. Run the notebook `x23379031_RP_PART2.ipynb` sequentially.
5. All plots, tables, and models will auto-generate.

6 Troubleshooting

- **Memory errors:** Restart the kernel and clear variables.
- **Plot rendering issues:** Install "matplotlib-inline".

References

K. S. James et al. (2021). *National Family Health Survey (NFHS-5), 2019–21: India*. Accessed: 10 December 2025. Mumbai. URL: https://dhsprogram.com/pubs/pdf/FR375/FR375.pdf.

Rahaman, M., R. Singh and P. Chouhan (2022). "Levels, patterns and determinants of using reversible contraceptives for limiting family planning in India". In: *BMC Women's Health* 22.124. DOI: 10.1186/s12905-022-01706-0. URL: https://doi.org/10.1186/s12905-022-01706-0.

Sharma, H., S.K. Singh and S. Srivastava (2021). "Spatial heterogeneity and major correlates of unmet need among young married women in India". In: *SAGE Open* 11.2. DOI: 10.1177/21582440211024615. URL: https://doi.org/10.1177/21582440211024615.