National College of Ireland

Configuration Manual

MSc Research Practicum- Part II
MSc in Data Analytics

Raghul Sureshbabu
Student ID: x23377755

School of Computing
National College of Ireland

Supervisor: Prof.Vikas Sahani

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Raghul Sureshbabu		
Student ID:	x23377755		
Programme:	MSc in Data Analytics	**Year:**	2026
Module:	Research Practicum-Part II		
Supervisor:	Prof. Vikas Sahani		
Submission Due Date:	28/01/2026		
Project Title:	Critical Analysis of Machine Learning and Deep Learning Models for Predicting Medical Equipment Demand in Hospitals		
Word Count:	1150	**Page Count:**	8

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: Raghul Sureshbabu

Date: 26/01/26

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):	

Critical Analysis of Machine Learning and Deep Learning Models for Predicting Medical Equipment Demand in Hospitals

Raghul Sureshbabu
x23377755

1 Introduction

Accurate demand forecasting is a major challenge in hospital management. This manual document the process of configuration and implementation of the thesis "Critical Analysis of Machine Learning and Deep Learning Models to Predict Medical Equipment Demand in Hospitals". The system predicts utilisation of Medicare Durable Medical Equipment (DME) using historical CMS data and uses machine learning and deep learning algorithms to assist in data-driven planning. SARIMAX, XGBoost, LSTM, GRU, and N-BEATS models are used to learn patterns in equipment usage, and SHAP is used to interpret the key drivers of the predictions.

This manual is presented in such a format that it will enable the readers to get a complete understanding on how to replicate the system. The Experimental Set up outlines the hardware and software as well as data source requirements. The Implementation section gives stepwise instructions on how to prepare the data, train and test the model.

2 Experimental Setup

2.1 Hardware and operating system

The experiments were conducted on a standard personal laptop to make sure that the setup is reproducible without any special equipment. The development and testing machine is powered by a 12th Gen Intel Core i5-1235U multi-core processor with 16 GB of RAM, which was sufficient to handle the multiyear CMS DME datasets, preprocess the data and train all forecasting models. The project was implemented on Windows 11, 64-bit version. The experiments did not require a dedicated GPU, although it can be used to shorten the training time of the deep-learning models LSTM, GRU and N-BEATS.

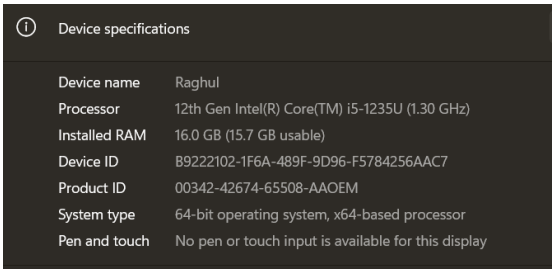

Figure 1: Device specifications

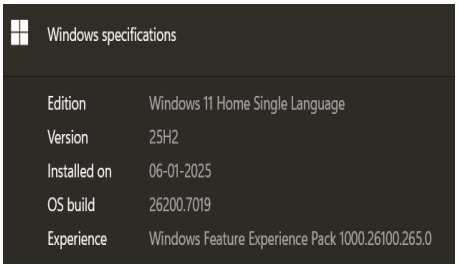

Figure 2: Windows specifications

2.2 Software Environment

- **Python (3.11.0):** This was the programming language in which the entire project was implemented and it was installed from the official Python distribution.
- **Jupyter Notebook:** This environment was used to execute the code for data pre-processing, model training, evaluation and explainability.
- **Dataset source:** The data used in this project was obtained from the Medicare Durable Medical Equipment, Devices and Supplies by Geography and Service dataset, which was offered publicly by Centres for Medicare and Medicaid Services (CMS). The CMS data portal provides the data at the following location:

```
C:\Users\ACER>python --version
Python 3.11.0
```

Figure 3: Environment Version

3 Implementation

3.1 Libraries

All python libraries required for processing, modelling and visualisation of data are installed using pip in the command line before running the experiments.

```
#libraries
import os
import re
import glob
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
import joblib
from statsmodels.tsa.statespace.sarimax import SARIMAX
import shap
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
```

Figure 4: Libraries imports

3.2 Loading the Dataset

- The CMS DME annual flat files (2014–2023) are stored locally in a designated data folder.
- Using the glob module and pandas. read_csv(), the notebook automatically locates all yearly CSV files, loads them into DataFrames and concatenates them into a single combined dataset

https://data.cms.gov/provider-summary-by-type-of-service/medicare-durable-medical-equipment-devices-supplies/medicare-durable-medical-equipment-devices-supplies-by-geography-and-service

```python
#Data directories
DATA_DIR = "."
OUTPUT_DIR = "./processed"

if not os.path.exists(OUTPUT_DIR):
    os.makedirs(OUTPUT_DIR)
    print(f"Created directory: {OUTPUT_DIR}")
else:
    print(f"Output directory already exists: {OUTPUT_DIR}")
def year_from_fname(path):
    """
    Extracts a year (20YY) from filenames like 'dy14_geor.csv'- 2014
    """
    filename = os.path.basename(path)
    match = re.search(r"dy(\d{2})", filename)
    if not match:
        raise ValueError(f"Could not find year in filename: {path}")
    yy = int(match.group(1))
    year = 2000 + yy
    return year

#CSV files with matching name pattern
pattern = os.path.join(DATA_DIR, "mup_dme_ry24_p05_v10_dy*_geor.csv")
files = sorted(glob.glob(pattern))
if not files:
    raise FileNotFoundError("No matching CSV files found. Please check your folder path or filenames.")
print(f"Found {len(files)} files.")
print("First few files:")
for f in files[:3]:
    print("  -", f)
```

Figure 5: Data loading

```python
frames = []

#Reading all the files and adding a year column
for f in files:
    year = year_from_fname(f)
    print(f"Reading file: {f} (Year: {year})")

    df = pd.read_csv(f)
    df["Year"] = year
    frames.append(df)

#Merging
combined = pd.concat(frames, ignore_index=True)
print("Merged DataFrame shape:", combined.shape)
```

Figure 6: Merging CSV files

3.3 Data Preprocessing

The merged data is then filtered to remove national-level records in order to make sure only state-level observations will remain. The geographic columns are then renamed into shorter, more descriptive names (e.g. GeoLevel, StateCode, State) to make it easier for processing.

```python
# Drop 'National' rows to keep data in the state-level
if "Rfrg_Prvdr_Geo_Lvl" in combined.columns:
    before = combined.shape[0]
    combined = combined[combined["Rfrg_Prvdr_Geo_Lvl"].str.lower() != "national"].copy()
    after = combined.shape[0]
    print(f"Removed {before - after} 'National' rows.")

# Rename columns
combined = combined.rename(columns={
    "Rfrg_Prvdr_Geo_Lvl": "Geo_Level",
    "Rfrg_Prvdr_Geo_Cd": "State_Code",
    "Rfrg_Prvdr_Geo_Desc": "State"
})
print("Columns renamed successfully.")
```

Figure 7: Data Preprocessing

3.4 Feature Engineering and Train/Test Split

The target variable (TotSuplrClms) and a series of structural characteristics (provider, supplier, beneficiary counts and average charge/payment amounts) are selected and combined together with year, state and HCPCS code to create the final training dataset (train_ready) which is used for all models.

```python
# Defining target and feature columns
TARGET = "Tot_Suplr_Clms"
FEATURES = [
    "Tot_Rfrg_Prvdrs",
    "Tot_Suplrs",
    "Tot_Suplr_Benes",
    "Tot_Suplr_Srvcs",
    "Avg_Suplr_Sbmtd_Chrg",
    "Avg_Suplr_Mdcr_Alowd_Amt",
    "Avg_Suplr_Mdcr_Pymt_Amt",
    "Avg_Suplr_Mdcr_Stdzd_Amt"
]
keep_cols = ["Year", "State", "HCPCS_Cd"] + FEATURES + [TARGET]
train_ready = agg[keep_cols].copy()
print(f"Selected {len(keep_cols)} columns for training data.")
print("Shape of training dataset:", train_ready.shape)
```

Figure 8: Target and Feature selection

3.5 Model Training

3.5.1 SARIMAX

In the SARIMAX baseline, the data is initially aggregated by year to calculate yearly totals (in the forms of claims, providers, suppliers, beneficiaries and services) and yearly averages (in the form of charge and payment amounts) and then stored in yearly_data before the model fitting.

```
# Aggregating data by Year for the SARIMAX model
yearly_data = df.groupby("Year").agg({
    "Tot_Suplr_Clms": "sum",
    "Tot_Rfrg_Prvdrs": "sum",
    "Tot_Suplrs": "sum",
    "Tot_Suplr_Benes": "sum",
    "Tot_Suplr_Srvcs": "sum",
    "Avg_Suplr_Sbmtd_Chrg": "mean",
    "Avg_Suplr_Mdcr_Alowd_Amt": "mean",
    "Avg_Suplr_Mdcr_Pymt_Amt": "mean",
    "Avg_Suplr_Mdcr_Stdzd_Amt": "mean"
}).reset_index()
print("Yearly aggregation complete.")
print(yearly_data.head())
```

Figure 9: SARIMAX model training

3.5.2 XGBoost

The XGBoost regression model is configured, with optimised hyperparameters (number of trees, learning rate, depth and subsampling settings) and trained on the ready feature set with the help of xgbmodel.fit(Xtrain, y_train).

```
# Creating and training the XGBoost model
xgb_model = XGBRegressor(
    n_estimators=400,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    objective="reg:squarederror"
)
print("Training XGBoost Model")
xgb_model.fit(X_train, y_train)
print("Model training completed!")
```

Figure 10: XGBoost model training

3.5.3 LSTM

The scaled sequence data is then provided to a custom PyTorch LSTM model (DME_LSTM) with two LSTM layers and a final linear layer and trained using the Adam optimiser and mean squared error loss to predict DME utilisation.

```python
# Step 5.6 – Define LSTM Model
class DME_LSTM(nn.Module):
    def __init__(self, n_features, hidden_size=64, num_layers=2):
        super(DME_LSTM, self).__init__()
        self.lstm = nn.LSTM(n_features, hidden_size, num_layers, batch_first=True, dropout=0.2)
        self.fc = nn.Linear(hidden_size, 1)
    def forward(self, x):
        out, _ = self.lstm(x)
        out = self.fc(out[:, -1, :])  # Last time step
        return out

model = DME_LSTM(n_features=X_train.shape[2]).to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0005)
epochs = 30
print("\n Starting LSTM training on scaled data\n")

# Step 5.7 – Training
for epoch in range(epochs):
    model.train()
    total_loss = 0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        preds = model(X_batch)
        loss = criterion(preds, y_batch)
        loss.backward()
        optimizer.step()
        total_loss += loss.item() * X_batch.size(0)
    avg_loss = total_loss / len(train_loader.dataset)
    print(f"Epoch [{epoch+1}/{epochs}] -> Loss: {avg_loss:.6f}")
print("\n Training finished successfully!")
```

Figure 11: LSTM model training

3.5.4 GRU

GRU model (DME_GRU) is initialised and trained over 30 epochs with the Adam optimiser and the mean squared error loss on the data in the size of mini-batches with data loader, learning to predict DME utilisation.

```python
#Model
model_gru = DME_GRU(n_features=X_train.shape[2]).to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model_gru.parameters(), lr=0.0005)
epochs = 30
print("\nTraining GRU\n")

# Training Loop
for epoch in range(epochs):
    model_gru.train()
    total_loss = 0.0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        preds = model_gru(X_batch)
        loss = criterion(preds, y_batch)
        loss.backward()
        optimizer.step()
        total_loss += loss.item() * X_batch.size(0)
    avg_loss = total_loss / len(train_loader.dataset)
    print(f"Epoch [{epoch+1}/{epochs}] - Loss: {avg_loss:.6f}")
print("\nGRU training complete!")
```

Figure 12: GRU model training

3.5.5 N-BEATS

A custom N-BEATS architecture was created using PyTorch by specifying N-BEATS block and classes that utilise fully connected layers and backcast/forecast output to predict the time series and train it on the sequence data that was prepared.

```python
# Step 5 - Define N-BEATS Model
class NBeatsBlock(nn.Module):
    def __init__(self, input_size, hidden_size, theta_size):
        super().__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, hidden_size)
        self.fc4 = nn.Linear(hidden_size, hidden_size)
        self.theta_layer = nn.Linear(hidden_size, theta_size)
        self.backcast_size = input_size
        self.forecast_size = 1

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        theta = self.theta_layer(x)
        backcast = theta[:, :self.backcast_size]
        forecast = theta[:, -self.forecast_size:]
        return backcast, forecast


class NBeats(nn.Module):
    def __init__(self, input_size, hidden_size=128, n_blocks=3):
        super().__init__()
        self.blocks = nn.ModuleList([
            NBeatsBlock(input_size, hidden_size, input_size + 1)
            for _ in range(n_blocks)
        ])

    def forward(self, x):
        residual = x.view(x.size(0), -1)
        forecast = torch.zeros(residual.size(0), 1, device=x.device)
        for block in self.blocks:
            backcast, block_forecast = block(residual)
            residual = residual - backcast
            forecast = forecast + block_forecast
        return forecast
print("GRU model defined")
```

Figure 13: N-Beats model training

3.6 Explainability (SHAP)

XGBoost

SHAP explainability of the XGBoost model is calculated with shap.TreeExplainer which calculates SHAP values on the test set in order to measure how each feature contributes to the model predictions.

N-BEATS

In the N-BEATS model, a prediction wrapper is defined and passed to shap.KernelExplainer, which applies sampled prediction background set to approximate SHAP values on smaller values of inputs and give feature-level attributions over time

SHAP Explainable AI (XAI) for XGBoost Model

```python
print("Starting SHAP Explainability for XGBoost")
explainer = shap.TreeExplainer(xgb_model)
shap_values = explainer.shap_values(X_test, check_additivity=False)
print("SHAP values calculated successfully!")
```

```
Starting SHAP Explainability for XGBoost
SHAP values calculated successfully!
```

Figure 14 : SHAP (XGBoost)

```python
#Explainable AI (XAI) for N-BEATS Model
print("Starting Explainable AI (SHAP) for N-BEATS")
model_nbeats.eval()
sample_X = torch.tensor(X_test_scaled[:100], dtype=torch.float32).to(device)
# Defining prediction function
def nbeats_predict(X):
    X_torch = torch.tensor(X.reshape(-1, 3, len(FEATURES)), dtype=torch.float32).to(device)
    with torch.no_grad():
        preds = model_nbeats(X_torch).cpu().numpy()
    return preds

# Flatten for SHAP input
sample_X_flat = sample_X.cpu().numpy().reshape(sample_X.shape[0], -1)
background = sample_X_flat[np.random.choice(sample_X_flat.shape[0], 20, replace=False)]

# Define KernelExplainer
explainer = shap.KernelExplainer(nbeats_predict, background)
shap_values = explainer.shap_values(sample_X_flat[:30])
expanded_features = []
for t in range(3):
    expanded_features.extend([f"{f}_t{t+1}" for f in FEATURES])
```

Figure 15: SHAP (N-Beats)

3.7 Extended Experiment

In the extended experiment, a new output folder is generated and all the raw files of 2014-2022 as well as the individual 2023 file are combined into a single list of inputs, preparing the data into a state to be processed and retrained on the entire 2014-2023 period

ANALYSIS 2 (LATEST YEAR DATASET) (2014-2023):

```
#Data directory
EXT_OUTPUT_DIR = "./processed_2014_2023"
if not os.path.exists(EXT_OUTPUT_DIR):
    os.makedirs(EXT_OUTPUT_DIR)
    print("Created new folder for extended experiment:", EXT_OUTPUT_DIR)
else:
    print("Extended output folder already exists:", EXT_OUTPUT_DIR)
DATA_DIR = "."
```

```
Extended output folder already exists: ./processed_2014_2023
```

```
#filenames
def year_from_filename(fname):
    base = os.path.basename(fname)
    try:
        dy_pos = base.index("dy")
        yr_str = base[dy_pos+2:dy_pos+4]
        yr = int(yr_str)
        return 2000 + yr
    except Exception as e:
        print("Warning: could not parse year from", base, ":", e)
        return None
```

Figure 16: Dataset loading (Analysis 2)

DATA PROCESSING

```
print("Load raw files for 2014-2023")
pattern_2014_22 = os.path.join(DATA_DIR, "mup_dme_ry24_p05_v10_dy*_geor.csv")
files_2014_22 = sorted(glob.glob(pattern_2014_22))

print("Found 2014-2022 files:")
for f in files_2014_22:
    print(" ", os.path.basename(f))

#2023 file
file_2023 = os.path.join(DATA_DIR, "mup_dme_ry25_p05_v10_dy23_geor.csv")
if not os.path.exists(file_2023):
    raise FileNotFoundError(f"2023 file not found: {file_2023}")
print("\nFound 2023 file:")
print(" ", os.path.basename(file_2023))

# Combining all file paths into single list
all_files_ext = files_2014_22 + [file_2023]
print("\nTotal files for extended experiment:", len(all_files_ext))
```

Figure 17: Data Processing (Analysis 2)

3.8 Extended Experiment Results

The results of the extended experiment are summed up in a small results table, indicating that both XGBoost and N-BEATS retrained on the 2014-2023 dataset have high accuracy, with XGBoost slightly better in RMSE, MAE and R2.

```
#Model comparison
results_ext = [
    {
        "Model": "XGBoost (2014-2023)",
        "RMSE": xgb_rmse_ext,
        "MAE": xgb_mae_ext,
        "R2": xgb_r2_ext
    }
]
results_ext.append({
    "Model": "N-BEATS (2014-2023)",
    "RMSE": round(rmse_ext, 2),
    "MAE": round(mae_ext, 2),
    "R2": round(r2_ext, 4)
})
df_results_ext = pd.DataFrame(results_ext)
print("\nExtended model comparison:\n")
print(df_results_ext)
```

```
Extended model comparison:

                Model         RMSE         MAE      R2
0  XGBoost (2014-2023)  1461.680000  256.330000  0.9883
1  N-BEATS (2014-2023)  2040.030029  496.480011  0.9777
```

Figure 18: Analysis 2 Results

3.9 Analysis 1 and Analysis 2 Comparison Results:

The comparison table is created comparing the original (2014-2022) and extended (2014-2023) results of XGBoost and N-BEATS, in which both models are highly accurate when the most recent year (2023) is added, with XGBoost having a slight advantage in RMSE, MAE and R2.

```
# Previous results comparison
xgb_r2_old = 0.9812
xgb_mae_old = 211.84
xgb_rmse_old = 1755.28
nbeats_r2_old = 0.9801
nbeats_mae_old = 517.92
nbeats_rmse_old = 1953.51
comparison = {
    "Model": ["XGBoost", "N-BEATS"],
    "R2 (2014-2022)": [xgb_r2_old, nbeats_r2_old],
    "R2 (2014-2023)": [round(xgb_r2_ext, 4), round(nbeats_r2_ext, 4)],
    "MAE (2014-2022)": [xgb_mae_old, nbeats_mae_old],
    "MAE (2014-2023)": [round(xgb_mae_ext, 2), round(nbeats_mae_ext, 2)],
    "RMSE (2014-2022)": [xgb_rmse_old, nbeats_rmse_old],
    "RMSE (2014-2023)": [round(xgb_rmse_ext, 2), round(nbeats_rmse_ext, 2)]
}
df_compare = pd.DataFrame(comparison)
print("\nXGBoost vs N-BEATS - OLD vs EXTENDED RESULTS:")
print(df_compare)
```

```
XGBoost vs N-BEATS - OLD vs EXTENDED RESULTS:
     Model  R2 (2014-2022)  R2 (2014-2023)  MAE (2014-2022)  MAE (2014-2023)  \
0  XGBoost          0.9812          0.9883           211.84           256.330000
1  N-BEATS          0.9801          0.9777           517.92           496.480011

   RMSE (2014-2022)  RMSE (2014-2023)
0           1755.28       1461.680000
1           1953.51       2040.030029
```

Figure 19: Comparison Results

References

1. Centers for Medicare & Medicaid Services (CMS) (n.d.) Medicare Durable Medical Equipment, Devices and Supplies by Geography and Service. Available at: https://data.cms.gov/provider-summary-by-type-of-service/medicare-durable-medical-equipment-devices-supplies/medicare-durable-medical-equipment-devices-supplies-by-geography-and-service.
2. Python Software Foundation (2019) Welcome to Python.org. [online] Available at: https://www.python.org.
3. The pandas development team (n.d.) pandas documentation. [online]Available at: https://pandas.pydata.org.
4. Pedregosa, F. et al. (n.d.) scikit-learn: machine learning in Python — scikit-learn documentation. [online]Available at: https://scikit-learn.org.
5. XGBoost Developers (n.d.) XGBoost documentation. [online]Available at: https://xgboost.ai.
6. Paszke, A. et al. (n.d.) PyTorch documentation. [online] Available at: https://pytorch.org.
7. TensorFlow Authors (n.d.) TensorFlow documentation. [online] Available at: https://www.tensorflow.org.
8. Keras Team (n.d.) Keras: The Python deep learning API. [online] Available at: https://keras.io.
9. Lundberg, S.M. (n.d.) SHAP documentation. [online] Available at: https://shap.readthedocs.io.