Configuration Manual

MSc Research Project
Data Analytics

Shakti Siva Raman Annamalai
Student ID: X23383569

School of Computing
National College of Ireland

Supervisor: Athanasios Staikopoulos

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shakti Siva Raman Annamalai
Student ID:	X23383569
Programme:	MSc in Data Analytics **Year:** 2025
Module:	Research Practicum
Lecturer:	Athanasios Staikopoulos
Submission Due Date:	11th December 2025
Project Title:	Early Student Dropout Prediction in Online Learning Using Explainable Machine Learning and Deep Learning Models
Word Count:1614	**Page Count:** 16

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: Shakti Siva Raman Annamalai

Date: 11/12/2025

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

Shakti Siva Raman Annamalai
Student ID: X23383569

1 Introduction

This configuration guide describes the process of configuring and operating the Early Dropout Prediction System that was created during the MSc research project. It describes the process of installing the necessary software, loading the OULAD dataset, performing the preprocessing processes, training the ML/DL models, and producing evaluation and explainability results.

Using this manual, users are able to replicate the model results and can also process the system workflow without much difficulty.

2 System Requirements

2.1 Hardware Requirements

Components	Minimum Requirement
Processor	Intel i5/ AMD equivalent
RAM	8 GB (Recommended: 16GB)
Storage	5-10 GB free space
System Type	64-bit Operating System

2.2 Software Requirements

Software	Version
Python	3.10.0 or higher
Anaconda/Jupiter Notebook	Latest Version
Operating System	Windows 10/11, macOS or Linux
IDE	VS code / PyCharm

2.3 Requited Python Libraries

- NumPy
- Pandas
- Scikit – learn
- Xgboost
- Shap
- TensorFlow
- Matplotlib
- Seaborn

3 Software Installation & Environmental Setup

3.1 Install Python /Anaconda

Install either:

- Anaconda Distribution or Python 3.10.0

3.2 Create a Virtual Environment

Open Anaconda Prompt on your terminal and run:

conda create -n dropout_env python=3.10.0
conda activate dropout_env

3.3 Install Required Python Libraries

pip install numpy pandas scikit-learn xgboost shap tensorflow matplotlib seaborn

These libraries assist in preprocessing of data, training the model, evaluation and explainability.

3.4 Launch Jupyter Notebook

Start the notebook interface by running the below in cmd

Jupiter notebook

4 Dataset Configuration

The project is based on the Open University Learning Analytics Dataset (OULAD), which comprises of student demographic information, virtual learning environment (VLE) clicks data and course registration data.

4.1 Dataset Source

Link:https://www.kaggle.com/datasets/anlgrbz/student-demographics-online-education-dataoulad?resource=download

4.2 Required Dataset Files

Download the following csv files from the OULAD:

- studentInfo.csv
- studentVle.csv
- studentRegistration.csv

4.3 Choosing Dataset to add to Project Folder

After downloading place all the three csv files in the same folder of the Jupiter Notebook.

4.4 Description of Dataset Files

- **studentInfo.csv:** Demographic information of students including gender, age band, highest education, disability and final result.
- **studentVle.csv:** This file has daily click records that are used to calculate early engagement (total_click_early).
- **studentRegistration.csv:** Includes data on registration and date_unregistration that are used to tag dropout.

5 Implementation Overview

In the following section, a situational overview of the steps needed to execute the early-dropout prediction system will be given. The execution commences with the importation of required Python libraries as well as loading of the OULAD data into the notebook. This data is then pre-processed and

processed with feature engineering, encoding and train-test splitting to generate a model-ready dataset that is clean. This data is used to train four predictive models, namely Logistic Regression, Decision Tree, XGBoost and ANN, and their performance is assessed by measuring metrics, ROC curves, and confusion matrices. SHAP explanations are created to understand model predictions and conclude risk scores are calculated in order to name high-risk students. The execution of each code block in the subsequent sections will replicate the entire system and produce all the needed outputs.

6 Preprocessing

6.1 Import Libraries

```
# Import Necessary Libaries
from IPython.display import display, HTML
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold
from scipy.stats import randint, uniform
import warnings, xgboost as xgb
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
    confusion_matrix, RocCurveDisplay
```

Fig – 1 Importing necessary python libraries

This cell downloads all the necessary Python libraries necessary to preprocess, train the model, evaluate the model, and explain the model.

6.2 Load Datasets and output

```
#1.2 Loading the OULAD Tables : Demographics,Click logs,Registration

H3("1.2 Loading OULAD tables: demographics, VLE clicks, registration")

student_info = pd.read_csv("studentInfo.csv")            # The table consist of demographics + final_result
student_vle  = pd.read_csv("studentVle.csv")             # The table consist of click logs (sum_click, date)
student_reg  = pd.read_csv("studentRegistration.csv")    # The table consist of date_registration, date_unregistration

print("Loaded tables:",
      len(student_info), "studentInfo rows |",
      len(student_vle),  "studentVle rows |",
      len(student_reg),  "studentRegistration rows")

HR()
```

1.2 Loading OULAD tables: demographics, VLE clicks, registration

```
Loaded tables: 32593 studentInfo rows | 10655280 studentVle rows | 32593 studentRegistration rows
```

Fig – 2 Loading OULAD Dataset files into the dataset and Output

The three csv files from OULAD are loaded into the pandas Data frames for further processing and viewing the output

6.3 Features Engineering

```
#1.3 Filtering Virtual Learning Environment(VLE) to the early window ( first 28 days)

H3("1.3 Filter VLE to first 28 days and aggregate clicks per student-module-presentation")

student_vle_early = student_vle[student_vle["date"] <= EARLY_DAYS].copy()

vle_agg = (
    student_vle_early
    .groupby(["id_student", "code_module", "code_presentation"])["sum_click"]
    .sum().reset_index()
    .rename(columns={"sum_click": "total_clicks_early"})
)

display(vle_agg.head())
print("Rows after early filter/aggregation:", len(vle_agg))
HR()
```

Fig – 3 Filtering first 28 days and aggregating early clicks per students

This action narrows down VLE activity to the 28 days and sums up all clicks per student to form the feature total_clicks_early, which is an indicator of early student activity.

```
#1.4 Merging the demographic and early clicks

H3("1.4 Merge demographics with early clicks")

merged = pd.merge(
    student_info, vle_agg,
    on=["id_student", "code_module", "code_presentation"],
    how="left"                          # keep zero-click students
)
merged["total_clicks_early"] = merged["total_clicks_early"].fillna(0).astype(int)

display(merged.head())
HR()
```

Fig – 4 Merging demographic with early clicks

The data on early engagement is combined with the demographic one based on the student ID, module, and presentation and zero-click students are retained.

```
#1.5 Adding Registration info

H3("1.5 Add registration/unregistration dates (for labelling & audit)")

merged = pd.merge(
    merged,
    student_reg[["id_student","code_module","code_presentation",
                 "date_registration","date_unregistration"]],
    on=["id_student","code_module","code_presentation"],
    how="left"
)
merged["date_registration"]   = merged["date_registration"].fillna(0).astype(int)
merged["date_unregistration"] = merged["date_unregistration"].fillna(-1).astype(int)

display(merged[["id_student","code_module","code_presentation",
                "date_registration","date_unregistration","total_clicks_early"]].head())
HR()
```

1.5 Add registration/unregistration dates (for labelling & audit)

	id_student	code_module	code_presentation	date_registration	date_unregistration	total_clicks_early
0	11391	AAA	2013J	-159	-1	401
1	28400	AAA	2013J	-53	-1	618
2	30268	AAA	2013J	-92	12	281
3	31604	AAA	2013J	-52	-1	494
4	32885	AAA	2013J	-176	-1	567

Fig – 5 Adding registration and unregistration dates

Registration information is incorporated into the data and any non-response values substituted in the data to verify the dropout labels.

```
#1.6 Creating target label (dropout) and Selecting the modeling view

H3("1.6 Create target label (dropped_out) and select modeling columns")

# Label: 1 if unregistered at any time (we predict early with early signals)
merged["dropped_out"] = (merged["date_unregistration"] != -1).astype(int)

model_data = merged[[
    # identifiers
    "id_student", "code_module", "code_presentation",
    # features
    "gender", "age_band", "highest_education", "disability",
    "total_clicks_early",
    # label
    "dropped_out",
    # audit
    "date_registration", "date_unregistration"
]].copy()

# defensive fill for categoricals

model_data[["gender","age_band","highest_education","disability"]] = \
    model_data[["gender","age_band","highest_education","disability"]].fillna("Unknown")

H3("Checks")
print("\nMissing values per column:\n", model_data.isna().sum())
print("\nClass balance (value counts):\n", model_data["dropped_out"].value_counts(dropna=False))
print("\nBasic stats (clicks):\n", model_data[["total_clicks_early"]].describe())
HR()
display(model_data.head(10))
```

	id_student	code_module	code_presentation	gender	age_band	highest_education	disability	total_clicks_early	dropped_out	date_registration	date_unregistration
0	11391	AAA	2013J	M	55<=	HE Qualification	N	401	0	-159	-1
1	28400	AAA	2013J	F	35-55	HE Qualification	N	618	0	-53	-1
2	30268	AAA	2013J	F	35-55	A Level or Equivalent	Y	281	1	-92	12
3	31604	AAA	2013J	F	35-55	A Level or Equivalent	N	494	0	-52	-1
4	32885	AAA	2013J	F	0-35	Lower Than A Level	N	567	0	-176	-1
5	38053	AAA	2013J	M	35-55	A Level or Equivalent	N	727	0	-110	-1
6	45462	AAA	2013J	M	0-35	HE Qualification	N	499	0	-67	-1
7	45642	AAA	2013J	F	0-35	A Level or Equivalent	N	414	0	-29	-1
8	52130	AAA	2013J	F	0-35	A Level or Equivalent	N	361	0	-33	-1

Fig – 6 creating dropout label, preparing final modelling dataset and performing data quality check

It is done by this step to form the dropped-out label, final features to model, missing categories filled, dataset quality checked and final clean dataset displayed.

```python
#1.7 Saving the cleaned dataset
H3("1.7 Save cleaned dataset")
os.makedirs(OUT_DIR, exist_ok=True)
model_data.to_csv(OUT_CSV, index=False)
print("succefully saved the dataset →", OUT_CSV)
HR()
```

1.7 Save cleaned dataset

```
succefully saved the dataset → C:\Users\A. shakti sivaraman\OneDrive\Desktop\RESEARCH\prepared_dropout_dataset_early.csv
```

Fig – 7 Code for Saving the final cleaned dataset to a csv file

The next step saves a pre-processed dataset (model_data) as a CSV file so that one can reuse it when training and evaluating the model.

6.4 Feature Encoding & Scaling

```python
#Importing Libaries
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer

df = pd.read_csv(OUT_CSV)

FEATURES_CAT = ["gender", "age_band", "highest_education", "disability"]
FEATURES_NUM = ["total_clicks_early"]
TARGET       = "dropped_out"

X = df[FEATURES_CAT + FEATURES_NUM].copy()
y = df[TARGET].astype(int)

preprocessor = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=True), FEATURES_CAT),
        ("num", StandardScaler(), FEATURES_NUM),
    ],
    remainder="drop",
    verbose_feature_names_out=False
)

display(X.head(3)); HR()
```

STEP-2 MODELING & EVALUATION

2.1 Feature encoding (one-hot) + numeric scaling (StandardScaler)

	gender	age_band	highest_education	disability	total_clicks_early
0	M	55<=	HE Qualification	N	401
1	F	35-55	HE Qualification	N	618
2	F	35-55	A Level or Equivalent	Y	281

Fig – 8 Encoding categorical features and scaling numerical features

In this step, One-Hot Encoding is used on categorical data and Standard Scaling is used on numerical data through a Column Transformer

6.5 Train–Test Split (80/20)

```python
#2.2 Training and Testing  split into (80/20)

H3("2.2 Training and Testing  split into (80/20)")

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y
)
print("Train:", X_train.shape, "| Test:", X_test.shape)
print("Dropout rate — Train:", y_train.mean().round(3), "| Test:", y_test.mean().round(3))
HR()
```

2.2 Training and Testing split into (80/20)

```
Train: (26074, 5) | Test: (6519, 5)
Dropout rate — Train: 0.305 | Test: 0.304
```

Fig – 9 Splitting the dataset into training 80% and Testing 20% sets

Stratified sampling is used to divide the dataset into the training and testing sets to ensure that the distribution of dropouts was similar across the splits.

7 Model Training

7.1 Logistic Regression

```
# 1) Logistic Regression Model

pipe_lr = Pipeline([
    ("prep", preprocessor),
    ("clf", LogisticRegression(max_iter=500, class_weight="balanced"))
])
pipe_lr.fit(X_train, y_train)
print("Logistic Regression is  trained.")
```

Fig – 10 Training the Logistic Regression model

To deal with the dropout imbalance, Logistic Regression is trained as a baseline interpretable model with class weights.

7.2 Decision Tree

```
# 2) Decision Tree Model

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

pipe_dt = Pipeline([
    ("prep", preprocessor),
    ("clf", DecisionTreeClassifier(random_state=42, class_weight="balanced"))
])

dt_params = {
    "clf__max_depth": randint(3, 20),
    "clf__min_samples_leaf": randint(1, 20),
    "clf__min_samples_split": randint(2, 20),
}

rs_dt = RandomizedSearchCV(
    pipe_dt,
    dt_params,
    n_iter=20,
    scoring="f1",
    n_jobs=-1,
    cv=cv,
    random_state=42,
)
rs_dt.fit(X_train, y_train)
pipe_dt_best = rs_dt.best_estimator_
print("DT best params:", rs_dt.best_params_)
```

Fig – 11 Training the Decision Tree model

The best depth and splitting parameters are identified with the help of a Decision Tree classifier, the cross-validation of 5-folding and the F1 score.

7.3 XGBoost

```
# 3) XGBoost Model

pipe_xgb = Pipeline([
    ("prep", preprocessor),
    ("clf", xgb.XGBClassifier(
        objective="binary:logistic",
        eval_metric="logloss",
        random_state=42,
        n_jobs=-1,
        scale_pos_weight=scale_pos_weight,   # handling imbalance
    ))
])

xgb_params = {
    "clf__n_estimators":     randint(200, 500),
    "clf__max_depth":        randint(3, 7),
    "clf__learning_rate":    uniform(0.03, 0.15),
    "clf__subsample":        uniform(0.7, 0.3),
    "clf__colsample_bytree": uniform(0.7, 0.3),
    "clf__reg_lambda":       uniform(0.0, 2.0),
}

rs_xgb = RandomizedSearchCV(
    pipe_xgb,
    xgb_params,
    n_iter=25,
    scoring="f1",
    n_jobs=-1,
    cv=cv,
    random_state=42,
)
rs_xgb.fit(X_train, y_train)
pipe_xgb_best = rs_xgb.best_estimator_
print("XGB best params:", rs_xgb.best_params_)
HR()
```

Fig – 12 Training the XGBoost model

To deal with the imbalance between the dropout and non-dropout students, XGBoost is trained with hyperparameters that are fine-tuned and a scale_pos_weight value.

7.4 ANN (MLP)

```
TF_OK = False
ann, Xt_train, Xt_test = None, None, None

try:
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    TF_OK = True
    print("TensorFlow detected:", tf.__version__)
except Exception as e:
    print("TensorFlow import failed → ANN skipped.\nReason:", e)

if TF_OK:
    # Using the same preprocessor to transform features
    Xt_train = preprocessor.fit_transform(X_train)
    Xt_test  = preprocessor.transform(X_test)

    # Converting sparse matrix to dense if needed (for Keras)
    if hasattr(Xt_train, "toarray"):
        Xt_train = Xt_train.toarray()
        Xt_test  = Xt_test.toarray()

    input_dim = Xt_train.shape[1]

    ann = keras.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(32, activation="relu"),
        layers.Dropout(0.25),
        layers.Dense(16, activation="relu"),
        layers.Dense(1, activation="sigmoid"),
    ])

    ann.compile(
        optimizer=keras.optimizers.Adam(1e-3),
        loss="binary_crossentropy",
        metrics=["accuracy"],
    )
```

```
    print("ANN is trained Successfully. (Validation accuracy during training:",
      np.max(hist.history["val_accuracy"]).round(3), ")")

HR()
```

Fig – 13 Training ANN (MLP) Model

The ANN is trained with two hidden layers using the same pre-processed features. Imbalance and overfitting are solved by use of class weights and early stopping.

8. Model Evaluation

The test data are tested against the trained models based on accuracy, precision, recall, F1 score and AUC. The screenshots below present the results of the evaluation and the ROC curves.

8.1 Metrics Comparison Table

```
#2.4 Evaluation of all models

H3("2.4 Evaluation — all models in one table (threshold = 0.50)")

from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
    confusion_matrix, RocCurveDisplay
)

# collecting probabilities
probas = [
    ("Logistic Regression", pipe_lr.predict_proba(X_test)[:, 1]),
    ("Decision Tree",       pipe_dt_best.predict_proba(X_test)[:, 1]),
    ("XGBoost",             pipe_xgb_best.predict_proba(X_test)[:, 1]),
]

if TF_OK and ann is not None and Xt_test is not None:
    probas.append(("ANN (MLP)", ann.predict(Xt_test, verbose=0).ravel()))

def eval_fixed(name, y_true, proba, thr=0.5):
    y_pred = (proba >= thr).astype(int)
    return {
        "Model": name, "Threshold": thr,
        "Accuracy":  accuracy_score(y_true, y_pred),
        "Precision": precision_score(y_true, y_pred, zero_division=0),
        "Recall":    recall_score(y_true, y_pred, zero_division=0),
        "F1":        f1_score(y_true, y_pred, zero_division=0),
        "AUC":       roc_auc_score(y_true, proba),
        "_y_pred":   y_pred
    }
```

```
rows = [eval_fixed(n, y_test, p, 0.50) for n,p in probas]
metrics_df = pd.DataFrame(rows)

display(
    metrics_df
    .style.format({"Threshold":"{:.2f}","Accuracy":"{:.4f}","Precision":"{:.4f}",
                   "Recall":"{:.4f}","F1":"{:.4f}","AUC":"{:.4f}"})
    .highlight_max(axis=0, subset=["Accuracy","Precision","Recall","F1","AUC"], color="#c6efce")
)

HR()
```

2.4 Evaluation — all models in one table (threshold = 0.50)

	Model	Threshold	Accuracy	Precision	Recall	F1	AUC	_y_pred
0	Logistic Regression	0.50	0.6013	0.4161	0.7673	0.5396	0.6993	[1 0 0 ... 0 1 0]
1	Decision Tree	0.50	0.7082	0.5195	0.5567	0.5375	0.7213	[1 0 0 ... 0 0 0]
2	XGBoost	0.50	0.7191	0.5381	0.5481	0.5430	0.7303	[1 0 0 ... 0 0 0]
3	ANN (MLP)	0.50	0.7052	0.5142	0.5743	0.5426	0.7265	[1 0 0 ... 0 0 0]

Fig – 14 Evaluation metrics table for all trained models

All models are assessed based on Accuracy, Precision, Recall, F1 score, and AUC in this code and presented in one comparison table.

8.2 ROC Curves Output

Logistic Regression

```
#Logistic Regression - ROC Curve

H3("ROC Curve – Logistic Regression")

plt.figure(figsize=(6,5))
RocCurveDisplay.from_predictions(
    y_test,
    probas[0][1],
    name="Logistic Regression"
)
plt.title("ROC Curve – Logistic Regression")
plt.tight_layout()
plt.show()

HR()
```

Fig – 15 LR ROC curve

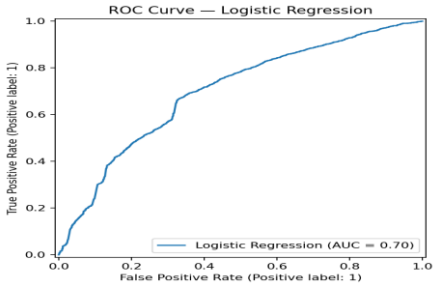

Fig – 16 Actual ROC curve image

Decision Tree

```
# Decision Tree - ROC Curve

H3("ROC Curve – Decision Tree")

plt.figure(figsize=(6,5))
RocCurveDisplay.from_predictions(
    y_test,
    probas[1][1],
    name="Decision Tree"
)
plt.title("ROC Curve – Decision Tree")
plt.tight_layout()
plt.show()

HR()
```

Fig – 17 DT ROC curve

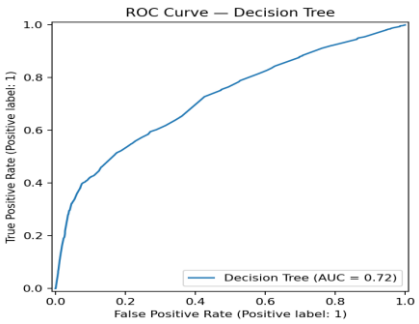

Fig – 18 Actual ROC curve image

ANN (MLP)

```
#Artifical Neural Network - ROC Curve

if len(probas) > 3:
    H3("ROC Curve – ANN (MLP)")

    plt.figure(figsize=(6,5))
    RocCurveDisplay.from_predictions(
        y_test,
        probas[3][1],
        name="ANN (MLP)"
    )
    plt.title("ROC Curve – ANN (MLP)")
    plt.tight_layout()
    plt.show()

    HR()
```

Fig – 19 ANN ROC curve

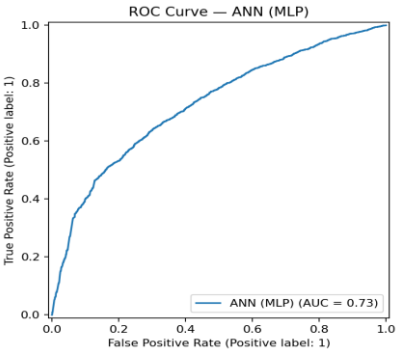

Fig – 20 Actual ROC curve image

XGBoost

```
#XGBoost - ROC Curve

H3("ROC Curve – XGBoost")

plt.figure(figsize=(6,5))
RocCurveDisplay.from_predictions(
    y_test,
    probas[2][1],
    name="XGBoost"
)
plt.title("ROC Curve – XGBoost")
plt.tight_layout()
plt.show()

HR()
```

Fig – 21 XGBoost ROC curve

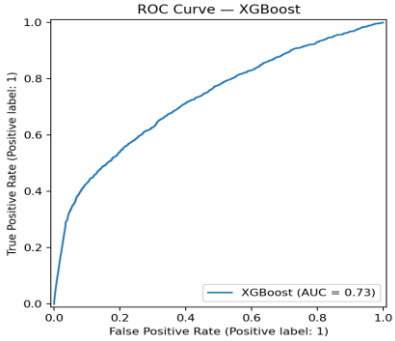

Fig – 22 Actual ROC curve image

9. Confusion Matrices

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def plot_cm(y_true, y_pred, title="Confusion Matrix"):
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)

    fig, ax = plt.subplots(figsize=(4, 4))
    disp.plot(ax=ax, cmap="Blues", colorbar=False)
    ax.set_title(title)
    ax.set_xlabel("Predicted label")
    ax.set_ylabel("True label")
    plt.tight_layout()
    plt.show()
```

```
# Logistic Regression - Confusion Matrix

plot_cm(y_test, rows[0]["_y_pred"], "Logistic Regression (thr=0.50)")
HR()

#Decision Tree - Confusion Matrix

plot_cm(y_test, rows[1]["_y_pred"], "Decision Tree (thr=0.50)")
HR()
# XGBoost - Confusion Matrix

plot_cm(y_test, rows[2]["_y_pred"], "XGBoost (thr=0.50)")
HR()
```

```
#Artifical Neural Network

if len(rows) > 3:
    plot_cm(y_test, rows[3]["_y_pred"], "ANN (MLP) (thr=0.50)")
    HR()
```

Fig -23 Code for generating confusion matrices for all models

9.1 Confusion Matrix Output

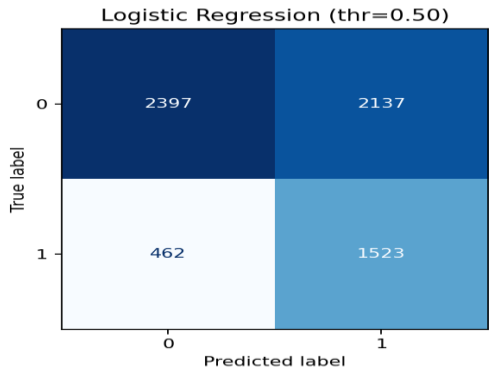

Fig – 24 Confusion Matrix for LR

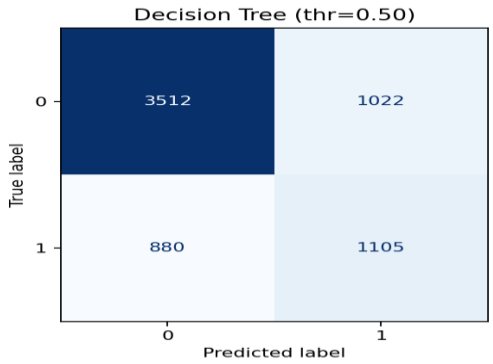

Fig – 25 Confusion Matrix for DT

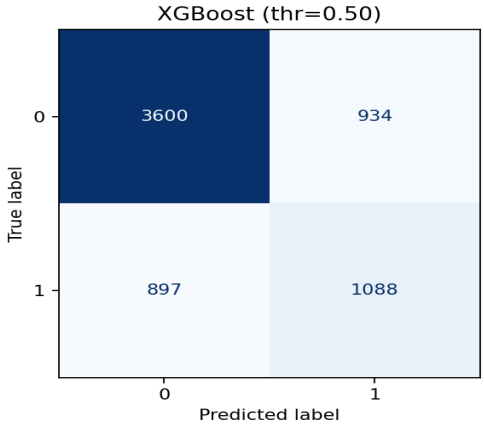

Fig – 26 Confusion Matrix for XGBoost

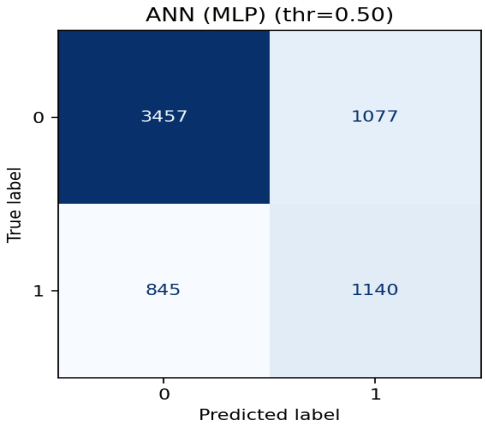

Fig – 27 Confusion Matrix for ANN

10. Explainability (SHAP)

10.1 SHAP Setup (XGBoost)

```
#Explainability with SHAP - XGBOOST

H3("2.5 Explainability with SHAP (XGBoost)")

import shap

# Get feature names
ohe = pipe_xgb_best.named_steps["prep"].named_transformers_["cat"]
feat_cat = ohe.get_feature_names_out(FEATURES_CAT)
feat_num = np.array(FEATURES_NUM)
feat_all = np.concatenate([feat_cat, feat_num])

# Transform training data with same preprocessing as the pipeline
Xt_train_xgb = pipe_xgb_best.named_steps["prep"].transform(X_train)
if hasattr(Xt_train_xgb, "toarray"):
    Xt_train_xgb = Xt_train_xgb.toarray()

# Sample up to 2000 rows for SHAP
idx = np.random.RandomState(42).choice(
    Xt_train_xgb.shape[0],
    size=min(2000, Xt_train_xgb.shape[0]),
    replace=False
)

# Create SHAP explainer and compute SHAP values
explainer = shap.TreeExplainer(pipe_xgb_best.named_steps["clf"])
shap_vals = explainer.shap_values(Xt_train_xgb[idx])
```

Fig – 28 Code for Preparing SHAP explainer for the tuned XGBoost model

It is a SHAP explainer of the XGBoost model and it calculates SHAP values on a sample of training records.

10.2 SHAP Global Feature Importance (Bar Chart)

```
# Global Feature Importance (Bar)
H3("Global Feature Importance (Bar)")
shap.summary_plot(shap_vals, Xt_train_xgb[idx],
                  feature_names=feat_all,
                  plot_type="bar",
                  show=False)
plt.tight_layout()
plt.show()
HR()
```

Fig – 29 Bar chart

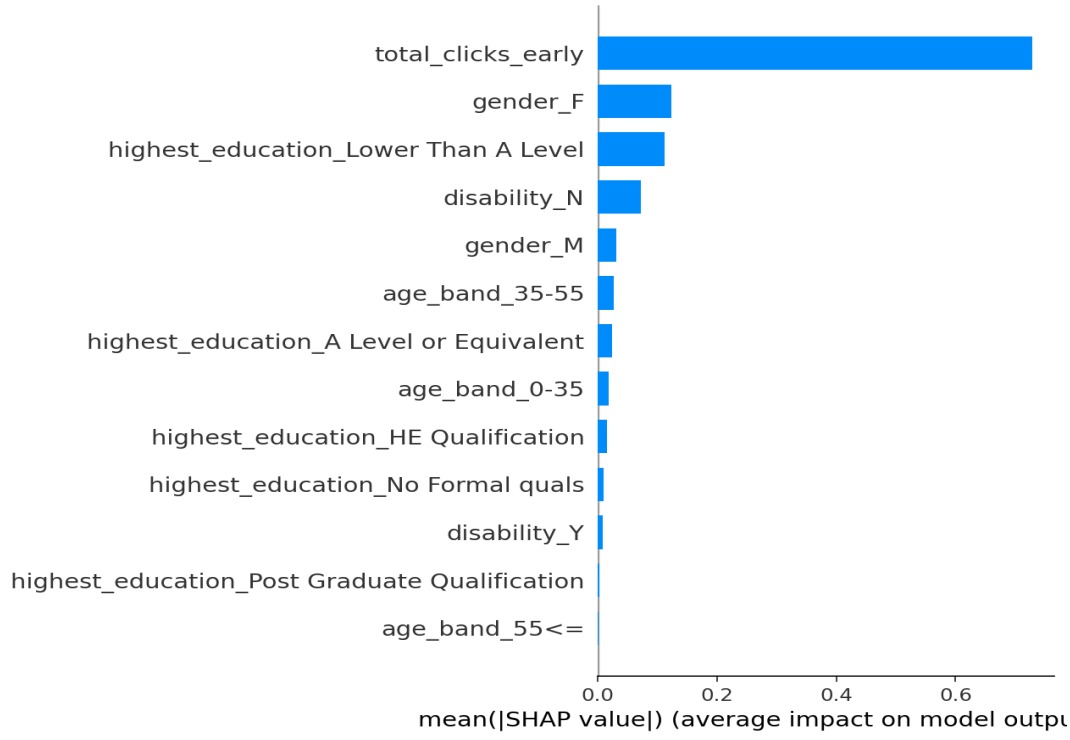

Fig – 30 SHAP Bar Chart

10.3 SHAP Summary (Beeswarm Plot)

```
#Beeswarm

H3("SHAP Summary (Beeswarm)")
shap.summary_plot(shap_vals, Xt_train_xgb[idx],
                  feature_names=feat_all,
                  show=False)
plt.tight_layout()
plt.show()
HR()
```

Fig – 31 Code for Beeswarm

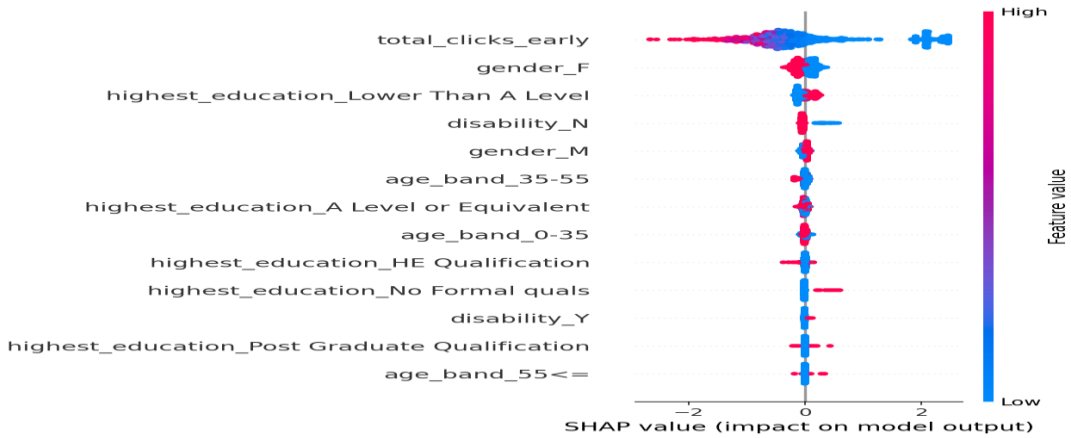

Fig – 32 SHAP Beeswarm pot for XGBoost model

10.4 Model Comparison Summary

```
#Comparison Summary and model compariosn

H3("2.6 Comparison summary")

# Creating ranked table (sorted by Recall → F1 → AUC → Accuracy)
ranked = metrics_df.sort_values(
    by=["Recall", "F1", "AUC", "Accuracy"],
    ascending=False
).reset_index(drop=True)

numeric_cols = ["Threshold", "Accuracy", "Precision", "Recall", "F1", "AUC"]

styled = (
    ranked
    .style
    .format({col: "{:.4f}" for col in numeric_cols if col in ranked.columns})
    .highlight_max(axis=0,
                   subset=["Accuracy", "Precision", "Recall", "F1", "AUC"],
                   color="#c6efce")
)

display(styled)

# Bar chart for visual and comparison
ax = ranked.set_index("Model")[["Recall", "F1", "AUC", "Accuracy"]].plot(
    kind="bar", figsize=(9, 6)
)
plt.title("Model Comparison — Higher is Better")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()

HR()
```

2.6 Comparison summary

	Model	Threshold	Accuracy	Precision	Recall	F1	AUC	_y_pred
0	Logistic Regression	0.5000	0.6013	0.4161	0.7673	0.5396	0.6993	[1 0 0 ... 0 1 0]
1	ANN (MLP)	0.5000	0.7052	0.5142	0.5743	0.5426	0.7265	[1 0 0 ... 0 0 0]
2	Decision Tree	0.5000	0.7082	0.5195	0.5567	0.5375	0.7213	[1 0 0 ... 0 0 0]
3	XGBoost	0.5000	0.7191	0.5381	0.5481	0.5430	0.7303	[1 0 0 ... 0 0 0]

Fig – 33 Code for comparison summary and model comparison Table

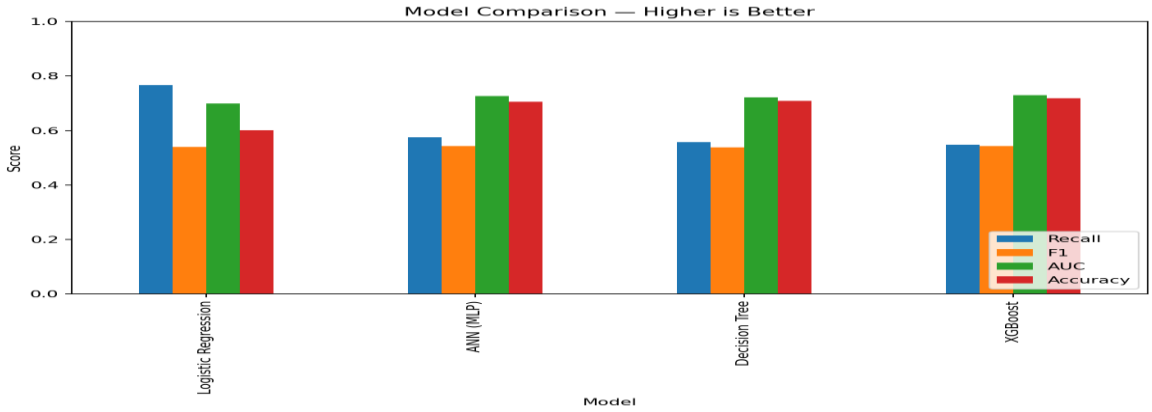

Fig – 34 Model comparison

11. Early Risk Stratification (High/Medium/Low)

11.1 Selecting the best model and getting probabilities

```
#STEP 11 — Early Risk Stratification (HIGH / MEDIUM / LOW)

H2("STEP 11 — Early Risk Stratification (High / Medium / Low)")

# 1) Select best model on (based on F1, then AUC, then Accuracy)

best_row = metrics_df.sort_values(
    by=["F1", "AUC", "Accuracy"],
    ascending=False
).iloc[0]

best_model_name = best_row["Model"]
print("Best model selected based on F1, AUC and Accuracy:", best_model_name)

# 2) Get dropout probabilities from the best model on the TEST set

if best_model_name == "XGBoost":
    best_proba = pipe_xgb_best.predict_proba(X_test)[:, 1]

elif best_model_name == "ANN (MLP)":

    best_proba = ann.predict(Xt_test, verbose=0).ravel()

elif best_model_name == "Decision Tree":
    best_proba = pipe_dt_best.predict_proba(X_test)[:, 1]

else:
    best_proba = pipe_lr.predict_proba(X_test)[:, 1]
```

Fig -35 Code for selecting the best model and computing dropout probabilities

11.2 Creating Risk Data frame & Risk Levels

```
# 3) Build risk_df with probabilities + risk levels

risk_df = X_test.copy().reset_index(drop=True)
risk_df["actual"] = y_test.values
risk_df["dropout_proba"] = best_proba

def risk_category(p):
    if p >= 0.70:
        return "High"
    elif p >= 0.50:
        return "Medium"
    else:
        return "Low"

risk_df["risk_level"] = risk_df["dropout_proba"].apply(risk_category)

# Sort by highest risk first

risk_df = risk_df.sort_values("dropout_proba", ascending=False).reset_index(drop=True)
```

Fig – 36 Code for Building the risk table and risk level

11.3 Top 50 High – Risk Students

```
# keep only High risk
high_risk = risk_df[risk_df["risk_level"] == "High"]

# columns to show - adjust if you renamed anything
cols_to_show = [
    "gender", "age_band", "highest_education", "disability",
    "total_clicks_early", "actual", "dropout_proba", "risk_level"
]

top_50 = high_risk[cols_to_show].head(50)

# styling for risk_level column
def color_risk(val):
    if val == "High":
        return "background-color: #f8d7da; font-weight: bold;"  # light red
    elif val == "Medium":
        return "background-color: #fff3cd;"  # light yellow
    else:
        return ""

styled_top_50 = (
    top_50
    .style
    .format({"dropout_proba": "{:.3f}"})
    .applymap(color_risk, subset=["risk_level"])
    .set_properties(**{"text-align": "center"})
    .set_table_styles([
        {"selector": "th",
         "props": [("text-align", "center"), ("font-weight", "bold")]}
    ])
)

display(styled_top_50)
HR()
```

Fig – 37 Code for generate the top 50 high risk student table

11.4 Overall Risk Distribution in Test Set

```
# 5) Overall Risk Distribution in Test Set (styled table)

H3("Overall Risk Distribution in Test Set")

#Compute counts per risk level
summary = (
    risk_df["risk_level"]
    .value_counts()
    .reindex(["High", "Medium", "Low"])   # enforce ordering
    .fillna(0)
    .astype(int)
)

#Convert to DataFrame with risk_level as normal column
summary_df = summary.reset_index()
summary_df.columns = ["risk_level", "count"]   # rename columns

# Add percentage column
summary_df["percentage"] = (summary_df["count"] / len(risk_df) * 100).round(2)

# Nice numeric formatting (no weird spacing, just plain table)
summary_df["percentage"] = summary_df["percentage"].astype(str) + "%"

# Display as a normal, clean table
display(summary_df)

# Print totals below
print("\nTotal students in test set:", len(risk_df))
print("High-risk students (prob ≥ 0.70):", summary_df.loc[summary_df["risk_level"]=="High", "count"].iloc[0])
print("Medium-risk students (0.50–0.69):", summary_df.loc[summary_df["risk_level"]=="Medium", "count"].iloc[0])
print("Low-risk students (< 0.50):", summary_df.loc[summary_df["risk_level"]=="Low", "count"].iloc[0])

HR()
```

Fig – 38 Code to Calculate the overall risk distribution in the test set

Overall Risk Distribution in Test Set

	risk_level	count	percentage
0	High	950	14.57%
1	Medium	1072	16.44%
2	Low	4497	68.98%

REFERENCE

1.Open University (2014) Open University Learning Analytics Dataset (OULAD). [online] Available at:https://www.kaggle.com/datasets/anlgrbz/student-demographics-online-education-dataoulad?resource=download

2.NumPy Developers (n.d.) NumPy Documentation. [online] Available at:https://numpy.org/

3.Pandas Development Team (n.d.) Pandas Documentation. [online] Available at:https://pandas.pydata.org/

4.Lundberg, S.M. (n.d.) SHAP Documentation. [online] Available at:https://shap.readthedocs.io/en/latest/
0
5.TensorFlow Authors (n.d.) TensorFlow Documentation. [online] Available at:https://www.tensorflow.org/