E-Adapt: Predicting Student Adaptability in Online Classes

By Atharva Mhetre , Ishaan Thapliyal , Snehas Kale , Vishnu Raj

The rapid development of technology has completely changed the way that education is delivered, resulting in the widespread use of online courses and e-learning tools. Online learning has many advantages, such as accessibility and flexibility, but it also has certain difficulties for students. The ability of students to adjust to the online learning environment is a key aspect in determining whether they succeed in those courses. This research intends to construct a prediction model called *e-Adapt* that evaluates and forecasts students' adaptation to online courses in order to address this problem.

Traditional classroom settings and online learning environments are disparate from one another. Without having to engage with peers or professors in person, they demand students to manage their time successfully, navigate virtual platforms, and maintain motivation. As a result, not all pupils have the abilities and traits needed to succeed in this brand-new educational environment. Educators can identify students who may struggle in online courses and offer focused support to improve their learning experiences by understanding and anticipating students' adaptability.

The *e-Adapt* model will combine machine learning algorithms with methods for educational data mining to examine various aspects of student adaptation in online courses. These variables could be the pupils' prior academic success, technological aptitude, abilities to manage their own learning, motivation, and levels of engagement. The *e-Adapt* model will produce insights into unique student adaptability profiles by gathering and analyzing data from numerous sources, including learning management systems, online exams, and student questionnaires.

Aim

The aim of *e-Adapt* is to predict student adaptability in online classrooms using machine learning, giving insights into students' capacities to flourish in digital learning environments and enabling focused interventions to increase student success and online course design.

Purpose of Doing This Project

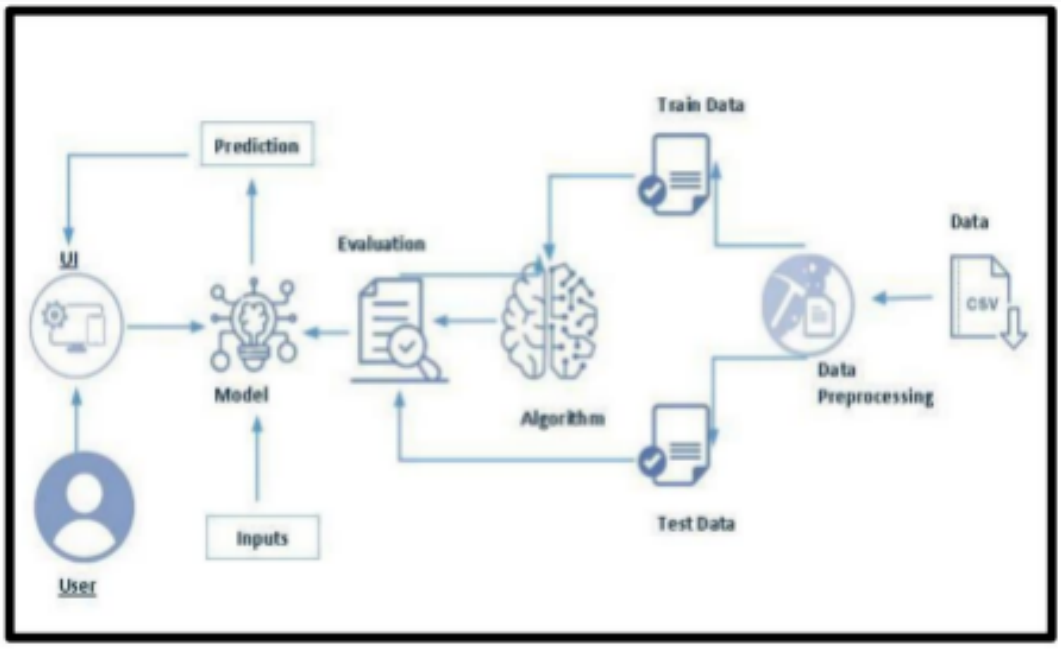
The purpose of undertaking the *e-Adapt* project is to address the increasing importance of student adaptability in online classes. With the rapid growth of online education, it is crucial to

identify students who may struggle in the digital learning environment. By developing a predictive model using machine learning techniques, the project aims to provide educators and institutions with valuable insights into students' adaptability profiles. This knowledge will enable targeted interventions and support strategies to enhance student success, improve course design, and ultimately enhance students' overall online learning experience.

Technical Architecture

The technical architecture of the *e-Adapt* project involves several components to predict student adaptability in online classes effectively. It starts with data collection from various sources such as learning management systems, online assessments, and student surveys. This data is then preprocessed and transformed to ensure its quality and compatibility with machine learning algorithms. Feature extraction techniques are applied to extract relevant information related to academic performance, technological proficiency, self-regulated learning skills, motivation, and engagement.

Machine learning models such as Logistic Regression, K-Nearest Neighbors, Random Forest Classifier, XGBoost Classifier, and CatBoost Classifier are trained on this processed data to predict student adaptability. The trained model is then deployed in a production environment, where it can receive new data inputs and provide real-time adaptability predictions for individual students.



Project Flow

The user is first shown the Home page. From there, the user can browse through the content and navigate to the "Predict My Adaptivity" section, where they will enter specified engagement metrics. Upon clicking the Predict button, the user is directed to the Results page, where the model analyzes the input data and presents the predicted adaptability level.

To accomplish this, the following stages and activities must be completed:

Define Problem / Problem Understanding

- Specify the business problem
- Business requirements
- Literature survey
- Social or business impact

Data Collection and Preparation

- Collect the dataset
- Data preparation

Exploratory Data Analysis

- Descriptive statistics
- Visual analysis

Model Building

- Create a function for model evaluation
- Train and test models using multiple algorithms

Performance Testing & Hyperparameter Tuning

- Evaluate models using multiple performance metrics
- Compare model accuracy before and after hyperparameter tuning
- Compare model accuracy using different feature sets
- Build a final model with the most appropriate features

Model Deployment

- Save the best-performing model
- Integrate it with the web framework for real-time predictions

Prior Knowledge

To successfully complete the *e-Adapt* project, a foundational understanding of several key topics is essential. These include core machine learning concepts and basic web development skills required for model deployment.

Machine Learning Concepts:

- **Supervised Learning:** Understanding how models learn from labeled data to make predictions. More details at: https://www.javatpoint.com/supervised-machine-learning
- **Linear Regression:** A fundamental algorithm for predicting continuous outcomes. Learn more at: https://www.javatpoint.com/linear-regression-in-machine-learning
- **Support Vector Machine (SVM):** A powerful classification technique useful for separating data points. Details here: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
- **Decision Tree:** A tree-based model used for classification and regression tasks. Explanation available at: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
- **Random Forest:** An ensemble method combining multiple decision trees for improved accuracy. See: https://www.javatpoint.com/machine-learning-random-forest-algorithm
- **Evaluation Metrics:** Metrics to assess model performance such as accuracy, precision, recall, and F1-score. A comprehensive guide is at: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- **Regularization:** Techniques to prevent overfitting and improve model generalization. More information at: https://www.javatpoint.com/regularization-in-machine-learning

Flask Basics:
Basic knowledge of Flask, a lightweight Python web framework used for deploying machine learning models as web applications, is necessary. An introductory tutorial can be found at: https://www.youtube.com/watch?v=lj4I_CvBnt0

Milestone 1: Defining Problem/ Problem Understanding

Activity 1: Specifying the Business Problem

Problem Statement:
The transition to online education has introduced new challenges for students, particularly in terms of adaptability. While online learning offers flexibility and accessibility, not all students possess the necessary skills to succeed in this environment. Factors such as low self-motivation, poor time management, limited technical proficiency, and lack of engagement can hinder student performance in virtual classrooms.

Educational institutions lack a data-driven method to identify students who may struggle with online learning. Without early detection, students at risk of falling behind receive support too late—if at all. This contributes to reduced academic performance, increased dropout rates, and lower overall course satisfaction.

Objective of the ML Project:
This project aims to develop a supervised machine learning model—**e-Adapt**—that predicts a student's adaptability to online learning environments. By analyzing various input features such as academic history, engagement metrics, technical skills, and self-regulated learning behaviors, the model will classify students into adaptability categories (e.g., high, medium, low).

Why This Matters:

- **Early Intervention:** Enables institutions to offer timely support to students predicted to struggle.
- **Improved Outcomes:** Helps reduce dropout rates and boost student success in online programs.
- **Scalable and Practical:** Once trained and validated, the model can be integrated into learning management systems to provide real-time predictions and insights.
- **Data-Driven Decision Making:** Supports educators in designing personalized interventions and improving overall course design.

By addressing a real and growing need in education, the *e-Adapt* model positions machine learning as a practical solution for improving student outcomes in digital learning environments.

Activity 2: Business Requirements

To successfully implement the *e-Adapt* model for predicting student adaptability in online learning environments, several core business requirements must be met to ensure technical robustness, usability, and real-world applicability.

First, the system must provide **accurate predictions** of individual students' adaptability levels in online classes using relevant and well-trained machine learning models. To achieve this, it must support **data collection** from diverse sources, including learning management systems (LMS), online assessments, and student surveys. This data must then undergo **data preprocessing** procedures to ensure it is clean, consistent, and compatible with the selected machine learning algorithms.

The system must also support **feature extraction** from the raw data to identify key indicators of student adaptability. These features include academic performance, technological proficiency, self-regulated learning skills, motivation, and levels of engagement. The predictive engine of the system should be powered by appropriate **machine learning models** such as decision trees, neural networks, or ensemble classifiers, which are trained on the preprocessed feature set to classify adaptability levels effectively.

Once the model is trained, the system must facilitate **model deployment** in a production environment, enabling it to receive real-time student data inputs and generate adaptability predictions on demand. A **user interface** should be developed to allow educators, instructors, or administrators to input student data easily, visualize predictions, and access actionable insights based on the model’s output.

Finally, the system should be **scalable and reliable**, capable of handling large datasets and delivering consistent performance across multiple predictions, ensuring that the solution remains effective as the user base or data volume grows.

Activity 3: Literature Survey

The literature survey on predicting student adaptability in online classes underscores the growing importance of adaptability as a critical determinant of student success in digital learning environments. Multiple studies have identified key factors that influence a student's ability to adapt to online education, including prior academic performance, technological proficiency, self-regulated learning skills, motivation, engagement, and personal characteristics such as learning styles and resilience.

The survey explores a range of methodologies used in previous research, from traditional statistical techniques such as linear regression and decision trees to more advanced machine learning algorithms like support vector machines, random forests, and neural networks. These models aim to uncover patterns in student behavior and performance that correlate with adaptability in virtual classrooms.

In addition to model types, the literature highlights various **data sources** used for prediction, including learning management system logs, student feedback surveys, online test results, and demographic information. Researchers emphasize the importance of integrating diverse data types to build robust prediction systems.

Overall, the review reveals a clear consensus: the development of accurate and interpretable prediction models is essential for identifying at-risk students, enabling timely interventions, and improving the overall effectiveness of online education. This body of work provides a strong foundation for the *e-Adapt* project, which aims to build upon these insights using modern machine learning techniques.

Activity 4: Social or Business Impact

Accurately predicting student adaptability in online classes carries substantial social and business implications. From a **social perspective**, such predictive capability enables institutions and educators to identify students who may face challenges in adapting to digital learning environments. Early identification allows for the implementation of targeted support strategies—such as personalized mentoring, skill development workshops, or enhanced learning resources—which can help bridge learning gaps, reduce dropout rates, and promote inclusivity. By ensuring that no student is left behind due to technological or motivational barriers, the system contributes to greater educational equity and empowers learners from diverse backgrounds.

On the **business side**, the impact is equally significant. Educational institutions can leverage adaptability insights to improve the design and delivery of online courses, making them more engaging and effective. Enhanced student satisfaction and improved academic outcomes contribute to stronger retention rates, reduced operational inefficiencies, and higher course completion statistics. These benefits, in turn, boost institutional reputation, attract more student enrollments, and ultimately contribute to increased revenue and long-term sustainability in the competitive education market. Thus, the *e-Adapt* system not only advances educational quality but also strengthens the strategic position of institutions offering online learning programs.

Milestone 2: Data Collection & Preparation

Activity 1: Collecting the Dataset

There are many popular open sources for collecting data, such as kaggle.com, the UCI Machine Learning Repository, and academic data repositories provided by universities and research organizations.

In this project, we have used a `.csv` dataset specifically designed to analyze and predict students' adaptability in online learning environments. This dataset has been downloaded from Kaggle and contains various features related to student engagement, technological proficiency, learning behaviors, and other factors influencing adaptability.

Please refer to the link below to download the dataset:

https://www.kaggle.com/datasets/mdmahmudulhasansuzan/students-adaptability-level-in-online-education

Once the dataset has been downloaded, we will proceed to read and understand the data using visualization and analysis techniques to gain a deeper insight into the data distribution and identify any immediate patterns or issues within the dataset.

Note: There are multiple techniques available for understanding and exploring data, including visual analysis, descriptive statistics, and advanced correlation studies. In this project, we will use a selected set of these techniques for initial analysis, with the flexibility to incorporate additional methods as needed to enhance our understanding of the dataset.

Activity 2: Importing the Libraries

To begin working with the dataset and prepare for data visualization and analysis, it is essential to import the necessary Python libraries commonly used in data science workflows.

```python
# Core Python & Visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Scikit-learn: preprocessing, model selection, evaluation
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score,accuracy_score, classification_report, confusion_matrix

# Scikit-learn classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# Gradient boosting classifiers
from xgboost import XGBClassifier
from catboost import CatBoostClassifier

# For saving the model
import pickle

# Style for plots
sns.set(style="whitegrid")
```

Activity 3: Read the Dataset

Our dataset format might be in `.csv`, Excel files, `.txt`, `.json`, or other formats commonly used for data storage. We can read the dataset efficiently using the `pandas` library, which offers convenient functions for data manipulation and exploration.

In `pandas`, we have a function called `read_csv()` to read `.csv` files. As a parameter, we need to provide the directory or path of the `.csv` file we wish to read.

```
[ ] # Load the dataset
    df = pd.read_csv("students_adaptability_level_online_education.csv")

    # Show first 5 rows
    df.head()
```

	Gender	Age	Education Level	Institution Type	IT Student	Location	Load-shedding	Financial Condition	Internet Type	Network Type	Class Duration	Self Lms	Device	Adaptivity Level
0	Boy	21-25	University	Non Government	No	Yes	Low	Mid	Wifi	4G	3-6	No	Tab	Moderate
1	Girl	21-25	University	Non Government	No	Yes	High	Mid	Mobile Data	4G	1-3	Yes	Mobile	Moderate
2	Girl	16-20	College	Government	No	Yes	Low	Mid	Wifi	4G	1-3	No	Mobile	Moderate
3	Girl	11-15	School	Non Government	No	Yes	Low	Mid	Mobile Data	4G	1-3	No	Mobile	Moderate
4	Girl	16-20	School	Non Government	No	Yes	Low	Poor	Mobile Data	3G	0	No	Mobile	Low

Activity 5: Minor Cleaning

To check for null values, we use `df.isna().any()` and to sum those null values, we use `.sum()`

```
# Check for any missing values in each column
df.isnull().sum()

# Optional: See how many total nulls
print(f"Total missing values: {df.isnull().sum().sum()}")
```

Drop null values and then check for total null values again

```
# Drop rows with any null values (There are no null values present in this case)
df.dropna(inplace=True)

# Confirm removal
df.isnull().sum()
```

Check for ouliers for each variable if possible

```python
# Checking for outliers
for col in df.columns:
    print(f"\nValue counts for {col}:")
    print(df[col].value_counts())
```

```
Value counts for Gender:
Gender
Boy     663
Girl    542
Name: count, dtype: int64

Value counts for Age:
Age
21-25    374
11-15    353
16-20    278
1-5       81
26-30     68
6-10      51
Name: count, dtype: int64

Value counts for Education Level:
Education Level
School        530
University    456
College       219
Name: count, dtype: int64

Value counts for Institution Type:
Institution Type
Non Government    823
Government        382
Name: count, dtype: int64

Value counts for IT Student:
IT Student
No     901
Yes    304
Name: count, dtype: int64

Value counts for Location:
Location
Yes    935
No     270
Name: count, dtype: int64

Value counts for Load-shedding:
Load-shedding
Low     1004
High     201
Name: count, dtype: int64

Value counts for Financial Condition:
Financial Condition
Mid     878
Poor    242
Rich     85
Name: count, dtype: int64

Value counts for Internet Type:
Internet Type
Mobile Data    695
```

Milestone 3: Exploratory Data Analysis (EDA)

Activity 1:Visualisation

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

1) Univariate analysis -

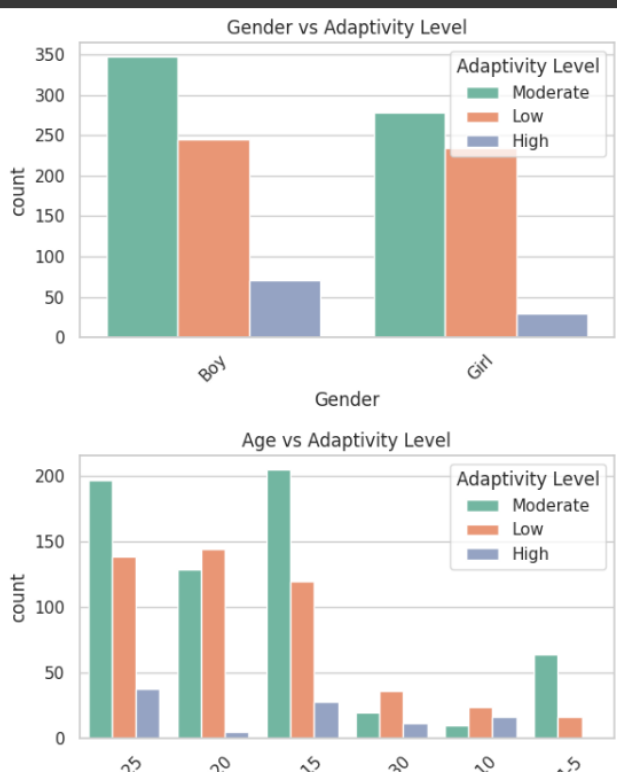
Univariate analysis involves exploring individual features in the dataset to understand their distribution and characteristics. It helps in identifying the frequency of categories, detecting skewness, and understanding the overall structure of each feature before applying machine learning models. Loop through each categorical column in the DataFrame and plot a countplot to visualize the frequency distribution of each category.

2)Multivariate analysis -

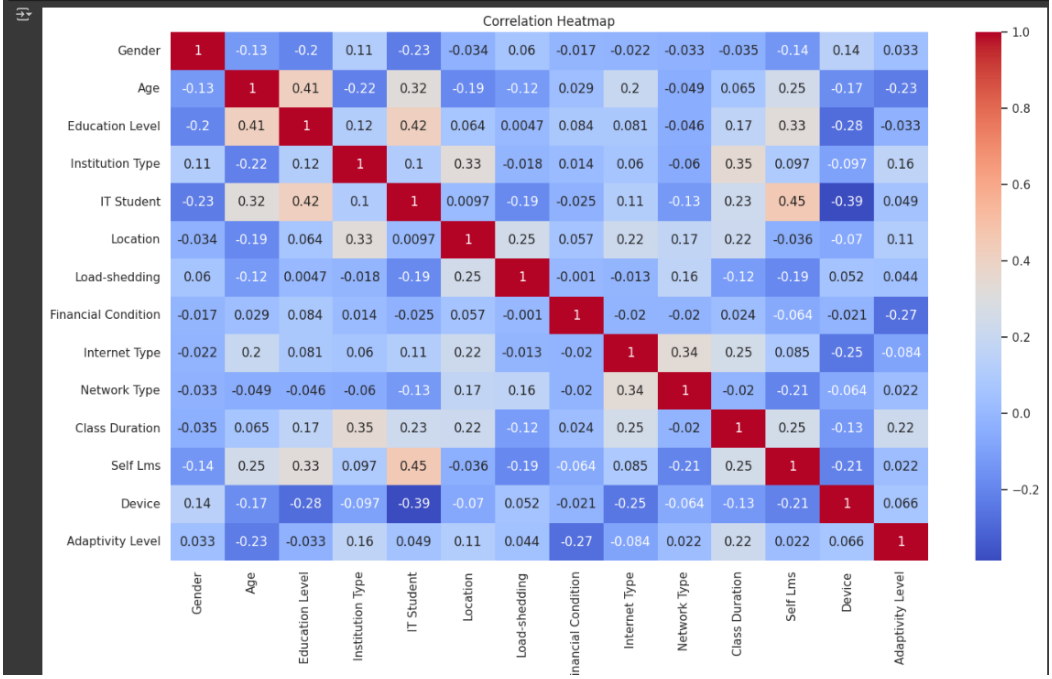
Multivariate analysis is used to find the relationships between multiple features in the dataset. It helps in identifying correlations, dependencies, and patterns that exist among variables, which is important for selecting relevant features and improving model performance.

In this project, we will use heatmaps to visualize correlations between numerical features and detect multicollinearity. Features that are highly correlated may be considered for removal to reduce redundancy in the dataset before training the machine learning model.We will also use bivariate analysis to check the relations between any 2 variables

```
# Identify categorical columns (excluding the target column)
categorical_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()
categorical_cols = [col for col in categorical_cols if col != 'Adaptivity Level']

# Bivariate barplots for each categorical column vs target
for col in categorical_cols:
    plt.figure(figsize=(6,4))
    sns.countplot(x=col, hue='Adaptivity Level', data=df, palette='Set2')
    plt.title(f'{col} vs Adaptivity Level')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```



```python
# Correlation heatmap using encoded dataset
numerical_cols = df_encoded.select_dtypes(include=np.number)

if not numerical_cols.empty:
    plt.figure(figsize=(16,9))
    sns.heatmap(numerical_cols.corr(), annot=True, cmap='coolwarm')
    plt.title('Correlation Heatmap')
    plt.show()
```



Activity 2: Pre-processing

As the data from kaggle was already usable , there no not much pre precessing require other than the minor cleaning in the previous milestone.

Activity 3:Encoding

The code snippet applies label encoding to each column in the copy of DataFrame df. A LabelEncoder object is created, and the fit_transform() method is used to encode the values of each column with unique numerical labels. The encoded values are then used as the respective columns in the original DataFrame.

```python
# Apply label encoding to all columns (categorical)
for col in df_encoded.columns:
    df_encoded[col] = le.fit_transform(df_encoded[col])
```

Activity 4: Train-test split

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state and stratify.

The data is then split into training and testing sets using the train_test_split() function, with 70% of the data allocated for training (X_train, y_train) and 20% for testing (X_test, y_test), ensuring reproducibility with a random state of 42.

```
[ ]  # Features and Target
     X = df_encoded.drop("Adaptivity Level", axis=1)
     y = df_encoded["Adaptivity Level"]

     # Train-Test Split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

     # Standardize
     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)
```

Scaling is a technique used to transform the values of a dataset to a similar scale to improve the performance of machine learning algorithms. Scaling is important because many machine learning algorithms are sensitive to the scale of the input features. Here we are using Standard Scaler. This scales the data to have a mean of 0 and a standard deviation of 1. The formula is given by: X_scaled = (X - X_mean) / X_std

At the core of any supervised machine learning task is the distinction between features (also known as independent variables or predictors) and the target variable (also known as the dependent variable or outcome).

- Features (X): These are the input variables or attributes of your data that you believe influence the target variable. A machine learning model learns patterns and relationships from these features to predict the target. For example, in predicting house prices, features might include the number of bedrooms, square footage, location, and age of the house.
- Target Variable (y): This is the variable you are trying to predict or explain. It's the outcome you're interested in. In the house price example, the target variable would be the house price itself.

Milestone 4: Model Building

Activity 1: Define Preprocessing

For the entirity of the model we set the target feature to adaptivity level and using the others to predict this.

```
# Features and Target
X = df_encoded.drop("Adaptivity Level", axis=1)
y = df_encoded["Adaptivity Level"]
```

```
[ ] # Train-Test Split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

    # Standardize
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

as we did this before test train split and used the X in the splitting, it will be applied to every model.

For each model, we used the already encoded data with the target to show their accuracy , F1 score , confusion matrix and the classification report. For example , here is the result for the 1st model , Random forest classifier

```
[ ] from sklearn.ensemble import RandomForestClassifier

    rfc = RandomForestClassifier(random_state=42)
    rfc.fit(X_train, y_train)
    y_pred_rfc = rfc.predict(X_test)

    print("Random Forest Classifier Results")
    print("Accuracy:", accuracy_score(y_test, y_pred_rfc))
    print("Macro F1 Score:", f1_score(y_test, y_pred_rfc, average='macro'))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rfc))
    print("Classification Report:\n", classification_report(y_test, y_pred_rfc))
    print("-"*50)
```

```
Random Forest Classifier Results
Accuracy: 0.9087136929460581
Macro F1 Score: 0.877725646988427
Confusion Matrix:
 [[ 14   2   4]
 [  0  88   8]
 [  1   7 117]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.70      0.80        20
           1       0.91      0.92      0.91        96
           2       0.91      0.94      0.92       125

    accuracy                           0.91       241
   macro avg       0.92      0.85      0.88       241
weighted avg       0.91      0.91      0.91       241

--------------------------------------------------
```

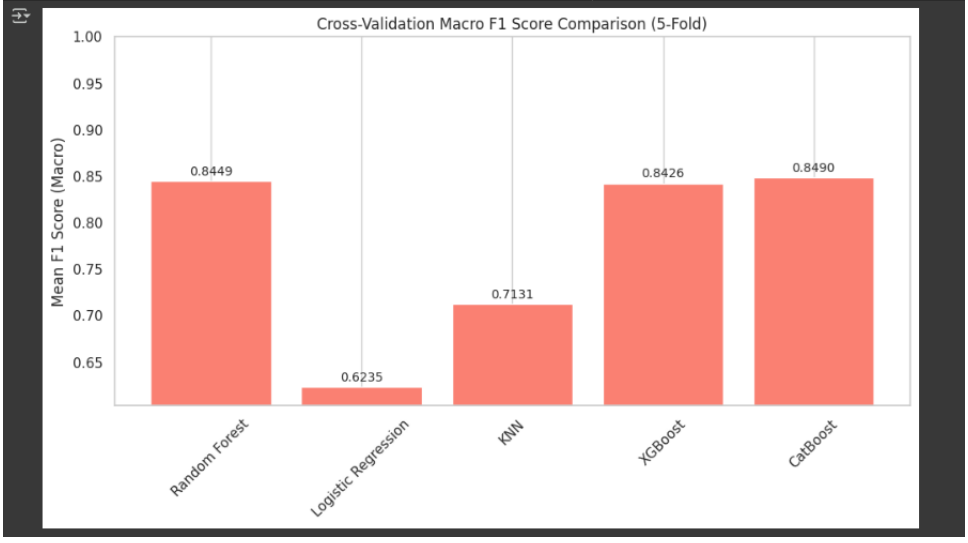
Activity 2: Model comparison

After running the code for our 5 chosen models, Random forest classifier , logistic regression , knn, XGboost and Catboost models. We chose to pick the models best based on their f1 score as f1 score is the metric because it effectively balances precision and recall, providing a robust measure of model performance, especially in the presence of class imbalance where accuracy can be misleading.

We did the initial comparison of our models using a graph as follows

```
[ ]  from sklearn.model_selection import cross_val_score

    # Store mean F1 scores
    cv_f1_scores = {}

    # Random Forest (no scaling needed)
    rfc = RandomForestClassifier(random_state=42)
    cv_f1_scores['Random Forest'] = cross_val_score(rfc, X_train, y_train, cv=5, scoring='f1_macro').mean()

    # Logistic Regression (scaled)
    logreg = LogisticRegression(max_iter=200)
    cv_f1_scores['Logistic Regression'] = cross_val_score(logreg, X_train_scaled, y_train, cv=5, scoring='f1_macro').mean()

    # KNN (scaled)
    knn = KNeighborsClassifier()
    cv_f1_scores['KNN'] = cross_val_score(knn, X_train_scaled, y_train, cv=5, scoring='f1_macro').mean()

    # XGBoost (no scaling needed)
    xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', verbosity=0)
    cv_f1_scores['XGBoost'] = cross_val_score(xgb, X_train, y_train, cv=5, scoring='f1_macro').mean()

    # CatBoost (no scaling needed)
    cat = CatBoostClassifier(verbose=0)
    cv_f1_scores['CatBoost'] = cross_val_score(cat, X_train, y_train, cv=5, scoring='f1_macro').mean()
```

```
bars = plt.bar(cv_f1_scores.keys(), cv_f1_scores.values(), color='salmon')

# Add value labels to each bar
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + 0.003, f"{height:.4f}",
             ha='center', va='bottom', fontsize=10)

plt.title('Cross-Validation Macro F1 Score Comparison (5-Fold)')
plt.ylabel('Mean F1 Score (Macro)')
plt.ylim(min(cv_f1_scores.values()) - 0.02, 1)
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



Activity 3: Hyperparameter tuning

Hyperparameter tuning is the process of selecting the optimal values for the hyperparameters of a machine learning model. It involves systematically searching through a defined space of hyperparameters and evaluating their impact on the model's performance. The goal is to find the hyperparameter configuration that maximizes the model's performance metrics, such as accuracy or F1 score.

As XGBoost and CATBoost models showed the most promise we are going to perform hyperparameter tuning for them.

The process for hyperparameter tuning was quite simple:-

1. **Define the Model Instance:**
    - You start by creating an initial instance of the machine learning model you want to tune (e.g., `XGBClassifier()` or `CatBoostClassifier()`). At this stage, the model uses its default hyperparameters or some initial values.
2. **Define the Hyperparameter Search Space:**
    - Hyperparameters are parameters that are *not* learned from the data during training but are set *before* training. Examples include `n_estimators` (number of trees), `learning_rate` (step size shrinkage), `max_depth` (maximum depth of a tree), etc.
    - You define a dictionary (`xgb_params`, `cat_params`) that specifies the range or list of values you want to explore for each hyperparameter. This search space should be chosen carefully based on prior knowledge, common practices, or initial experimentation.
3. **Choose a Search Strategy:**
    - **Randomized Search (`RandomizedSearchCV`):** (Used for XGBoost in your code)
        - Instead of trying every single combination, Randomized Search samples a fixed number of parameter settings from the defined distributions.
        - **Benefit:** More efficient than Grid Search when the search space is large, as it often finds a good combination in fewer iterations. It's good for quickly exploring a wide range.
        - `n_iter`: Specifies the number of parameter settings that are sampled.
    - **Grid Search (`GridSearchCV`):** (Used for CatBoost in your code)
        - This method systematically tries every single combination of hyperparameter values defined in your search space.
        - **Benefit:** Guarantees finding the absolute best combination within the defined grid.
        - **Drawback:** Can be computationally very expensive and time-consuming, especially with many hyperparameters or large ranges, as the number of

combinations grows exponentially.

4. **Define the Scoring Metric:**
    - You specify the metric that will be used to evaluate the performance of each hyperparameter combination during the search.
    - In your code, `'f1_weighted'` is used. This means the F1 score will be calculated for each class and then averaged, weighted by the number of true instances for each label. This is particularly useful for multi-class classification problems and when dealing with class imbalance.
5. **Set up Cross-Validation (CV):**
    - **`cv=5`**: This indicates that 5-fold cross-validation will be performed. During each fold:
        - The data is split into 5 parts.
        - The model is trained on 4 parts (training set) and evaluated on the remaining 1 part (validation set).
        - This process is repeated 5 times, with each part serving as the validation set once.
    - **Benefit:** Cross-validation provides a more robust estimate of the model's performance by reducing the variance associated with a single train-test split. It helps prevent overfitting to a specific data split.
6. **Execute the Search (`.fit(X_train, y_train)`):**
    - The `fit` method initiates the hyperparameter tuning process. The `RandomizedSearchCV` or `GridSearchCV` object trains multiple models (one for each hyperparameter combination within each CV fold) on the training data (`X_train`, `y_train`), evaluates their performance using the specified scoring metric on the validation sets, and identifies the combination that yields the best average score across all CV folds.
    - `n_jobs=-1`: Utilizes all available CPU cores for parallel processing, significantly speeding up the search.
7. **Retrieve the Best Model and Parameters:**
    - After the search completes, the `best_estimator_` attribute of the search object (e.g., `xgb_search.best_estimator_`) holds the model trained with the optimal hyperparameters found during the search.
    - `best_params_` provides the specific hyperparameter values that resulted in the best performance.
8. **Evaluate the Best Model on Test Data:**
    - The `best_estimator_` (the model with the best hyperparameters) is then used to make predictions on the completely unseen `X_test` data.
    - The F1 score (or your chosen evaluation metric) is calculated on these predictions using the true `y_test` values to get an unbiased estimate of the model's generalization performance.
9. **Compare and Save the Overall Best Model:**

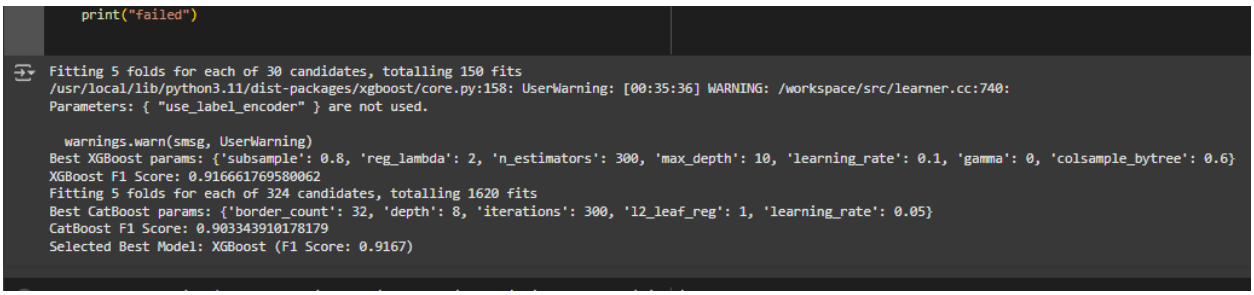
- You compare the F1 scores obtained by the best-tuned XGBoost and CatBoost models.
- The model with the higher F1 score is then selected as the overall best performing model.
- This best model is saved using `pickle` for later use, deployment, or further analysis without needing to retrain it.

Once we obtained the model with the better F1 score we simply saved it as a pkl file. Here is the tuning for the XGboost model

```
# Hyperparameter tuning XGboost

# Define model
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

# Define search space
xgb_params = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7, 10],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.2],
    'reg_lambda': [1, 1.5, 2]
}

# Randomized Search
xgb_search = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=xgb_params,
    n_iter=30,
    scoring='f1_weighted',
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

# Get best model and print f1-score
xgb_search.fit(X_train, y_train)
best_xgb = xgb_search.best_estimator_
xgb_pred = best_xgb.predict(X_test)
xgb_f1 = f1_score(y_test, xgb_pred, average='weighted')
print("Best XGBoost params:", xgb_search.best_params_)
print("XGBoost F1 Score:", xgb_f1)
```

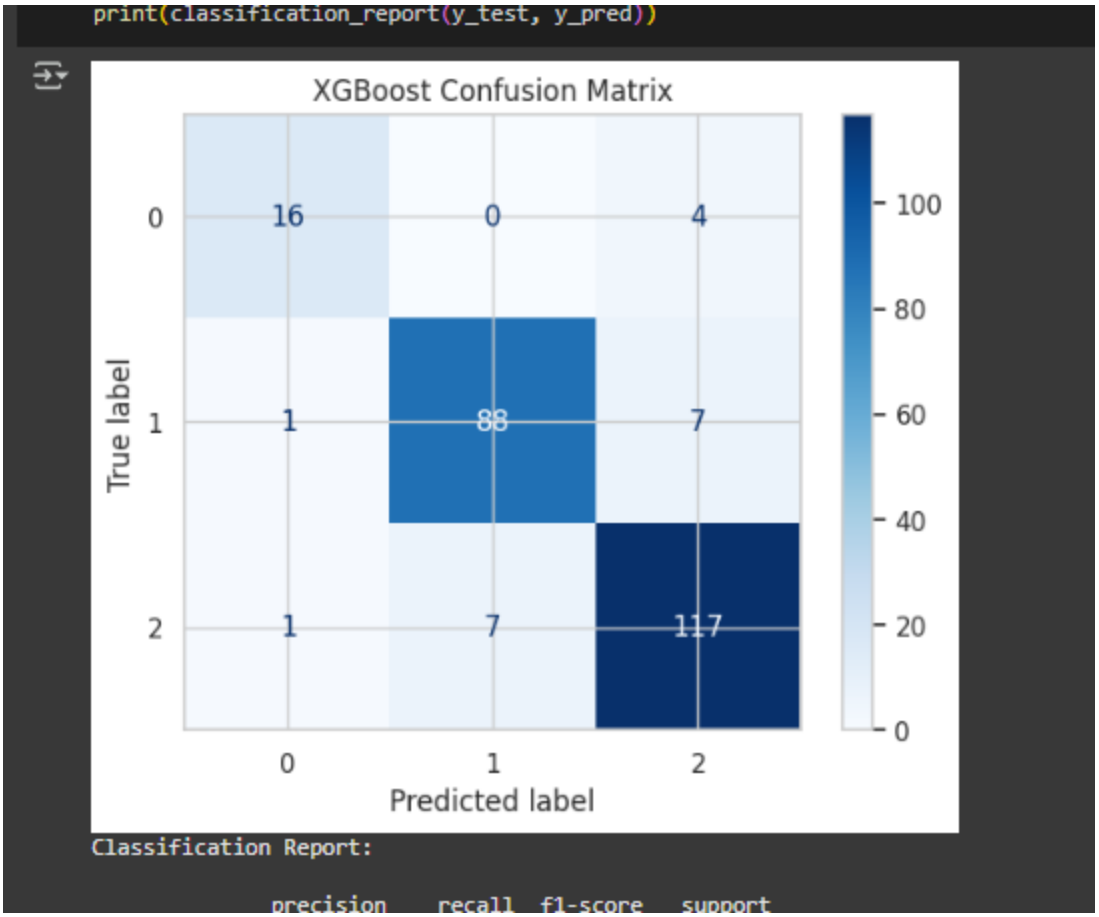
Activity 4: Model comparison 2

Once our tuning was done , we ran with the best f1 score of .9167 for the XGBoost model.



Milestone 5:Performance testing

Activity 1:Visualizing the confusion matrix



The figure above shows the **confusion matrix** for our XGBoost classifier evaluated on the test set. The matrix summarizes the performance of the model by showing how many samples were correctly or incorrectly classified for each class:

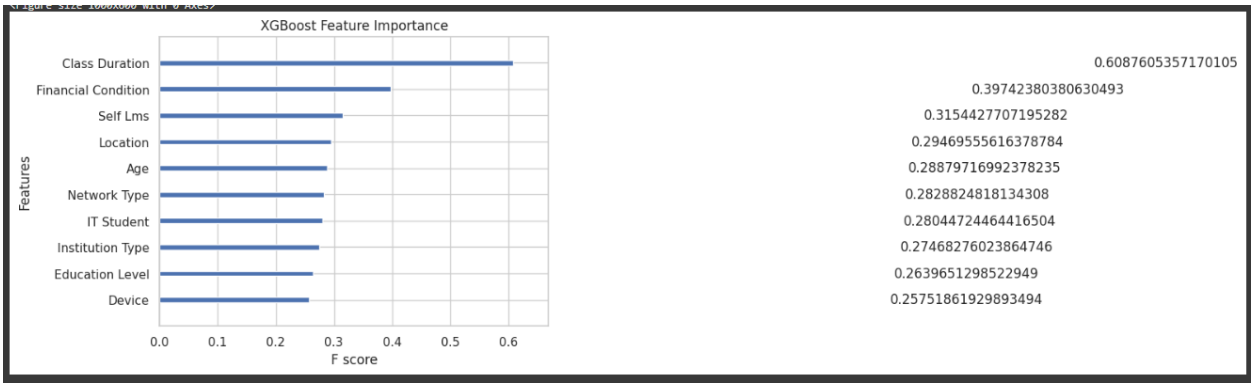
- The rows represent the **true labels**.
- The columns represent the **predicted labels**.
- Diagonal cells (from top left to bottom right) show the number of **correct predictions** for each class.
- Off-diagonal cells show **misclassifications**.

Observations:

- **Class 0**: 16 correctly predicted, 4 misclassified as class 2.
- **Class 1**: 88 correctly predicted, 1 misclassified as class 0, 7 misclassified as class 2.
- **Class 2**: 117 correctly predicted, 1 misclassified as class 0, 7 misclassified as class 1.

Overall, the model demonstrates good classification performance, with most values concentrated along the diagonal. Here the 0,1,2 refers to low,medium and high.

Activity 2: Feature Importance and visualization



The figure above shows the **feature importance** scores from the trained XGBoost model. Feature importance helps us understand which input variables contributed most to the model's predictions.

How to read this plot:

- **Features** are listed on the vertical axis.
- **F score** on the horizontal axis represents the importance of each feature, typically measured as the number of times a feature was used to split data across all trees in the model.
- A higher F score means the feature was more influential in the model's decision-making.

Observations:

- **Class Duration** has the highest importance score (~0.61). This indicates it is the most influential predictor in the model.
- **Financial Condition** and **Self LMS** also show relatively high importance (~0.40 and ~0.31, respectively).
- Other features like **Location**, **Age**, **Network Type**, **IT Student**, **Institution Type**, **Education Level**, and **Device** have lower but still meaningful contributions.

Milestone 6:Model deployment

Activity 1: Save model

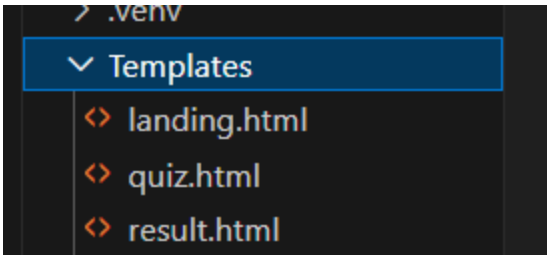
```
print(f"Selected Best Model: {model_name} (F1 Score: {max(xgb_f1, cat_f1):.4f})")

# Save to pickle
with open("best_model.pkl", "wb") as f:
    pickle.dump(best_model, f)
```

Using pkl file format we save the XGBoost model to be used in our flask app. The file is saved as best_model.pkl and will be used in the app.

Activity 2:Integrate with web

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks.

We built 3 pages , a landing page or the home page to give context , a quiz page to take the values to make the predictions and the result page to print the result/prediction.



We used the Flask framework to integrate all the components into a full app.

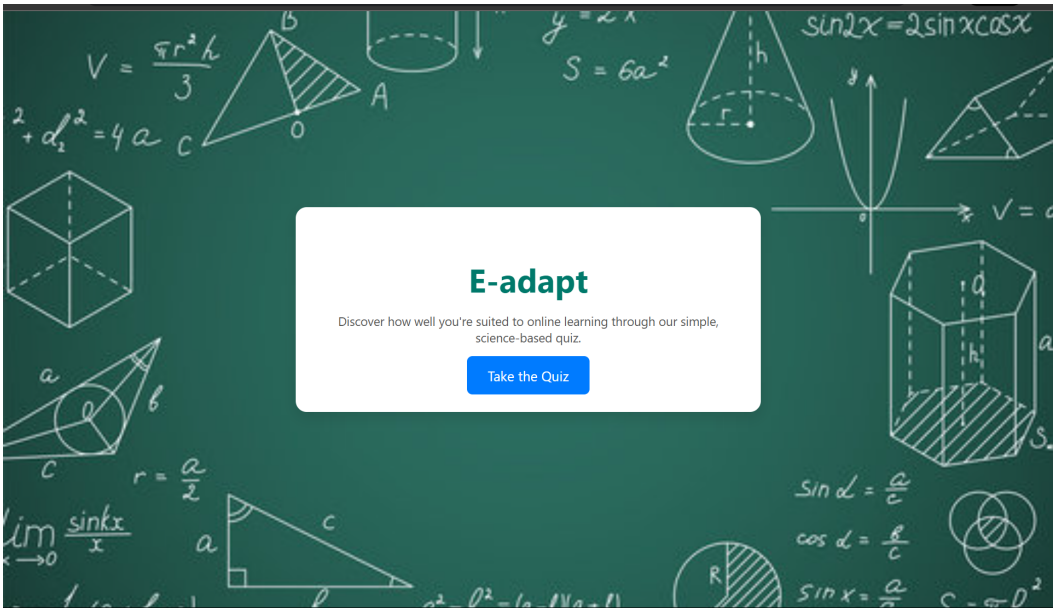
```python
from flask import Flask, render_template, request
import pandas as pd
import pickle
from sklearn.preprocessing import LabelEncoder
```

We used these libraries to load the flask app, load the pkl file and decode the encoded values.

Activity 3:Build the web page

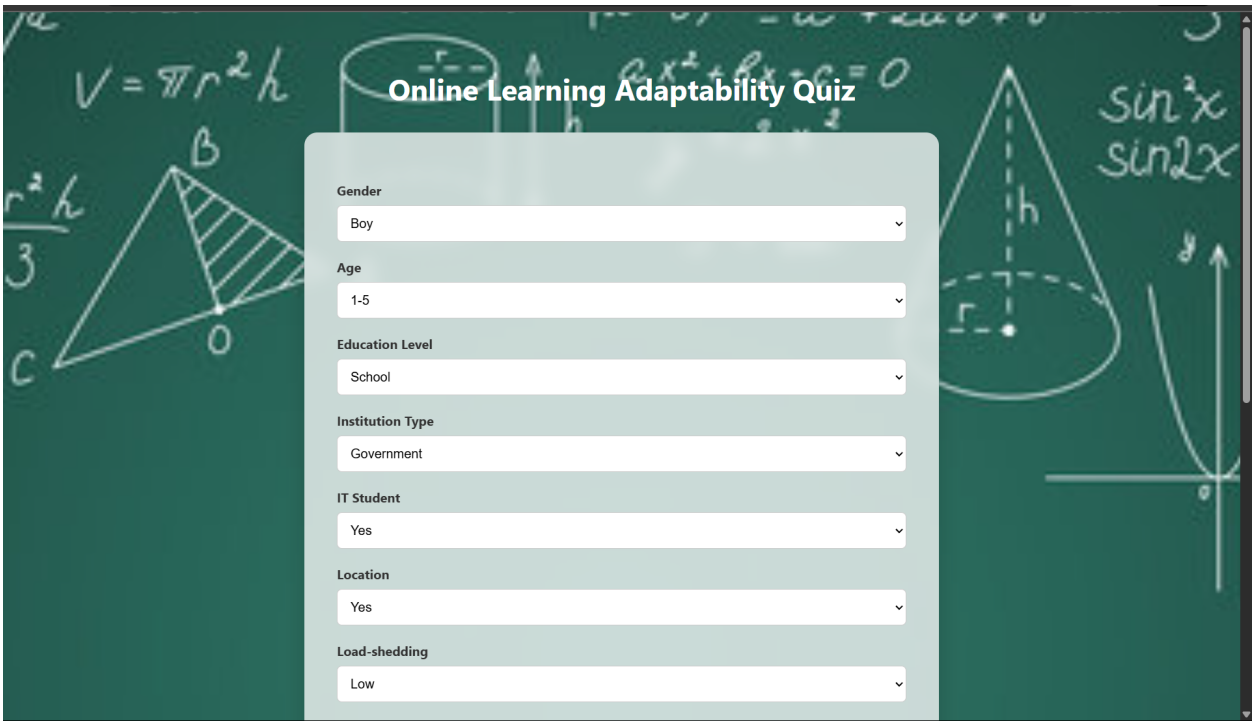
We first start with the landing or home page which is a simple page with our title , E-adapt, followed with a button to take us to the quiz

```
                margin: 20px;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>E-adapt</h1>
        <p>Discover how well you're suited to online learning through our simple, science-based quiz.</p>
        <a href="/quiz">Take the Quiz</a>
    </div>
</body>
</html>
```



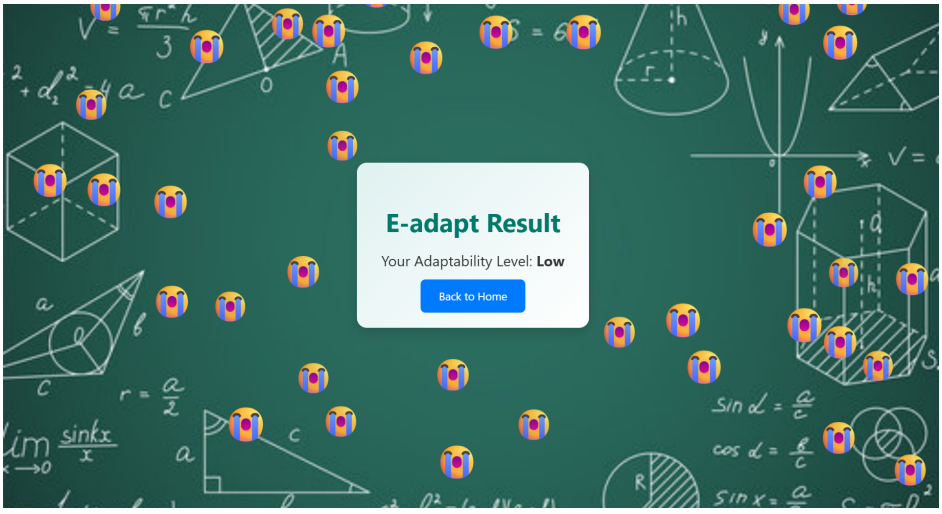
We next followed this up with the quiz page to build out our variables to be able to make the predictions.

```
81              }
82          }
83      </style>
84  </head>
85  <body>
86      <h2>Online Learning Adaptability Quiz</h2>
87      <form action="/predict" method="post">
88          {% for field in fields %}
89              <label for="{{ field }}">{{ field.replace('_', ' ') }}</label>
90              <select name="{{ field }}" id="{{ field }}" required>
91                  {% for option in options[field] %}
92                      <option value="{{ option }}">{{ option }}</option>
93                  {% endfor %}
94              </select>
95          {% endfor %}
96          <button type="submit">Submit</button>
97      </form>
98  </body>
99  </html>
100
```



And finally we built the result page for showing the adaptibility of the user based on the given information.

```
</head>
<body>
    <div class="container">
        <h1>E-adapt Result</h1>
        <div class="result-text">Your Adaptability Level: <strong>{{ result }}</strong></div>
        <a href="/">Back to Home</a>
    </div>

    <!-- Confetti Canvas -->
    <canvas id="confetti-canvas"></canvas>

    <!-- Emoji Container -->
    <div id="emoji-container"></div>

    <script>
        const result = "{{ result }}";
        const emojiContainer = document.getElementById("emoji-container");
        const canvas = document.getElementById("confetti-canvas");

        function resizeCanvas() {
            canvas.width = window.innerWidth;
            canvas.height = window.innerHeight;
        }

        function popEmojis(emoji) {
            for (let i = 0; i < 40; i++) {
                const span = document.createElement("span");
                span.classList.add("emoji");
                span.textContent = emoji;
                span.style.left = Math.random() * 100 + "vw";
                span.style.top = Math.random() * 100 + "vh";
                span.style.animationDelay = Math.random() + "s";
                emojiContainer.appendChild(span);
            }
        }

        resizeCanvas();
        window.addEventListener("resize", resizeCanvas);

        if (result.toLowerCase() === "high") {
            const confettiScript = document.createElement("script");
            confettiScript.src = "https://cdn.jsdelivr.net/npm/canvas-confetti@1.6.0/dist/confetti.browser.min.js";
            confettiScript.onload = () => {
                const myConfetti = confetti.create(canvas, { resize: true, useWorker: true });
                myConfetti({
                    particleCount: 300,
                    spread: 200,
                    origin: { y: 0.6 }
                });
            };
            document.body.appendChild(confettiScript);
        } else if (result.toLowerCase() === "moderate") {
            popEmojis("🔥");
        } else if (result.toLowerCase() === "low") {
            popEmojis("😭");
        }
    </script>
</body>
</html>
```



Activity 4:Python Code

```python
from flask import Flask, render_template, request
import pandas as pd
import pickle
from sklearn.preprocessing import LabelEncoder

app = Flask(__name__)

# Load model
with open("best_model.pkl", "rb") as f:
    model = pickle.load(f)
```

We first import our libraries and create our flask app. We then use the pickle module to load our XGBoost model and prepare ourself for predictions.

```python

# Load original dataset
df = pd.read_csv("students_adaptability_level_online_education.csv")

# Set target and categorical feature columns
target_col = 'Adaptivity Level'
object_cols = df.select_dtypes(include=['object']).columns.tolist()
if target_col in object_cols:
    object_cols.remove(target_col)

# Prepare encoders
encoders = {}
for col in object_cols:
    le = LabelEncoder()
    le.fit(df[col].astype(str))
    encoders[col] = le

# Target encoder
target_encoder = LabelEncoder()
target_encoder.fit(df[target_col].astype(str))
```

Next we import the original data and set out target feature that being the adaptibility level , and also encode it so we can easily decode our encoded output once the

prediction is made.

```python
# Dropdown options for the quiz
dropdown_options = {
    'Gender': ['Boy', 'Girl'],
    'Age': ["1-5", "6-10", "11-15", "16-20", "21-25", "26-30"],
    'Education Level': ['School', 'College', 'University'],
    'Institution Type': ['Government', 'Non Government'],
    'IT Student': ['Yes', 'No'],
    'Location': ['Yes', 'No'],
    'Load-shedding': ['Low', 'High'],
    'Financial Condition': ['Poor', 'Mid', 'Rich'],
    'Internet Type': ['Mobile Data', 'Wifi'],
    'Network Type': ['2G', '3G', '4G'],
    'Class Duration': ['0', '1-3', '3-6'],
    'Self Lms': ['Yes', 'No'],
    'Device': ['Mobile', 'Computer', 'Tab']
}
```

Give all the possible options for the dropdown in our quiz.html file so that the user can pick their details accurately.

```python
Tabnine | Edit | Test | Explain | Document
@app.route("/")
def landing():
    return render_template("landing.html")
```

Render out our landing page.

```python
Tabnine | Edit | Test | Explain | Document
@app.route("/quiz")
def quiz():
    return render_template("quiz.html", fields=dropdown_options.keys(), options=dropdown_options)
```

Render our our quiz page with the dropdown options keys as the label and then the

options as the values for the given keys.

```
Tabnine | Edit | Test | Explain | Document
@app.route("/predict", methods=["POST"])
def predict():
    user_input = {field: [request.form[field]] for field in dropdown_options}
    sample_df = pd.DataFrame(user_input)

    # Encode input using the trained label encoders
    for col in sample_df.columns:
        if col in encoders:
            sample_df[col] = encoders[col].transform(sample_df[col].astype(str))

    # Predict
    prediction = model.predict(sample_df)
    result = target_encoder.inverse_transform(prediction)[0].strip().capitalize()


    return render_template("result.html", result=result)
```

Take in the user input and then use it in the model after encoding to obtain the output , decode it and then print out the result by rendering it with the result.html page.

Activity 4:Run the app

To run the app , run it like a normal python file as it will work as long as the py file is called directly and satisfies the main condition

```
if __name__ == "__main__":
    app.run(debug=True)
```