E-ADAPT: PREDICTING STUDENT ADAPTABILITY IN ONLINE CLASSES

Team ID: SWTID1749622261

Team Members:

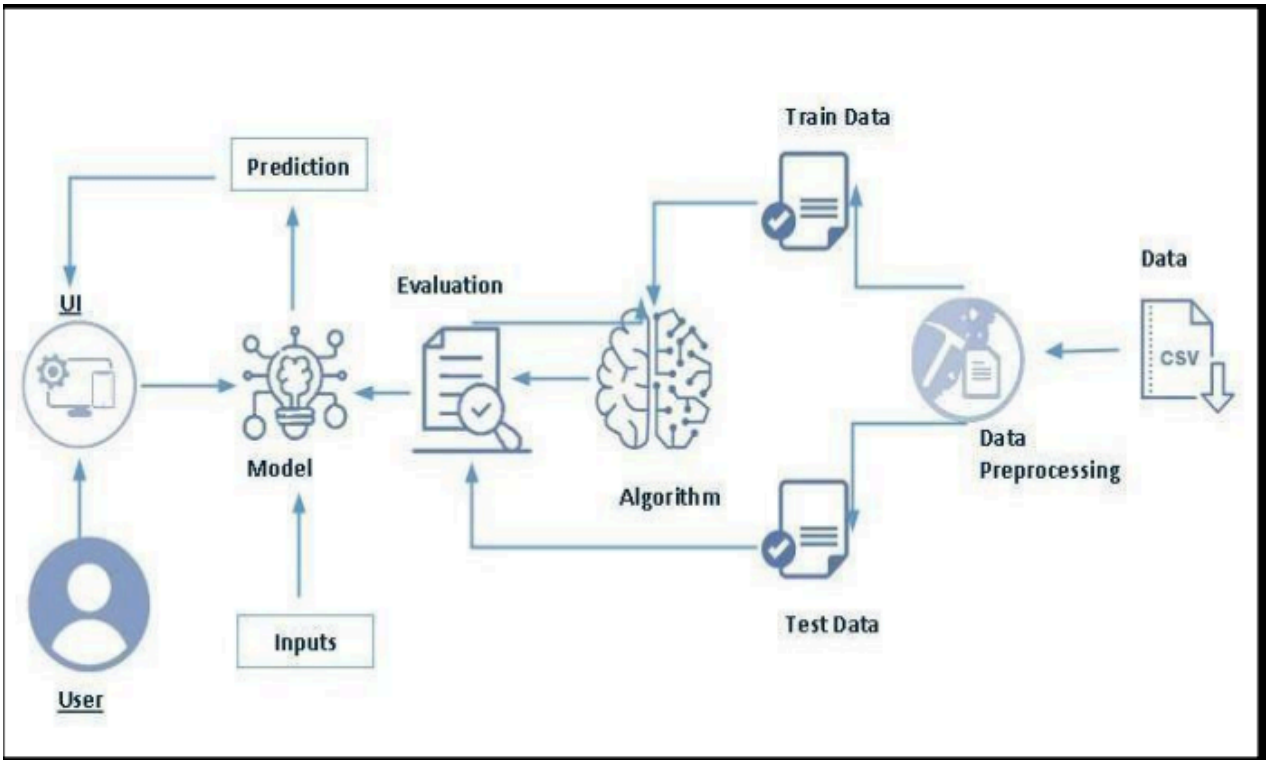
- K R Faseeha Nazli (Team Leader)
- Sahishna Rajesh
- Devipreya R
- Himani Dasari

E-ADAPT: PREDICTING STUDENT ADAPTABILITY IN ONLINE CLASSES

The rapid rise of technology in the world has transformed education, resulting in widespread adoption of online learning. Online courses offer flexibility and easy accessibility, but they also present challenges such as time management, motivation, and technical skill issues. All students would not be equally equipped enough to adapt to these conditions. To address this challenge, the E-Adapt project attempts to develop a machine learning-based prediction model that evaluates the ability of students to adapt to online learning platforms. By observing factors such as technical ability, self-regulation, motivation, and engagement, the model will identify students who may need additional support.

The primary aim of E-Adapt is to provide educators with insights into student adaptability that can be acted upon, enabling targeted interventions to improve learning outcomes. By making use of data from learning platforms, assessments, and surveys, the model will assist in enhancing the design of online courses and enable successful implementation. In summary, this project seeks to customize and enhance the digital learning experience in a consistently evolving educational environment.

Technical Architecture:



PROJECT FLOW

- User is shown the Home page. The user will browse through the Home page and go to predict my adaptivity and enter the specified engagement metrics.
- After clicking the Predict button the user will be directed to the Results page where the model will analyse the inputs given by the user and showcase the prediction of the Adaptivity level

.

To accomplish this we have to complete all the activities listed below:

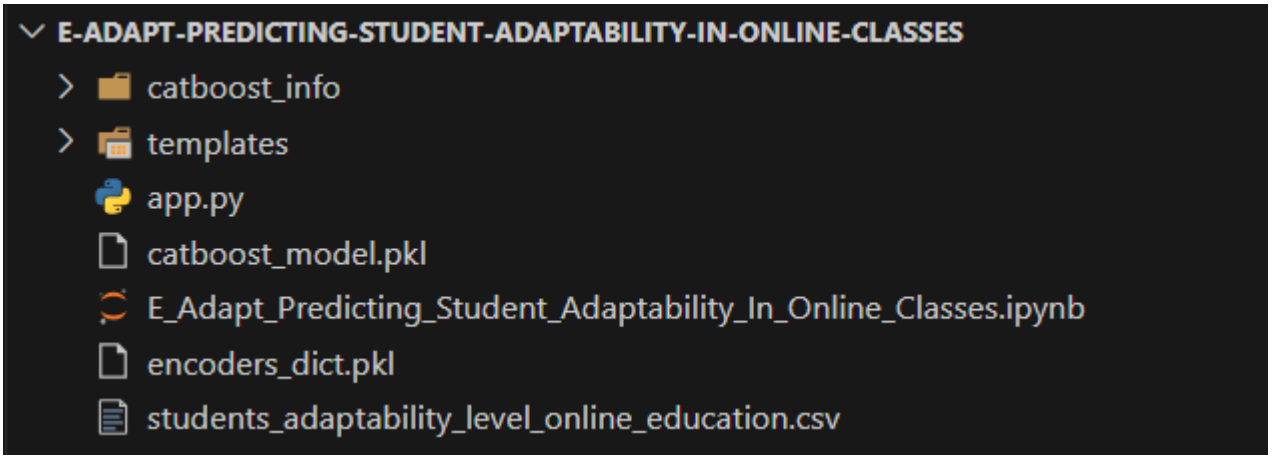
- **Problem Definition / Understanding**
    - Specify the Business Problem
    - Business Requirements
    - Literature Survey
    - Social or Business Impact
- **Data Collection and Preparation**
    - Collect the Dataset
    - Data Preparation
- **Exploratory Data Analysis (EDA)**
    - Descriptive Statistical Analysis
    - Visual Analysis
- **Model Building**
    - Creating a Function for Evaluation
    - Training and Testing the Models using Multiple Algorithms
- **Performance Testing & Hyperparameter Tuning**
    - Testing Model with Multiple Evaluation Metrics
    - Comparing Model Accuracy
        - Before & After Applying Hyperparameter Tuning
        - For Different Number of Features
    - Building Model with Appropriate Features
- **Model Deployment**
    - Save the Best Model
    - Integrate with Web Framework
-

Prior Knowledge:

- · ML Concepts
    - Unsupervised learning: https://www.geeksforgeeks.org/machine-learning/unsupervised-learning/
    - Supervised learning: https://www.geeksforgeeks.org/supervised-machine-learning/
        - Catboost: https://www.geeksforgeeks.org/machine-learning/catboost-ml/
        - Random forest: https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/
        - Xgboost: https://www.geeksforgeeks.org/machine-learning/xgboost/
        - Evaluation metrics: https://www.geeksforgeeks.org/machine-learning/metrics-for-machine-learning-model/
- Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

PROJECT STRUCTURE

Create project folder which contains files as shown below:



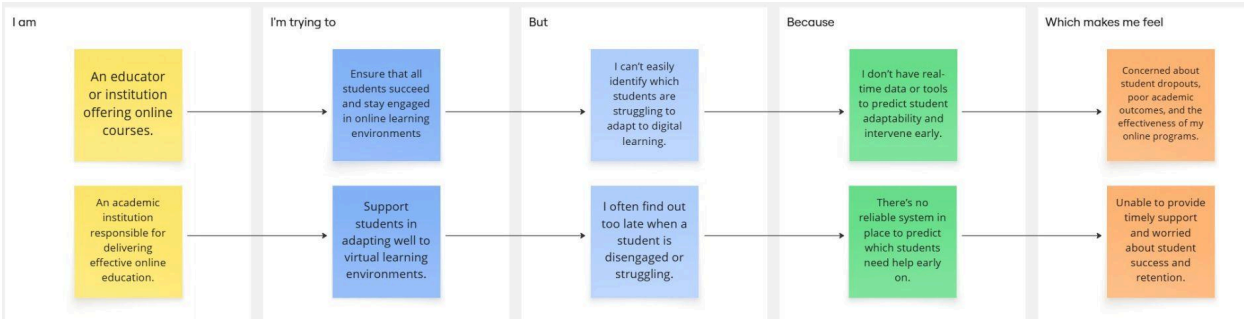
- The data obtained is in .csv files for training and testing.
- We are building a Flask application that will require the HTML files to be stored in the templates folder.
- The app.py file is used for routing purposes using scripting.
- catboost_model.pkl is the saved model. This will further be used in the Flask integration.

1. Define Problem/ Problem Understanding

Activity 1: Specify the business problem

Institutions are increasingly finding it difficult to promote student success and retention in online learning settings. More classes now move to online platforms, leaving institutions challenged to manage and track student flexibility effectively. Several students face challenges like low participation, demotivation, and limited ability to navigate online systems—factors that may not easily be noticed until performance drops. These issues constrain the institution from making early interventions, thereby affecting student outcomes, course completion rates, and overall effectiveness of online programs. To promote student support and refine digital learning strategies, we seek to resolve these important pain points. By predicting and identifying students' levels of adaptability through data-informed approaches, institutions can deliver focused interventions, tailor learning experiences, and create more inclusive and impactful online courses—all in alignment with academics and student needs.

Example:



Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	An educator or institution offering online courses.	Ensure that all students succeed and stay engaged in online learning environments.	I can't easily identify which students are struggling to adapt to digital learning.	I don't have real-time data or tools to predict student adaptability and intervene	Concerned about student dropouts, poor academic outcomes, and the effectiveness of my online programs.

				early.	
PS-2	An academic institution responsible for delivering effective online education.	Support students in adapting well to virtual learning environments.	I often find out too late when a student is disengaged or struggling.	There's no reliable system in place to predict which students need help early on.	Unable to provide timely support and worried about student success and retention.

Activity 2: Business requirements

A student adaptability prediction project like **e-Adapt** can have many crucial business requirements, depending on the specific goals and objectives of the project with respect to online education. Some potential requirements may include:

- **Accurate predictions**: The system should provide reliable and precise predictions of individual student adaptability levels to online classes, using validated machine learning models.

- **Robust data collection**: The system should gather required data from different sources such as LMS platforms, online evaluations, and surveys to enable a comprehensive analysis.

- **Effective preprocessing and feature extraction**: Data should be cleaned, transformed, and features like academic performance, technical skills, and motivation should be accurately extracted and compatibility with algorithms should be ensured, to effectively train the model.

- **Model performance and deployment**: The system must use appropriate, well-performing machine learning algorithms and deploy the model in a production environment for real-time prediction.

- **User-friendly interface**: The system must offer an interactive dashboard for educators or institutions to input data, view the adaptability scores, and access relevant insights easily.

- **Scalability and reliability**: The system must be scalable and support a large number of users and consistently operate dependably under various loads.

Activity 3: Literature Survey

The literature review of predicting student adaptability in online courses highlights the role of student adaptability as a determinant of student success in virtual learning environments. With the growth of online learning, it becomes critical to know the determinants of student adaptability. Student adaptability is the capacity for students to modify their learning approaches and behaviors as a response to the specific challenges and opportunities of online learning environments. Studies show that high adaptability learners tend to perform better academically and be more participative in relation to course material.

A few main determinants drive student adaptability in online courses. Academic performance history is frequently cited as an indicator of continued success within virtual learning spaces. Research indicates that students who have a history of high academic performance also adjust better to online learning because they have the foundational knowledge and skills required to handle sophisticated content. This correlation suggests that educators should consider students' academic backgrounds when designing online courses and interventions.

Technological competence is another important influencing factor on the adaptability of students. In a growingly digitalized world, technology-friendly students are in a better position to access online learning platforms and work effectively with digital materials. Studies have proven that students with higher technological skills are more engaged and perform better in online classes. The observation highlights the significance of offering students chances to build their technological competencies either prior to or alongside their online learning processes.

Self-regulated learning (SRL) competencies are also crucial for the adaptability of students. SRL refers to a set of behaviors, such as goal-setting, self-monitoring, and self-reflection, that are critical for the regulation of one's learning process within a more open-ended online environment. Research has established that students who are competent in SRL competencies are likely to perform well within online courses since they are able to manage time effectively, set achievable goals, and request assistance when necessary. Thus, developing SRL skills must be an object of priority for teachers who seek to increase student flexibility in virtual learning settings.

Motivation and engagement play central roles in developing adaptability in students. Intrinsic motivation is especially associated with increased levels of online learning engagement and perseverance. Students who are engaged are more probable to effectively use available resources and ask for help when required, thus increasing their adaptability. Recognizing the motivational forces that influence student engagement can aid educators in creating more efficient online learning experiences that ensure adaptability.

Personal attributes, including resilience, self-efficacy, and personality, also significantly influence student adaptability. Resilient students are more likely to cope with failures and difficulties in online education, whereas students with high self-efficacy will be more inclined to have faith in their capacity for success. Research indicates that the study of these personal attributes can facilitate deeper insights into student adaptability profiles, enabling teachers to make corresponding support strategies.

The survey discusses current methods of estimating student adaptability, ranging from conventional statistical models to sophisticated machine learning methods. Conventional statistical models, including regression analysis, have been used to forecast student success using several factors. These models tend to make use of past data and can give insights into the influence of a variety of variables on adaptability. They might not catch the nuance of the interaction among many factors.

Conversely, machine learning algorithms have picked up steam in educational data mining. Logistic Regression, K-Nearest Neighbors, Random Forest Classifier, XGBoost, and CatBoost have been used to create predictive models of student flexibility. These can deal with large data sets and look for patterns that the standard approaches might miss. For example, Random Forest Classifiers have indicated that they can precisely predict student performance from the combination of academic and behavioral data.

Successful prediction models are dependent on varied data sources such as learning management systems (LMS), online quizzes, and student surveys. Data from LMS has the potential to uncover student participation and interaction behavior, while quizzes can unveil levels of academic performance. Surveys have the potential to collect self-reported measures of motivation, IT skills, and personal traits, supplementing the dataset for predictive modeling. Merging these diverse data sources is essential for formulating robust predictive models that can effectively gauge the adaptability of students.

The literature is concerned with the necessity for precise prediction models to improve student outcomes and optimize the effectiveness of online learning. Predicting students that will struggle in online learning environments allows instructors to apply tailored interventions and support strategies. This proactive strategy not only serves students' success but also guides course design, so that online courses are designed to facilitate adaptability and engagement.

In summary, the research on forecasting student adaptability in online courses highlights the need to comprehend the complex factors underpinning student success in virtual learning environments. As online learning remains on the forefront of educational development, the creation of sturdy predictive models will become critical for promoting student adaptability and overall learning outcomes. Through the application of knowledge gained through research, instructors can design more efficient online learning platforms that facilitate all students to reach their academic potential.

Activity 4: Social or Business Impact:

Social Impact:

The e-Adapt implementation can bring about a notable social impact by promoting educational equity. Early detection of students who are at risk of falling behind allows for the pinpointing of support mechanisms, saving them from dropout and encouraging inclusive education. It ensures all students have an equal chance of success in online learning atmospheres, irrespective of their background or digital proficiency.

Additionally, this model helps reduce the digital divide by providing educators with the capabilities to assist students who are not digitally literate or have no means. It also promotes creating accessible and inclusive course content that accommodates different needs. In the long run, improving flexibility in online learning can result in improved workforce preparedness since students become increasingly comfortable and adept in tech-savvy environments.

Business Impact:

The e-Adapt model also provides quantifiable business value for institutions of learning. Through the use of data-driven analysis, it enables institutions to adapt course design, enhance student engagement and satisfaction, and ultimately enhance retention and completion. These improvements lead to a more robust institutional reputation, enhanced student enrollment, and higher revenue streams, supporting the financial viability of online education programs.

Furthermore, institutions adopting adaptive learning frameworks such as e-Adapt can establish a competitive advantage within the fast-expanding edtech market. It facilitates scalable personalization, which appeals to a more diverse student population, including working individuals and foreign students. By facilitating more effective allocation of resources and minimizing dropout costs, e-Adapt helps institutions to pursue long-term growth, sustainability, and innovation in the delivery of digital education.

2: Data Collection & Preparation

Activity 1: Collect the dataset

Machine Learning depends heavily on data. It is the most crucial aspect that makes algorithm training possible.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below.
https://www.kaggle.com/datasets/mdmahmudulhasansuzan/students-adaptability-level-in-online-education

Activity 1.1: Importing the libraries

Import the necessary libraries as required.

1. Import Necessary Libraries

```
!pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from catboost) (1.15.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (4.58.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly->catboost) (9.1.2)
Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl (99.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.2/99.2 MB 6.6 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.8
```

```
[ ] !pip install optuna
```

```
Collecting optuna
  Downloading optuna-4.4.0-py3-none-any.whl.metadata (17 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.16.1-py3-none-any.whl.metadata (7.3 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from optuna) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from optuna) (24.2)
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.11/dist-packages (from optuna) (2.0.41)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from optuna) (4.67.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.11/dist-packages (from optuna) (6.0.2)
Requirement already satisfied: Mako in /usr/lib/python3/dist-packages (from alembic>=1.5.0->optuna) (1.1.3)
Requirement already satisfied: typing-extensions>=4.12 in /usr/local/lib/python3.11/dist-packages (from alembic>=1.5.0->optuna) (4.14.0)
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.11/dist-packages (from sqlalchemy>=1.4.2->optuna) (3.2.2)
Downloading optuna-4.4.0-py3-none-any.whl (395 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 395.9/395.9 kB 6.8 MB/s eta 0:00:00
Downloading alembic-1.16.1-py3-none-any.whl (242 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 242.5/242.5 kB 14.9 MB/s eta 0:00:00
Downloading colorlog-6.9.0-py3-none-any.whl (11 kB)
Installing collected packages: colorlog, alembic, optuna
Successfully installed alembic-1.16.1 colorlog-6.9.0 optuna-4.4.0
```

```
!pip install termcolor
```

```
Requirement already satisfied: termcolor in /usr/local/lib/python3.11/dist-packages (3.1.0)
```

```
# Data
import numpy as np
import pandas as pd

#Visualization
import seaborn as sns
import matplotlib as mpl
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from plotly.subplots import make_subplots

#Preprocessing
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

#Modeling Libraries
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

#Cluster Libraries
#from sklearn.cluster import eans
#from sklearn.decomposition import PCA
#from sklearn.metrics import silhouette_score
#from yellowbrick.cluster import KELbowvisualizer
```

Activity 1.2: Read the Dataset

Our dataset format is in .csv. We can read the dataset with the help of pandas.
We use read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
[ ] #Tuning Libraries
    import optuna
```

```
[ ] from google.colab import files
    uploaded = files.upload()
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving students_adaptability_level_online_education.csv to students_adaptability_level_online_education.csv
```

```
from termcolor import colored
df = pd.read_csv("students_adaptability_level_online_education.csv")
print(colored('Shape of DataFrame: ','blue'), df.shape, '\n\n')
df.head()
```

```
Shape of DataFrame:  (1205, 14)
```

	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 2: Data Preparation

This activity includes the following steps.

- Handling missing values
- Handling Outliers

These are the general steps of pre-processing the data before using it for machine learning.

According to the condition of our dataset, we decide whether or not to go through with these steps.

Activity 2.1: Handling missing values

For checking the null values, df.isna().any( ) function is used. To sum those null values we use .sum() function. We found that there are no null values present in our dataset, so we can skip handling the missing values step.

```
df.isna().any()
```

	0
Gender	False
Age	False
Education Level	False
Institution Type	False
IT Student	False
Location	False
Load-shedding	False
Financial Condition	False
Internet Type	False
Network Type	False
Class Duration	False
Self Lms	False
Device	False
Adaptivity Level	False

dtype: bool

```
print(df.isnull().sum())
```

```
Gender                0
Age                   0
Education Level       0
Institution Type      0
IT Student            0
Location              0
Load-shedding         0
Financial Condition   0
Internet Type         0
Network Type          0
Class Duration        0
Self Lms              0
Device                0
Adaptivity Level      0
dtype: int64
```

Activity 2.2: Handling Outliers

With the help of boxplot, outliers are visualized. However, boxplots are used to identify outliers in numeric columns. None of the columns in our dataset are purely continuous and numeric.

Boxplots visualize the spread and potential outliers in numerical distributions. Since our data is made of qualitative categories, there's:

- No concept of "extremely high or low values"
- No mathematical mean/median/standard deviation to compare against

3. Exploratory Data Analysis

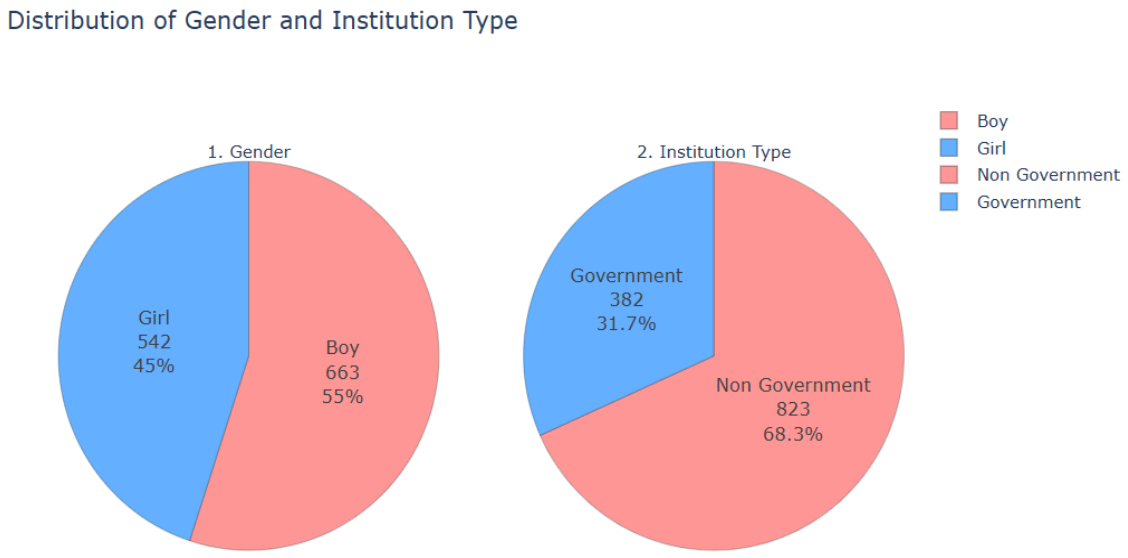
Activity 1: Visualisation

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.

Activity 1.1 Univariate Analysis

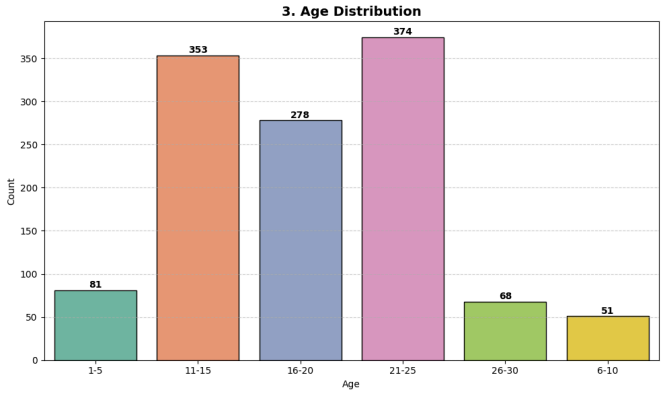
- **Gender and Institute type Distribution**
  The code creates a subplot with two pie charts using Plotly to show the distribution of Gender and Institution Type from a DataFrame. Each chart displays labels, counts, and percentages with a custom color scheme and titles.

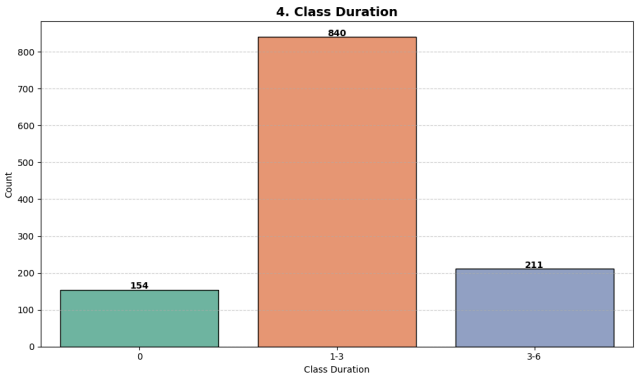
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots


series_1 = []
series_1_title = ["Gender", "Institution Type"]
series_1.append(df['Gender'].value_counts())
series_1.append(df['Institution Type'].value_counts())

color_scheme = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']


fig = make_subplots(rows=1, cols=2, specs=[[{"type": "pie"}, {"type": "pie"}]])

for idx, series in enumerate(series_1):
    fig.add_trace(
        go.Pie(
            values=series.values,
            labels=series.index,
            marker=dict(colors=color_scheme),
            title=str(idx+1) + '. ' + series_1_title[idx]
        ),
        row=1,
        col=idx + 1
    )


fig.update_traces(textinfo="label+percent+value", textfont_size=13,
                  marker=dict(line=dict(color="#000000", width=0.2)))
fig.update_layout(title_text="Distribution of Gender and Institution Type")
fig.show()
```



- **Age And Class Distribution**
  The code defines a reusable function to create customized bar plots using Seaborn and Matplotlib. It visualizes the distribution of Age and Class Duration from the DataFrame, adding value labels on bars, grid lines, and proper titles and axis labels for clarity.

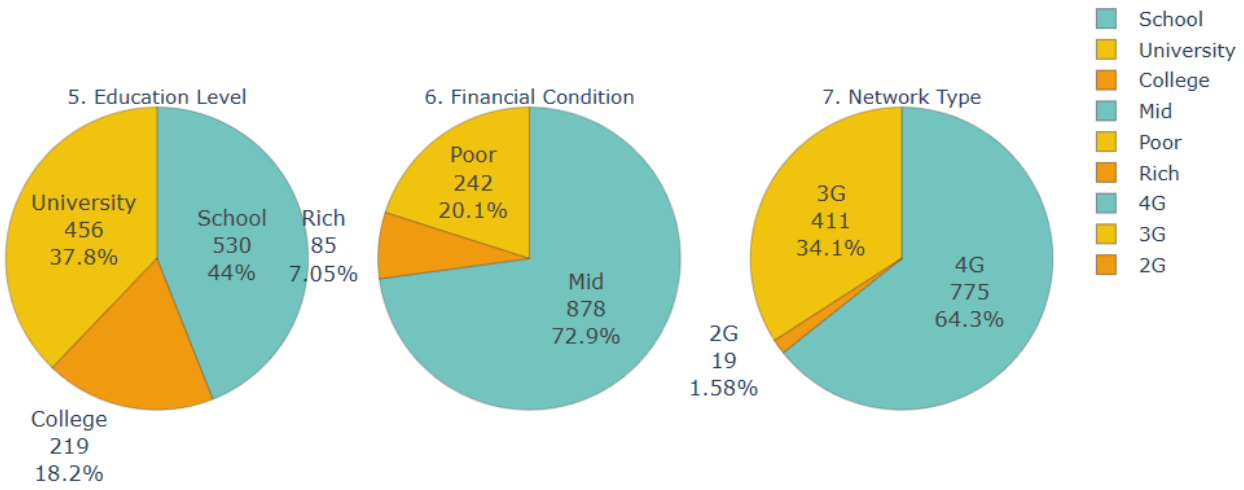
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


def p_bar_plot(x, y, xlabel, ylabel, title):
    plt.figure(figsize=(10, 6))
    sns.barplot(x=x, y=y, palette='Set2', edgecolor='black')


    for i, v in enumerate(y):
        plt.text(i, v + 3, str(v), ha='center', fontweight='bold')

    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title, fontsize=14, fontweight='bold')
    plt.grid(axis='y', linestyle='--', alpha=0.6)
    plt.tight_layout()
    plt.show()


age = df['Age'].value_counts().sort_index()
class_duration = df['Class Duration'].value_counts().sort_index()

p_bar_plot(age.index, age.values, 'Age', 'Count', '3. Age Distribution')
p_bar_plot(class_duration.index, class_duration.values, 'Class Duration', 'Count', '4. Class Duration')
```





- **Education Level, Financial Condition, and Network Type**
  The code creates a subplot with three pie charts using Plotly to visualize the distributions of Education Level, Financial Condition, and Network Type. Each chart includes labels, values, and percentages with a defined color scheme and individual titles.

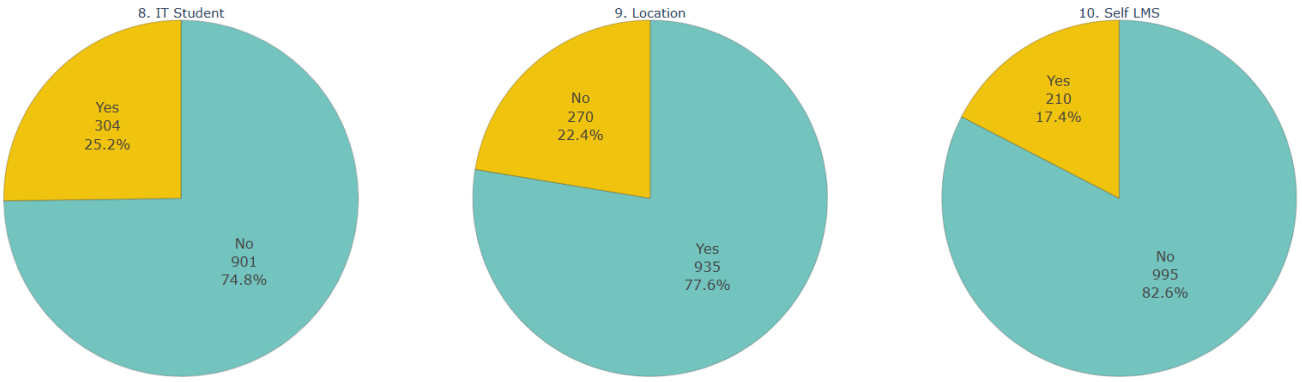
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

series_2 = []
series_2_title = ['Education Level', 'Financial Condition', 'Network Type']

series_2.append(df['Education Level'].value_counts())
series_2.append(df['Financial Condition'].value_counts())
series_2.append(df['Network Type'].value_counts())

color_scheme = ['#76c7c0', '#f1c40f', '#f39c12', '#9ecae1']

fig = make_subplots(rows=1, cols=3, specs=[[{"type": "pie"}, {"type": "pie"}, {"type": "pie"}]])

for idx, series in enumerate(series_2):
    fig.add_trace(
        go.Pie(
            values=series.values,
            labels=series.index,
            marker=dict(colors=color_scheme),
            title=str(idx + 5) + ". " + series_2_title[idx]
        ),
        row=1,
        col=idx + 1
    )

fig.update_traces(
    textinfo="label+percent+value",
    textfont_size=13,
    marker=dict(line=dict(color="#10000e", width=0.2))
)

fig.show()
```



- **IT Student, Location and Self LMS**

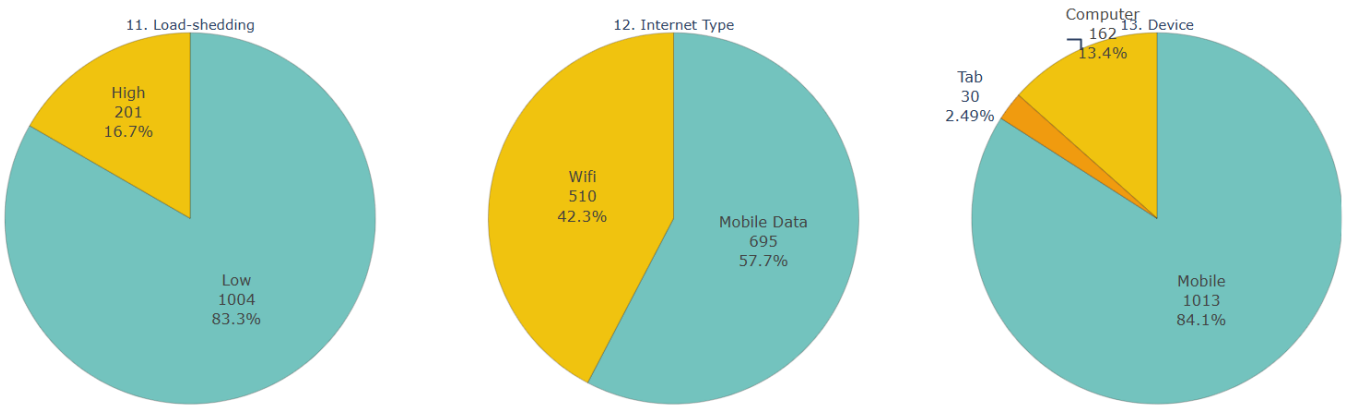
  The code generates three pie charts using Plotly to show the distribution of IT Student, Location, and Self LMS variables. Each chart includes category labels, counts, and percentages, with a consistent color scheme and individual titles.

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

series_3 = []
series_3_title = ['IT Student', 'Location', 'Self LMS']
series_3.append(df['IT Student'].value_counts())
series_3.append(df['Location'].value_counts())
series_3.append(df['Self Lms'].value_counts())

color_scheme = ['#76c7c0', '#f1c40f']

fig = make_subplots(rows=1, cols=3, specs=[[{"type": "pie"}, {"type": "pie"}, {"type": "pie"}]])

for idx, series in enumerate(series_3):
    fig.add_trace(
        go.Pie(
            values=series.values,
            labels=series.index,
            marker=dict(colors=color_scheme),
            title=str(idx + 8) + ". " + series_3_title[idx]
        ),
        row=1,
        col=idx + 1
    )

fig.update_traces(
    textinfo="label+percent+value",
    textfont_size=13,
    marker=dict(line=dict(color="#100000", width=0.2))
)

fig.show()
```



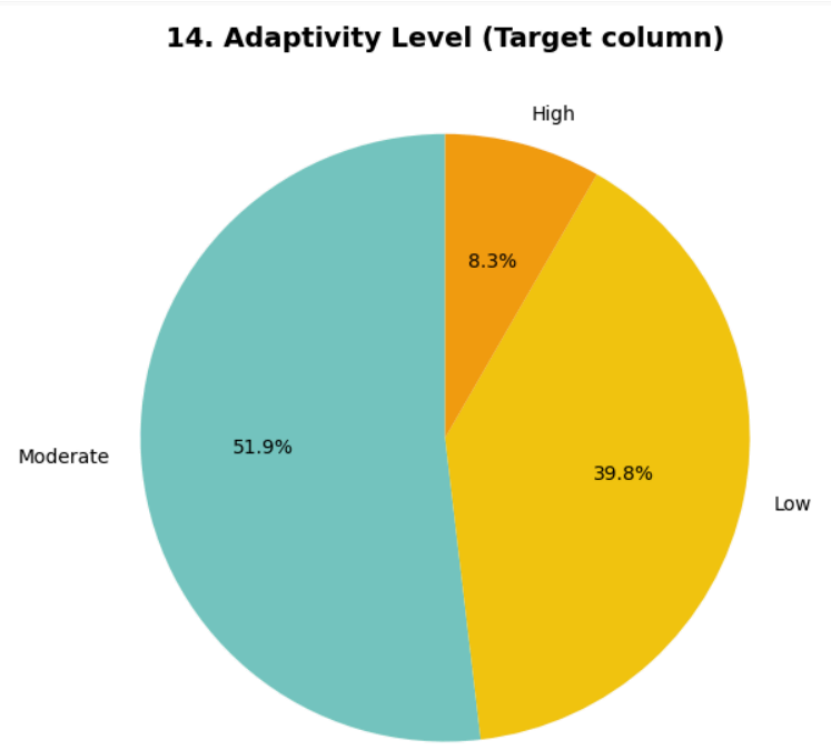
- **Load Shedding, Internet Type and Device**
  The code creates a row of three pie charts using Plotly to display the distribution of Load-shedding, Internet Type, and Device. Each chart includes labels, counts, and percentages with a specified color scheme and individual titles.

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

series_4 = []
series_4_title = ['Load-shedding', 'Internet Type', 'Device']
series_4.append(df['Load-shedding'].value_counts())
series_4.append(df['Internet Type'].value_counts())
series_4.append(df['Device'].value_counts())

color_scheme = ['#76c7c0', '#f1c40f', '#f39c12', '#9ecae1', '#d2b4de']

fig = make_subplots(rows=1, cols=3, specs=[[{"type": "pie"}, {"type": "pie"}, {"type": "pie"}]])

for idx, series in enumerate(series_4):
    fig.add_trace(
        go.Pie(
            values=series.values,
            labels=series.index,
            marker=dict(colors=color_scheme),
            title=str(idx + 11) + ". " + series_4_title[idx]
        ),
        row=1,
        col=idx + 1
    )

fig.update_traces(
    textinfo="label+percent+value",
    textfont_size=13,
    marker=dict(line=dict(color="#100000", width=0.2))
)

fig.show()
```



- **Adaptability Level**
  The code defines a function to create a pie chart using Matplotlib and visualizes the distribution of the Adaptivity Level column. The chart includes percentage labels, custom colors, and a centered title.

```
import matplotlib.pyplot as plt

def p_pie_plot(series, title):
    plt.figure(figsize=(6, 6))
    colors = ['#76c7c0', '#f1c40f', '#f39c12', '#d2b4de', '#9ecae1']
    plt.pie(series.values, labels=series.index, autopct='%1.1f%%', startangle=90, colors=colors)
    plt.title(title, fontsize=14, fontweight='bold')
    plt.axis('equal')
    plt.tight_layout()
    plt.show()

p_pie_plot(df['Adaptivity Level'].value_counts(), '14. Adaptivity Level (Target column)')
```

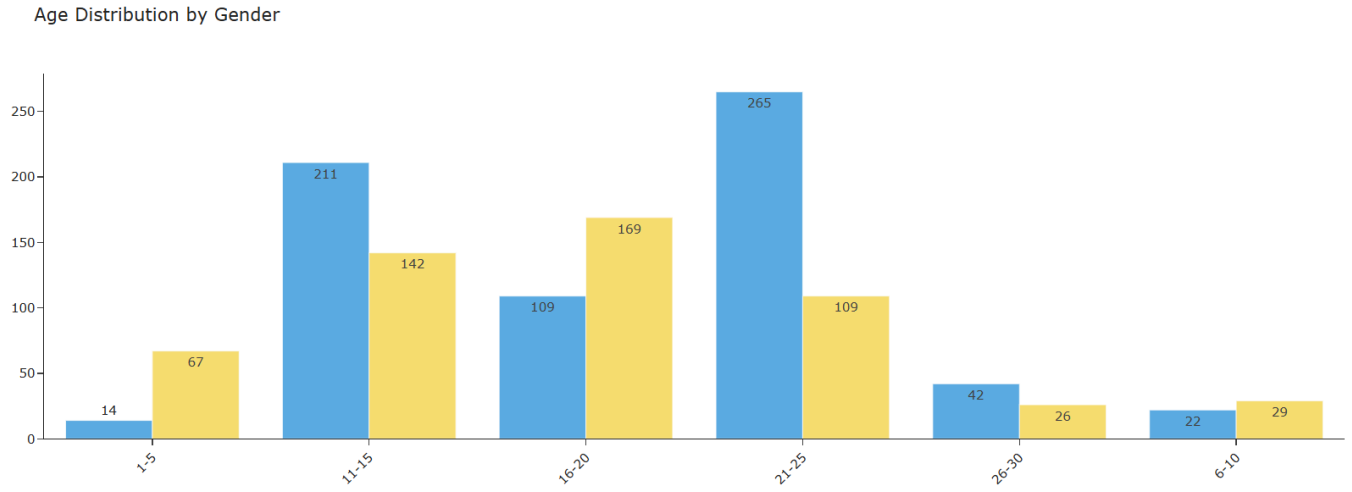


Activity 1.2 Multivariate Analysis

- **Age Distribution by Gender**
  The code creates a grouped bar chart using Plotly to display the distribution of Age by Gender. It compares the counts for boys and girls across different age groups with distinct colors and value labels.

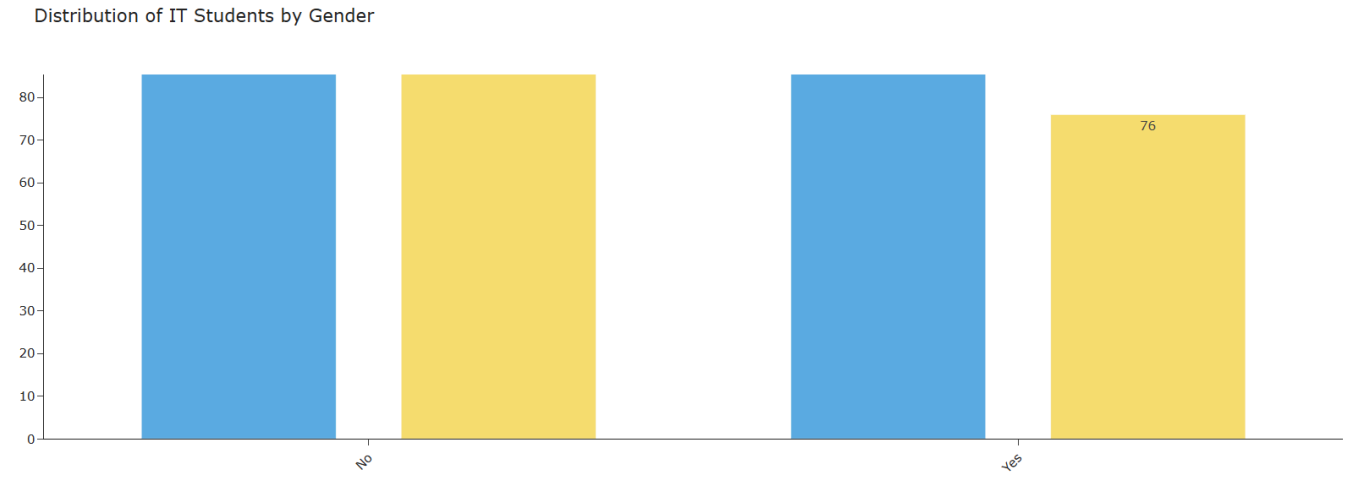
Multivariate Analysis

```
gender_age = df.groupby(['Gender', 'Age']).size()
fig = go.Figure(data=[
    go.Bar(name='Boy', x=gender_age['Boy'].index, y=gender_age['Boy'].values,
           text=gender_age['Boy'].values, marker_color='#5DADE2'),
    go.Bar(name='Girl', x=gender_age['Girl'].index, y=gender_age['Girl'].values,
           text=gender_age['Girl'].values, marker_color='#F7DC6F')
])

fig.update_layout(barmode='group', xaxis_tickangle=-45, title='Age Distribution by Gender',
                  template='simple_white')

fig.show()
```



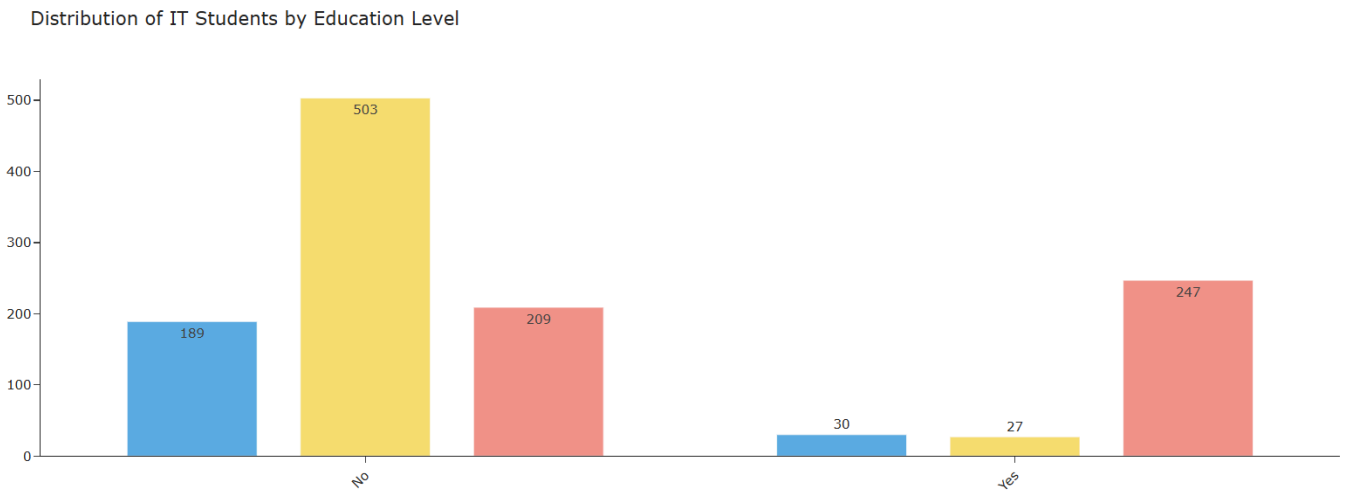
- **IT Student Distribution by Gender**
  The code generates a grouped bar chart using Plotly to show the distribution of IT Student status by Gender. It compares the counts for boys and girls, with labeled bars, distinct colors, and adjusted bar width.

```
gender_IT = df.groupby(['Gender', 'IT Student']).size()
fig = go.Figure(data=[
    go.Bar(name='Boy', x=gender_IT['Boy'].index, y=gender_IT['Boy'].values,
           text=gender_IT['Boy'].values, marker_color='#5DADE2'),
    go.Bar(name='Girl', x=gender_IT['Girl'].index, y=gender_IT['Girl'].values,
           text=gender_IT['Girl'].values, marker_color='#F7DC6F')
])

fig.update_layout(barmode='group', xaxis_tickangle=-45, title='Distribution of IT Students by Gender',
                  template='simple_white')

fig.update_traces(width=0.3)

fig.show()
```



- **IT Student Distribution by Education**
  The code creates a grouped bar chart using Plotly to visualize the distribution of IT Student status across different Education Levels. It displays counts for College, School, and University groups with distinct colors and labeled bars.

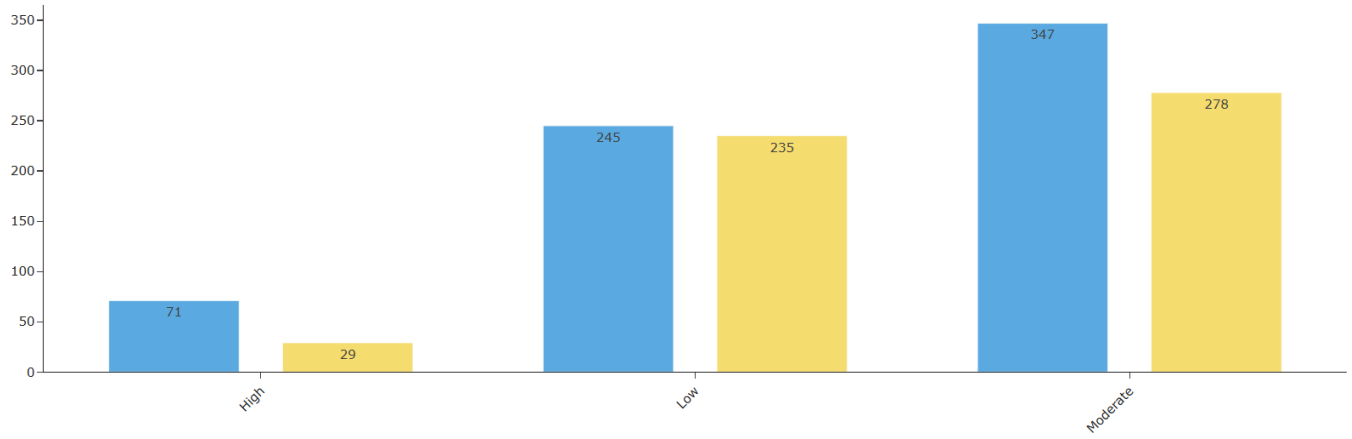
```
Education_IT = df.groupby(['Education Level', 'IT Student']).size()
fig = go.Figure(data=[
    go.Bar(name='College', x=Education_IT['College'].index, y=Education_IT['College'].values,
           text=Education_IT['College'].values, marker_color='#5DADE2'),
    go.Bar(name='School', x=Education_IT['School'].index, y=Education_IT['School'].values,
           text=Education_IT['School'].values, marker_color='#F7DC6F'),
    go.Bar(name='University', x=Education_IT['University'].index, y=Education_IT['University'].values,
           text=Education_IT['University'].values, marker_color='#F1948A')
])

fig.update_layout(barmode='group', xaxis_tickangle=-45, title='Distribution of IT Students by Education Level',
                  template='simple_white')

fig.update_traces(width=0.2)

fig.show()
```



- **Gender Distribution by Adaptivity Level**
  The code generates a grouped bar chart using Plotly to show the distribution of Gender across different Adaptivity Levels. It compares counts for boys and girls with labeled bars, distinct colors, and grouped formatting.

```
Gender_Adaptivity = df.groupby(['Gender', 'Adaptivity Level']).size()
fig = go.Figure(data=[
    go.Bar(name='Boy', x=Gender_Adaptivity['Boy'].index, y=Gender_Adaptivity['Boy'].values,
           text=Gender_Adaptivity['Boy'].values, marker_color='#5DADE2'),
    go.Bar(name='Girl', x=Gender_Adaptivity['Girl'].index, y=Gender_Adaptivity['Girl'].values,
           text=Gender_Adaptivity['Girl'].values, marker_color='#F7DC6F')
])

fig.update_layout(barmode='group', xaxis_tickangle=-45, title='Distribution of Gender by Adaptivity Level',
                  template='simple_white')

fig.update_traces(width=0.3)

fig.show()
```

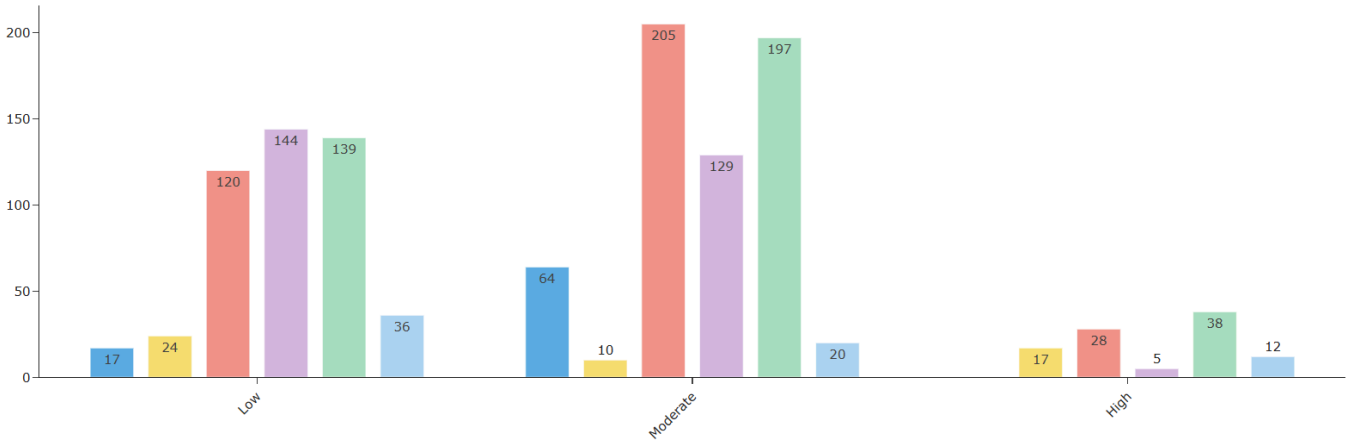
Distribution of Gender by Adaptivity Level



- **Age Distribution by Adaptivity Level**
  The code creates a grouped bar chart using Plotly to display the distribution of Age across different Adaptivity Levels. It includes age ranges as separate bars with unique colors, labeled values, and a grouped layout for comparison.

```
Age_Adaptivity = df.groupby(['Age', 'Adaptivity Level']).size()
fig = go.Figure(data=[
    go.Bar(name='1-5', x=Age_Adaptivity['1-5'].index, y=Age_Adaptivity['1-5'].values,
           text=Age_Adaptivity['1-5'].values, marker_color='#5DADE2'),
    go.Bar(name='6-10', x=Age_Adaptivity['6-10'].index, y=Age_Adaptivity['6-10'].values,
           text=Age_Adaptivity['6-10'].values, marker_color='#F7DC6F'),
    go.Bar(name='11-15', x=Age_Adaptivity['11-15'].index, y=Age_Adaptivity['11-15'].values,
           text=Age_Adaptivity['11-15'].values, marker_color='#F1948A'),
    go.Bar(name='16-20', x=Age_Adaptivity['16-20'].index, y=Age_Adaptivity['16-20'].values,
           text=Age_Adaptivity['16-20'].values, marker_color='#D2B4DE'),
    go.Bar(name='21-25', x=Age_Adaptivity['21-25'].index, y=Age_Adaptivity['21-25'].values,
           text=Age_Adaptivity['21-25'].values, marker_color='#A9DFBF'),
    go.Bar(name='26-30', x=Age_Adaptivity['26-30'].index, y=Age_Adaptivity['26-30'].values,
           text=Age_Adaptivity['26-30'].values, marker_color='#AED6F1')
])

fig.update_layout(barmode='group', xaxis_tickangle=-45, title='Distribution of Age by Adaptivity Level',
                  template='simple_white')

fig.update_traces(width=0.1)

fig.show()
```

Distribution of Age by Adaptivity Level



Activity 2: Pre-Processing

Pre-processing involves selecting and organizing relevant features from the dataset to prepare it for analysis or model training.

The code defines two lists — features and columns — containing the names of all selected attributes, including input and target variables. This step helps structure the data for consistent access and further processing.

Pre-Processing

```
[ ] features = ['Gender', 'Age', 'Education Level', 'Institution Type', 'IT Student', 'Location',
                'Load-shedding', 'Financial Condition', 'Internet Type', 'Network Type',
                'Class Duration', 'Self Lms', 'Device', 'Adaptivity Level']

    columns = ['Gender', 'Age', 'Education Level', 'Institution Type', 'IT Student', 'Location',
               'Load-shedding', 'Financial Condition', 'Internet Type', 'Network Type',
               'Class Duration', 'Self Lms', 'Device', 'Adaptivity Level']

    print("Features:")
    print(features)
    print("Columns:")
    print(columns)
```

```
Features:
['Gender', 'Age', 'Education Level', 'Institution Type', 'IT Student', 'Location', 'Load-shedding', 'Financial Condition', 'Internet Type', 'Network Type', 'Class Duration', 'Self Lms', 'Device', 'Adaptivity Level']
Columns:
['Gender', 'Age', 'Education Level', 'Institution Type', 'IT Student', 'Location', 'Load-shedding', 'Financial Condition', 'Internet Type', 'Network Type', 'Class Duration', 'Self Lms', 'Device', 'Adaptivity Level']
```

Activity 3: Encoding

Encoding is the process of converting categorical data into numerical form so that it can be used in machine learning models.

```
[ ] columns = ['Gender', 'Age', 'Education Level', 'Institution Type', 'IT Student', 'Location',
               'Load-shedding', 'Financial Condition', 'Internet Type', 'Network Type',
               'Class Duration', 'Self Lms', 'Device', 'Adaptivity Level']

    encoders = {}

    for column in columns:
        le = LabelEncoder()
        df[column] = le.fit_transform(df[column])
        encoders[column] = le

    joblib.dump(encoders, 'encoders_dict.pkl')
```

```
['encoders_dict.pkl']
```

Activity 4: Test-Train Split

This code performs the train-test split, a key step in preparing data for machine learning.

It separates the target variable (Adaptivity Level) into y and the features into X. Then, it splits the data into training and testing sets using a 70-30 ratio — 70% for training (X_train, y_train) and 30% for testing (X_test, y_test). The random_state ensures reproducibility of the split. The shapes of the resulting sets are printed to confirm the division.

Test-Train Split

```
[ ] y=df['Adaptivity Level']
    X= df.drop('Adaptivity Level', axis=1)
    X_train, X_test, y_train, y_test=train_test_split(X, y, test_size = 0.5, random_state = 116)
```

```
[ ] print(colored('Shape of X_train: ', 'blue'), X_train.shape, '\n\n')
    print(colored('Shape of X_test: ', 'red'), X_test.shape)
```

```
Shape of X_train:  (843, 13)


Shape of X_test:  (362, 13)
```

4: Model Building

Activity 1: Training and Testing the model in multiple algorithms

To build the model, we can train our data on different algorithms. For this project we are applying five supervised machine learning algorithms usually used for classification tasks. The best model is saved based on its performance.

The CONFIG class defines configuration settings for the machine learning model. It includes the number of cross-validation folds (FOLD) and the random state (RANDOM_STATE). It also provides a dictionary of classifiers (CLASSIFIERS) with their respective parameter settings. The FEATURES list contains the selected features for the model. Additionally, it specifies the best parameter settings for CatBoost (CB_BEST_PARAMS) and XGBoost (XGB_BEST_PARAMS) based on optimization using Optuna.

Define Pre-Processing and Modelling Function

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier

class CONFIG:
    FOLD = 5
    RANDOM_STATE = 116

    CLASSIFIERS = {
        "LogisticRegression": LogisticRegression(max_iter=1000, random_state=RANDOM_STATE),
        "KNearest": KNeighborsClassifier(),
        "RandomForestClassifier": RandomForestClassifier(random_state=RANDOM_STATE),
        "XGBClassifier": XGBClassifier(random_state=RANDOM_STATE),
        "CatBoostClassifier": CatBoostClassifier(random_state=RANDOM_STATE, logging_level='Silent')
    }

    FEATURES = [
        'Gender', 'Age', 'Education Level', 'Institution Type', 'IT Student', 'Location',
        'Load-shedding', 'Financial Condition', 'Internet Type', 'Network Type',
        'Class Duration', 'Self Lms', 'Device'
    ]

    CB_BEST_PARAMS = {
        'iterations': 815,
        'depth': 5,
        'learning_rate': 0.08616774389778385,
        'random_strength': 45,
        'bagging_temperature': 0.23946457882908667,
        'od_type': 'Iter',
        'od_wait': 18,
        'logging_level': 'Silent',
        'random_state': RANDOM_STATE
    }

    XGB_BEST_PARAMS = {
        'n_estimators': 634,
        'max_depth': 15,
        'reg_alpha': 0,
        'reg_lambda': 3,
        'min_child_weight': 0,
        'gamma': 0,
        'learning_rate': 0.2616305059064822,
        'colsample_bytree': 0.65,
        'random_state': RANDOM_STATE
    }
```

```
[ ]  # Scaler function
     def _scale(train_data, val_data, features):
         scaler = StandardScaler()

         scaled_train = scaler.fit_transform(train_data[features])
         scaled_val = scaler.transform(val_data[features])

         train = train_data.copy()
         val = val_data.copy()

         train[features] = scaled_train
         val[features] = scaled_val

         return train, val
```

```
def classifiers_modeling(classifiers, X_train, y_train_, X_test, y_test, features):
    accuracy_list = []
    classifiers_name = list(classifiers.keys())

    fold = KFold(n_splits=CONFIG.FOLD, random_state=CONFIG.RANDOM_STATE, shuffle=True)

    for idx, classifier in enumerate(classifiers.values()):
        accuracy = 0

        for fold_idx, (train_idx, val_idx) in enumerate(fold.split(X_train, y_train_)):
            x_train, x_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
            y_train, y_val = y_train_.iloc[train_idx], y_train_.iloc[val_idx]

            x_train, x_val = _scale(x_train, x_val, features)

            model = classifier.fit(x_train[features], y_train)
            val_preds = model.predict(x_val[features])

            accuracy += accuracy_score(y_val, val_preds) / CONFIG.FOLD

        accuracy_list.append(round(accuracy, 5))

        print(f"({idx+1}) {classifiers_name[idx]} cross validation (5 fold)")
        print("Mean Accuracy Score:", round(accuracy, 5))
        print("Mean Accuracy Score:", colored(round(accuracy, 5), 'green'))

    return accuracy_list
```

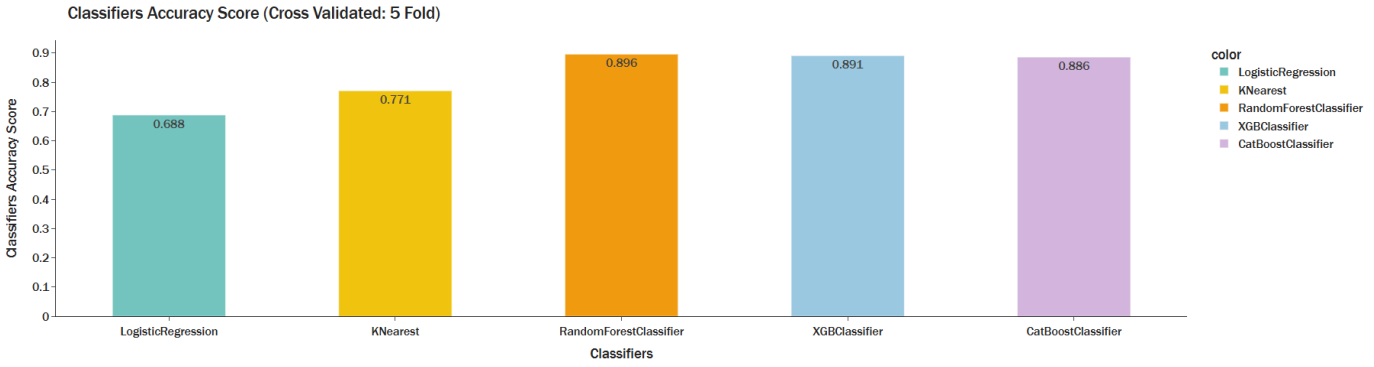
```
[ ] def p_bar_plot(x, y, width, x_axis_title, y_axis_title, title):
        fig = px.bar(
            x=x,
            y=y,
            color=x,
            text=y,
            color_discrete_sequence=color_scheme,
            template='simple_white'
        )

        fig.update_layout(
            xaxis_title=x_axis_title,
            yaxis_title=y_axis_title,
            title=title,
            font=dict(size=17, family="Franklin Gothic")
        )

        fig.update_traces(width=width)
        fig.show()
```

Model Comparison (part -1)

```
[ ] classifiers_name = list(CONFIG.CLASSIFIERS.keys())
    classifiers_accuracy = []
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

classifiers_accuracy = classifiers_modeling(CONFIG.CLASSIFIERS, X_train, y_train, X_test, y_test, CONFIG.FEATURES)
```

```
(1) LogisticRegression cross validation (5 fold)
Mean Accuracy Score: 0.68809
Mean Accuracy Score: 0.68809
(2) KNearest cross validation (5 fold)
Mean Accuracy Score: 0.77112
Mean Accuracy Score: 0.77112
(3) RandomForestClassifier cross validation (5 fold)
Mean Accuracy Score: 0.89568
Mean Accuracy Score: 0.89568
(4) XGBClassifier cross validation (5 fold)
Mean Accuracy Score: 0.89094
Mean Accuracy Score: 0.89094
(5) CatBoostClassifier cross validation (5 fold)
Mean Accuracy Score: 0.88619
Mean Accuracy Score: 0.88619
```



This code performs 5-fold cross-validation on multiple classifiers to evaluate their performance. The _scale function standardizes features using StandardScaler, ensuring fair comparison across models. For each classifier, the data is split into 5 folds, scaled, trained, and validated, and the average accuracy is computed and printed. Finally, it returns a list of mean accuracies for all classifiers.

Activity 1.1: Tuned XGBoost model

First xgboost Library is installed, and XGBClassifier is imported from it and initialised. The model is then tuned. We use Optuna to automatically tune hyperparameters for an

XGBClassifier to improve its performance. The xgb_objective function defines the search space for key parameters like n_estimators, max_depth, learning_rate, and regularization terms using trial.suggest_* methods. It performs K-Fold cross-validation to ensure dependable assessment by averaging accuracy across different train-validation splits. The goal is to maximize accuracy, and Optuna explores 120 different parameter combinations or until 600 seconds pass. This tuning process helps find the best hyperparameter set to avoid underfitting or overfitting as far as possible, resulting in a more accurate and robust model.

```
Times = 5
y_train_ = y_train
def xgb_objective(trial):

    param = {
            "n_estimators" : trial.suggest_int('n_estimators', 0, 1000),
            'max_depth':trial.suggest_int('max_depth', 2, 25),
            'reg_alpha':trial.suggest_int('reg_alpha', 0, 5),
            'reg_lambda':trial.suggest_int('reg_lambda', 0, 5),
            'min_child_weight':trial.suggest_int('min_child_weight', 0, 5),
            'gamma':trial.suggest_int('gamma', 0, 5),
            'learning_rate':trial.suggest_loguniform('learning_rate',0.005,0.5),
            'colsample_bytree':trial.suggest_discrete_uniform('colsample_bytree',0.1,1,0.01),
            'nthread' : -1
            }
    accuracy = 0
    xgb_model = XGBClassifier(**param)
    fold = KFold(n_splits = CONFIG.FOLD, random_state = CONFIG.RANDOM_STATE, shuffle=True)
    for fold_idx, (train_idx, val_idx) in enumerate(fold.split(X_train, y_train_)):
        x_train, x_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
        x_train, x_val = _scale(x_train, x_val, CONFIG.FEATURES)
        y_train, y_val = y_train_.iloc[train_idx], y_train_.iloc[val_idx]
        model = xgb_model.fit(x_train[CONFIG.FEATURES], y_train)
        val_preds = model.predict(x_val[CONFIG.FEATURES])

        accuracy += accuracy_score(y_val, val_preds) / CONFIG.FOLD
    return accuracy

xgb_study = optuna.create_study(direction="maximize")
xgb_study.optimize(xgb_objective, n_trials=120, timeout=600)
```

Activity 1.2: Tuned Catboost

First catboost Library is installed, and CatBoostClassifier is imported from it and initialised. The model is then tuned.

```python
def cb_objective(trial):
    params = {
        'iterations': trial.suggest_int('iterations', 50, 1000),
        'depth': trial.suggest_int('depth', 4, 10),
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.3),
        'random_strength': trial.suggest_int('random_strength', 0, 100),
        'bagging_temperature': trial.suggest_loguniform('bagging_temperature', 0.01, 100.00),
        'od_type': trial.suggest_categorical('od_type', ['IncToDec', 'Iter']),
        'od_wait': trial.suggest_int('od_wait', 10, 50),
        'logging_level': 'Silent',
        'random_state': CONFIG.RANDOM_STATE
    }

    accuracy = 0
    cb_model = CatBoostClassifier(**params)

    fold = KFold(n_splits=CONFIG.FOLD, random_state=CONFIG.RANDOM_STATE, shuffle=True)

    for fold_idx, (train_idx, val_idx) in enumerate(fold.split(X_train, y_train)):
        x_train_fold, x_val_fold = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_train_fold, y_val_fold = y_train.iloc[train_idx], y_train.iloc[val_idx]

        x_train_fold, x_val_fold = _scale(x_train_fold, x_val_fold, CONFIG.FEATURES)

        cb_model.fit(x_train_fold[CONFIG.FEATURES], y_train_fold)
        val_preds = cb_model.predict(x_val_fold[CONFIG.FEATURES])
        accuracy += accuracy_score(y_val_fold, val_preds) / CONFIG.FOLD

    return accuracy

cb_study = optuna.create_study(direction="maximize")
cb_study.optimize(cb_objective, n_trials=50, timeout=600)
```

We use Optuna to tune hyperparameters for a CatBoostClassifier by evaluating model performance through K-Fold cross-validation. It defines a search space for parameters like iterations, depth, learning_rate, and more. For each trial, the model is trained and validated across multiple folds, and the average accuracy is returned. The goal is to find the best combination of parameters that maximizes accuracy, with up to 50 trials or 600 seconds of search time.

Activity 1.3: Random Forest Classifier

We train a RandomForestClassifier imported from sklearn.ensemble on training data, evaluate it on test data using accuracy and classification report, and visualize the performance with a confusion matrix plot. The results (predictions, accuracy, and report) are stored in a dictionary.

Activity 1.4: KNN

We train a KNeighborsClassifier imported from sklearn.neighbors using specified hyperparameters, evaluate its performance on test data using accuracy and classification report, and visualize the results with a confusion matrix plot. The evaluation outputs are stored in results dictionary.

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

CONFIG.KNN_BEST_PARAMS = {
    'n_neighbors': 5,
    'weights': 'uniform',
    'algorithm': 'auto'
}


from sklearn.neighbors import KNeighborsClassifier

model4 = KNeighborsClassifier(**CONFIG.KNN_BEST_PARAMS)
model4.fit(X_train[CONFIG.FEATURES], y_train)

predictions4 = model4.predict(X_test[CONFIG.FEATURES])

accuracy = accuracy_score(y_test, predictions4)
class_report = classification_report(y_test, predictions4)

print("Accuracy:", accuracy)
print("Classification Report:")
print(class_report)

results = {
    'predictions': predictions4,
    'accuracy': accuracy,
    'classification_report': class_report
}

cm = confusion_matrix(y_test, predictions4)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model4.classes_)

disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```

Activity 1.5: Logistic Regression

We train a LogisticRegression model imported from sklearn.linear_model with specified hyperparameters, evaluate its performance on test data using accuracy and a classification report, and visualize the results using a confusion matrix. The predictions and metrics are stored in results dictionary.

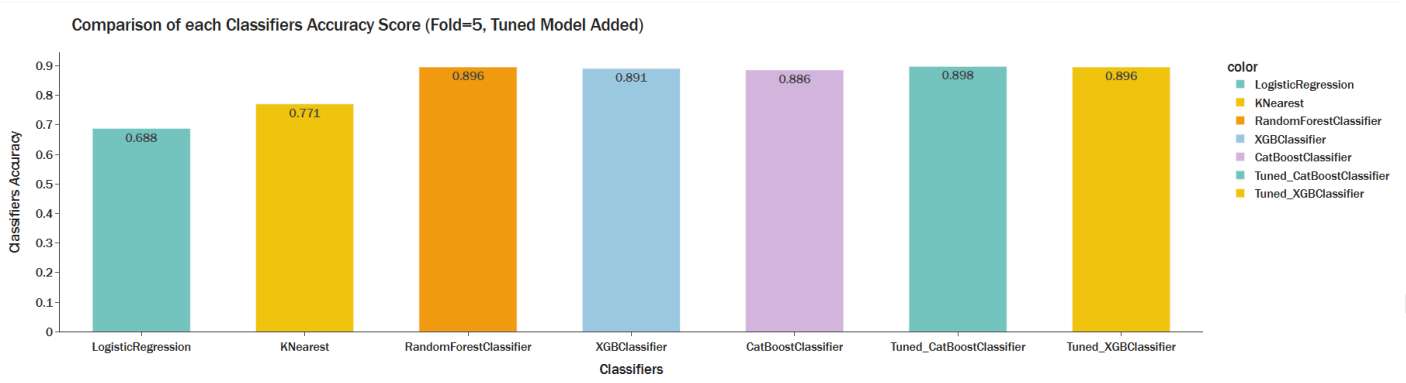
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

CONFIG.LOGREG_BEST_PARAMS = {
    'penalty': 'l2',
    'C': 1.0,
    'solver': 'lbfgs',
    'max_iter': 1000,
    'random_state': CONFIG.RANDOM_STATE
}

model5 = LogisticRegression(**CONFIG.LOGREG_BEST_PARAMS)
model5.fit(X_train[CONFIG.FEATURES], y_train)

predictions5 = model5.predict(X_test[CONFIG.FEATURES])

accuracy = accuracy_score(y_test, predictions5)
class_report = classification_report(y_test, predictions5)

print("Accuracy:", accuracy)
print("Classification Report:")
print(class_report)

results = {
    'predictions': predictions5,
    'accuracy': accuracy,
    'classification_report': class_report
}

cm = confusion_matrix(y_test, predictions5)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model5.classes_)

disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```

```
[ ] tuned_classifiers = {
        "CatBoostClassifier": CatBoostClassifier(**CONFIG.CB_BEST_PARAMS),
        "XGBClassifier": XGBClassifier(**CONFIG.XGB_BEST_PARAMS)
    }
```

```
[ ] classifiers_name.append('Tuned_CatBoostClassifier')
    classifiers_name.append('Tuned_XGBClassifier')
    tuned_accuracy = classifiers_modeling(tuned_classifiers, X_train, y_train, X_test, y_test, CONFIG.FEATURES)

    for tuned_accuracy_score in tuned_accuracy:
        classifiers_accuracy.append(tuned_accuracy_score)
```

```
(1) CatBoostClassifier cross validation (5 fold)
Mean Accuracy Score: 0.89806
Mean Accuracy Score: 0.89806
(2) XGBClassifier cross validation (5 fold)
Mean Accuracy Score: 0.89567
Mean Accuracy Score: 0.89567
```

The p_bar_plot function is called to visualize the accuracy scores of the classifiers. The x-axis represents the names of the classifiers, while the y-axis represents the accuracy scores. The bar plot shows the accuracy scores for each classifier, with the height of each bar indicating the corresponding score.



We test all the algorithms with the help of predict() function.

5. Performance Testing and Hyperparameter Tuning

Activity 1 and 2: Comparing all the models & Visualisation of Confusion Matrices

Multiple evaluation metrics refer to assessing a model's performance using various performance measures rather than relying on a single score. This approach provides a more comprehensive understanding of the model's strengths and weaknesses.

In this classification task, we are using the following evaluation metrics:

- **Accuracy**: Measures the overall correctness of the model by calculating the ratio of correctly predicted instances to the total instances.
- **Precision**: Indicates the proportion of correctly predicted positive observations to the total predicted positives.
- **Recall**: Reflects the ability of the model to find all relevant cases (true positives).
- **F1-score**: The harmonic mean of precision and recall, useful when classes are imbalanced.
- Support: The number of actual occurrences of each class in the test set.

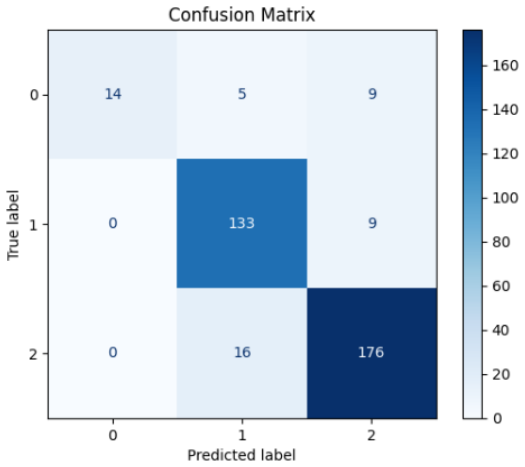
These metrics together help us analyze how well the model performs across different classes, highlighting not just how many predictions are right, but also how and where the model succeeds or struggles.

In our case we will be comparing the top 3 models (Tuned XGBoost, Tuned CatBoost, and RandomForest) and choosing the best of all

a) XGBoost

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
model1 = XGBClassifier(**CONFIG.XGB_BEST_PARAMS)
model1.fit(X_train[CONFIG.FEATURES], y_train)
predictions1 = model1.predict(X_test[CONFIG.FEATURES])
accuracy = accuracy_score(y_test, predictions1)
class_report = classification_report(y_test, predictions1)
print("Accuracy:", accuracy)
print("Classification Report:")
print(class_report)
results = {
    'predictions': predictions1,
    'accuracy': accuracy,
    'classification_report': class_report
}
cm = confusion_matrix(y_test, predictions1)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model1.classes_)
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.8922651933701657
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67        28
           1       0.86      0.94      0.90       142
           2       0.91      0.92      0.91       192

    accuracy                           0.89       362
   macro avg       0.92      0.78      0.83       362
weighted avg       0.90      0.89      0.89       362
```



The code evaluates the performance of the trained XGBoost model using accuracy, classification report, and a confusion matrix. The results are saved in a dictionary for further use and the confusion matrix is visualized using ConfusionMatrixDisplay.

The evaluation results are as follows:

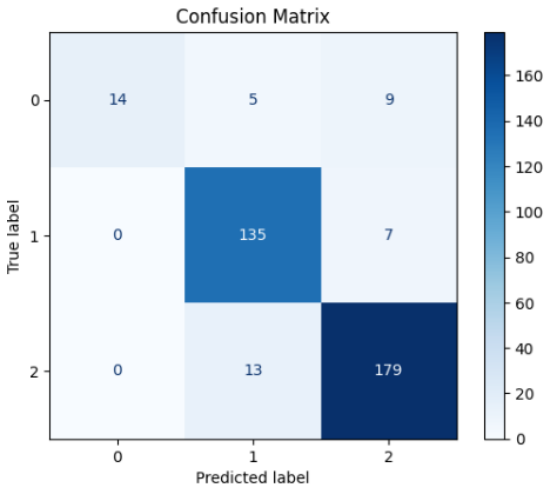
1. **Accuracy**: The model achieves an accuracy of **0.8922**, meaning approximately 89.22% of the predictions on the test data are correct.
2. **Classification Report**:
   - **Class 0**: Precision = 1.00, Recall = 0.50
     The model identifies class 0 with perfect precision, but only 50% of actual class 0 instances are correctly predicted, indicating low recall.
   - **Class 1**: Precision = 0.86, Recall = 0.94
     The model performs well on class 1 with high recall, successfully identifying most of its instances.
   - **Class 2**: Precision = 0.91, Recall = 0.92
     The model shows consistent performance for class 2 with both precision and recall above 90%.
3. **Averages**:
   - **Macro Average F1-Score**: 0.83
     This average treats all classes equally and shows moderate performance overall, impacted by the low recall in class 0.
   - **Weighted Average F1-Score**: 0.89
     This average considers the number of instances in each class and indicates strong overall model performance.

In summary, the model demonstrates strong performance for classes 1 and 2. However, the low recall for class 0 suggests it struggles to correctly identify all instances of that class.

b) Catboost

```
from catboost import CatBoostClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
model2 = CatBoostClassifier(**CONFIG.CB_BEST_PARAMS)
model2.fit(X_train[CONFIG.FEATURES], y_train)
predictions = model2.predict(X_test[CONFIG.FEATURES])
accuracy = accuracy_score(y_test, predictions)
class_report = classification_report(y_test, predictions)
print("Accuracy:", accuracy)
print("Classification Report:")
print(class_report)
results = {
    'predictions': predictions,
    'accuracy': accuracy,
    'classification_report': class_report
}
cm = confusion_matrix(y_test, predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model2.classes_)
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.9060773480662984
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67        28
           1       0.88      0.95      0.92       142
           2       0.92      0.93      0.93       192

    accuracy                           0.91       362
   macro avg       0.93      0.79      0.84       362
weighted avg       0.91      0.91      0.90       362
```



The code above performs model evaluation using a CatBoost classifier. It calculates several classification metrics after fitting the model with the best-tuned parameters and making predictions on the test set.

Evaluation Results:

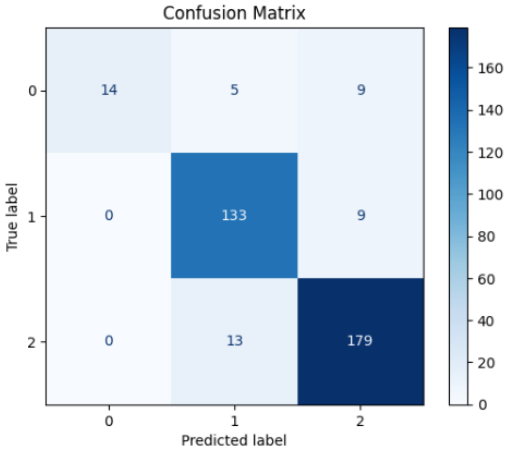
1. **Accuracy**: 0.9061 – This indicates that the model correctly predicted approximately 90.61% of the test samples.
2. **Classification Report**:
   - **Class 0**: The model achieved perfect precision (1.00), meaning all instances predicted as class 0 were correct. However, the recall is 0.50, which shows that only 50% of actual class 0 instances were captured.
   - **Class 1**: The model performs well with a precision of 0.88 and recall of 0.95, suggesting it effectively identifies class 1 instances.
   - **Class 2**: It also performs strongly for class 2, with both precision (0.92) and recall (0.93) being high.
3. **Macro Average**:
   - F1-score: 0.84 – This gives equal weight to each class and highlights slightly lower performance for class 0.
   - Recall: 0.79 – Reflects a relatively lower ability to capture all actual instances, mainly due to the low recall for class 0.
4. **Weighted Average**:
   - F1-score: 0.90 – Takes class support into account and shows strong overall model performance.

In summary, the CatBoost model performs well for classes 1 and 2, showing both high precision and recall. The main area for improvement is class 0, where the model struggles to recall all relevant instances.

c) RandomForest Classifier

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
model3 = RandomForestClassifier(random_state=42)
model3.fit(X_train[CONFIG.FEATURES], y_train)
predictions3 = model3.predict(X_test[CONFIG.FEATURES])
accuracy = accuracy_score(y_test, predictions3)
class_report = classification_report(y_test, predictions3)
print("Accuracy:", accuracy)
print("Classification Report:")
print(class_report)
results = {
    'predictions': predictions3,
    'accuracy': accuracy,
    'classification_report': class_report
}
cm = confusion_matrix(y_test, predictions3)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model3.classes_)
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.9005524861878453
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67        28
           1       0.88      0.94      0.91       142
           2       0.91      0.93      0.92       192

    accuracy                           0.90       362
   macro avg       0.93      0.79      0.83       362
weighted avg       0.90      0.90      0.90       362
```



The code above evaluates the performance of a Random Forest classifier. The model is trained using the training data, and predictions are made on the test set. Several evaluation metrics are computed to assess how well the model performs.

Evaluation Results:

1. **Accuracy**: 0.9005 – This means the model correctly predicted around 90.05% of the test instances.
2. **Classification Report**:
    - **Class 0**: The model shows perfect precision (1.00), meaning all predicted class 0 instances were accurate. However, the recall is 0.50, which indicates it only identified 50% of all actual class 0 instances.
    - **Class 1**: With a precision of 0.88 and recall of 0.94, the model performs well for this class.
    - **Class 2**: It also performs strongly here, achieving a precision of 0.91 and recall of 0.93.
3. **Macro Average**:
    - F1-score: 0.83 – Reflects the average F1-score across all classes, giving equal importance to each.
    - Recall: 0.79 – Slightly affected by the lower recall for class 0.
4. **Weighted Average**:
    - F1-score: 0.90 – Takes into account the number of samples in each class and shows the model performs consistently well overall.

In conclusion, the Random Forest model is effective in classifying instances of class 1 and class 2. However, it has relatively lower performance on class 0, primarily due to its lower recall.

Overall Analysis:
Of the three models compared — XGBoost, CatBoost, and Random Forest — CatBoost had the best performance, with the highest accuracy of 0.9060 and good precision and recall for all classes. Random Forest trailed closely with 0.9005 accuracy and also had regular performances, particularly on classes 1 and 2. XGBoost, though still performing well, had poorer performance with 0.8922 accuracy. Generally, CatBoost is the most consistent model in this comparison, followed closely by Random Forest.

Activity 3: Feature Importance Analysis and Visualization

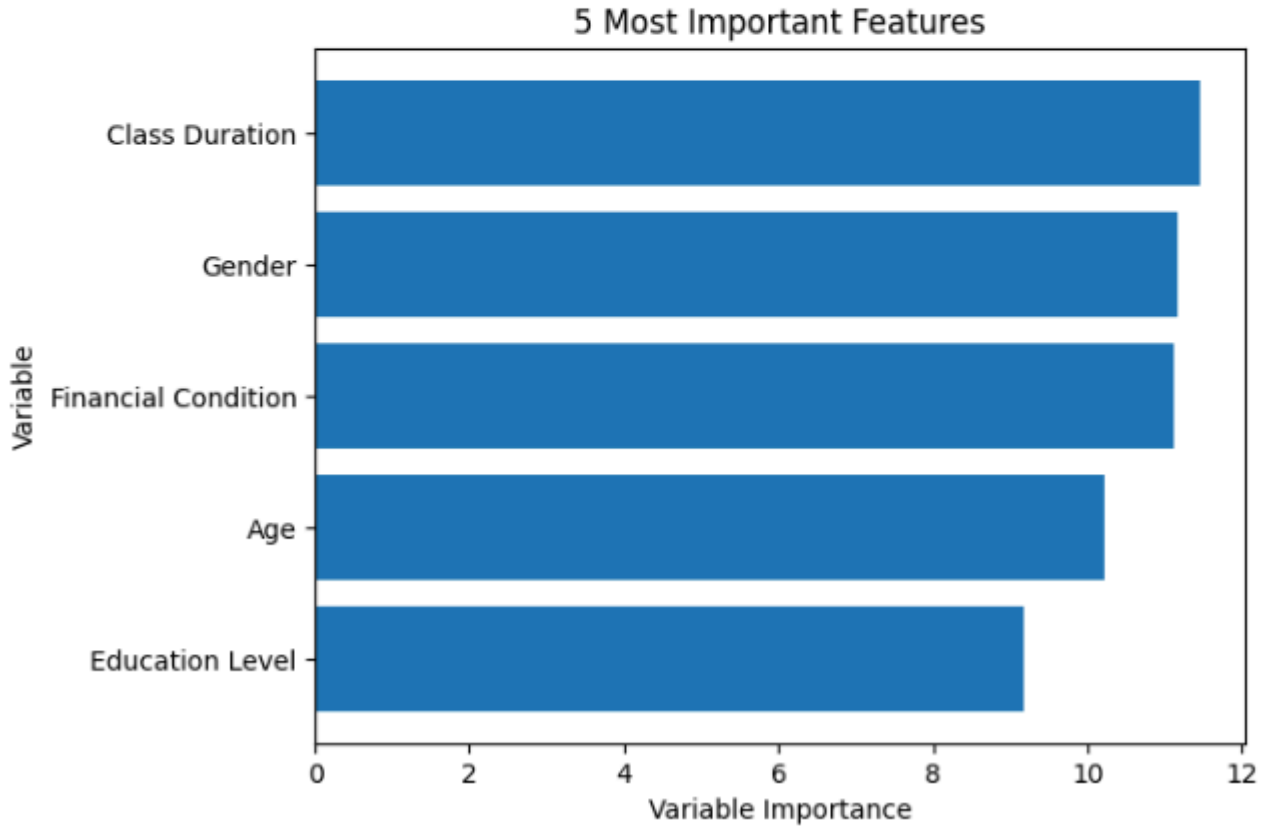
```
feature_importances = model2.feature_importances_
feature_importances_df = pd.DataFrame({
    'Variable': X_train.columns,
    'Variable Importance': feature_importances
})
feature_importances_df = feature_importances_df.sort_values('Variable Importance', ascending=False)
print(feature_importances_df)
top_features = feature_importances_df.nlargest(5, 'Variable Importance')
plt.barh(top_features['Variable'], top_features['Variable Importance'])
plt.title("5 Most Important Features")
plt.xlabel("Variable Importance")
plt.ylabel("Variable")
plt.gca().invert_yaxis()
plt.show()
```

This code segment is used to evaluate and visualize the feature importance derived from the trained CatBoost model. Feature importance refers to the contribution each input variable makes to the model's prediction, and analyzing it provides insight into which features have the most significant influence on the output.

The process begins by extracting the importance scores using the feature_importances_ attribute of the model. These scores are then organized into a DataFrame along with their corresponding feature names. The DataFrame is sorted in descending order to rank the features based on their influence. To focus on the most relevant inputs, the top five features with the highest importance values are selected. A horizontal bar chart is then plotted to visually present these top five features, allowing for easier interpretation. The y-axis is inverted so that the most important feature appears at the top of the chart.

This analysis aids in model interpretability and can also guide feature selection by highlighting the variables that play the most crucial role in predicting student adaptability.

	Variable	Variable Importance
10	Class Duration	11.465912
0	Gender	11.158527
7	Financial Condition	11.113674
1	Age	10.216659
2	Education Level	9.182005
9	Network Type	8.813280
8	Internet Type	7.511637
3	Institution Type	7.230960
12	Device	5.486979
5	Location	4.915748
11	Self Lms	4.706302
4	IT Student	4.469861
6	Load-shedding	3.728456



6. Model Deployment

Activity 1: Save and Load the Best Model

The provided code snippet demonstrates the usage of the joblib library for saving and loading a CatBoost model. After training the model (in this case, model2), it is saved to a file named catboost_model.pkl using the joblib.dump() function.

Later, when predictions need to be made on new data, the saved model is loaded from the file using the joblib.load() function. Once loaded, the model can be used to make predictions on the test dataset X_test using the model.predict() method. The feature columns to be used for prediction are specified by CONFIG.FEATURES.

```
[ ]  import joblib
     joblib.dump(model2, 'catboost_model.pkl')
```

```
[ ]  import joblib

     model = joblib.load('catboost_model.pkl')

     predictions = model.predict(X_test[CONFIG.FEATURES])
```

This process ensures that the best performing model can be reused later without retraining, making it suitable for deployment or future analysis.

Activity 2: Integrate with Web Framework

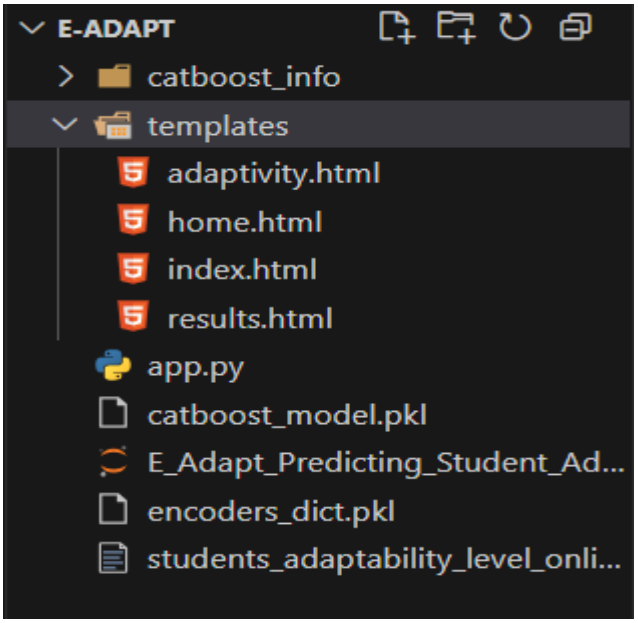
In this section, we will build a web application that interacts with the trained machine learning model. Users can input the required features through a web interface. These inputs are passed to the saved model, and the prediction is displayed on the same page.

This section includes the following tasks:

- Building HTML pages for user input and output
- Writing a server-side script using Flask
- Running the web application

Activity 2.1: Building HTML Pages

In the E-Adapt web application, four main HTML files are created and stored inside a special folder called templates/. This folder is a core part of any Flask web framework setup. Flask automatically looks inside this folder to render HTML pages in response to web requests. The templates use Jinja2 syntax, allowing the backend to pass dynamic data (like model predictions) into the frontend pages.



home.html – Project Landing Page
The home.html file serves as the homepage for the E-Adapt application. It introduces the purpose of the platform: helping users assess and improve their adaptability in online learning environments. Designed with responsive UI using Tailwind CSS, it includes engaging animations, floating visual elements, and call-to-action buttons that guide users to start their assessment or explore further features. This page is mapped to the root route (/) in the Flask app.

index.html – User Input Form
The index.html file is the main input form where users enter their personal, educational, and technical details required for the prediction. It is neatly divided into sections such as Personal Information, Educational Background, and Technology Infrastructure. Once the user fills out the form and submits it, the data is sent to the backend via the POST method for model prediction. The input form is designed with dropdowns and selection

menus, ensuring a user-friendly interface. This page is served when users access the /predict route with a GET request.

results.html – Prediction and Recommendations Page
The results.html page is where users are shown their predicted adaptability level, based on the model's output. It dynamically displays the result using templating syntax (e.g., {{ prediction }}) and includes a colorful progress bar to visualize the adaptability level (Low, Moderate, or High). Below the result, tailored recommendation cards appear, offering practical advice depending on the user's score. This page is returned when the form data from index.html is submitted and the model generates a prediction. It uses advanced animations and interactivity, creating a visually engaging experience.

adaptivity.html – Tips to Improve Adaptability
The adaptivity.html file provides users with a rich set of strategies to improve their online learning adaptability. It features multiple cards, each focusing on a different topic such as developing a growth mindset, staying flexible with scheduling, improving tech readiness, and strengthening communication skills. Each card includes an icon, description, and
actionable tips, supported by engaging illustrations. The page layout is responsive and visually enhanced with hover effects and animated transitions. It is accessible via the /adapt route and is particularly useful for users who wish to increase their adaptability level based on the suggestions in results.html.

templates/ Folder Structure
All the above HTML files are placed inside the templates/ directory, which is a convention in Flask applications. Flask uses this directory to locate HTML templates that are rendered using render_template() in the backend Python script. This structure ensures a clean separation of logic (Python code) and presentation (HTML/CSS), allowing efficient web development.

Activity 2.2: Build Python code

In this section, we build the core of our web application using Flask, a lightweight Python web framework. The app.py file handles routing, form input processing, and prediction using the pre-trained machine learning model.

1. Import the Necessary Libraries

The code begins by importing essential libraries:

```
from flask import Flask, render_template, request
import pandas as pd
import joblib
```

Flask: for building the web server and handling routing.

- render_template: to render HTML templates from the templates/ folder.
- request: to collect form input data from the user.
- pandas: for handling and manipulating input data in tabular form.
- joblib: for loading the trained ML model and encoders.

2. Load the Trained Model and Encoders
The machine learning model and corresponding label encoders are loaded using joblib.load():

```
model = joblib.load('catboost_model.pkl')
encoders = joblib.load('encoders_dict.pkl')
```

- catboost_model.pkl: contains the trained CatBoost model.
- encoders_dict.pkl: contains a dictionary of LabelEncoder objects used during training to preprocess categorical inputs.

This avoids the need to retrain or recreate the model and preprocessing pipeline.

3. Create a Flask App Instance

```
app = Flask(__name__)
```

This line initializes the Flask application. The __name__ variable ensures Flask knows where to look for templates and static files.

4. Define Field Mapping (Optional)

```
field_map = {
    'age': 'Age',
    'gender': 'Gender',
    ...
}
```

This dictionary maps frontend field names to the expected names used by the ML model. It's helpful when HTML form input names differ from those used in your training dataset.

5. Define Web Application Routes
Home Page Route

```
@app.route('/')
def home():
    return render_template('home.html')
```

Displays the project landing page (home.html).

Prediction Form Route

```
@app.route('/predict')
def index():
    return render_template('index.html')
```

Displays the prediction form (index.html), where users input their data.

Adaptivity Guide Page

```
@app.route('/adapt')
def adapt():
    return render_template('adaptivity.html')
```

Displays tips to improve learning adaptability (adaptivity.html).

6. Handle Form Submission and Make Predictions

```
@app.route('/predict', methods=["POST"])
def predict():
```

This route handles form submissions. When a user submits the form:

a. Collect Input Data

```
input_data = {}
for key, value in request.form.items():
    mapped_key = field_map.get(key, key)
    input_data[mapped_key] = value
```

The submitted form values are captured and mapped to their respective field names using the field map.

b. Convert to DataFrame

```
input_df = pd.DataFrame([input_data])
```

This creates a DataFrame row from the input data, matching the expected format of the model.

c. Encode Categorical Features

```
for col in input_df.columns:
    if col in encoders:
        le = encoders[col]
        if input_df[col].values[0] not in le.classes_:
            return f"Invalid input for '{col}': '{input_df[col].values[0]}'"
        input_df[col] = le.transform(input_df[col])
```

Each categorical feature is encoded using its corresponding LabelEncoder. It also handles invalid user input by returning an error message.

d. Drop Target Column if Present

```
if 'Adaptivity Level' in input_df.columns:
    input_df.drop(columns=['Adaptivity Level'], inplace=True)
```

This ensures the target column isn't passed into the model during prediction.

e. Make Prediction and Interpret Result

```
prediction = model.predict(input_df)[0]
adaptivity_level = 'High' if prediction == 0 else ('Moderate' if prediction == 2 else 'Low')
```

The prediction is mapped from a numerical class to a readable label for display.

f. Render the Results

```
return render_template("results.html", prediction=adaptivity_level)
```

The result is passed to the results.html template for display.

7. Main Function to Run the App

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

This ensures the Flask development server runs only when the script is executed directly (not imported). The debug=True enables live reloading and better error messages.

Activity 2.3: Run The Web Application

To run our E-Adapt web app, we just need to open the terminal and run the app.py file. Once we do that, a message appears in the terminal saying something like:
Running on http://127.0.0.1:5000
The user can then copy this URL and paste it into a web browser. This action opens the web application in the browser window.

Home Page (home.html)
When the user accesses the base URL (http://127.0.0.1:5000), they are directed to the home page of the application, which is rendered from the home.html file. This page introduces the E-Adapt platform and its purpose — to help users assess and improve their adaptability in online education.

The homepage features a modern, responsive design with animated backgrounds and a top navigation bar. The navigation bar includes links to the following sections:

- Home
- Predict My Adaptivity
- Increase My Adaptivity

SMARTBRIDGE
Let's Bridge the Gap
a Veranda Enterprise

Smart Internz

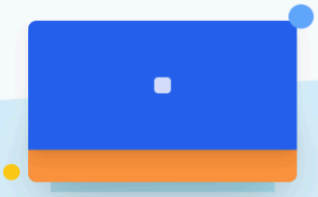
E-Adapt

Home | Predict My Adaptability | Increase My Adaptability

Navigate Your Online Learning Journey

Discover your adaptability to online learning and get personalized strategies to succeed in virtual classrooms.

Predict My Adaptability | Learn More



How E-Adapt Helps You Succeed

Personalized Assessment

Our algorithm analyzes your learning style, preferences, and current skills to provide insights tailored specifically to you.

Tailored Strategies

Get customized recommendations to improve your online learning experience and maximize your potential for success.

Progress Tracking

Monitor your improvement over time and celebrate your learning milestones.

Ready to Discover Your Online Learning Potential?

Take our quick assessment today and get personalized insights to help you excel in your virtual classroom journey.

Start My Assessment

E-Adapt

Your partner in building resilience and adaptability for effective online learning.

Quick Links

Home
Assess Your Adaptability
Grow Your Adaptability

Contact Us

123@e-adapt.com
+91 99999 99999

Prediction Page (index.html)
Upon clicking the “Predict My Adaptivity” link in the navigation bar, the user is redirected to the form page, which is rendered using index.html.

This page presents a form that collects various user details relevant to online learning adaptability. The fields in the form include:

- Gender
- Age
- Class Duration
- Financial Condition
- Institution Type
- Network Type
- Education Level
- Location
- Internet Type
- IT Student
- Device
- Use of Self LMS
- Load Shedding

All form elements are dropdowns for easy and consistent data entry. Once the user completes the form and clicks the “Predict” button, the input is sent to the Flask backend for processing.

SMARTBRIDGE
Let's Bridge the Gap
a Veranda Enterprise

Smart Internz

E-Adapt

Home | Predict My Adaptability | Increase My Adaptability

Discover Your Learning Adaptability

Answer a few quick questions about your learning environment and preferences to get personalized insights about your online learning adaptability.

Personal Information

Gender: Girl

Age Group: 6-10 years

Financial Condition: Mid

Technology & Infrastructure

Primary Device: Mobile

Internet Type: Wifi

Network Type: 4G

Power Outage Frequency: Low

Educational Background

Education Level: School

Institution Type: Non-Government

IT Student: No

Class Duration (hours): 1-3 hours

Learning Environment

Suitable Study Location: Yes

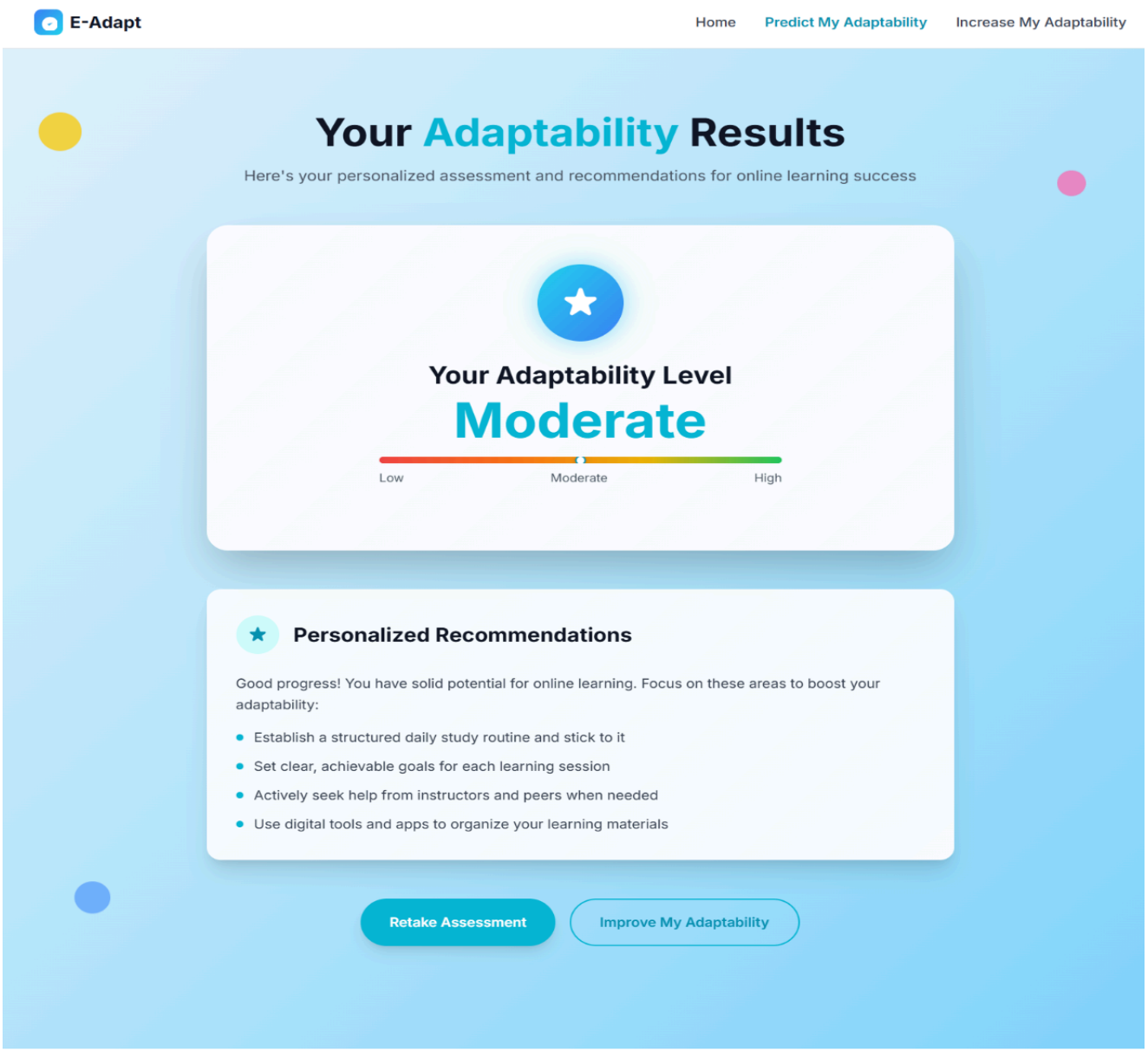
Self-Learning Management: No

Predict My Adaptability Level

Results Page (results.html)
After submitting the form, the backend model (a pre-trained CatBoost classifier) processes the input data and returns a predicted Adaptivity Level.

The result is displayed on the results.html page. This page shows:

- The predicted adaptivity level: High, Moderate, or Low.
- A visual progress bar that represents the score.
- Personalized recommendations based on the user's result.
- Action buttons to either retake the assessment or explore improvement strategies.



Universal Tips for Online Learning Success

Time Management
Create a consistent schedule and break large tasks into manageable chunks.

Stay Focused
Minimize distractions and use techniques like the Pomodoro method.

Stay Connected
Engage with classmates and instructors regularly through forums and video calls.

E-Adapt
Helping students navigate the waters of online learning with confidence and success.

Quick Links
Home
Predict My Adaptability
Increase My Adaptability

Resources
Learning Tips
Success Stories
Blog

Contact
info@e-adapt.com
(205) 123-4567

Adaptivity Strategy Page (adaptivity.html)
If the user chooses to improve their adaptability, they are redirected to the adaptivity.html page. This page contains a collection of visually appealing cards, each offering advice and actionable strategies for improving adaptability in online learning.

Topics covered include:

- Developing a growth mindset
- Managing time and schedules
- Handling change and uncertainty
- Enhancing technical readiness
- Improving communication skills

Each card is supported with icons, images, and animations to enhance user engagement.

SMARTBRIDGE
Let's Bridge the Gap
a Veranda Enterprise

Smart Internz

E-Adapt

Home | Predict My Adaptability | Increase Adaptability

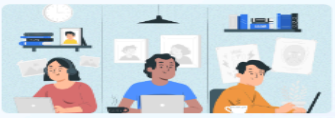
Your Personalized Adaptability Strategies

Here are actionable insights to enhance your learning agility and resilience.

Develop a Growth Mindset

- Embrace challenges as learning opportunities instead of obstacles.
- Replace "I can't do this" with "I can't do this yet."
- Celebrate small improvements to stay motivated.

Stay Flexible with Your Schedule



- Create a weekly plan but be open to adjustments.
- If a class is postponed or a topic is tough, reschedule with intention — not guilt.
- Use digital planners that let you drag and drop tasks (like Notion or Google Calendar).

Be Comfortable with Change

- Online platforms, tools, or even formats might change — adapt without resistance.
- Learn to explore new interfaces (like Zoom updates or a different LMS) quickly and calmly.
- Practice using new tools before class (e.g., breakout rooms, Padlet, Miro).

Upgrade Your Tech Readiness

- Keep your devices updated and learn basic troubleshooting.
- Back up your notes and files regularly using cloud storage (Google Drive, Dropbox).
- Familiarize yourself with multiple learning tools so you're not dependent on just one.

Strengthen Communication Skills

- Be proactive in reaching out to peers or teachers when confused.
- Use proper email/online etiquette to ask questions or request help.
- Participate in forums or chats to stay connected with the learning community.

Manage Stress and Emotions

- Use mindfulness techniques like deep breathing or meditation to handle uncertainty.
- Keep a journal to reflect on what's working and what's overwhelming.
- Don't aim for perfection — aim for progress and balance.

Adaptive Habits & Self-Assessment

- Each week, reflect: "What did I adapt to this week? How did I handle it?"
- Rate yourself on flexibility, time management, and emotional control.
- Set mini goals like: "This week, I'll try a new study technique," or "I'll reach out in the discussion board."
- Switch between learning styles: videos, podcasts, reading, or hands-on.
- Review material in different formats — it makes you flexible and improves retention.
- Test yourself under different conditions (different times, locations, or devices).

Cultivate Curiosity & Lifelong Learning

- Actively seek out new information and skills, even outside your immediate curriculum.
- Develop a habit of asking "why" and "how" to deepen understanding.
- See every new piece of software or concept as an opportunity to expand your knowledge base.
- Stay updated with trends in online learning and educational technology.

Practice Mindfulness & Focus

- Engage in short meditation or breathing exercises before study sessions.
- Minimize digital distractions by closing unnecessary tabs and muting notifications.
- Develop the ability to stay present and focused on the task at hand, even during complex topics.

Re-Assess My Adaptability | Explore Growth Resources

CONCLUSION:

In this e-Adapt project, we successfully built a machine learning model to predict student adaptability in online education landscapes. After evaluating multiple models including Logistic Regression, K-Nearest Neighbors, Random Forest, and XGBoost, the Tuned CatBoost Classifier emerged as the best algorithm, achieving the highest accuracy (90.6%) and stronger F1-scores across the adaptability classes.

The model identified students with varying levels of adaptability—low, moderate, and high—based on features such as academic performance, device usage, network quality, motivation, and learning behavior. By integrating CatBoost, the system handled categorical variables efficiently and minimized the need for heavy preprocessing.

This predictive system can be used by educators and institutions to efficiently identify students who might struggle in online classes, leading to targeted support and interventions. E-Adapt contributes to enhancing the overall success rate and learning experience in online education platforms.

Future improvements could include integrating explainable AI techniques, expanding the dataset with more diverse student populations, and deploying the model through an interactive dashboard for educators to use in real-time.

7. Video:

https://drive.google.com/file/d/1fG1gV14CoLP8-DNNN71smHRz6nUoQwkY/view?usp=drive_link