



**e-Adapt:**

# **Predicting Student Adaptivity in Online Classes**

Project Hand-Out  
-Kamya Paliwal

# **e-Adapt: Predicting Student Adaptivity in Online Classes**

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 very different from one another. Without having to engage with peers or professors in person, they demand students to successfully manage their time, 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:**

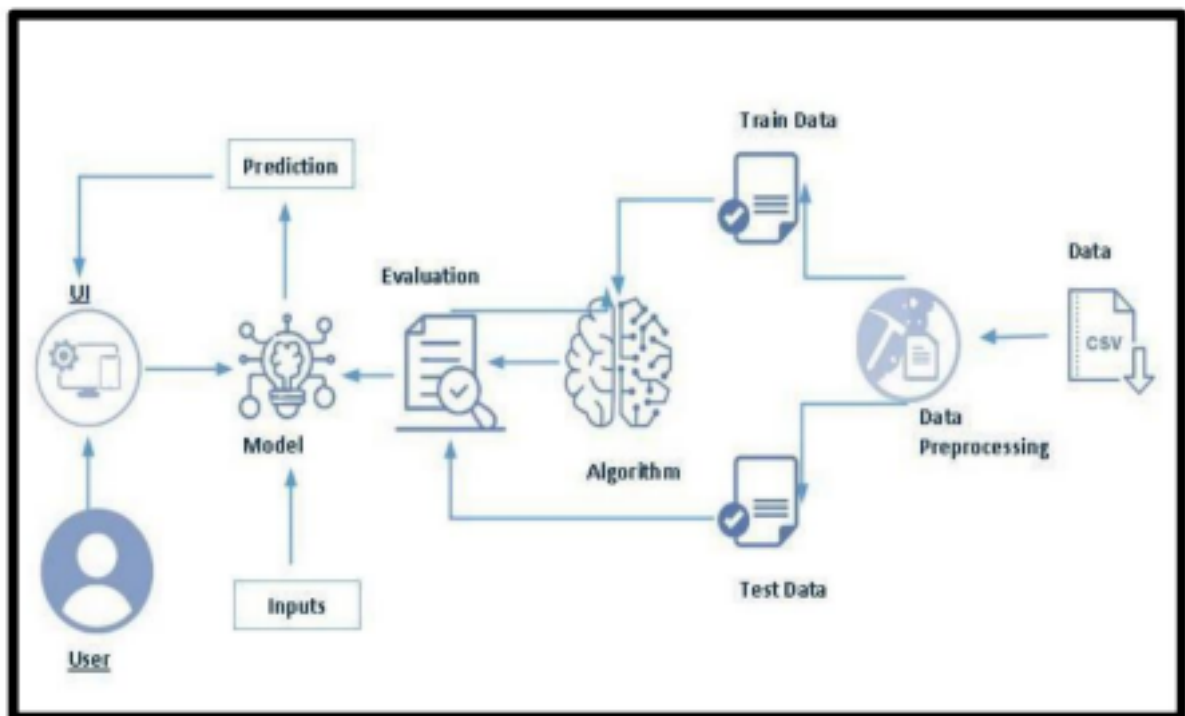
**The aim of e-Adapt is to predict student adaptability in online classrooms using machine learning, giving insights into students' capacities to flourish in digital learning environments and enabling focused interventions to increase student success and online course design.**

## **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 effectively predict student adaptability in online classes. 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, Random Forest Classifier, XGB classifier and Cat Boost 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:**

- User is shown the Home page. The user will browse through 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:

- 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 statistical
  - 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
  - Comparing model accuracy for different number of features.
  - Building model with appropriate features.
- Model Deployment
  - Save the best model
  - Integrate with Web Framework

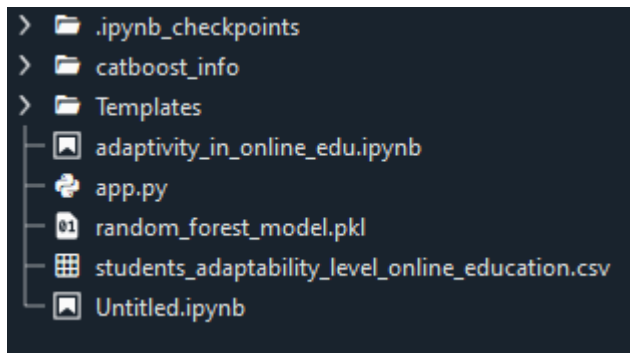
## **Prior Knowledge:**

You must have the prior knowledge of the following topics to complete this project.

- ML Concepts:
  - Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
  - Linear Regression: <https://www.javatpoint.com/linear-regression-in-machine-learning>
  - SVM: <https://www.javatpoint.com/machine-learning-support-vector-machine algorithm>
  - Decision tree: <https://www.javatpoint.com/machine-learning-decision-tree classificationalgorithm>
  - Random forest: <https://www.javatpoint.com/machine-learning-random-forest algorithm>
  - Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-modevaluation-error-metrics/>
  - Regularisation: <https://www.javatpoint.com/regularization-in-machine-learning>
- Flask Basics: [https://www.youtube.com/watch?v=lj4L\\_CvBnt0](https://www.youtube.com/watch?v=lj4L_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 which will require the html files to be stored in the templates folder.
- app.py file is used for routing purposes using scripting.
- random\_forest\_model.pkl is the saved model. This will further be used in the Flask integration.

## Milestone 1: Define Problem/ Problem Understanding

### **Activity 1: Specify the Business Problem**

The business problem in this scenario is to accurately predict the number of views a YouTube video will receive, which is crucial for advertisers to make informed decisions about which videos to sponsor and for content creators to optimize their content for maximum views and revenue. The goal is to develop a data-driven approach to forecasting YouTube video views that can be used by both advertisers and content creators to maximize their return on investment.

### **Activity 2: Business Requirements**

- **Accurate prediction:** The system should be able to accurately predict the adaptability level of individual students in online classes.
- **Data collection:** The system should be capable of collecting relevant data from various sources, such as learning management systems, online assessments, and student surveys.
- **Data preprocessing:** The system should preprocess and transform the collected data to ensure its quality and compatibility with machine learning algorithms.
- **Feature extraction:** The system should extract relevant features from the data that are indicative of student adaptability, such as academic performance, technological proficiency, self-regulated learning skills, motivation, and engagement.
- **Machine learning models:** The system should employ appropriate machine learning models, such as decision trees, neural networks, or ensemble methods, to train on the preprocessed data and predict student adaptability.
- **Model deployment:** The system should be able to deploy the trained model in a production environment to receive new data inputs and provide real-time adaptability predictions.
- **User interface:** The system should have a user-friendly interface that allows educators and instructors to input student data, visualize predictions, and access relevant insights.
- **Scalability and reliability:** The system should be scalable to handle a large volume of data and reliable to ensure accurate and consistent predictions for multiple students simultaneously.

### **Activity 3: Literature Survey**

The literature survey on predicting student adaptability in online classes highlights the significance of adaptability for student success in digital learning environments. It explores factors influencing adaptability such as prior academic performance, technological proficiency, self-regulated learning skills, motivation, engagement, and personal characteristics. The survey examines existing approaches, ranging from traditional statistical models to machine learning algorithms, and discusses relevant data sources for prediction. It emphasizes the need for accurate prediction models to enhance student outcomes and improve the effectiveness of online education.

### **Activity 4: Social or Business Impact.**

#### **Social Impact:**

Accurately predicting student adaptability in online classes can have a positive social impact by ensuring equal opportunities for all learners. By identifying students who may struggle in the digital learning environment, targeted interventions can be implemented to provide necessary support, thereby reducing dropout rates and enhancing educational equity.

#### **Business Impact:**

The prediction of student adaptability in online classes has significant business implications. Educational institutions can use the insights provided by the prediction model to optimize course design, enhance student satisfaction and retention rates, and improve the overall effectiveness of online education. This can lead to improved institutional reputation, increased student enrollment, and higher revenue generation.

## **Milestone 2: Data Collection and Preparation:**

Machine Learning depends heavily on data. It is the most crucial part aspect that makes algorithm training possible. So, this section guides on how to download dataset.

### **Activity 1: Collect the dataset**

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link:

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

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

**Note:** There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

## Activity 1.1: Importing the Libraries

### 1. Import Necessary Libraries

```
In [1]: # (1) Data
import numpy as np
import pandas as pd

# (2) 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

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

# (4) 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

# (5) Cluster Libraries
# from sklearn.cluster import KMeans
# from sklearn.decomposition import PCA
# from sklearn.metrics import silhouette_score
# from yellowbrick.cluster import KElbowVisualizer

# (6) Tuning Libraries
import optuna

# (7) Metrics & Scoring Libraries
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, plot_confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# (8) Options
%matplotlib inline
from termcolor import colored
plt.rcParams['axes.unicode_minus'] = False
pd.reset_option('display.float_format')
pd.set_option('display.max_columns', None)

color_scheme = px.colors.qualitative.Pastel
```

## Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called `read_csv()` to read the dataset. As a parameter we have to give the directory of the csv file.

### Read .csv file:

```
In [3]: 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)
```

Out[3]:

	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	18-20	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	18-20	School	Non Government	No	Yes	Low	Poor	Mobile Data	3G	0	No	Mobile	Low



## Milestone 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.

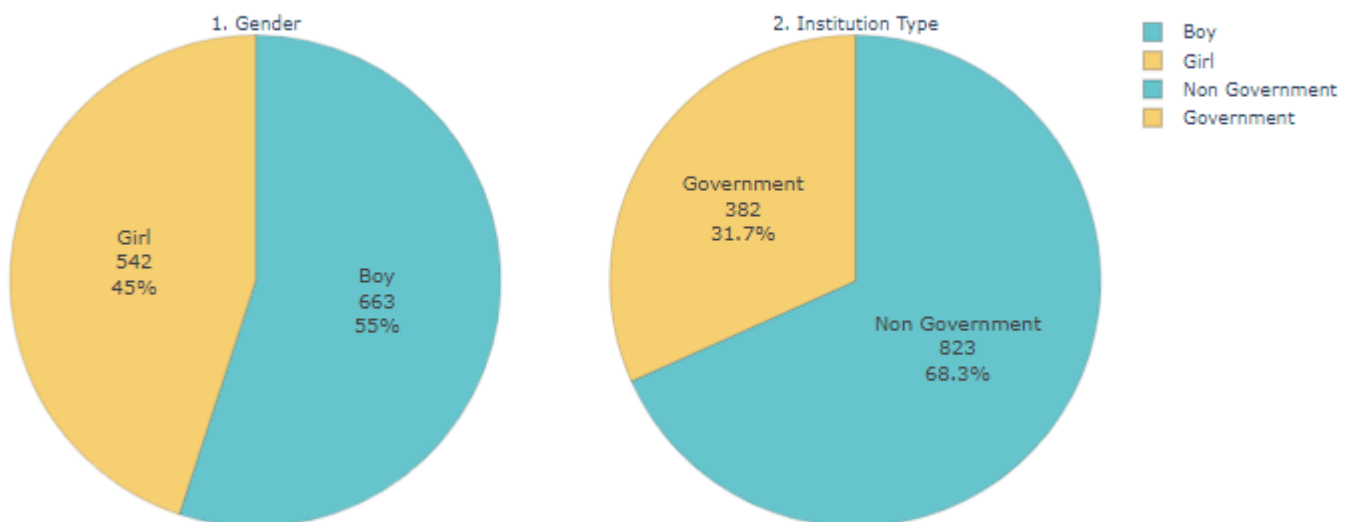
#### Activity 1.1: Univariate Analysis

- Gender and Institute type Distribution

```
In [4]: series_1 = []
series_1_title = ['Gender', 'Institution Type']
series_1.append(df['Gender'].value_counts())
series_1.append(df['Institution Type'].value_counts())

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='#100000', width=0.2)))
fig.show()
```

The given code is creating a subplot with two pie charts using the plotly library. The first pie chart represents the distribution of genders, and the second pie chart represents the distribution of institution types. Each chart is labeled with its respective title and shows the values, labels, and percentages. The color scheme is applied to distinguish different categories.



**Male and Female separately accounts for 55% and 45%**

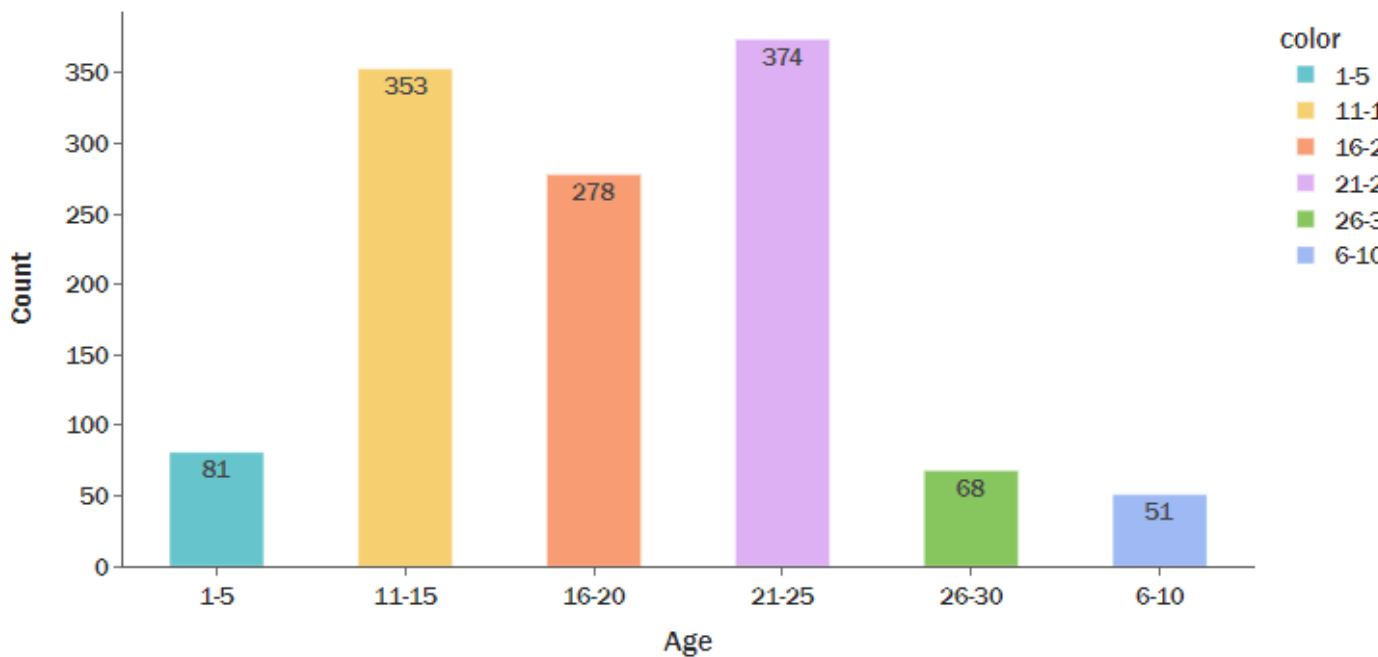
**Non Government type accounts for 68.3%**

- **Age Distribution**

```
In [5]: 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')
```

The code snippet calculates the frequency distribution of the 'Age' and 'Class Duration' variables from a DataFrame and sorts them in ascending order. Then, it creates two bar plots using the 'p\_bar\_plot' function. The first plot represents the distribution of ages, while the second plot represents the distribution of class durations. The plots are labeled with their respective titles and show the counts of each category.

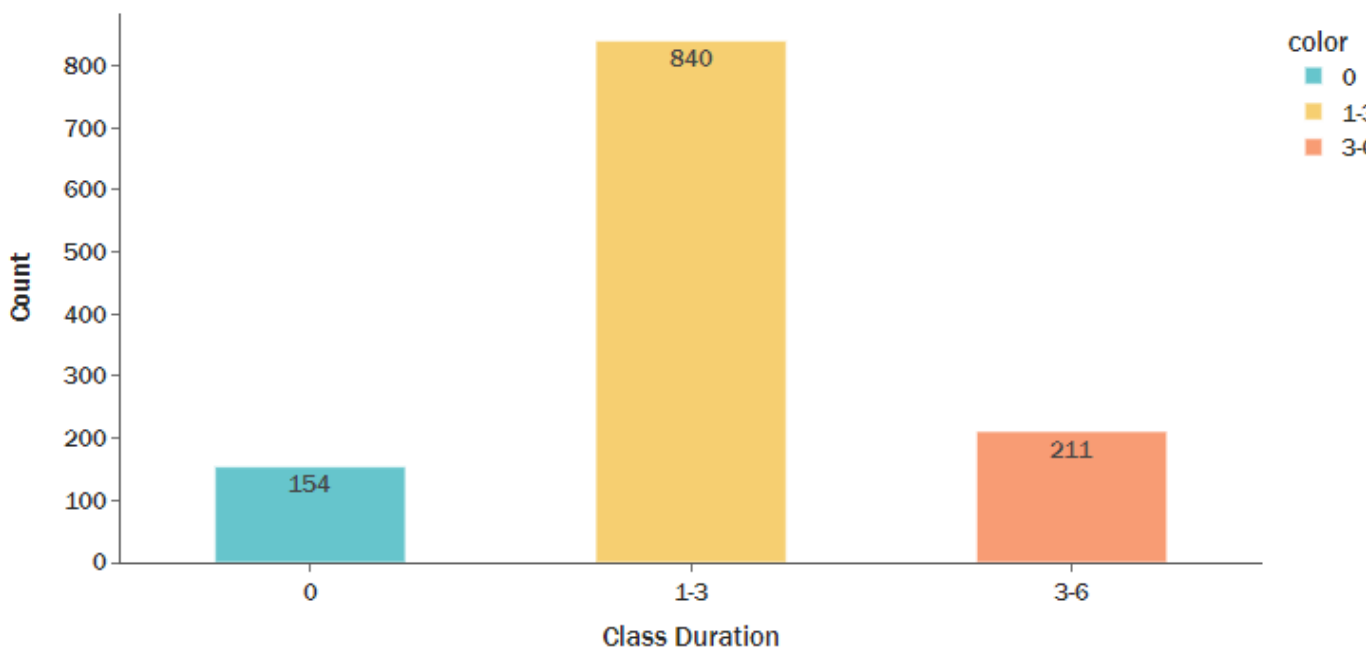
### 3. Age Distribution



**Age of the respondents is mainly distributed between 11 and 25**

- **Class Distribution**

### 4. Class Duration



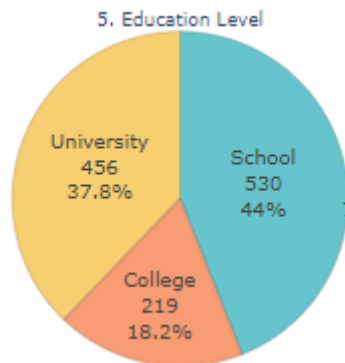
**Class Duration mainly distributed between 1-3 hours**

- **Education Level**

```
In [6]: 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())

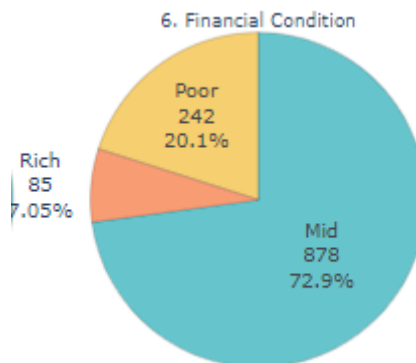
fig = make_subplots(rows=1, cols=3, specs=[[{"type": "pie"}, {"type": "pie"}, {"type": "pie"}]])
for idx, series in enumerate(series_2):
    fig.add_trace(go.Pie(values = series.values,
                        labels = series.index,
                        marker = dict(colors = color_scheme),
                        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='#100000', width=0.2)))
fig.show()
```

The provided code generates a subplot with three pie charts using the plotly library. Each pie chart represents the distribution of a categorical variable: 'Education Level', 'Financial Condition', and 'Network Type'. The charts are labeled with their respective titles and display the values, labels, and percentages. The color scheme is applied to differentiate the categories.



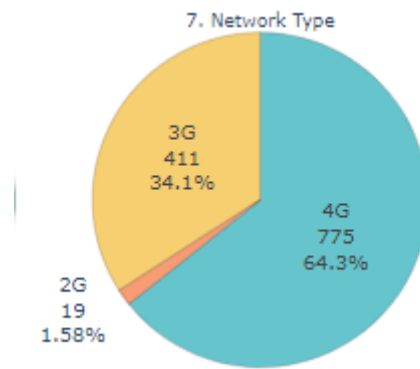
**Most of the respondents attend School and University**

- **Financial Condition**



**Most of them are situated in Mid financial condition**

- **Network Type**



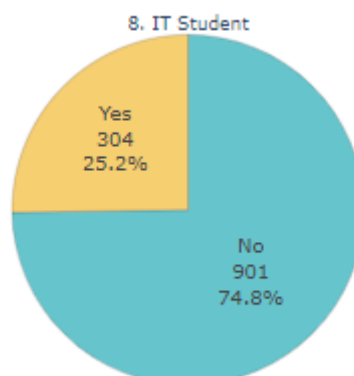
**There are just 1.58% of students using 2G network**

- **IT Student**

```
In [7]: 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())

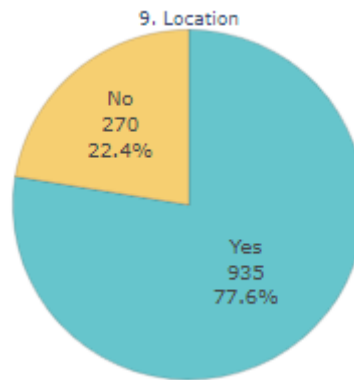
fig = make_subplots(rows=1, cols=3, specs=[[{"type": "pie"}, {"type": "pie"}, {"type": "pie"}]])
for idx, series in enumerate(series_3):
    fig.add_trace(go.Pie(values = series.values,
                        labels = series.index,
                        marker = dict(colors = color_scheme),
                        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()
```

The code snippet creates a subplot with three pie charts using the plotly library. Each pie chart represents the distribution of a categorical variable: 'IT Student', 'Location', and 'Self Lms'. The charts are labeled with their respective titles and display the values, labels, and percentages. A color scheme is applied to differentiate the categories.



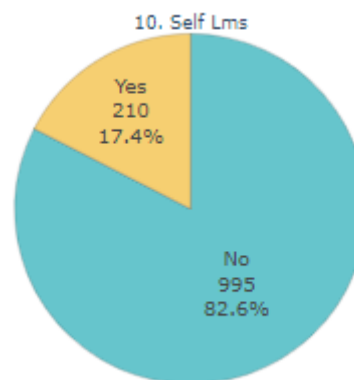
**About 25.2% of respondents are studying as IT student**

- **Location**



**And 77.6% of them are located in town**

- **Self Lms**



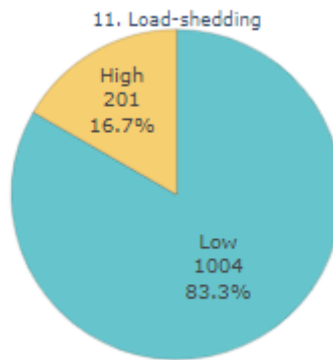
**Furthermore, 17.4% of their institutions own LMS availability**

- **Load-shedding**

```
In [8]: 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())

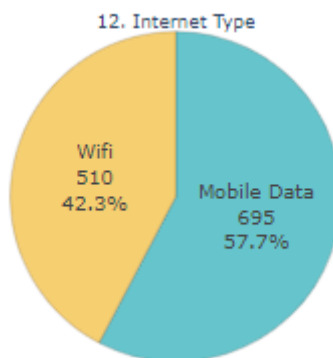
fig = make_subplots(rows=1, cols=3, specs=[[{"type": "pie"}, {"type": "pie"}, {"type": "pie"}]])
for idx, series in enumerate(series_4):
    fig.add_trace(go.Pie(values = series.values,
                        labels = series.index,
                        marker = dict(colors = color_scheme),
                        title = str(idx+1)+'.' +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()
```

The given code generates a subplot with three pie charts using the plotly library. Each pie chart represents the distribution of a categorical variable: 'Load-shedding', 'Internet Type', and 'Device'. The charts are labeled with their respective titles and display the values, labels, and percentages. The color scheme is applied to distinguish the different categories.



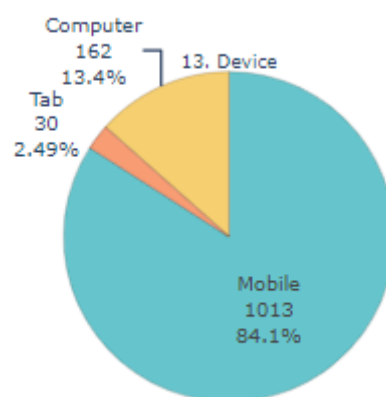
**Load-shedding refers to level of load shedding, and type 'low' accounts for 83.3%**

- **Internet Type**



**57.7% of respondents use mobile data to take online education**

- **Device**

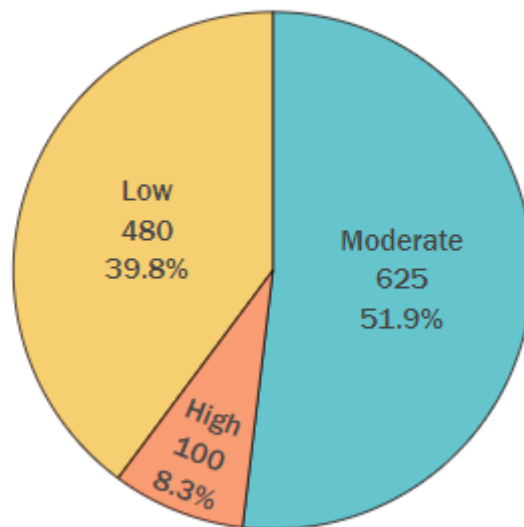


**Furthermore, 84.1% of them use mobile to take online class**

- **Adaptivity Level (Target Column)**

```
In [9]: p_pie_plot(df['Adaptivity Level'].value_counts(), '14. Adaptivity Level (Target Column)')
```

### 14. Adaptivity Level (Target Column)



**Adaptability level which is our target column refers to the adaptability level of the student during online education.**

**It can be seen that about 51.9% respond they have moderate adaptability, while low accounts for 39.8%, and high the rest.**

## Activity 1.2: Multivariate Analysis

- Age Distribution by Gender

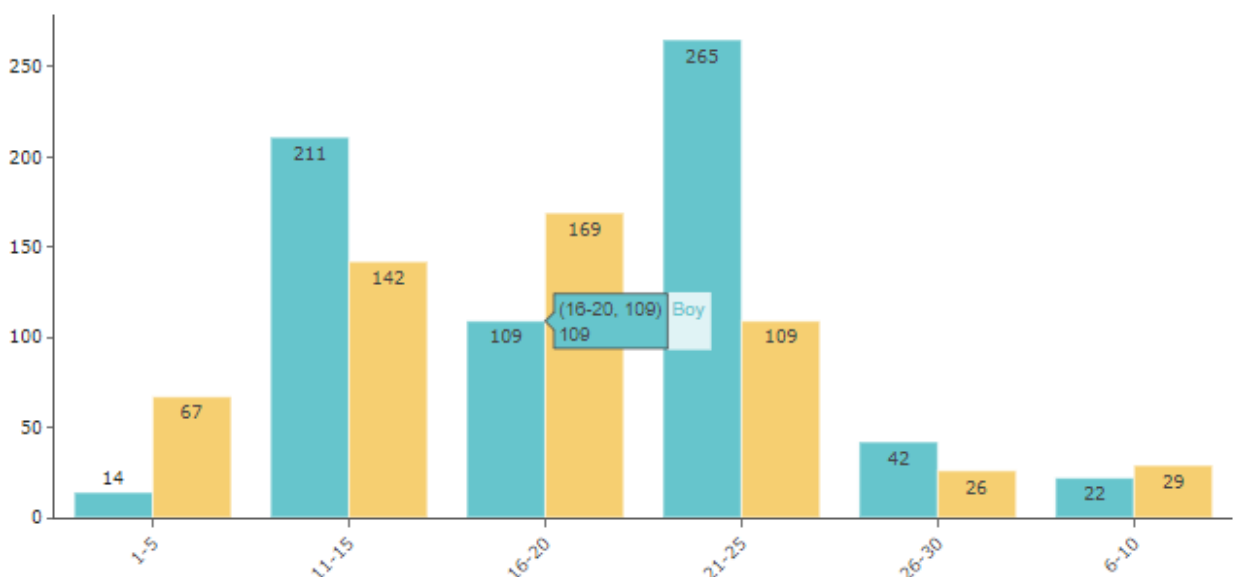
```
In [10]: gender_age = df.groupby(['Gender', 'Age']).size()
fig = go.Figure(data=[
    go.Bar(name='Boy', x=gender_age['Boy'].index, y=gender_age['Boy'].values,
        text=gender_age['Boy'].values, marker_color=color_scheme[0]),

    go.Bar(name='Girl', x=gender_age['Girl'].index, y=gender_age['Girl'].values,
        text=gender_age['Girl'].values, marker_color=color_scheme[1])
])
fig.update_layout(barmode='group', xaxis_tickangle=-45, title='Age Distribution by Gender',
    template = 'simple_white')

fig.show()
```

The provided code generates a grouped bar chart using the plotly library. It groups the data based on 'Gender' and 'Age' variables from a DataFrame. The chart consists of two bars, representing 'Boy' and 'Girl' categories. The x-axis represents the age groups, while the y-axis represents the count of individuals in each age group. The bars are labeled with their respective values, and different color schemes are used to differentiate the categories. The chart's title is set as 'Age Distribution by Gender', and a simple white template is applied to the layout.

Age Distribution by Gender



**Female's age mainly distributed in 11-20 while age of male mostly distributed between 11-15 and 21-25**



- **IT Student Distribution by Gender**

```
In [11]: 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=color_scheme[0]),

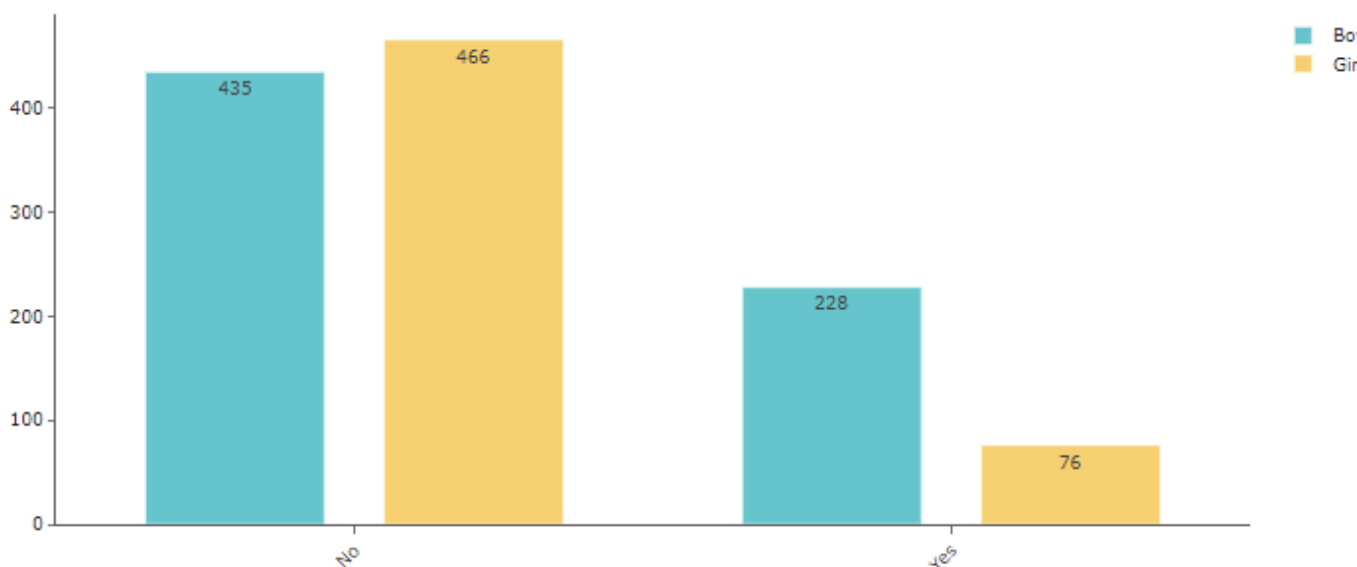
    go.Bar(name='Girl', x=gender_IT['Girl'].index, y=gender_IT['Girl'].values,
        text=gender_IT['Girl'].values, marker_color=color_scheme[1])
])
fig.update_layout(barmode='group', xaxis_tickangle=-45, title='Distribution of IT Students by Gender',
    template = 'simple_white')

fig.update_traces(width=0.3)

fig.show()
```

The given code creates a grouped bar chart using the plotly library. It groups the data based on 'Gender' and 'IT Student' variables from a DataFrame. The chart consists of two bars, representing 'Boy' and 'Girl' categories. The x-axis represents the 'IT Student' categories, while the y-axis represents the count of individuals in each category. The bars are labeled with their respective values, and different color schemes are used to distinguish the categories. The chart's title is set as 'Distribution of IT Students by Gender', and a simple white template is applied to the layout. The width of the bars is adjusted to 0.3 for better visibility.

Distribution of IT Students by Gender



**We can notice that males tend to study as IT students compared to females in the survey**

- **IT Student Distribution by Education**

```
In [12]: 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=color_scheme[0]),

    go.Bar(name='School', x=Education_IT['School'].index, y=Education_IT['School'].values,
        text=Education_IT['School'].values, marker_color=color_scheme[1]),

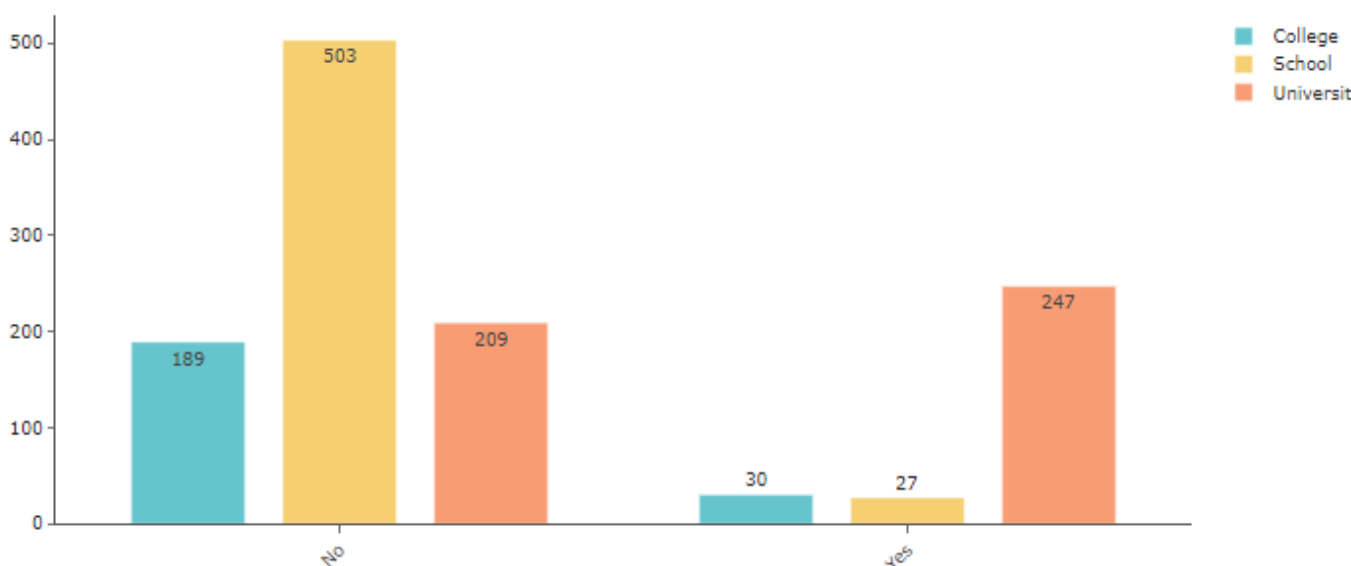
    go.Bar(name='University', x=Education_IT['University'].index, y=Education_IT['University'].values,
        text=Education_IT['University'].values, marker_color=color_scheme[2])
])
fig.update_layout(barmode='group', xaxis_tickangle=-45,title='Distribution of IT Students by Edu',
    template = 'simple_white')

fig.update_traces(width=0.2)

fig.show()
```

The provided code generates a grouped bar chart using the plotly library. It groups the data based on 'Education Level' and 'IT Student' variables from a DataFrame. The chart consists of three bars, representing 'College', 'School', and 'University' categories. The x-axis represents the 'Education Level' categories, while the y-axis represents the count of individuals in each category. The bars are labeled with their respective values, and different color schemes are used to differentiate the categories. The chart's title is set as 'Distribution of IT Students by Education Level', and a simple white template is applied to the layout. The width of the bars is adjusted to 0.2 for better visibility.

Distribution of IT Students by Edu



**It is obvious that university got the most IT students, whereas 30 and 27 IT students in college and school.**

- **Gender Distribution by Adaptivity Level**

```
In [13]: 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=color_scheme[0]),

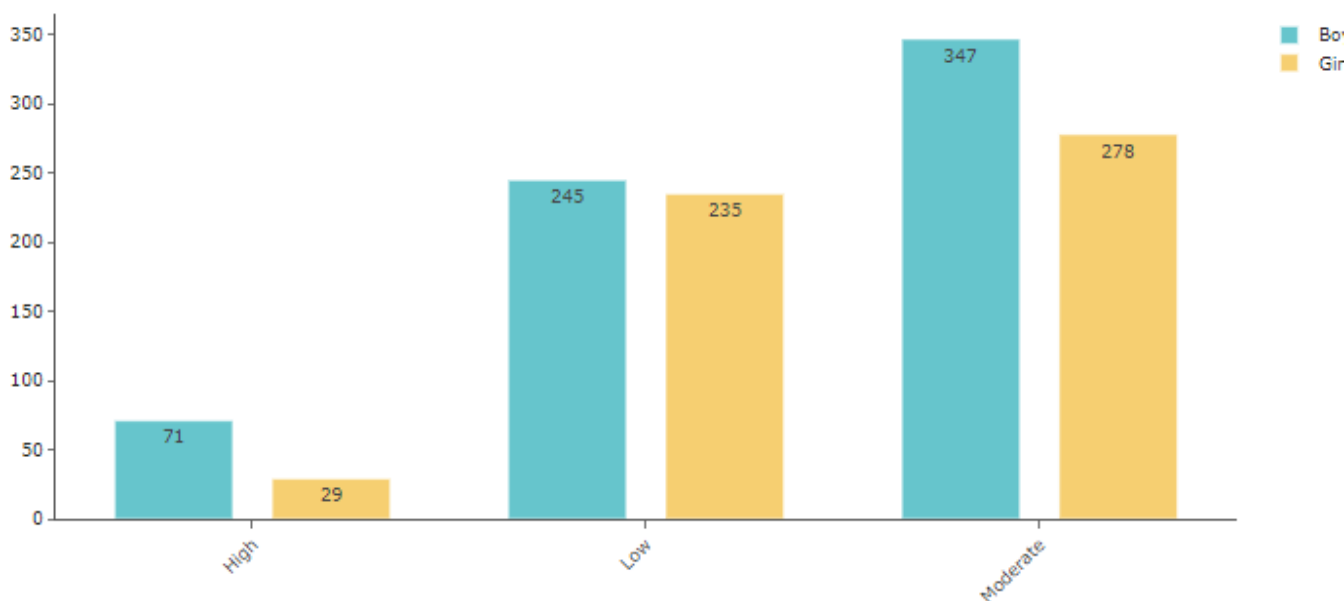
    go.Bar(name='Girl', x=Gender_Adaptivity['Girl'].index, y=Gender_Adaptivity['Girl'].values,
        text=Gender_Adaptivity['Girl'].values, marker_color=color_scheme[1]),
])
fig.update_layout(barmode='group', xaxis_tickangle=-45,title='Distribution of Gender by Adaptivity Level',
    template = 'simple_white')

fig.update_traces(width=0.3)

fig.show()
```

The given code generates a grouped bar chart using the plotly library. It groups the data based on 'Gender' and 'Adaptivity Level' variables from a DataFrame. The chart consists of two bars, representing 'Boy' and 'Girl' categories. The x-axis represents the 'Adaptivity Level' categories, while the y-axis represents the count of individuals in each category. The bars are labeled with their respective values, and different color schemes are used to differentiate the categories. The chart's title is set as 'Distribution of Gender by Adaptivity Level', and a simple white template is applied to the layout. The width of the bars is adjusted to 0.3 for better visibility.

Distribution of Gender by Adaptivity Level



**In the survey, 71 male respondents answered as they have a high adaptivity level in online education compared to 29 females.**

**Additionally, both male and female overall got moderate adaptivity levels.**

- **Age Distribution by Adaptivity Level**

```
In [14]: 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=color_scheme[0]),

    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=color_scheme[1]),

    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=color_scheme[2]),

    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=color_scheme[3]),

    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=color_scheme[4]),

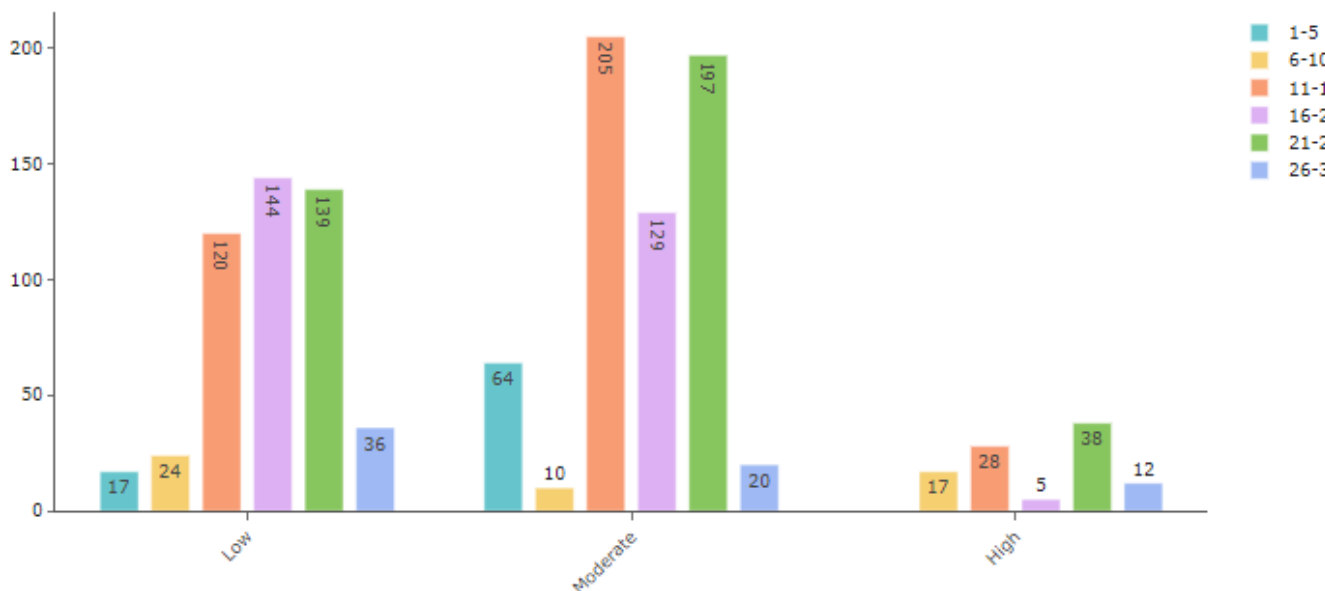
    go.Bar(name='26-30', x=Age_Adaptivity['26-30'].index, y=Age_Adaptivity['26-30'].values,
        text=Age_Adaptivity['26-30'].values, marker_color=color_scheme[5]),
])
fig.update_layout(barmode='group', xaxis_tickangle=-45, title='Distribution of Age by Adaptivity Level',
    template = 'simple_white')

fig.update_traces(width=0.1)

fig.show()
```

The provided code generates a grouped bar chart using the plotly library. It groups the data based on 'Age' and 'Adaptivity Level' variables from a DataFrame. The chart consists of multiple bars representing different age groups. Each age group is labeled on the x-axis, while the y-axis represents the count of individuals in each category. The bars are labeled with their respective values, and different color schemes are used to differentiate the age groups. The chart's title is set as 'Distribution of Age by Adaptivity Level', and a simple white template is applied to the layout. The width of the bars is adjusted to 0.1 for better visibility.

Distribution of Age by Adaptivity Level



**11-15, 21-25 and 26-30 relatively got high adaptivity level while obvious that none of 1-5 got high adaptivity**

## Activity 3: Pre- Processing

### Get columns' name of dataframe

```
In [15]: # df.columns.tolist()
'''
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 3.1: Encoding

Since data are composed of categorical values, use `LabelEncoder()` to encode into numeric values

```
In [16]: for column in columns:
         le = LabelEncoder()
         df[column] = le.fit_transform(df[column])
```

```
In [17]: df.head(4)
```

```
Out[17]:
```

	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	0	3	2	1	0	1	1	0	1	2	2	0	2	2
1	1	3	2	1	0	1	0	0	0	2	1	1	1	2
2	1	2	0	0	0	1	1	0	1	2	1	0	1	2
3	1	1	1	1	0	1	1	0	0	2	1	0	1	2

The code snippet applies label encoding to each column in the 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 assigned back to the respective columns in the DataFrame.

## Activity 3.2: Train-Test Split

The code splits the DataFrame `df` into input features `X` and the target variable `y`. The target variable is assigned to `y`, while the input features are assigned to `X` after removing the 'Adaptivity Level' column using the `drop()` function. 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 30% for testing (`X_test`, `y_test`), ensuring reproducibility with a random state of 116.

### Split dataframe into train and test set

```
In [18]: 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 = 0.3, random_state = 116)

In [19]: 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)
```

## Milestone 4: Model Building

### Activity 4.1: Define Pre-Processing and Modelling Function

- 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.

```
In [20]: class CONFIG:
    FOLD = 5                # Cross Validation Fold Num
    RANDOM_STATE = 116     # Random State
    # 4 models
    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
    FEATURES = ['Gender', 'Age', 'Education Level', 'Institution Type', 'IT Student', 'Location',
                'Load-shedding', 'Financial Condition', 'Internet Type', 'Network Type',
                'Class Duration', 'Self Lms', 'Device']

    # CatBoost Best Parameters (using optuna)
    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}

    # XGBoost Best Parameters (using optuna)
    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}
```

- The `_scale` function takes in training and validation data along with a list of features. It applies the `StandardScaler` to standardize the numerical features in the training data. It then uses the computed scaler to transform both the training and validation data. The function returns the updated training and validation data with the scaled features. This ensures that the features have zero mean and unit variance, which can be beneficial for certain machine learning algorithms.

```
In [21]: # 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
```

- The `classifiers_modeling` function takes in a dictionary of classifiers, training and testing data, and a list of features. It performs cross-validation with 5 folds using `KFold` and trains each classifier on the training data. The function scales the features using the `_scale` function. It then calculates the accuracy score for each fold and computes the mean accuracy score across all folds. The function returns a list of the mean accuracy scores for each classifier. The accuracy scores are printed for each classifier during the process.

```
In [22]: def classifiers_modeling(classifiers, X_train, y_train_, X_test, y_test, features):
    accuracy_list = []
    classifiers_name = list(classifiers.keys())
    # 5 Fold
    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('(',idx+1,')', classifiers_name[idx], 'cross validation (5 fold)')
    print('Mean Accuracy Score: ', round(accuracy, 5))
    # print('Mean Accuracy Score: ', colored(round(accuracy,5)))
    return accuracy_list
```

- The `p_bar_plot` function generates a bar plot using Plotly Express (`px.bar`) based on the provided data. It takes in parameters such as x-values, y-values, bar width, x-axis title, y-axis title, and plot title. The function creates a bar plot with colored bars based on the x-values, and the y-values are displayed as text on the bars. The plot is displayed with a white template and appropriate axis titles and title.

```
In [23]: # Model Score Visualization
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()
```

## Activity 4.2: Model Comparison (Part-1)

- The code provided performs cross-validation using the `classifiers_modeling` function on a set of classifiers defined in the `CONFIG.CLASSIFIERS` dictionary. The function calculates the mean accuracy score for each classifier using 5-fold cross-validation on the training data. The results show the index of each classifier, its name, and the mean accuracy score obtained. The Logistic Regression classifier achieved a mean accuracy score of 0.68809, followed by KNearest with 0.76875, RandomForestClassifier with 0.89568, XGBClassifier with 0.89094, and CatBoostClassifier with 0.88619.

```
In [24]: classifiers_name = list(CONFIG.CLASSIFIERS.keys())
classifiers_accuracy = []
```

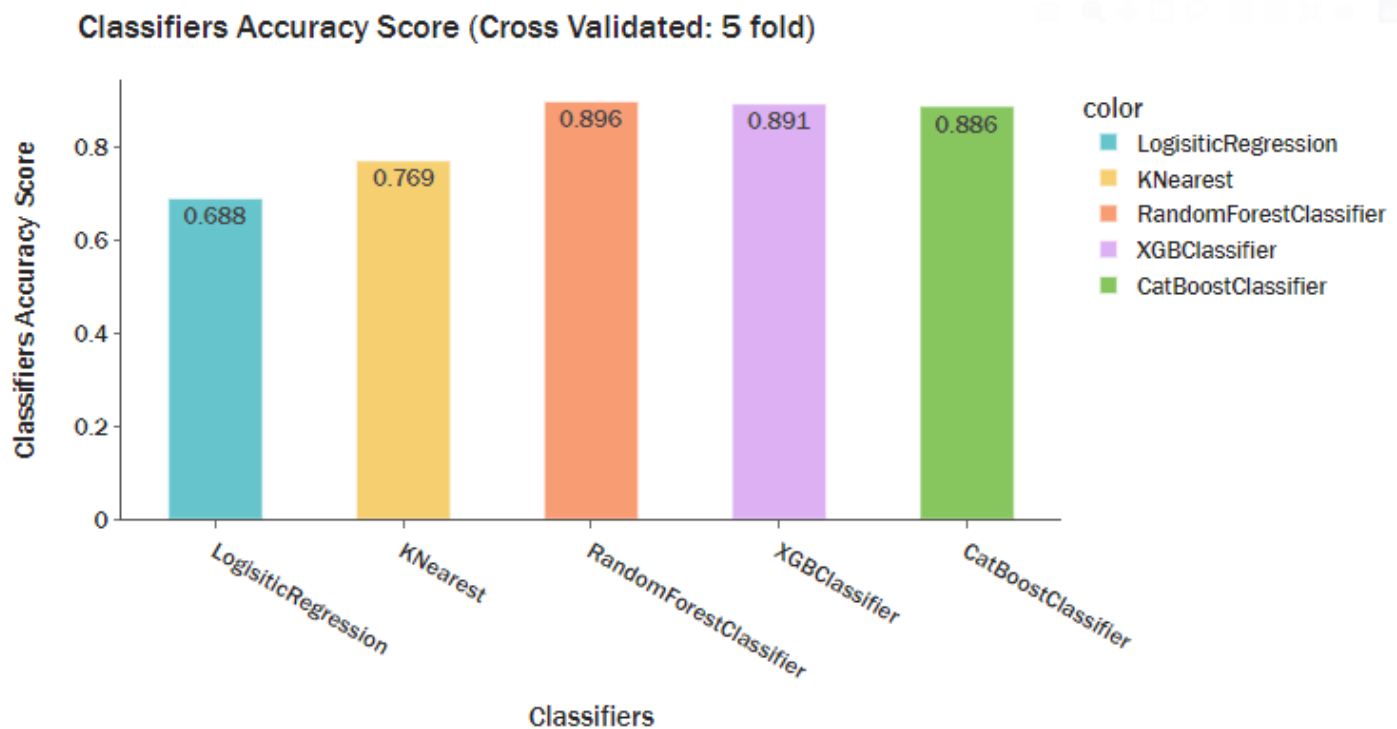
```
In [25]: classifiers_accuracy = classifiers_modeling(CONFIG.CLASSIFIERS, X_train, y_train, X_test, y_test, CONFIG.FEATURES)

( 1 ) LogisticRegression cross validation (5 fold)
Mean Accuracy Score: 0.68809
( 2 ) KNearest cross validation (5 fold)
Mean Accuracy Score: 0.76875
( 3 ) RandomForestClassifier cross validation (5 fold)
Mean Accuracy Score: 0.89568
( 4 ) XGBClassifier cross validation (5 fold)
Mean Accuracy Score: 0.89094
( 5 ) CatBoostClassifier cross validation (5 fold)
Mean Accuracy Score: 0.88619
```



- 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. The plot is titled "Classifiers Accuracy Score (Cross Validated: 5 fold)" and provides a visual comparison of the accuracy performance of different classifiers.

```
In [26]: p_bar_plot(x = classifiers_name,  
                    y = np.round(classifiers_accuracy,3),  
                    width = 0.5,  
                    x_axis_title = 'Classifiers',  
                    y_axis_title = 'Classifiers Accuracy Score',  
                    title = 'Classifiers Accuracy Score (Cross Validated: 5 fold)'  
                    )
```



We can see that RandomForestClassifier, XGBClassifier and CatBoostClassifier got highest accuracy.

I've chosen XGBClassifier and CatBoostClassifier to get optimum hyperparameters using optuna

## Activity 4.3: Hyper-parameter 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.

### Activity 4.3.1: CatBoost Classifier Hyperparameter Tuning

CatBoost Classifier is a gradient boosting algorithm that is known for its high performance and ability to handle categorical features without the need for explicit encoding. When tuning the hyperparameters of the CatBoost Classifier, important parameters to consider include the number of iterations, depth of the trees, learning rate, random strength, and bagging temperature. Hyperparameter tuning for the CatBoost Classifier can be done using techniques like grid search, random search, or more advanced methods like Bayesian optimization.

```
def cb_objective(trial):
    # CB Classifier Params
    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
    }
    # Set Accuracy to zero
    accuracy = 0
    # Create Model
    cb_model = CatBoostClassifier(**params)
    # 5 Fold
    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 = cb_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 /5 fold accuracy score
    return accuracy

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

```
[I 2023-06-27 17:20:42,778] A new study created in memory with name: no-name-eac7551b-3ab6-4627-90b2-d1aee276ff01
C:\Users\kamya\AppData\Local\Temp\ipykernel_78540\3106833583.py:6: FutureWarning:
```

The code defines the objective function `cb_objective` for hyperparameter tuning of the CatBoost Classifier using Optuna. The function takes a trial object as input and suggests values for various hyperparameters such as `iterations`, `depth`, `learning_rate`, `random_strength`, `bagging_temperature`, `od_type`, and `od_wait`. It then trains and evaluates the `CatBoostClassifier` model using 5-fold cross-validation. The objective is to maximize the accuracy score on the validation set. The `optuna.create_study` function creates a study object, and the `study.optimize` function performs hyperparameter optimization by running the objective function for a specified number of trials or timeout.

### Activity 4.3.2: XGBClassifier Hyperparameter Tuning

The XGBClassifier hyperparameter tuning involves optimizing the parameters of the XGBoost classifier algorithm to improve its performance. This can be done using techniques like grid search, random search, or Bayesian optimization. By systematically exploring different combinations of hyperparameters, the goal is to find the set of values that maximizes the desired evaluation metric, such as accuracy or F1 score.

```
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
    }

    # Get Accuracy
    accuracy = 0
    # Create Model
    xgb_model = XGBClassifier(**param)
    # 5 Fold
    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)
```

The given code is performing hyperparameter optimization for an XGBoost classifier using Optuna library. It defines an objective function, "xgb\_objective," which takes a set of hyperparameters and trains an XGBoost model using k-fold cross-validation. The objective is to maximize the accuracy score. The code then creates an Optuna study and runs the optimization process, aiming to find the best hyperparameters within a specified time limit of 600 seconds (10 minutes) or until 120 trials have been completed.

## Activity 4.4: Model Comparison (part 2)

The code defines a dictionary called "tuned\_classifiers" that contains two tuned classifiers, CatBoostClassifier and XGBClassifier, with their respective best parameters. The names of these classifiers are then added to a list called "classifiers\_name." The code then proceeds to evaluate the performance of these tuned classifiers using the "classifiers\_modeling" function, which takes the classifiers, training data (X\_train and y\_train), test data (X\_test and y\_test), and a list of features. The results of the evaluation, stored in the "tuned\_accuracy" list, are then appended to the "classifiers\_accuracy" list.

```
In [27]: tuned_classifiers = {
        "CatBoostClassifier": CatBoostClassifier(**CONFIG.CB_BEST_PARAMS),
        "XGBClassifier": XGBClassifier(**CONFIG.XGB_BEST_PARAMS)
        }

In [28]: 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)

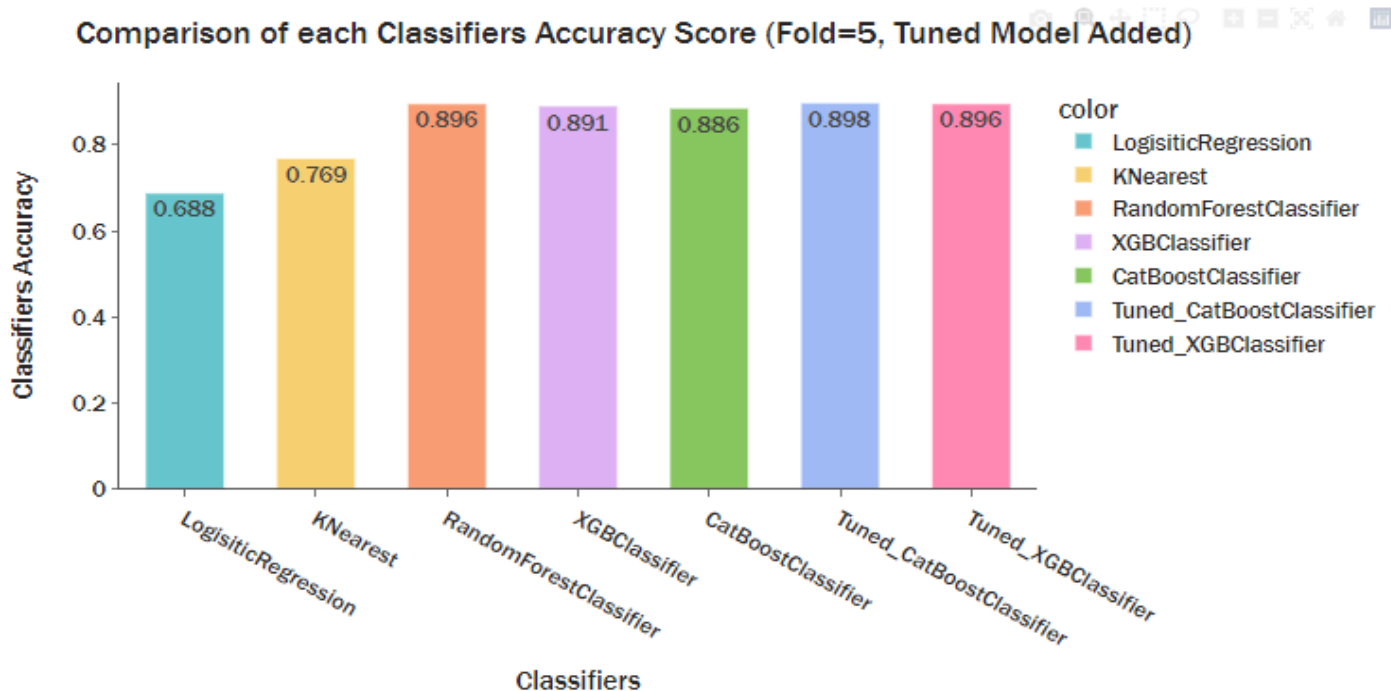
        for tuned_accuracy_score in tuned_accuracy:
            classifiers_accuracy.append(tuned_accuracy_score)

( 1 ) CatBoostClassifier cross validation (5 fold)
Mean Accuracy Score: 0.89806
( 2 ) XGBClassifier cross validation (5 fold)
Mean Accuracy Score: 0.89568
```

Interpreting the results:

- CatBoostClassifier cross-validation (5-fold): The mean accuracy score achieved by the CatBoostClassifier, based on 5-fold cross-validation, is 0.89806. This indicates that, on average, the classifier predicts the correct class with an accuracy of 89.806%.
- XGBClassifier cross-validation (5-fold): The mean accuracy score achieved by the XGBClassifier, based on 5-fold cross-validation, is 0.89568. This means that, on average, the classifier predicts the correct class with an accuracy of 89.568%.

```
In [29]: p_bar_plot(x = classifiers_name,
                    y = np.round(classifiers_accuracy,3),
                    width = 0.6,
                    x_axis_title = 'Classifiers',
                    y_axis_title = 'Classifiers Accuracy',
                    title = 'Comparison of each Classifiers Accuracy Score (Fold=5, Tuned Model Added) ')
```



Accuracy has improved by about 1% compared to before.

Random Forest Performed the best, therefore we will save this model.

## Milestone 5 : Performance Testing

### Activity 5.1: Comparing all the Models.

The code snippet performs evaluation and saves the results of a model. It imports several metrics from `sklearn.metrics`, including `accuracy_score`, `classification_report`, `confusion_matrix`, `roc_curve`, and `roc_auc_score`. The model's predictions are compared with the true labels (`y_test`), and the accuracy score is calculated. Additionally, a classification report is generated using the predicted labels and the true labels. Finally, the results, including the predictions, accuracy, and classification report, are stored in a dictionary called "results."

```
In [40]: from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, roc_auc_score

# Evaluate the Model
accuracy = accuracy_score(y_test, predictions)
classification_report = classification_report(y_test, predictions)

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

# Save Results
results = {
    'predictions': predictions,
    'accuracy': accuracy,
    'classification_report': classification_report,
}
```

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

     0       0.94         0.54         0.68         28
     1       0.88         0.94         0.91        142
     2       0.91         0.93         0.92        192

 accuracy          0.90         0.90         0.90        362
  macro avg       0.91         0.80         0.84        362
 weighted avg     0.90         0.90         0.90        362
```

The evaluation results are as follows:

1. **Accuracy: 0.9005524861878453**- The accuracy score of the model is 0.9005, indicating that approximately 90.05% of the predictions are correct.
2. **Classification Report:** The classification report provides metrics such as precision, recall, and F1-score for each class (0, 1, and 2), along with their respective support (number of instances).
3. **Based on the report:**
  - Class 0 has a precision of 0.94, meaning that 94% of the predicted instances of class 0 are correct. However, its recall is 0.54, indicating that only 54% of the actual instances of class 0 are correctly predicted.
  - Class 1 has a precision of 0.88 and a recall of 0.94, indicating that the model performs well in predicting instances of class 1.
  - Class 2 also shows good performance, with a precision of 0.91 and a recall of 0.93.
  - The macro average F1-score is 0.84, which is the average of the F1-scores across all classes, giving equal weight to each class. The weighted average F1-score is 0.90, considering the class support.

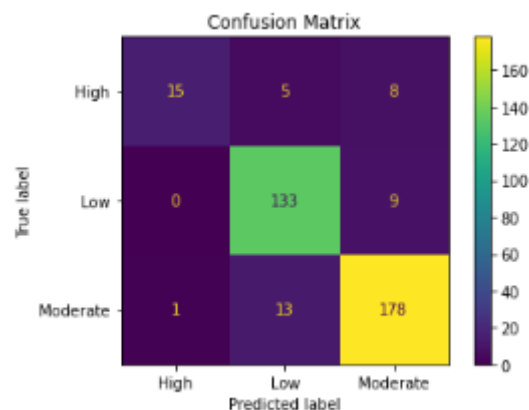
Overall, the model demonstrates relatively high accuracy and performs well in predicting classes 1 and 2. However, it has a lower performance in predicting class 0, which is reflected in its precision and recall values for that class.

## Activity 5.2: Visualization of Confusion Matrix

The code calculates the confusion matrix using the true labels (`y_test`) and predicted labels (`predictions`). It then displays the confusion matrix using `ConfusionMatrixDisplay`, with class labels "High," "Low," and "Moderate." The resulting plot shows the counts of true positives, true negatives, false positives, and false negatives for each class, providing a visual representation of the model's performance.

```
In [39]: import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, predictions)
disp = ConfusionMatrixDisplay(cm, display_labels=["High", "Low", "Moderate"])
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```



## Activity 5.4: Feature Importance Analysis and Visualization

Calculates the feature importances for a random forest model and stores them in a DataFrame. The DataFrame is then sorted in descending order based on importance values. The sorted feature importances are displayed and the top 5 important features are visualized in a horizontal bar plot, showing their respective importance values.

```
In [44]: # Obtain feature importances
feature_importances = random_forest.feature_importances_

# Create a DataFrame to store feature importances
feature_importances_df = pd.DataFrame({'Variable': X_train.columns, 'Variable Importance': feature_importances})

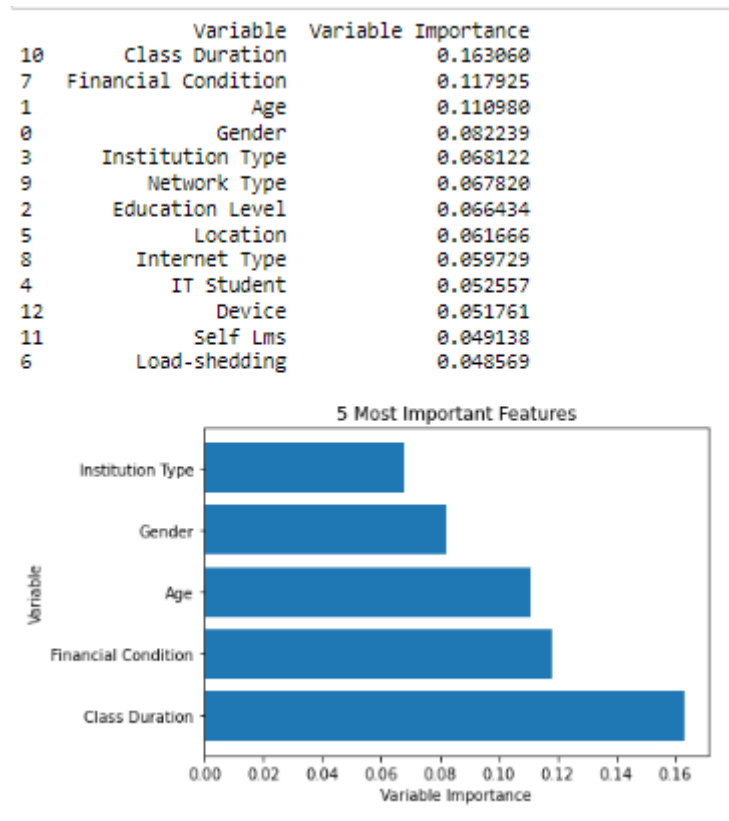
# Sort the DataFrame by variable importance in descending order
feature_importances_df = feature_importances_df.sort_values('Variable Importance', ascending=False)

# Display the sorted feature importances
print(feature_importances_df)

# Plot the 5 most important features
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.show()
```

The feature importance analysis reveals the importance of various variables in predicting the target variable. The top five most important features are "Class Duration" (16.3%), "Financial Condition" (11.8%), "Age" (11.1%), "Gender" (8.2%), and "Institution Type" (6.8%). These features have the highest impact on the model's predictions. Other features also contribute to the model's performance but to a lesser extent, as indicated by their lower importance values.





## Milestone 6: Model Deployment

### Activity 1: Save and load the best model

The provided code snippet demonstrates the usage of joblib library for training and saving a Random Forest Classifier model. First, the Random Forest Classifier is trained using the training data `X_train` and target labels `y_train`. The trained model is then saved to a file named `'random_forest_model.pkl'` using the `joblib.dump()` function.

Later, to make predictions on new data, the saved model is loaded from the file using the `joblib.load()` function. The loaded model can then be used to make predictions on the test data `X_test` by calling the `model.predict()` method, which returns an array of predicted labels.

```
In [31]: import joblib

# Train the Random Forest Classifier
random_forest = CONFIG.CLASSIFIERS["RandomForestClassifier"]
random_forest.fit(X_train[CONFIG.FEATURES], y_train)

# Save the trained model to a file
joblib.dump(random_forest, 'random_forest_model.pkl')

Out[31]: ['random_forest_model.pkl']

In [32]: import joblib

# Load the saved model from file
model = joblib.load('random_forest_model.pkl')

# Make predictions on new data
predictions = model.predict(X_test[CONFIG.FEATURES])
```

We save the model using the pickle library into a file named `random_forest_model.pkl`

## Activity 2: Integrate with Web Framework

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

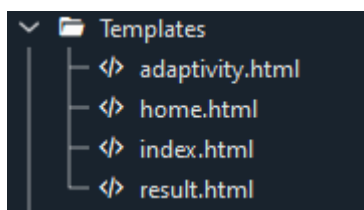
- Building HTML Pages
- Building server-side script
- Run the web application

### Activity 2.1: Building HTML pages:

For this project we create two HTML files namely

- Index.html
- Results.html
- home.html
- adaptivity.html

And we will save them in the templates folder.



### Activity 2.2: Build Python code

Create a new app.py file which will be stored in the Flask folder.

- Import the necessary Libraries.

```
from flask import Flask, render_template, request
import joblib
import numpy as np
```

This code uses the joblib module to load a saved machine learning model and a saved preprocessing object. The joblib.load function is used to load the model object from a file called model.pkl and the scaler object from a file called random\_forest\_model.pkl. These objects are then stored in the model and scaler variables, respectively. The loaded model object can be used to make predictions on new data and the loaded scaler object can be used to preprocess the data in the same way as it was done during training. This process of loading saved model and preprocessing objects can save time and resources when working on a new project or dataset.

```
app = Flask(__name__)
model = joblib.load('random_forest_model.pkl')
```

This code creates a new instance of a Flask web application using the Flask class from the Flask library. The \_\_name\_\_ argument specifies the name of the application's module or package.

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



The provided code represents a Flask web application with three routes. The first route, '/', is mapped to the 'home' function. When a user accesses the root URL of the application, the 'home.html' template is rendered, which serves as the landing page. The second route, '/predict', is mapped to the 'index' function. This route is responsible for rendering the 'index.html' template, which typically contains a form or interface where users can input data and make predictions. Finally, the '/adapt' route is mapped to the 'adapt' function, which renders the 'adaptivity.html' template. This route is intended for displaying information or results related to adaptivity. These routes and their associated templates define the structure and functionality of the web application, enabling users to navigate and interact with different pages or features.

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

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

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

The provided code snippet defines a Flask route '/predict' that accepts POST requests. When a POST request is received, the 'predict' function is executed. Inside this function, form data is retrieved using the 'request.form' object. The code assigns the form values to respective variables such as 'gender', 'age\_range', 'class\_duration\_range', 'financial\_condition', 'institute\_type', 'network\_type', 'education\_level', 'location', 'internet\_type', 'it\_student', 'device', 'self\_lms', and 'load\_shedding'. These variables capture the user's input from the form. This data can then be used for further processing or prediction purposes within the function.

```
@app.route('/predict', methods=['POST'])
def predict():
    gender = request.form['gender']
    age_range = request.form['age']
    class_duration_range = request.form['class_duration']
    financial_condition = request.form['financial_condition']
    institute_type = request.form['institute_type']
    network_type = request.form['network_type']
    education_level = request.form['education_level']
    location = request.form['location']
    internet_type = request.form['internet_type']
    it_student = request.form['it_student']
    device = request.form['device']
    self_lms = request.form['self_lms']
    load_shedding = request.form['load_shedding']
```

The provided code snippet performs encoding for the categorical variables obtained from the form data. Each variable is transformed into a numerical representation using specific encoding rules. For example, 'gender' is encoded as 1 for 'girl' and 0 for other genders. 'Age' and 'class\_duration' are converted from ranges to numerical values by taking the average of the range. Other variables such as 'financial\_condition', 'institute\_type', 'network\_type', 'education\_level', 'location', 'internet\_type', 'it\_student', 'device', 'self\_lms', and 'load\_shedding' are encoded using similar logic based on their respective categories. These encoded values can be used for further analysis or feeding into a machine learning model.

```
# Encode categorical variables
gender = 1 if gender == 'girl' else 0
age_range = age_range.split('-')
age = (int(age_range[0]) + int(age_range[1])) / 2
class_duration_range = class_duration_range.split('-')
class_duration = (int(class_duration_range[0]) + int(class_duration_range[1])) / 2
financial_condition = 2 if financial_condition == 'poor' else (1 if financial_condition == 'mild' else 0)
institute_type = 1 if institute_type == 'Government' else 0
network_type = 2 if network_type == '2G' else (1 if network_type == '3G' else 0)
education_level = 2 if education_level == 'college' else (1 if education_level == 'university' else 0)
location = 1 if location == 'yes' else 0
internet_type = 1 if internet_type == 'Wi-fi' else 0
it_student = 1 if it_student == 'yes' else 0
device = 2 if device == 'tab' else (1 if device == 'computer' else 0)
self_lms = 1 if self_lms == 'yes' else 0
load_shedding = 1 if load_shedding == 'high' else 0
```

The provided code snippet creates an input array called 'input\_data' using the encoded values of the variables. The array is constructed as a numpy array with a single row containing the encoded values for 'gender', 'age', 'class\_duration', 'financial\_condition', 'institute\_type', 'network\_type', 'education\_level', 'location', 'internet\_type', 'it\_student', 'device', 'self\_lms', and 'load\_shedding'. This input array can be used as input for further processing or prediction tasks, such as feeding it into a machine learning model.

```
# Create input array
input_data = np.array([[gender, age, class_duration, financial_condition, institute_type, network_type, ed
                        location, internet_type, it_student, device, self_lms, load_shedding]])
```

The provided code snippet makes a prediction using a pre-trained model on the input data. It uses the model's predict method to predict the adaptivity level based on the input data. The prediction is then mapped to a corresponding adaptivity level category ('High', 'Moderate', or 'Low'). Finally, the result is rendered in the 'result.html' template, where the predicted adaptivity level is displayed.

```
# Make prediction
prediction = model.predict(input_data)[0]

# Map prediction to adaptivity level
adaptivity_level = 'High' if prediction == 2 else ('Moderate' if prediction == 1 else 'Low')

return render_template('result.html', prediction=adaptivity_level)
```

## Main Function:

This code runs the Flask application if the script is being executed directly (i.e. not imported as a module in another script). The `if __name__ == '__main__':` line checks if the script is the main module being executed, and if so, runs the Flask application using the `app.run()` method. This method starts the Flask development server, allowing the application to be accessed via a web browser at the appropriate URL.

```
if __name__ == '__main__':  
    app.run()
```

## Activity 2.3: Run the Web Application

When you run the “app.py” file this window will open in the console or output terminal. Copy the URL given in the form <http://127.0.0.1:5000/> and paste it in the browser.

```
In [23]: runfile('C:/Users/kamya/OneDrive/Desktop/Smart_B/ML/e-  
Adapt_Predicting Student Adaptability in Online Classes/app.py',  
wdir='C:/Users/kamya/OneDrive/Desktop/Smart_B/ML/e-  
Adapt_Predicting Student Adaptability in Online Classes')  
* Serving Flask app "app" (lazy loading)  
* Environment: production  
  WARNING: This is a development server. Do not use it in a  
  production deployment.  
  Use a production WSGI server instead.  
* Debug mode: off  
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

When we paste this URL in browser it will redirect us to home.html page, which is the first page of the Newspaper article- ‘Daily Express’. It also contains a Navigation bar including - ‘Home’, ‘Predict My Adaptivity’, ‘Increase my Adaptivity’, ‘Bollywood Mania’, and ‘Horoscope’.

It contains an article on **Adaptivity in Online Education**.

### Daily Express

HomePredict My AdaptivityIncrease My AdaptivityDaily UpdatesBollywood ManiaHoroscope for Today

#### Adaptivity in Online Education



The world of education has witnessed a significant shift with the rise of online learning. In this digital age, adaptivity has become a crucial aspect of successful online education. The ability to adapt and personalize the learning experience for each student holds the key to unlocking their full potential.

Adaptivity in online education refers to the tailored approach of delivering content, assessments, and learning experiences based on the unique needs and preferences of individual students. By leveraging technology and data-driven insights, educators can create personalized learning paths, ensuring that students receive the most relevant and effective instruction.

Through adaptive learning platforms and intelligent algorithms, educational institutions can analyze students' performance, learning patterns, and preferences to deliver targeted interventions and support. Adaptive systems can dynamically adjust the difficulty level of assignments, provide real-time feedback, and offer additional resources to address individual learning gaps.

This personalized approach to online education has several advantages. It allows students to learn at their own pace, focusing on areas where they need improvement and accelerating through concepts they have mastered. By tailoring the learning experience, students are more engaged, motivated, and likely to achieve better learning outcomes.


Furthermore, adaptivity in online education promotes inclusivity by catering to diverse learning styles, abilities, and backgrounds. It accommodates the unique challenges faced by each student and provides them with equitable opportunities to succeed.

As technology continues to advance, the potential for adaptivity in online education grows exponentially. Artificial intelligence, machine learning, and big data analytics enable educators to harness the power of data and provide increasingly personalized learning experiences. This transformative approach not only enhances student outcomes but also empowers educators to make data-informed decisions and continually improve their teaching strategies.


In conclusion, adaptivity is a cornerstone of effective online education. By embracing personalized learning approaches and leveraging technology, educators can create dynamic and tailored learning experiences that empower students to reach their full potential. The future of education lies in adaptivity, and its impact on shaping the next generation of learners will undoubtedly be profound.

All Rights Reserved @Daily\_EXPress  
Contact us at [www.DailyEXPress.com](http://www.DailyEXPress.com)

When you click on 'Predict My adaptivity' in Navigation bar you'll be redirected to index.html page which contains a few engagement metrics such as - **Gender, Age, Class Duration, Financial Condition, Institution Type, Network Type, Education Level, Location, Internet Type, IT Student, Device, Self LMS Load- Shedding.**



## Daily Express




[Home](#) [Predict My Adaptivity](#) [Increase My Adaptivity](#) [Daily Updates](#) [Bollywood Mania](#) [Horoscope](#)

**Predict your adaptivity level by filling out this simple form:**


<b>Gender:</b>	<input type="text" value="Girl"/>	<b>Location:</b>	<input type="text" value="No"/>
<b>Age:</b>	<input type="text" value="21-25"/>	<b>Internet Type:</b>	<input type="text" value="Mobile Data"/>
<b>Class Duration:</b>	<input type="text" value="3-6"/>	<b>IT Student:</b>	<input type="text" value="Wi-Fi"/>
<b>Financial Condition:</b>	<input type="text" value="Mild"/>	<b>Device:</b>	<input type="text" value="Mobile"/>
<b>Institution Type:</b>	<input type="text" value="Non-Government"/>	<b>Self LMS:</b>	<input type="text" value="Yes"/>
<b>Network Type:</b>	<input type="text" value="3G"/>	<b>Load-shedding:</b>	<input type="text" value="Low"/>
<b>Education Level:</b>	<input type="text" value="University"/>		

**Predict**

Fill in the required details carefully and hit on predict button.



## Daily Express



[Home](#) [Predict My Adaptivity](#) [Increase My Adaptivity](#) [Daily Updates](#) [Bollywood Mania](#) [Horoscope](#)

**Predict your adaptivity level by filling out this simple form:**

<b>Gender:</b>	<input type="text" value="Girl"/>	<b>Location:</b>	<input type="text" value="No"/>
<b>Age:</b>	<input type="text" value="21-25"/>	<b>Internet Type:</b>	<input type="text" value="Wi-Fi"/>
<b>Class Duration:</b>	<input type="text" value="3-6"/>	<b>IT Student:</b>	<input type="text" value="Yes"/>
<b>Financial Condition:</b>	<input type="text" value="Mild"/>	<b>Device:</b>	<input type="text" value="Computer"/>
<b>Institution Type:</b>	<input type="text" value="Non-Government"/>	<b>Self LMS:</b>	<input type="text" value="Yes"/>
<b>Network Type:</b>	<input type="text" value="3G"/>	<b>Load-shedding:</b>	<input type="text" value="High"/>
<b>Education Level:</b>	<input type="text" value="University"/>		

**Predict**

After hitting on predict button you will be redirected to result.html page which contains information about the **Levels of Adaptivity and Measures to Increase it**.

## Daily Express



[Home](#)[Predict My Adaptivity](#)[Increase My Adaptivity](#)[Daily Updates](#)[Bollywood Mania](#)[Horoscope](#)

Adaptivity Level	Measures to Increase Adaptivity
High	Engage in interactive learning activities, participate in group discussions, and seek challenging tasks.
Moderate	Establish a structured study routine, set goals, and seek help when needed.
Low	Identify and address any learning gaps, seek additional resources, and maintain a positive mindset.



When you scroll down you will be able to see your **Predicted Adaptivity** level out of- ‘**High**’, ‘**Moderate**’and ‘**Low**’.

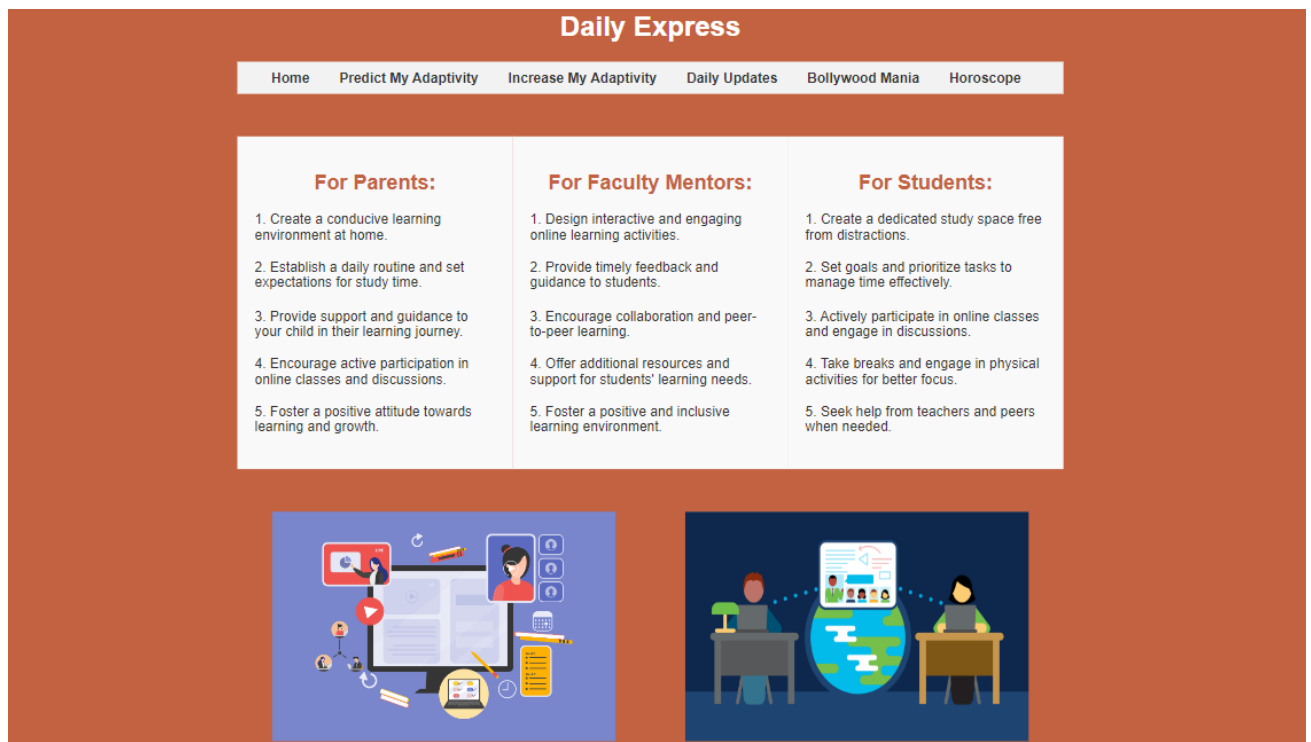
Adaptivity Level	Measures to Increase Adaptivity
High	Engage in interactive learning activities, participate in group discussions, and seek challenging tasks.
Moderate	Establish a structured study routine, set goals, and seek help when needed.
Low	Identify and address any learning gaps, seek additional resources, and maintain a positive mindset.



**Adaptivity Level Prediction Result:**

Adaptivity Level: Low

When you will click on **Increase my Adaptivity** , you will be redirected to adaptivity.html page which contains few points on - ‘**How to increase Adaptivity for- Parents, Faculty Mentors and Students**’.



### **Link to GitHub Repository:**

<https://github.com/Kamya-Paliwal/e-Adapt-Predicting-Student-Adaptivity-in-Online-Classes>