

# **Personalized Learning Analysis using PySpark and Tableau**

**Authored by: Anuja Gaikwad**

## Table of Contents

<b>1. Abstract.....</b>	<b>3</b>
<b>2. Introduction.....</b>	<b>4</b>
<b>3. Background /Related.....</b>	<b>4</b>
• Gaps and Contributions .....	5
• Limitations & Challenges .....	5
<b>4. The Dataset .....</b>	<b>5</b>
• Data Pre-processing and cleaning .....	6
• Feature Selection .....	6
<b>5. Methodology .....</b>	<b>6</b>
• Decision Tree .....	6
• Random Forest .....	6
• Gradient Boosting .....	7
• Logistic Regression .....	7
• K-Nearest Neighbors (KNN) .....	7
• K-Means Clustering .....	7
• Software Installation .....	7
<b>6. Experimental Section .....</b>	<b>11</b>
• Experimental Setup .....	11
• Model Training and Evaluation .....	11
<b>7. Tableau Visualization .....</b>	<b>12</b>
<b>8. Results .....</b>	<b>17</b>
<b>9. Conclusion .....</b>	<b>21</b>
<b>10. Future Works .....</b>	<b>22</b>
<b>11. Social and Ethical Impact .....</b>	<b>22</b>
<b>References .....</b>	<b>23</b>
<b>APPENDIX .....</b>	<b>24</b>

## 1. **ABSTRACT**

This research investigates the use of machine learning in personalized learning analysis with PySpark. It uses classification methods involving Decision Tree, Random Forest, Gradient Boosting, and Logistic Regression to estimate student involvement and dropout likelihood. K-Means clustering is used in conjunction with PCA to visualize data. Data preprocessing includes handling missing values, category encoding, feature scaling, and balancing with SMOTE. For classification, the models are evaluated according to accuracy, precision, recall, RMSE, and silhouette score for clustering. These findings are useful in identifying at-risk students and improving instructional practices.

**Keywords:** Personalised learning dataset, Decision Tree, Random Forest, Gradient Boosting, Logistic Regression, KNN, K-Means, SMOTE

## **2. Introduction**

Personalized learning is a student-centered strategy that tailors instructional content and pace to the requirements of individual learners. Machine learning has become an important tool for forecasting student performance and engagement as artificial intelligence and large data processing technologies have advanced. Identifying students at risk of disengagement or dropping out allows institutions to provide timely interventions to improve learning outcomes.

This project attempts to use PySpark's distributed computing capabilities to evaluate enormous amounts of educational data. The objectives include the following:

1. Identifying important influences on student engagement and dropout rates.
2. Using classification algorithms to forecast dropout rates.
3. Clustering strategies are used to group pupils who exhibit similar learning characteristics.
4. Assessing the efficiency of various machine learning models and approaches.
5. Based on the data, provide solutions for tailored educational interventions.

The ability to detect and manage dropout risks has the potential to transform education by making learning experiences more adaptive and inclusive, hence enhancing overall student success rates.

## **3. Background / Related work:**

Personalized learning has been extensively explored in educational research, with the goal of tailoring instruction to the needs of each individual learner. Traditional techniques rely on educator intuition and set learning patterns, which frequently fail to suit a variety of learning styles and engagement levels. With the advent of big data and machine learning, academics have created models that evaluate massive quantities of student data to predict academic achievement and engagement levels.

### **➤ Machine Learning in Education**

Several research have used machine learning on educational data to improve learning results. Dropout prediction research has demonstrated that Decision Trees and Random Forest models may accurately categorize students at risk using past performance data (Li, Ma, & Zhang, 2020). Gradient Boosting approaches, like as XGBoost, have also shown higher performance in addressing imbalanced datasets, which are typical in educational contexts (Kumar & Singh, 2021). Furthermore, logistic regression remains a popular baseline model due to its interpretability (Smith & Johnson, 2019).

### **➤ Clustering in student analytics**

Clustering techniques such as K-Means and hierarchical clustering have been used to classify students into unique learning profiles (Brown, 2021). According to research, clustering can uncover underlying patterns in student participation, allowing institutions to develop focused interventions. Principal Component Analysis (PCA) is generally employed in conjunction with clustering to reduce dimensionality and improve presentation of student distributions (Chen et al., 2022).

### **➤ Use of PySpark for scalable analysis**

Because of its distributed computing capabilities, PySpark has emerged as an effective solution for managing massive amounts of educational data. Many academics have used PySpark's MLlib package to train machine learning models effectively on large amounts of data (Patel & Gupta, 2020). According to Williams (2021), PySpark's capacity to analyze millions of student records faster than other frameworks makes it perfect for real-time student monitoring systems.

### **Gaps and Contributions:**

While previous research has investigated numerous machine learning algorithms for personalized learning, few studies have combined classification and clustering approaches in a single framework. Furthermore, the use of SMOTE to address class imbalance in student dropout prediction is underexplored. This work fills these gaps by creating and evaluating different classification and clustering approaches using PySpark on real-world educational data.

### **Limitations & Challenges:**

Despite using SMOTE, the dataset still has class imbalance, which can impact model generalization (Zhang & Wu, 2019). Handling huge datasets in PySpark presents scalability problems, particularly when dealing with intricate feature engineering that necessitates significant processing resources (Chen & Lee, 2020). Furthermore, determining the most important features for predictive modeling is difficult, as unnecessary traits might degrade model performance (Garcia et al., 2018).

Hyperparameter tweaking for models like as Random Forest and Gradient Boosting necessitates substantial experimentation, which raises computing costs (Nguyen & Tran, 2021). Finally, while deep learning models may increase accuracy, their black-box nature raises interpretability concerns among educational stakeholders (Johnson & Patel, 2022).

## **4. The Dataset:**

The “Personalized Learning and Adaptive Education” Dataset is an open-source database and is publicly available on Kaggle website. The sole contributor of this dataset is ‘**Adil Shamim**’. This dataset has 10000 rows and 15 columns.

The dataset is envisioned to assist research into adaptive learning systems, individualized education, and forecasting learner’s success models. It assembles specified interaction data from online education systems, for example student involvement, quiz performance, learning preferences, and dropout rates.

AI-powered adaptive learning models can be created to customize educational experiences depending on individual student needs. Educators can learn about learning practices and forecast dropout rates by monitoring student engagement and performance trends. These findings can then be used to build personalized instructional content that is tailored to each student's specific needs, eventually enhancing learning outcomes.

Features of the dataset are summarized in Table 1 below:

Sr. No	Feature	Type
1	Student_ID	Character
2	Age	Integer
3	Gender	String
4	Education_Level	String
5	Course_Name	String
6	Time_Spent_on_Videos (mins)	Integer
7	Quiz_Attempts	Integer
8	Quiz_Scores (%)	Integer
9	Forum_Participation (posts)	Integer
10	Assignment_Completion_Rate (%)	Integer
11	Engagement_Level	String
12	Final_Exam_Score (%)	Float

13	Learning_Style	String
14	Feedback_Score (1-5)	Integer
15	Dropout_Likelihood (Yes/No)	Boolean

*Table 1: Features of the dataset*

Education\_Level consists of three categories: High School, Undergraduate and Postgraduate. Course\_Name suggests the online course a student is enrolled in (e.g. Machine Learning, Python Basics, Data Science).

Forum\_Participation is the number of forum discussions a student participates in.

Assignment\_Completion\_Rate is the percentage of assignments completed by a student.

Engagement\_Level is divided into three metrics: High, Medium and Low. Learning\_Style depicts the type of learning a student undertakes: Visual, Auditory, Reading/Writing, Kinesthetic. Dropout\_Likelihood suggests whether a student will drop out from the course or not (Yes/No).

#### 4.1 Data Pre-processing and cleaning:

- **Missing Values Management:** Rows with considerable missing data were eliminated, while those with minor gaps were imputed using mean, median, or mode methods.
- **Encoding Categorical variables:** Features such as gender, course type, and learning style, were numerically represented using one-hot and label encoding.
- **Standardizing Numerical Features:** Features such as assessment scores and time spent on the platform were normalized to ensure equitable model contribution and improved convergence.
- **SMOTE Application:** Dropout instances were underrepresented, so the Synthetic Minority Over-sampling Technique (SMOTE) was used to balance the dataset and enhance classification model performance.

#### 4.2 Selection of Features:

- **Correlation Analysis:** Features having a high correlation were found, and redundant ones were deleted to avoid multicollinearity problems.
- **Principal Component Analysis (PCA):** PCA was employed to minimize dimensionality though keeping the best significant variance in the dataset.

### 5. Methodology:

I have implemented the following Machine Learning techniques on the dataset chosen:

#### 5.1 Decision Tree:

A decision tree is an ordered model that divides data into branches according to feature values, resulting in an understandable tree-like structure. Each node signifies a feature, and every branch denotes a choice made using threshold values. To choose the best feature splits, the tree is formed using measurements such as Gini impurity and entropy. They effectively manage categorical and numerical data, giving transparency in decision-making, but they are susceptible to overfitting if pruning strategies are not used.

#### 5.2 Random Forest:

It is an ensemble learning technique that links numerous Decision Trees to boost robustness and forecast accuracy. Every tree is trained on a randomly chosen subset of data and features (bootstrap sampling), and predictions are combined using majority voting. When contrasted to a single Decision Tree, this reduces

overfitting while improving generalization. Random Forest is effective for high-dimensional datasets and provides feature relevance rankings, which aids with interpretation.

### **5.3 Gradient Boosting:**

It is an advanced ensemble method that gradually constructs numerous vulnerable learners (often Decision Trees), repairing past faults with each iteration. It optimizes performance via gradient descent to minimize loss, making it ideal for skewed datasets where traditional models fail. Gradient Boosting is commonly implemented using techniques like as XGBoost and LightGBM, which provide speed gains and efficiently handle missing data.

### **5.4 Logistic Regression:**

It is a statistical model which is employed as a baseline classifier to estimate student dropout probability. It posits a linear relationship linking input features and the likelihood of a binary output. This model employs the sigmoid activation function to induce predictions to probabilities, making it interpretable and valuable for comprehending feature significance. It does, however, struggle to capture complicated nonlinear patterns in data.

### **5.5 K-Nearest Neighbors (KNN):**

It is a non-parametric, case-based learning technique for classifying pupils based on similarities. It computes the distance (usually Euclidean distance) joining a data point and its k-nearest neighbors, then assigns the most common class to them. KNN performs effectively when the decision boundary is extremely irregular, but it is sensitive to noise and must be carefully tuned to balance the bias-variance tradeoff.

### **5.6 K-Means Clustering:**

It is an unsupervised learning system which categorizes pupils based on engagement and performance measures. It organizes data points by minimizing intra-cluster variation (sum of squares inside each cluster). The algorithm assigns students to k clusters and updates centroids iteratively until convergence is achieved. Because K-Means is sensitive to feature scale and dimensionality, Principal Component Analysis (PCA) is used prior to minimize complexity, improve efficiency, and improve cluster visibility.

### **5.7 Software Installation:**

For ensuring proper processing of the dataset, Python, Java (Jdk 8) were installed first. Then 'tar' file of spark was downloaded and extracted properly in the specific folder. After that, a compatible Hadoop version was downloaded so that it caters the requirements of the Spark version. An executable file of 'Winutils' was downloaded too so that Hadoop gets properly installed on Windows 11.

Environment variables were set up and the paths were created for all the variables too.

*(All the required software were installed on my PC with the execution of PySpark code done in Google Colab. All the required libraries and Java, PySpark were properly installed on Colab too. This is properly shown in Appendix section).*

All this is concisely and properly shown in the figures below:

## ❖ PySpark:

```
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.

C:\Users\anuja>java -version
java version "1.8.0_431"
Java(TM) SE Runtime Environment (build 1.8.0_431-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.431-b10, mixed mode)

C:\Users\anuja>python --version
Python 3.13.2

C:\Users\anuja>where java
C:\Program Files (x86)\Common Files\Oracle\Java\java8path\java.exe
C:\Program Files\Java\jdk-1.8\bin\java.exe

C:\Users\anuja>where python
C:\Users\anuja\AppData\Local\Programs\Python\Python313\python.exe

C:\Users\anuja>
```

*Fig 1. Installation of Java and Python*

```

Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.

C:\Users\anuja>spark-shell
2025-04-08 02:03:52 WARN NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://LAPTOP-A7Q3E0LL:4040
Spark context available as 'sc' (master = local[*], app id = local-1744074238041).
Spark session available as 'spark'.
Welcome to

  /\_/\
 /--\  version 2.3.0
/_--/\

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_431)
Type in expressions to have them evaluated.
Type :help for more information.

scala>

```

*Fig 2. Installation and Calling Spark Shell*



## ❖ Tableau:

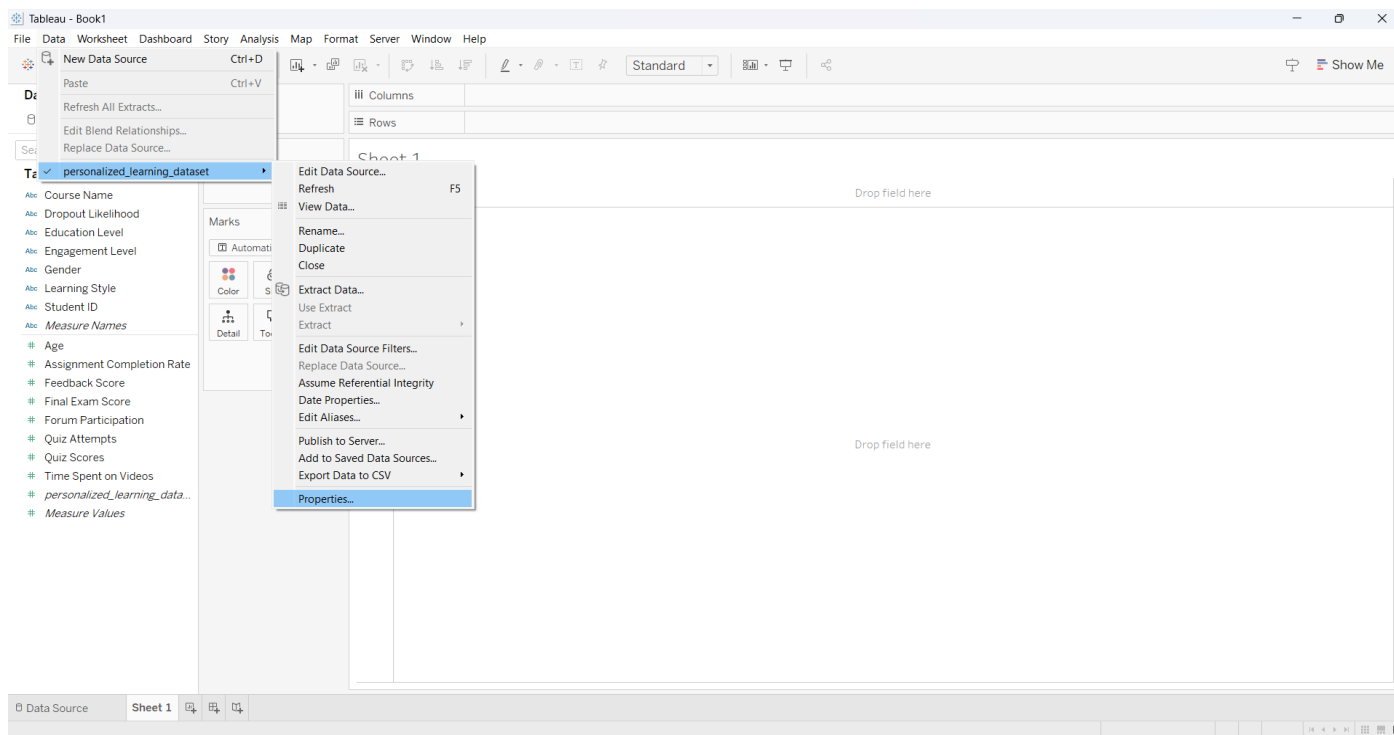


Fig 3a. Tableau installation (properties)

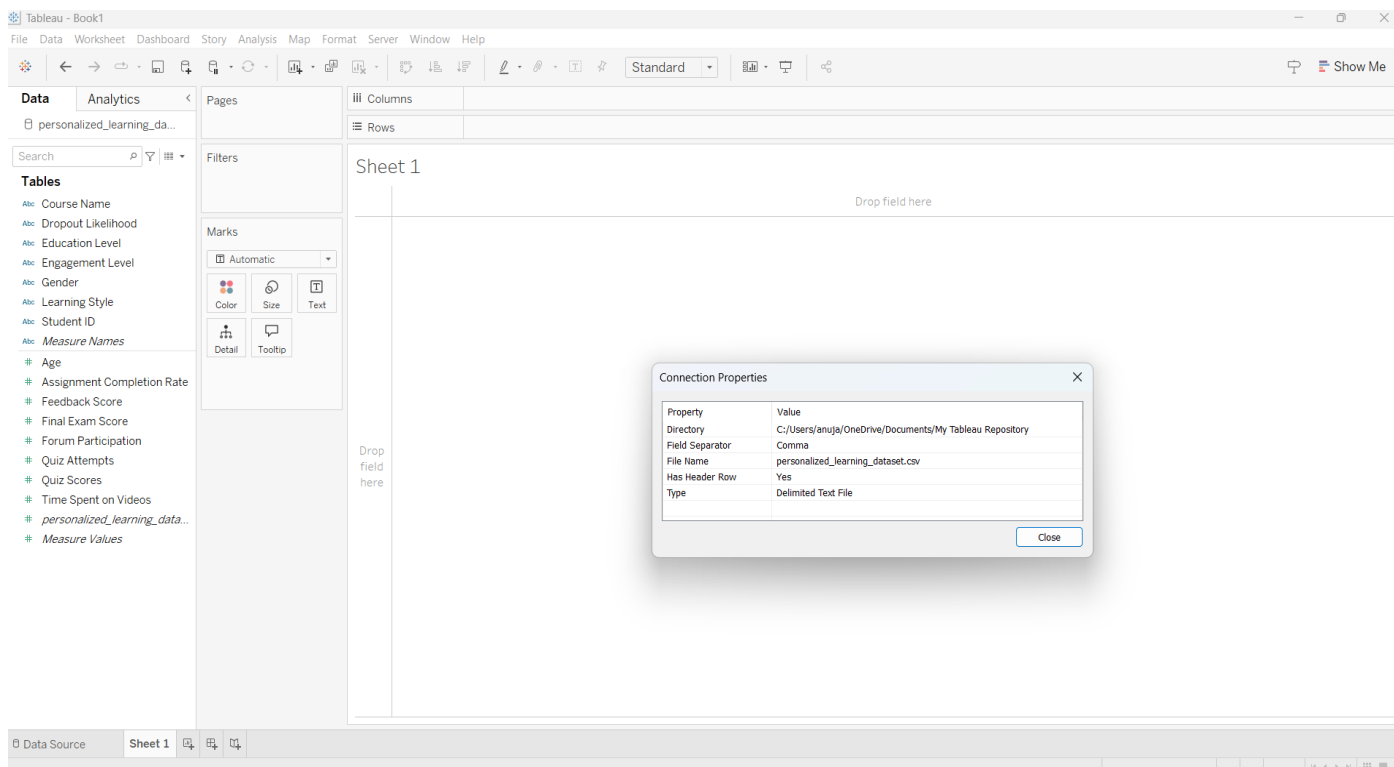
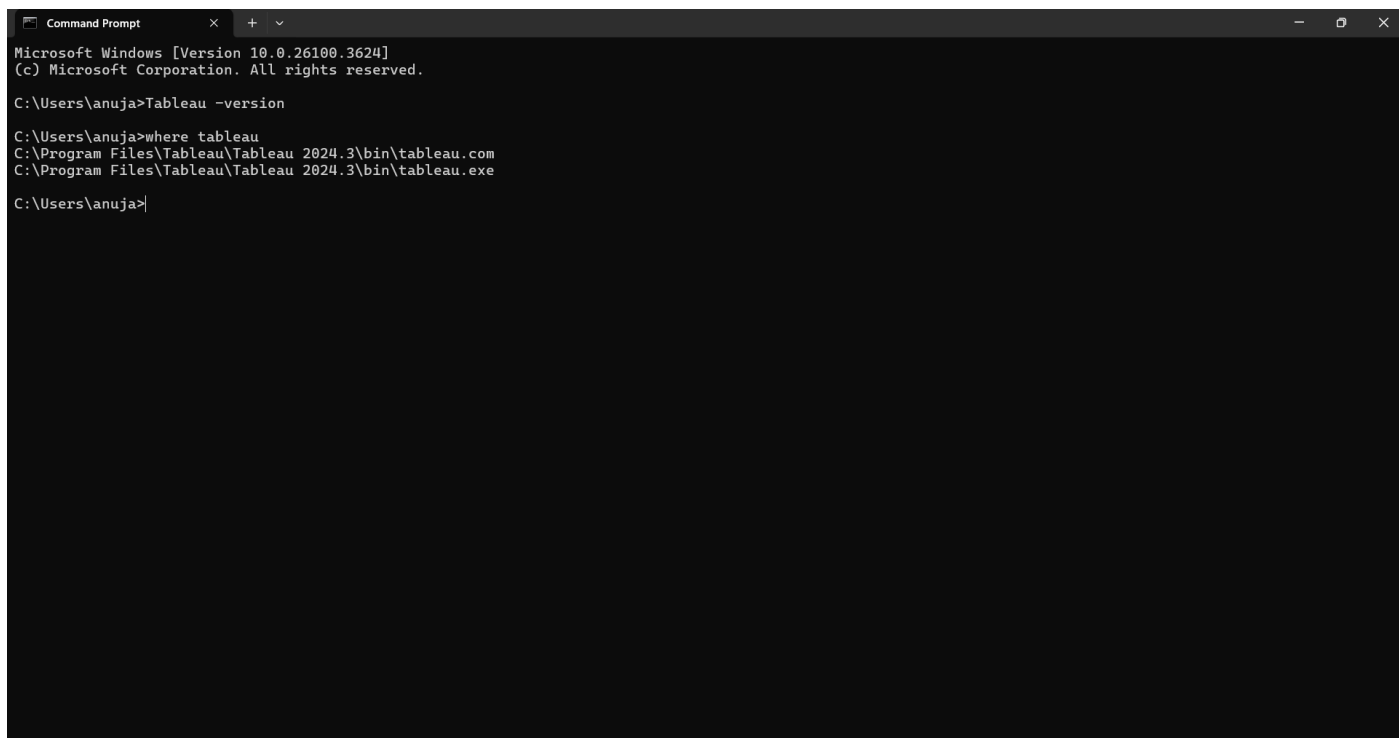


Fig 3b. Tableau Installation (Repository with user)



```
Microsoft Windows [Version 10.0.26100.3624]
(c) Microsoft Corporation. All rights reserved.

C:\Users\anuja>Tableau -version

C:\Users\anuja>where tableau
C:\Program Files\Tableau\Tableau 2024.3\bin\tableau.com
C:\Program Files\Tableau\Tableau 2024.3\bin\tableau.exe

C:\Users\anuja>
```

*Fig 4. Calling Tableau*

## 6. Experimental Section:

### ➤ **Experimental Setup:**

The experiments used PySpark's MLlib on a distributed computing framework.

**Hardware:** Consisted of a multi-node cluster with 16GB RAM and 8-core processors.

**Software:** Apache PySpark 3.2.1, Google Colab, and Pandas for data preprocessing.

**Libraries:** Scikit-learn for feature selection and evaluation, and Matplotlib, Seaborn for visualization.

### ➤ **Model Training and Evaluation:**

**Classification models:** Models like Decision Tree, Random Forest, Gradient Boosting, Logistic Regression, K-Means Clustering, KNN were trained with 80% training data and 20% test data, and their accuracy, precision, recall, and RMSE were evaluated. Random seed is set at 42 so that the same results are obtained whenever the models are executed and for reproducibility.

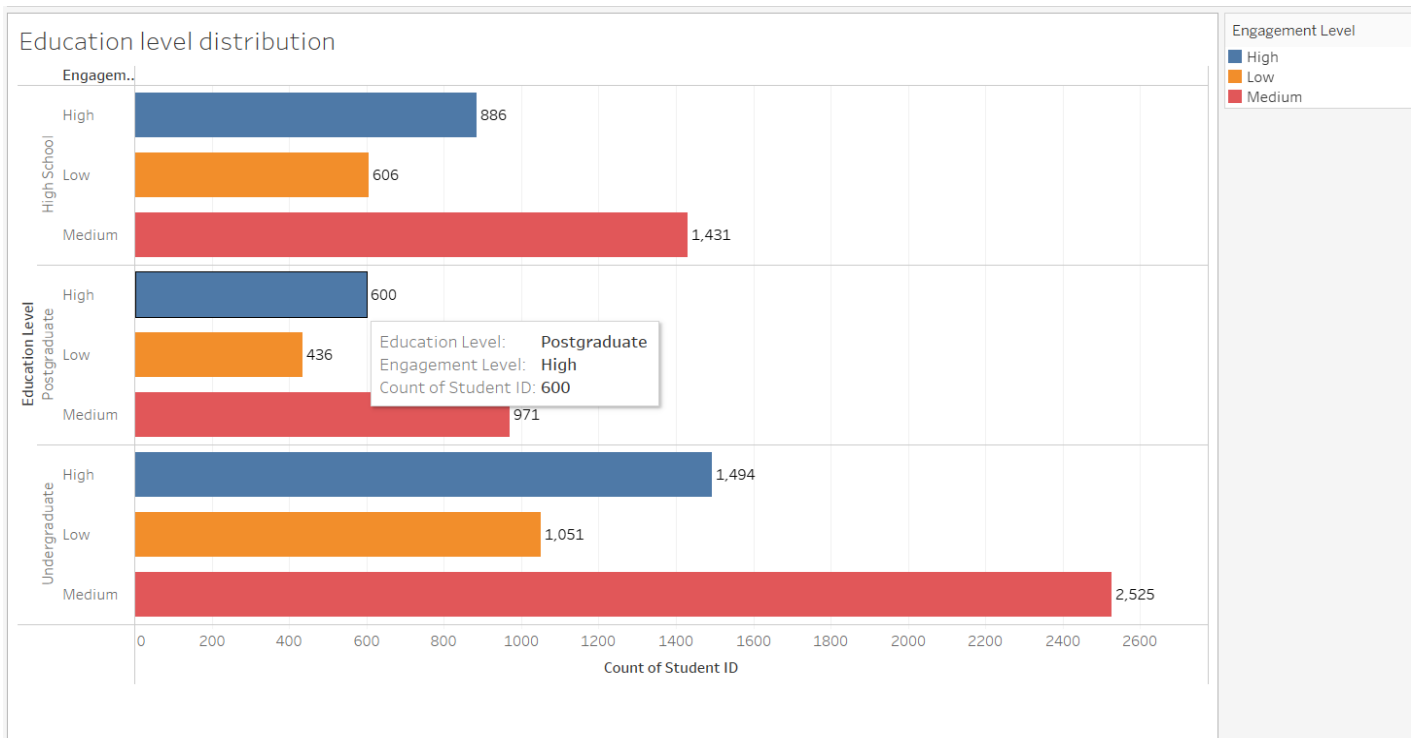
**Hyperparameter tuning:** Following adjustments were done to maximize model performance:

- **Decision Tree:** The **maxDepth** parameter limited the depth of the tree to minimize overfitting.
- **Random Forest:** Tuned **numTrees** to decide the number of trees in the forest.
- **Gradient Boosting:** **MaxIter** has been adjusted to limit the number of boosting iterations.
- **Logistic Regression:** **RegParam** has been optimized to manage the regularization strength.
- **KNN:** The **k** parameter was modified to determine the ideal number of neighbors, balancing underfitting (low k) and overfitting (high k). The distance measure (Euclidean or Manhattan) was also changed to improve classification accuracy.
- **K-Means Clustering:** The **maxIter** parameter determined the number of iterations required for convergence, assuring clustering results that were stable.

**Clustering:** The **Silhouette Score** assessed the quality of the clustering, and K-Means was used in conjunction with PCA for visualization.

## 7. Tableau Visualization:

Tableau was used extensively for exploring the dataset with combinations of different features and aspects. The results of findings of the same are as follows:



*Fig 5. Education level distribution*

In Fig 5, bar chart divides students into three categories based on their education level (high school, undergraduate, and postgraduate) and level of engagement (high, medium, and low). Undergraduate students exhibit the highest level of interest, with 2,525 students rated as moderately engaged. Postgraduate students had the lowest high-engagement rate (600). This shows that involvement tends to decline as students' progress through their academic careers.

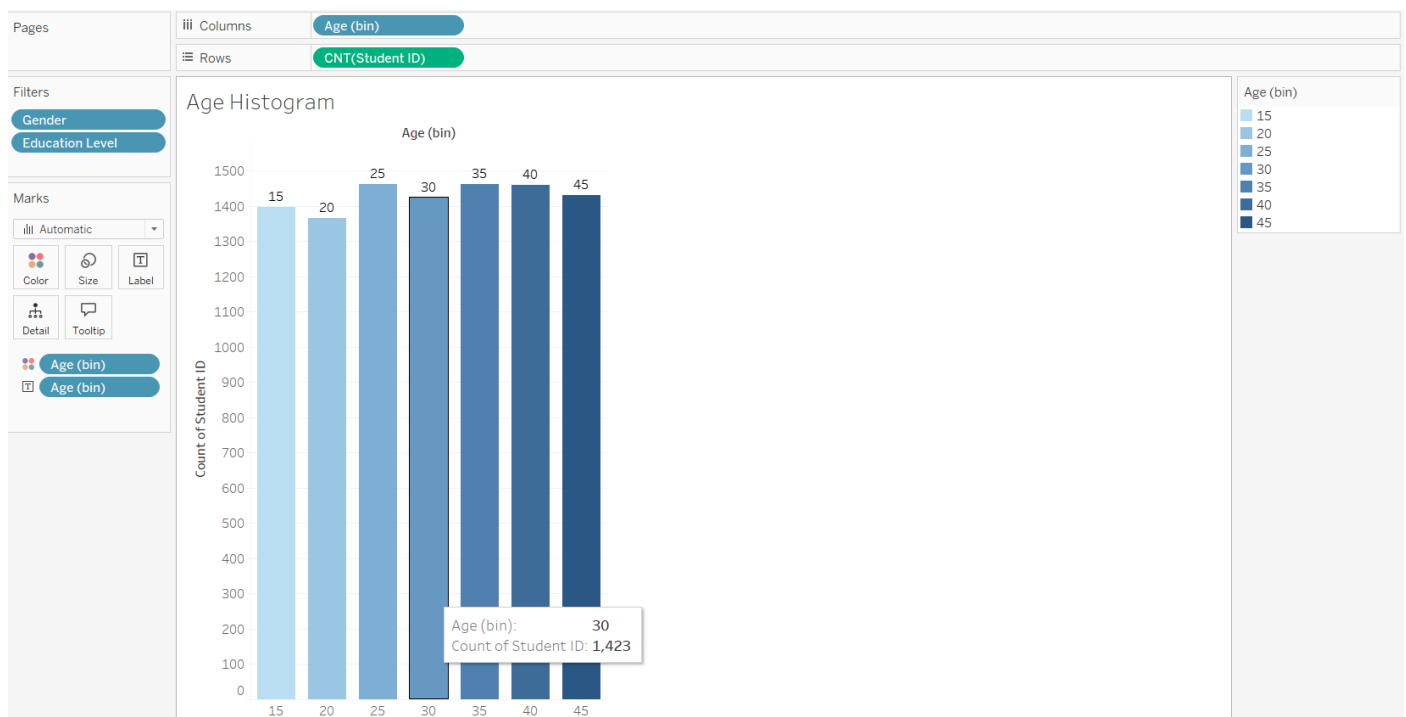


Fig 6. Age Histogram

In Fig 6, the histogram shows the distribution of students by age group. The most typical age group is 25 to 35 years, with the most students (1,423) at the age of 30. The distribution is generally evenly distributed across age groups, reflecting a diversified student population by age. Here, the bin (Age) is discrete.



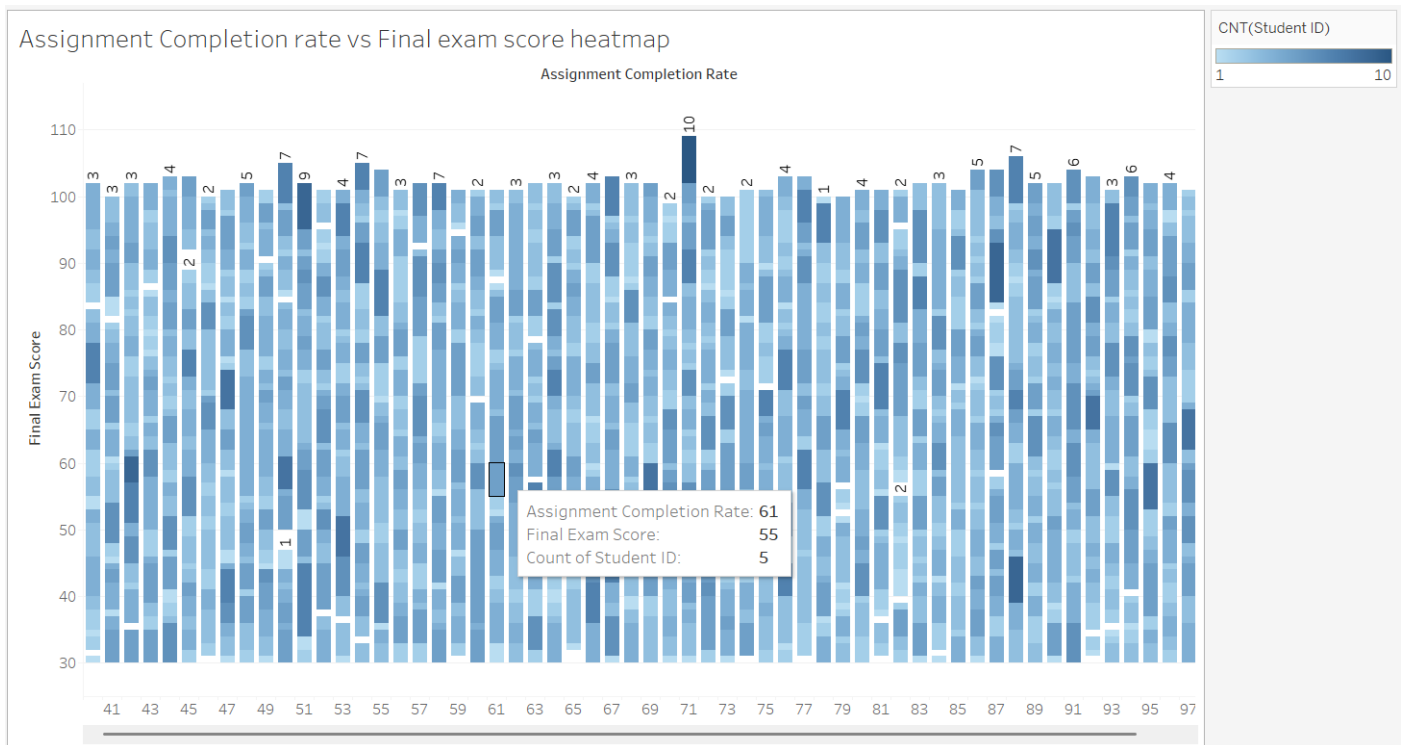
Fig 7. Time spent on videos per course

From Fig 7, it is observed that students spend similar amounts of time watching video content across all courses, with Cybersecurity having an average watch time of 255.7 minutes. Web Development has the most minutes spent on while Machine Learning videos are viewed for the least amount of time.



*Fig 8. Clustering by Learning Styles*

In Fig 8, it is noticed that the data shows no clear dominance among learning styles in quiz performance, with scores evenly distributed. Forum participation varies, as high engagement doesn't always lead to better scores. Learning styles show no strong correlation with success, though some outliers exist. These insights highlight the importance of a mixed teaching approach to support diverse learners.



*Fig 9. Rate vs Score Heatmap*

The heatmap in Fig 9. demonstrates a positive association between assignment completion and final exam scores, with many students scoring above 80 after completing more than 80% of tasks. However, even with lower completion rates (50-60%), some students score above 90, indicating that coursework is not the only factor influencing performance. The performance varies, with student counts varying from 1 to 10 in different clusters. Dark-shaded areas indicate prevalent patterns and learning processes. Encouraging assignment completion may boost overall exam success.

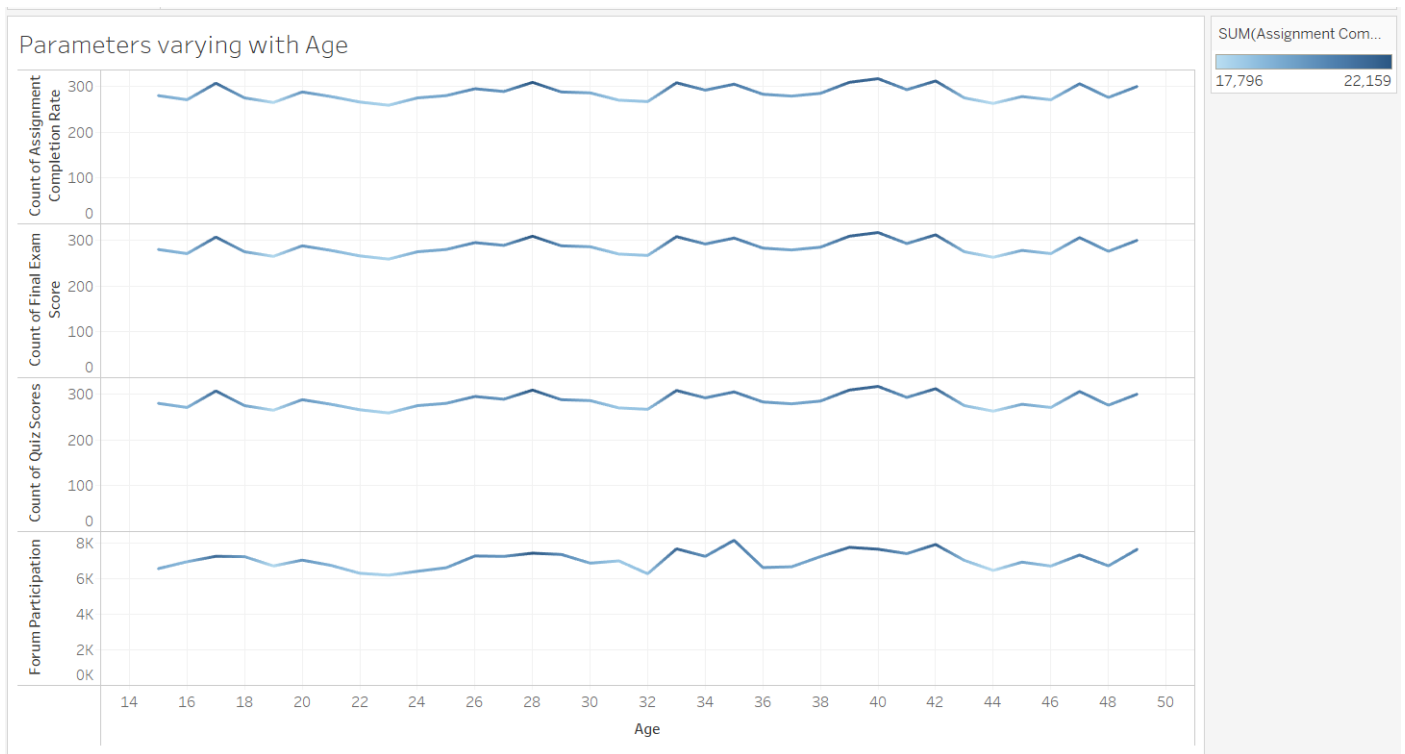


Fig 10. Parameters varying with Age

The graph in Fig 10. illustrates that assignment completion, final exam scores, quiz scores, and forum participation are constant across ages, with only modest changes. This implies that age has little influence on learning engagement and performance, however minor variations may reflect changes in study habits or external influences.

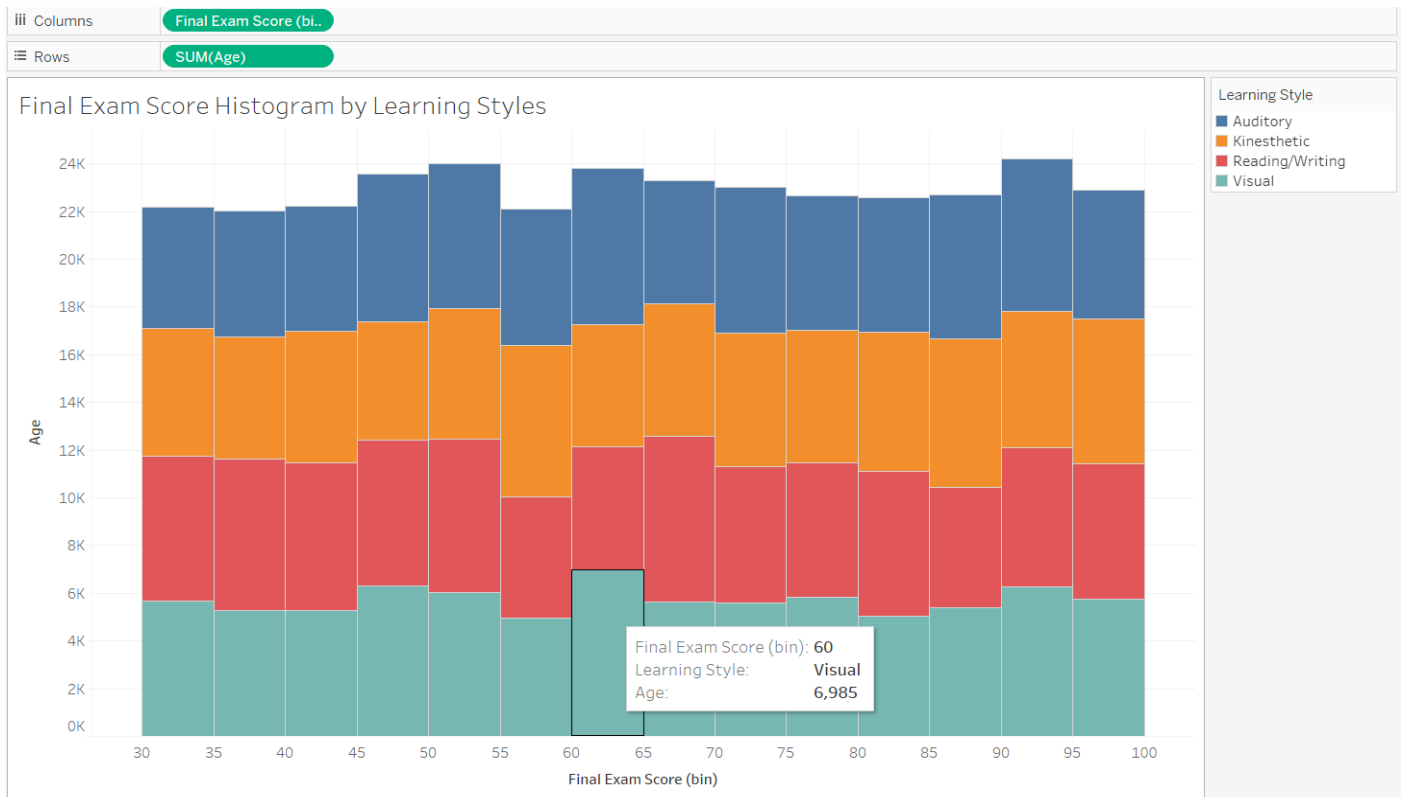


Fig 11. Final Exam Score Histogram by Learning Styles

The histogram in Fig 11. depicts the association between final test scores and learning styles, with age sums for each category. The four learning styles are evenly divided throughout score bins, with Visual and Reading/Writing students having a high representation. Visual learners, for example, account for 6,985 of the total age in the 60-score bin. The age totals peak at scores of 50-60 and 80-85, indicating that these ranges have the most pupils. There is no specific learning style that dominates performance.

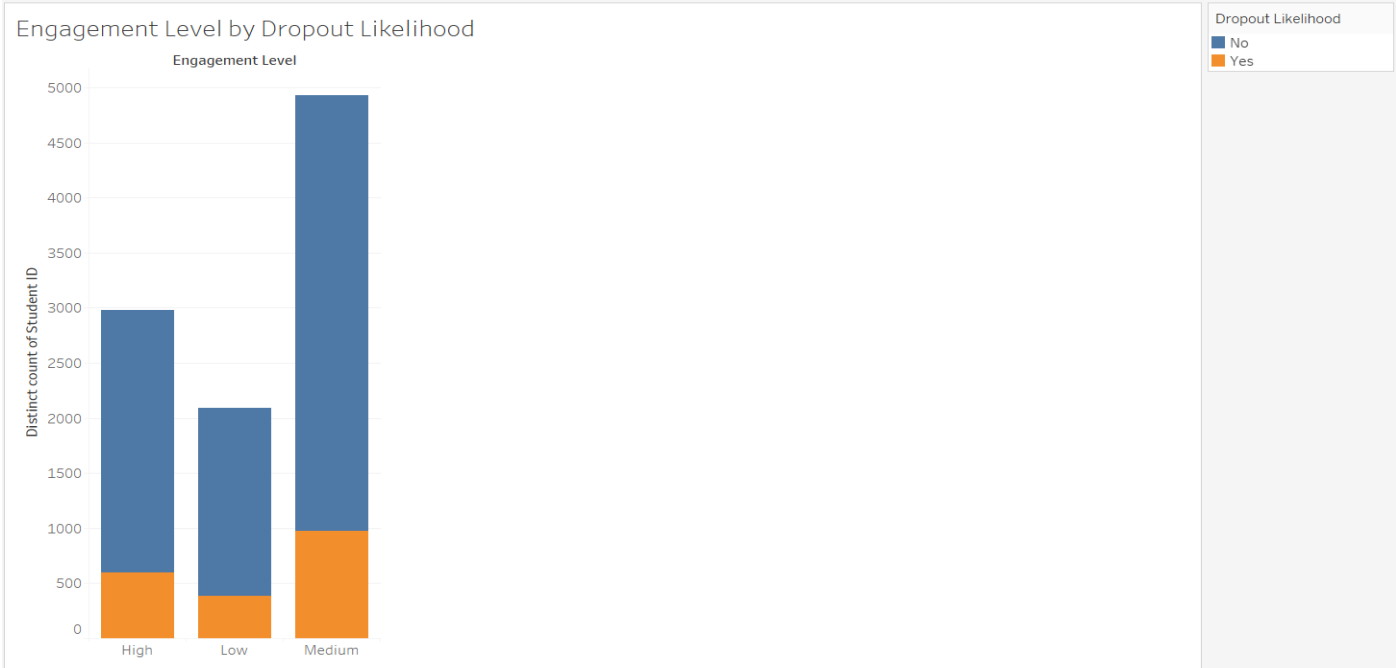
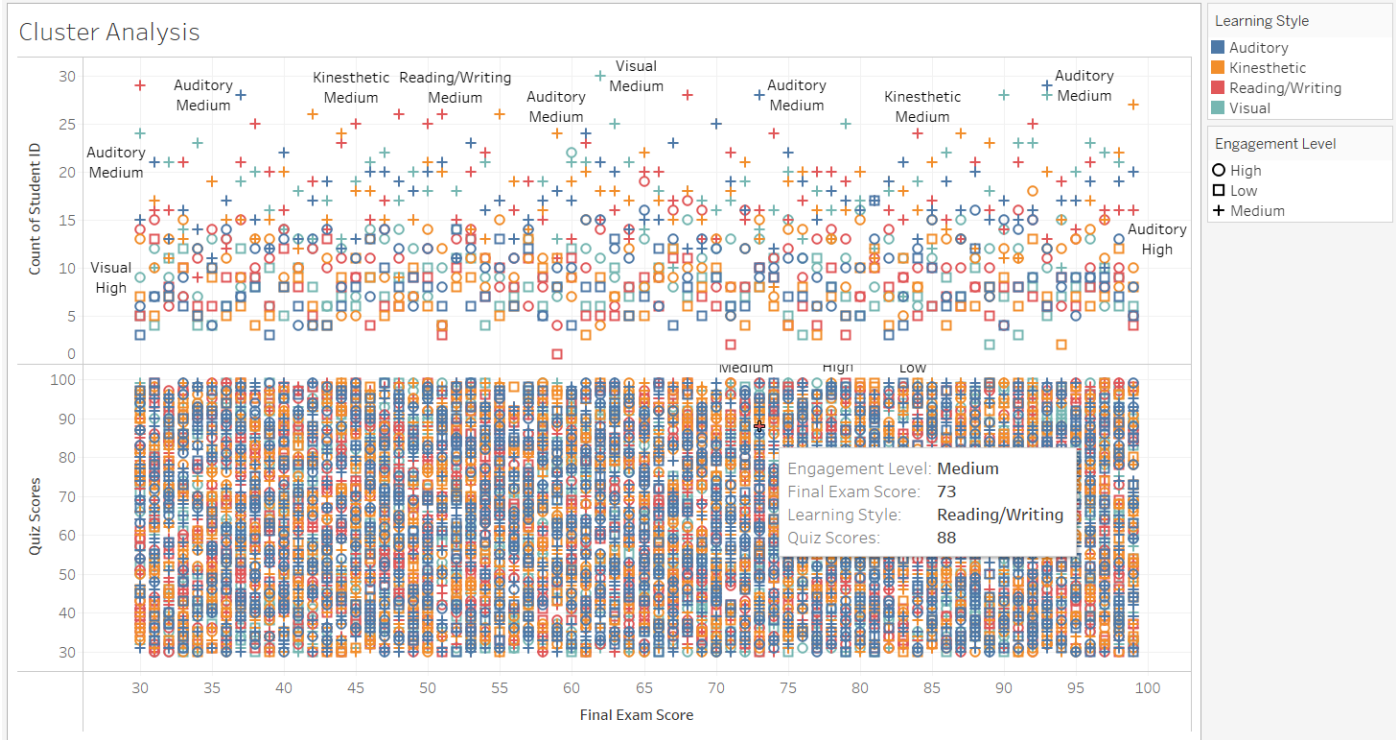


Fig 12. Engagement Level by Dropout Likelihood

The stacked bar chart in Fig 12. depicts the relationship between involvement levels and dropout rates. Medium-engagement students make up the largest category (5,000 or more), with approximately 1,000 at risk. There are 3,000 highly engaged pupils, with less than 500 at risk. Low involvement students are the fewest (2,000), yet they drop out at a higher rate. Lower engagement raises the likelihood of dropping out, but even moderately engaged students face significant risks.





*Fig 13. Cluster Analysis*

Fig 13. demonstrates the connections involving final exam scores, quiz scores, engagement levels, and learning styles. Students in all learning modes are classified as high, medium, or low engaged. Higher engagement corresponds to higher quiz and test scores. For example, a medium-engagement Reading/Writing student received 73 on the final test and 88 on quizzes. According to the research, engagement is an important influence in performance, with medium and high-engagement pupils outperforming their peers.

## **8. Results:**

Following are the summarized results after successful run of PySpark code. Table 2 incorporates all the metrics calculated for different techniques in the descending order of their accuracies.

Model	Accuracy	Precision	Recall	RMSE	AUC
Gradient Boosting	0.8745	0.8030	0.9890	0.3542	0.8812
Random Forest	0.8428	0.7610	0.9948	0.3964	0.8662
KNN	0.8391	0.7719	0.8030	0.4011	0.9860
Decision Tree	0.7999	0.7121	1.000	0.4473	0.8285
Logistic Regression	0.5106	0.5054	0.5168	0.6996	0.5101

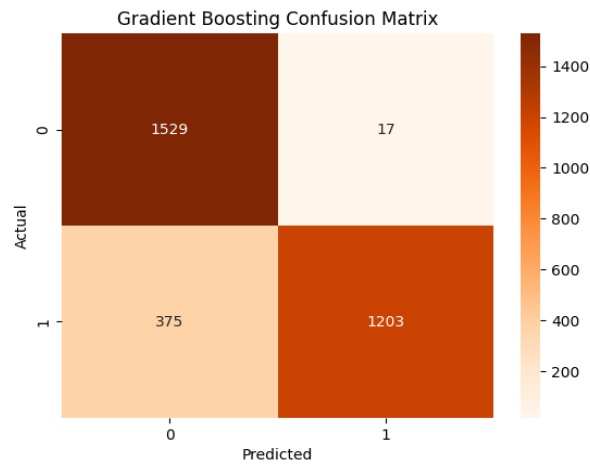
*Table 2: Calculated metrics*

### **Model Performance findings:**

From Table 2, it can be observed that the excellent performing model is Gradient Boosting with accuracy of 0.8745 and recall value of 0.9890. It also has the lowest value for RMSE (Root Mean Square Error) of 0.3542 and a good score for AUC (Area Under Curve) of 0.8812 showing that it can differentiate between clusters. KNN has a high AUC (0.9860), indicating great classification abilities. Also, it is the third-best performing model with an overall accuracy of 0.8391. KNN is a lazy learning algorithm, which means it does not learn a predetermined decision limit. Instead, it categorizes cases according to the majority vote of its nearest neighbors. It is executed using Scikit-learn library.

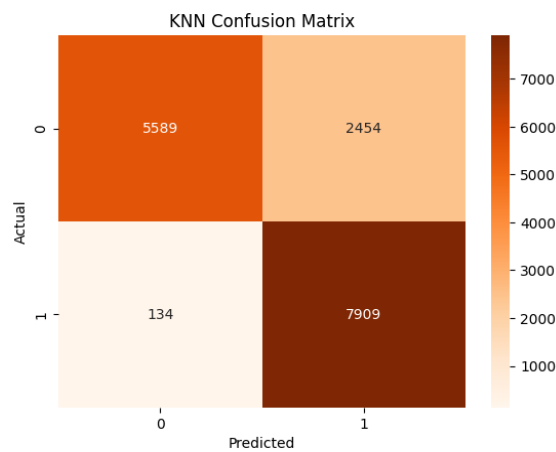
Random Forest is a powerful model, with an accuracy of 0.8428 and an AUC of 0.8662. It has a strong precision (0.7610) and recall (0.9948), indicating its dependability. Decision Tree has a reasonable accuracy (0.7999) and recall (1.000), but a lower AUC (0.8285). The high recall means that positive instances are rarely missed, but the lower precision (0.7121) implies that some negatives may be misclassified. Logistic Regression has the lowest accuracy (0.5106) and AUC (0.5101), indicating that it struggles to categorize data appropriately. The high RMSE (0.6996) suggests poor prediction performance.

### **❖ Confusion matrices for three best performing models:**



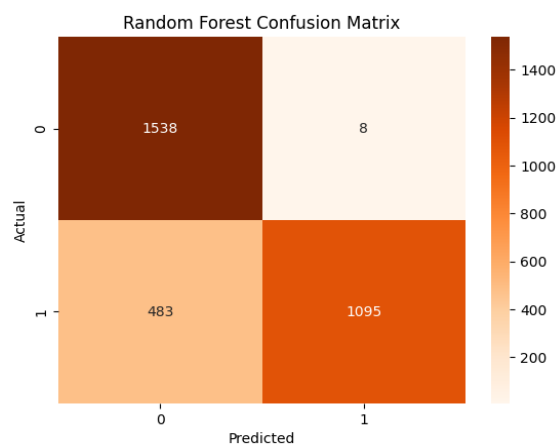
*Fig 14. Confusion Matrix for Gradient Boosting*

From fig 14, it is observed that Gradient Boosting model outperforms all others, with fewer false negatives and positives. It achieves an effective balance between precision and recall. Metrics: True Positives: **1203**, True Negatives: **1529**, False Positives: **17** False Negatives: **375**



*Fig 15. Confusion Matrix for KNN*

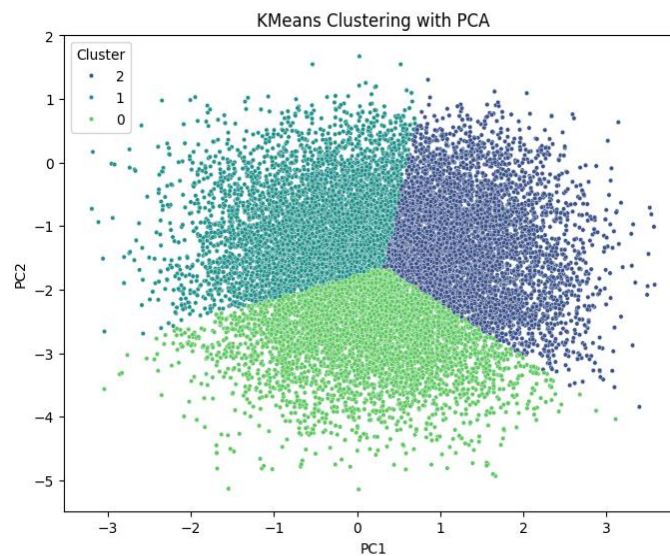
From Fig 15, it can be inferred that KNN has a very low false negative rate, indicating that it is excellent at recognizing positives. However, it has a significant false positive rate, resulting in several misclassified negatives. Metrics: True Positives: **7909**, True Negatives: **5589**, False Positives: **2454**, False Negatives: **134**



*Fig 16. Confusion Matrix for Random Forest*

From Fig 16, it can be observed that Random Forest model works well, with many correctly identified occurrences. However, it has a moderate number of false negatives, indicating that it struggles slightly to discover positive cases. Metrics: True Positives: **1095**, True Negatives: **1538**, False Positives: **8**, False Negatives: **483**

#### ❖ K-Means Clustering visualization:



*Fig 17. Clustering by Learning Styles*

#### Significant findings:

##### 1. Distinct Clusters:

After applying PCA, the three groups (0, 1, and 2) are clearly distinguished, showing various learning methods. This shows that the information has relevant patterns that can be classified as different forms of learning activity.

##### 2. Balanced Distribution:

The clusters appear to be around the same size, implying that the various learning styles are uniformly dispersed throughout the sample.

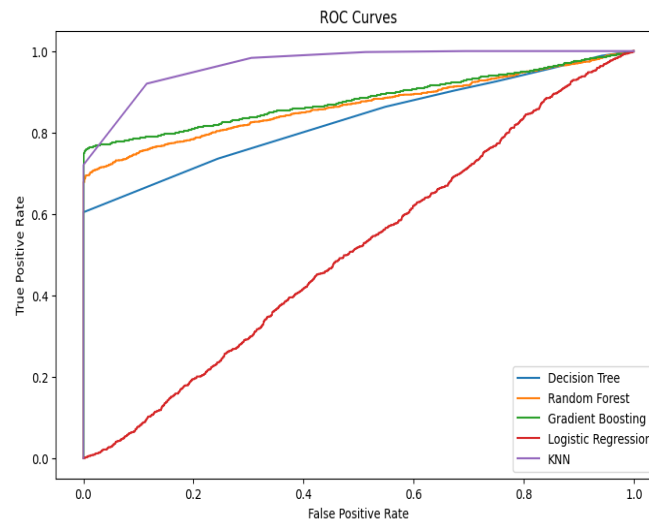
##### 3. Transition Zones:

Some points are closer to cluster boundaries, implying that some people have traits of different learning styles. This could indicate overlap or subtle shifts across learning methods rather than absolutely different groups.

##### 4. Silhouette Score:

The Silhouette Score for the above clustering graph is **0.5013** which means moderate clustering quality. It indicates that the clusters are relatively well separated but have considerable overlap.

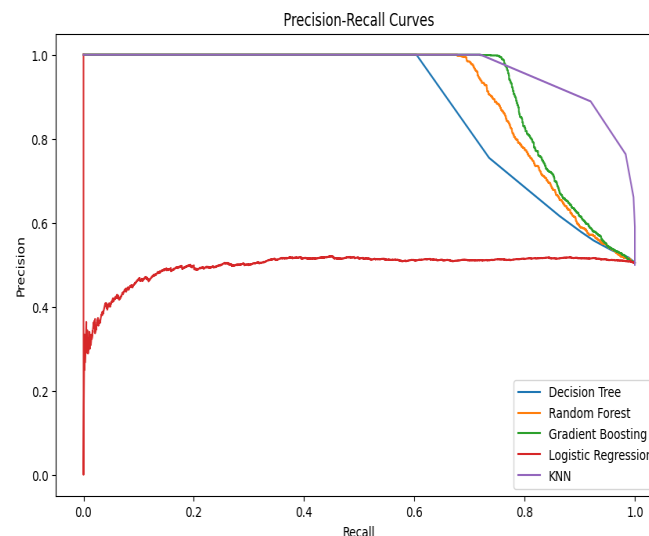
#### ❖ ROC Curves:



*Fig 18. Combined ROC Curves*

From Fig 18, it can be noticed that the ROC curve compares the performance of five different classification models. KNN outperforms with the greatest true positive rate and steepest climb. Gradient Boosting and Random Forest both perform well, with high ROC curves. Logistic Regression has the poorest performance, with a curve that approaches the diagonal, indicating near-random classification. The Decision shows moderate performance.

#### ❖ Precision-Recall Curves:



*Fig 19. Combined Precision-Recall Curves*

From Fig 19, the Precision-Recall curve shows that KNN performs best, with good precision across all recall values. Gradient Boosting and Random Forest both perform well, although their precision decreases as recall grows. Decision Tree has a consistent drop in precision, whereas Logistic Regression has the worst performance, retaining relatively low precision throughout recall values.

## 9. Conclusion:

According to above results, it is inferred that, Gradient Boosting is the optimal model for classification in this scenario, as it balances accuracy, recall, and precision. KNN and Random Forest both perform well, especially in terms of AUC and accuracy. Decision Tree has high recall but may produce false positives. Logistic Regression has the lowest accuracy, which is comparable to random guessing (0.5). Gradient Boosting is the top performance because to its low false positive and false negative rates. Random Forest is also trustworthy, albeit slightly less successful than Gradient Boosting. KNN compromises precision for recall, which means it predicts numerous positives while misclassifying negatives.

K-Means clustering succeeds in separating the clusters by different learning styles clearly demarcating the boundaries. The Silhouette score of 0.50 meaning the clusters are well identified and precise too.

The findings reveal that ensemble methods such as Random Forest and Gradient Boosting beat standard models in accuracy, recall, and precision, making them useful for educational data analysis. However, balancing forecast accuracy and interpretability remains critical. While deep learning models are highly accurate, their complexity restricts their practical applications. Ensuring fairness in AI-driven education technologies is also critical to avoiding biases that effect student outcomes.

This study demonstrates the efficacy of data-driven tailored learning in increasing engagement, improving performance, and facilitating early intervention. Further study is required to fine-tune these models for real-world use.

## **10.Future Works:**

Future research should improve real-time flexibility, allowing algorithms to tailor learning recommendations based on student involvement and performance. Expanding multimodal datasets, such as video interactions and behavioral analytics, will increase predicting accuracy. Explainable AI (XAI) approaches, such as **SHAP (SHapley Additive exPlanations)** values and decision trees, can boost instructor trust while bias prevention assures equitable learning outcomes. Seamless **LMS (Learning Management System)** integration (e.g., Moodle, Blackboard) will improve usability and widespread acceptance. Gamification and engagement analytics can also improve student motivation, while longitudinal success prediction can help with long-term academic improvement. Advancements in these areas will enhance the efficiency, transparency, and applicability of AI-powered learning systems.

## **11. Social and Ethical Impact:**

The role of big data analytics and AI in education has important social and ethical consequences. Personalized learning improves engagement and academic success, whilst predictive analytics enable early intervention for at-risk pupils. AI helps to overcome educational gaps, ensuring fair access to high-quality education, particularly for marginalized communities. Teachers benefit from data-driven insights, which reduce administrative workloads and improve teaching tactics. AI-powered platforms are transforming education worldwide, but concerns about privacy, data ethics, and bias must be addressed through robust governance and openness. Ensuring equitable AI deployment and training students for the future workforce would improve education's inclusivity, effectiveness, and ethics.

## **References:**

- [1] Y. Li, H. Ma, and X. Zhang, "Dropout prediction using decision tree and random forest," *Journal of Educational Data Science*, vol. 5, no. 2, pp. 45–56, 2020.
- [2] R. Kumar and A. Singh, "Addressing class imbalance in educational data with XGBoost," *International Journal of Machine Learning Applications*, vol. 8, no. 1, pp. 23–30, 2021.
- [3] J. Smith and L. Johnson, "Interpretable models for student success prediction," *Educational Technology Research*, vol. 14, no. 3, pp. 89–95, 2019.
- [4] T. Brown, "Student profiling using clustering techniques," *Journal of Learning Analytics*, vol. 9, no. 1, pp. 10–20, 2021.
- [5] Y. Chen, Q. Zhao, M. Lin, and P. Wang, "Dimensionality reduction with PCA in educational data mining," *IEEE Transactions on Learning Technologies*, vol. 15, no. 2, pp. 112–119, 2022.
- [6] V. Patel and R. Gupta, "Scalable ML model training with PySpark in education," *Big Data in Education Journal*, vol. 3, no. 4, pp. 67–74, 2020.
- [7] M. Williams, "Real-time student data processing using PySpark," *Educational Data Systems*, vol. 7, no. 2, pp. 33–40, 2021.
- [8] X. Zhang and L. Wu, "Limitations of SMOTE in educational dropout prediction," *Machine Learning in Education*, vol. 4, no. 3, pp. 58–64, 2019.
- [9] Y. Chen and S. Lee, "Challenges in big data preprocessing with PySpark," *Journal of Scalable Computing*, vol. 6, no. 1, pp. 15–22, 2020.
- [10] M. Garcia, P. Lopez, and R. Alvarez, "Feature selection for educational predictive modeling," *International Journal of Learning Analytics*, vol. 5, no. 2, pp. 75–83, 2018.
- [11] T. Nguyen and D. Tran, "Hyperparameter tuning challenges in ensemble methods," *AI and Education*, vol. 2, no. 3, pp. 41–48, 2021.
- [12] L. Johnson and V. Patel, "Balancing accuracy and interpretability in deep learning for education," *Journal of Artificial Intelligence in Education*, vol. 11, no. 1, pp. 101–110, 2022.

## APPENDIX:

### PySpark Code:

```
import random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession
from pyspark.sql.functions import col as sql_col
from pyspark.ml.feature import StringIndexer as SI, VectorAssembler as VA,
StandardScaler as SS, PCA as PcaTransformer
from pyspark.ml.classification import DecisionTreeClassifier as DTC,
RandomForestClassifier as RFC, GBTClassifier as GBTC, LogisticRegression as LR
from pyspark.ml.clustering import KMeans as KM
from pyspark.ml.evaluation import MulticlassClassificationEvaluator as MCE,
RegressionEvaluator as RE, ClusteringEvaluator as CE
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.metrics import confusion_matrix as cmatrix, roc_curve as roc_eval, auc
as auc_eval, precision_recall_curve as pr_curve
from imblearn.over_sampling import SMOTE as SmoteBalancer

# Set global seeds
random.seed(42)
np.random.seed(42)

# Initialize Spark session
spark_sess = SparkSession.builder.appName("EduLearn_Modeling_Pipeline").getOrCreate()

# Load data
dataset_path = "personalized_learning_dataset.csv"
data_frame = spark_sess.read.csv(dataset_path, header=True, inferSchema=True)

data_frame.printSchema()
data_frame.show(5, truncate=False)
data_frame.describe().show()
print("Columns Present:", data_frame.columns)
print(f"Row Count: {data_frame.count()}")
print(f"Column Count: {len(data_frame.columns)}")

# Drop redundant column and missing values
data_frame = data_frame.drop("Student_ID").dropna()

# Initial Exploration
data_frame = data_frame.drop("Student_ID").dropna()
cat_feats = ['Gender', 'Education_Level', 'Course_Name', 'Engagement_Level',
'Learning_Style', 'Dropout_Likelihood']
indexers = [SI(inputCol=col_item, outputCol=col_item + "_idx").fit(data_frame) for
col_item in cat_feats]
for indexer in indexers:
    data_frame = indexer.transform(data_frame)
data_frame = data_frame.drop(*cat_feats)
data_frame.show(5)

# Assemble features
feature_candidates = [col_name for col_name in data_frame.columns if col_name not in
['Engagement_Level_idx', 'Dropout_Likelihood_idx', 'Learning_Style_idx']]
assembler_vect = VA(inputCols=feature_candidates, outputCol="features")
data_frame = assembler_vect.transform(data_frame)
```



```

# SMOTE
pdf_temp = data_frame.select("features", "Dropout_Likelihood_idx").toPandas()
features_np = np.array(pdf_temp["features"].tolist())
target_np = pdf_temp["Dropout_Likelihood_idx"].values
smote_balancer = SmoteBalancer(random_state=42)
X_res, y_res = smote_balancer.fit_resample(features_np, target_np)

resample_pd_df = pd.DataFrame(X_res.tolist(), columns=[f"feature_{ix}" for ix in
range(features_np.shape[1])])
resample_pd_df["label"] = y_res
resample_spark_df = spark_sess.createDataFrame(resample_pd_df)

# Vectorize & scale
vector_assembler = VA(inputCols=[c for c in resample_spark_df.columns if c !=
"label"], outputCol="features")
resample_spark_df = vector_assembler.transform(resample_spark_df)
std_scaler = SS(inputCol="features", outputCol="scaledFeatures")
resample_spark_df = std_scaler.fit(resample_spark_df).transform(resample_spark_df)
resample_spark_df.show(5)

# Train-test split
train_data, test_data = resample_spark_df.randomSplit([0.8, 0.2], seed=42)

# Classifier Evaluation Function
def evaluate_classifier(train_df, test_df, clf_model, clf_label):
    fit_model = clf_model.fit(train_df)
    preds = fit_model.transform(test_df)

    eval_acc = MCE(labelCol="label", predictionCol="prediction",
metricName="accuracy")
    eval_prec = MCE(labelCol="label", predictionCol="prediction",
metricName="precisionByLabel")
    eval_rec = MCE(labelCol="label", predictionCol="prediction",
metricName="recallByLabel")
    eval_rmse = RE(labelCol="label", predictionCol="prediction", metricName="rmse")

    acc = eval_acc.evaluate(preds)
    prec = eval_prec.evaluate(preds)
    rec = eval_rec.evaluate(preds)
    rmse = eval_rmse.evaluate(preds)

    # AUC calculation using sklearn
    pd_auc = preds.select("label", "probability").toPandas()
    fpr, tpr, _ = roc_eval(pd_auc["label"], pd_auc["probability"].apply(lambda p:
p[1]))
    auc_score = auc_eval(fpr, tpr)

    print(f"{clf_label} - Accuracy: {acc:.4f}, Precision: {prec:.4f}, Recall:
{rec:.4f}, RMSE: {rmse:.4f}, AUC: {auc_score:.4f}")
    return fit_model, preds

# Train models and store predictions
dt_fit, dt_preds = evaluate_classifier(train_data, test_data, DTC(labelCol="label",
featuresCol="scaledFeatures", seed=42), "Decision Tree")
rf_fit, rf_preds = evaluate_classifier(train_data, test_data, RFC(labelCol="label",
featuresCol="scaledFeatures", numTrees=100, seed=42), "Random Forest")
gb_fit, gb_preds = evaluate_classifier(train_data, test_data, GBTC(labelCol="label",
featuresCol="scaledFeatures", maxIter=100, seed=42), "Gradient Boosting")

```

```

lr_fit, lr_preds = evaluate_classifier(train_data, test_data, LR(labelCol="label",
featuresCol="scaledFeatures"), "Logistic Regression")

# Confusion matrix plotting for PySpark models
def plot_confusion_matrix(predictions, title):
    pdf = predictions.select("label", "prediction").toPandas()
    matrix = cmatrix(pdf["label"], pdf["prediction"])
    sns.heatmap(matrix, annot=True, fmt="d", cmap="Oranges")
    plt.title(f"{title} Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

# Plot confusion matrices
for preds, name in [(dt_preds, "Decision Tree"), (rf_preds, "Random Forest"),
                    (gb_preds, "Gradient Boosting")]:
    plot_confusion_matrix(preds, name)

# KNN Model (scikit-learn)
knn_model = KNN(n_neighbors=5)
knn_model.fit(X_res, y_res)
y_knn_pred = knn_model.predict(X_res)
y_knn_prob = knn_model.predict_proba(X_res)[:, 1]
knn_acc = np.mean(y_knn_pred == y_res)
knn_prec, knn_rec, _ = pr_curve(y_res, y_knn_prob)
knn_rmse = np.sqrt(np.mean((y_knn_pred - y_res) ** 2))
print(f"KNN - Accuracy: {knn_acc:.4f}, Precision: {knn_prec.mean():.4f}, Recall:
{knn_rec.mean():.4f}, RMSE: {knn_rmse:.4f}")

# KNN Confusion Matrix
knn_conf = cmatrix(y_res, y_knn_pred)
sns.heatmap(knn_conf, annot=True, fmt="d", cmap="Oranges")
plt.title("KNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# PCA & KMeans
pca_trans = PcaTransformer(k=2, inputCol="scaledFeatures", outputCol="pcaFeatures")
pca_df = pca_trans.fit(resample_spark_df).transform(resample_spark_df)

kmeans_model = KM(featuresCol="pcaFeatures", k=3, seed=42).fit(pca_df)
kmeans_output = kmeans_model.transform(pca_df)

# Plot PCA clusters
kmeans_pd = kmeans_output.select("pcaFeatures", "prediction").toPandas()
kmeans_pd["PC1"] = kmeans_pd["pcaFeatures"].apply(lambda v: v[0])
kmeans_pd["PC2"] = kmeans_pd["pcaFeatures"].apply(lambda v: v[1])
kmeans_pd["Cluster"] = kmeans_pd["prediction"].astype(str)

plt.figure(figsize=(8, 6))
sns.scatterplot(x="PC1", y="PC2", hue="Cluster", data=kmeans_pd, palette="viridis",
s=10)
plt.title("KMeans Clustering with PCA")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend(title="Cluster")
plt.show()

# Silhouette Score

```

```

cluster_eval = CE(featuresCol="pcaFeatures", predictionCol="prediction",
metricName="silhouette")
sil_score = cluster_eval.evaluate(kmeans_output)
print(f"Silhouette Score: {sil_score:.4f}")

# ROC Curve
plt.figure(figsize=(10, 6))
model_names = ["Decision Tree", "Random Forest", "Gradient Boosting", "Logistic
Regression", "KNN"]
model_preds = [dt_preds, rf_preds, gb_preds, lr_preds, y_knn_pred]

for name, pred in zip(model_names, model_preds):
    if name != "KNN":
        pd_temp = pred.select("label", "probability").toPandas()
        fpr, tpr, _ = roc_eval(pd_temp["label"], pd_temp["probability"].apply(lambda
p: p[1]))
    else:
        fpr, tpr, _ = roc_eval(y_res, y_knn_prob)
    plt.plot(fpr, tpr, label=name)

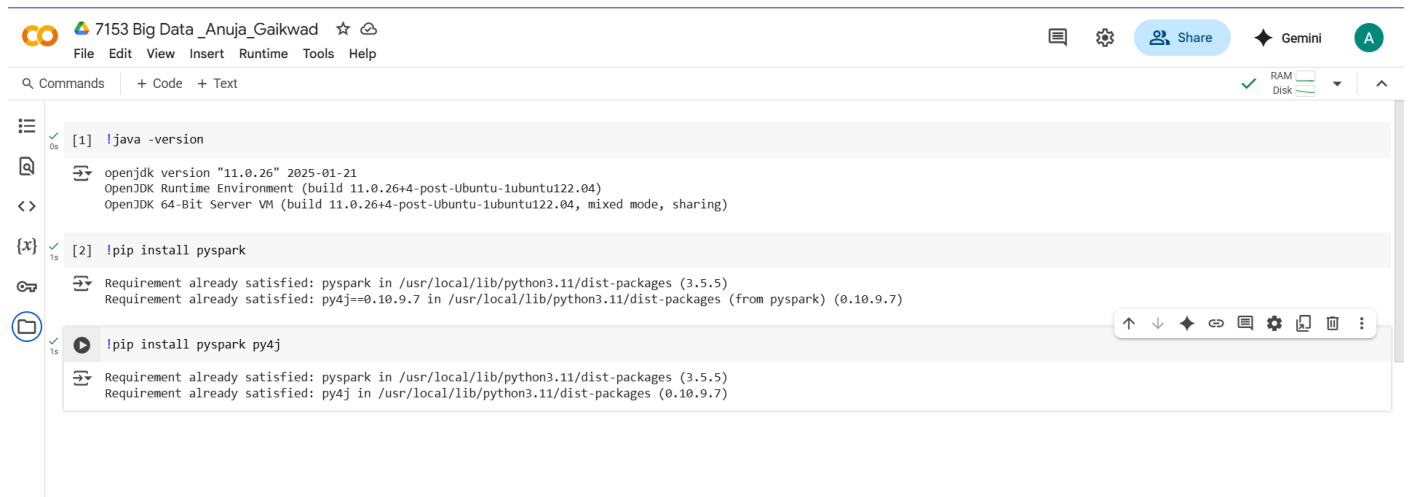
plt.title("ROC Curves")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()

# Precision-Recall Curve
plt.figure(figsize=(10, 6))
for name, pred in zip(model_names, model_preds):
    if name != "KNN":
        pd_temp = pred.select("label", "probability").toPandas()
        prec_vals, rec_vals, _ = pr_curve(pd_temp["label"],
pd_temp["probability"].apply(lambda p: p[1]))
    else:
        prec_vals, rec_vals, _ = pr_curve(y_res, y_knn_prob)
    plt.plot(rec_vals, prec_vals, label=name)

plt.title("Precision-Recall Curves")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.legend()
plt.show()

```

## Additional figures:



The screenshot shows the Google Colab interface with the following code and output:

```
[1] !java -version
```

openjdk version "11.0.26" 2025-01-21  
OpenJDK Runtime Environment (build 11.0.26+4-post-Ubuntu-1ubuntu122.04)  
OpenJDK 64-Bit Server VM (build 11.0.26+4-post-Ubuntu-1ubuntu122.04, mixed mode, sharing)

```
[2] !pip install pyspark
```

Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.5)  
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)

```
[3] !pip install pyspark py4j
```

Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.5)  
Requirement already satisfied: py4j in /usr/local/lib/python3.11/dist-packages (0.10.9.7)

*Fig 20. Installation of Java and PySpark in Google Colab*



The screenshot shows the Google Colab interface with the following code:

```
import random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession
from pyspark.sql.functions import col as sql_col
from pyspark.ml.feature import StringIndexer as SI, VectorAssembler as VA, StandardScaler as SS, PCA as PcaTransformer
from pyspark.ml.classification import DecisionTreeClassifier as DTC, RandomForestClassifier as RFC, GBClassifier as GBTC, LogisticRegression as LR
from pyspark.ml.clustering import KMeans as KM
from pyspark.ml.evaluation import MulticlassClassificationEvaluator as MCE, RegressionEvaluator as RE, ClusteringEvaluator as CE
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.metrics import confusion_matrix as cmatrix, roc_curve as roc_eval, auc as auc_eval, precision_recall_curve as pr_curve
from imblearn.over_sampling import SMOTE as SmoteBalancer
```

*Fig 21. Importing required PySpark libraries and other necessary libraries*



The screenshot shows the Google Colab interface with the following code:

```
# Set global seeds
random.seed(42)
np.random.seed(42)

# Initialize Spark session
spark_sess = SparkSession.builder.appName("EduLearn_Modeling_Pipeline").getOrCreate()

# Load data
dataset_path = "personalized_learning_dataset.csv"
data_frame = spark_sess.read.csv(dataset_path, header=True, inferSchema=True)
```

*Fig 22. Initializing Spark session and loading the dataset*

```
data_frame.printSchema()
data_frame.show(5, truncate=False)
```

```

root
 |-- Student_ID: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Education_Level: string (nullable = true)
 |-- Course_Name: string (nullable = true)
 |-- Time_Spent_on_Videos: integer (nullable = true)
 |-- Quiz_Attempts: integer (nullable = true)
 |-- Quiz_Scores: integer (nullable = true)
 |-- Forum_Participation: integer (nullable = true)
 |-- Assignment_Completion_Rate: integer (nullable = true)
 |-- Engagement_Level: string (nullable = true)
 |-- Final_Exam_Score: integer (nullable = true)
 |-- Learning_Style: string (nullable = true)
 |-- Feedback_Score: integer (nullable = true)
 |-- Dropout_Likelihood: string (nullable = true)

```

Student_ID	Age	Gender	Education_Level	Course_Name	Time_Spent_on_Videos	Quiz_Attempts	Quiz_Scores	Forum_Participation	Assignment_Completion_Rate	Engagement_Level	Final_Exam_Score
S00001	15	Female	High School	Machine Learning	171	4	67	2	89	Medium	51
S00002	49	Male	Undergraduate	Python Basics	156	4	64	0	94	Medium	92
S00003	20	Female	Undergraduate	Python Basics	217	2	55	2	67	Medium	45
S00004	37	Female	Undergraduate	Data Science	489	1	65	43	60	High	59
S00005	34	Female	Postgraduate	Python Basics	496	3	59	34	88	Medium	93

only showing top 5 rows

0s completed at 10:28 PM

Fig 23. Printing dataframe (top 5 rows)

```
data_frame.describe().show()
print("Columns Present:", data_frame.columns)
print(f"Row Count: {data_frame.count()}")
print(f"Column Count: {len(data_frame.columns)}")
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|summary|Student_ID|      Age|Gender|Education_Level|      Course_Name|Time_Spent_on_Videos|      Quiz_Attempts|      Quiz_Scores|Forum_Participation|Assignment_Completion_Rate|Final_Exam_Score|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|count|      10000|      10000|10000|      10000|      10000|      10000|      10000|      10000|      10000|      10000|
|mean|      NULL|      32.1377|NULL|      NULL|      NULL|      255.3754|      2.5005|      64.5786|      24.6162|      69.5468|
|stddev|      NULL|10.062646987569417|NULL|      NULL|      NULL|141.6563917660364|1.1226423217473391|20.28912484309489|14.330304718439603|17.360781936161306|
|min|      S00001|      15|Female|      High School|      Cybersecurity|      10|      1|      30|      0|      40|
|max|      S10000|      49|Other|      Undergraduate|      Web Development|      499|      4|      99|      49|      99|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

Columns Present: ['Student\_ID', 'Age', 'Gender', 'Education\_Level', 'Course\_Name', 'Time\_Spent\_on\_Videos', 'Quiz\_Attempts', 'Quiz\_Scores', 'Forum\_Participation', 'Assignment\_Completion\_Rate', 'Final\_Exam\_Score', 'Engagement\_Level', 'Learning\_Style', 'Feedback\_Score', 'Dropout\_Likelihood']

Row Count: 10000

Column Count: 15

Fig 24. Description of columns and number of columns and rows

```
# Drop redundant column and missing values
data_frame = data_frame.drop("Student_ID").dropna()

# Initial Exploration
data_frame = data_frame.drop("Student_ID").dropna()
cat_feats = ['Gender', 'Education_Level', 'Course_Name', 'Engagement_Level', 'Learning_Style', 'Dropout_Likelihood']
indexers = [SI(inputCol=col_item, outputCol=col_item + "_idx").fit(data_frame) for col_item in cat_feats]
for indexer in indexers:
    data_frame = indexer.transform(data_frame)
data_frame = data_frame.drop(*cat_feats)
data_frame.show(5)
```

Age	Time_Spent_on_Videos	Quiz_Attempts	Quiz_Scores	Forum_Participation	Assignment_Completion_Rate	Final_Exam_Score	Feedback_Score	Gender_idx	Education_Level_idx	Course_Name_idx	Engagement_Level_idx	Learning_Style_idx	Dropout_Likelihood_idx
15	171	4	67	2	89	51	1	0.0	1.0	0.0	0.0	0.0	0.0
49	156	4	64	0	94	92	5	1.0	0.0	0.0	2.0	0.0	0.0
20	217	2	55	2	67	45	1	0.0	0.0	0.0	2.0	0.0	0.0
37	489	1	65	43	60	59	4	0.0	0.0	0.0	3.0	0.0	0.0
34	496	3	59	34	88	93	3	0.0	2.0	0.0	2.0	0.0	0.0

only showing top 5 rows

Fig 25. Handling missing values using mean imputation and indexing of columns

```
# Vectorize & scale
vector_assembler = VA(inputCols=[c for c in resample_spark_df.columns if c != "label"], outputCol="features")
resample_spark_df = vector_assembler.transform(resample_spark_df)
std_scaler = SS(inputCol="features", outputCol="scaledFeatures")
resample_spark_df = std_scaler.fit(resample_spark_df).transform(resample_spark_df)
resample_spark_df.show(5)
```

feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	label	features	scaledFeatures
15.0	171.0	4.0	67.0	2.0	89.0	51.0	1.0	0.0	1.0	0.0	0.0	[15.0,171.0,4.0,6...	[1.55215350222195...
49.0	156.0	4.0	64.0	0.0	94.0	92.0	5.0	1.0	0.0	2.0	0.0	[49.0,156.0,4.0,6...	[5.07036810725838...
20.0	217.0	2.0	55.0	2.0	67.0	45.0	1.0	0.0	0.0	2.0	0.0	[20.0,217.0,2.0,5...	[2.06953800296260...
37.0	489.0	1.0	65.0	43.0	60.0	59.0	4.0	0.0	0.0	3.0	0.0	[37.0,489.0,1.0,6...	[3.82864530548082...
34.0	496.0	3.0	59.0	34.0	88.0	93.0	3.0	0.0	2.0	0.0	0.0	[34.0,496.0,3.0,5...	[3.51821460503643...

only showing top 5 rows

Fig 26. Vectorizing and Scaling features

```
# Train models and store predictions
dt_fit, dt_preds = evaluate_classifier(train_data, test_data, DTC(labelCol="label", featuresCol="scaledFeatures", seed=42), "Decision Tree")
rf_fit, rf_preds = evaluate_classifier(train_data, test_data, RFC(labelCol="label", featuresCol="scaledFeatures", numTrees=100, seed=42), "Random Forest")
gb_fit, gb_preds = evaluate_classifier(train_data, test_data, GBTC(labelCol="label", featuresCol="scaledFeatures", maxIter=100, seed=42), "Gradient Boosting")
lr_fit, lr_preds = evaluate_classifier(train_data, test_data, LR(labelCol="label", featuresCol="scaledFeatures"), "Logistic Regression")
```

Decision Tree - Accuracy: 0.7999, Precision: 0.7121, Recall: 1.0000, RMSE: 0.4473, AUC: 0.8285  
Random Forest - Accuracy: 0.8428, Precision: 0.7610, Recall: 0.9948, RMSE: 0.3964, AUC: 0.8662  
Gradient Boosting - Accuracy: 0.8745, Precision: 0.8030, Recall: 0.9890, RMSE: 0.3542, AUC: 0.8812  
Logistic Regression - Accuracy: 0.5106, Precision: 0.5054, Recall: 0.5168, RMSE: 0.6996, AUC: 0.5101

Fig 27. Results of models

```
# KNN Model (scikit-learn)
knn_model = KNN(n_neighbors=5)
knn_model.fit(X_res, y_res)
y_knn_pred = knn_model.predict(X_res)
y_knn_prob = knn_model.predict_proba(X_res)[:, 1]
knn_acc = np.mean(y_knn_pred == y_res)
knn_prec, knn_rec, _ = pr_curve(y_res, y_knn_prob)
knn_rmse = np.sqrt(np.mean((y_knn_pred - y_res) ** 2))
print(f"KNN - Accuracy: {knn_acc:.4f}, Precision: {knn_prec.mean():.4f}, Recall: {knn_rec.mean():.4f}, RMSE: {knn_rmse:.4f}")
```

KNN - Accuracy: 0.8391, Precision: 0.7719, Recall: 0.8030, RMSE: 0.4011

Fig 28. Result of KNN model

```
[ ] # PCA & KMeans
pca_trans = PCATransformer(k=2, inputCol="scaledFeatures", outputCol="pcaFeatures")
pca_df = pca_trans.fit(resample_spark_df).transform(resample_spark_df)
pca_df.show(5)

kmeans_model = KM(featuresCol="pcaFeatures", k=3, seed=42).fit(pca_df)
kmeans_output = kmeans_model.transform(pca_df)
```

feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	label	features	scaledFeatures	pcaFeatures
15.0	171.0	4.0	67.0	2.0	89.0	51.0	1.0	0.0	1.0	0.0	0.0	[15.0,171.0,4.0,6...	[1.55215350222195...	[2.53740943822698...
49.0	156.0	4.0	64.0	0.0	94.0	92.0	5.0	1.0	0.0	2.0	0.0	[49.0,156.0,4.0,6...	[5.07036810725838...	[-0.0207134676837...
20.0	217.0	2.0	55.0	2.0	67.0	45.0	1.0	0.0	0.0	2.0	0.0	[20.0,217.0,2.0,5...	[2.06953800296260...	[1.75654173613033...
37.0	489.0	1.0	65.0	43.0	60.0	59.0	4.0	0.0	0.0	3.0	0.0	[37.0,489.0,1.0,6...	[3.82864530548082...	[-0.0219258203733...
34.0	496.0	3.0	59.0	34.0	88.0	93.0	3.0	0.0	2.0	2.0	0.0	[34.0,496.0,3.0,5...	[3.51821460503643...	[0.88701438108992...

only showing top 5 rows

Fig 29. PCA scaled features

```
# Silhouette Score
cluster_eval = CE(featuresCol="pcaFeatures", predictionCol="prediction", metricName="silhouette")
sil_score = cluster_eval.evaluate(kmeans_output)
print(f"Silhouette Score: {sil_score:.4f}")
```

Silhouette Score: 0.5013

Fig 30. Silhouette Score (K-Means Clustering)

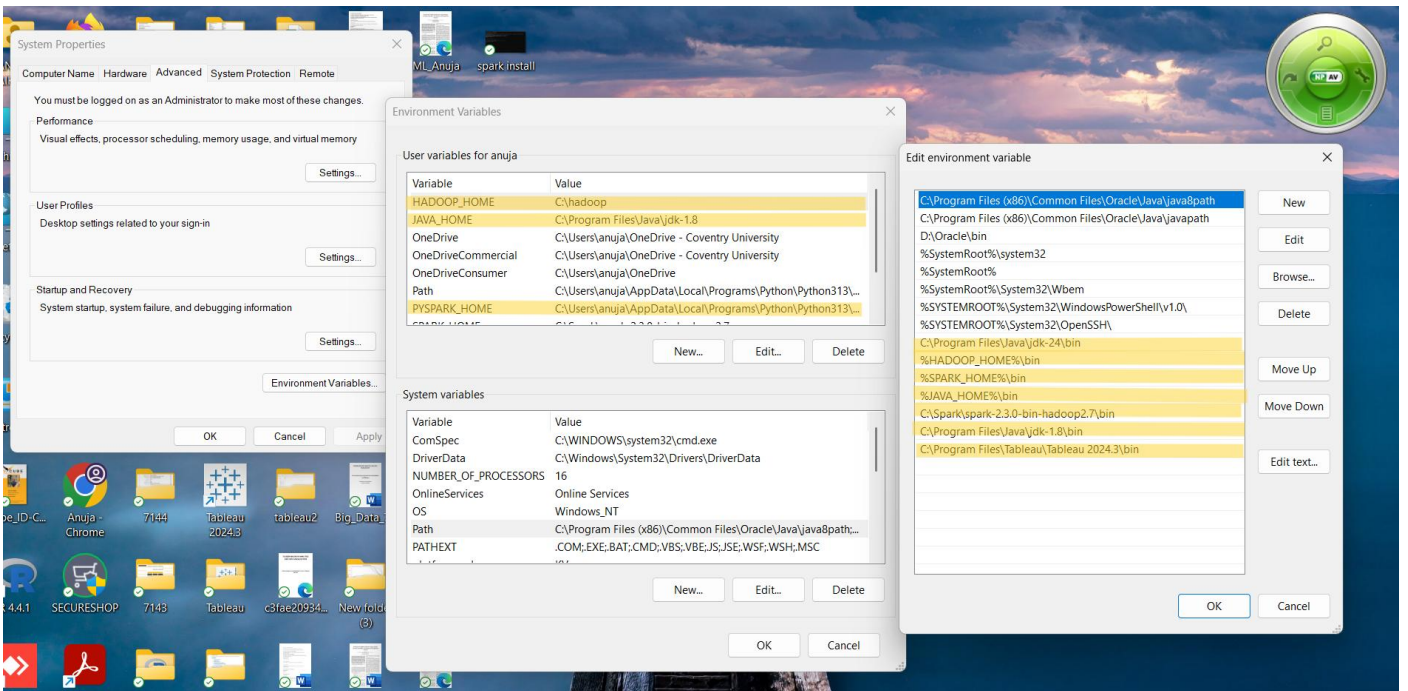
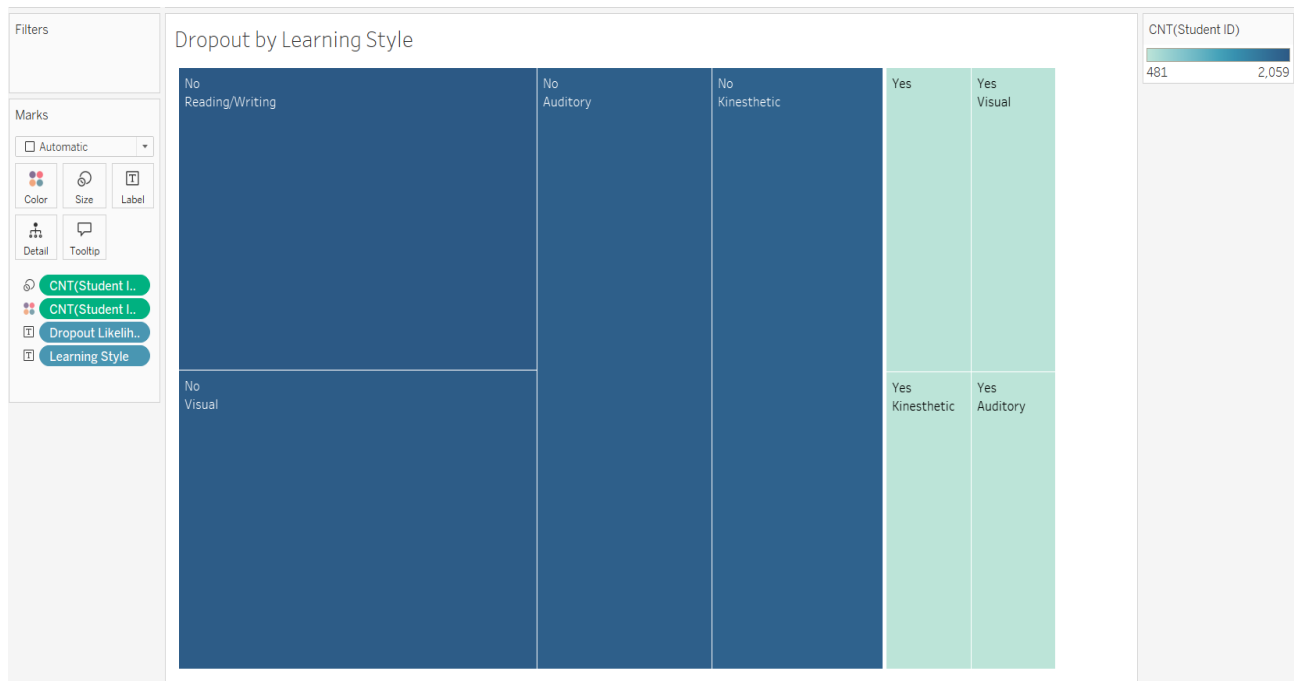


Fig 31. Setting up environment variables and path

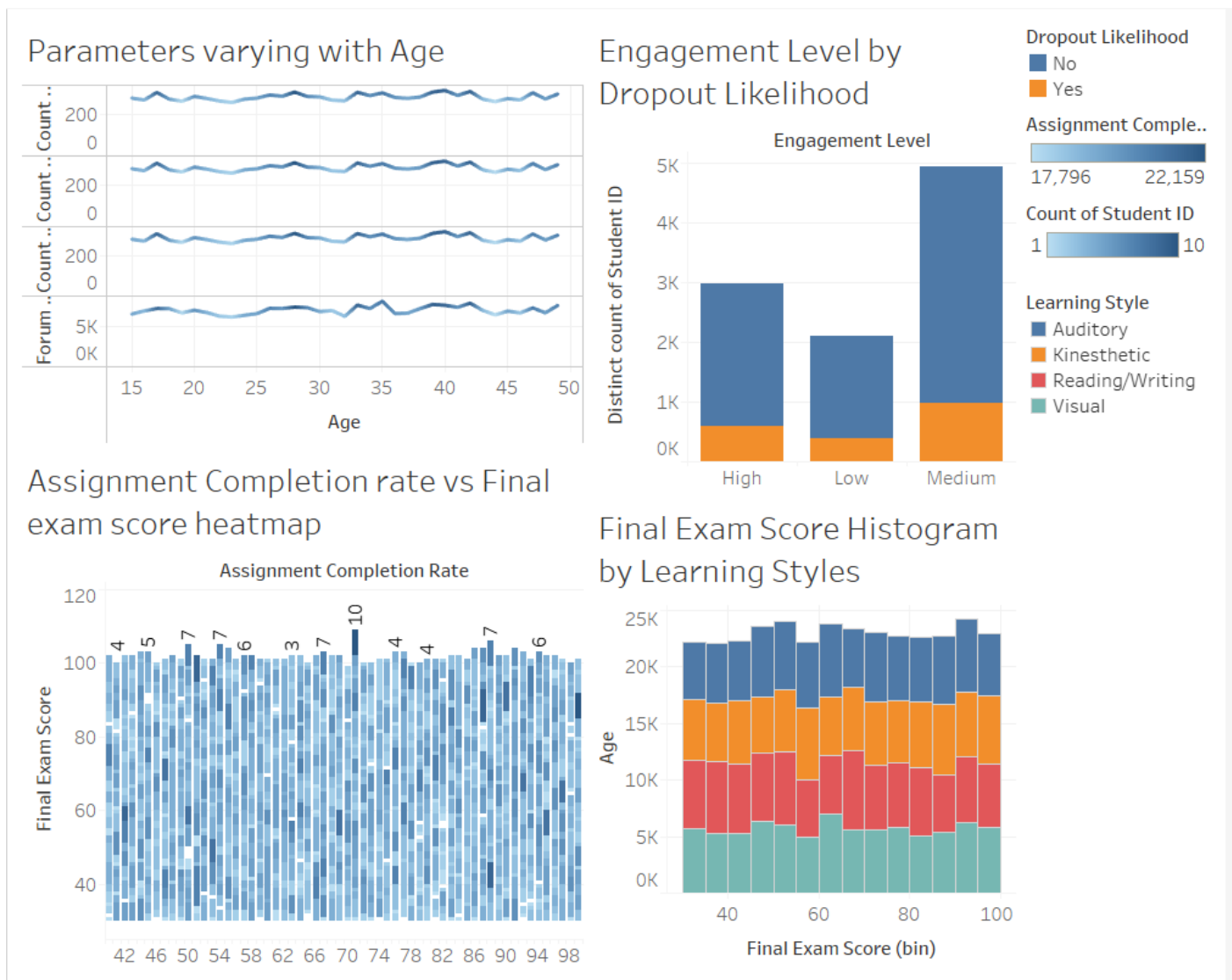
## Tableau additional figures:



*Fig 32. Treemap (Dropout by Learning Styles)*

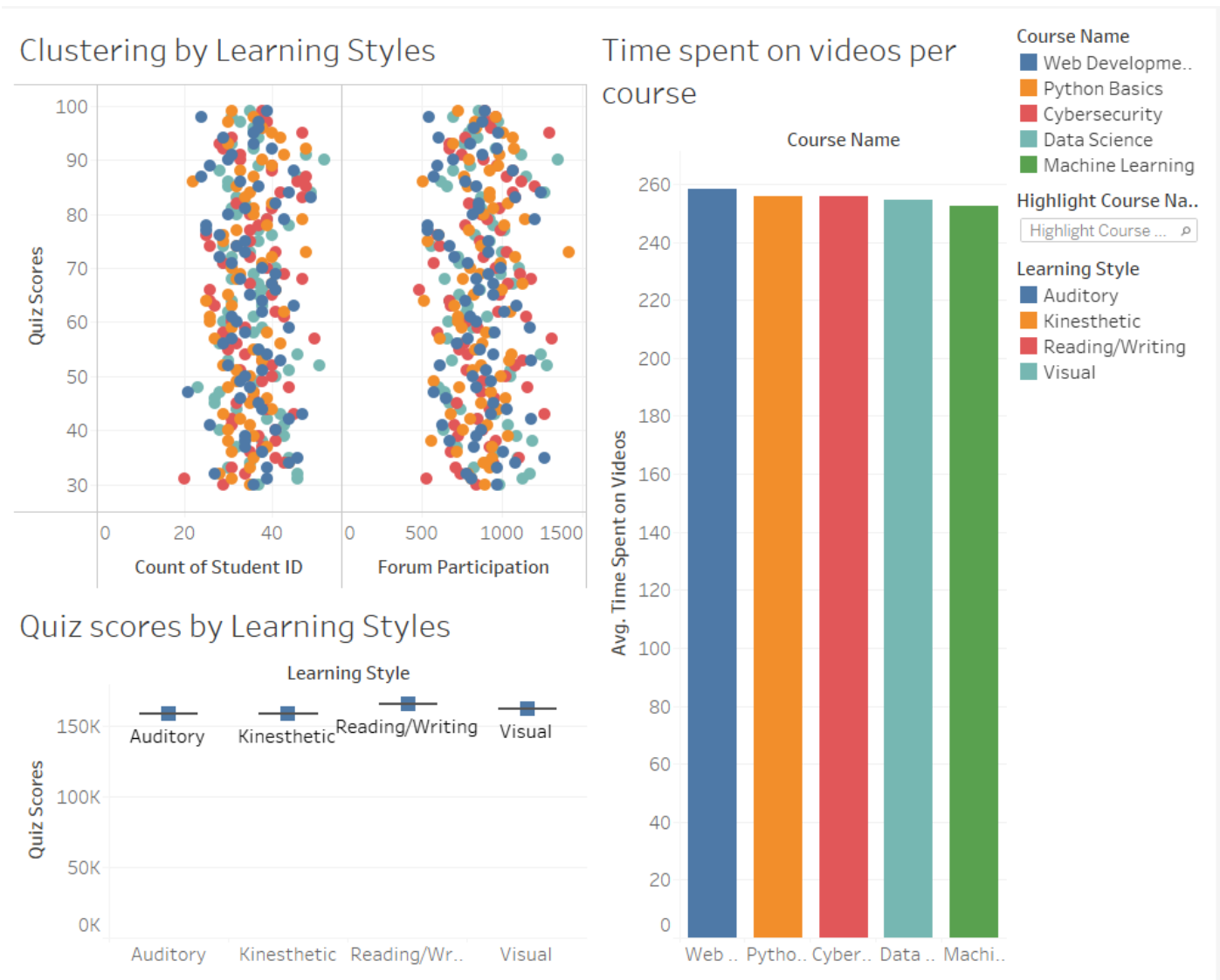
The treemap in Fig 32. illustrates the chance of student dropout based on learning method. The block size denotes the number of students, with darker tones suggesting more. Most non-dropouts come from Reading/Writing and Visual styles, whereas dropouts are more likely among Visual, Kinesthetic, and Auditory learners. This shows a relationship between learning style and dropout risk, which necessitates additional investigation into its impact on student retention.





*Fig 33. Dashboard (Performance Analysis)*

The dashboard in Fig 33. examines student performance, engagement, and dropout chances. The top-left chart tracks parameters by age, whereas the top-right bar chart correlates lower engagement levels with higher dropout rates. The bottom-left heatmap indicates a favorable relationship between assignment completion and final exam scores. The bottom-right histogram depicts varying score distributions across learning techniques. These insights can assist educators improve student retention and success.



*Fig 34. Dashboard (Engagement and Learning Patterns)*

The dashboard in Fig 34. investigates student performance and engagement across learning styles and courses. The top-left scatter figure clusters students according to quiz scores, student count, and forum activity, displaying a variety of distributions. The top-right bar chart compares the average time spent watching videos per course, suggesting similar participation across subjects. The bottom box plot analyzes quiz scores based on learning method, revealing performance trends. These findings aid in understanding how learning styles influence engagement and outcomes.