Enhancing Disease Prediction in Healthcare: A Data Mining Approach Using Supervised and Unsupervised Learning Models

Pranav Gujjar-100443924

May 2024

Abstract

In order to anticipate diseases, this study uses data mining techniques on a patient-related factor dataset to examine the Knowledge Discovery in Databases (KDD) process in healthcare. This prepares the dataset for both supervised and unsupervised learning models by incorporating pre-processing activities like data cleansing and variable encoding. With the goal of improving patient health outcomes, the project investigates a variety of algorithms to create a prediction model that will increase diagnosis accuracy and guide treatment plans.

1 Introduction

Data mining has become an essential tool in the healthcare industry for gathering insights from patient data that can be used to improve treatment effectiveness and disease prediction. Using the Knowledge Discovery in Databases (KDD) process, the research finds important characteristics that impact disease outcomes by sifting through intricate healthcare records. The study analyzes and interprets data patterns through a thorough investigation of different supervised and unsupervised learning models, which is essential for creating a prediction framework. The study's prediction models are intended to aid in clinical judgment and greatly advance individualized patient care.

2 Data Description

The dataset includes two files: 'disease_train.csv' and 'disease_test.csv'. It is derived from a collection of healthcare data that focuses on a certain ailment. The training set consists of 4250 entries across 24 different parameters. These parameters aim to encompass a wide range of patient-related information that is critical for evaluating health status and disease risk. The variables include patient ID, age, gender, sickness status, pregnancy status, various text descriptions (text_X1 to text_X6), types of concerns, enlargement and tumor status, presence of disorders, medication usages (medication_A and medication_B), mental health indicators, mood stabilizer usage, history of surgery, types of treatments received, suspicions related to the disease, and the target variable, which likely indicates disease presence or severity. For a more complete description of each variable, please refer the appendix.

3 Methodology

In the field of healthcare analytics, particularly disease prediction, the KDD approach highlights the utilization of both supervised and unsupervised models, reflecting the necessity for several methods to handle different datasets. The researcher and team [Lorena Gallego-Viñarás(2024)] used supervised and unsupervised models to predict Alzheimer's disease, highlighting the need for adaptability in medical diagnostics. In addition,the researcher and team [Junbo Peng and Mingdong Fan(2024)] used unsupervised manifold learning to reduce CT scan artifacts, highlighting the use of unsupervised methods in medical imaging, which is crucial for illness detection and progression tracking. From the given dataset with variable details,Adopt KDD process as follows in the figure 1.

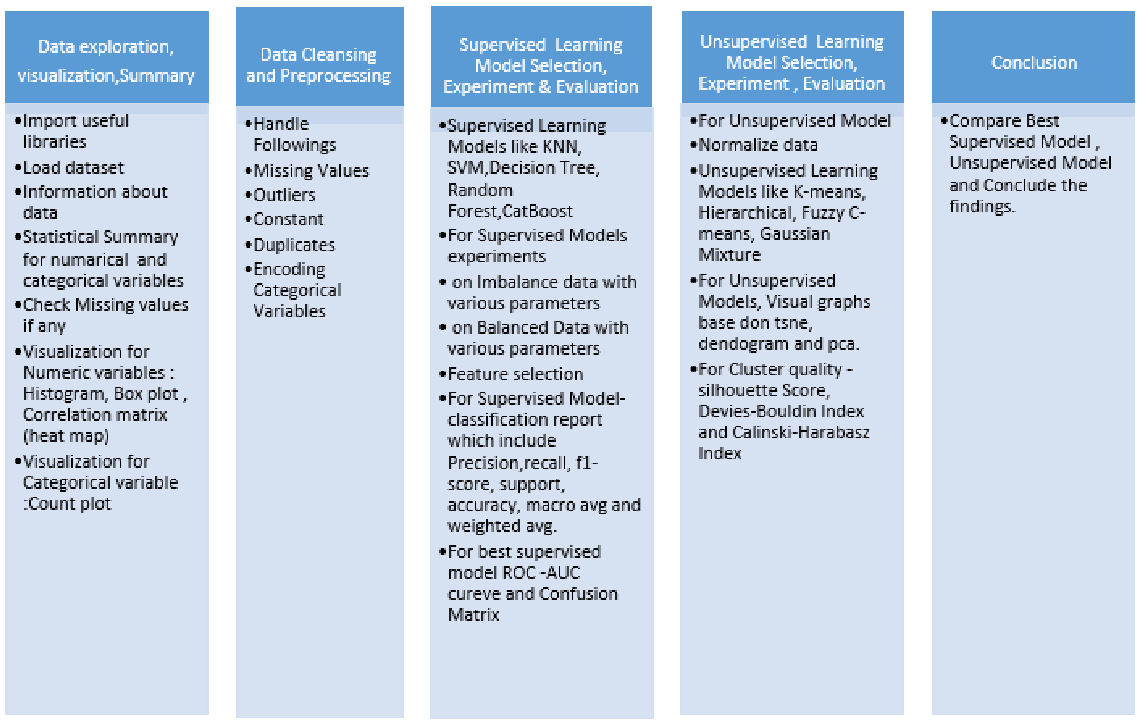
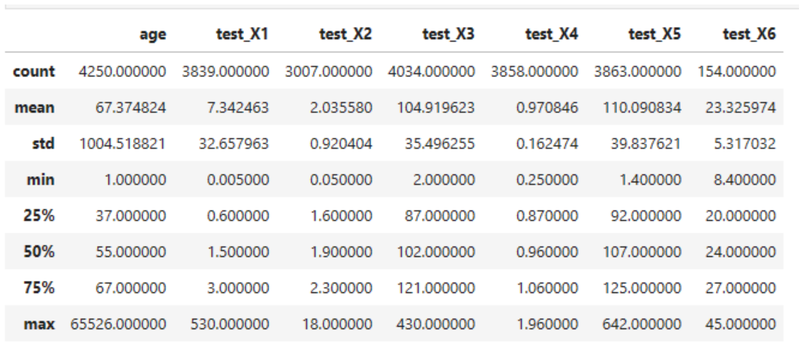

Data exploration, visualization,Summary
•Import useful libraries
•Load dataset
•Information about data
•Statistical Summary for numarical and categorical variables
•Check Missing values if any
•Visualization for Numeric variables : Histogram, Box plot , Correlation matrix (heat map)
•Visualization for Categorical variable :Count plot
Data Cleansing and Preprocessing
•Handle Followings
•Missing Values
•Outliers
•Constant
•Duplicates
•Encoding Categorical Variables
Supervised Learning Model Selection, Experiment & Evaluation
•Supervised Learning Models like KNN, SVM,Decision Tree, Random Forest,CatBoost
•For Supervised Models experiments
• on Imbalance data with various parameters
• on Balanced Data with various parameters
•Feature selection
•For Supervised Model- classification report which include Precision,recall, f1-score, support, accuracy, macro avg and weighted avg.
•For best supervised model ROC -AUC cureve and Confusion Matrix
Unsupervised Learning Model Selection, Experiment , Evaluation
•For Unsupervised Model
•Normalize data
•Unsupervised Learning Models like K-means, Hierarchical, Fuzzy C-means, Gaussian Mixture
•For Unsupervised Models, Visual graphs base don tsne, dendogram and pca.
•For Cluster quality - silhouette Score, Devies-Bouldin Index and Calinski-Harabasz Index
Conclusion
•Compare Best Supervised Model , Unsupervised Model and Conclude the findings.

Figure 1: KDD-Methodology

3.1 Data Exploration,Visualization,Summary

The dataset presents a combination of numerical and categorical variables, all of which contain non-null values, offering a rich source for analysis. This diverse mix facilitates a comprehensive examination, integrating quantitative metrics and categorical descriptors for strong insights.
Numerical variables summary statistics in figure 2: A dataset with a variety of ranges and distributions is suggested by the summary statistics, especially one that has an exceptionally high maximum age value that may indicate outliers or problems with data entry. Missing data is indicated by count disparities across test variables, and varying standard deviations suggest varying

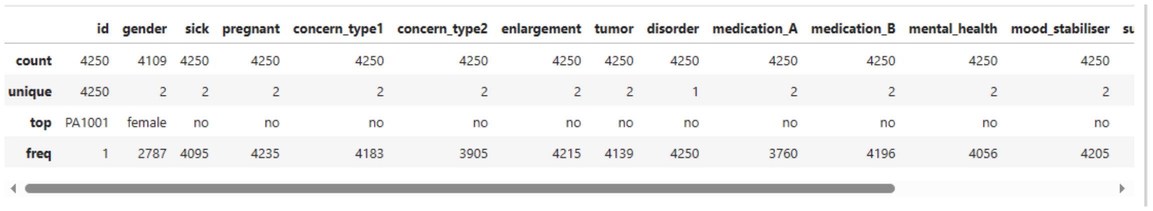
degrees of test result variability. These first findings imply that prior to conducting a thorough analysis or modeling, data cleaning and potentially standardization are necessary.



	age	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6
count	4250.000000	3839.000000	3007.000000	4034.000000	3858.000000	3863.000000	154.000000
mean	67.374824	7.342463	2.035580	104.919623	0.970846	110.090834	23.325974
std	1004.518821	32.657963	0.920404	35.496255	0.162474	39.837621	5.317032
min	1.000000	0.005000	0.050000	2.000000	0.250000	1.400000	8.400000
25%	37.000000	0.600000	1.600000	87.000000	0.870000	92.000000	20.000000
50%	55.000000	1.500000	1.900000	102.000000	0.960000	107.000000	24.000000
75%	67.000000	3.000000	2.300000	121.000000	1.060000	125.000000	27.000000
max	65526.000000	530.000000	18.000000	430.000000	1.960000	642.000000	45.000000

Figure 2: Numerical Variable Statistical Summary

Categorical Variable Summary Statistics in figure 3: According to the summary statistics for categorical data, there are 4,250 entries in each column representing categorical variables, with the exception of "id," which has less non-null values than the other columns, suggesting some missing data. Except for "id," which is unique for every entry, and "target," which has three unique values, every column has two unique values. The majority of the variables fall into the category "no," which suggests binary traits; the exceptions are "gender," where "female" is the most common, and "target," where "low_risk" is more common. With "id" serving as a unique identifier and "target" as a ternary result variable, the majority of the variables in the data seem to be binary categorical (yes/no).



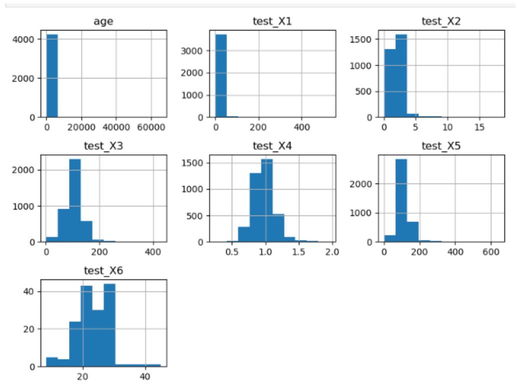
	id	gender	sick	pregnant	concern_type1	concern_type2	enlargement	tumor	disorder	medication_A	medication_B	mental_health	mood_stabiliser	su
count	4250	4109	4250	4250	4250	4250	4250	4250	4250	4250	4250	4250	4250	
unique	4250	2	2	2	2	2	2	2	1	2	2	2	2	
top	PA1001	female	no	no	no	no	no	no	no	no	no	no	no	
freq	1	2787	4095	4235	4183	3905	4215	4139	4250	3760	4196	4056	4205	

Figure 3: Categorical Variable Statistical Summary

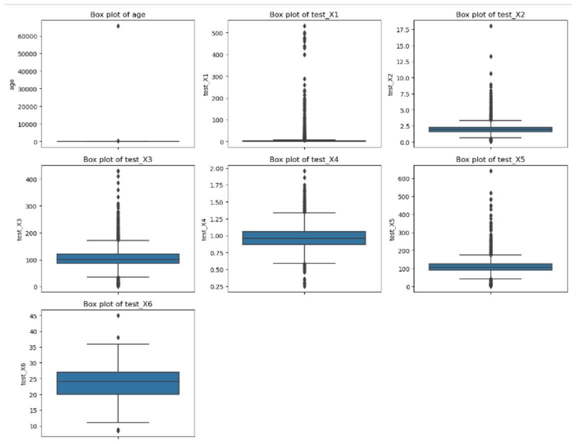
Upon reviewing the missing value information from the summary statistics of the numerical and categorical variables, it was discovered that only the variables gender and tests_x1 through test_X6 had missing values. At 4096 out of 4250, test_X6 has the highest missing value. The second-highest value is text_X2, with 1243 out of 4250, while the gender variable has the fewest missing values (141). These missing values will be addressed throughout the preprocessing and data cleaning stages.

Visualization of Numerical Variables as follows:

Histogram: Figure 4a presents the distribution of test results among age groups in the dataset through histograms. An outlier much to the right of the age histogram suggests a potential data input issue. The distribution of the data is left-skewed. The right-skewed distributions with extended tails displayed by Tests_X1, Test_X3, and Test_X5 point to a concentration of low values and a small number of high ones. The bimodal distributions of Test_X2 and Test_X4 show two shared values or ranges in which the observations are concentrated. Test_X6 has a relatively regular distribution and a very small range of values, centered around 10 to 30. This data visualization highlights the skewness and possible outliers in the dataset, which may need to be cleaned up or transformed before being subjected to additional analysis.



(a) Histogram for numerical variables



(b) Boxplot for numerical variables

Figure 4: Histogram and boxplot for numerical variables

Boxplot: For the numerical data, the box plots in figure 4b show a variety of distributions and possible outliers. The 'age' variable displays a notable anomaly that could potentially be a mistake due to its disconnection from the remaining data. Test_X1, Test_X3, and Test_X5 show a high number of high-value outliers, whereas Test_X2 and Test_X4 show outliers on both ends. These tests are the ones that show outliers the most. With a few outliers, Test_X6's interquartile range is not very wide. Test_X1, X3, and X5 have a right skew in the data, as indicated by the median of each box plot being closer to the bottom quartile, but Test_X2 and X4 have a more symmetric distribution. The large number of outliers raises the possibility that preprocessing the data will be necessary to control these extreme values before more analysis can be done.
Correlation Matrix(heatmap): The correlation matrix figure 6 shows the link between age and test factors. Notably, test_X2 and test_X3 exhibit a high positive connection, as do test_X3 and test_X5, indicating that as one increases, the other tends to increase too. Test_X2 has a substantial negative correlation with test_X6, which means that while test_X2 grows, test_X6 decreases, and vice versa. Age appears to have a modest association with all test variables except test_X6, where there is a slight positive correlation. Other correlations between tests vary from moderate to weak. Overall, the matrix shows a combination of strong, moderate, and weak correlations, which may inform future research of how these variables interact .
Visualization for Categorical Variables as follow:
Count Plot:

The count plots figure 5 indicate that in this dataset, for binary categorical variables, one category significantly dominates over the other, which is consistent across most of the plotted features. Specifically, 'gender' skews towards female, most individuals are not 'sick' or 'pregnant', and there's

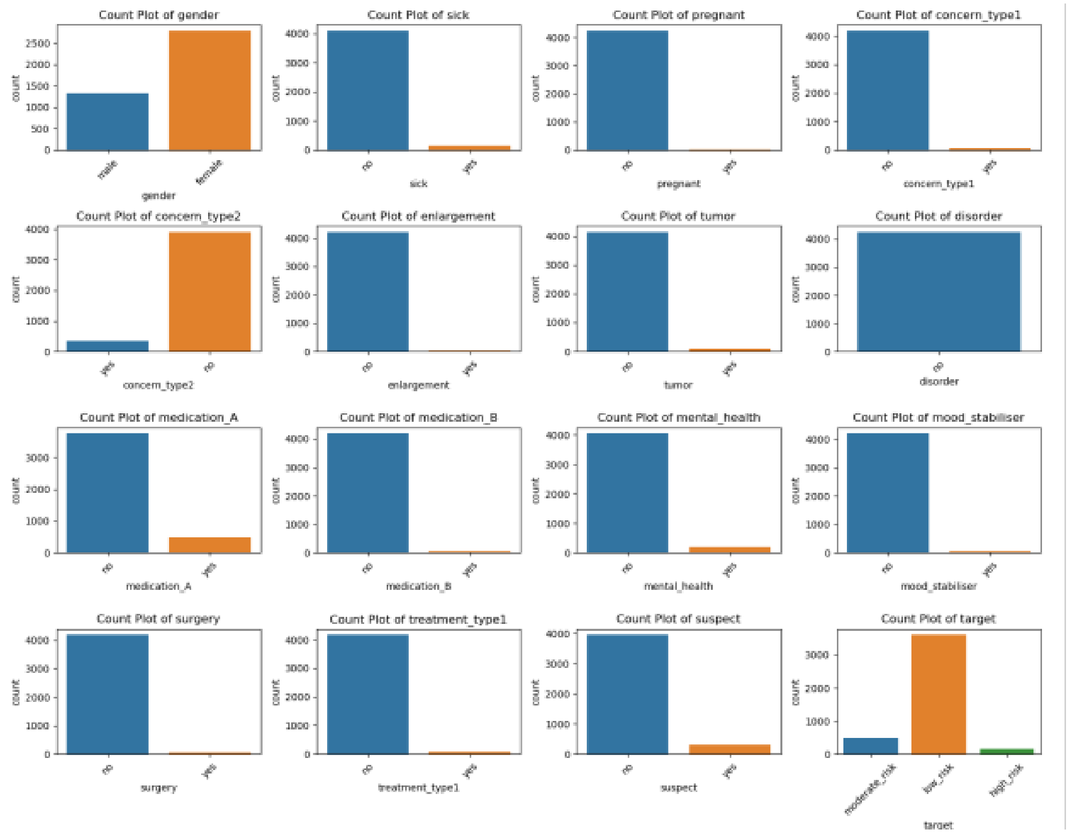


Figure 5: Count plot for Categorical Variables

a large prevalence of 'no' responses in 'concern_type1', 'concern_type2', 'enlargement', 'tumor', 'disorder', both 'medication_A' and 'medication_B', 'mental_health', 'mood_stabiliser', 'surgery', and 'treatment_type1'. 'Suspect' also shows a majority 'no' response. In the 'target' variable, 'low_risk' is the most common category, followed by 'mid_risk', with 'high_risk' being the least frequent. This could suggest imbalances that may need to be considered in predictive modeling or further analysis.

3.2 Data Cleansing and Preprocessing

Effective data cleansing and preprocessing are critical for dealing with difficulties such as missing values, outliers, and uneven data distribution in datasets containing both numerical and categorical variables. Recent study highlights the importance of these processes in assuring the quality and dependability of future analyses. Preparing datasets for predictive modeling and analysis requires core techniques including data cleansing, transformation, feature extraction, and coping with missing data [Sarada(2024), Kapsis(2024)]. These approaches make it easier to extract relevant insights from data, which improves the outcomes of data-driven decision-making processes across a variety of disciplines.
Based on insights from the mentioned research and the exploratory data analysis (EDA) conducted,

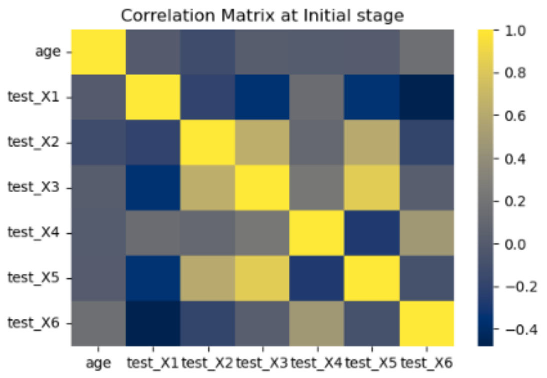


Figure 6: Correlation matrix for Numerical Variables

the data preparation process encompassed the following steps:
Handled Missing Values: From the above EDA, Numerical variable test_X6 has the greatest missing value 4096 out of 4250, meaning 96.36%, which is greater than 60%. The test_X6 column was removed for further analysis, and missing values for the remaining variables (test_X1 to test_X5) were handled using median imputation, which is resilient to outliers. For the categorical variable 'gender', mode imputation was used to fill in missing values with the most common category. Also examined and confirmed that no missing values remained in the dataset.
Handled Outliers: The strategy is aimed to reduce the influence of outliers exhibited in EDA box plots. The box plots depict data distribution and outliers for different tests and ages. Many data points are outside the normal range (beyond the whiskers), which the IQR technique would classify as outliers. By using the winsorize function on these numerical variables, the method replaces extreme data points with the greatest or lowest value within the estimated "fences," putting the outliers into a range more typical of the core data. This is done to prevent these extreme numbers from confounding statistical studies, such as mean or standard deviation calculations, and to improve the performance of machine learning models that can be sensitive to such outliers.
Handled Constant: Removing constant columns, like 'disorder' from this dataset, makes the dataset simpler and more efficiently processed. Constant columns lack variability and are therefore not useful for analysis or predictive modeling.
Handled Duplicates: Duplicates are eliminated to avoid distorting the data and analysis findings; 42 duplicates were identified and removed from this dataset, guaranteeing that each record adds distinctively to the understanding and precision of the model.
Encoding Categorical Variables: Encoding is an important data preprocessing technique that converts categorical variables into numerical representations so that machine learning models can process them. The binary variables were one-hot encoded, resulting in unique columns for each category with binary indicators, while the multi-class 'target' column was label encoded. These methods help algorithms identify patterns in categorical data, which is vital for making accurate predictions [Umrao and Bansal(2024)].

3.3 Supervised Learning Model Selection, Experiments and Evaluation

Supervised learning models include K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Trees, Random Forests, and CatBoost, which are all designed to address different types of data and learning tasks. KNN classifies or regresses based on the similarity of data points, whereas SVM separates classes using a high-dimensional hyperplane. Decision Trees use tree structures to partition data into subsets, whereas Random Forests combine Decision Trees to form an ensemble that reduces overfitting. CatBoost, a gradient boosting method, excels at handling categorical data thanks to its advanced encoding strategies. The selection of hyperparameters has a considerable impact on model performance, highlighting the significance of precise parameter tuning [Hu and Xiong(2024), Lidia Pascual-Sánchez(2024)].
Because datasets are so different and bring unique challenges, experimentation with various parameter combinations and models is critical. Different data distributions, class imbalances, and feature correlations can all have a major impact on model accuracy.The study can determine the best model and parameter set for a given dataset by thoroughly evaluating multiple configurations. This experimental method is crucial for creating generalizable and resilient models that can handle real-world data variability [Abu Sarwar Zamani(2024), Al-Alshaikh(2024)].
To discover the best model, divide the data into training (70%) and test sets (30%), use GridSearchCV to automate the search for the optimal parameter combination, and evaluate each model's performance using metrics like accuracy, ROC curves, and confusion matrices. This systematic examination allows for equal model comparisons, guaranteeing that the selection is based on empirical evidence of task performance. The ultimate goal is to create a model that not only performs well on training data but also generalizes to unseen data, resulting in reliable predictions or classifications in practical applications [Hu and Xiong(2024), Lidia Pascual-Sánchez(2024)].

3.3.1 Experimental outcomes for an imbalanced dataset across different supervised learning models:

The table 1 presents a summary of model performance metrics, including accuracy, precision (weighted avg), recall (weighted avg), and F1-score (weighted avg), for various classification models. CatBoost demonstrates the highest accuracy of 96.28%, followed closely by Random Forest at 96.04%. In terms of precision and recall, CatBoost also achieves the highest values among the models, indicating its effectiveness in correctly classifying instances and capturing all positive instances. Decision Tree and SVM show slightly lower performance compared to CatBoost and Random Forest but still exhibit strong metrics overall. KNN lags behind the other models, demonstrating the lowest accuracy and precision values. Overall, the table provides a clear comparison of model performance, with CatBoost and Random Forest emerging as the top performers.

3.3.2 Experimental outcomes for an balanced dataset across different supervised learning models:

Balancing the dataset is crucial to prevent bias towards majority classes, ensuring fair model performance across all classes. SMOTE (Synthetic Minority Over-sampling Technique) is selected for its ability to generate synthetic samples for the minority class, effectively expanding the dataset and improving model generalization without introducing duplicate information. By synthesizing minority class samples, SMOTE enhances the model's ability to learn from underrepresented classes,

Model	Performance Metrics (Imbalanced Data)			
	Accuracy	Precision (weighted avg)	Recall (weighted avg)	F1-score (weighted avg)
KNN	0.8979	0.89	0.90	0.88
SVM	0.9390	0.94	0.94	0.94
Decision Tree	0.9549	0.96	0.95	0.95
Random Forest	0.9604	0.96	0.96	0.96
CatBoost	0.9628	0.97	0.96	0.96

Table 1: Summary of Model Performance with imbalanced dataset

resulting in more balanced and accurate predictions overall.

Model	Performance Metrics (Balanced Data)			
	Accuracy	Precision (weighted avg)	Recall (weighted avg)	F1-score (weighted avg)
KNN	0.9322	0.93	0.93	0.93
SVM	0.9788	0.98	0.98	0.98
Decision Tree	0.9757	0.98	0.98	0.98
Random Forest	0.9819	0.98	0.98	0.98
CatBoost	0.9826	0.98	0.98	0.98

Table 2: Summary of Model Performance with Balanced dataset

The table 2 provides a comparative analysis of model performance based on key classification metrics. CatBoost outperforms other models with the highest accuracy of 98.26% and consistently high precision, recall, and F1-score (weighted avg) values. Random Forest follows closely with an accuracy of 98.19% and similarly strong metrics across all categories. SVM and Decision Tree also demonstrate notable performance, with accuracies of 97.88% and 97.57%, respectively. KNN shows slightly lower performance with an accuracy of 93.22%. Overall, CatBoost and Random Forest exhibit the most robust performance across all metrics, while KNN trails behind the other models in terms of accuracy and precision.

3.3.3 Feature selection:

Feature selection is essential for improving supervised learning models by choosing the most relevant features. To choose the best-performing features, SelectKBest, a statistical test similar to ANOVA, is used. The algorithm selects the most informative characteristics from the dataset based on the target number (k). Visualizing these selected features and their accompanying scores sheds light on their significance for classification problems. This simplified procedure increases model correctness and interpretability while decreasing computational complexity.

Evaluation of Feature selected bar chart: The bar chart figure 7 visualizes feature scores, presumably of a model's variables. Test_X1 has the highest score, indicating its significant predictive power or importance in the model, followed by test_X5 and test_X2. In contrast, variables such as

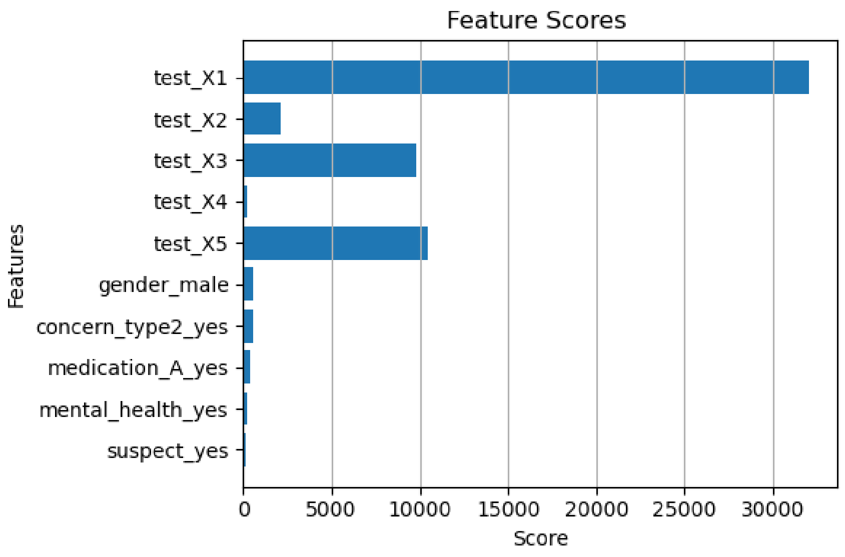


Figure 7: Feature selection graph

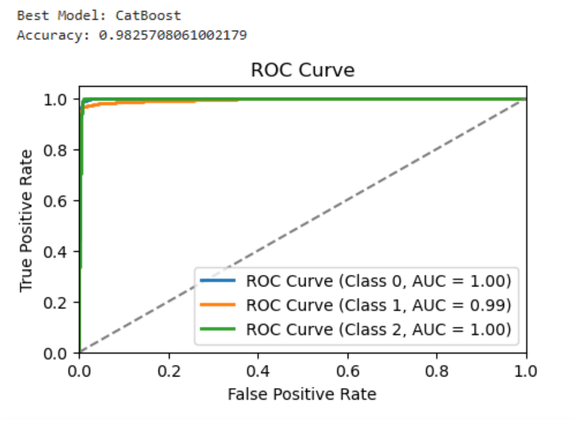
gender_male, concern_type2_yes, medication_A_yes, mental_health_yes, and suspect_yes have much lower scores, suggesting they have a smaller impact on the model's predictions or outcomes. The distinction between the test variables and the binary categorical variables (yes/no features) in terms of score suggests different types of features contribute unequally to the model's performance.

3.3.4 Summary of experiments and cross validation on Best Supervised Learning model Catboost:

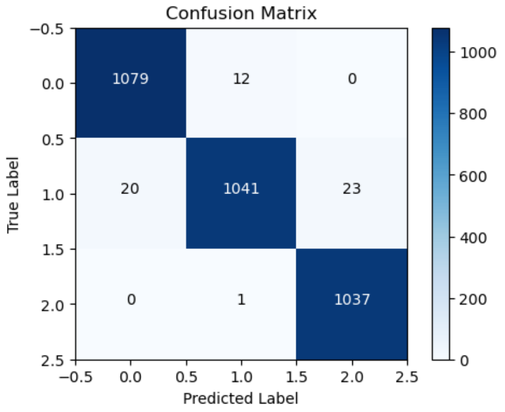
Based on experimental results from imbalanced and balanced datasets, as well as feature selection, CatBoost regularly emerges as the best-performing model, with excellent accuracy, precision, recall, and F1-score values. Its effectiveness is maintained even after feature selection, indicating robustness in identifying situations. Cross-validation, ROC-AUC curve, and Confusion matrix analysis are used to test CatBoost's reliability, revealing insights into its capacity to generalize to new data and reliably categorize cases across several classes. This detailed review supports CatBoost as the best alternative for categorization jobs, highlighting its dependability and performance under various scenarios.

ROC-AUC Curve based on highest accuracy of supervised learning model CatBoost: The figure 8a displays a Receiver Operating Characteristic (ROC) curve for a CatBoost model, which boasts a high accuracy of approximately 98.3%. The ROC curves for Class 0 and Class 2 are perfect, with an Area Under the Curve (AUC) of 1.00, indicating a model with perfect classification ability for these classes. Class 1 has a nearly perfect AUC of 0.99, showing that the model also performs exceptionally well for this class. Overall, the model demonstrates excellent discriminatory power with minimal false positive rates across all classes.

Confusion Matrix based on highest accuracy of supervised learning model CatBoost: The confusion matrix 8b presented shows the performance of a classification model with three classes (0, 1, 2). The diagonal cells (1079, 1041, 1037) represent the number of correct predictions



(a) ROC-AUC Curve



(b) Confusion Matrix

Figure 8: ROC-AUC Curve and Confusion Matrix for CatBoost

for each class, indicating high accuracy. Misclassifications are minimal, as shown by the off-diagonal cells, with only 12 instances of Class 0 being misclassified as Class 1, 20 instances of Class 1 being misclassified as Class 0, and similarly low numbers for other misclassifications. The model appears to be highly effective with very few errors, which aligns with the high accuracy noted previously.

3.4 Unsupervised Learning Model Selection, Experiments and Evaluation:

Unsupervised learning methods are vital for detecting hidden patterns and structures in datasets without the help of labeled outcomes. These models, which include K-means clustering, Hierarchical Clustering, Fuzzy C-means, and Gaussian Mixture Models (GMM), meet a variety of analytical requirements. K-means clustering divides datasets into k unique non-overlapping subgroups based on the mean distance from the centroid, which optimizes intra-cluster variance [Ramineni Anuraag(2024)]. Hierarchical Clustering creates a dendrogram, providing data segmentation in a tree-like structure that demonstrates the arrangement of clusters generated at various levels, boosting interpretability [Fard S.S.(2024)]. Fuzzy C-means enables data points to belong to numerous clusters with degrees of membership, providing flexibility in cluster assignment [Matheus d.f.O. Baffa(2024)]. GMM provides a probabilistic model for cluster assignment by assuming data is created from a mixture of a finite number of Gaussian distributions with unknown parameters [Lun(2024)].
In unsupervised learning, **Normalization via the Standard Scaler** is crucial because it standardizes features to a mean of zero and a standard deviation of one, providing equal contribution across all dimensions and enhancing the pattern-detection performance of models such as K-means and Gaussian Mixture Models [G. and Gramfort A. Michel V.(2011)].
Visualization techniques such as t-SNE, PCA, and dendrograms are instrumental in interpreting the results of unsupervised learning. t-SNE excels in preserving local structures, revealing clusters in high-dimensional data by mapping it to a lower-dimensional space [Ramineni Anuraag(2024)]. PCA identifies the directions of maximum variance in high-dimensional data, reducing its dimen-

sions while retaining as much variability as possible [Fard S.S.(2024)]. Dendrograms, particularly useful in Hierarchical Clustering, visually represent the arrangement and nested relationships of clusters, offering insights into data structure at different levels of granularity [Matheus d.f.O. Baffa(2024)]. **The evaluation of cluster quality** is performed using metrics such as the Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index, which assess compactness, separation, and density of clusters to determine the most suitable model and parameters for the dataset [Lun(2024)].

3.4.1 Determine K value:

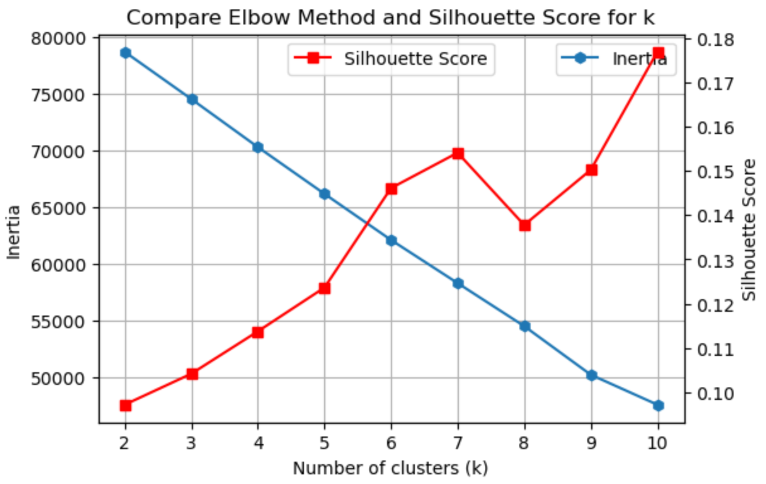


Figure 9: Determine K value by Elbow and Silhouette methos

Both the Elbow Method and the Silhouette Score offer useful information when figuring out the number of clusters (k-value) for different unsupervised learning models. The Elbow Method concentrates on the point at which the rate of decline of inertia decreases, indicating an appropriate number of clusters, while the Silhouette Score evaluates the clarity of cluster definition, with higher scores reflecting better-separated clusters. The provided figure 9 in appedix does not show a distinct elbow, but the rising Silhouette Score up to k=5 hints at five as the potential optimal cluster count.Ultimately, deciding the k value based on domain expertise is often more relevant. Here,selected k = 3 for further analysis.

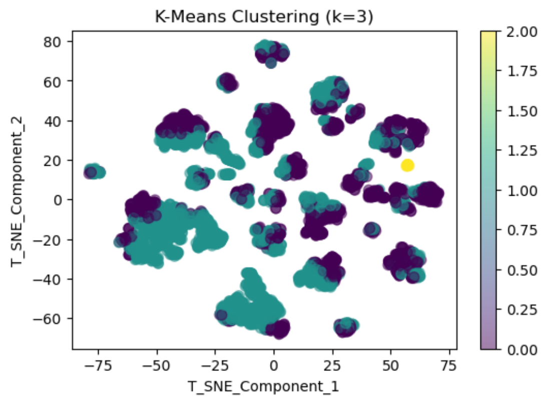
3.4.2 K-means Clustering:

Based on the Elbow Method, Silhouette Score, and practical knowledge, the scaled dataset was subjected to K-means clustering, selecting k=3. Following this, t-SNE reduced the data to two dimensions for visualization in figure 10a, clearly separating the clusters with distinct colors. This confirms the algorithm's success in discerning the data's structure, as both the quantitative metrics and visual representation demonstrate cohesive clustering that aligns with domain expectations
The K-means algorithm has identified three clusters with the rescaled centroids delineating their core traits. Cluster 0 and 1 are substantial, containing 1906 and 2287 data points, respectively,

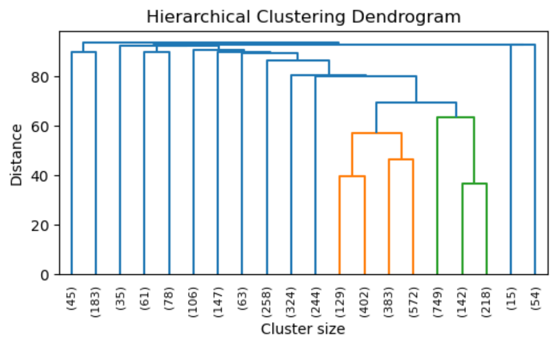
while cluster 2 is significantly smaller, with just 15 data points, mirroring the disparities observed in the t-SNE visualization.

3.4.3 Hierarchical Clustering:

Upon executing Hierarchical Clustering on the scaled dataset, three distinct clusters emerged, with a dominant cluster containing 3911 instances, highlighting the skewed distribution of data points. The dendrogram in figure 10b produced from the Ward linkage method illustrates this imbalance, showing a high degree of similarity within the major cluster, while the smaller clusters, with 228 and 69 instances, indicate more discrete data subsets. This suggests a hierarchical organization where most data points share common characteristics, as visualized by the dendrogram's varying branch lengths and cluster sizes.



(a) K-means clustering with t-SNE for visualization



(b) Hierarchical Clustering Dendrogram

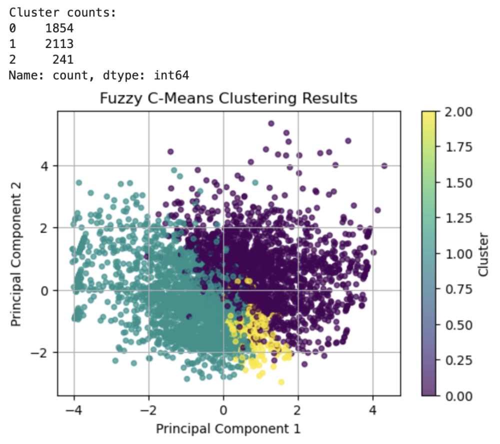
Figure 10: Comparison of clustering techniques

3.4.4 Fuzzy C-means Clustering:

The Fuzzy C-Means clustering results displayed in figure 11a through PCA show three clusters with balanced sizes: 1854, 2113, and 241 points respectively. The visualization highlights the algorithm's characteristic of soft assignment, with significant overlap between clusters, especially notable between the densest areas of green and purple points, illustrating the gradience in membership that Fuzzy C-Means allows for each point.

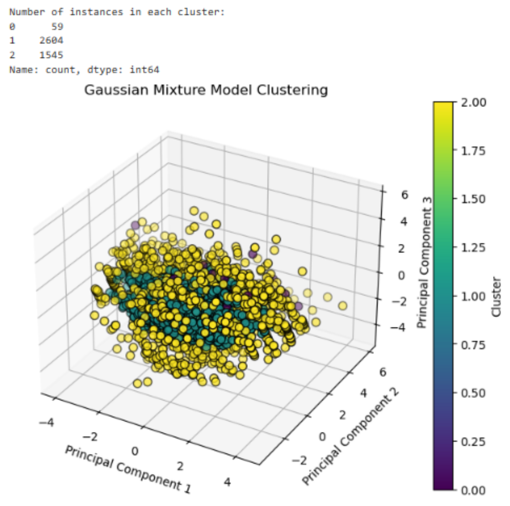
3.4.5 Gaussian Mixture Model Clustering:

In figure 11b 3D PCA visualization of Gaussian Mixture Model clustering, three clusters are differentiated by size and density: one with 59 instances, another with 2604, and a third containing 1545. The spatial distribution reflects the probabilistic nature of GMM, capturing a compact cluster of 59 data points, a dense aggregation of 2604, and a moderately dispersed set of 1545, highlighting the model's effectiveness in identifying varying degrees of grouping within the dataset.



```
Cluster counts:
0    1854
1    2113
2     241
Name: count, dtype: int64
```

(a) Fuzzy C-means Clustering results



```
Number of instances in each cluster:
0      59
1    2604
2    1545
Name: count, dtype: int64
```

(b) Gaussian Mixture Model

Figure 11: Comparison of clustering algorithms

3.4.6 Evaluation Cluster Quality with different models and methods:

The table 3 summarizes the evaluation metrics for different clustering models and methods. It includes the Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index for four clustering techniques: k-Means, Hierarchical Clustering, Fuzzy C-Means, and Gaussian Mixture Model.

Evaluation Metric	**k-Means**	**Hierarchical Clustering**	**Fuzzy C-Means**	**Gaussian Mixture Model**
Silhouette Score	0.098	0.306	0.018	0.211
Davies-Bouldin Index	2.308	1.656	3.521	3.665
Calinski-Harabasz Index	275.483	242.075	165.863	184.262

Table 3: Comparison of clustering models and methods based on evaluation metrics

The Silhouette Score measures the cohesion and separation of clusters, where higher values indicate better-defined clusters. Hierarchical Clustering exhibits the highest Silhouette Score (0.306), followed by Gaussian Mixture Model (0.211), while k-Means and Fuzzy C-Means have lower scores (0.098 and 0.018, respectively).

The Davies-Bouldin Index assesses cluster separation, with lower values indicating better clustering. Hierarchical Clustering achieves the lowest Davies-Bouldin Index (1.656), suggesting better cluster separation compared to the other methods.

The Calinski-Harabasz Index evaluates cluster dispersion, with higher values indicating denser and more well-separated clusters. Here, k-Means has the highest Calinski-Harabasz Index (275.483), followed by Hierarchical Clustering (242.075), while Fuzzy C-Means and Gaussian Mixture Model have lower values (165.863 and 184.262, respectively).

Overall, Hierarchical Clustering demonstrates competitive performance across all metrics, indicating well-separated and cohesive clusters. Conversely, Fuzzy C-Means exhibits relatively weaker performance, suggesting less distinct cluster boundaries

4 Conclusion:

In conclusion, after a comprehensive exploration of both supervised and unsupervised learning models applied to healthcare data for disease prediction, CatBoost emerges as the superior model among the supervised methods. Its strong performance, underscored by high accuracy and favorable precision, recall, and F1-score values, demonstrates its proficiency in classification tasks. The model's effectiveness is further validated through cross-validation, with ROC-AUC curves and confusion matrix analysis validating its reliability and ability to generalize well to new data. Among the unsupervised approaches, Hierarchical Clustering stands out with the highest Silhouette Score, indicative of well-separated and cohesive clusters. Despite the strong performance of individual models within each category, CatBoost's consistent superiority across multiple metrics positions it as the best model for this project, offering a promising tool for disease prediction in the healthcare industry.

References

[Abu Sarwar Zamani(2024)] Abdallah Saleh Ali Shatat Md. Mobin Akhtar Mohammed Rizwanullah Sara Saadeldeen Ibrahim Mohamed Abu Sarwar Zamani, Aisha Hassan Abdalla Hashim. Implementation of machine learning techniques with big data and iot to create effective prediction models for health informatics. *Biomedical Signal Processing and Control*, 94: 106247, 2024. ISSN 1746-8094. doi: https://doi.org/10.1016/j.bspc.2024.106247. URL `https://www.sciencedirect.com/science/article/pii/S1746809424003057`.

[Al-Alshaikh(2024)] P Prabu Poonia Ramesh Chandra Saudagar Abdul Khader Jilani Yadav Manoj AlSagri Hatoon S. AlSanad Abeer A. Al-Alshaikh, Halah A. Comprehensive evaluation and performance analysis of machine learning in heart disease prediction. *Scientific Reports*, 14(1):7819, 04 2024. ISSN 2045-2322. doi: 10.1038/s41598-024-58489-7. URL `https://doi.org/10.1038/s41598-024-58489-7`.

[Fard S.S.(2024)] Wells P.S. Fard S.S., Perkins T.J. Machine learning analysis of bleeding status in venous thromboembolism patients. *Research and Practice in Thrombosis and Haemostasis*, 2024. doi: 10.1016/j.rpth.2024.102403.

[G. and Gramfort A. Michel V.(2011)] Pedregosa F. Varoquaux G. and Grisel O. Blondel M. Prettenhofer P. Weiss R. Dubourg V. Vanderplas J. Passos A. Cournapeau D. Brucher M. Perrot M. Duchesnay E. Gramfort A. Michel V., Thirion B. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 2011. URL `https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf`.

[Hu and Xiong(2024)] Ting Hu and Jing Xiong. Sparse online regression algorithm with insensitive loss functions. *Journal of Multivariate Analysis*, page 105316, 2024. ISSN 0047-259X. doi: https://doi.org/10.1016/j.jmva.2024.105316. URL `https://www.sciencedirect.com/science/article/pii/S0047259X2400023X`.

[Junbo Peng and Mingdong Fan(2024)] Huiqiao Xie Junbo Peng, Chih-Wei Chang and Justin Roper Richard L. J. Qiu Xiangyang Tang Xiaofeng Yang Mingdong Fan, Tonghe Wang. Metal artifact reduction in CT using unsupervised sinogram manifold learning. In Rebecca Fahrig, John M. Sabol, and Ke Li, editors, *Medical Imaging 2024: Physics of Medical Imaging*, volume 12925, page 129252G. International Society for Optics and Photonics, SPIE, 2024. doi: 10.1117/12.3006947. URL `https://doi.org/10.1117/12.3006947`.

[Kapsis(2024)] D. Kapsis. Applications for e-government: Global terrorism analysis. *University of Piraeus*, 2024. URL `http://dx.doi.org/10.26267/unipi`$_d$*ione*/3766.

[Lidia Pascual-Sánchez(2024)] Fernando Cruz-Roldán Antonio Hernández-Madrid Manuel Blanco-Velasco Lidia Pascual-Sánchez, Rebeca Goya-Esteban. Machine learning based detection of t-wave alternans in real ambulatory conditions. *Computer Methods and Programs in Biomedicine*, page 108157, 2024. doi: 10.1016/j.cmpb.2024.108157. URL `https://www.sciencedirect.com/science/article/pii/S0169260724001536`.

[Lorena Gallego-Viñarás(2024)] Anna Michela-Gaeta Gerard Pinol-Ripoll Ferrán Barbé Pablo M. Olmos Arrate Muñoz-Barrutia Lorena Gallego-Viñarás, Juan Miguel Mira-Tomás. Alzheimer's disease detection in psg signals, 2024.

[Lun(2024)] Yuan H. Ma P. et al. Lun, Y. A prediction model based on random survival forest analysis of the overall survival of elderly female papillary thyroid carcinoma patients: a seer-based study. *Endocrine*, 2024. doi: 10.1007/s12020-024-03797-1.

[Matheus d.f.O. Baffa(2024)] Friedrich Feuerhake Thomas M. Deserno Matheus d.f.O. Baffa, Nadine Sarah Schaadt. Unsupervised deep learning for clustering tumor subcompartments in histopathological images of non-small cell lung cancer. In Hiroyuki Yoshida and Shandong Wu, editors, *Medical Imaging 2024: Imaging Informatics for Healthcare, Research, and Applications*, volume 12931, page 129310Z. International Society for Optics and Photonics, SPIE, 2024. doi: 10.1117/12.3009180. URL `https://doi.org/10.1117/12.3009180`.

[Ramineni Anuraag(2024)] J Jayashree Sannapareddy Deepak Konduri Saketh Ramineni Anuraag, Konda Rishita. A comparative analysis using various algorithm approaches to enhance heart disease prognosis. *EAI Endorsed Transactions on Pervasive Health and Technology*, 10, Apr. 2024. doi: 10.4108/eetpht.10.5615. URL `https://publications.eai.eu/index.php/phat/article/view/5615`.

[Sarada(2024)] B. Sarada. Data wizard: Platform for preprocessing and feature extraction engineer. *Journal of Analytical and Experimental Modal Analysis*, 2024. URL `https://ir.mallareddyecw.com/id/eprint/552`.

[Umrao and Bansal(2024)] M. Umrao and V. Bansal. A machine learning based personalized yoga asanas recommendation engine. *Multimedia Tools and Applications*, 2024. doi: 10.1007/s11042-024-18983-6.

A Appendix:

Given dataset variable details:
Dataset Features: - - - - - - - - - - - - - - - - - -

'id': Unique patient ID.
'age': Age of the patient.
'gender': Gender of the patient.
'sick': Is the patient currently sick?
'pregnant': Is the patient currently pregnant?
'test_X1' to 'test_X6': Related to various medical tests.
'concern_type1' and 'concern_type2': Related to concerns.
'enlargement': Indicating enlargement.
'tumor': Does the patient have a tumor?
'disorder': Indicating a certain gland disorder.
'medication_A': Is the patient on medication A?
'medication_B': Is the patient on medication B?
'mental_health': Is the patient undergoing psychiatric evaluation?
'mood_stabiliser': Related to mood stabilization medication.
'surgery': Has the patient undergone surgery?
'treatment_type1': Is the patient undergoing treatment A?
'suspect': Does the patient suspect disease?
'target': Medical diagnosis (target variable/label).

B Juypiter Notebook Code:

Data Mining the Healthcare Dataset

Given Information about Dataset Dataset Features: - - - - - - - - - - - - - - - - - - - 'id': Unique patient ID. 'age': Age of the patient. 'gender': Gender of the patient. 'sick': Is the patient currently sick? 'pregnant': Is the patient currently pregnant? 'test_X1' to 'test_X6': Related to various medical tests. 'concern_type1' and 'concern_type2': Related to concerns. 'enlargement': Indicating enlargement. 'tumor': Does the patient have a tumor? 'disorder': Indicating a certain gland disorder. 'medication_A': Is the patient on medication A? 'medication_B': Is the patient on medication B? 'mental_health': Is the patient undergoing psychiatric evaluation? 'mood_stabiliser': Related to mood stabilization medication. 'surgery': Has the patient undergone surgery? 'treatment_type1': Is the patient undergoing treatment A? 'suspect': Does the patient suspect disease? 'target': Medical diagnosis (target variable/label).

In [1]:
```
#pip install catboost
```

In [2]:
```
#pip install -U scikit-fuzzy
```

Import useful Libraries

In [3]:
```
# For Complete Analysis, import the following useful libraries
import pandas as pd
import numpy as np
import math

# For Visualization, import the following libraries
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# For Data Preprocessing
# Import libraries for handling missing values
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

# Import libraries for encoding
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize

# Import library for balancing
from imblearn.over_sampling import SMOTE

# Import library for feature selection
from sklearn.feature_selection import SelectKBest, f_classif

# For Supervised Models
# Import the following libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from catboost import CatBoostClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc

# For Unsupervised Models
# Import the following libraries
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
```

```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
import skfuzzy as fuzz
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture
```

Load the dataset and display the first few rows to understand its structure.

In [4]:
```
patient_train_df = pd.read_csv(r"disease_train.csv")
patient_train_df.head()
```

Out[4]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medication_B	mental_health	mood_stabiliser	surgery	treatment_
0	PA1001	59	male	no	no	7.8	NaN	89.0	0.85	105.0	...	no	no	no	no	no	no	no	
1	PA1002	48	female	no	no	1.5	2.5	101.0	0.97	104.0	...	no	no	yes	no	no	yes	no	
2	PA1003	77	male	no	no	7.3	1.2	57.0	1.28	44.0	...	no	no	no	no	no	no	no	
3	PA1004	42	female	no	no	1.2	2.5	106.0	0.98	108.0	...	no	no	no	no	no	no	no	
4	PA1005	38	female	no	no	0.6	1.9	95.0	NaN	NaN	...	no	no	no	no	no	no	no	

5 rows × 24 columns

Data Exploration, Visualisations, and Summary:

In [5]:
```
#Learn about the dataset's datatypes, total rows and columns, and whether or not null values are available.
patient_train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4250 entries, 0 to 4249
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   id               4250 non-null   object 
 1   age              4250 non-null   int64  
 2   gender           4109 non-null   object 
 3   sick             4250 non-null   object 
 4   pregnant         4250 non-null   object 
 5   test_X1          3839 non-null   float64
 6   test_X2          3007 non-null   float64
 7   test_X3          4034 non-null   float64
 8   test_X4          3858 non-null   float64
 9   test_X5          3863 non-null   float64
 10  test_X6          154 non-null    float64
 11  concern_type1    4250 non-null   object 
 12  concern_type2    4250 non-null   object 
 13  enlargement      4250 non-null   object 
 14  tumor            4250 non-null   object 
 15  disorder         4250 non-null   object 
 16  medication_A     4250 non-null   object 
 17  medication_B     4250 non-null   object 
 18  mental_health    4250 non-null   object 
 19  mood_stabiliser  4250 non-null   object 
 20  surgery          4250 non-null   object 
 21  treatment_type1  4250 non-null   object 
 22  suspect          4250 non-null   object 
 23  target           4250 non-null   object 
dtypes: float64(6), int64(1), object(17)
memory usage: 797.0+ KB
```

#Observation: Dataset have 3 datatypes : float64 (6nos.),int64(1nos.) and object(17nos.). Dataset have 4250 rows(0 to 4249) and 24 columns. Dataset have no null values.

In [6]:
```python
# Summary statistics for numerical columns from the dataset
patient_train_df.describe()
```

Out[6]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6
count	4250.000000	3839.000000	3007.000000	4034.000000	3858.000000	3863.000000	154.000000
mean	67.374824	7.342463	2.035580	104.919623	0.970846	110.090834	23.325974
std	1004.518821	32.657963	0.920404	35.496255	0.162474	39.837621	5.317032
min	1.000000	0.005000	0.050000	2.000000	0.250000	1.400000	8.400000
25%	37.000000	0.600000	1.600000	87.000000	0.870000	92.000000	20.000000
50%	55.000000	1.500000	1.900000	102.000000	0.960000	107.000000	24.000000
75%	67.000000	3.000000	2.300000	121.000000	1.060000	125.000000	27.000000
max	65526.000000	530.000000	18.000000	430.000000	1.960000	642.000000	45.000000

In [7]:
```python
# Summary statistics for categorical columns from the dataset
patient_train_df.describe(include='object')
```

Out[7]:

	id	gender	sick	pregnant	concern_type1	concern_type2	enlargement	tumor	disorder	medication_A	medication_B	mental_health	mood_stabiliser	surgery	treatment_type
count	4250	4109	4250	4250	4250	4250	4250	4250	4250	4250	4250	4250	4250	4250	425
unique	4250	2	2	2	2	2	2	2	1	2	2	2	2	2	
top	PA1001	female	no	no	no	no	no	no	no	no	no	no	no	no	n
freq	1	2787	4095	4235	4183	3905	4215	4139	4250	3760	4196	4056	4205	4188	416

In [8]:
```python
# Check missing values from the dataset
missing_data = patient_train_df.isnull().sum()
print("Summary of Missing values (Descending order):")
print("Column_Name"," "*7,"Missing Value")
for column, count in missing_data.sort_values(ascending=False).items():
    print(f"{column.ljust(20)} {count}")
```

```
Summary of Missing values (Descending order):
Column_Name         Missing Value
test_X6              4096
test_X2              1243
test_X1              411
test_X4              392
test_X5              387
test_X3              216
gender               141
id                   0
medication_A         0
suspect              0
treatment_type1      0
surgery              0
mood_stabiliser      0
mental_health        0
medication_B         0
concern_type2        0
disorder             0
tumor                0
enlargement          0
age                  0
concern_type1        0
pregnant             0
sick                 0
target               0
```

#From the above table 1.Gender: There are 141 missing values out of 4250 observations. 2.Test_X1 to Test_X6: These columns also have missing values. Among them, Test_X6 has the highest number of missing values, with 4096 out of 4250 observations being missing. 3.Rest of columns have no missing values. Handling missing values in these columns is crucial for ensuring the integrity of the dataset and the accuracy of any subsequent analyses or models. Depending on the context, strategies such as imputation or deletion may be applied to address these missing values

In [9]:
```python
# Drop id column for further anaysis
patient_train_df.drop('id',axis=1,inplace=True)
patient_train_df.head()
```

Out[9]:

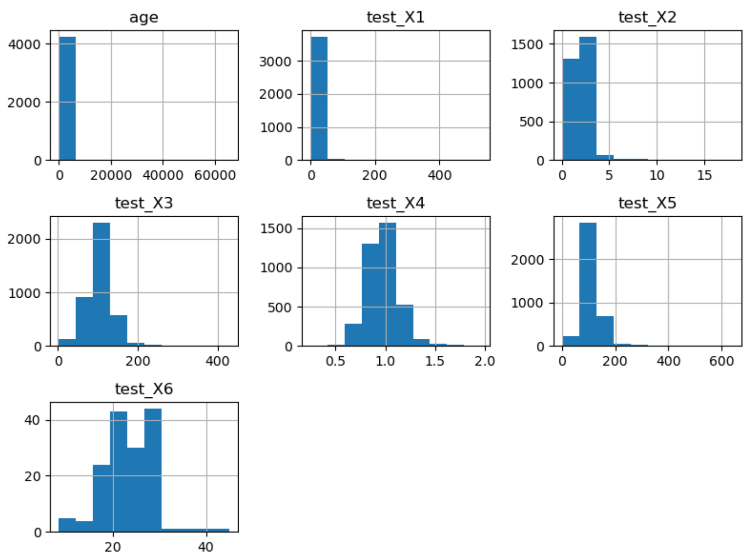
	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6	...	tumor	disorder	medication_A	medication_B	mental_health	mood_stabiliser	surgery	treatment
0	59	male	no	no	7.8	NaN	89.0	0.85	105.0	NaN	...	no	no	no	no	no	no	no	
1	48	female	no	no	1.5	2.5	101.0	0.97	104.0	NaN	...	no	no	yes	no	no	yes	no	
2	77	male	no	no	7.3	1.2	57.0	1.28	44.0	NaN	...	no	no	no	no	no	no	no	
3	42	female	no	no	1.2	2.5	106.0	0.98	108.0	27.0	...	no	no	no	no	no	no	no	
4	38	female	no	no	0.6	1.9	95.0	NaN	NaN	NaN	...	no	no	no	no	no	no	no	

5 rows × 23 columns

Visualizations for Numerical Columns in the preliminary investigation

Histograms for numerical columns:

In [10]:
```python
patient_train_df.hist(figsize=(8,6))
plt.tight_layout()
plt.show()
```

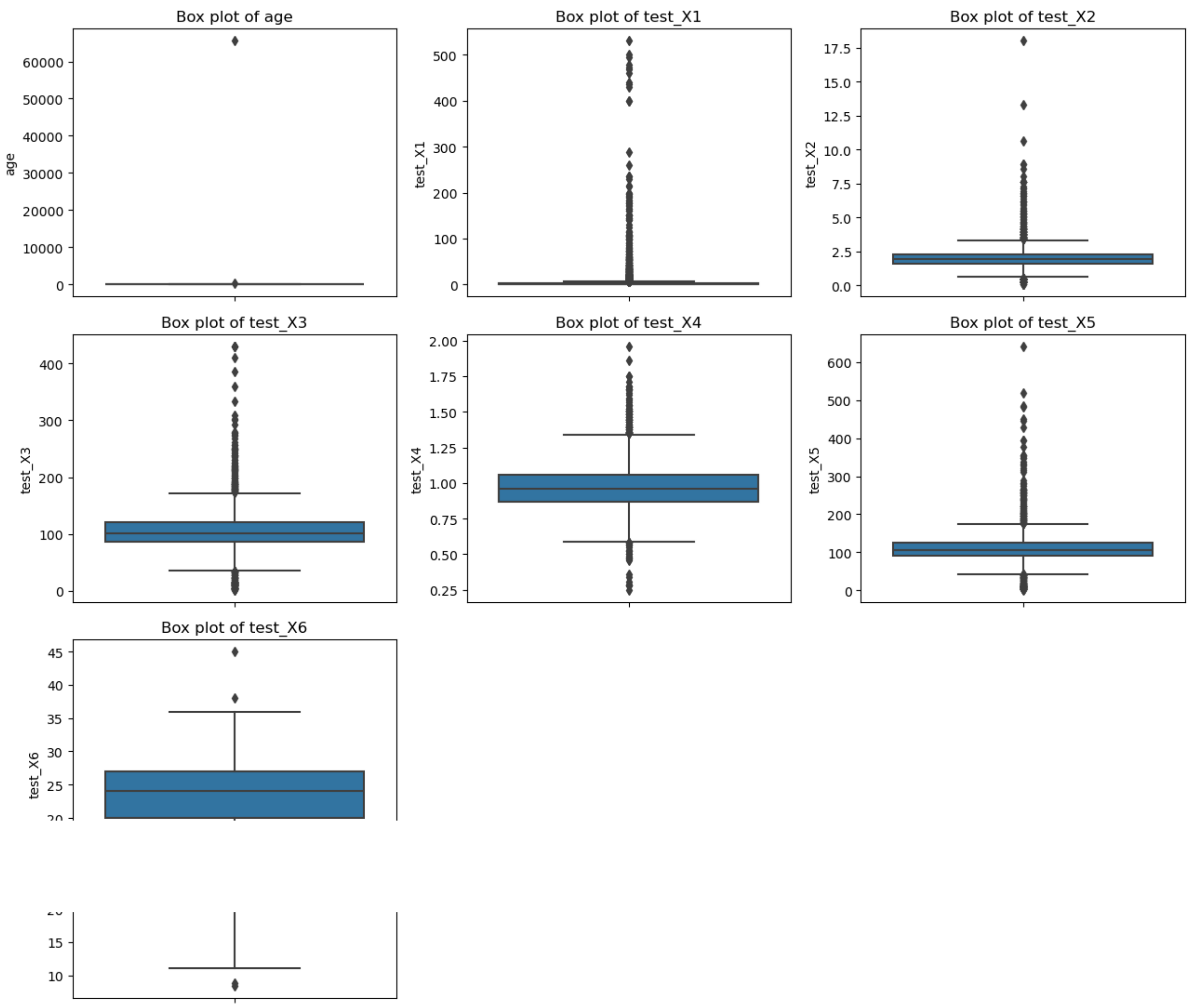


In [11]:
```python
# Seperate Numerical columns from Dataset for further anaysis
numerical_columns = patient_train_df.select_dtypes(include=['int64','float64']).columns.tolist()
numerical_columns
```

Out[11]:
```
['age', 'test_X1', 'test_X2', 'test_X3', 'test_X4', 'test_X5', 'test_X6']
```

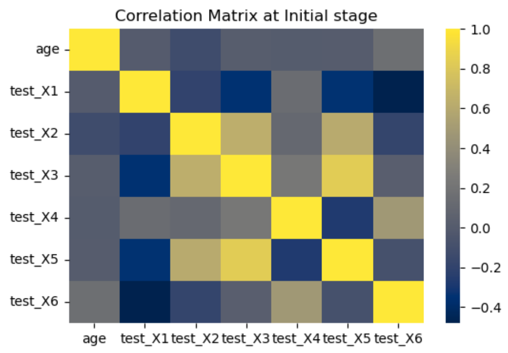
Box plots for numerical Columns

In [12]:
```python
# Create a new figure and axis object
plt.figure(figsize=(13, 10))
# for creating subplots
number_of_plots = len(numerical_columns)
number_of_columns = 3
# Calculate number of rows needed
number_of_rows = (number_of_plots // number_of_columns) + (number_of_plots % number_of_columns)

#To generate a box plot in a subplot, iterate over each numerical feature.
for i, feature in enumerate(numerical_columns, start=1):
    plt.subplot(number_of_rows, number_of_columns, i)
    sns.boxplot(y=patient_train_df[feature])
    plt.title(f'Box plot of {feature}')

#To prevent overlap,adjust the layout
plt.tight_layout()
plt.show()
```



Correlation matrix at Initial Visualization stage

```
In [13]: corr_matrix = patient_train_df[numerical_columns].corr()
         plt.figure(figsize=(6,4))
         sns.heatmap(corr_matrix,cmap='cividis',fmt=".2f")
         plt.title('Correlation Matrix at Initial stage')
         plt.show()
```



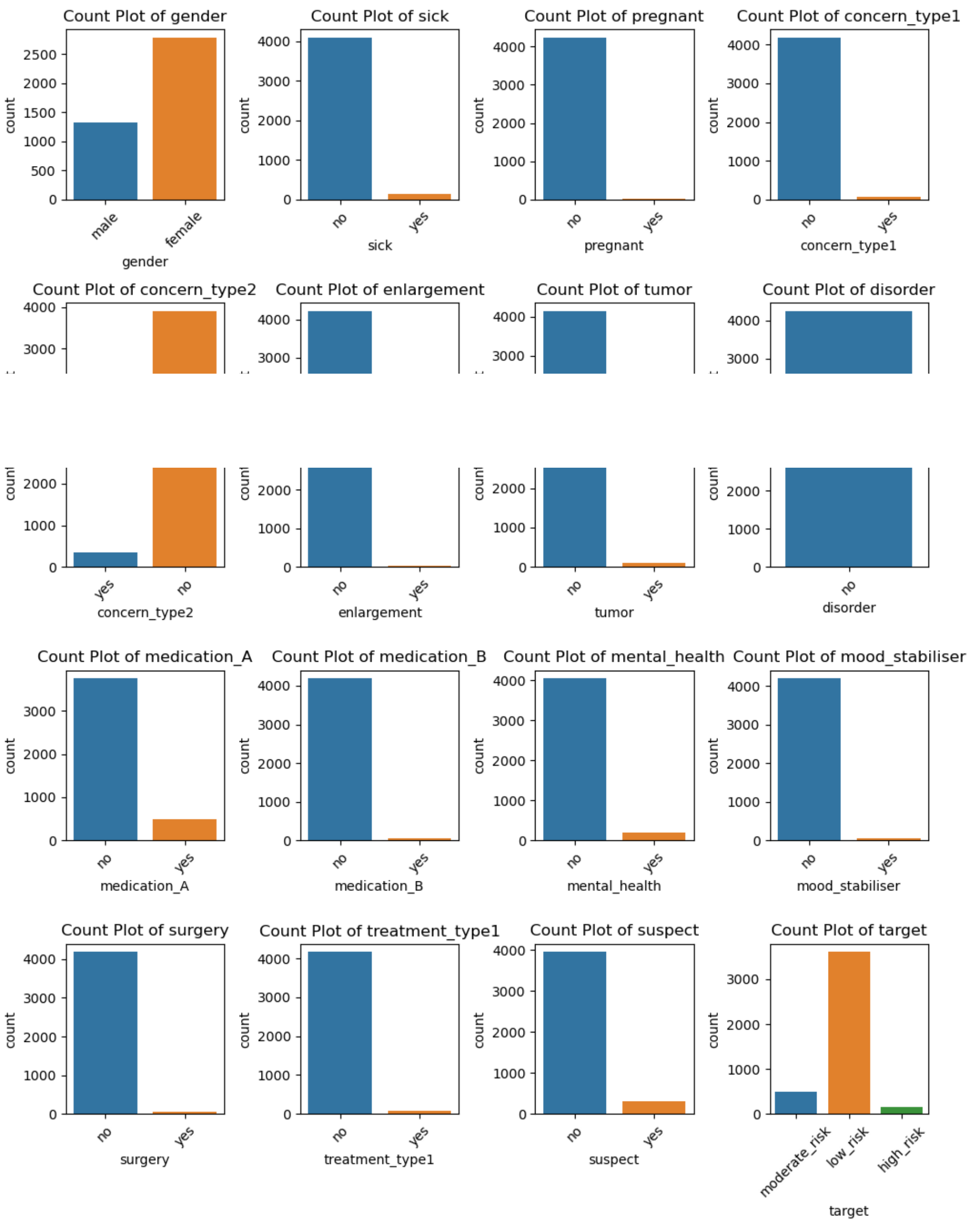
Visualizations for Categorical Columns in the preliminary investigation

```
In [14]: # Seperate Categorical columns from dataset for further anaysis
         categorical_columns = patient_train_df.select_dtypes(include=['object']).columns.tolist()
         categorical_columns
```

```
Out[14]: ['gender',
          'sick',
          'pregnant',
          'concern_type1',
          'concern_type2',
          'enlargement',
          'tumor',
          'disorder',
          'medication_A',
          'medication_B',
          'mental_health',
          'mood_stabiliser',
          'surgery',
          'treatment_type1',
          'suspect',
          'target']
```

Visualization through Count plots for categorical columns

```
In [15]: # Determine the number of rows and columns for subplots
         number_columns = 4
         number_rows = math.ceil(len(categorical_columns) / number_columns)

         # Create subplots for count plots
         plt.figure(figsize=(10, 3 * number_rows))
         for i, column in enumerate(categorical_columns, 1):
             plt.subplot(number_rows, number_columns, i)
             sns.countplot(x=column, data=patient_train_df)
             plt.title(f'Count Plot of {column}')
             plt.xticks(rotation=45)

         plt.tight_layout()
         plt.show()
```



Data Cleansing and Pre-processing

Handle missing values in Dataset

```
In [16]: # calculate percentage of missing values and suggest whose percentages > threshold vlaue

         threshold = 60  # Set threshold here
```

```
missing_percentage = ((patient_train_df.isnull().sum() / len(patient_train_df)) * 100).sort_values(ascending=False)
columns_to_drop = missing_percentage[missing_percentage > threshold].index.tolist()

print("Columns with missing values > {}%:".format(threshold))
for column in columns_to_drop:
    print("{}: {:.2f}% missing values.".format(column, missing_percentage[column]))
```

```
Columns with missing values > 60%:
test_X6: 96.38% missing values.
```

In [17]:
```
# Drop coulmns whose missing percentages > threshold values.
patient_train_df.drop(columns=columns_to_drop, inplace=True)
```

In [18]:
```
# Verify columns after droping missing value columns based on threshold value percentages
patient_train_df.columns
```

Out[18]:
```
Index(['age', 'gender', 'sick', 'pregnant', 'test_X1', 'test_X2', 'test_X3',
       'test_X4', 'test_X5', 'concern_type1', 'concern_type2', 'enlargement',
       'tumor', 'disorder', 'medication_A', 'medication_B', 'mental_health',
       'mood_stabiliser', 'surgery', 'treatment_type1', 'suspect', 'target'],
      dtype='object')
```

In [19]:
```
# After Droping highest missing value column based on threshold value,handle remaining missing values.
updated_numerical_col = patient_train_df.select_dtypes(include=['int64','float64']).columns.tolist()
updated_numerical_col
```

Out[19]:
```
['age', 'test_X1', 'test_X2', 'test_X3', 'test_X4', 'test_X5']
```

In [20]:
```
# For Numerical columns: Impute missing values with median
numeric_imputer = SimpleImputer(strategy="median")
patient_train_df[updated_numerical_col] = numeric_imputer.fit_transform(patient_train_df[updated_numerical_col])

# For Categorical column 'gender': Impute missing values with mode
categorical_imputer = SimpleImputer(strategy="most_frequent")
patient_train_df['gender'] = categorical_imputer.fit_transform(patient_train_df[['gender']]).ravel()

# Check if there are any remaining missing values
remaining_missing_values = patient_train_df.isnull().sum().sum()
if remaining_missing_values == 0:
    print("All missing values have been handled.")
else:
    print(f"There are still {remaining_missing_values} missing values after imputation.")
```

```
All missing values have been handled.
```

Handle outliers in Dataset

In [21]:
```
# Handle outliers in numerical columns using IQR
Q1 = patient_train_df[updated_numerical_col].quantile(0.25)
Q3 = patient_train_df[updated_numerical_col].quantile(0.75)
IQR = Q3 - Q1
def winsorize(column):
    lower_bound = Q1[column] - 1.5 * IQR[column]
    upper_bound = Q3[column] + 1.5 * IQR[column]
    column_values = patient_train_df[column].copy()
    column_values[column_values < lower_bound] = lower_bound
    column_values[column_values > upper_bound] = upper_bound
    return column_values

# Apply winsorization transformation to all numerical columns
for column in updated_numerical_col:
    patient_train_df[column] = winsorize(column)
```

Handle Constant in Dataset

In [22]:
```
#find constant column if any in dataset
constant_columns = []
for col in patient_train_df.columns:
    if patient_train_df[col].nunique() == 1:
        constant_columns.append(col)

# Print constant columns
print("Constant columns:", constant_columns)
```

```
Constant columns: ['disorder']
```

In [23]:
```
# Drop Constant column from column list
patient_train_df.drop(columns=constant_columns, inplace=True)
```

In [24]:
```
# check remaining colmns in dataframe after droping 'disorder' colum from dataset
patient_train_df.columns
patient_train_df.shape
```

Out[24]:
```
(4250, 21)
```

Handle Duplicates in Dataset

In [25]:
```
# handle Duplicated Rows in dataset if any
duplicate_rows = patient_train_df[patient_train_df.duplicated()]
duplicate_rows
# Drop Duplicate values from dataset
patient_train_df.drop_duplicates(inplace=True)
patient_train_df.shape
```

Out[25]:
```
(4208, 21)
```

From the dataset of 4250 , 42 records found duplicate

In [26]:
```
# After handling Constant and remove target column, updated categorical col
update_categorical_col = patient_train_df.select_dtypes(include=['object']).columns.tolist()
update_categorical_col.remove('target')
update_categorical_col
```

```
Out[26]: ['gender',
          'sick',
          'pregnant',
          'concern_type1',
          'concern_type2',
          'enlargement',
          'tumor',
          'medication_A',
          'medication_B',
          'mental_health',
          'mood_stabiliser',
          'surgery',
          'treatment_type1',
          'suspect']
```

```
In [27]: #Check the number of categories and the count of data available in each category for the 'target' column
         patient_train_df['target'].value_counts()
```

```
Out[27]: target
         low_risk         3570
         moderate_risk     489
         high_risk         149
         Name: count, dtype: int64
```

Encoding the Categorical columns of the Dataset

```
In [28]: # Encoding on update_categorical_col for further analysis
         patient_train_df_encoded = pd.get_dummies(patient_train_df,columns=update_categorical_col,drop_first=True)
```

```
In [29]: # label encoder for target column because it has three categories like low risk, modarate risk and high risk

         le = LabelEncoder()
         patient_train_df_encoded['target'] = le.fit_transform(patient_train_df_encoded['target'])
```

```
In [30]: # Check dataset after encoding
         patient_train_df_encoded = patient_train_df_encoded.astype(int)
         patient_train_df_encoded.head()
```

Out[30]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	target	gender_male	sick_yes	pregnant_yes	...	concern_type2_yes	enlargement_yes	tumor_yes	medication_A_yes	medication_B_y
0	59	5	1	89	0	105	2	1	0	0	...	1	0	0	0	
1	48	1	2	101	0	104	1	0	0	0	...	0	0	0	1	
2	77	5	1	57	1	50	2	1	0	0	...	0	0	0	0	
3	42	1	2	106	0	108	1	0	0	0	...	0	0	0	0	
4	38	0	1	95	0	107	1	0	0	0	...	0	0	0	0	

5 rows × 21 columns

```
In [31]: # From the above count plot for categorical columns, we identify that 'target' column has imbalnce data.
         # visualize encoding of 'target' column after encoding.
         patient_train_df_encoded['target'].value_counts()
```

```
Out[31]: target
         1    3570
         2     489
         0     149
         Name: count, dtype: int64
```

Define Variable X and y for further anaysis.

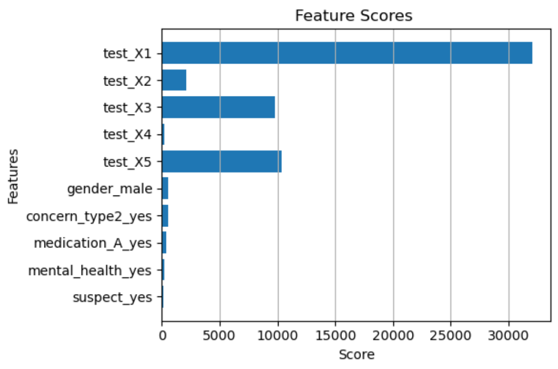
```
In [32]: X = patient_train_df_encoded.drop(columns=['target'])  # Features
         y = patient_train_df_encoded['target']  # Target variable
```

Balance data for Training Model

```
In [33]: #To balance the dataset,apply SMOTE.
         smote = SMOTE(random_state=42)
         X_rsmpd, y_rsmpd = smote.fit_resample(X, y)
```

Feature selection for further analysis

```
In [34]: # Perform feature selection using SelectKBest
         f_s = SelectKBest(score_func=f_classif, k=10)  # change k as per need
         # if want to check on imbalance data change X_rsmpd to X and y_smpd to y
         X_selected = f_s.fit_transform(X_rsmpd, y_rsmpd)

         # Get scores and p-values of features
         f_scores = f_s.scores_
         f_p_values = f_s.pvalues_

         # Get the names of selected features
         sel_fea_names = X_rsmpd.columns[f_s.get_support()]

         # Reverse the order of features and scores
         sel_fea_names = sel_fea_names[::-1]
         f_scores = f_scores[f_s.get_support()][::-1]

         # Plotting the scores of features
         plt.figure(figsize=(6, 4))
         plt.barh(sel_fea_names, f_scores)
         plt.xlabel('Score')
         plt.ylabel('Features')
         plt.title('Feature Scores ')
         plt.grid(axis='x')
         plt.tight_layout()
         plt.show()

         # Transform the original dataset using the selected features
         X_selected_df = pd.DataFrame(X_selected, columns=sel_fea_names)
```



Follow these instructions to train models using different datasets: 1.For training models with imbalanced data: Use independent variables X and the target variable y. 2.For training models with balanced data: Use independent variables X_rsmpd and the target variable y_rsmpd. 3.For training models with feature-selected data: Use independent variables X_selected_df and the dependent target variable y. Ensure to choose the appropriate dataset according to your specific training requirements.

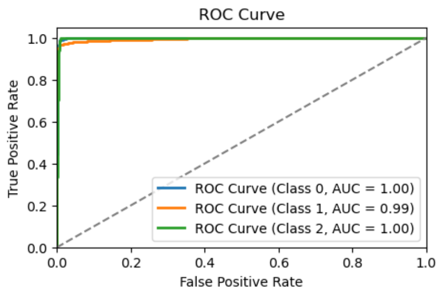
Supervised Model Training, Tuning and Evalualte and On Best Model Create Confusion Matrix

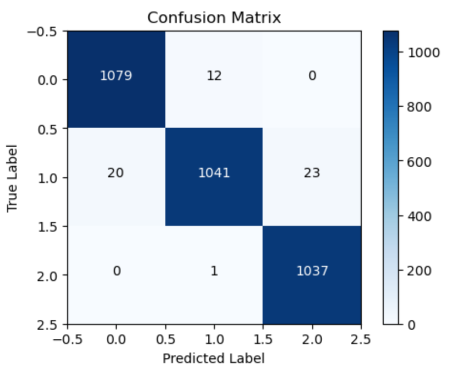
and Roc-Auc Curve and generate csv file for predicted lables

In [35]:
```
# Define models and their respective parameter grids for tuning
models = {
    "KNN": (KNeighborsClassifier(), {'n_neighbors': [3, 5, 7]}),
    "SVM": (SVC(), {'C': [1, 10, 100], 'gamma': [0.1, 0.01, 0.001]}),
    "Decision Tree": (DecisionTreeClassifier(), {'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10]}),
    "Random Forest": (RandomForestClassifier(), {'n_estimators': [100, 200, 300],'max_depth': [None, 10, 20],'min_samples_split': [2, 5, 10]}),
    "CatBoost": (CatBoostClassifier(verbose=False), {'iterations': [100, 200, 300],'learning_rate': [0.01, 0.05, 0.1]})
}

best_model_name = ""
best_accuracy = 0

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_rsmpd, y_rsmpd, test_size=0.3, random_state=42)

# Train and evaluate models
for name, (model, param_grid) in models.items():
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train, y_train)
    y_pred = grid_search.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {accuracy}")
    print(f"{name} Classification Report:")
    print(classification_report(y_test, y_pred))
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model_name = name
        best_model = grid_search.best_estimator_

print(f"\nBest Model: {best_model_name}")
print(f"Accuracy: {best_accuracy}")

# Generate  ROC curve and confusion matrix for the best model
y_pred_proba = best_model.predict_proba(X_test)

# Compute ROC curve and ROC AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
num_classes = len(best_model.classes_)
for i in range(num_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test == i, y_pred_proba[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for the best model
plt.figure(figsize=(5, 3))
for i in range(num_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'ROC Curve (Class {i}, AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()


# Generate confusion matrix for the best model
y_pred = best_model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix with values
plt.figure(figsize=(6, 4))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)

# Add value annotations
thresh = conf_matrix.max() / 2.
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        plt.text(j, i, format(conf_matrix[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if conf_matrix[i, j] > thresh else "black")

plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Predict target values for X_test using the best model
predicted_target = best_model.predict(X_test)
```

```python
# Decode the predicted target labels
predicted_target_original = le.inverse_transform(predicted_target)

# Create a DataFrame with the decoded predicted target values
predicted_df = pd.DataFrame(predicted_target_original, columns=['Predicted_Target'])

# Save the DataFrame to a CSV file
predicted_df.to_csv('predictedTarget.csv', index=False)
```

```
KNN Accuracy: 0.9321506380329909
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      1091
           1       0.94      0.85      0.89      1084
           2       0.91      0.98      0.94      1038

    accuracy                           0.93      3213
   macro avg       0.93      0.93      0.93      3213
weighted avg       0.93      0.93      0.93      3213

SVM Accuracy: 0.9788359788359788
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      1091
           1       0.98      0.95      0.97      1084
           2       0.98      1.00      0.99      1038

    accuracy                           0.98      3213
   macro avg       0.98      0.98      0.98      3213
weighted avg       0.98      0.98      0.98      3213

Decision Tree Accuracy: 0.9757236227824463
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1091
           1       0.97      0.96      0.96      1084
           2       0.98      0.99      0.98      1038

    accuracy                           0.98      3213
   macro avg       0.98      0.98      0.98      3213
weighted avg       0.98      0.98      0.98      3213

Random Forest Accuracy: 0.9819483348895114
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1091
           1       0.99      0.96      0.97      1084
           2       0.98      1.00      0.99      1038

    accuracy                           0.98      3213
   macro avg       0.98      0.98      0.98      3213
weighted avg       0.98      0.98      0.98      3213

CatBoost Accuracy: 0.9825708061002179
CatBoost Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1091
           1       0.99      0.96      0.97      1084
           2       0.98      1.00      0.99      1038

    accuracy                           0.98      3213
   macro avg       0.98      0.98      0.98      3213
weighted avg       0.98      0.98      0.98      3213

Best Model: CatBoost
Accuracy: 0.9825708061002179
```





```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/preprocessing/_label.py:155: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Unsupervised Learning using Clustering Algorithm

In [36]:
```
#Prior to doing unsupervised learning, make sure the dataset has undergone pretreatment
# by handling missing values and outliers.
#Manage duplicates, handle constants, and encode categorical variables
patient_train_df_encoded.head()
```

Out[36]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	target	gender_male	sick_yes	pregnant_yes	...	concern_type2_yes	enlargement_yes	tumor_yes	medication_A_yes	medication_B_y
0	59	5	1	89	0	105	2	1	0	0	...	1	0	0	0	
1	48	1	2	101	0	104	1	0	0	0	...	0	0	0	1	
2	77	5	1	57	1	50	2	1	0	0	...	0	0	0	0	
3	42	1	2	106	0	108	1	0	0	0	...	0	0	0	0	
4	38	0	1	95	0	107	1	0	0	0	...	0	0	0	0	

5 rows × 21 columns

In [37]:
```
# For Unsupervised learning using clustering exclude medical diagnosis ('target')
usc_patient_df = patient_train_df_encoded.drop(['target'], axis=1)
usc_patient_df.columns
```

Out[37]:
```
Index(['age', 'test_X1', 'test_X2', 'test_X3', 'test_X4', 'test_X5',
       'gender_male', 'sick_yes', 'pregnant_yes', 'concern_type1_yes',
       'concern_type2_yes', 'enlargement_yes', 'tumor_yes', 'medication_A_yes',
       'medication_B_yes', 'mental_health_yes', 'mood_stabiliser_yes',
       'surgery_yes', 'treatment_type1_yes', 'suspect_yes'],
      dtype='object')
```

Noramlize Data for further anaysis

In [38]:
```
# Initialize StandardScaler
scaler = StandardScaler()

# Standardize the data
scaled_data = scaler.fit_transform(usc_patient_df)

# Convert the standardized data back to a DataFrame
scaled_df = pd.DataFrame(scaled_data, columns=usc_patient_df.columns)
print("scaled_df shape:",scaled_df.shape)
scaled_df.head()
```

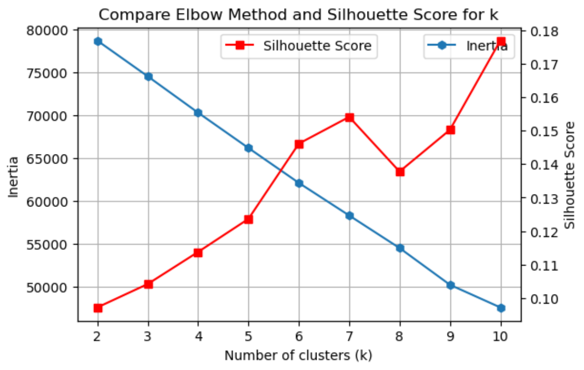
```
scaled_df shape: (4208, 20)
```

Out[38]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	gender_male	sick_yes	pregnant_yes	concern_type1_yes	concern_type2_yes	enlargement_yes	tumor_yes	medicatic
0	0.374479	1.987737	-0.734259	-0.546479	-0.778922	-0.129863	1.481601	-0.195559	-0.059811	-0.127199	3.362156	-0.091582	-0.164599	
1	-0.212414	-0.343315	1.361917	-0.106339	-0.778922	-0.169438	-0.674946	-0.195559	-0.059811	-0.127199	-0.297428	-0.091582	-0.164599	
2	1.334850	1.987737	-0.734259	-1.720186	1.283825	-2.306530	1.481601	-0.195559	-0.059811	-0.127199	-0.297428	-0.091582	-0.164599	
3	-0.532538	-0.343315	1.361917	0.077052	-0.778922	-0.011135	-0.674946	-0.195559	-0.059811	-0.127199	-0.297428	-0.091582	-0.164599	
4	-0.745953	-0.926078	-0.734259	-0.326409	-0.778922	-0.050711	-0.674946	-0.195559	-0.059811	-0.127199	-0.297428	-0.091582	-0.164599	

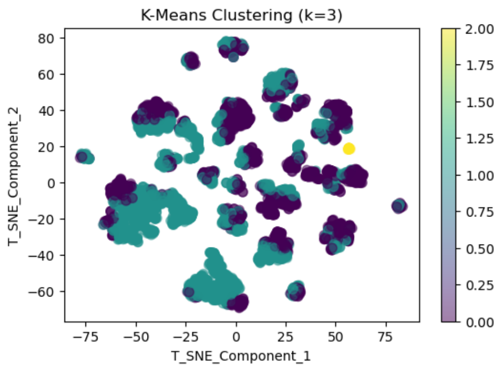
Define K-value based on Elbow Method and Silhouette Score

In [39]:
```
# Create initial lists to hold silhouette scores and inertia for various values of k.
inertia = []
silhouette_scores = []

# Experiment with several values of k and determine the silhouette score and inertia for each value.
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42,n_init=7)
    kmeans.fit(scaled_df)
    inertia.append(kmeans.inertia_)

    kmeans_labels = kmeans.predict(scaled_df)
    silhouette_scores.append(silhouette_score(scaled_df, kmeans_labels))

# Plot both the elbow curve and silhouette scores on the same graph
plt.figure(figsize=(6, 4))

# Plot the elbow curve
plt.plot(range(2, 11), inertia, marker='h', label='Inertia')
plt.title('Compare Elbow Method and Silhouette Score for k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.xticks(range(2, 11))
plt.grid(True)
plt.legend(loc='upper right')

# Plot the silhouette scores
plt.twinx()
plt.plot(range(2, 11), silhouette_scores, marker='s', color='r', label='Silhouette Score')
plt.ylabel('Silhouette Score')
plt.legend(loc='upper center')

plt.show()
```



Perform K-means clustering

In [40]:
```
# Perform k-means clustering with k=3 based on silhouette score and elbow graph
kmeans = KMeans(n_clusters=3, random_state=42,n_init=10)

scaled_df['kmeans_labels'] = kmeans.fit_predict(scaled_df)

# Apply dimensionality reduction for visualization
tsne = TSNE(n_components=2, random_state=42)
tsne_results = tsne.fit_transform(scaled_df.drop('kmeans_labels', axis=1))

# Plotting t-SNE results with k-Means cluster labels
plt.figure(figsize=(6, 4))
plt.scatter(tsne_results[:, 0],tsne_results[:, 1],c=scaled_df['kmeans_labels'],cmap='viridis',s=50,marker='o',alpha=0.5)
plt.title('K-Means Clustering (k=3)')
plt.xlabel('T_SNE_Component_1')
plt.ylabel('T_SNE_Component_2')
plt.colorbar()
plt.show()
```



In [41]:
```
# Calculate the cluster centroids
cluster_centroids = kmeans.cluster_centers_

# Convert centroids into a dataframe for better readability
centroids_df = pd.DataFrame(scaler.inverse_transform(cluster_centroids),columns=scaled_df.columns[:-1])

centroids_df
```

Out[41]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	gender_male	sick_yes	pregnant_yes	concern_type1_yes	concern_type2_yes	enlargement_yes	tumor_yes	medicatio
0	51.538421	0.857292	1.539467	124.470988	0.490329	124.557240	0.243596	0.029273	1.691355e-17	0.018296	0.126503	0.009932	0.035546	
1	52.479386	2.210965	1.190351	86.526316	0.280263	94.654825	0.373246	0.043421	2.168404e-17	0.013596	0.040789	0.007018	0.017982	
2	32.733333	0.400000	1.533333	121.000000	0.800000	103.800000	0.000000	0.000000	1.000000e+00	0.066667	0.466667	0.000000	0.133333	

In [42]:
```
# Count the number of instances in each cluster
cluster_counts = scaled_df['kmeans_labels'].value_counts().sort_index()

cluster_counts
```

Out[42]:
```
kmeans_labels
0    1906
1    2287
2      15
Name: count, dtype: int64
```

Hierarchical Clsutering

In [43]:
```
# Preparing the dataset (excluding the kmeans_labels column if it exists)
hierarchical_c_df = scaled_df.drop('kmeans_labels', axis=1, errors='ignore')
```

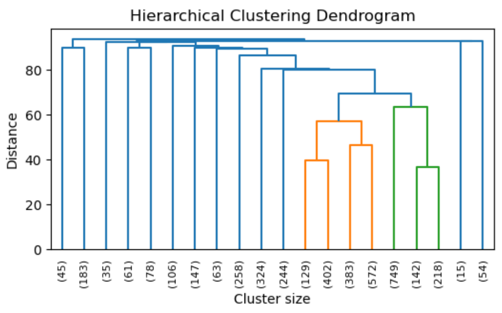
```
# Applying Hierarchical Clustering with 3 clusters
hierarchical_clustering = AgglomerativeClustering(n_clusters=3, linkage='ward')
hierarchical_c_df['hierarchical_labels'] = hierarchical_clustering.fit_predict(hierarchical_c_df)

# Count the number of instances in each cluster
hierarchical_cluster_counts = hierarchical_c_df['hierarchical_labels'].value_counts().sort_index()
hierarchical_cluster_counts
```

Out[43]:

```
hierarchical_labels
0      69
1     228
2    3911
Name: count, dtype: int64
```

In [44]:

```
# Generating the linkage matrix
linkage_matrix_d =linkage(hierarchical_c_df.drop('hierarchical_labels',axis=1),method='ward',metric='euclidean')

# Plotting the dendrogram
plt.figure(figsize=(6,3))
dendrogram(linkage_matrix_d, truncate_mode='lastp', p=20, leaf_rotation=90., leaf_font_size=8.)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Cluster size')
plt.ylabel('Distance')
plt.show()
```



Fuzzy C-Means

For Fuzzy C-means first install following pip install -U scikit-fuzzy

In [45]:

```
# Assuming df_normalized is your normalized DataFrame and we are excluding any non-feature columns
# Transpose to match skfuzzy input
X = scaled_df.drop(['kmeans_labels', 'hierarchical_labels'], axis=1, errors='ignore').values.T

# Number of clusters and fuzziness parameter
num_clusters_fuz_para = 3
m_fuz_para = 2  # Common choice for fuzziness parameter
```

```
# Apply Fuzzy C-Means
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    X, c=num_clusters_fuz_para, m=m_fuz_para, error=0.005, maxiter=1000, init=None)

# cntr: Cluster centers
# u: Final fuzzy partitioned matrix (membership grades)
# fpc: Fuzzy partition coefficient (performance indicator)
'''
- Determine Cluster Membership:The code uses np.argmax(u, axis=0) to determine the cluster membership of
  each data point based on the highest membership grade.
- Count Data Points per Cluster:It then counts the number of data points assigned to each cluster
  using np.unique() and np.bincount().
- Visualization:PCA is applied to reduce the dimensionality of the data to 2D for visualization purposes.
  A scatter plot is created where each data point is plotted in the reduced 2D space, with different colors
  representing different clusters.
- Labels and Legend:Labels are added to the plot to indicate the clusters.A legend is included to
  explain the colors used for each cluster.
  Overall, this code should provide a visual representation of the Fuzzy C-Means clustering results,
  allowing you to observe the clusters in a 2D scatter plot. Adjustments can be made to the plot's appearance,
  such as colors, markers, and titles, based on your preferences and requirements.
'''
```

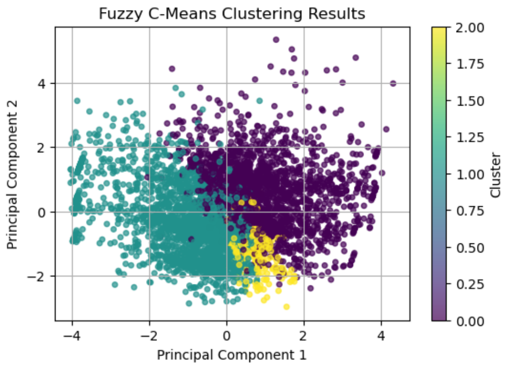
Out[45]:

```
"\n- Determine Cluster Membership:The code uses np.argmax(u, axis=0) to determine the cluster membership of \n  each data point based on the highest membership grade.\
n- Count Data Points per Cluster:It then counts the number of data points assigned to each cluster \n  using np.unique() and np.bincount().\n- Visualization:PCA is app
lied to reduce the dimensionality of the data to 2D for visualization purposes.\n  A scatter plot is created where each data point is plotted in the reduced 2D space,
with different colors \n  representing different clusters.\n- Labels and Legend:Labels are added to the plot to indicate the clusters.A legend is included to\n  explai
n the colors used for each cluster.\n  Overall, this code should provide a visual representation of the Fuzzy C-Means clustering results, \n  allowing you to observe t
he clusters in a 2D scatter plot. Adjustments can be made to the plot's appearance, \n  such as colors, markers, and titles, based on your preferences and requirement
s.\n"
```

In [46]:

```
# Step 1: Determine cluster membership
# consider 'u' is the matrix of membership grades from Fuzzy C-Means
cluster_membership = np.argmax(u, axis=0)

# Step 2: Counting Data Points per Cluster
unique, counts = np.unique(cluster_membership, return_counts=True)
cluster_counts_series = pd.Series(counts, index=unique, name='count')
print("Cluster counts:")
print(cluster_counts_series)

# Step 3: Visualization
# Using PCA for dimensionality reduction for visualization purposes
pca_f_c = PCA(n_components=2)
X_reduced = pca_f_c.fit_transform(scaled_df.drop(['kmeans_labels','hierarchical_labels'],axis=1,errors='ignore'))

# Calculate marker size based on cluster size
marker_size = 15

# Scatter plot of the first two principal components
plt.figure(figsize=(6, 4))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=cluster_membership, cmap='viridis', s=marker_size, alpha=0.7)
plt.colorbar(label='Cluster')
plt.title('Fuzzy C-Means Clustering Results')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```

```
Cluster counts:
0    1854
1    2113
2     241
Name: count, dtype: int64
```

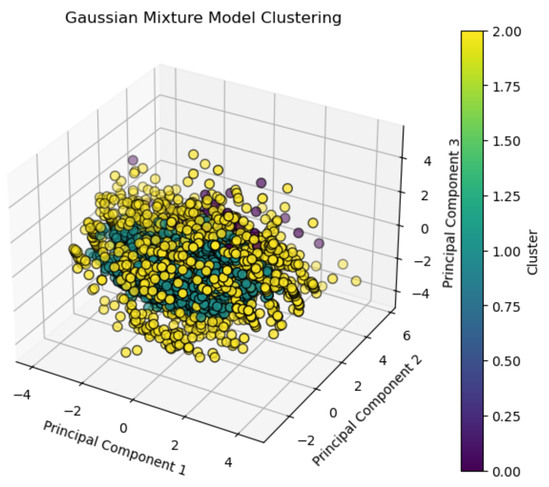


Gaussian Mixture Model

In [47]:
```
# Assuming scaled_df is your scaled dataset
scaled_df_gmm = scaled_df.drop('kmeans_labels', axis=1)

# Assuming n_components is the number of components for GMM
n_components_value = 3

# Initialize Gaussian Mixture Model
gmm = GaussianMixture(n_components=n_components_value,covariance_type="spherical",init_params='k-means++',random_state=42).fit(scaled_df_gmm)

# Fit GMM to your data
gmm.fit(scaled_df_gmm)

# Get cluster labels
cluster_labels = gmm.predict(scaled_df_gmm)

# Count the number of instances in each cluster
cluster_counts = pd.Series(cluster_labels).value_counts().sort_index()

# Display cluster counts with their corresponding labels
print("Number of instances in each cluster:")
print(cluster_counts)
```

```
# Perform PCA for visualization (assuming 3 components for 3D plot)
pca = PCA(n_components=3)
scaled_df_pca = pca.fit_transform(scaled_df_gmm)

# Plot the clusters in 3D
fig = plt.figure(figsize=(10,6))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
scatter = ax.scatter(scaled_df_pca[:, 0],scaled_df_pca[:, 1],scaled_df_pca[:, 2],c=cluster_labels,cmap='viridis',edgecolor='k',s=50)

# Add labels and title
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
plt.title('Gaussian Mixture Model Clustering')

# Add color bar
plt.colorbar(scatter, ax=ax, label='Cluster')

# Show plot
plt.show()
```

```
Number of instances in each cluster:
0      59
1    2604
2    1545
Name: count, dtype: int64
```



Evaluation of Different Model and their values for comparision

In [48]:
```python
# Extracting cluster labels for silhouette score calculation
labels_kmeans = scaled_df['kmeans_labels'].values
labels_hierarchical = hierarchical_c_df['hierarchical_labels'].values
labels_fuzzy = np.argmax(u, axis=0)  # Assuming 'u' represents membership grades for Fuzzy C-Means
labels_gmm = gmm.predict(scaled_df_gmm)  # Assuming 'gmm' is your Gaussian Mixture Model

# Silhouette scores
silhouette_kmeans = silhouette_score(scaled_df.drop(['kmeans_labels', 'hierarchical_labels'], axis=1, errors='ignore'),labels_kmeans)
silhouette_hierarchical =silhouette_score(scaled_df.drop(['kmeans_labels','hierarchical_labels'],axis=1, errors='ignore'),labels_hierarchical)
silhouette_fuzzy = silhouette_score(scaled_df.drop(['kmeans_labels', 'hierarchical_labels'], axis=1, errors='ignore'),labels_fuzzy)
silhouette_gmm = silhouette_score(scaled_df_gmm, labels_gmm)

# Assuming df_normalized is your dataset and you have labels from k-Means, Hierarchical,and Fuzzy C-Means clustering
X = scaled_df.drop(['kmeans_labels', 'hierarchical_labels'], axis=1, errors='ignore')

# Davies-Bouldin Index
davies_bouldin_kmeans = davies_bouldin_score(X, labels_kmeans)
davies_bouldin_hierarchical = davies_bouldin_score(X, labels_hierarchical)
davies_bouldin_fuzzy = davies_bouldin_score(X, np.argmax(u, axis=0))
davies_bouldin_gmm = davies_bouldin_score(X, labels_gmm)

# Calinski-Harabasz Index
calinski_harabasz_kmeans = calinski_harabasz_score(X, labels_kmeans)
calinski_harabasz_hierarchical = calinski_harabasz_score(X, labels_hierarchical)
calinski_harabasz_fuzzy = calinski_harabasz_score(X, np.argmax(u, axis=0))
calinski_harabasz_gmm = calinski_harabasz_score(X,labels_gmm)


print('Evaluation cluster quality with diffrrent models and methods:')
print('\n')
print(f"Silhouette Score Comparison:")
print(f"For k-Means                 : {silhouette_kmeans:.3f}")
print(f"For Hierarchical Clustering : {silhouette_hierarchical:.3f}")
print(f"For Fuzzy C-Means           : {silhouette_fuzzy:.3f}")
print(f"For Gaussian Mixture Model  : {silhouette_gmm:.3f}")
print('\n')
print("\nDavies-Bouldin Index Comparison:")
print(f"For K-Means                 : {davies_bouldin_kmeans:.3f}")
print(f"For Hierarchical Clustering : {davies_bouldin_hierarchical:.3f}")
print(f"For Fuzzy C-Means           : {davies_bouldin_fuzzy:.3f}")
print(f"For Gaussian Mixture Model  : {davies_bouldin_gmm:.3f}")
print('\n')
print("Calinski-Harabasz Index Comparison:")
print(f"For K-Means                 : {calinski_harabasz_kmeans:.3f}")
print(f"For Hierarchical Clustering : {calinski_harabasz_hierarchical:.3f}")
print(f"For Fuzzy C-Means           : {calinski_harabasz_fuzzy:.3f}")
print(f"For Gaussian Mixture Model  : {calinski_harabasz_gmm:.3f}")
```

```
Evaluation cluster quality with diffrrent models and methods:


Silhouette Score Comparison:
For k-Means                 : 0.098
For Hierarchical Clustering : 0.306
For Fuzzy C-Means           : 0.018
For Gaussian Mixture Model  : 0.211




Davies-Bouldin Index Comparison:
For K-Means                 : 2.308
For Hierarchical Clustering : 1.656
For Fuzzy C-Means           : 3.521
For Gaussian Mixture Model  : 3.665


Calinski-Harabasz Index Comparison:
For K-Means                 : 275.483
For Hierarchical Clustering : 242.075
For Fuzzy C-Means           : 165.863
For Gaussian Mixture Model  : 184.262
```

#Note on Evaluation matrics: 1.Silhouette Score: Definition : It measures how well an object fits into its assigned cluster compared to other clusters. Range : From -1 to 1, where higher values indicate better cluster fitting. Interpretation : A high Silhouette Score implies that objects are well-matched to their own cluster and poorly-matched to neighboring clusters. Davies-Bouldin Index: Definition : It assesses the average similarity between each cluster and its most similar cluster. Interpretation : A lower Davies-Bouldin Index suggests better clustering, as it indicates lower similarity between clusters. Calinski-Harabasz Index: Definition : Also known as the Variance Ratio Criterion, it evaluates the ratio of between-cluster dispersion to within- cluster dispersion. Interpretation : A higher Calinski-Harabasz Index signifies better clustering, indicating greater separation between clusters.

Diesease_test.csv test and generate predict Target from best Train model

In [49]:
```python
# Load Test dataset for check model
patient_test_df = pd.read_csv("disease_test.csv")
```

```
patient_test_df.head()
```

Out[49]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	enlargement	tumor	disorder	medication_A	medication_B	mental_health	mood_stabiliser	surge
0	PA6001	22	female	no	no	0.500	1.7	83.0	0.86	97.0	...	no	no	no	no	no	no	no	
1	PA6002	42	male	no	no	0.060	2.0	79.0	0.81	98.0	...	no	no	no	no	no	no	no	
2	PA6003	42	male	no	no	0.045	2.1	111.0	0.89	125.0	...	no	no	no	no	no	no	no	
3	PA6004	42	female	no	no	1.900	2.0	114.0	1.33	86.0	...	no	no	no	no	no	no	no	
4	PA6005	55	male	no	no	0.570	1.4	75.0	0.72	104.0	...	no	no	no	no	no	yes	no	

5 rows × 23 columns

In [50]:
```
# Drop id column for further anaysis
patient_test_df.drop('id',axis=1,inplace=True)
patient_test_df.head()
```

Out[50]:

	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6	...	enlargement	tumor	disorder	medication_A	medication_B	mental_health	mood_stabiliser	surge
0	22	female	no	no	0.500	1.7	83.0	0.86	97.0	NaN	...	no	no	no	no	no	no	no	
1	42	male	no	no	0.060	2.0	79.0	0.81	98.0	NaN	...	no	no	no	no	no	no	no	
2	42	male	no	no	0.045	2.1	111.0	0.89	125.0	NaN	...	no	no	no	no	no	no	no	
3	42	female	no	no	1.900	2.0	114.0	1.33	86.0	NaN	...	no	no	no	no	no	no	no	
4	55	male	no	no	0.570	1.4	75.0	0.72	104.0	NaN	...	no	no	no	no	no	yes	no	

5 rows × 22 columns

In [51]:
```
# Seperate Numerical columns from Dataset for further anaysis
test_numerical_columns = patient_test_df.select_dtypes(include=['int64','float64']).columns.tolist()
test_numerical_columns
```

Out[51]:
```
['age', 'test_X1', 'test_X2', 'test_X3', 'test_X4', 'test_X5', 'test_X6']
```

In [52]:
```
# Seperate Categorical columns from dataset for further analysis
test_categorical_columns = patient_test_df.select_dtypes(include=['object']).columns.tolist()
test_categorical_columns
```

Out[52]:
```
['gender',
 'sick',
 'pregnant',
 'concern_type1',
 'concern_type2',
 'enlargement',
 'tumor',
 'disorder',
 'medication_A',
 'medication_B',
 'mental_health',
 'mood_stabiliser',
 'surgery',
 'treatment_type1',
 'suspect']
```

In [53]:
```
# calculate percentage of missing values and suggest whose percentages > threshold vlaue

threshold = 60  # Set threshold here

test_missing_percentage = ((patient_test_df.isnull().sum() / len(patient_test_df)) * 100).sort_values(ascending=False)
test_columns_to_drop = test_missing_percentage[test_missing_percentage > threshold].index.tolist()

print("Columns with missing values > {}%:".format(threshold))
for column in test_columns_to_drop:
    print("{}: {:.2f}% missing values.".format(column, test_missing_percentage[column]))
```

```
Columns with missing values > 60%:
test_X6: 96.67% missing values.
```

In [54]:
```
# Drop coulmns whose missing percentages > threshold values.
patient_test_df.drop(columns=test_columns_to_drop, inplace=True)
```

In [55]:
```
# Verify columns after droping missing value columns based on threshold value percentages
patient_test_df.columns
```

Out[55]:
```
Index(['age', 'gender', 'sick', 'pregnant', 'test_X1', 'test_X2', 'test_X3',
       'test_X4', 'test_X5', 'concern_type1', 'concern_type2', 'enlargement',
       'tumor', 'disorder', 'medication_A', 'medication_B', 'mental_health',
       'mood_stabiliser', 'surgery', 'treatment_type1', 'suspect'],
      dtype='object')
```

In [56]:
```
# After Droping highest missing value column based on threshold value,handle remaining missing values.
test_updated_numerical_col = patient_test_df.select_dtypes(include=['int64','float64']).columns.tolist()
test_updated_numerical_col
```

Out[56]:
```
['age', 'test_X1', 'test_X2', 'test_X3', 'test_X4', 'test_X5']
```

In [57]:
```
# For Numerical columns: Impute missing values with median
test_numeric_imputer = SimpleImputer(strategy="median")
patient_test_df[test_updated_numerical_col] = test_numeric_imputer.fit_transform(patient_test_df[test_updated_numerical_col])

# For Categorical column 'gender': Impute missing values with mode
test_categorical_imputer = SimpleImputer(strategy="most_frequent")
patient_test_df['gender'] = test_categorical_imputer.fit_transform(patient_test_df[['gender']]).ravel()

# Check if there are any remaining missing values
```

```
test_remaining_missing_values = patient_test_df.isnull().sum().sum()
if test_remaining_missing_values == 0:
    print("All missing values have been handled.")
else:
    print(f"There are still {test_remaining_missing_values} missing values after imputation.")
```

```
All missing values have been handled.
```

In [58]:
```
#find constant column if any in dataset
test_constant_columns = []
for col in patient_test_df.columns:
    if patient_test_df[col].nunique() == 1:
        test_constant_columns.append(col)

# Print constant columns
print("test Constant columns:", test_constant_columns)
```

```
test Constant columns: ['disorder']
```

In [59]:
```
# Drop Constant column from column list
patient_test_df.drop(columns=test_constant_columns, inplace=True)
```

In [60]:
```
# check remaining columns in dataframe after droping 'disorder' colum from dataset
patient_test_df.columns
patient_test_df.shape
```

Out[60]:
```
(750, 20)
```

In [61]:
```
"""
# handle Duplicated Rows in dataset if any
test_duplicate_rows = patient_test_df[patient_test_df.duplicated()]
test_duplicate_rows
# Drop Duplicate values from dataset
patient_test_df.drop_duplicates(inplace=True)
patient_test_df.shape
"""
```

Out[61]:
```
'\n# handle Duplicated Rows in dataset if any\ntest_duplicate_rows = patient_test_df[patient_test_df.duplicated()]\ntest_duplicate_rows\n# Drop Duplicate values from d
ataset\npatient_test_df.drop_duplicates(inplace=True)\npatient_test_df.shape\n'
```

In [62]:
```
# After handling Constant column, updated categorical col
test_update_categorical_col = patient_test_df.select_dtypes(include=['object']).columns.tolist()
test_update_categorical_col
```

Out[62]:
```
['gender',
 'sick',
 'pregnant',
 'concern_type1',
 'concern_type2',
 'enlargement',
 'tumor',
 'medication_A',
 'medication_B',
 'mental_health',
 'mood_stabiliser',
 'surgery',
 'treatment_type1',
 'suspect']
```

In [63]:
```
# Encoding on update_categorical_col for further analysis
patient_test_df_encoded = pd.get_dummies(patient_test_df,columns=test_update_categorical_col,drop_first=True)
```

In [64]:
```
# Check dataset after encoding
patient_test_df_encoded = patient_test_df_encoded.astype(int)
patient_test_df_encoded.head()
```

Out[64]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	gender_male	sick_yes	pregnant_yes	concern_type1_yes	concern_type2_yes	enlargement_yes	tumor_yes	medication_A_yes	medica
0	22	0	1	83	0	97	0	0	0	0	0	0	0	0	
1	42	0	2	79	0	98	1	0	0	0	1	0	0	0	
2	42	0	2	111	0	125	1	0	0	0	1	0	0	0	
3	42	1	2	114	1	86	0	0	0	0	0	0	0	0	
4	55	0	1	75	0	104	1	0	0	0	0	0	0	0	

In [65]:
```
# Predict target values for X_test using the best model
test_predicted_target = best_model.predict(patient_test_df_encoded)

# Decode the predicted target labels
test_predicted_target_original = le.inverse_transform(test_predicted_target)

# Create a DataFrame with the decoded predicted target values
test_predicted_df = pd.DataFrame(test_predicted_target_original,columns=['Target'])
```

```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/preprocessing/_label.py:155: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [66]:
```
# Save the DataFrame to a CSV file
test_predicted_df.to_csv('Predicted_Target_best_model.csv', index=False)
```

In [ ]: