Final Submission – CA02 Artefact

Student Name: Delvin Vallooran Jose(20047713)

Muhammed Mukhsith Thekke Mattara(20040105)

Eldhose Thelakkattu Benny(20044108)

Course Title: B9DA110 Data and Network Mining

Lecturer Name : Terry Hoare

No of Words :1665

Dataset link

https://www.kaggle.com/datasets/prasad22/healthcare-dataset

1. Introduction & Business Understanding

This project aims to use real healthcare data to help predict whether a patient's medical test result will come back as "Normal" or "Abnormal." In real-world hospitals, doctors often rely on many test results and patient history to make quick decisions. A model that can predict outcomes accurately could make this process faster and more efficient.

By using machine learning, the project tries to support earlier diagnosis and reduce delays in treatment. This is especially helpful for medical staff who deal with large volumes of data daily. Stakeholders such as doctors, hospital administrators, and medical analysts can use this system as a supportive tool in their decision-making.

The dataset used includes over 55,000 patient records. Each record contains information like gender, blood type, medical condition, type of admission, and insurance provider. With more than 15 meaningful variables, the dataset provides a strong base for building and comparing models.

The main goal is to apply different machine learning algorithms to this dataset, evaluate how well they work, and choose the one that provides the best results in terms of accuracy, balance, and usefulness in a healthcare setting.

2. Data Understanding

The initial dataset included:

- 55,500 patient records
- Key features: Gender, Blood Type, Admission Type, Insurance Provider, Medication, and Medical Condition
- Target: Test Results (originally Normal, Abnormal, Inconclusive → converted to binary)

Key findings:

- No missing values
- Class imbalance detected (more Abnormal than Normal)
- High-cardinality fields (Doctor, Hospital) were excluded due to limited predictive value

Importing libraries for the model and pre-processing

```
#importing library
import pandas as pd
from pandas import read_csv, get_dummies, DataFrame,Series
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn import metrics
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
```

Data details

```
#to know the data info
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55500 entries, 0 to 55499
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Name                55500 non-null  object 
 1   Age                 55500 non-null  int64  
 2   Gender              55500 non-null  object 
 3   Blood Type          55500 non-null  object 
 4   Medical Condition   55500 non-null  object 
 5   Date of Admission   55500 non-null  object 
 6   Doctor              55500 non-null  object 
 7   Hospital            55500 non-null  object 
 8   Insurance Provider  55500 non-null  object 
 9   Billing Amount      55500 non-null  float64
 10  Room Number         55500 non-null  int64  
 11  Admission Type      55500 non-null  object 
 12  Discharge Date      55500 non-null  object 
 13  Medication          55500 non-null  object 
 14  Test Results        55500 non-null  object 
dtypes: float64(1), int64(2), object(12)
memory usage: 6.4+ MB
```

Checking null values in the data

```
#to check the null value
data.isnull().sum()
```

	0
Name	0
Age	0
Gender	0
Blood Type	0
Medical Condition	0
Date of Admission	0
Doctor	0
Hospital	0
Insurance Provider	0
Billing Amount	0
Room Number	0
Admission Type	0
Discharge Date	0
Medication	0
Test Results	0

dtype: int64

```
[23] data_cleaned['Test Results'].value_counts()
```

Test Results	count
1	36983
0	18517

3. Data Preparation

Steps taken:

- Dropped non-essential fields: Name, Doctor, Hospital, Date of Admission, Room Number

to drop unwanted colunms

```
[14] data_to_drop=['Name','Doctor',
                   'Hospital','Date of Admission','Room Number',
                   'Discharge Date']
```

- Encoded Gender numerically

converting gender colunm into 1 and 2 beacuse it is categorical values to numeric values

```
[17] data_cleaned['Gender']=data_cleaned['Gender'].map({'Male':1,'Female':0})
```

- Taking dummies : Insurance Provider, Medical Condition, Admission Type, etc.

taking dummies of objects

```
[22] data_cleaned=get_dummies(data_cleaned,['Blood Type','Medical Condition',
                                            'Insurance Provider','Admission Type','Medication'])
```

- Target column converted: Normal = 0, Abnormal/Inconclusive = 1

converting numberical values although it 3 values like normal,abnormal and Inconclusive we covert it into two normal and abnormal

```
[19] data_cleaned['Test Results'] = data_cleaned['Test Results'].apply(lambda x: 0 if x == 'Normal' else 1)
```

- Applied StandardScaler for normalization
- Used SMOTE to address class imbalance
- Split data into 70% train, 30% test

```
X_scaled=StandardScaler().fit_transform(X)
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_size = 0.3, random_state = 1)#spliting train and test 70% and 30% respectively
X_train, Y_train =SMOTE(random_state=1).fit_resample(X_train, Y_train)#smoting the data because the data is note balanced
```

4. Modeling

This project used three machine learning models: **Logistic Regression**, **Decision Tree**, and **Random Forest**. Each model was implemented in two versions — a basic version with default settings and a tuned version with optimized parameters. The goal was to compare how tuning affects the performance and identify the best-performing model for the healthcare dataset.

🔹 Logistic Regression

Logistic Regression was used to predict whether a test result is normal or abnormal based on patient features. It is known for being interpretable and fast to train. The basic version used default settings, while the tuned version adjusted two key parameters:

- C: This controls regularization strength, which helps avoid overfitting. Smaller values mean stronger regularization.
- penalty: This defines the type of regularization, such as L1 or L2.

Tuning was performed using **GridSearchCV**, which tests different combinations of these parameters and selects the one that performs best during cross-validation.

🔹 Decision Tree

Decision Trees are simple, rule-based models that split the dataset into smaller groups based on feature values. The basic model used a fixed maximum depth to prevent overfitting.

For the tuned version, the max_depth parameter was optimized using GridSearchCV. This controlled how deep the tree could grow and helped balance accuracy and generalization.

🔹 Random Forest

Random Forest is an ensemble model that builds multiple Decision Trees and combines their predictions. It usually gives better performance and avoids overfitting.

In the tuned version, we used GridSearchCV to optimize:

- n_estimators: The number of trees in the forest
- max_depth: How deep each tree can go

- max_features: The number of features to consider when splitting a node

Each model was evaluated using accuracy, precision, recall, and F1-score. This helped us measure not just how often the model was correct, but also how well it handled false positives and false negatives — which is especially important in medical applications.

5. Evaluation

Final metrics comparison: All models were evaluated using four key classification metrics: **Accuracy**, **Precision**, **Recall**, and **F1-Score**. These metrics give a well-rounded view of each model's strengths and weaknesses, especially in a healthcare context where both **false positives** and **false negatives** can carry serious implications.

Model	Accuracy	Precision	Recall	F1-Score
Auto Decision Tree	0.3320			
Auto Random Forest	0.3350			
Auto GBT	0.3530			
Logistic Regression	0.5082	0.6732	0.5166	0.5846
Tuned Logistic Reg.	0.5082	0.6732	0.5166	0.5846
Basic Random Forest	0.6027	0.7040	0.7020	0.7030
Tuned Random Forest	0.6522	0.7082	0.8175	0.7590
Basic Decision Tree	0.6695	0.6698	0.9992	0.8020
Tuned Decision Tree	0.5812	0.7028	0.6494	0.6750

The **Auto Model results** showed relatively low accuracy, ranging from 33% to 35%. This indicates that default, automated model configurations without data preprocessing,

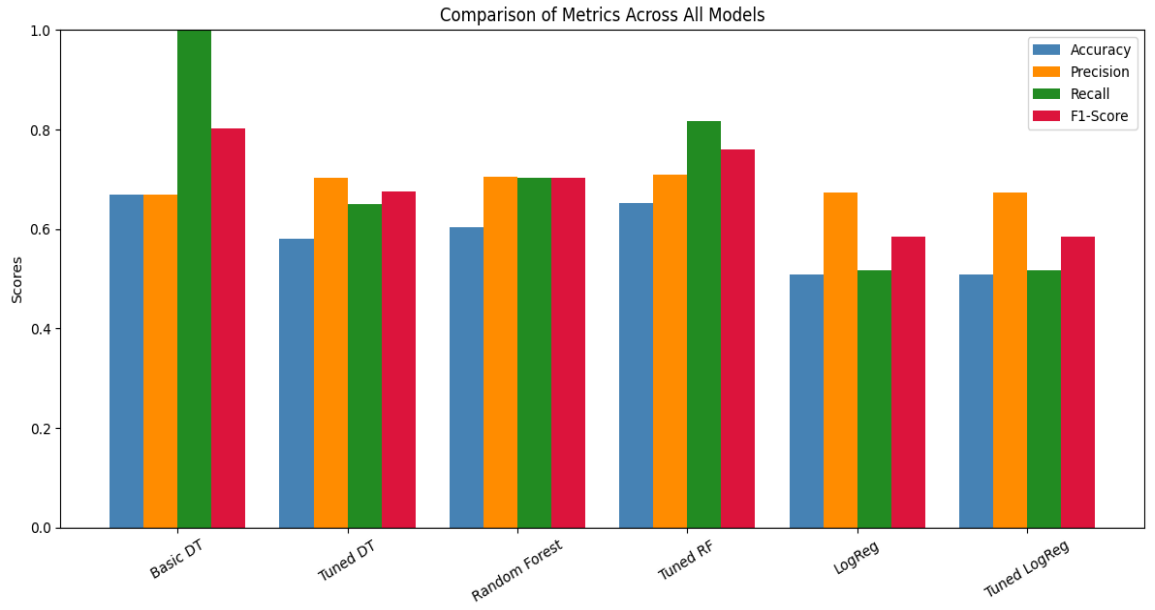
tuning, or class balancing were not effective for this dataset.

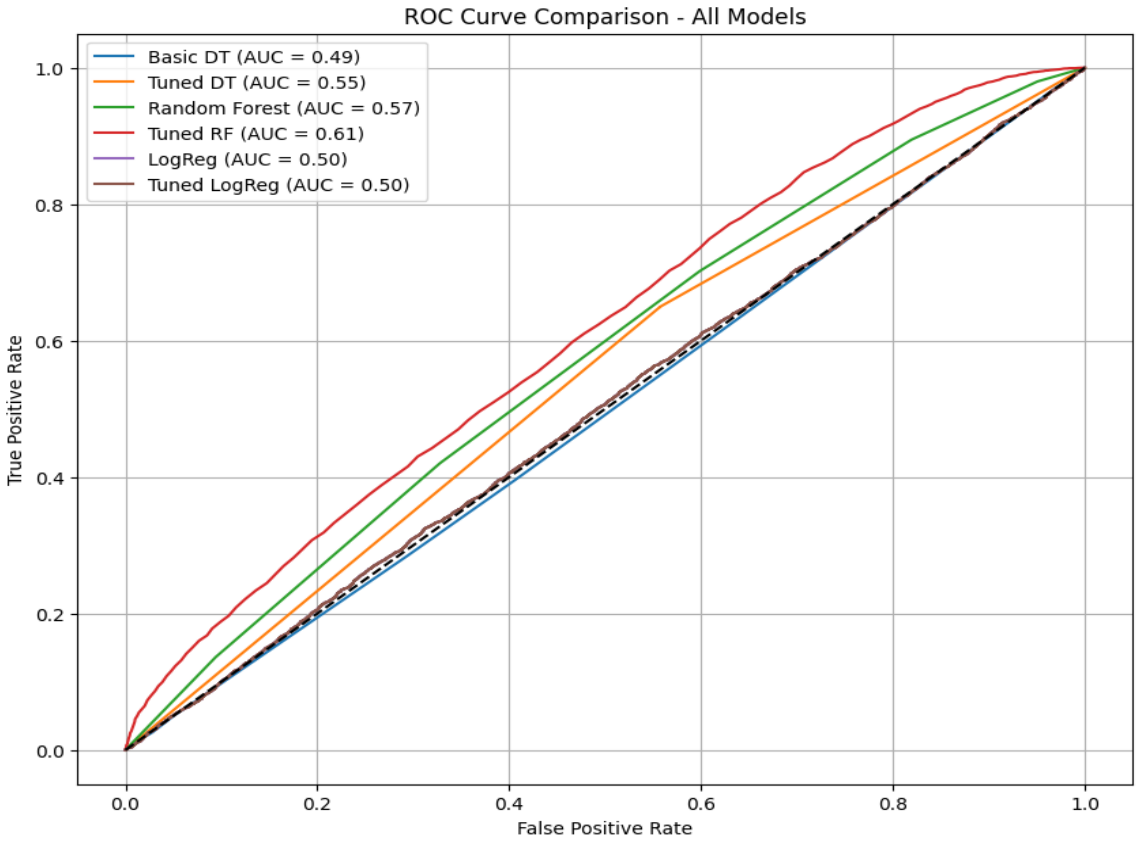
In contrast, the **manually implemented models** performed significantly better. Among them:

- **Logistic Regression** provided moderate performance but stood out for its interpretability. It had decent precision but struggled with recall, meaning it missed a number of abnormal test results.
- **Decision Tree**, especially the basic version, achieved extremely high recall (0.9992), meaning it identified almost all abnormal cases. However, its precision was lower, leading to more false positives.
- **Random Forest**, particularly the tuned version, offered the most balanced results with strong scores across all four metrics. It had the **best overall F1-score (0.7590)** and very high recall (0.8175), making it the most reliable model for real-world deployment.

This evaluation confirms that model tuning, along with careful preprocessing and feature engineering, plays a vital role in improving classification outcomes. The comparison also shows that while Logistic Regression is useful for transparency, ensemble models like Random Forest deliver stronger predictive performance for this dataset.

	A	B	C	D	E	F	G
1	Model	Classification Error	Standard Deviation	Gains	Total Time	Training Time (1,000 Rows)	Scoring Time (1,000 Rows)
2	Logistic Regression	0.667613138	0.005437228	-104	17087	30.99099099	732.6126126
3	Decision Tree	0.667780779	0.003872835	-74	7416	3.693693694	240.3603604
4	Random Forest	0.664648873	0.002834375	-28	186764	205.4954955	2105.135135
5	Gradient Boosted Trees	0.647285754	0.008473095	576	616047	497.5855856	1832.207207
6							





6. Deployment Plan

Once the best-performing model has been identified, the next step is to consider how it can be deployed into a real-world environment. For this project, the **Tuned Random Forest** model is recommended for deployment due to its strong balance of accuracy, recall, and overall F1-score. Although not the most interpretable model, its performance and reliability make it suitable for real-time classification tasks.

Model Deployment Strategy

- The model can be **exported as a serialized file** (e.g., .pkl or .joblib) and deployed via a **RESTful API** or embedded in a **backend service** for integration with hospital software systems.
- It can be hosted in cloud environments (like AWS, Azure, or GCP) or within hospital IT infrastructure depending on data security requirements.

Clinical Integration

- The model can be used to **automatically flag abnormal test result predictions** based on input data received from Electronic Health Records (EHRs).
- When an abnormal result is predicted, the system can **generate alerts** for further manual review by

a clinician or specialist, improving triage speed.

- Results can be integrated into existing **hospital dashboards** or health monitoring platforms with user-friendly interfaces.

Explainability and Trust

- While Random Forest is a black-box model, tools like **SHAP (SHapley Additive Explanations)** and **LIME** can be used to interpret individual predictions.
- SHAP can highlight which features (e.g., Medical Condition or Medication) contributed most to an abnormal prediction, helping medical professionals trust and understand the model's reasoning.
- A simplified version of Logistic Regression can also be deployed in parallel for transparency-focused use cases.

Model Monitoring and Retraining

- After deployment, it is essential to **monitor the model's performance over time**. This includes tracking input data drift, changes in prediction confidence, and model accuracy using real patient data.
- A **feedback loop** should be created to collect user inputs (e.g., corrections or overrides by clinicians) to continuously improve model performance.
- Retraining can be scheduled periodically (e.g., monthly or quarterly) using updated datasets.

Ethical and Regulatory Considerations

- Since the application is related to healthcare, it is important to ensure compliance with **GDPR** and **data protection regulations**.
- Model outputs should always be used as **decision support**, not standalone decisions, to prevent over-reliance on AI in clinical settings.
- All decisions should be documented, and the system should maintain logs of model predictions for transparency and auditability.

7. Conclusion

This project aimed to develop a predictive system that classifies healthcare test outcomes as either "Normal" or "Abnormal" using structured patient data. By applying the CRISP-DM methodology, the entire data science process was followed—from understanding the business context and dataset, to cleaning and transforming the data, building and tuning models, and finally considering deployment strategies.

Three machine learning algorithms were explored in detail: **Logistic Regression**, **Decision Tree**, and **Random Forest**, each with basic and tuned versions. The results showed that while Logistic Regression

offered transparency and consistent results, it struggled with identifying abnormal cases due to lower recall. Decision Trees, especially the basic version, provided extremely high recall but at the cost of precision. Random Forest proved to be the most balanced and reliable model overall, especially after tuning, and is therefore recommended for real-world use.

Auto Model benchmarks performed poorly, confirming the value of manual data preparation, class balancing, and hyperparameter tuning. This highlights the importance of understanding the dataset and model behavior rather than relying on automated tools alone.

Deployment planning focused on using Random Forest in production, with Logistic Regression maintained for transparency when needed. Tools like SHAP and LIME were recommended for enhancing interpretability and trust. Future work may include model ensembling, adding interaction terms, or integrating live data pipelines for real-time prediction.

Overall, the project demonstrated that with thoughtful preprocessing and tuning, machine learning can provide valuable support in healthcare environments—offering faster, more consistent insights that complement human decision-making.

8. References

1 **Pedregosa, F. et al.** (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830.

(Used for Logistic Regression, Random Forest, and Decision Tree implementations)

2 **Brownlee, J.** (2020). *How to Tune Machine Learning Models in Python with GridSearchCV*. MachineLearningMastery.com

(Referenced for model tuning with GridSearchCV)

3 **Krish Naik.** (2021, June 30). *Hyperparameter Tuning with GridSearchCV in Python* [Video]. YouTube.

(Used to understand the application of GridSearchCV in real projects)