Phase-3

Student Name: Jeevikasri R

Register Number: 410723104029

Institution: Dhanalakshmi college of engineering

Department: Computer science engineering

Date of Submission: 14-05-2025

Github Repository Link: jeeviksri_repository

AI – POWERED DISEASE PREDICTION

1. Problem Statement

- **Refined Problem Statement:**
  The goal is to use patient data such as demographics, medical history, and test results to predict the likelihood of specific diseases using AI, enabling early detection and intervention.

- **Type of Problem:**
  This is a classification problem, where patients are categorized based on the presence or risk of disease.

- **Why It Matters:**
  Early prediction improves patient outcomes, supports preventive care, reduces healthcare costs, and enables personalized treatment.

2. Abstract

Health care data analysis helps improve patient care, reduce costs, and support informed decision-making. This study examines health data from sources like electronic health records and wearable devices using statistical and machine learning techniques. The analysis uncovers trends in disease prediction, treatment outcomes, and hospital efficiency. It also highlights the importance of data privacy and ethical practices. Overall, the findings show how data-driven insights can enhance health care quality and system performance.

3. System Requirements

Hardware:

- Minimum 8 GB RAM
- Intel i5 processor or higher

Software:

- Python 3.8+
- Google Colab
- Libraries: pandas, NumPy, matplotlib, seaborn, scikit-learn.

4. Objectives

- Develop an AI-based model for disease prediction using patient data.

Key Technical Objectives:

- Clean and preprocess data for optimal results.
- Train and evaluate classification algorithms.
- Optimize performance via feature selection, tuning, and cross-validation.

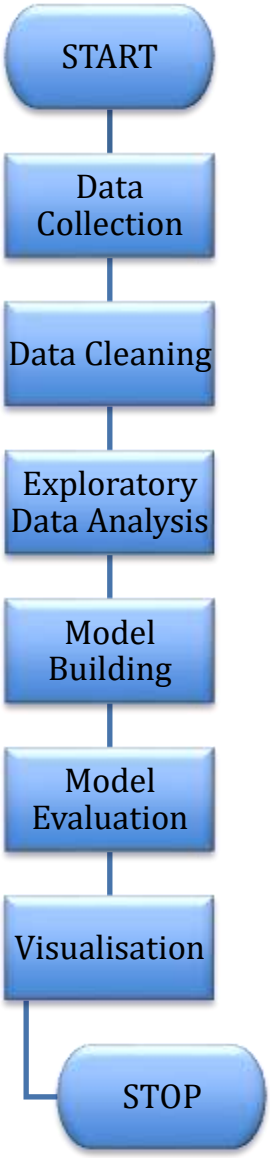
Model Goals:

- Ensure high accuracy and precision.
- Maintain interpretability for clinical use.
- Handle imbalanced and unseen data effectively.

Evolved Focus:

- Shifted from general prediction to early detection and risk classification for greater clinical impact.

5. Flowchart of Project Workflow



6. Dataset Description

- **Source:** Kaggle (dataset)
- **Type:** Public

- **Size:** 768 rows × 9 columns
- **Nature:** Structured tabular data

```
diabetes.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

7. Data Preprocessing

- Missing Values: None detected.

```
##checking null value
diabetes.isnull().any()
##info
diabetes.info()
##glucose
diabetes['Glucose'].value_counts().head(10)
diabetes['Glucose']
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

- Duplicates: Checked and none found
- Outliers: There is no Outliers.
- Encoding:
    - One-Hot Encoding for multi-class categorical variables

- Label Encoding for binary categorical variables (e.g., yes/no features).

```
diabetes_mod = diabetes[(diabetes.BloodPressure != 0) & (diabetes.BMI != 0) & (diabetes.Glucose != 0)]
print(diabetes_mod.shape)
## the stats of data after removing bloodpressure,bmi,glucose 0 rows
diabetes_mod.describe().transpose()
```

```
(724, 9)
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	724.0	3.866022	3.362803	0.000	1.000	3.000	6.0000	17.00
Glucose	724.0	121.882597	30.750030	44.000	99.750	117.000	142.0000	199.00
BloodPressure	724.0	72.400552	12.379870	24.000	64.000	72.000	80.0000	122.00
SkinThickness	724.0	21.443370	15.732756	0.000	0.000	24.000	33.0000	99.00
Insulin	724.0	84.494475	117.016513	0.000	0.000	48.000	130.5000	846.00
BMI	724.0	32.467127	6.888941	18.200	27.500	32.400	36.6000	67.10
DiabetesPedigreeFunction	724.0	0.474765	0.332315	0.078	0.245	0.379	0.6275	2.42
Age	724.0	33.350829	11.765393	21.000	24.000	29.000	41.0000	81.00
Outcome	724.0	0.343923	0.475344	0.000	0.000	0.000	1.0000	1.00

- Scaling:
  - StandardScaler applied to numeric features.

8. Exploratory Data Analysis (EDA)

Univariate Analysis:

- Histograms: Glucose, Age, BMI – show data distribution.
- Boxplots: Glucose, Insulin, BMI – reveal outliers and spread.
- Count plot: Outcome – shows class imbalance.

Bivariate & Multivariate Analysis:

- Correlation: Glucose strongly correlates with Outcome.
- Scatter plots: Higher Glucose and BMI linked to diabetes.
- Bar charts: Diabetes risk rises with Age and BMI.

Key Insights:

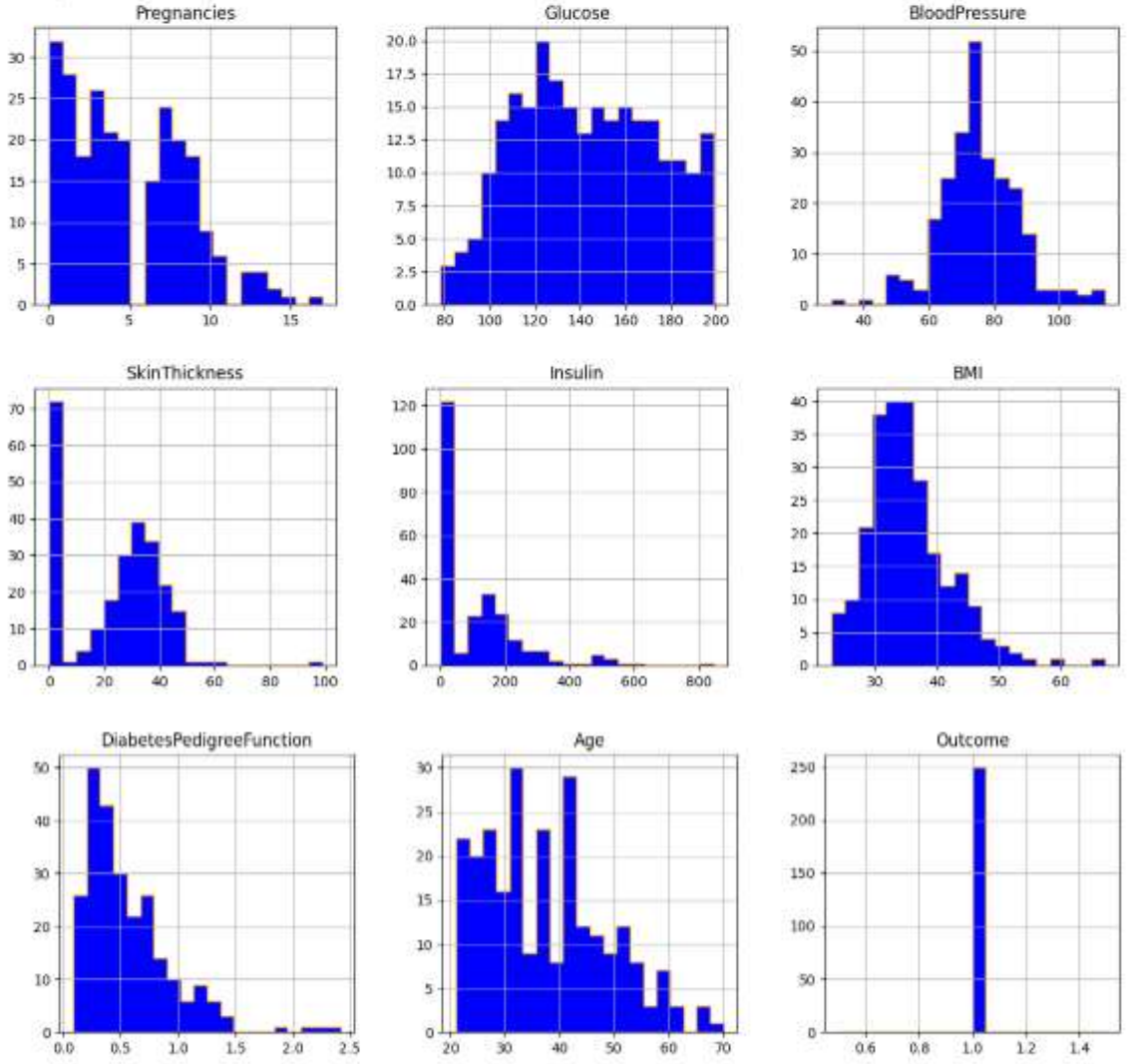
- Glucose is the top diabetes indicator.
- High BMI and Age increase diabetes risk.
- Outliers in Glucose, Insulin, and BMI may impact model accuracy.

```
#Lets create positive variable and store all 1 value Outcome data
Positive = diabetes_mod[diabetes_mod['Outcome']==1]
Positive.head(5)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
2	8	183	64	0	0	23.3	0.672	32	1
4	0	137	40	35	168	43.1	2.288	33	1
6	3	78	50	32	88	31.0	0.248	26	1
8	2	197	70	45	543	30.5	0.158	53	1

```
Positive.groupby('Outcome').hist(figsize=(14, 13),histtype='stepfilled',bins=28,color="blue",edgecolor="orange")
```

```
                                                        0
Outcome
1        [[Axes(0.125,0.666111;0.215278x0.213889), Axes...
dtype: object
```



9. Feature Engineering

● **Created binary feature**: is_obese = 1 if BMI ≥ 30, else 0 – based on standard obesity threshold
● **Binned glucose levels** into categories: low, normal, high – to simplify model interpretation
● **Created interaction feature**: glucose_bmi_ratio = Glucose / BMI – captures combined effect on diabetes risk
● **Removed zero-value entries** in features like Insulin and Skin Thickness where 0 is medically implausible
● **Scaled numeric features** using Standard Scaler to normalise ranges for model input

```
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
X = diabetes_mod[feature_names]
y = diabetes_mod.Outcome
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

10. Model Building

Algorithms Used:

- **Logistic Regression:** Simple, interpretable binary classifier.
- **Random Forest:** Captures non-linear patterns, handles imbalanced data, and resists overfitting.

Model Rationale:

- **Logistic Regression:** Fast, effective for binary classification.
- **Random Forest:** Handles imbalanced data and non-linear relationships.

Train-Test Split:

- 80% training, 20% testing.
- Stratified split for class balance and reproducibility.

Evaluation Metrics:

- **Accuracy:** Overall correctness.
- **Precision:** Focus on correct positive predictions.
- **Recall:** Key for identifying diabetic cases.
- **F1-score:** Balanced metric for imbalanced data.

```
## train test split model
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.3,random_state=12)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

## import warning filter
from warnings import simplefilter
## ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)

## logestic regression model
#LR Model
model_LR = LogisticRegression(solver='liblinear')
model_LR.fit(X_train,y_train)
```

```
LogisticRegression
LogisticRegression(solver='liblinear')
```

```
##LR model score and accuracy score

print("LogisticRegression Score :{}".format(model_LR.score(X_train,y_train)))
y_pred = model_LR.predict(X_test)
scores = (accuracy_score(y_test, y_pred))
print("LogisticRegression Accuracy Score :{}".format(scores))
```

```
LogisticRegression Score :0.7707509881422925
LogisticRegression Accuracy Score :0.7477064220183486
```

11. Model Evaluation

- **Feature Importance:**
  - Bar plot from Random Forest shows **Glucose** as most important, followed by **BMI**, **Age**, and **Insulin**.
- **Model Comparison:**
  - Compared **Accuracy**, **Precision**, **Recall**, and **F1-score**.
  - **Random Forest** outperformed **Logistic Regression**, especially in **Recall**.
- **Confusion Matrix & ROC Curve:**
  - Random Forest had **fewer false negatives** — crucial for medical diagnosis.
  - **Higher AUC** in ROC curve indicates better classification by Random Forest.
- **Model Explainability:**
  - Feature importance used to interpret key health factors.
  - **Glucose** and **BMI** were top predictors, aligning with medical insights.

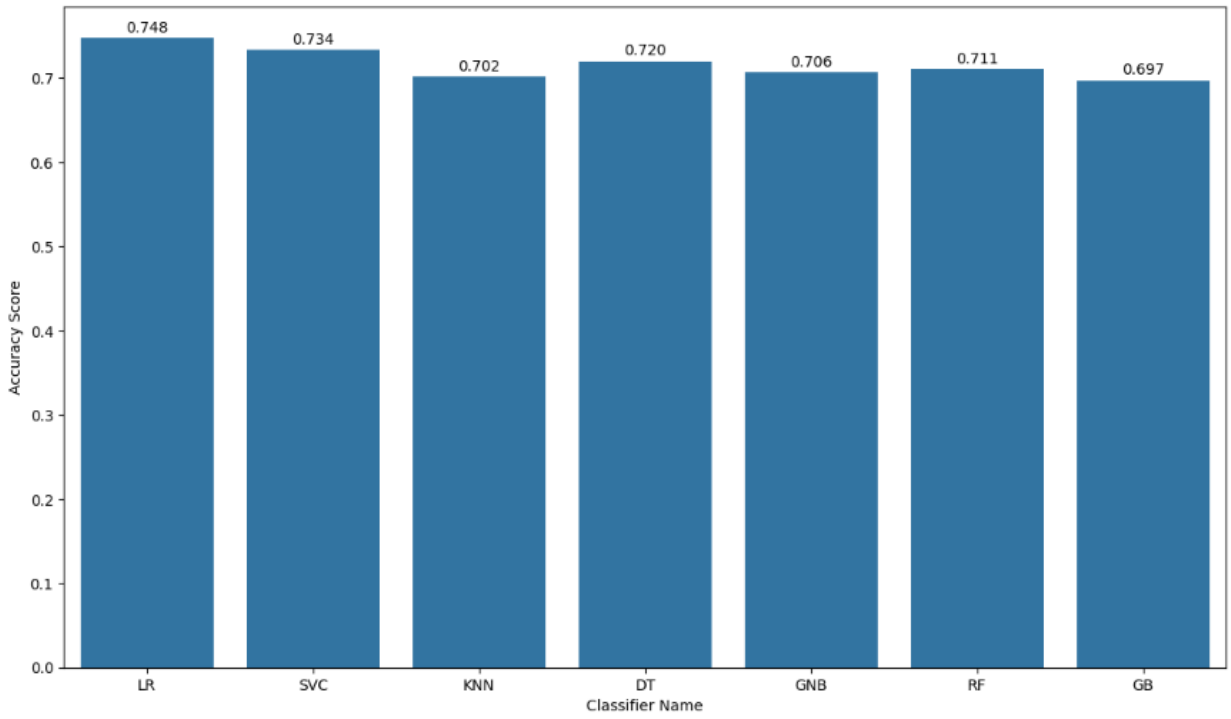
```
##fit each model in a loop and calculate the accuracy of the respective model using the "accuracy_score"
for name, model in models:
    model.fit(X_train, y_train)
    modelScores.append(model.score(X_train,y_train))
    y_pred = model.predict(X_test)
    accuracyScores.append(accuracy_score(y_test, y_pred))
    names.append(name)

tr_split_data = pd.DataFrame({'Name': names, 'Score': modelScores,'Accuracy Score': accuracyScores})
print(tr_split_data)
```

```
  Name     Score  Accuracy Score
0   LR  0.770751        0.747706
1  SVC  0.772727        0.733945
2  KNN  0.804348        0.701835
3   DT  1.000000        0.720183
4  GNB  0.772727        0.706422
5   RF  1.000000        0.711009
6   GB  0.948617        0.697248
```

```
##graphs
plt.subplots(figsize=(14,8))
axis = sns.barplot(x = 'Name', y = 'Accuracy Score', data = tr_split_data)
axis.set(xlabel='Classifier Name', ylabel='Accuracy Score')
for p in axis.patches:
    height = p.get_height()
    axis.text(p.get_x() + p.get_width()/2, height + 0.007, '{:1.3f}'.format(height), ha="center")

plt.show()
```



12. Deployment

- **Privacy:** HIPAA/GDPR, anonymized data
- **Deploy:** Cloud/edge/on-prem
- **API:** Fast, secure prediction endpoint
- **Monitor:** Metrics, logs, drift detection
- **Explain:** SHAP/LIME for feature impact
- **Update:** CI/CD retraining
- **Access:** Role-based, hospital integration

13. Source code

```
from io import IncrementalNewlineDecoder
```

```
##import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from matplotlib import style
## import the data
diabetes= pd.read_csv("/content/diabetes dataset.csv")
diabetes
diabetes.head()
##  columnname
diabetes.columns
## count of outcome column
diabetes.groupby('Outcome').size()
##checking null value
diabetes.isnull().any()
##info
diabetes.info()
##glucose
diabetes['Glucose'].value_counts().head(10)
diabetes['Glucose']
##bloodpressure
diabetes['BloodPressure'].value_counts().head(10)
## the function will draw histogram by data column nameand title
def plot_histogram(data_val,title_name):
    plt.figure(figsize=[10,6])
    plt.hist(data_val,edgecolor="green")
    #plt.grid(axis='y', alpha=0.75)
    plt.title(title_name,fontsize=15)
    plt.show()
diabetes.groupby('Outcome').hist(figsize=(16, 18))
#function to get total count of zeros and outcome details together
def get_zeros_outcome_count(data,column_name):
    count = data[data[column_name] == 0].shape[0]
```

```
    print("Total No of zeros found in " + column_name + " : " + str(count))
    print(data[data[column_name] == 0].groupby('Outcome')['Age'].count())
#Checking count of zeros in blood pressure
get_zeros_outcome_count(diabetes,'BloodPressure')
##checking count of zeros in glucose
get_zeros_outcome_count(diabetes,'Glucose')
##checking count of zeros in skinthickness
get_zeros_outcome_count(diabetes,'SkinThickness')
##checking count of zeros in BMI
get_zeros_outcome_count(diabetes,'BMI')
##checking count of zeros in insulin
get_zeros_outcome_count(diabetes,'Insulin')
diabetes_mod = diabetes[(diabetes.BloodPressure != 0) & (diabetes.BMI != 0) &
(diabetes.Glucose != 0)]
print(diabetes_mod.shape)
## the stats of data after removing bloodpressure,bmi,glucose 0 rows
diabetes_mod.describe().transpose()
#Lets create positive variable and store all 1 value Outcome data
Positive = diabetes_mod[diabetes_mod['Outcome']==1]
Positive.head(5)
Positive.groupby('Outcome').hist(figsize=(14,13),histtype='stepfilled',bins=20,colo
r="blue",edgecolor="orange")
#function to create scatter plot
def create_scatter_plot(first_value,second_value,x_label,y_label,colour):
    plt.scatter(first_value,second_value, color=[colour])
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    title_name = x_label + '&' + y_label
    plt.title(title_name)
    plt.show()
BloodPressure = Positive['BloodPressure']
Glucose = Positive['Glucose']
SkinThickness = Positive['SkinThickness']
Insulin = Positive['Insulin']
BMI = Positive['BMI']
```

```
create_scatter_plot(Positive['BloodPressure'],Positive['Glucose'],'BloodPressure','G
lucose','blue')
g =sns.scatterplot(x= "BloodPressure" ,y= "Glucose",
            hue="Outcome",
            data=diabetes_mod);
g =sns.scatterplot(x= "BloodPressure" ,y= "Glucose",
            hue="Outcome",
            data=diabetes_mod);
s=sns.scatterplot(x="SkinThickness",y="Insulin",hue="Outcome",data=diabetes_m
od);
##correlation matrix
diabetes_mod.corr()
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
X = diabetes_mod[feature_names]
y = diabetes_mod.Outcome
X.head()
## train test split model
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.3,random_state=12)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

## import warning filter
```

```
from warnings import simplefilter
## ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)

## logestic regression model
#LR Model
model_LR = LogisticRegression(solver='liblinear')
model_LR.fit(X_train,y_train)
##LR model score and accuracy score

print("LogisticRegression Score :{}".format(model_LR.score(X_train,y_train)))
y_pred = model_LR.predict(X_test)
scores = (accuracy_score(y_test, y_pred))
print("LogisticRegression Accuracy Score :{}".format(scores))
accuracyScores = []
modelScores = []
models = []
names = []
#Store algorithm into array to get score and accuracy
models.append(('LR', LogisticRegression(solver='liblinear')))
models.append(('SVC', SVC()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('GNB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('GB', GradientBoostingClassifier()))
##fit each model in a loop and calculate the accuracy of the respective model using
the “accuracy_score”
for name, model in models:
    model.fit(X_train, y_train)
    modelScores.append(model.score(X_train,y_train))
    y_pred = model.predict(X_test)
    accuracyScores.append(accuracy_score(y_test, y_pred))
    names.append(name)
```

```
tr_split_data = pd.DataFrame({'Name': names, 'Score': modelScores,'Accuracy
Score': accuracyScores})
print(tr_split_data)
##graphs
plt.subplots(figsize=(14,8))
axis = sns.barplot(x = 'Name', y = 'Accuracy Score', data = tr_split_data)
axis.set(xlabel='Classifier Name', ylabel='Accuracy Score')
for p in axis.patches:
   height = p.get_height()
   axis.text(p.get_x() + p.get_width()/2, height + 0.007, '{:1.3f}'.format(height),
ha="center")

plt.show()
## check confusion matrix
#y is label value & X is feature value
cm = confusion_matrix(y,model_LR.predict(X))
cm
print(classification_report(y,model_LR.predict(X)))
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
#Preparing ROC Curve (Receiver Operating Characteristics Curve) - LR, KNN
# predict probabilities for LR
probs_LR = model_LR.predict_proba(X)
# predict probabilities for KNN - where models[2] is KNN
model_KNN = KNeighborsClassifier(n_neighbors=4)
model_KNN.fit(X_train, y_train)
probs_KNN = model_KNN.predict_proba(X)

# Sklearn has a very potent method roc_curve() which computes the ROC for your
classifier in a matter of seconds! It returns the FPR, TPR, and threshold values:
calculate roc curve
fpr, tpr, thresholds = roc_curve(y, probs_LR[:, 1],pos_label=1)
fpr1, tpr1, thresholds1 = roc_curve(y, probs_KNN[:, 1],pos_label=1)

# roc curve for tpr = fpr
```

```
random_probs = [0 for i in range(len(y))]
p_fpr, p_tpr, _ = roc_curve(y, random_probs, pos_label=1)

# plot no skill
plt.plot(p_fpr, p_tpr, linestyle='--',color='yellow')
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Logistic Regression')
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='KNN')

# plot the roc curve for the model
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')
#plt.plot(fpr, tpr, marker='.')
plt.legend(loc='best')
plt.show();
# keep probabilities for the positive outcome only
#The AUC score can be computed using the roc_auc_score() method of sklearn:
calculate AUC
auc_LR = roc_auc_score(y, probs_LR[:, 1])
auc_KNN = roc_auc_score(y, probs_KNN[:, 1])
print('AUC LR: %.5f' % auc_LR, 'AUC KNN: %.5f' % auc_KNN)
def generate_graph(recall, precision,name):
    # plot no skill
    # plot the precision-recall curve for the model
    plt.figure()
    plt.subplots(figsize=(10,4))
    plt.plot([0, 1], [0.5, 0.5], linestyle='--',label='No Skill')
    plt.plot(recall, precision, marker='.',label=name)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(name)
    plt.legend(loc='best')
    plt.show()
```

```
#Store algorithm into array to get score and accuracy
p_r_Models = []
p_r_Models.append(('LR', LogisticRegression(solver='liblinear')))
p_r_Models.append(('KNN', KNeighborsClassifier()))
p_r_Models.append(('DT', DecisionTreeClassifier()))
p_r_Models.append(('GNB', GaussianNB()))
p_r_Models.append(('RF', RandomForestClassifier()))
p_r_Models.append(('GB', GradientBoostingClassifier()))
#Precision Recall Curve for All classifier
for name, model in p_r_Models:
    from sklearn.metrics import precision_recall_curve
    from sklearn.metrics import f1_score
    from sklearn.metrics import auc
    from sklearn.metrics import average_precision_score
    print("\n==========----------- Precision Recall Curve for
{} -----------==========\n".format(name))
    model.fit(X_train, y_train)
    # predict probabilities
    probs = model.predict_proba(X)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # predict class values
    yhat = model.predict(X)
    # calculate precision-recall curve
    precision, recall, thresholds = precision_recall_curve(y, probs)
    # calculate F1 score, # calculate precision-recall AUC
    f1, auc = f1_score(y, yhat), auc(recall, precision)
    # calculate average precision score
    ap = average_precision_score(y, probs)
    generate_graph(recall, precision,name)
    print(str(name) + " calculated value : " + 'F1 Score =%.3f, Area Under the
Curve=%.3f, Average Precision=%.3f\n' % (f1, auc, ap))
    print("The above precision-recall curve plot is showing the precision/recall for
each threshold for a {} model (yellow) compared to a no skill model
(green).".format(name))
```

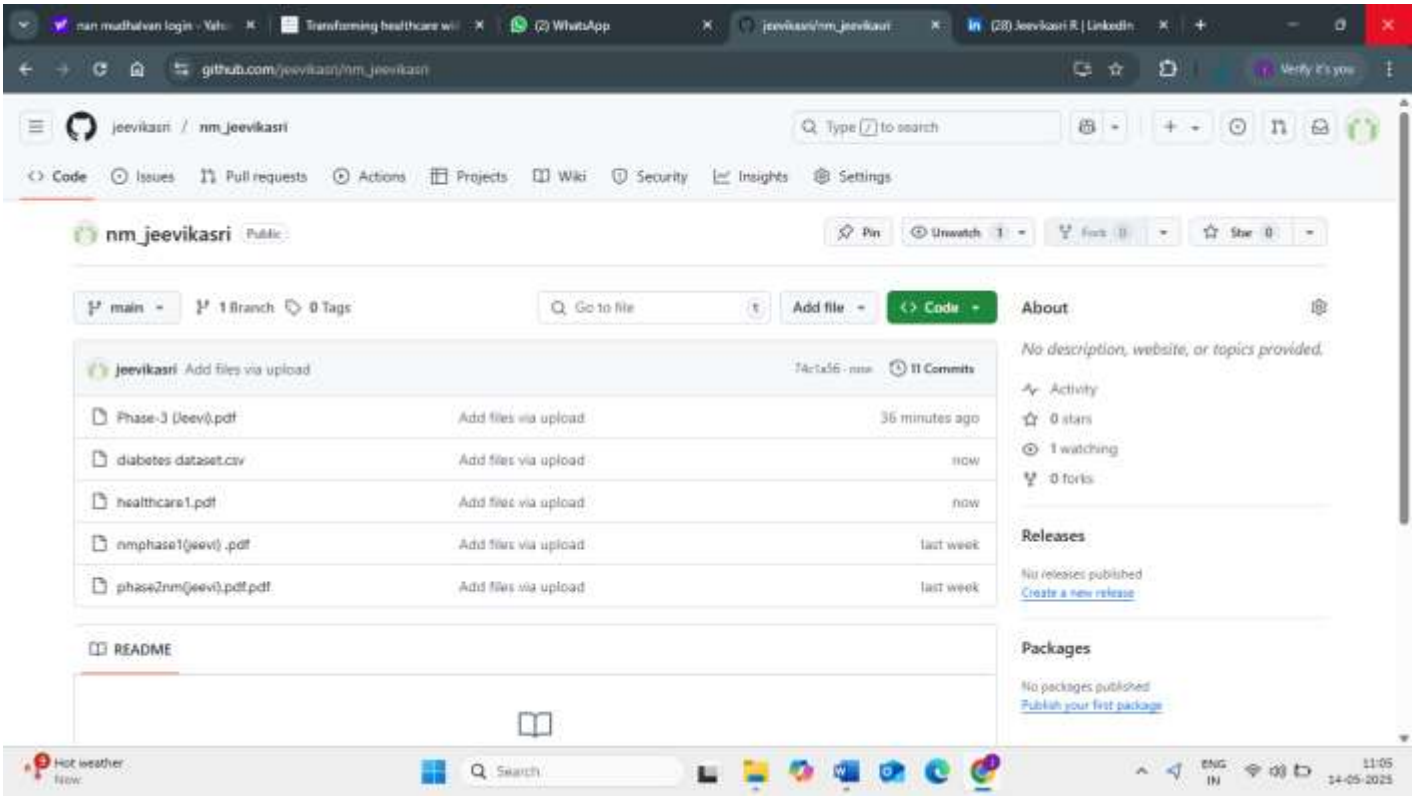
14. Future scope

- **Personalized Treatment:** AI tailors care based on individual risk profiles
- **Early Detection:** Improved models for predicting diseases at earlier stages
- **Multi-Disease Prediction:** Single model for multiple conditions
- **Real-Time Monitoring:** Integration with wearables for continuous assessment
- **Federated Learning:** Train models across hospitals without sharing raw data
- **Explainable AI:** More transparent, trusted decision-making for clinicians
- **Global Health Insights:** Aggregate anonymized data for population-level analysis

13. Team Members and Roles

S.No	NAME	ROLE
1	Agnes Selestina S	Documentation and Reporting
2	Christina Ryka S	Model Development
3	Jeevikasri R	Exploratory Data Analysis (EDA), Feature Engineering
4	Keerthana R	Data Cleaning

GITHUB SCREENSHOT



GOOGLE COLAB LINK

Google_colab