

## Healthcare PGP – Project 2

To create a dynamic and responsive Java e-healthcare web application for ordering medicines.

### Prerequisites

- Angular
- Java
- MySQL
- Spring Boot
- Jenkins
- Docker
- AWS
- Git and GitHub

### ChatGPT

#### Project Writeup: Dynamic and Responsive Java E-Healthcare Web Application

##### Introduction

In the modern healthcare industry, there is a growing demand for convenient and accessible ways to order medicines online. To meet this demand, we propose the development of a dynamic and responsive Java e-healthcare web application for ordering medicines. This application will provide users with a user-friendly interface to browse, search, and purchase medicines online, thereby enhancing the overall healthcare experience.

##### Objectives

Develop a robust and scalable web application using Angular for the frontend and Java with Spring Boot for the backend.

Implement user authentication and authorization to ensure secure access to the application.

Integrate MySQL database for storing user information, medicine details, and order history.

Utilize Docker for containerization to ensure consistency and portability across different environments.

Employ AWS for deployment to leverage its scalability, reliability, and security features.

Set up a continuous integration and continuous deployment (CI/CD) pipeline using Jenkins for automated testing and deployment.

Utilize Git and GitHub for version control and collaboration among team members.

## Technologies Used

**Angular:** Angular will be used to develop the frontend of the web application. It offers a robust framework for building dynamic and interactive user interfaces.

**Java with Spring Boot:** Java along with Spring Boot will be used to develop the backend of the application. Spring Boot provides a streamlined development experience for building RESTful APIs and microservices.

**MySQL:** MySQL will serve as the relational database management system (RDBMS) for storing user data, medicine details, and order information.

**Docker:** Docker will be used for containerization, allowing the application to run consistently across different environments and facilitating easy deployment.

**AWS:** AWS will be utilized for deploying the application, taking advantage of its cloud infrastructure services such as EC2, RDS, and S3 for scalability, database management, and storage.

**Jenkins:** Jenkins will be used to set up a CI/CD pipeline for automating the testing and deployment processes, ensuring rapid and reliable delivery of updates to the application.

**Git and GitHub:** Git will be used for version control, and GitHub will serve as a collaborative platform for storing and managing the source code, enabling seamless collaboration among team members.

## Project Workflow

**Frontend Development:** Angular will be used to develop the frontend of the application, including user interfaces for browsing medicines, managing user accounts, and placing orders.

**Backend Development:** Java with Spring Boot will be used to develop the backend RESTful APIs, including endpoints for user authentication, medicine management, and order processing.

**Database Design:** MySQL will be utilized to design and implement the database schema, including tables for storing user data, medicine details, and order information.

**Containerization with Docker:** Docker will be used to containerize the application components, including the frontend, backend, and database, ensuring consistency and portability.

**Deployment to AWS:** The containerized application will be deployed to AWS, leveraging its cloud infrastructure services for scalability, reliability, and security.

**CI/CD Pipeline with Jenkins:** Jenkins will be used to set up a CI/CD pipeline for automating the testing and deployment processes, ensuring rapid and reliable delivery of updates to the application.

**Version Control and Collaboration:** Git and GitHub will be used for version control and collaboration among team members, allowing for seamless integration of changes and efficient management of the project codebase.

## Conclusion

In conclusion, the development of a dynamic and responsive Java e-healthcare web application for ordering medicines represents an innovative solution to meet the evolving needs of the healthcare industry. By leveraging technologies such as Angular, Java with Spring Boot, MySQL, Docker, AWS, Jenkins, Git, and GitHub, we can create a robust and scalable application that provides users with a seamless and secure online shopping experience for medicines. Through effective collaboration and utilization of best practices in software development, we aim to deliver a high-quality and user-friendly application that enhances the overall healthcare experience for users.

## DESCRIPTION

NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases.

The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

Build a model to accurately predict whether the patients in the dataset have diabetes or not.

### Dataset Description

The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

### Variables Description

Pregnancies Number of times pregnant

Glucose Plasma glucose concentration in an oral glucose tolerance test

BloodPressure Diastolic blood pressure (mm Hg)

SkinThickness Triceps skinfold thickness (mm)

Insulin Two hour serum insulin

BMI Body Mass Index

DiabetesPedigreeFunction Diabetes pedigree function

Age Age in years

Outcome Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

### Project Task: Week 1

#### Data Exploration:

1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

Glucose

BloodPressure

SkinThickness

Insulin

BMI

2. Visually explore these variables using histograms. Treat the missing values accordingly.

3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

#### Data Exploration:

4. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

5. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

6. Perform correlation analysis. Visually explore it using a heat map.

## Project Task: Week 2

### Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.
2. Apply an appropriate classification algorithm to build a model.
3. Compare various models with the results from KNN algorithm.
4. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

### Python:

#### Importing / Data Wrangling / EDA / Machine Learning

##### Healthcare PGP

#### Importing Libraries

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('health care diabetes.csv')
```

```
Pregnancies Glucose BloodPressure SkinThickness Insulin BMI
```

```
\
```

```
0 6 148 72 35 0 33.6
```

```
1 1 85 66 29 0 26.6
```

```
2 8 183 64 0 0 23.3
```

```
3 1 89 66 23 94 28.1
```

```
4 0 137 40 35 168 43.1
```

```
.. .. .
```

```
763 10 101 76 48 180 32.9
```

```
764 2 122 70 27 0 36.8
```

```
765 5 121 72 23 112 26.2
```

```
766 1 126 60 0 0 30.1
```

```
767 1 93 70 31 0 30.4
```

```
DiabetesPedigreeFunction Age Outcome
```

```
0 0.627 50 1
```

```
1 0.351 31 0
```

```
2 0.672 32 1
```

```
3 0.167 21 0
```

```
4 2.288 33 1
```

```
.. .. .
```

```
763 0.171 63 0
```

```
764 0.340 27 0
```

```
765 0.245 30 0
```

```
766 0.349 47 1
```

767 0.315 23 0

[768 rows x 9 columns]

Descriptive Analysis

df.isna().sum()

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

BMI 0

DiabetesPedigreeFunction 0

Age 0

Outcome dtype:

int64

0

df.info()

<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

sns.heatmap(df.isna())

<AxesSubplot:>

df.describe().T

25% \

Pregnancies

1.00000

Glucose

99.00000

BloodPressure

62.00000

SkinThickness

0.00000

```

Insulin
0.00000 BMI
27.30000
DiabetesPedigreeFunction
0.24375
Age
24.00000
count mean std min
768.0 3.845052 3.369578 0.000
768.0 120.894531 31.972618 0.000
768.0 69.105469 19.355807 0.000
768.0 20.536458 15.952218 0.000
768.0 79.799479 115.244002 0.000
768.0 31.992578 7.884160 0.000
768.0 0.471876 0.331329 0.078
768.0 33.240885 11.760232 21.000
Outcome
0.00000
768.0 0.348958 0.476951 0.000
50% 75% max
Pregnancies 3.0000 6.00000 17.00
Glucose 117.0000 140.25000 199.00
BloodPressure 72.0000 80.00000 122.00
SkinThickness 23.0000 32.00000 99.00
Insulin 30.5000 127.25000 846.00
BMI 32.0000 36.60000 67.10
DiabetesPedigreeFunction 0.3725 0.62625 2.42
Age 29.0000 41.00000 81.00
Outcome 0.0000 1.00000 1.00
Visual Exploration of Data
Distribution of data and their skewness.
for i in ['Glucose', 'BMI', 'Insulin', 'SkinThickness', 'BloodPressure']:
plt.hist(df[i], color = 'teal', label = i) plt.legend()
plt.show()
500 Insulin
400
JOO
200
100
0 200 400 600 BOO
250
BMI
200
150

```

```

100
0 10 20 TO 40 50
Indexes with Erroneous Values
for i in ['Glucose', 'BMI', 'Insulin', 'SkinThickness', 'BloodPressure']:
print(df.loc[df[i] == 0].index)
Int64Index([75, 182, 342, 349, 502], dtype='int64')
Int64Index([9, 49, 60, 81, 145, 371, 426, 494, 522, 684, 706],
dtype='int64')
Int64Index([ 0, 1, 2, 5, 7, 9, 10, 11, 12, 15,
...
754, 756, 757, 758, 759, 761, 762, 764, 766, 767],
dtype='int64', length=374)
Int64Index([ 2, 5, 7, 9, 10, 11,
...
734, 739, 743, 749, 750, 757,
12,
758,
15,
759,
17,
762,
21,
766],
dtype='int64', length=227)
Int64Index([ 7, 15, 49, 60, 78, 81, 172, 193, 222, 261, 266,
269, 300,
332, 336, 347, 357, 426, 430, 435, 453, 468, 484, 494,
522, 533,
535, 589, 601, 604, 619, 643,
dtype='int64')
697, 703, 706],
Treating erroneous data with Median
for i in ['Glucose', 'BMI', 'Insulin', 'SkinThickness', 'BloodPressure']:
df[i] = df[i].replace(0, df[i].median()) df.info()
<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 float64
5 BMI 768 non-null float64

```

```

6 DiabetesPedigreeFunction 768 non-null float64
7 Age 768 non-null int64
8dtypes: float64(3), Outcome int64(6) 768 non-null int64
memory usage: 54.1 KB
df.iloc[75]
Pregnancies 1.00
Glucose 117.00
BloodPressure 48.00
SkinThickness 20.00
Insulin 30.50
BMI 24.70
DiabetesPedigreeFunction 0.14
Age 22.00
Outcome
Name: 75, dtype: float64
0.00
Checking if the values are applied
df[:1]
Pregnancies Glucose BloodPressure SkinThickness Insulin
BMI \
0 6 148 72 35 30.5 33.6
DiabetesPedigreeFunction Age Outcome 0
0.627 50 1
df.nunique()
Pregnancies 17
Glucose 135
BloodPressure 46
SkinThickness 50
Insulin 186
BMI 247
DiabetesPedigreeFunction 517
Age 52
Outcome 2
dtype: int64
Data Types and the Count of Variables
df.dtypes.value_counts().plot(kind = 'bar', color = 'green').grid(False)
# df.to_csv('health care diabetes(No error).csv', index = False)
EDA
df['Outcome'].value_counts(normalize = True)
0 0.651042
1 0.348958
Name: Outcome, dtype: float64 sns.countplot(df['Outcome'], palette =
'deep')
<AxesSubplot:xlabel='Outcome', ylabel='count'>

```



```

sns.pairplot(df)
<seaborn.axisgrid.PairGrid at 0x20ba3c55580>
Individuals with Glucose levels greater than 140 are more susceptible of Diabetic conditions.
sns.scatterplot(data = df, x = 'Glucose', y = 'Insulin', hue = 'Outcome')
plt.legend(loc = 'upper right', labels = ['Diabetic','Non-Diabetic'])
<matplotlib.legend.Legend at 0x20ba840f5b0>
Individuals with greater SkinThickness show higher diabetic conditions.
sns.scatterplot(data = df, x = 'Glucose', y = 'SkinThickness', hue = 'Outcome')
plt.legend(loc = 'upper right', labels = ['Diabetic','Non-Diabetic'])
<matplotlib.legend.Legend at 0x20ba8eda1c0>
A higher BMI could be very well be a case of Diabetes.
sns.scatterplot(data = df, x = 'Glucose', y = 'BMI', hue = 'Outcome') plt.legend(loc = 'upper
right', labels = ['Diabetic','Non-Diabetic'])
<matplotlib.legend.Legend at 0x20ba8f2db80>
Diabetic patients do show a greater blood pressure levels.
sns.scatterplot(data = df, x = 'Glucose', y = 'BloodPressure', hue = 'Outcome')
plt.legend(loc = 'upper right', labels = ['Diabetic','Non-Diabetic'])
<matplotlib.legend.Legend at 0x20ba8fa7310>
Higher Blood Pressure doesn't necessarily mean a spike in Insulin Levels.
sns.scatterplot(data = df, x = 'Insulin', y = 'BloodPressure', hue = 'Outcome')
plt.legend(loc = 'upper right', labels = ['Diabetic','Non-Diabetic'])
<matplotlib.legend.Legend at 0x20ba84de0a0>
df.corr() SkinThickness
\
Pregnancies Glucose BloodPressure
Pregnancies
0.032568
Glucose
1.000000
0.128213
0.128213
1.000000
0.208615
0.218937
0.172143
BloodPressure 0.208615 0.218937 1.000000
0.147809
SkinThickness 0.032568 0.172143 0.147809
1.000000
Insulin -0.055697 0.357573 -0.028721
0.238188
BMI 0.021546 0.231400 0.281132
0.546951
DiabetesPedigreeFunction -0.033523 0.137327 -0.002378

```

```

0.142977
Age 0.544341 0.266909 0.324915
0.054514
Outcome 0.221898 0.492782 0.165723
0.189065
\
Insulin BMI DiabetesPedigreeFunction
Pregnancies -0.055697 0.021546 -0.033523
Glucose 0.357573 0.231400 0.137327
BloodPressure -0.028721 0.281132 -0.002378
SkinThickness 0.238188 0.546951 0.142977
Insulin 1.000000 0.189022 0.178029
BMI 0.189022 1.000000 0.153506
DiabetesPedigreeFunction 0.178029 0.153506 1.000000
Age -0.015413 0.025744 0.033561
Outcome 0.148457 0.312249 0.173844
Age Outcome
Pregnancies 0.544341 0.221898
Glucose 0.266909 0.492782
BloodPressure 0.324915 0.165723
SkinThickness 0.054514 0.189065
Insulin -0.015413 0.148457
BMI 0.025744 0.312249
DiabetesPedigreeFunction 0.033561 0.173844
Age 1.000000 0.238356
Outcome 0.238356 1.000000
plt.figure(figsize = (8,8))
sns.heatmap(df.corr(), annot = True, center = 1, cbar = False)
# sns.heatmap(df.corr(), annot = True, cmap = 'BuPu', cbar = False)
<AxesSubplot:>
df.corr().to_csv('correlation matrix.csv')
Predictive Analysis
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, precision_recall_curve
from sklearn.metrics import roc_curve, auc, roc_auc_score
X = df.drop(['Outcome'], axis = 1) y =
df['Outcome']
Random State 12 yielded the best output in every Category
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 12,
stratify = y)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)

```

```

print(y_test.shape)
(614, 8)
(154, 8)
(614,)
(154,)
Scaling
from sklearn.preprocessing import StandardScaler sc =
StandardScaler()
X_train = sc.fit_transform(X_train) X_test =
sc.transform(X_test)
Logistic Regression
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
log.fit(X_train, y_train)
LogisticRegression()
y_pred = log.predict(X_test) y_pred_train =
log.predict(X_train)
print(accuracy_score(y_train, y_pred_train))
print(accuracy_score(y_test, y_pred))
0.7719869706840391
0.7727272727272727
print('Train', classification_report(y_train, y_pred_train)) print('Test',
classification_report(y_test, y_pred))
Train precision recall f1-score support
0 0.79 0.89 0.84 400
1 0.73 0.55 0.63 214
accuracy 0.77 614
macro avg 0.76 0.72 0.73 614
weighted avg 0.77 0.77 0.76 614
Test precision recall f1-score support
0 0.81 0.85 0.83 100
1 0.69 0.63 0.66 54
accuracy 0.77 154
macro avg 0.75 0.74 0.74 154
weighted avg 0.77 0.77 0.77 154
sns.heatmap(confusion_matrix(y_train, y_pred_train), annot = True, fmt
= 'g')
<AxesSubplot:>
sns.heatmap(confusion_matrix(y_test, y_pred), annot = True)
<AxesSubplot:>
probs_log = log.predict_proba(X_test) probs_log =
probs_log[:, 1]
auc_log = roc_auc_score(y_test, probs_log) print('AUC:
%.3f' %auc_log)

```

```

fpr, tpr, thresholds = roc_curve(y_test, probs_log)
plt.plot([0, 1], [0, 1], linestyle='--', label = 'Logistic Regression (auc = %0.3f)' % auc_log)
plt.plot(fpr, tpr, marker='.') plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve"); AUC: 0.870
KNN
from sklearn.neighbors import KNeighborsClassifier
• n_neighbors 13 yielded the best possible results,
• 'brute' algorithm gave the best results in my testing
• in order to prevent overfitting/underfitting the following set was chosen
knn = KNeighborsClassifier(n_neighbors = 13, metric = 'euclidean', algorithm = 'brute')
knn.fit(X_train, y_train)
KNeighborsClassifier(algorithm='brute', metric='euclidean', n_neighbors=13)
knn_pred = knn.predict(X_test) knn_train =
knn.predict(X_train)
print(accuracy_score(y_train, knn_train))
print(accuracy_score(y_test, knn_pred))
0.7785016286644951
0.7727272727272727
print('Train', classification_report(y_train, knn_train)) print('Test'
,classification_report(y_test, knn_pred))
Train precision recall f1-score support
0 0.80 0.88 0.84 400
1 0.72 0.59 0.65 214
accuracy 0.78 614
macro avg 0.76 0.74 0.74 614
weighted avg 0.77 0.78 0.77 614
Test precision recall f1-score support
0 0.80 0.87 0.83 100
1 0.71 0.59 0.65 54
accuracy 0.77 154
macro avg 0.75 0.73 0.74 154
weighted avg 0.77 0.77 0.77 154
sns.heatmap(confusion_matrix(y_train, knn_train), annot = True, fmt = 'g')
<AxesSubplot:>
sns.heatmap(confusion_matrix(y_test, knn_pred), annot = True, fmt = 'g')
<AxesSubplot:>
probs_knn = knn.predict_proba(X_test) probs_knn
= probs_knn[:, 1]
auc_knn = roc_auc_score(y_test, probs_knn) print('AUC:
%.3f' % auc_knn)
fpr, tpr, thresholds = roc_curve(y_test, probs_knn)
plt.plot([0, 1], [0, 1], linestyle='--', label = 'KNN (auc = %0.3f)'

```

```

% auc_knn)
plt.plot(fpr, tpr, marker='.') plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve"); AUC: 0.842
SVM
from sklearn.svm import SVC
• kernel = linear proved to better suited than default rbf, poly and signoid
• probability is True for ROC-AUC Graph to work
model = SVC(kernel = 'linear', degree = 5, probability = True) model.fit(X_train,
y_train)
SVC(degree=5, kernel='linear', probability=True) model_pred =
model.predict(X_test)
model_train = model.predict(X_train)
print(accuracy_score(y_train, model_train))
print(accuracy_score(y_test, model_pred))
0.7703583061889251
0.7792207792207793
print('Train', classification_report(y_train, model_train)) print('Test'
,classification_report(y_test, model_pred))
Train precision recall f1-score support
0 0.78 0.90 0.84 400
1 0.74 0.53 0.62 214
accuracy 0.77 614
macro avg 0.76 0.72 0.73 614
weighted avg 0.77 0.77 0.76 614
Test precision recall f1-score support
0 0.81 0.86 0.83 100
1 0.71 0.63 0.67 54
accuracy 0.78 154
macro avg 0.76 0.74 0.75 154
weighted avg 0.78 0.78 0.78 154
sns.heatmap(confusion_matrix(y_train, model_train), annot = True, fmt
= 'g')
<AxesSubplot:>
sns.heatmap(confusion_matrix(y_test, model_pred), annot = True)
<AxesSubplot:>
probs_model = model.predict_proba(X_test)
probs_model = probs_model[:, 1]
auc_model = roc_auc_score(y_test, probs_model) print('AUC:
%.3f' % auc_model)
fpr, tpr, thresholds = roc_curve(y_test, probs_model)
plt.plot([0, 1], [0, 1], linestyle='--', label = 'SVM (auc = %0.3f)'
% auc_model)

```

```

plt.plot(fpr, tpr, marker='.') plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve"); AUC: 0.861
Decision Tree
from sklearn.tree import DecisionTreeClassifier
• Entropy gave the best Overfitting/Underfitting Results
dt = DecisionTreeClassifier(criterion = 'entropy', max_depth = 2, random_state = 3)
dt.fit(X_train, y_train)
DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=3)
dt_pred = dt.predict(X_test) dt_train =
dt.predict(X_train)
print('Train :', accuracy_score(y_train, dt_train)) print('Test :'
,accuracy_score(y_test, dt_pred))
Train : 0.7719869706840391
Test : 0.7662337662337663
print('Train', classification_report(y_train, dt_train)) print('Test'
,classification_report(y_test, dt_pred))
Train precision recall f1-score support
0 0.79 0.88 0.83 400
1 0.72 0.57 0.63 214
accuracy 0.77 614
macro avg 0.76 0.72 0.73 614
weighted avg 0.77 0.77 0.76 614
Test precision recall f1-score support
0 0.78 0.89 0.83 100
1 0.72 0.54 0.62 54
accuracy 0.77 154
macro avg 0.75 0.71 0.72 154
weighted avg 0.76 0.77 0.76 154
sns.heatmap(confusion_matrix(y_train, dt_train), annot = True, fmt = 'g')
<AxesSubplot:>
sns.heatmap(confusion_matrix(y_test, dt_pred), annot = True, fmt = 'g')
<AxesSubplot:>
probs_dt = dt.predict_proba(X_test) probs_dt =
probs_dt[:, 1]
auc_dt = roc_auc_score(y_test, probs_dt) print('AUC:
%.3f' % auc_dt)
fpr, tpr, thresholds = roc_curve(y_test, probs_dt)
plt.plot([0, 1], [0, 1], linestyle='--', label = 'Decision Tree (auc =
%0.3f)' % auc_dt) plt.plot(fpr, tpr,
marker='.')
plt.legend()
plt.xlabel("False Positive Rate")

```

```

plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve"); AUC: 0.784
Random Forest
from sklearn.ensemble import RandomForestClassifier
• Number of trees is being set to 200
• Entropy proved to be better suited than Gini & Log_loss
rf = RandomForestClassifier(n_estimators = 200, criterion = 'entropy', random_state = 13,
max_depth = 2, max_features = 'log2') rf.fit(X_train, y_train)
RandomForestClassifier(criterion='entropy', max_depth=2,
max_features='log2',
n_estimators=200, random_state=13)
rf_pred = rf.predict(X_test) rf_train =
rf.predict(X_train)
print('Train :', accuracy_score(y_train, rf_train)) print('Test :'
,accuracy_score(y_test, rf_pred))
Train : 0.7768729641693811
Test : 0.7662337662337663
print('Train', classification_report(y_train, rf_train)) print('Test'
,classification_report(y_test, rf_pred))
Train precision recall f1-score support
0 0.77 0.94 0.85 400
1 0.80 0.48 0.60 214
accuracy 0.78 614
macro avg 0.78 0.71 0.72 614
weighted avg 0.78 0.78 0.76 614
Test precision recall f1-score support
0 0.76 0.93 0.84 100
1 0.78 0.46 0.58 54
accuracy 0.77 154
macro avg 0.77 0.70 0.71 154
weighted avg 0.77 0.77 0.75 154
sns.heatmap(confusion_matrix(y_train, rf_train), annot = True, fmt = 'g')
<AxesSubplot:>
sns.heatmap(confusion_matrix(y_test, dt_pred), annot = True, fmt = 'g')
<AxesSubplot:>
prob_rf = rf.predict_proba(X_test) prob_rf =
prob_rf[:, 1]
auc_rf = roc_auc_score(y_test, prob_rf) print('AUC:
%.3f' % auc_rf)
fpr, tpr, thresholds = roc_curve(y_test, prob_rf)
plt.plot([0, 1], [0, 1], linestyle='--', label = 'Decision Tree (auc =
%0.3f)' % auc_rf) plt.plot(fpr, tpr,
marker='.')
plt.legend()

```

```

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve"); AUC: 0.866
Best ROC-AUC Performance
randomforest = 'AUC: %.3f' % auc_rf
decisiontree = 'AUC: %.3f' % auc_dt svm = 'AUC:
%.3f' % auc_model
KNN = 'AUC: %.3f' % auc_knn Log =
'AUC: %.3f' % auc_log
plt.figure(figsize = (15,5))
plt.barh(randomforest, label = 'Random Forest', width = 0.5) plt.barh(KNN, label =
'KNN', width = 0.5)
plt.barh(svm, label = 'SVM', width = 0.5)
plt.barh(Log, label = 'Logistic Regression', width = 0.5) plt.barh(decisiontree, label =
'Decision Tree', width = 0.5)
plt.legend()

```

<matplotlib.legend.Legend at 0x20bad1b29d0>

As per the results of ROC-AUC Curve, Logistic Regression and Support Vector Machine both perform better than K-Nearest Neighbor. Random Forest Classifier and Decision Tree Classifier both failed to provide the desired output.

TABLEAU

TABLEAU LINK:

[https://public.tableau.com/shared/WSBH2B3ND?:display\\_count=n&:origin=viz\\_share\\_link](https://public.tableau.com/shared/WSBH2B3ND?:display_count=n&:origin=viz_share_link)  
ChatGPT

It seems like you've done a comprehensive analysis of the healthcare dataset, including data exploration, preprocessing, and model building. Here's a summary of the key points and findings from your analysis:

Data Exploration:

Descriptive analysis: Checked for missing values and explored the distribution of variables. Visually explored variables using histograms and scatter plots to understand their distributions and relationships.

Conducted correlation analysis using a heatmap to identify relationships between variables.

Data Preprocessing:

Identified and treated erroneous values (zeros) in certain columns by replacing them with the median of each respective column.

Ensured data types were appropriate for analysis.

Data Modeling:

Used various classification algorithms including Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Decision Tree, and Random Forest.



Evaluated models using accuracy scores, classification reports, confusion matrices, and ROC-AUC curves.

Selected the best performing model based on the ROC-AUC metric, which was Logistic Regression and SVM.

Tableau Visualization:

Provided a Tableau visualization link for further exploration and analysis of the dataset.

Overall, your analysis demonstrates a thorough understanding of the dataset and the application of various machine learning algorithms for predictive modeling. Keep up the good work! If you have any further questions or need additional assistance, feel free to ask.

## Description

To create a dynamic and responsive Java e-healthcare web application for ordering medicines.

## Prerequisites

- Angular
- Java
- MySQL
- Spring Boot
- Jenkins
- Docker
- AWS
- Git and GitHub

## Motivation

**Problem Statement:** In this project, you should be able to develop a front-end web app using Angular and a Java back-end using Spring Boot, configure the applications using Docker containers, and deploy the project on AWS using the CI/CD pipeline.

Real-World Scenario: Medicare is a company that supplies medicines and other healthcare essentials at an affordable price. They found that online ordering of medicines with companies such as 100mg and Mfine is gaining more profits by eliminating middlemen. As a result, the team decided to hire a full-stack developer to develop a healthcare web application with a rich and user-friendly interface.

Project Plan:

Requirement Gathering: Understand the specific requirements and features needed for the healthcare web application. This includes features like user authentication, medicine catalog, order management, etc.

Designing the Database Schema: Define the database schema to store user information, medicine details, orders, etc. Choose MySQL as the database management system.

Setting Up the Backend: Use Spring Boot to create the backend of the application. Implement RESTful APIs to handle requests from the frontend, including user authentication, fetching medicine details, placing orders, etc.

Implementing the Frontend: Utilize Angular to develop the frontend of the application. Design a user-friendly interface for browsing medicines, placing orders, managing user accounts, etc.

Integration and Testing: Integrate the frontend with the backend APIs. Perform thorough testing to ensure all features are working as expected and handle any bugs or issues encountered.

Containerization with Docker: Dockerize both the frontend and backend applications to create portable, self-sufficient containers. This ensures consistency in deployment across different environments.

Setting Up CI/CD Pipeline: Use Jenkins to automate the CI/CD process. Configure pipelines to automatically build, test, and deploy the application whenever changes are pushed to the repository.

Deployment on AWS: Deploy the Docker containers on AWS cloud infrastructure. Utilize services like Amazon EC2 for hosting containers, Amazon RDS for database management, and Amazon S3 for storing static assets.

Monitoring and Maintenance: Implement monitoring tools to track application performance and health. Set up alerts for any anomalies or errors. Regularly maintain and update the application to incorporate new features and security patches.

User Training and Support: Provide user training and documentation to ensure smooth adoption of the healthcare web application. Offer ongoing support to address any user queries or issues.

Feedback and Iteration: Gather feedback from users and stakeholders to identify areas for improvement. Iterate on the application based on feedback received to enhance user experience and functionality.

By following this project plan, you can successfully develop and deploy the healthcare web application for Medicare, meeting their requirements and ensuring a seamless user experience.

Backend (Spring Boot)

UserController.java

java

Copy code

@RestController

@RequestMapping("/api/users")

public class UserController {

    @Autowired

    private UserService userService;

    @PostMapping("/login")

    public ResponseEntity<?> loginUser(@RequestBody LoginRequest loginRequest) {

        // Logic to authenticate user

        UserDetails userDetails = userService.authenticateUser(loginRequest.getUsername(),

loginRequest.getPassword());

        return ResponseEntity.ok(userDetails);

    }

    // Other endpoints for user management

}

MedicineController.java

java

Copy code

@RestController

@RequestMapping("/api/medicines")

public class MedicineController {

    @Autowired

    private MedicineService medicineService;

    @GetMapping

    public ResponseEntity<List<Medicine>> getAllMedicines() {

```

        List<Medicine> medicines = medicineService.getAllMedicines();
        return ResponseEntity.ok(medicines);
    }

    // Other endpoints for medicine management
}

@Service
public class UserService {

    // Method to authenticate user
    public UserDetails authenticateUser(String username, String password) {
        // Implementation logic
    }

    // Other methods for user management
}
MedicineService.java

```

```

@Service
public class MedicineService {

    // Method to fetch all medicines
    public List<Medicine> getAllMedicines() {
        // Implementation logic
    }

    // Other methods for medicine management
}
Frontend (Angular)
app.component.html
html

```

```

<div>
    <h1>Welcome to Medicare</h1>
    <router-outlet></router-outlet>
</div>
user-login.component.html
html
Copy code
<form (ngSubmit)="login()">

```

```
<input type="text" placeholder="Username" [(ngModel)]="username" name="username"
required>
  <input type="password" placeholder="Password" [(ngModel)]="password" name="password"
required>
  <button type="submit">Login</button>
</form>
```

user-login.component.ts

typescript

Copy code

```
import { Component } from '@angular/core';
import { UserService } from './user.service';
```

```
@Component({
  selector: 'app-user-login',
  templateUrl: './user-login.component.html',
  styleUrls: ['./user-login.component.css']
})
export class UserLoginComponent {

  username: string;
  password: string;

  constructor(private userService: UserService) {}

  login() {
    this.userService.login(this.username, this.password).subscribe(
      response => {
        // Handle successful login
      },
      error => {
        // Handle login error
      }
    );
  }
}
```

}

user.service.ts

typescript

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
```

```

@Injectable({
  providedIn: 'root'
})
export class UserService {

  baseUrl = 'http://localhost:8080/api/users';

  constructor(private http: HttpClient) { }

  login(username: string, password: string): Observable<any> {
    const loginData = { username, password };
    return this.http.post(`${this.baseUrl}/login`, loginData);
  }
}

```

This is just a basic example to get you started. You would need to implement the remaining functionality for user and medicine management as per your project requirements. Additionally, don't forget to set up routing, authentication, and error handling in your Angular application.

User

Provide source code to implement the remaining functionality for user and medicine management as per your project requirements. Additionally, don't forget to set up routing, authentication, and error handling in your Angular application.

ChatGPT

Sure, I'll provide you with source code snippets to implement the remaining functionality for user and medicine management, as well as setting up routing, authentication, and error handling in your Angular application.

user.service.ts

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

```

```

@Injectable({
  providedIn: 'root'
})
export class UserService {

```

```

  baseUrl = 'http://localhost:8080/api/users';

```

```
constructor(private http: HttpClient) { }
```

```
login(username: string, password: string): Observable<any> {  
  const loginData = { username, password };  
  return this.http.post(`${this.baseUrl}/login`, loginData);  
}
```

```
register(user: any): Observable<any> {  
  return this.http.post(`${this.baseUrl}/register`, user);  
}
```

```
// Add other user management methods like updateUser, deleteUser, etc.
```

```
}
```

user-register.component.html

html

Copy code

```
<form (ngSubmit)="register()">  
  <input type="text" placeholder="Username" [(ngModel)]="username" name="username"  
required>  
  <input type="password" placeholder="Password" [(ngModel)]="password" name="password"  
required>  
  <!-- Add more fields for registration -->  
  <button type="submit">Register</button>  
</form>
```

user-register.component.ts

typescript

Copy code

```
import { Component } from '@angular/core';  
import { UserService } from './user.service';
```

```
@Component({  
  selector: 'app-user-register',  
  templateUrl: './user-register.component.html',  
  styleUrls: ['./user-register.component.css']  
})
```

```
export class UserRegisterComponent {
```

```
  username: string;
```

```
  password: string;
```

```
  constructor(private userService: UserService) { }
```

```
  register() {  
    const user = { username: this.username, password: this.password };
```

```

    this.userService.register(user).subscribe(
      response => {
        // Handle successful registration
      },
      error => {
        // Handle registration error
      }
    );
  }
}

```

}  
 Medicine Management

medicine.service.ts

typescript

Copy code

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

```

```

@Injectable({
  providedIn: 'root'
})
export class MedicineService {

```

```

  baseUrl = 'http://localhost:8080/api/medicines';

```

```

  constructor(private http: HttpClient) { }

```

```

  getAllMedicines(): Observable<any> {
    return this.http.get(this.baseUrl);
  }

```

// Add other medicine management methods like addMedicine, updateMedicine,  
 deleteMedicine, etc.

```

}

```

Routing

app-routing.module.ts

typescript

Copy code

```

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { UserLoginComponent } from './user-login/user-login.component';
import { UserRegisterComponent } from './user-register/user-register.component';
import { MedicineListComponent } from './medicine-list/medicine-list.component';

```



```
const routes: Routes = [
  { path: 'login', component: UserLoginComponent },
  { path: 'register', component: UserRegisterComponent },
  { path: 'medicines', component: MedicineListComponent },
  { path: '', redirectTo: '/login', pathMatch: 'full' }, // Default route
];
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
```

```
export class AppRoutingModule { }
```

Authentication

auth.guard.ts

typescript

Copy code

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import { AuthService } from './auth.service';
```

```
@Injectable({
  providedIn: 'root'
})
```

```
export class AuthGuard implements CanActivate {
```

```
  constructor(private authService: AuthService, private router: Router) { }
```

```
  canActivate(): boolean {
    if (this.authService.isAuthenticated()) {
      return true;
    } else {
      this.router.navigate(['/login']);
      return false;
    }
  }
}
```

```
}
```

auth.service.ts

```
import { Injectable } from '@angular/core';
```

```
@Injectable({
```

```

    providedIn: 'root'
  })
  export class AuthService {

    isAuthenticated(): boolean {
      // Add logic to check if user is authenticated (e.g., check token)
      return true; // For demonstration purposes, always return true
    }
  }

```

```

}

```

Error Handling

error-interceptor.ts

typescript

Copy code

```

import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent, HttpResponse } from
'@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { Router } from '@angular/router';

```

```

@Injectable({
  providedIn: 'root'
})
export class ErrorInterceptor implements HttpInterceptor {

```

```

  constructor(private router: Router) { }

```

```

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    return next.handle(request).pipe(
      catchError((error: HttpResponse) => {
        if (error.status === 401) {
          // Handle unauthorized error (e.g., redirect to login page)
          this.router.navigate(['/login']);
        }
        return throwError(error);
      })
    );
  }

```

```

}

```

app.module.ts

typescript

Copy code

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { UserLoginComponent } from './user-login/user-login.component';
import { UserRegisterComponent } from './user-register/user-register.component';
import { MedicineListComponent } from './medicine-list/medicine-list.component';
import { ErrorInterceptor } from './error-interceptor';

```

```

@NgModule({
  declarations: [
    AppComponent,
    UserLoginComponent,
    UserRegisterComponent,
    MedicineListComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

This code provides a foundation for implementing the remaining functionality for user and medicine management, setting up routing, authentication, and error handling in your Angular application. You would need to further customize and expand upon these components and services according to your specific project requirements.

User

further customize and expand upon these components and services according to specific project requirements.

ChatGPT

Certainly! Let's further customize and expand upon the components and services based on the specific project requirements.

User Management

user.service.ts

typescript

Copy code

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class UserService {

  baseUrl = 'http://localhost:8080/api/users';

  constructor(private http: HttpClient) { }

  login(username: string, password: string): Observable<any> {
    const loginData = { username, password };
    return this.http.post(`${this.baseUrl}/login`, loginData);
  }

  register(user: any): Observable<any> {
    return this.http.post(`${this.baseUrl}/register`, user);
  }

  updateUser(userId: number, userData: any): Observable<any> {
    return this.http.put(`${this.baseUrl}/${userId}`, userData);
  }

  deleteUser(userId: number): Observable<any> {
    return this.http.delete(`${this.baseUrl}/${userId}`);
  }
}

```

Medicine Management

medicine.service.ts

typescript

Copy code

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class MedicineService {

  baseUrl = 'http://localhost:8080/api/medicines';

```

```
constructor(private http: HttpClient) { }
```

```
getAllMedicines(): Observable<any> {  
  return this.http.get(this.baseUrl);  
}
```

```
getMedicineById(medicineId: number): Observable<any> {  
  return this.http.get(`${this.baseUrl}/${medicineId}`);  
}
```

```
addMedicine(medicineData: any): Observable<any> {  
  return this.http.post(this.baseUrl, medicineData);  
}
```

```
updateMedicine(medicineId: number, medicineData: any): Observable<any> {  
  return this.http.put(`${this.baseUrl}/${medicineId}`, medicineData);  
}
```

```
deleteMedicine(medicineId: number): Observable<any> {  
  return this.http.delete(`${this.baseUrl}/${medicineId}`);  
}  
}
```

Routing

app-routing.module.ts

```
import { NgModule } from '@angular/core';  
import { RouterModule, Routes } from '@angular/router';  
import { UserLoginComponent } from './user-login/user-login.component';  
import { UserRegisterComponent } from './user-register/user-register.component';  
import { MedicineListComponent } from './medicine-list/medicine-list.component';  
import { AuthGuard } from './auth.guard';
```

```
const routes: Routes = [  
  { path: 'login', component: UserLoginComponent },  
  { path: 'register', component: UserRegisterComponent },  
  { path: 'medicines', component: MedicineListComponent, canActivate: [AuthGuard] },  
  { path: '', redirectTo: '/login', pathMatch: 'full' }, // Default route  
];
```

```
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})
```

```
export class AppRoutingModule { }  
Authentication  
auth.service.ts  
typescript  
Copy code  
import { Injectable } from '@angular/core';
```

```
@Injectable({  
  providedIn: 'root'  
})  
export class AuthService {  
  
  private isAuthenticatedValue: boolean = false;  
  
  isAuthenticated(): boolean {  
    // Check if the user is authenticated (e.g., check token validity)  
    return this.isAuthenticatedValue;  
  }  
  
  setAuthenticated(status: boolean): void {  
    this.isAuthenticatedValue = status;  
  }  
}  
user-login.component.ts  
typescript
```

```
import { Component } from '@angular/core';  
import { Router } from '@angular/router';  
import { UserService } from '../user.service';  
import { AuthService } from '../auth.service';
```

```
@Component({  
  selector: 'app-user-login',  
  templateUrl: './user-login.component.html',  
  styleUrls: ['./user-login.component.css']  
})  
export class UserLoginComponent {  
  
  username: string;  
  password: string;  
  
  constructor(private router: Router, private userService: UserService, private authService:  
AuthService) { }
```

```

login() {
  this.userService.login(this.username, this.password).subscribe(
    response => {
      // Login successful, set authentication status and navigate to medicines
      this.authService.setAuthenticated(true);
      this.router.navigate(['/medicines']);
    },
    error => {
      // Handle login error
    }
  );
}
}

```

user-register.component.ts

typescript

Copy code

```

import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { UserService } from './user.service';

```

```

@Component({
  selector: 'app-user-register',
  templateUrl: './user-register.component.html',
  styleUrls: ['./user-register.component.css']
})

```

```

export class UserRegisterComponent {

```

```

  username: string;
  password: string;

```

```

  constructor(private router: Router, private userService: UserService) { }

```

```

  register() {
    const user = { username: this.username, password: this.password };
    this.userService.register(user).subscribe(
      response => {
        // Handle successful registration
        this.router.navigate(['/login']); // Redirect to login page after registration
      },
      error => {
        // Handle registration error
      }
    );
  }
}

```

```
}
```

```
}
```

Error Handling

error-interceptor.ts

```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent, HttpResponse } from
'@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { Router } from '@angular/router';
```

```
@Injectable({
  providedIn: 'root'
})
export class ErrorInterceptor implements HttpInterceptor {
```

```
  constructor(private router: Router) { }
```

```
  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    return next.handle(request).pipe(
      catchError((error: HttpResponse) => {
        if (error.status === 401) {
          // Handle unauthorized error (e.g., redirect to login page)
          this.router.navigate(['/login']);
        }
        return throwError(error);
      })
    );
  }
}
```

```
}
```

app.module.ts

typescript

Copy code

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { UserLoginComponent } from './user-login/user-login.component';
import { UserRegisterComponent } from './user-register/user-register.component';
```



```
import { MedicineListComponent } from './medicine-list/medicine-list.component';
import { ErrorInterceptor } from './error-interceptor';
import { AuthGuard } from './auth.guard';
import { AuthService } from './auth.service';
import { UserService } from './user.service';
import { MedicineService } from './medicine.service';
```

```
@NgModule({
  declarations: [
    AppComponent,
    UserLoginComponent,
    UserRegisterComponent,
    MedicineListComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true },
    AuthGuard,
    AuthService,
    UserService,
    MedicineService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

This expansion provides a more comprehensive implementation of user and medicine management, routing, authentication, and error handling in your Angular application. Adjustments and additional features can be made based on specific project requirements, such as adding more fields to user registration, implementing authorization, and enhancing error