INTERNSHIP AT PRIXA AI AS DATA SCIENTIST: E-RECOMMENDATION PRESCRIPTION

by

Dave Danadiva Agusta Peerera

12006002

Internship Report
Submitted to

BIOMEDICAL ENGINEERING STUDY PROGRAM

The Prominence Tower
Alam Sutera, Tangerang 15143
INDONESIA
February 2022

ABSTRACT

INTERNSHIP AT PRIXA AI AS DATA SCIENTIST:

E-RECOMMENDATION PRESCRIPTION

By

DAVE DANADIVA AGUSTA PEERERA

12006002

The internship report in broad contains 5 chapters in which the author discusses the project development of e-recommendation prescription. E-recommendation prescription is a feature in Prixa.ai's teleconsultation service with the goal to assist and provide doctors with a drug recommendation system that recommends medicines and its dosage for the consulting patient. The project development comprises data gathering, exploratory data analysis (EDA), data preprocessing and splitting, AI model development using collaborative filtering and k-nearest neighbor algorithm, evaluation with k-fold cross validation, and model refinement in a form of parameter tuning. The result of this project is a refined model chosen from the different trials of refinement scenarios. This model was deployed into a web application with patient information and differential diagnosis as the input and recommended medicines and their dosage as the output.

Keywords: e-recommendation prescription, collaborative filtering, k-nearest neighbor (KNN), k-fold cross validation.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF EQUATION

CHAPTER 1

INTRODUCTION

1.1 Background

Prixa.ai is a healthcare technology company based in Jakarta, with the mission to humanize healthcare through data and technology by providing a robust telemedicine experience to its users. This mission is realize through their products and services which enable users to utilize a self symptom checker with an output of disease with it's probability, a teleconsultation platform that connect patient directly to doctor via chat, and several other web based services, such as making appointments with doctor, lab test, and pharmacy delivery. These modes of services align with Prixa.ai's mission, aimed to make healthcare more available, faster, and reliable.

All of Prixa.ai's services are keep being improved and optimized to provide users with quality experience in accessing medication. In one part of it, the teleconsultation service has added a feature called "e-Recommendation Prescription" for doctors to be given recommendations of prescription for the patient consulting using the teleconsultation service. This prescription will include the recommended medicine along with its dosage, where this output is based on a model developed from previous patients' medical records and International Classification of Diseases 10th Revision (ICD-10).

The goal of e-Recommendation Prescription feature is to assist and provide doctors with drug recommendation systems which include choices of ready to use prescriptions made for the patient. With this goal in mind, the feature is meant to shorten the overall process of the teleconsultation. Approximately, doctors spent 23 minutes for each teleconsultation session, with 6 to 10 minutes working on

prescription. This is because doctors need to decide the right medicine and dosage for each patient based on their personal condition, such as weight, age, gender, etc. With the recommendation system, this part can be automated; hence not only shorten the time but also provide personalized suggestions for each patient. Despite the advantages, this feature is an optional choice to make by the doctor but it can give a starting point to make the best prescription for the patient.

The e-Recommendation Prescription is a data-driven feature, which utilizes data to develop the model. The utilization involves the process of gathering, preparing, exploring, and selecting data that are needed for the project. Furthermore, these data will also be used to develop AI models as a train and test data. For this reason, this is a project handled by the Prixa.ai Data Scientist Team.

1.2 Prixa.ai Organizational Structure

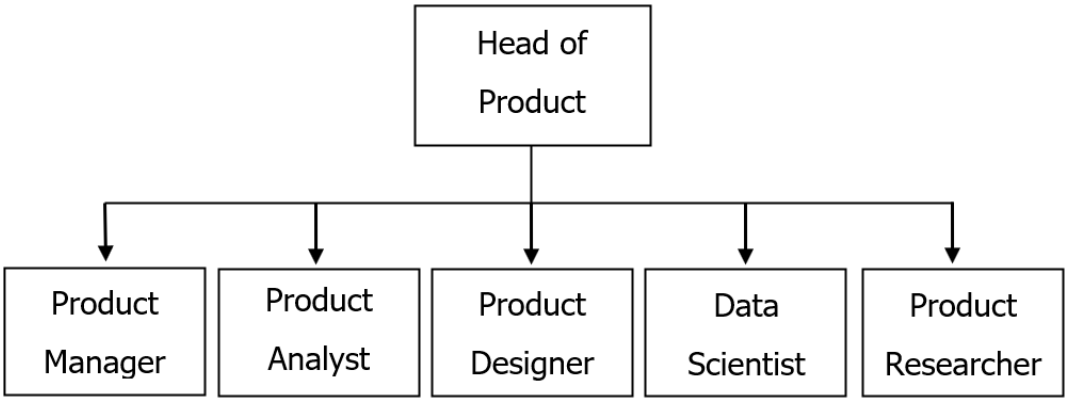

Figure 1.1 Prixa.ai Product Structure

Internship placement was in the Data Scientist Team, which currently is under the Head of Product as shown in figure 1.1.

1.3 Internship Purpose

Aligned with the Biomedical Engineering Department of SGU semester 3 learning goals, the internship program is meant for students to experience the real world working environment and to nurture skills by putting it into practice. This internship program opportunity gives the author an understanding of project workflow in a technology company, day-to-day tools used, and also techniques particularly in programming, all pour directly into a project-based learning experience. The main objective of the internship project is to retrace from the beginning, the development of an e-recommendation prescription system to assist doctors as a feature of the teleconsultation service.

1.4 Internship Boundaries

The boundaries of this internship project is focussed on computing power. Due to limited personal computing power, Google Colab as an alternative tool was used throughout the internship. This limitation affects the runtime when executing the program; hence standard short and long runtime need to be redefined when doing the model evaluation.

1.5 Methodology

The internship project was a retrace of an ongoing project done by the Prixa.ai Data Scientist Team at that moment, as it was in the middle period of its development. The development comprises data gathering and preparation, exploratory data analysis, data preprocessing, data splitting, AI model development, model evaluation, model refinement & selection, and lastly, model deployment. For the model deployment process, the model selected will be deployed to a personally made private website for a test purpose. All the tools used in the project development process are stated in table 1.1 and the methodology of each development stage will be explained subsequently in the subpart of subchapter 1.4.

Table 1.1 Tools Used in Project Development

Development Stage	Tools	Usage
Data Gathering	ArangoDB	Database
Data Cleaning - Model Refinement	Google Colab	IDE
	Python 3.7	Programming Language
Exploratory Data Analysis	Tableau	Data Visualization
Model Deployment	Visual Studio Code	IDE
	Python 3.10	Programming Language

1.5.1 Data Gathering and Data Cleaning

The first stage of the project development is data gathering and data cleaning. In this stage, Data needed for the development was gathered from ArangoDB as the database source used. ArangoDB is an open source database, which natively supports graph, document, and search. Then data gathered needs to be cleaned and prepared for the next process. Data cleaning in this development stage is done in Google Colab IDE in a form of Python notebook; hence python code is written in a customized cell. This gives the benefit of easier data observation. The cleaning process mostly is done using the python pandas library.

1.5.2 Exploratory Data Analysis (EDA)

The second stage is the exploratory data analysis, where in this stage, prepared data was explored to find correlation between each feature contained in the datasets. This stage will give tremendous insight regarding the data obtained. This stage also helps to identify, filter, and manage effectively the datasets that will be needed for the project development. The

EDA process was done using Tableau for desktop software, an interactive data visualization software, which enables users to find correlations between features conveniently and time-efficiently.

1.5.3 Data Preprocessing and Splitting

In this stage, datasets were once again filtered based on findings and correlation discovered from the previous stage (EDA). Not only filtered, additional data are also added into a certain dataset; hence make the development process more prepared. After being preprocessed, dataset used will be splitted into test and train set for building and testing the model

1.5.4 AI Model Development and Evaluation

During this stage, the predictor model was developed using a machine learning (ML) algorithm and was developed based on splitted data, where test set as the input data and train set as the available data. The model development was done by building functions following the chosen ML algorithms. Two ML models were developed with the same purpose and evaluated afterwards. To obtain insight about model performance, scoring was also done after each of the model development using K-Fold Cross Validation with the metrics being used are recall, precision, F1 score, and runtime. However, due to processing capabilities, prescription data was reduced to 80% of the original dataset size.

1.5.5 Model Refinement and Model Selection

After obtaining insight into a form of scoring metrics for each model developed, both models were refined to increase the metrics score by tuning the parameters used by the model. After obtaining the results of the

refinement, all tested scenarios were documented in a form tabulated data and compared. The best model was chosen based on the target requirements.

1.5.6 Model Deployment

In this stage, the chosen and refined model was deployed into a privately made web application. This was done using python streamlit library, an open-source python library which enables users to deploy ML models. However, since it is deployed for private and education purposes, the web application runs in the local host of the personal computer; hence locally saved IDEs need to be used. For this, Visual Studio Code was used with the python version installed is python 3.10.

CHAPTER 2

LITERATURE REVIEW

2.1 International Classification of Diseases (ICD)

In 1893, the World Health Organization (WHO) published the International List of Causes of Death, mainly to classify causes of injury and death to keep on track morbidity and mortality statically. In the following year of 1989, it was changed into the International Classification of Diseases (ICD) with the main function to categorize similar diseases with more specific conditions listed; hence able to promote international compatibility in collecting health data and reports.

The ICD has gone through numerous revisions with the latest one being published is the ICD-11 (11th revision). However, the most commonly used today is still the ICD-10 version (10th revision). Despite the version, in general, a list of all known injuries and diseases resides in ICD, along with each of them having descriptions of their diagnostic characteristic and unique identifier. This unique identifier is in a form of code, which appears in mortality data on death certificates as well as patient and clinical record morbidity data. ICD-10 code is labeled in a form of four-alphanumeric-character codes starting from A00.0 to Z99.0. The first letter indicates a different chapter, with 22 chapters in total (a group of letters are embedded in the same chapter together).

Table 2.1 Example of ICD-10 Code Chapter and Range

Chapter	Code Range	Description
1	A00-B99	Certain Infectious and Parasitic Diseases
6	G00-G99	Diseases of the Nervous System

2.2 Collaborative Filtering

Collaborative filtering is one example of a machine learning recommendation system algorithm that collaborates with agents, datasource, viewpoints, etc (Kumar, 2014). This algorithm carries out an underlying approach that if one person has a similar opinion with another person on a certain issue, then that one person would likely have the same opinion with the same person that they had on the previous issue on another different issue compared with a randomly chosen person. For this reason, it is called collaborative filtering, since it is a user-based approach which collaborates with the users as the agent.

The algorithm comes into use when data of users' historical preference for a certain item are available. This data will provide information about the different variety of users on a certain problem which can be differentiated based on chosen parameters. The bigger the dataset, the more it will provide the variety of users and as a result it can serve more accurate recommendation(s).

In general, the workflow of collaborative filtering systems consists of data collection, where users rate a specific item which later on can be used as a scoring parameter. However, the scoring parameter didn't have to be item rating from the user but it can also be the user information itself, which can lead them into choosing or being chosen for the item of choice. This scoring will be used to filter and obtain the pool of similar users, which then their preferences will be taken and transformed into recommendation(s) for the other similar users (Luo, 2018). The weighted scoring system itself can be vary based on importance of the parameter or by cosine similarity and after the scoring, result can be filtered by threshold that is setted based on preferred calculation method.

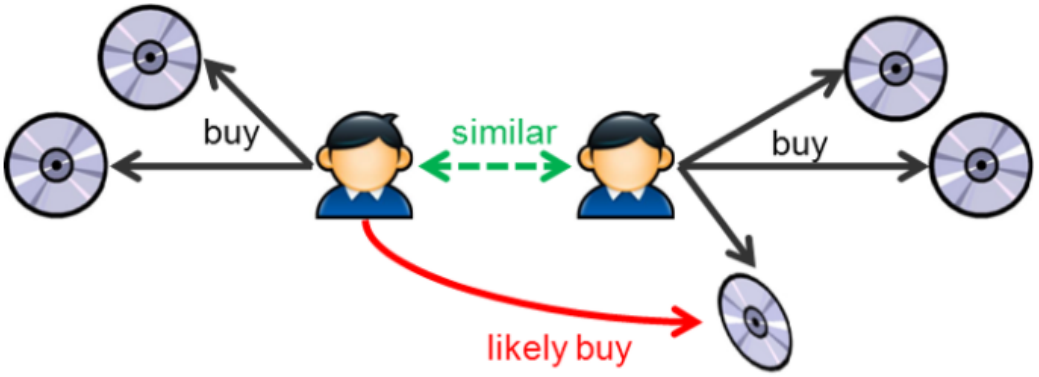

Figure 2.1 Collaborative Filtering Illustration
(Source: https://dzone.com/articles/recommendation-engine-models)

2.3 K-Nearest Neighbor (KNN)

Similar to collaborative filtering, K-Nearest Neighbor (KNN) is also a machine learning algorithm which is one example of a recommendation system algorithm. KNN is a user-based algorithm which works by grouping the input datum's with the other training samples by their distance with each other and determine the K-nearest neighbors. It then set the majority of these K-neighbor classes to be the prediction of the classification or in this case the recommendation (Lee, 2019).

For instance, can be seen in figure 2.2, if k for the nearest neighbors is set to 3, the closest neighbors to the input data points is 2 squares and 1 triangle. Based on KNN working mechanism, it will take the majority of classes among the 3 neighbors and set it as the prediction result. However, the number of neighbors (k) should also be taken into account. Currently, there is no absolute way to determine the number of neighbors before seeing the scoring result of the prediction. However, it can be taken approximately by taking the square root of the total sample in the dataset. After the prediction scoring result is obtained, the

number of neighbors can be tuned to obtain the optimal value based on the prediction requirements.

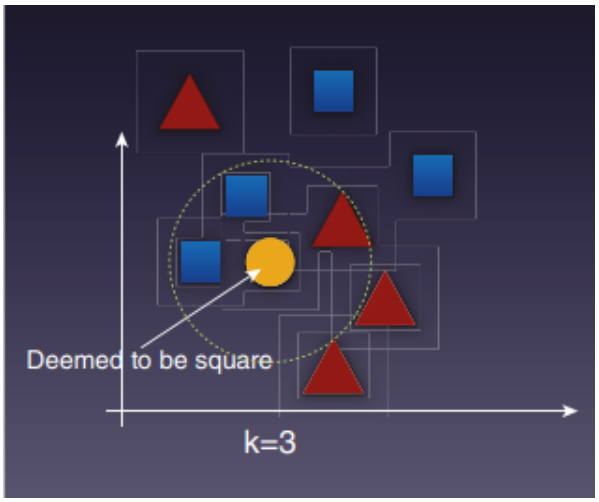

Figure 2.2 KNN Working Mechanism
(Source: Python Machine Learning, First Edition, Wei-Meng Lee.)

The number of neighbours for KNN algorithms affect the fitting of the model. Fitting describes how well the model fits into the training data. When k is increased, more data will be included as the neighbor of the input datum. As a result, it will make the model more robust to be used when applied to another dataset, this situation is considered as low variance. Although it makes the model versatile, increasing k too high might also cause the model to underfit, meaning it tries to overgeneralize the prediction, which is considered as high bias. On the other hand, when the number of k decreases, it will narrow the number of neighbors; hence will give a more accurate prediction with low bias. However, when it is too low, it can cause overfitting, where it only works greatly for the tested dataset and is considered to be high variance. For this reason, obtaining a model with as low bias and variance as possible by optimizing the value of K when using the KNN algorithm is important (Lee, 2019).

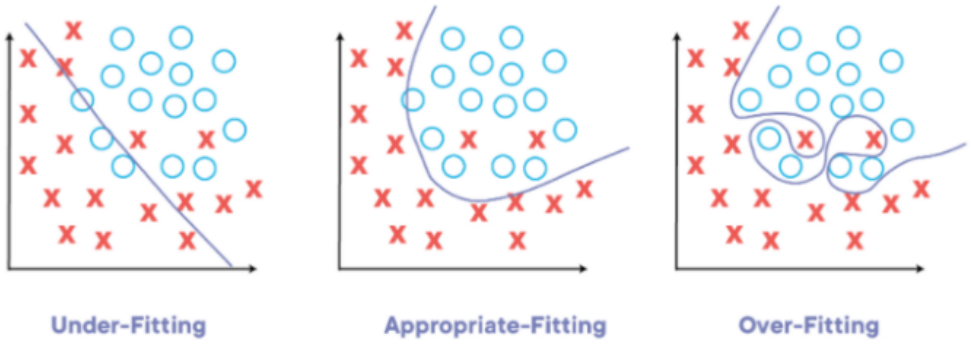

Figure 2.3 Overfitting, Underfitting, Good Fit Illustration
(Source: https://medium.com, Arman Hussein.)

2.4 K-Fold Cross Validation

In every development of a machine learning model, evaluating the model needs to be done in order to know how good the model performs. K-Fold Cross Validation is one of machine learning model evaluation techniques, which involves resampling procedures. In general, resampling procedure involves repeatedly drawing samples from a dataset and calculating the metrics on the samples taken; hence obtaining specific information needed. In machine learning algorithms, this information is what describes the model performance (Montero, 2021).

The main intention of doing K-Fold Cross Validation is to be able to develop a more generalized model, which can perform well on unseen data. This is why when testing a machine learning model, the dataset needs to be splitted into a train and test set. When using K-Fold Cross Validation, the set will be divided into k-number of folds with each containing nearly equal size of data. for each fold data will take a chance in a form of set into becoming a train and test set. The procedure will then be repeated for k times (Pramoditha, 2020). With this, all data in the dataset will be able to participate as the test set and the train set; hence doesn't leave any unseen data and give a more generalized scoring on the chosen metrics.

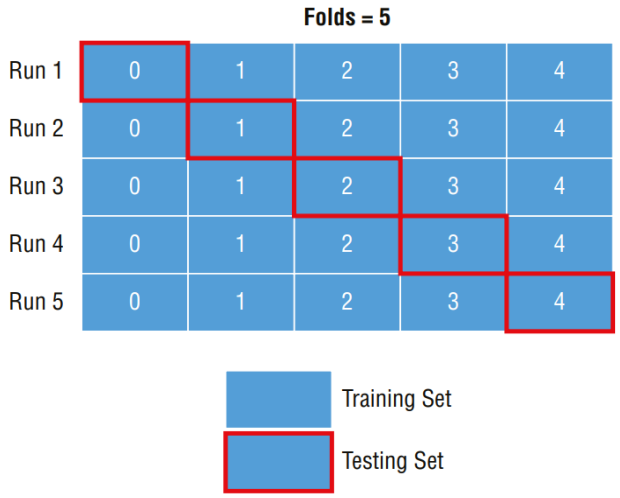

Figure 2.4 K-Fold Cross Validation Working Mechanism
(Source: Python Machine Learning, First Edition, Wei-Meng Lee.)

2.5 Performance Metrics

Performance metrics are used to know how well a product performs based on a certain criteria. In machine learning model development, model performance needs to be scored and this can be done using performance metrics. For binary classification, confusion matrix comes in to mapped the output result of a machine learning model. Confusion matrix is a size of 2x2 matrix, which sets the actual value on one axis and predicted value on the other axis (Jayaswal, 2020). As seen in table 2.2, Each individual cell in the table represents their own meaning. True Positive happens when the model developed is able to correctly predict the output to be positive. False positive happens when the model developed incorrectly predicts the output to be positive. Same goes to True negative, where the model correctly predicted the output to be negative and false negative with the model incorrectly predicted the output to be negative.

		Actual	
		0	1
Predicted	0	True Negative	False Negative
	1	False Positive	True Positive

Figure 2.5 Confusion Matrix

Based on the confusion matrix, scoring metrics can be obtained and can be applied to score the model performance. Few of these metrics are: accuracy (sum of all correct prediction), precision (the number of correct positive prediction among all positive outputs), Recall (The number of correctly predicted positive event among all outputs that should be positive), and F1 Score (the harmonic mean of precision and recall) (Lee, 2019). These all can be seen more clearly as shown in figure 2.7.

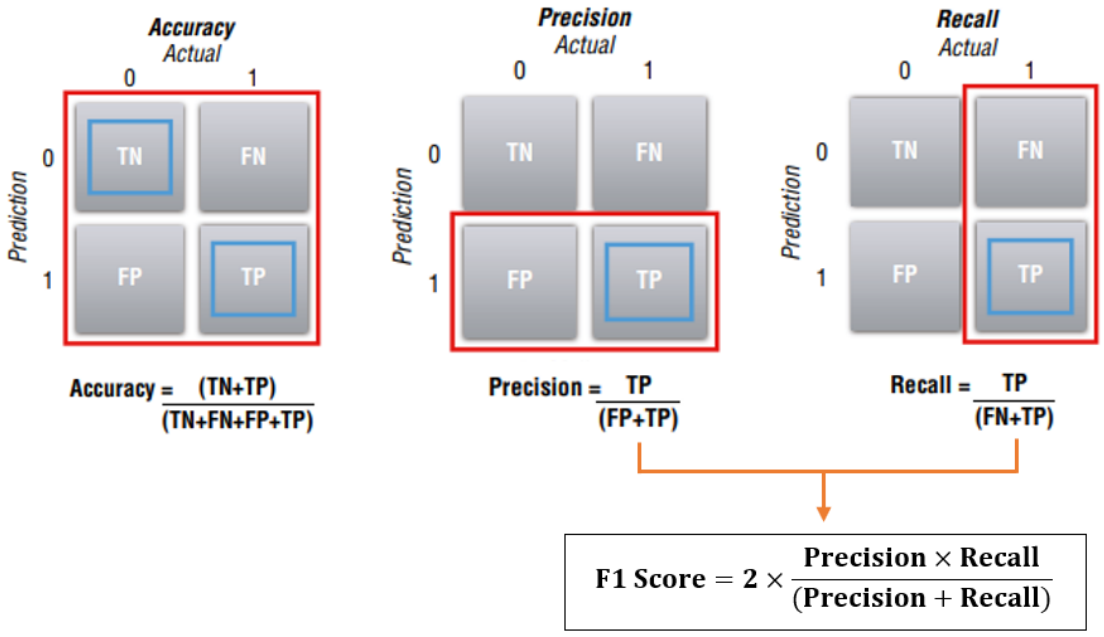

Figure 2.6 Performance Metrics

(Source: Python Machine Learning, First Edition, Wei-Meng Lee.)

CHAPTER 3

PROJECT DEVELOPMENT AND ANALYSIS

3.1 Data Gathering

All of the involved dataset in the development was gathered from ArangoDB. Dataset gathered is in javascript object notation (json) format and was converted into comma separated value (csv). Most of the dataset files used throughout the development will also be in csv except the "Top 50 ICD-10 and Medicines". List of data gathered can be seen in table 3.1 with it's content in brief.

After the datasets were gathered, they still need to be cleaned so it can be used throughout the development process. The type of data that needs to be excluded from the dataset is dummy data used by the product team to test the application and usually contain keywords such as 'dummy', 'test' ,'tech', 'fix', and the tester name. Also in the doctors data, dummy doctors were also needed to be excluded, which usually contain keywords mentioned before and the doctor tester name. Other than that, the patient dataset was also filtered based on the organization they came from and since the development will focus on prescription, patients who use the tele-consultation without obtaining prescription were also excluded. This exclusion applied to both prescription and pivot prescription reports.

Table 3.1 List of Data and it's Content

Data	Format	Number of Records	Content
doctors_051121	json	93	Prixa.ai's doctors data

formularies_121121	json	6	Prixa.ai's insurance partner formularies data
medicine_categories_111121	json	54	Category of all medicines contain in medicines_091121
medicine_formularies_121121	json	1419	Data of all medicines that is covered based on formulary provided by Prixa.ai's insurance partner
medicine_prescription_121121	json	25989	Data of all prescription for medicine prescribed
medicines_091121	json	1897	Data of all Prescribed medicine
patients_091121	json	185939	Data of all patients up to 9th November 2021
preconditions_051121	json	14047	Preconditions of patient following the patients_091121 data
prescription_pivot_report_051121	json	11358	Pivot table of the prescriptions_051121 data in more detail
prescriptions_051121	json	11203	Data of all prescriptions given to patient up to 5th

			November 2021
Top 50 ICD-10 and Medicines	xlsx	1514	Data of top 50 ICD-10 diseases and the prescribed medicine

More into the data gathered, the recommendation system will give recommended medicine based on International Classification of Diseases 10th revision (ICD-10), where the data obtained (Top 50 ICD-10 and Medicines) comprise of the list of top 50 ICD-10 with the list of medicines for each disease and number of it being prescribed, where the data had been consulted with Prixa.ai's doctor. However, this data was still in Excel spreadsheet (xlsx) format; hence it needs to be mapped and converted into json format for further usage. This data was mapped and converted into json format due to json's simple nature which consists of attribute-value pairs; hence it fits perfectly when considering the content of the data (as previously mentioned). For more clarity regarding the features inside each dataset, list of them can be seen in table 3.2. However, features listed are only the features that might and will be used throughout the whole project development.


```
"A09.9 Gastroenteritis and colitis of unspecified origin": {
    "ANTACID 200 MG/200 MG Tablet": "20",
    "ATTAPULGITE 600 MG Tablet": "18",
    "RANITIDINE 150 MG Tablet": "16",
    "PARACETAMOL 500 MG Tablet": "15",
    "GARAM ORALIT  Sachet": "11",
    "OMEPRAZOLE 20 MG Kapsul": "11",
    "DOMPERIDONE 10 MG Tablet": "10",
    "ZINC 20 MG Tablet": "10",
    "ZINC 20 MG/5 ML Botol": "10",
    "DOMPERIDONE 5 MG/ML Botol": "8"
},
```


Figure 3.1 Preview of Drug Mapping in json Format

Table 3.2 Data Gathered and the Important Features

Data	Important Features
doctors_051121	[‘id’]
formularies_121121	[‘id’, ‘name’]
medicine_categories_11121	[‘id’, ‘name’]
medicine_formularies_121121	[‘id’, ‘formulariId’, ‘medicineId’]
medicine_prescription_121121	[‘id’, ‘medicineId’, ‘prescriptionId’, ‘frequency’, ‘frequencyDd’, ‘timing’, ‘duration’, ‘durationUnit’, ‘amountUnit’]
medicines_091121	[‘id’, ‘categoryId’, ‘generic’, ‘brand’, ‘unit’]
patients_091121	[‘id’, ‘patientGender’, ‘patientWeight’, ‘patientHeight’, ‘patientAge’]
preconditions_051121	-
prescription_pivot_report_051121	[‘id’, ‘differentialDiagnosis1’, ‘differentialDiagnosis2’, ‘differentialDiagnosis3’, differentialDiagnosis4’, ‘differentialDiagnosis5’, ‘medicineBrand1’, ‘medicineBrand2’, ‘medicineBrand3’, ‘medicineBrand4’, ‘medicineBrand5’]
prescriptions_051121	[‘id’, ‘patientId’, ‘differentialDiagnosis’]
Top 50 ICD-10 and Medicines	[ICD name, medicine name, prescribed count] *listed names are not column name, since file is in json format*

3.2 Exploratory Data Analysis

After data was gathered and cleaned, prepared data was explored to find correlation between each feature contained in the datasets using Tableau. Important insights between features and datasets were discovered and can be divided into 2 parts, patient and prescription part and medicine part.

3.2.1 EDA of Patient and Prescription Data

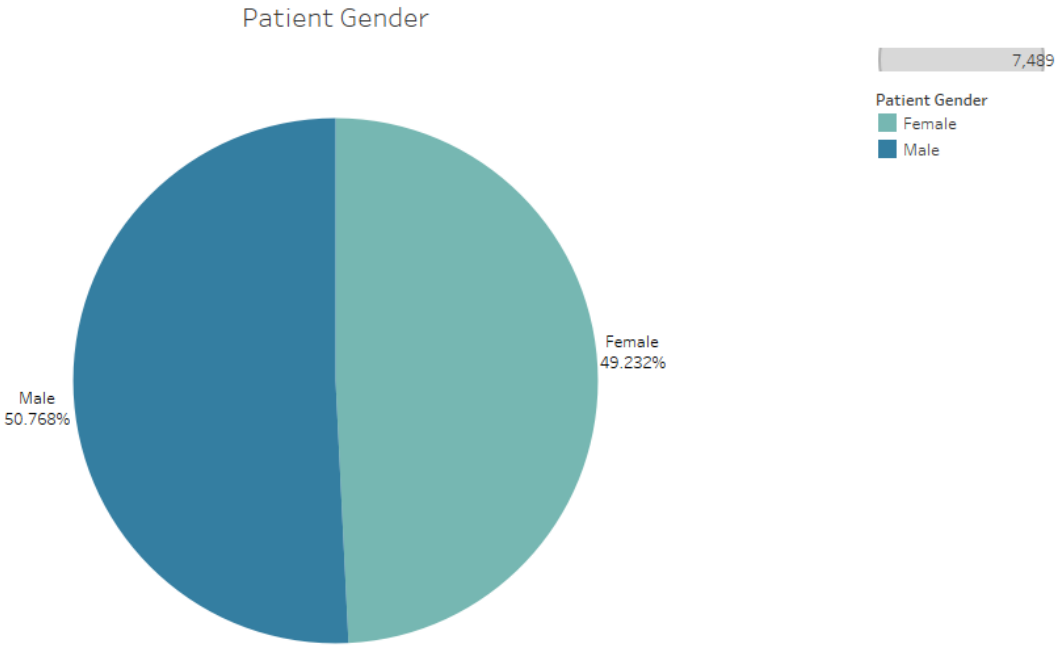

Figure 3.2 Patient Gender Distribution Chart

As seen in figure 3.2, Prixa.ai's patient gender distributions are almost equal between male and female. Since patient gender information may correlate and affect the medicine prescribed for each disease (either by dosage or type), it can also be taken into account as a determining parameter later on when developing the model.

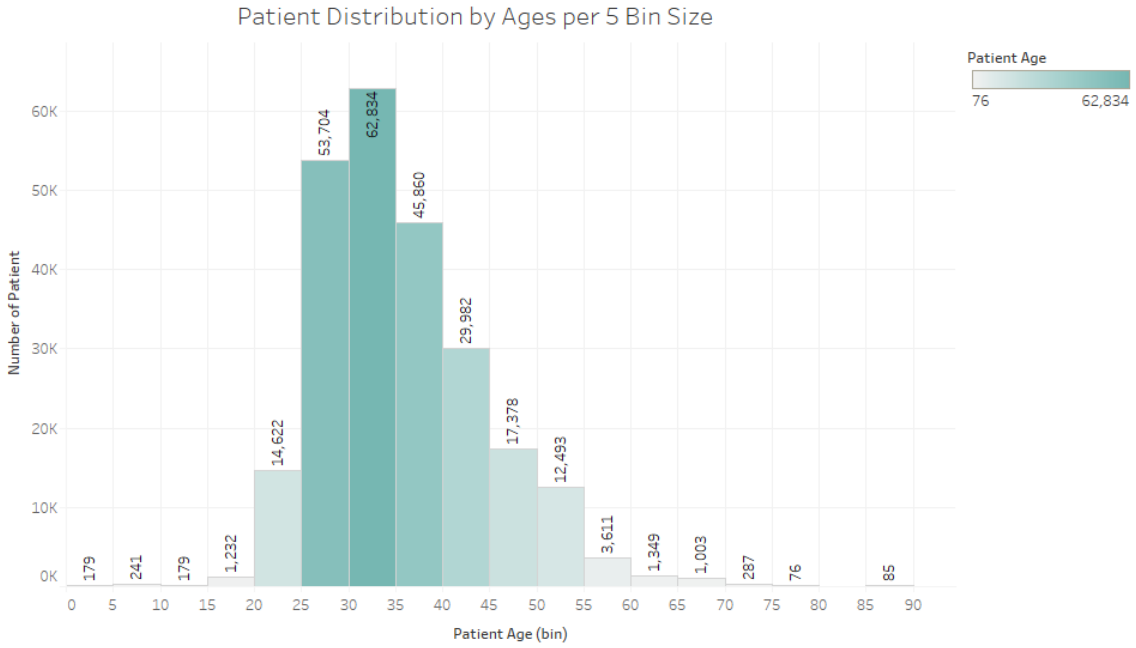

Figure 3.3 Patient Distribution by Ages per 5 Bin Size Graph

Previewed in figure 3.3 the distribution of Prixa.ai's patient distribution which are binned with the size of 5 ages. Statistically the graph is positively skewed (to the right), with the mode bin is the patients with the age of 30-35.

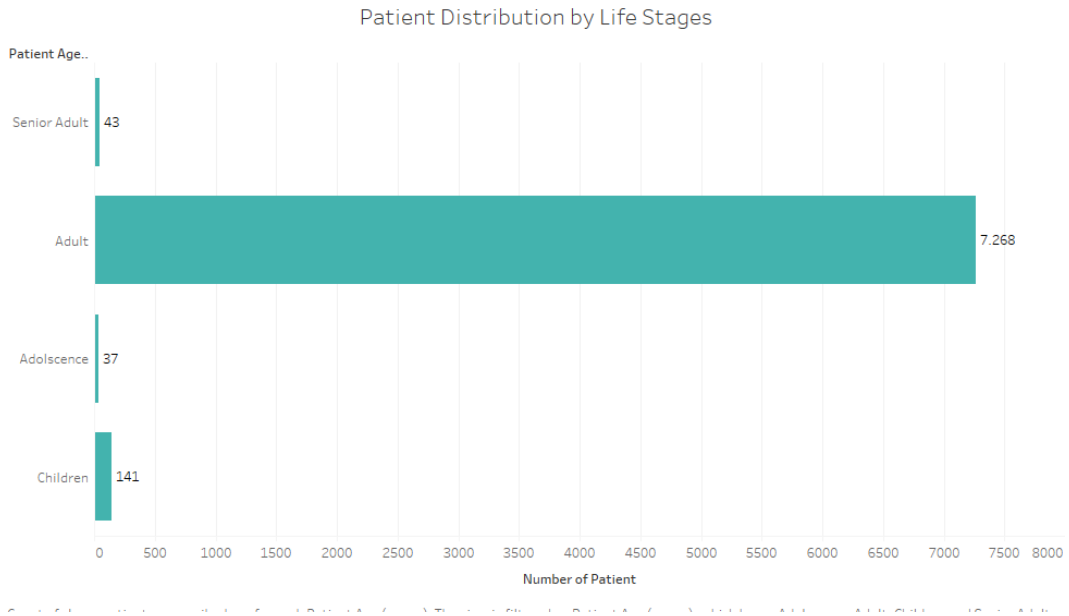

Figure 3.4 Patient Distribution by Life Stage Graph

Patient ages then were categorized into 4 life stages, which is children, adolescence, adults, and senior adults, with the distribution and details can be seen in figure 3.4. Based on the findings, most Prixa.ai's patients are in their adult life stage and this correlates with the graph shown in figure 3.3. Since patient age is really important, particularly when prescribing medicine (in terms of dosage and type) , categorizing patients' age will make identifying patient age easier later on if used as a parameter for the model development.

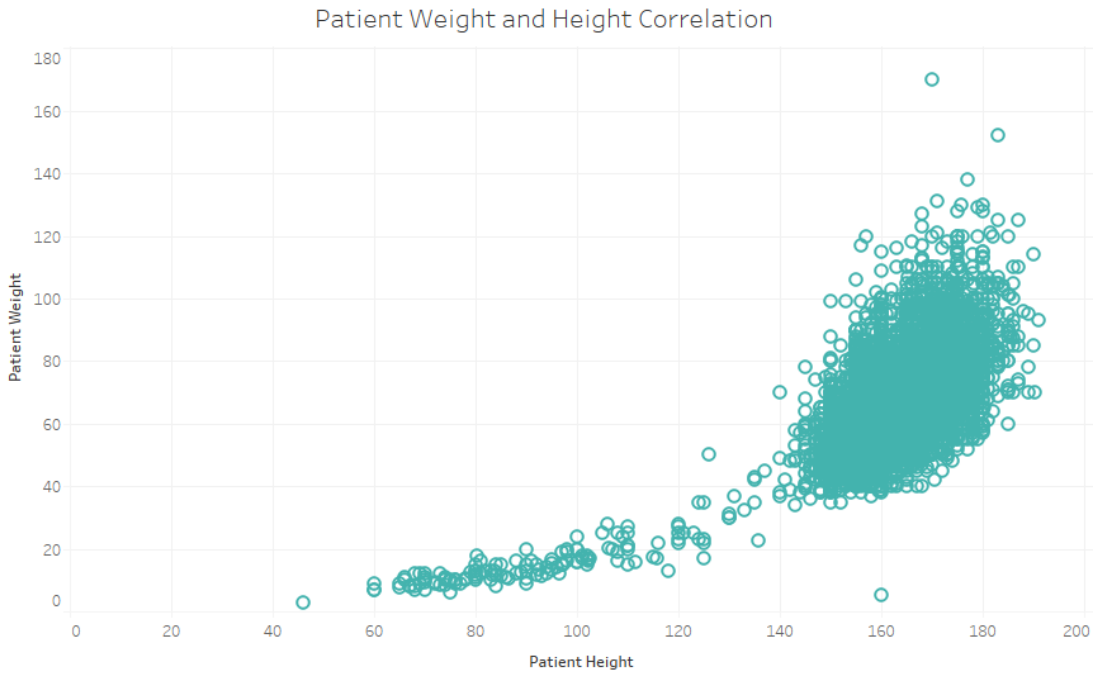

Figure 3.5 Patient Height and Weight Correlation

Shown in figure 3.5, a scattered plot was made to show and verify the correlation between patient height and weight. It can be inferred that the higher the height, the higher the weight, with an exponential relationship. However, as seen in the graph, it can be seen that some height value is not

proportional to the weight. With the existence of this kind of data points, different types of data point can be categorized based on a measurement which involves both height and weight, which is body mass index (BMI). For later usage, the BMI category can also be a parameter to build the model.

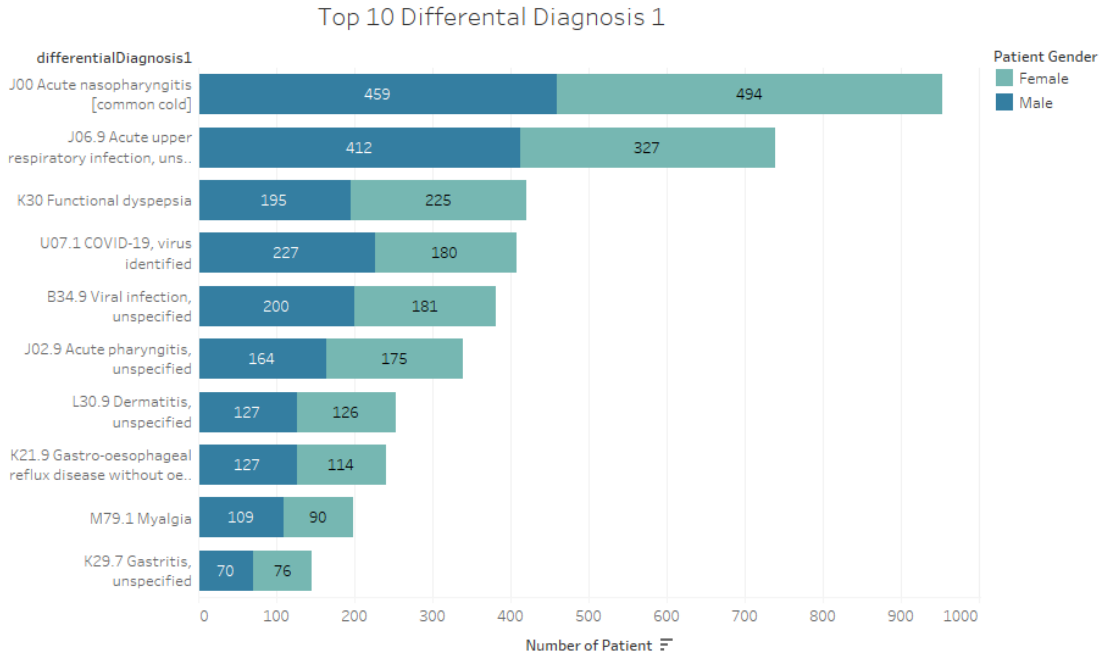

Figure 3.6 Top 10 Differential Diagnosis 1

As mentioned in the data gathering, doctors can choose up to 5 differential diagnoses based on the diagnosis of the patient following the diseases listed in the ICD-10. Seen in figure 3.6, Acute nasopharyngitis or common cold was at the first place of the overall patient diagnosis. Correlations of the gender are also described by the graph where the balance of male and female for each differential diagnosis in the top 10 is quite stable and equal (the gender for each diagnosis), with an average of 18.4 difference in count comparing both genders.

3.2.2 EDA of Medicine Data

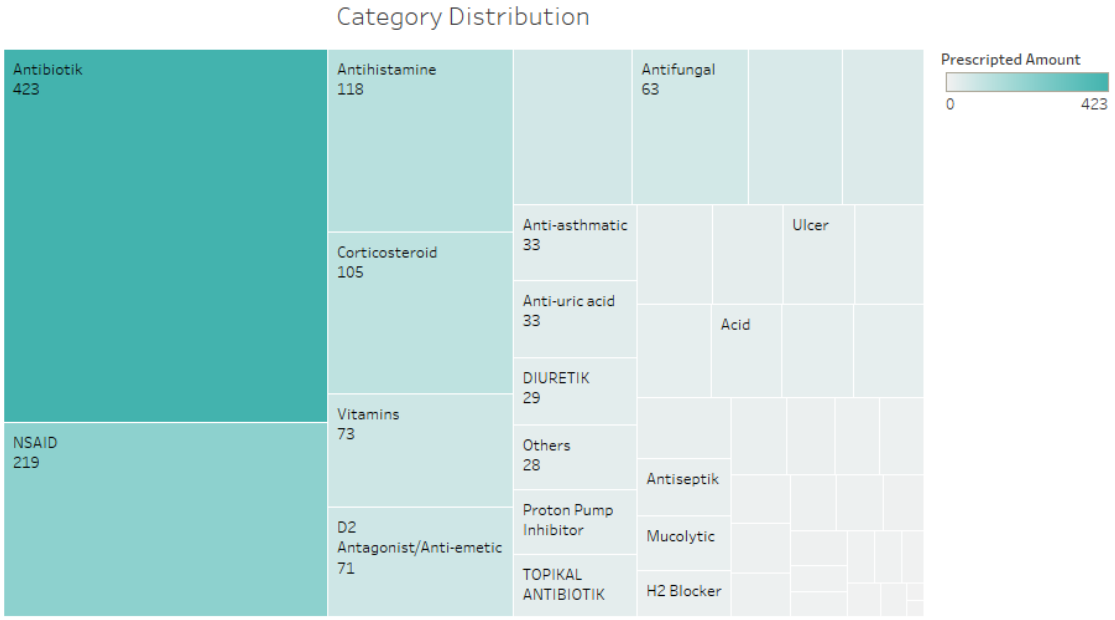

Figure 3.7 Medicine Category Distribution

Based on findings shown in figure 3.7, antibiotics was the medicine category with the most count on the medicines type being prescribed, which is 423 medicines. This analysis is based on the medicine categories and medicines data

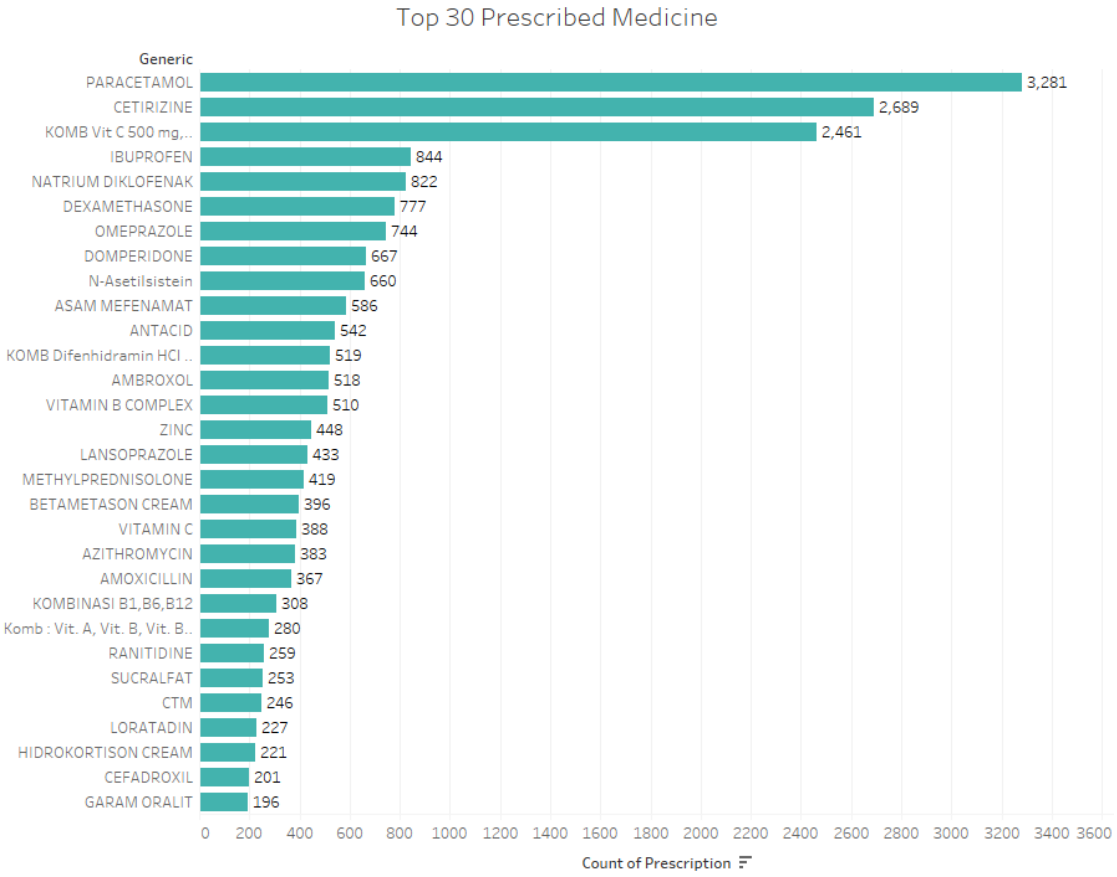

Figure 3.8 Top 30 Prescribed Medicine

Top prescribed medicines seen in figure 3.8 are aligned with the patients differential diagnosis seen in figure 3.6. For instance, first place in the top prescribed medicine is paracetamol which can be used to relieve cold-related symptoms such as headache, joint pain, and earache (Picon, 2013). Comparing analysis obtained from figure 3.7 and 3.8, with the antibiotics category taking the first place, seeing the top 30 medicines there are 4 medicines that are categorized as antibiotics they are: cetirizine, azithromycin, amoxicillin, cefadroxil.

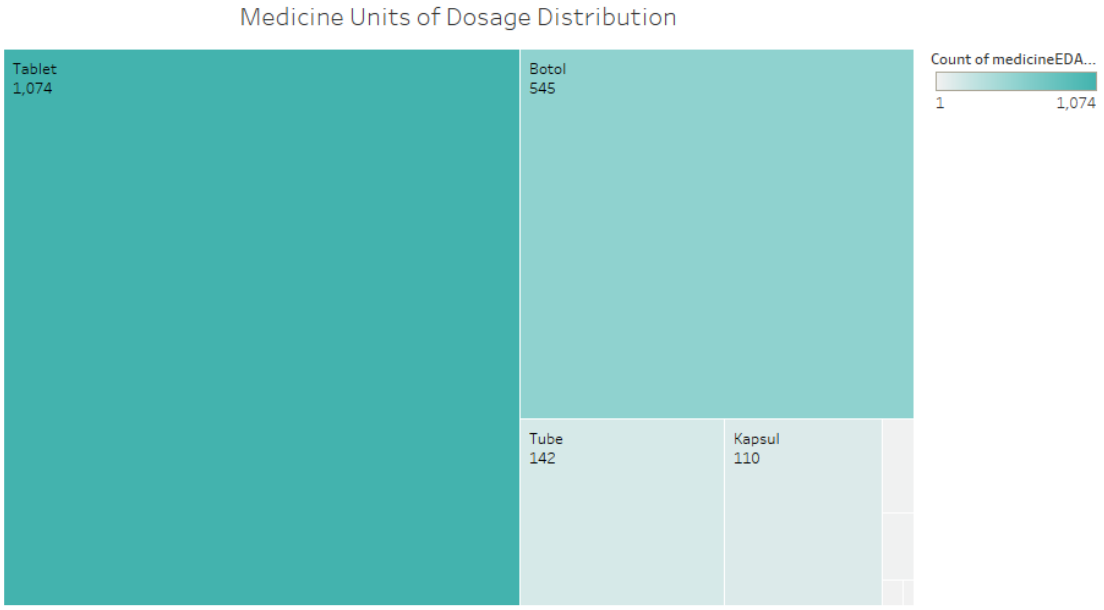

Figure 3.9 Medicine Unit of Dosage Distribution

Correlating figures 3.8 and 3.9, it is found out that prescribed medicines are mostly in tablet form.

3.3 Data Preprocessing and Splitting

When datasets have been explored, numerous insights were obtained in terms of processing the data. For instance, patients' age need to be binned into categories, which are children, adolescence, adult, and senior adult. Patients' weight and height was also used to calculate their BMI and categorize them into 3 different classes: underweight, normal, and overweight. Here, the BMI classification doesn't include obesity class 1, class 2, and class 3 to reduce complexity in computation. Categorizing patients this way makes it more convenient later on when using this information as a parameter for the AI model.

Table 3.3 Age Range and its Life Stage Category

Age Range	Life Stage

0 - 12	Children
13 - 18	Adolescence
19 - 59	Adult
≥ 60	Senior Adult

Table 3.4 BMI Range and its Category

BMI Range	**BMI Category**
< 18.5	Underweight
18.5 - 24.9	Normal
≥ 25	Overweight

Top 50 ICD-10 data that was previously mapped was also planned to be used as substitution recommendations due to collaborative filtering cold start characteristic to rare diseases. Another task done was handling missing value. This is done by firstly separating columns that have object and numeric data type. If the columns have object data type, missing values will be filled with the mode value. On the other hand, if the column has numeric data type, then skewness of the data distribution was checked using equation 3.1 (can also use skew() function from pandas library). If the skewness is close or equal to 0 (meaning normal distribution), then missing values were filled with mean value. However, when the skewness is greater or less than 0 (meaning positively/negatively skewed), missing values were filled with median value.

$$\bar{\mu}_3 = \frac{\sum_i^N (X_i - \bar{X})^3}{(N-1)\times\sigma^3}$$

Equation 3.1 Skewness

With μ3 is the skewness, N as the number of variables in the distribution, Xi as random variables, X head as mean of the distribution, and lastly as the standard deviation.

After data was preprocessed, prescription data was splitted into train and test data with the division of 80% train set and 20% test set. This set will be used for testing the functionality of the model developed. However, it won't be used for model performance scoring because k-fold cross validation will be used for this task.

3.4 AI Model Development and Evaluation

In the model development stage, 2 machine learning models were made using different algorithms. First one uses the collaborative filtering algorithm and second one uses the k-nearest neighbor algorithm. Both algorithms then were tested using the previously splitted data.

3.4.1 Model with Collaborative Filtering

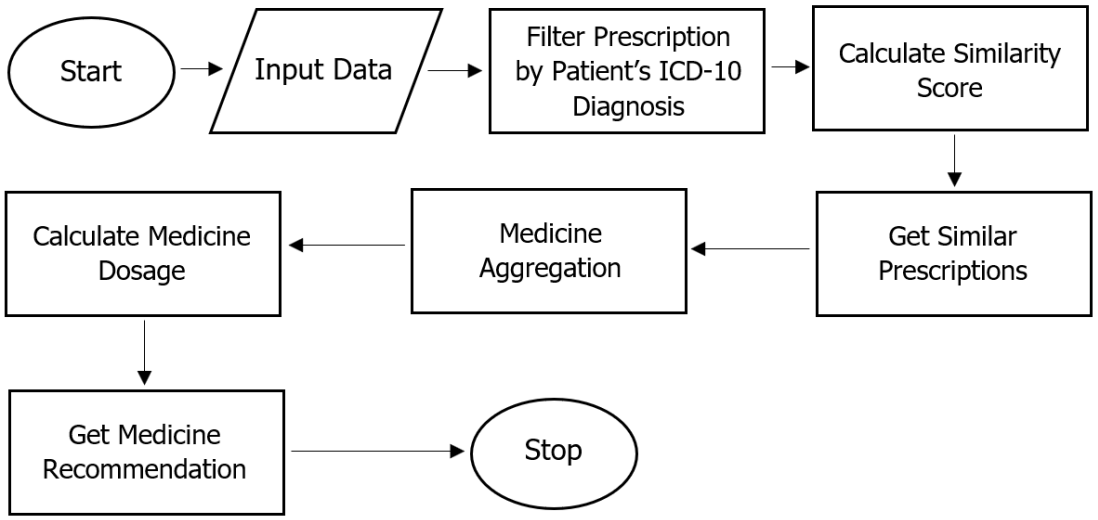

Figure 3.10 Collaborative Filtering Flowchart

The model follows the workflow as seen in the collaborative filtering flowchart figure 3.10. The input data consist of prescription (train and test set), patient, medicine, medicine prescription, and pivot prescription report (train set) data. As explained in the literature review subchapter 2.2, collaborative filtering collaborates with the user, which in this case by patients' historical records. The parameter that leads into a certain patient choosing or being chosen (in this case being chosen by doctor) a certain item (in this case medicine and dosage) can be used to find similarity score; hence 4 parameters were chosen from the patient consultation record following the scoring system shown in table 3.5.

Table 3.5 Collaborative Filtering Parameter Scoring

Parameter	**Range**	**Description**	**Score**
ICD-10 Diagnosis	-	If all of the ICD-10 diagnoses match with the other patient, it will give a full score. If it only matches partially, ex. 2 out 5; hence it will be ⅖.	4
Age Category	Difference: 1 - 2 [-0.25 Score] Difference: 2 - 4 [-0.50 Score] and so on [+ -0.25 Score]	For a different of more than one age category to the active patient, 0.25 will be subtracted from 1 (=100% similarity) for each additional age bin	1
BMI Category	-	If the BMI category matched, score is 1, else 0.	4
Gender	-	If gender matched score is 1, else 0.	1
Total Score			10

Comparing back to figure 3.10, test prescriptions will be scored based on their differential diagnosis (in a form of ICD-10). It is then filtered with a threshold score of 2; hence only prescriptions with score equal and greater than 2 were passed. Next, the similarity score based on other parameters was calculated and totaled. The most similar prescription was then obtained by threshold which is setted by the average of the cluster score. Recommended medicine then will be retrieved from the medicine data with the expected outcome sorted in python dictionary format based on their prescribed count (as seen in figure 3.11.


```
{
    "SURBEX T (KOMB Vit C 500 mg, Niasinamida 100 mg, Kalsium Pantotenat
20 mg,Vit. B1 15 mg, Vit. B2 10 mg,Vit. B6 5 mg, Vit. B12 4 mg) TAB": 124,
    "CETIRIZINE 10 MG Tablet": 78,
    "PARACETAMOL 500 MG Tablet": 77,
    "AMBROXOL 30 MG Tablet": 48,
    "Acetylcystein 200mg Tablet": 33,
    "SANADRYL DMP (Dekstrometorfan HBr 10 mg, Difenhidramin HCl 12,5 mg,
Ammonium Cl 100 mg, Na Sitrat 50 mg, Mentol 1 mg/ 5ML) SYRUP, BOTOL
120ML": 23,
    "DEXAMETHASONE 0.5 MG Tablet": 20,
    "PARACETAMOL 600 MG Tablet": 19,
    "CETIRIZINE 5 MG/5 ML Botol": 18,
    "ELKANA Syrup Botol 60ML": 18
}
```


Figure 3.11 Medicine List Output Sample

Medicine dosage can now be retrieved from medicine dose data by calculating and taking the mode value. The medicine brand needs to be compared first to be able to retrieve the right dosage. Dosage obtain consist of 'frequency', the number of medicine need to be consume, 'frequencyDd', which inform per how many day the number of medicine stated in frequency need to be consumed, 'timing', when the medicine should be consume, 'duration', the

duration of the medicine need to be consume (day), and 'amount', the amount of medicine need to be consume in total. This can be seen more clearly with the example shown in figure 3.12.


```
"SURBEX T (KOMB Vit C 500 mg, Niasinamida 100 mg, Kalsium Pantotenat 20
 mg,Vit. B1 15 mg, Vit. B2 10 mg,Vit. B6 5 mg, Vit. B12 4 mg) TAB": {
    "frequency": 1.0,
    "frequencyDd": 1.0,
    "timing": "Setelah Makan",
    "duration": 5.0,
    "amount": 5.0
}
```


Figure 3.12 Medicine Dosage Output Sample

3.4.2 Model with K-Nearest Neighbor

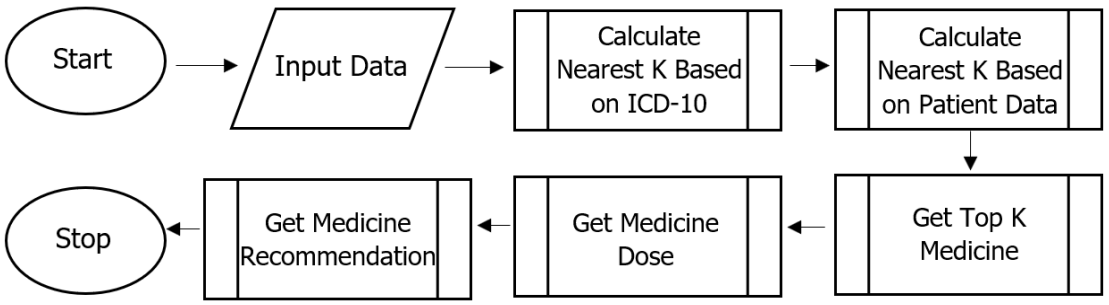

Figure 3.13 KNN Algorithm Flowchart

The workflow of the model using KNN algorithm follows the flowchart shown in figure 3.13. Input data was taken from prescription (train and test set), pivot prescription report (train set), patients, medicine, medicine prescription data. Two step similarity was done in the algorithm. first one, find a similar prescription based on the patient's differential diagnosis (in ICD-10 form) by finding the k nearest neighbors. The k being used was 94 and this is obtained from taking the square root of the size of the sample dataset. Then those similar neighbors (similar prescription) were filtered again by finding k most

nearest neighbor (similar) based on patients data by taking patient's BMI category and age life stage into consideration. However, this time the k being used was 40, which was obtained by taking the square root of the patients data size but divided by 2. This was done due to the run time concern, where the more k being used, the longer the runtime will be. Obtained prescriptions then were in a sorted descending order.

With the similar prescriptions on hand, medicines can now be retrieved from the medicines data. Retrieved medicines were sorted and 10 medicines were taken. The medicine dosage was obtained by calculation of the medicines mode from medicine prescription data. However, the result was saved in a form of dictionary and model only needed to retrieve the dosage needed; hence able to reduce runtime. (show output also)

3.4.3 Model Evaluation

To know the best model from both algorithms, scoring and evaluation needs to be done. To obtain a more generalized scoring with less unseen data, k-fold cross validation technique was used for this task. Considering the size of the dataset, the number of fold (k) used was 10. For each fold, the mean value for all metrics scores was calculated and used as the representation of the whole fold.

Scoring metrics that were chosen are recall@k, precision@k, f1 score, and also the runtime as consideration. Recall@k represents the proportion of relevant medicines found in the recommendation medicines set (top k set; k=10). On the other hand, precision@k represents the proportion of recommended medicines (in the top k set; k=10) that are relevant. Relevant medicines are medicines that are already known in the dataset while

recommended medicines are the medicines that are generated by the recommendation system (Malaeb, 2017). In addition, a combination of both recall@k and precision@k, f1 score was calculated as representation of both metrics. Whereas, runtime was calculated to make sure that the recommendation process will not take too long for each input data.

Table 3.6 Collaborative Filtering Score Results

Input	
Parameter	**Value**
Threshold	2
Score Results	
Metric	**Value**
Recall@k	77.04
Precision@k	24.19
F1 Score	36.67
Runtime	2.14 s/item

Scores shown in table 3.6 are shown in 2 decimal places (maximum value is 100). Listed in table 3.6, results of collaborative filtering models are more weighted on the recall@k score compared with the precision@k. This is something normal considering that a model can't have both metrics high near to 100 but balance is possible when metrics scores are near. However, in this case the difference between recall@k and precision@k are quite far. Looking back at both definitions, it is more important to improve recall@k compared to precision@k. This is because the model built was a recommendation system which gives more than 1 recommended item; thus can pick another medicine choice from the set if one is not relevant. Furthemore, it is an optional choice

for the doctor to use the recommendation system, meaning that doctor can still make a custom-made prescription for the patient.

Focussing more on the recall@k score, it is considerably high because it is already higher than 70. However, the runtime per item was still noticeably too long. This means further refinement needs to be done to reduce the runtime for at least in the range of 1 second/item.

Table 3.7 KNN Score Results

Input	
Parameter	**Value**
K ICD-10	94
K Patients	40
Score Results	
Metric	**Value**
Recall@k	63.51
Precision@k	15.11
F1 Score	23.69
Runtime	1.06 s/it

Different from collaborative score results, KNN as seen in table 3.7 have lower recall@k which is at 63.51. However, the runtime is approximately 2 times faster. This can be inferred that collaborative filtering is good with recall@k, whereas KNN is good with the runtime. With the result in mind, refinement can be done to cut runtime for collaborative filtering and improve recall@k for KNN.

3.5 Model Refinement and Selection

As what stated in the previous sub chapter, both models need to be refined accordingly. Refinements were done by tuning the parameter value for each model to achieve the best score result as needed.

Table 3.8 Collaborative Filtering Refinement Scenarios

Attempt	**Threshold**	**Scoring**			
		Recall@k	**Precision**@k	**F1 Score**	**Runtime**
Original	2	77.04	24.19	36.67	2.14 s/it
Trial 1	1	72.22	22.34	34.02	3.14 s/it
Trial 2	4	42.96	17.46	23.54	1.61 s/it
Trial 3	3	44.44	18.02	24.33	1.37 s/it
Trial 4	2.5	65.55	23.40	33.37	1.57 s/it
Trial 5	2.3	66.30	23.69	33.78	1.52 s/it
Trial 6	2.1	66.30	23.69	33.78	1.81 s/it
Trial 7	2.05	66.30	23.69	33.78	1.66 s/it
Trial 8	2.01	66.30	23.69	33.78	1.48 s/it

As seen in table 3.8, 8 scenarios were tested to obtain the best fit score for the model that uses collaborative filtering. Trial 1 and trial 2 were meant to scale the difference when the threshold increased and decreased. Logically, recall@k will increase when more data can passed, this mean threshold needs to be decreased.

While runtime will become faster when less data is passed, meaning threshold needs to be increased. With both of these rules in mind, scaling was done accordingly and it is found out that decreasing threshold value to 1 makes the recall@k decrease (72.22) although it doesn't decrease as much as when threshold is increased to 3 and 4. This means that 2 is already the best value for threshold in terms of recall@k. As a result, the objective now is to decrease runtime, which is described by trial 4 to trial 8. Threshold is increased slightly not far from 2; hence the best value for both recall@k and runtime is at trial 8 which is 2.01 as the threshold. However, noticeably, changes of score value seems quite far when threshold were only added 0.01 from the original threshold (2). This means that the model has high variance that it might be a model with less generalization when used for a different dataset.

Table 3.9 KNN Refinement Scenarios

Attempt	**K ICD**	**K Patient**	**Scoring**			
			Recall@k	**Precision**@k	**F1 Score**	**Runtime**
Original	94	40	63.51	15.11	23.69	1.06 s/it
Trial 1	100	50	63.62	15.15	23.76	1.02 s/it
Trial 2	99	49	63.14	15.04	23.59	1.03 s/it
Trial 3	111	51	63.62	15.14	23.75	1.04 s/it
Trial 4	91	49	63.33	15.10	23.67	1.06 s/it
Trial 5	131	85	62.64	14.98	23.47	1.06 s/it
Trial 6	151	37	63.33	15.15	23.74	1.13 s/it

Trial 7	141	50	63.86	15.21	23.85	1.05 s/it
Trial 8	151	51	63.91	15.20	23.84	1.07 s/it
Trial 9	130	71	62.71	14.97	23.46	1.04 s/it
Trial 10	167	46	63.72	15.14	23.75	1.23 s/it
Trial 11	155	53	63.56	15.11	23.70	1.04 s/it
Trial 12	145	51	63.88	15.19	23.83	1.06 s/it
Trial 13	153	49	63.59	15.16	23.77	1.09 s/it
Trial 14	152	52	63.88	15.18	23.81	1.15 s/it
Trial 15	149	49	63.61	15.17	23.78	1.06 s/it

In contrast KNN model refinement has small changes to the score when parameters (K ICD-10 and K patients) are turned nearly to the threshold. This means that the KNN model has a low variance, meaning that it is good when applied to a different dataset. This is known by doing scaling on the first 4 trials, which are only tuned near the original values. Then next until the end trial, high changes are done to both parameters, to k ICD-10 only, and to k-patients only. This way the effect of each parameter to the scoring metrics can be known. Based on the result shown in table 3.9, small changes in k patients further than around 50 affect the recall@k to be decreased. This means that we can keep increasing the k for ICD-10. However, when it is too high runtime will be affected into becoming too long; hence the best value for both parameters is trial 8 with the maximum recall@k obtained 63.91 with a considerably small difference on the runtime compared with the original.

Table 3.10 KNN Refinement Scenarios with Threshold

Attempt	Threshold		K ICD	K Patient	Scoring			
	Similarity	Medicine			Recall @k	Precision @k	F1 Score	Runtime
Original	0.5	0.1	94	40	60.19	14.29	22.41	1.17 s/it
Trial 1	0.5	0.1	100	50	59.90	14.22	22.31	1.43 s/it
Trial 2	0.5	0.1	65	25	59.09	14.11	22.12	1.16 s/it
Trial 3	0.08	0.5	85	65	59.50	14.13	22.16	1.65 s/it
Trial 4	0.08	0.5	150	40	60.32	14.31	22.45	1.32 s/it
Trial 5	0.04	0.4	150	40	62.20	14.73	23.12	1.34 s/it
Trial 6	0.05	0.4	130	50	62.33	14.79	23.20	1.75 s/it
Trial 7	0.035	0.4	95	65	62.03	14.744	23.12	1.62 s/it
Trial 8	0.04	0.35	93	59	62.08	14.71	23.09	1.50 s/it

In comparison with the best scenario obtained from collaborative filtering, the best scenario of KNN algorithm seems to still need improvements in its recall@k. For this reason, modifications were made by adding a threshold based on ICD-10 similarity score (results of KNN for ICD-10) and threshold for medicine retrieval. Threshold for KNN similarity score was setted to determine if the score is above threshold then medicine retrieval will be done normally. However if the score is below threshold, a default set of medicines were retrieved instead. These default medicines are also being setted by threshold. The medicine retrieval process is different however. Medicine was retrieved from the drug map data (Top 50 ICD-10

and its medicines data) and 1 medicine representation was taken from each ICD-10. Representative medicine was based on a scoring, which is by calculating the prescribed count of the highest medicine in the ICD-10 group, divided by the total prescribed count of all medicines in the ICD-10 group, which can be seen more clearly in figure 3.14. This way medicine taken will not be subjective only for each ICD-10 they are in. This score was then filtered by threshold and 10 best medicines were taken. These medicines were saved into a variable and can be retrieved anytime to save processing power.

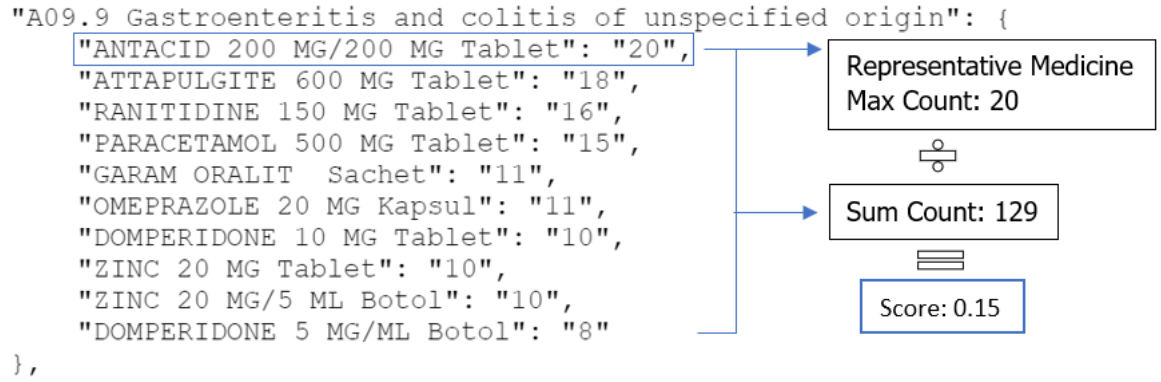

Figure 3.14 Drug Map Scoring Method for Each ICD-10

By doing this, it is hoped that the recall@k score can increase while it is still acceptable to increase run time because KNN overall runtimes are still far below collaborative filtering. However, as seen in table 3.10 adding thresholds does not produce higher recall@k compared to without threshold. This is because adding threshold means adding complexity hence affecting the score also. As mentioned before, runtimes are also affected to be increased because of more steps added to the overall process. With this the best result obtained overall is the model obtained from using collaborative filtering with trial 8 scenario.

CHAPTER 4

RESULT AND DEPLOYMENT

4.1 Model Deployment

The chosen model, which is the collaborative filtering trial 8 model, was then deployed into a simple web application using streamlit, an open-source python library used for deploying data science projects into a web app. To run the web application privately, it was run in localhost:8501. All of the python code functions made for the model were wrapped into a python module named "collaborative_filtering". This way, the web app can be coded in a different file named "app.py" and the module will only need to be imported to the web app file.

The web application works by asking input information regarding the patient, which consists of the patient's id, gender, age, height (cm), and weight (kg) as seen in figure 4.1. Each information will then be saved inside a variable which is processed first before it can be used. Patient's age information will be categorized based on life stage seen in table 3.3 and patient's height and weight will be used to calculate BMI and categorized based on the BMI category seen in table 3.4.

Aside from the patient's information, the patient's differential diagnoses were also asked. Users can input up to 5 differential diagnoses and the NaN option is selected when the slot is not being used as seen in figure 4.2. Each differential diagnosis is saved in a separate variable (including if the slot is filled with NaN). Both patient's information and differential diagnoses are then put into a pandas series. However, for the differential diagnoses, they are conditioned if the string value saved inside the variable is "NaN", then it won't be inserted into the series.

When the "Give Recommendation" button is pressed, the series made from the input information will show up, as shown in figure 4.3. Then the main predictor function from the collaborative filtering module is then activated and all the program will run and return the recommended medicines list and medicines dosage as seen in figure 4.3.

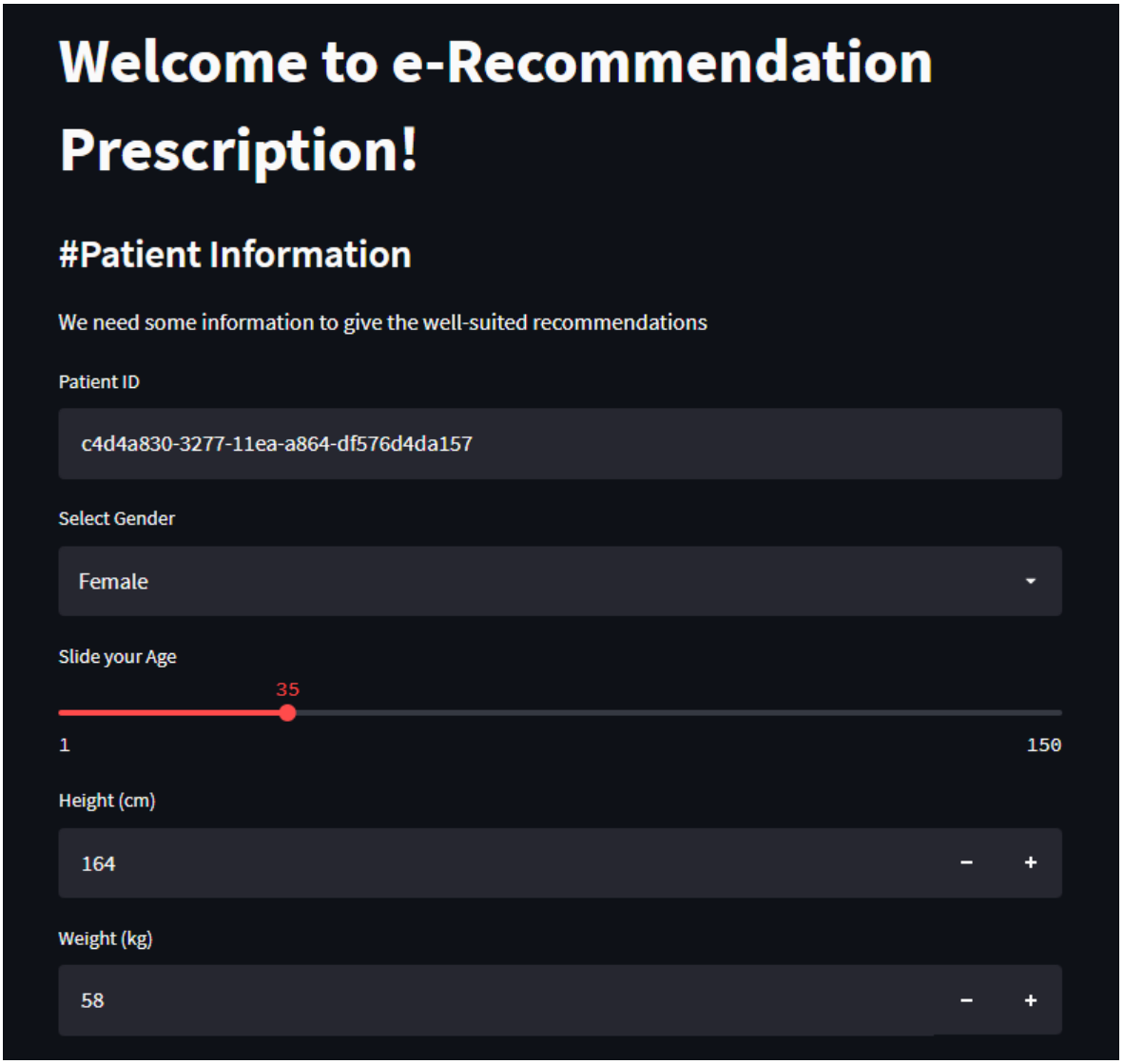

Figure 4.1 Input for Patient Information

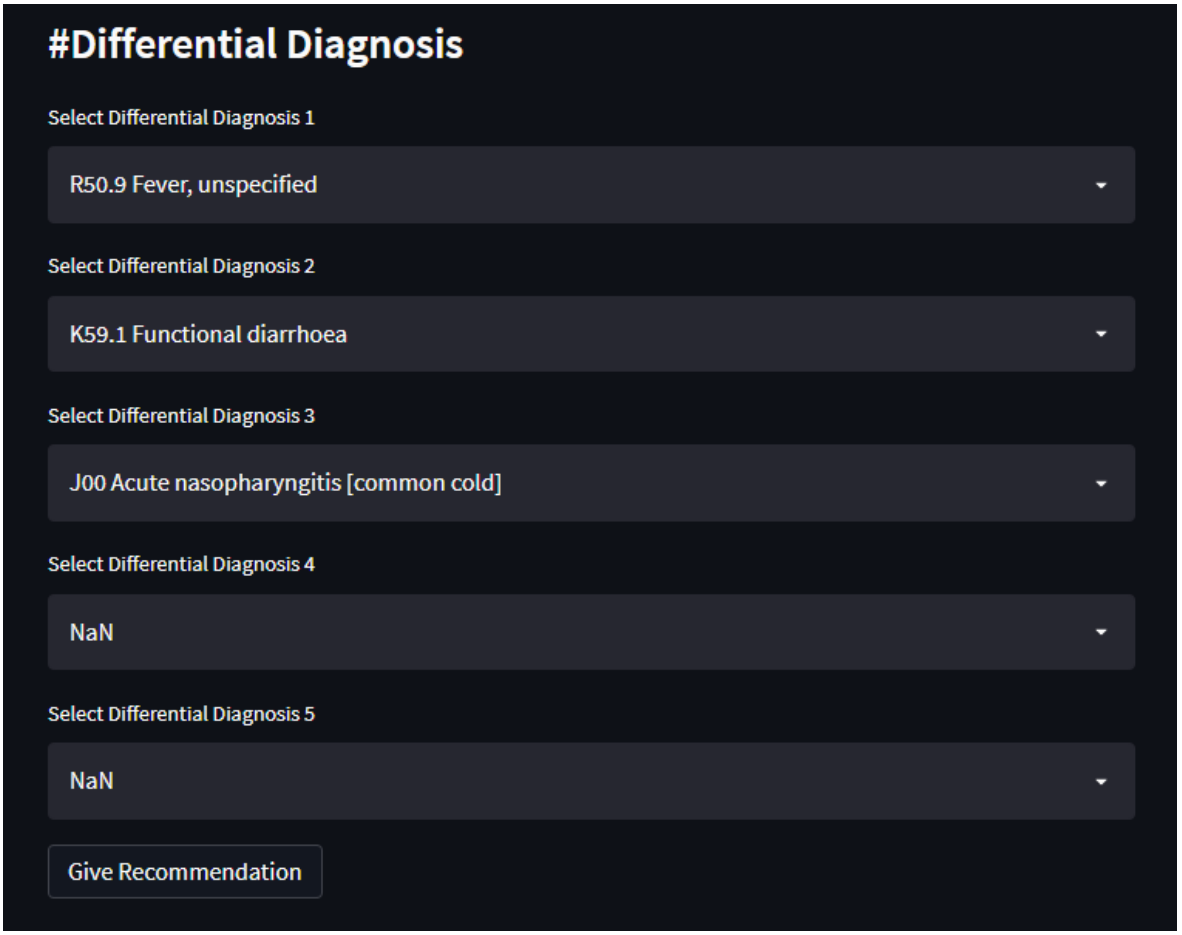

Figure 4.2 Input for Patient Differential Diagnosis

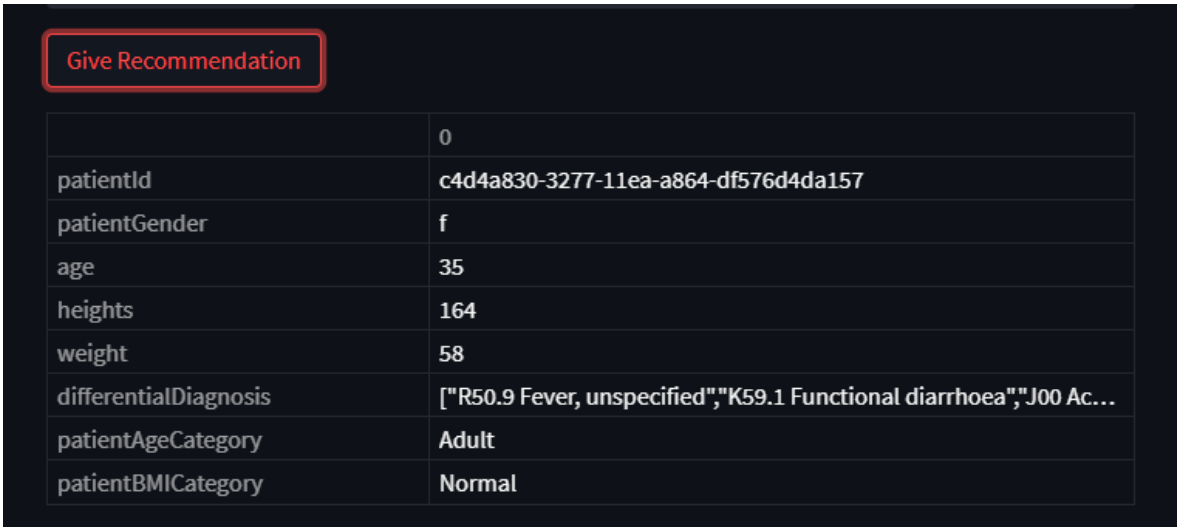

Figure 4.3 Input Information Saved in Series

Seeing the recommended medicine results in figure 4.4 and comparing them back with input differential diagnoses ICD-10 in figure 4.2, the medicines are relevant with the input. The fever and common cold diagnosis gave paracetamol with different dosage recommendations, which according to the UK National Health Service (NHS) it is used to relieve high temperature for fever and relieve common cold symptoms. Second diagnosis is diarrhoea, with the relevant medicine being recommended are new diatab (molagit), domperidone, and loperamide, which according to UK Monthly Index of Medical Specialities (MIMS), a pharmaceutical prescribing reference, are examples of medicine used for diarrhoea and gastric diseases. On the other hand, surbex T based on UK MIMS is a food supplement used for maintaining the body's immune system. Lastly, cetirizine is a medicine for allergy according to UK MIMS and this recommendation can still be used if the fever or cold or diarrhoea is caused by an allergic reaction, which only the doctor being consulted know; hence the recommendation system for this sample test are able to give medicine relevant to the input data.


```
{
  "PARACETAMOL 500 MG Tablet" : 4
  "CETIRIZINE 5 MG/5 ML Botol" : 3
  "PARACETAMOL 100 MG/ML Botol" : 3
  "
  SURBEX T (KOMB Vit C 500 mg, Niasinamida 100 mg, Kalsium Pantotenat 20
  mg,Vit. B1 15 mg, Vit. B2 10 mg,Vit. B6 5 mg, Vit. B12 4 mg) TAB
  "
   : 2
  "PARACETAMOL 600 MG Tablet" : 2
  "PARACETAMOL 120 MG/5 ML Botol" : 1
  "NEW DIATAB TABLET (MOLAGIT)" : 1
  "DOMPERIDONE 10 MG Tablet" : 1
  "CETIRIZINE 10 MG Tablet" : 1
  "LOPERAMIDE 2 MG Tablet" : 1
}
```


Figure 4.4 Medicine List Output

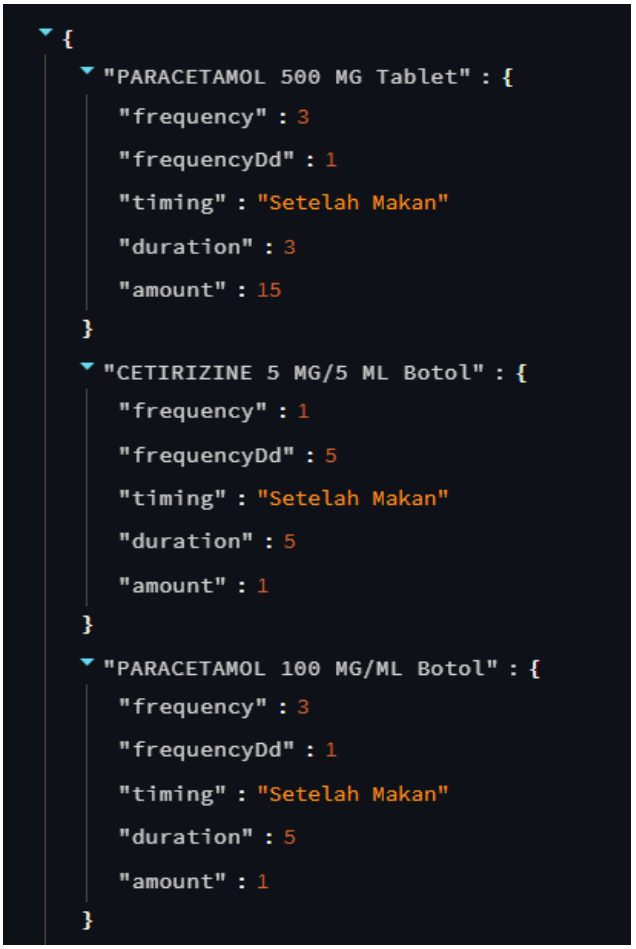

Figure 4.5 Medicine Dosage Output

CHAPTER 5

CONCLUSION

E-prescription recommendation is a feature in Prixa.ai teleconsultation service, which gives recommendation of medicines and its dosage to doctors. This feature carries the goal to assist and provide doctors with a drug recommendation system for the consulting patient. The recommendation system will give output based on a model made from patients' medical records with the diagnosis based on International Classification of Diseases 10th Revision (ICD-10).

The project development started with data gathering and cleaning. This data then was explored and visualized to gain insight about each data and correlation of different features, with 8 insights were obtained. Data then was preprocessed with patient age being binned into life stages and patient height and weight into BMI categories. Actions like dealing with missing value, cleaning, and drug mapping were also done. Processed data was reduced to 80% of the original size and splitted into 80% train set and 20% test set.

Two AI models were developed based on the collaborative filtering and k-nearest neighbor (KNN) algorithm with them being evaluated afterwards using 10-folds cross validation. Result shows that collaborative filtering's model needs faster runtime and KNN's model needs higher recall. With this in mind, the model was refined and selected based on the parameter tuning result scenarios and collaborative filtering model trial 8 was chosen. Lastly, the obtained model was wrapped into a python module and deployed into a web application made using streamlit. Patient information and differential diagnosis were needed as the input and the output will give the list of recommended medicines and their dosage.

REFERENCES

[1] Berrar, D. (2019). Cross-Validation. Encyclopedia of Bioinformatics and Computational Biology, pp.542–545.

[2] Encyclopedia Britannica. (n.d.). International Classification of Diseases. [online] Available at: https://www.britannica.com/topic/International-Classification-of-Diseases#:~:text=International%20Classification%20of%20Diseases%20(ICD.

[3] Darapureddy, N., Karatapu, N. and Battula, T.K. Research of Machine Learning algorithms using K-fold cross validation. (2019). International Journal of Engineering and Advanced Technology, 8(6S), pp.215–218.

[4] Hussain, A. (2020). K-Nearest Neighbors (KNN) and its Applications. [online] Medium. Available at: https://medium.com/@arman_hussain786/k-nearest-neighbors-knn-and-its-applications-7891a4a916c6.

[5] Jayaswal, V. (2020). Performance Metrics: Confusion matrix, Precision, Recall, and F1 Score. [online] Medium. Available at: https://towardsdatascience.com/performance-metrics-confusion-matrix-precision-recall-and-f1-score-a8fe076a2262.

[6] Kumar, V and Reddy, R. (2014). A Survey on Recommender Systems (RSS) and Its Applications. International Journal of Innovative Research in Computer and Communication Engineering, 2(8).

[7] Luo, S. (2018). Intro to Recommender System: Collaborative Filtering. [online] Towards Data Science. Available at: https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26.

[8] Maher Malaeb (2017). Recall and Precision at k for Recommender Systems. [online] Medium. Available at: https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54.

[9] Montero, M. (2021). Resampling Methods for Machine Learning modeling. [online] Geek Culture. Available at: https://medium.com/geekculture/resampling-methods-for-machine-learning-modeling-d2cdc1d3640f.

[10] NHS (2019). Paracetamol For Adults. [online] NHS. Available at: https://www.nhs.uk/medicines/paracetamol-for-adults/.

[11] Picon, P.D., Costa, M.B., da Veiga Picon, R., Fendt, L.C.C., Suksteris, M.L., Saccilotto, I.C., Dornelles, A.D. and Schmidt, L.F.C. (2013). Symptomatic treatment of the common cold with a fixed-dose combination of paracetamol, chlorphenamine and phenylephrine: a randomized, placebo-controlled trial. BMC infectious diseases, [online] 13, p.556. Available at: https://pubmed.ncbi.nlm.nih.gov/24261438/.

[12] Pramoditha, R. (2020). k-fold cross-validation explained in plain English. [online] Medium. Available at:

https://towardsdatascience.com/k-fold-cross-validation-explained-in-plain-english-659e33c0bc0.

[13] Sanjay.M (2018). Why and how to Cross Validate a Model? [online] Medium. Available at: https://towardsdatascience.com/why-and-how-to-cross-validate-a-model-d6424b45261f.

APPENDICES

A. Collaborative Filtering Code Functions

```
def similarityICD10(newPresc, allPresc):
  newData = json.loads(newPresc['differentialDiagnosis'])
  allData = json.loads(allPresc['differentialDiagnosis'])
  count = 0
  for i in newData:
    if i in allData:
      count += 1
  similarity = ((count)/len(newData)) * 4
  return similarity
```

Figure A.1 Function to Calculate Similarity ICD-10 Score

```
def filterICD10(newPresc, allPresc, threshold=2.05):
  dictTemp = dict()
  for idx, record in allPresc.iterrows():
    sim = similarityICD10(newPresc, record)
    dictTemp[record['id']] = {'patientId': record['patientId'],'similarity': sim}
  filtered = dict((k, v) for k, v in dictTemp.items() if v['similarity'] >= threshold)
  prescId = [*filtered]
  #print(prescId)
  allPrescFiltered = allPresc[allPresc['id'].isin(prescId)]
  return filtered, allPrescFiltered
```

Figure A.2 Function to Filter Prescription Input Based on ICD-10

```
def findSimilarityScore(newPresc, allPresc, icdSim):
  totalScore = icdSim.get(allPresc['id'], {}).get('similarity')

  # age score
  ageCategory = ["Child", "Adolescence", "Adult", "Senior Adult"]
  age1 = patients.loc[patients['id'] == newPresc['patientId'],'patientAgeCategory'].iloc[0]
  age2 = patients.loc[patients['id'] == allPresc['patientId'],'patientAgeCategory'].iloc[0]

  # print(age1)
  index1 = ageCategory.index(age1)
  index2 = ageCategory.index(age2)
  if index1 != index2:
    totalScore += 1 - ((abs(index1-index2))/4)
  else:
    totalScore += 1

  # bmi score
  bmi1 = patients.loc[patients['id'] == newPresc['patientId'],'patientBMICategory'].iloc[0]
  bmi2 = patients.loc[patients['id'] == allPresc['patientId'],'patientBMICategory'].iloc[0]
  if bmi1 == bmi2:
    totalScore += 1
    totalScore *= 4
  else:
    totalScore += 0

  # gender score
  if newPresc["patientGender"] == allPresc["patientGender"]:
    totalScore += 1
  else:
    totalScore += 0
  return totalScore
```

Figure A.3 Function to Calculate Total Similarity Score

```
def getSimilarPrescriptions(newPresc, allPresc, icdSim):
  dictTemp = dict()
  for idx, record in allPresc.iterrows():
    #print("pertama:",record['id'])
    sim = findSimilarityScore(newPresc, record, icdSim)
    dictTemp[record['id']] = sim
    #print("kedua: ", record['id'])
  mostSimilar = dict(sorted(dictTemp.items(), key=itemgetter(1), reverse=True))
  #print(len(mostSimilar))
  return mostSimilar
```

Figure A.4 Function to Get Similar Prescriptions

```
def getMedicine(similarPresc, allPivot):
  prescId = similarPresc.keys()
  medList = dict()
  prescResult = allPivot[allPivot["id"].isin(prescId)]
  for idx, row in prescResult.iterrows():
      tempString = 'medicineBrand'
      for i in range(5):
          stringCounter = str(i+1)
          meds = tempString.join(stringCounter)
          if (not pd.isnull(row[meds])):
              if(row[meds] not in medList):
                medList[row[meds]] = 1
              else:
                 medList[row[meds]] += 1
  return dict(sorted(medList.items(), key=itemgetter(1), reverse=True)[:10])
```

Figure A.5 Function to Get Recommended Medicine List

```
def getGTMedicine(testRow):
  meddList = dict()
  tempString = 'medicineBrand'
  for i in range(5):
      stringCounter = str(i + 1)
      meds = tempString + stringCounter
      if (not pd.isnull(testRow[meds].item())):
          if (testRow[meds].item() not in meddList or testRow[meds].item() is not np.nan):
              meddList[testRow[meds].item()] = 1
  return meddList
```

Figure A.6 Function to Get Medicine Ground Truth

```
def getMedicineDosage(medList):
  doseDict = dict()
  medError = ["OBH 200 IKAP sir Komb btl 200 ml", "Nystatin NOVE susp100.000 IU/ml btl 15 ml"]
  medBrand = [*medList]
  #print(medBrand)
  for i in medError:
    if i in medBrand: medBrand.remove(i)
  if len(medBrand)==0:
    pass
  else:
    for brand in medBrand:
      if brand in medicine_dose['brand']:
          doseDict[brand] = {'frequency': medicine_dose.loc[medicine_dose['brand'] == brand, 'frequency'].values[0],
                             'frequencyDd': medicine_dose.loc[medicine_dose['brand'] == brand, 'frequencyDd'].values[0],
                             'timing': medicine_dose.loc[medicine_dose['brand'] == brand, 'timing'].values[0],
                             'duration': medicine_dose.loc[medicine_dose['brand'] == brand, 'duration'].values[0],
                             'amount': medicine_dose.loc[medicine_dose['brand'] == brand, 'amount'].values[0]}
      else:
          pass
  return doseDict
```

Figure A.7 Function to Get Medicine Dosage

B. K-Nearest Neighbors Code Functions

```
def similarityICD10(testPresc,trainPresc):
  testPresc = json.loads(testPresc)
  trainPresc = json.loads(trainPresc)
  count = 0
  for i in range(4):
    if len(trainPresc) < 5:
      trainPresc.append('')
    if len(testPresc) < 5:
      testPresc.append('')
  for i, value in enumerate(testPresc):
    if value in trainPresc:
      count+= 1/((abs(trainPresc.index(value) - testPresc.index(value)))+1)
  similarity = count/len(testPresc)
  return similarity
```

Figure B.1 Function to Calculate Similarity ICD-10 Score

```
def similarityPatientData(testPatients, trainPatients):
  ageCategory = ['Child', 'Adolescence', 'Adult', 'Senior Adult']
  BMICategory = ['Underweight','Normal','Overweight']
  ageTest = ageCategory.index(testPatients['patientAgeCategory'].item())
  ageTrain = ageCategory.index(trainPatients['patientAgeCategory'].item())
  BMITest = BMICategory.index(testPatients['patientBMICategory'].item())
  BMITrain = BMICategory.index(trainPatients['patientBMICategory'].item())
  ageDistance = abs(ageTest - ageTrain)
  BMIDistance = abs(BMITest - BMITrain)
  finalDistance = ageDistance + BMIDistance
  return finalDistance
```

Figure B.2 Function to Get Distance Between Test and Train Patient Data

```
def filterICD10(testI,train, K):
  dictTemp = dict()
  for index, row in train.iterrows():
    sim = similarityICD10(testI['differentialDiagnosis'], row['differentialDiagnosis'])
    dictTemp[row['prescriptionNumber']] = {'Similarity': sim, 'id':row['patientId']}
  topK = dict(sorted(dictTemp.items(), key=lambda item: item[1]['Similarity'],reverse=True)[:K])
  #print(topK)
  return topK
```

Figure B.2 Function to Filter Prescription Based on ICD-10

```
def filterPatientData(testI, train, testPatients, trainPatients, K):
  #print(testPatients['id'])
  #print(testI['patientId'])
  testRow = testPatients.loc[testPatients['id'] == testI['patientId']]
  for i in train:
    trainRow = trainPatients.loc[trainPatients['id'] == train[i]['id']]
    train[i]['patientSimilarity'] = similarityPatientData(testRow,trainRow)
  sortedDict = dict(sorted(train.items(), key=lambda item:
                           (item[1]['Similarity'], item[1]['patientSimilarity']),reverse=True)[:K])
  return sortedDict
```

Figure B.3 Function to Filter Train Data Using Patients Data

```
def getMedicine(topK, allPivot):
  topPresc = topK.keys()
  allMedList = dict()
  prescResult = allPivot[allPivot['prescriptionNumber'].isin(topPresc)]
  for index, row in prescResult.iterrows():
    tempString = 'medicineBrand'
    for i in range(5):
        stringCounter = str(i+1)
        meds = tempString+stringCounter
        if (not pd.isnull(row[meds])):
            if(row[meds] not in allMedList):
                allMedList[row[meds]] = 1
            else:
                allMedList[row[meds]] +=1
  return dict(sorted(allMedList.items(), key=itemgetter(1), reverse=True)[:10])
```

Figure B.4 Function to Get Recommended Medicine List

```
def getGTMedicine(testRow):
  medList = dict()
  tempString = 'medicineBrand'
  for i in range(5):
      stringCounter = str(i + 1)
      meds = tempString + stringCounter
      if (not pd.isnull(testRow[meds].item())):
        if (testRow[meds].item() not in medList or testRow[meds].item() is not np.nan):
          medList[testRow[meds].item()] = 1
  return medList
```

Figure B.5 Function to Get Ground Truth Medicine

```
def getMedicineDosage(medList):
  doseDict = dict()
  medError = ["OBH 200 IKAP sir Komb btl 200 ml", "Nystatin NOVE susp100.000 IU/ml btl 15 ml"]
  medBrand = [*medList]
  #print(medBrand)
  for i in medError:
    if i in medBrand: medBrand.remove(i)
  if len(medBrand)==0:
    pass
  else:
    for brand in medBrand:
        doseDict[brand] = {'frequency': medicine_dose.loc[medicine_dose['brand'] == brand, 'frequency'].values[0],
                           'frequencyDd': medicine_dose.loc[medicine_dose['brand'] == brand, 'frequencyDd'].values[0],
                           'timing': medicine_dose.loc[medicine_dose['brand'] == brand, 'timing'].values[0],
                           'duration': medicine_dose.loc[medicine_dose['brand'] == brand, 'duration'].values[0],
                           'amount': medicine_dose.loc[medicine_dose['brand'] == brand, 'amount'].values[0]}
  return doseDict
```

Figure B.6 Function to Get Medicine Dosage

C. Metric Evaluation Code Functions

```
def reCall(prediction, groundTruth):
    count = 0
    for i in groundTruth:
        if i in prediction:
            count += 1
    if len(groundTruth) > 0 :
        recall = float(count / len(groundTruth))
        return recall
    else:
        return 0
```

Figure C.1 Function to Calculate Recall

```
def Precision(prediction, groundTruth):
  count = 0
  for i in groundTruth:
    if i in prediction:
        count += 1
  if len(prediction) > 0:
    precision = float(count / len(prediction))
    return precision
  else:
    return 0
```

Figure C.2 Function to Calculate Precision

```
def f1Score(recall, precision):
  if (float(precision + recall) > 0):
    f1score = float(float(2) * (float(precision * recall)) / (float(precision + recall)))
    return f1score
  else:
    return 0
```

Figure C.3 Function to Calculate F1 Score

```
def predict(newPresc, allPresc, allPivot):
  start = time.perf_counter()
  prescDict, similarICD10 = filterICD10(newPresc, allPresc)
  similarPrescription = getSimilarPrescriptions(newPresc, similarICD10, prescDict)
  medicineList = getMedicine(similarPrescription, allPivot)
  groundTruth = getGTMedicine(allPivot)
  medsWithDosage = getMedicineDosage(medicineList)
  recall = reCall(medicineList, groundTruth)
  precision = Precision(medicineList, groundTruth)
  f1score = f1Score(recall, precision)
  end = time.perf_counter()
  runtime = end - start
  return medsWithDosage, recall, precision, f1score, runtime
```

Figure C.4 Function to Run the Model

D. **Collaborative Filtering K-Fold Cross Validation (Trial 8)**

```
def splitKFold(data, k):
  dataCollection = dict()
  for i in range(1, k):
    test = data.iloc[(int(len(data)/k)*i):(int(len(data)/k)*(i+1))]
    train = data[~data['prescriptionNumber'].isin(test['prescriptionNumber'])]
    dataCollection[i] = dict()
    dataCollection[i]['Test'] = test
    dataCollection[i]['Train'] = train
  return dataCollection
```

Figure D.1 Function to Split Data Into K Fold(s)

```
def validate(data, k):
  allScoreSum = dict()
  for i in tqdm(data):
    print(f'Start of {(i)}-st/th Fold Validation')
    print('================================')
    trainSet = data[i]['Train']
    testSet = data[i]['Test']
    trainData = splitTrainTest(trainSet, patients, allPivot)
    testData = splitTrainTest(testSet, patients, allPivot)
    # print(testSet.info)
    scoreSum = dict()
    runtimeAll = 0
    generalRecallSum = 0
    generalPrecisionSum = 0
    generalf1ScoreSum = 0
    with tqdm(total=testData.shape[0]) as pbar:
      for index, rows in testData.iterrows():
          #print(rows)
          result, recall, precision, f1score, runtime = predict(rows, trainData, allPivot)
          firstICD10 = json.loads(rows['differentialDiagnosis'])[0]
          if str(firstICD10) not in scoreSum:
              scoreSum[firstICD10] = {'Recall': recall,
                                      'Precision': precision,
                                      'F1 Score': f1score}
              generalRecallSum += recall
              generalPrecisionSum += precision
              generalf1ScoreSum += f1score
              runtimeAll += runtime
              pbar.update(1)
          else:
              scoreSum[firstICD10] = {'Recall': scoreSum[firstICD10]['Recall'] + recall,
                                      'Precision': scoreSum[firstICD10]['Precision'] + precision,
                                      'F1 Score': scoreSum[firstICD10]['F1 Score'] + f1score}
              generalRecallSum += recall
              generalPrecisionSum += precision
              generalf1ScoreSum += f1score
              runtimeAll += runtime
              pbar.update(1)
    allScoreSum[i] = scoreSum
    avgRecall = generalRecallSum/len(data)
    avgPresicion = generalPrecisionSum/len(data)
    avgF1Score = generalf1ScoreSum/len(data)
    avgRuntime = runtimeAll/len(data)
  return allScoreSum, avgRecall, avgPresicion, avgF1Score, avgRuntime
```

Figure D.2 Function to Run Collaborative Filtering K-Fold Cross Validation

E. K-Nearest Neighbour K-Fold Cross Validation (Trial 8)

```python
def splitKFold(data, k):
  dataCollection = dict()
  for i in range(1, k):
    test = data.iloc[(int(len(data)/k)*i):(int(len(data)/k)*(i+1))]
    train = data[~data['prescriptionNumber'].isin(test['prescriptionNumber'])]
    dataCollection[i] = dict()
    dataCollection[i]['Test'] = test
    dataCollection[i]['Train'] = train
  return dataCollection
```

Figure E.1 Function to Split Data Into K Fold(s)

```python
def validate(data, k):
  allScoreSum = dict()
  for i in tqdm(data):
    print(f'Start of {(i)}-st/th Fold Validation')
    print('==============================')
    trainSet = data[i]['Train']
    testSet = data[i]['Test']
    # print(testSet.info)
    trainData, trainPatients, trainPivot = prepro(trainSet, realPatients, realPivot)
    testData, testPatients, testPivot = prepro(testSet, realPatients, realPivot)
    scoreSum = dict()
    runtimeAll = 0
    generalRecallSum = 0
    generalPrecisionSum = 0
    generalf1ScoreSum = 0
    with tqdm(total=testData.shape[0]) as pbar:
        for index, rows in testData.iterrows():
            # print(rows)
            result, recall, precision, f1score, runtime = predict(rows,trainData, testPivot,trainPivot,testPatients,trainPatients,151,51)
            firstICD10 = json.loads(rows['differentialDiagnosis'])[0]
            #print(firstICD10)
            if str(firstICD10) not in scoreSum:
               scoreSum[firstICD10] = {'Recall': recall,
                                       'Precision': precision,
                                       'F1 Score': f1score}
               #print(scoreSum[firstICD10])
               generalRecallSum += recall
               generalPrecisionSum += precision
               generalf1ScoreSum += f1score
               runtimeAll += runtime
               pbar.update(1)
            else:
               scoreSum[firstICD10] = {'Recall': scoreSum[firstICD10]['Recall'] + recall,
                                       'Precision': scoreSum[firstICD10]['Precision'] + precision,
                                       'F1 Score': scoreSum[firstICD10]['F1 Score'] + f1score}
               #print(scoreSum[firstICD10])
               generalRecallSum += recall
               generalPrecisionSum += precision
               generalf1ScoreSum += f1score
               runtimeAll += runtime
               pbar.update(1)
    allScoreSum[i] = scoreSum
    avgRecall = generalRecallSum/len(data)
    avgPresicion = generalPrecisionSum/len(data)
    avgF1Score = generalf1ScoreSum/len(data)
    avgRuntime = runtimeAll/len(data)
  return allScoreSum, avgRecall, avgPresicion, avgF1Score, avgRuntime
```

Figure E.2 Function to Run KNN K-Fold Cross Validation

F. Model Deployment Code Functions

```
def show_app_page():
    st.title("Welcome to e-Recommendation Prescription!")
    st.subheader("#Patient Information")
    st.write("We need some information to give the well-suited recommendations")
```

Figure F.1 Function to Show Title and Header of The Web App

```
def getDifferentialDiagnosis(newPresc, diagnosis1, diagnosis2, diagnosis3, diagnosis4, diagnosis5):
    tempString = "diagnosis"
    symbol = ["*&", "+&", "-&", "//&"]
    #symbolIndex = [-1,0,1,2]
    for i in range(5):
            currentCounter = i+1
            stringCounter = str(i+1)
            inputDiagnosis = locals()[tempString+stringCounter]
            currentDiagnosis = str(tempString+stringCounter)
            if inputDiagnosis == "NaN":
                newPresc['differentialDiagnosis'] = newPresc['differentialDiagnosis'].replace(currentDiagnosis, '^')
                newPresc['differentialDiagnosis'] = newPresc['differentialDiagnosis'].replace('"^"', '')
                if currentCounter > 1:
                    newPresc['differentialDiagnosis'] = newPresc['differentialDiagnosis'].replace(symbol[i-1], '')
            else:
                newPresc['differentialDiagnosis'] = newPresc['differentialDiagnosis'].replace(currentDiagnosis, inputDiagnosis)
                if currentCounter > 1:
                    newPresc['differentialDiagnosis'] = newPresc['differentialDiagnosis'].replace(symbol[i-1], ",")
    return newPresc
```

Figure F.2 Function to Get Input Differential Diagnosis

```
def classifyAge(age):
    age = int(age)
    if 0 <= age <=12:
        category = 'Child'
    elif 13 <= age <= 18:
        category = 'Adolescence'
    elif 19 <= age <= 59:
        category = 'Adult'
    else:
        category = 'Senior Adult'
    return category
```

Figure F.3 Function to Classify Input Age Datum

```
def calculateBMI(weight, height):
  w = weight
  h = 0.01 * height
  bmi = w/(h*h)
  return round(bmi, 1)
```

Figure F.4 Function to Calculate Patient's BMI

```
def classifyBMI(weight, height):
    bmi = calculateBMI(weight, height)
    if bmi < 18.5:
        category = 'Underweight'
    elif 18.5 <= bmi <= 24.9:
        category = 'Normal'
    #elif 25 <= df['patientBMI'][i] <= 29.9:
    elif bmi >= 25:
        category = 'Overweight'
    return category
```

Figure F.5 Function to Classify Input BMI Datum

```
def classifyGender(gender):
    if gender == "Female":
        gender = "f"
    else:
        gender = "m"
    return gender
```

Figure F.6 Function to Classify Input Gender Datum

```
def getPatientInfo(age, weight, height, gender):
    patientAgeCategory = classifyAge(age)
    patientBMICategory = classifyBMI(weight, height)
    gender = classifyGender(gender)
    return patientAgeCategory, patientBMICategory, gender
```

Figure F.7 Function to Get Patient Information in Age, BMI, and Gender Category

```
def getNewPresc(newPresc, diagnosis1, diagnosis2, diagnosis3, diagnosis4, diagnosis5):
    newPresc = getDifferentialDiagnosis(newPresc, diagnosis1, diagnosis2, diagnosis3, diagnosis4, diagnosis5)
    return newPresc
```

Figure F.8 Function to Compile Differential Diagnoses

```
def giveRecommendaion()
  ok = st.button("Give Recommendation")
  if ok:
    newPresc = pd.Series({
    "patientId": "{}".format(patientId),
    "patientGender": "{}".format(gender),
    "age": str(age),
    "heights": str(height),
    "weight": str(weight),
    "differentialDiagnosis": '["diagnosis1"*&"diagnosis2"+&"diagnosis3"-&"diagnosis4"//&"diagnosis5"]',
    "patientAgeCategory": "{}".format(patientAgeCategory),
    "patientBMICategory":"{}".format(patientBMICategory)
    })

      newPresc = getNewPresc(newPresc, diagnosis1, diagnosis2, diagnosis3, diagnosis4, diagnosis5)
      medicineList, medsWithDosage, recall, precision, f1score, runtime = cf.predict(newPresc, allPresc, allPivot,
                                                                                      patients, medicine_dose)

    return medicineList, medsWithDosage
```

Figure F.9 Function to Give Recommendations