Hospital Readmission Prediction Using Machine Learning

Project Description:

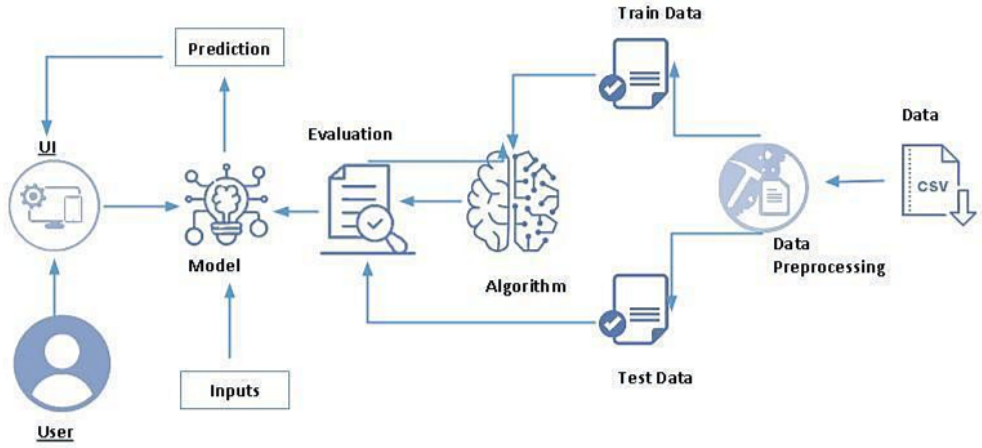
If a hospital has multiple readmissions, it means that the hospital needs to work on the quality of services it is providing with respect to the health and wellness of its patients. Being able to predict whether a person will be readmitted to the hospital within 30 days or not, will be of great help to the hospital in developing an idea of the incoming number of repeated patients which in turn helps to provide better services for patients with increased risk of disease.

One patient population that is at increased risk of hospitalization and readmission is diabetes. Diabetes is a medical condition that affects approximately 1 in 10 patients in the United States. So in this project, we will be focusing on hospital readmission prediction for patients who are having diabetes.

This study used the Health Facts database (Cerner Corporation, Kansas City, MO), a national data warehouse that collects comprehensive clinical records across hospitals throughout the United States. The Health Facts data we used was an extract representing 10 years (1999–2008) of clinical care at 130 hospitals and integrated delivery networks throughout the United States.

The main purpose of this project is to predict whether a person who is suffering from diabetes and consulting a specific hospital will be readmitted or not, based on multiple factors.
We will be using classification algorithms such as Logistic Regression, KNN, Decision tree, Random Forest, AdaBoost, and GradientBoost. We will train and test the data with these algorithms. From this, the best model is selected and saved in pkl format. We will also be deploying our model locally using Flask.

Technical Architecture:



Pre requisites:

To complete this project, you will require the following software, concepts, and packages

Anaconda navigator:

- Refer to the link below to download the anaconda navigator

Python packages:

- Open anaconda prompt as administrator
- Type “pip install numpy” and click enter.
- Type “pip install pandas” and click enter.
- Type “pip install scikit-learn” and click enter.
- Type ”pip install matplotlib” and click enter.
- Type ”pip install scipy” and click enter.
- Type ”pip install pickle-mixin” and click enter.
- Type ”pip install seaborn” and click enter.
- Type “pip install Flask” and click enter.

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- ML Concepts

Supervised learning:
Unsupervised learning
Metrics
Flask

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding of data.
- Know how to deal with imbalanced target variables.
- Have knowledge of pre-processing the data/transformation techniques and some visualization concepts before building the model
- Learn how to build a machine learning model and tune it for better performance
- Know how to evaluate the model and deploy it using flask

Project Flow:

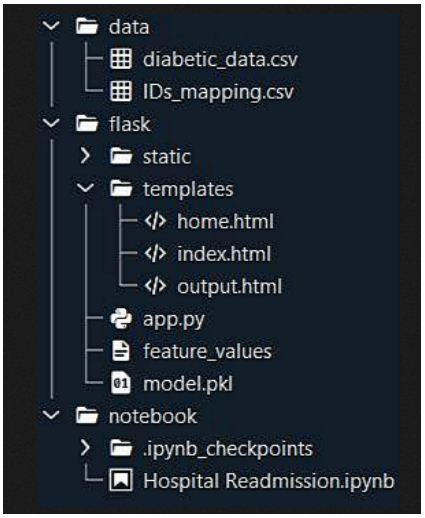
- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- The predictions made by the model are showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data collection:
  - Download the dataset
- Data pre-processing:
  - Handling null values and removing unnecessary columns
- Visualising and analyzing data:
  - Univariate analysis
  - Bivariate analysis
  - Descriptive analysis
- Model building:
  - Handling categorical values
  - Dividing data into train and test sets
  - Sampling the data
  - Dividing train data into train and validation sets
  - Comparing the performance of various models

    - Feature Selection
    - Repeat the process of dividing data into train and test sets
    - Evaluating final model performance
    - Save the final model
- Application Building:
    - Building HTML pages
    - Build python code

Project Structure:



Create the Project folder which contains files as shown below

- The data folder contains the .csv file used for our project. diabetic_data.csv contains the main data and the IDs_mapping.csv file gives us information about various IDs present in our dataset.
- The notebook folder contains the HospitalReadmisssion .ipynb file for building the ML model.
- We are building a flask application that needs the HTML pages to be stored in the templates folder and a python script app.py for scripting.
- model.pkl is our saved model. We will use this model for flask integration.

Data Collection:

Download the dataset:

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project, we have used diabetic_data.csv. This data is downloaded from the following research paper:

Load the dataset using read_csv() function:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sb
```

```
1 data = pd.read_csv('D:/SB_Projects/Hospital Readmission Prediction/data/diabetic_data.csv')
```

Inside the read_csv() function, specify the path to your dataset.

To observe the first 5 rows of our data, we use the head() method and to observe the last 5 rows of the data, we use the tail() method.

```
1 data.head()
```

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	...
0	2278392	8222157	Caucasian	Female	[0-10)	?	6	25	1	1	...
1	149190	55629189	Caucasian	Female	[10-20)	?	1	1	7	3	...
2	64410	86047875	AfricanAmerican	Female	[20-30)	?	1	1	7	2	...
3	500364	82442376	Caucasian	Male	[30-40)	?	1	1	7	2	...
4	16680	42519267	Caucasian	Male	[40-50)	?	1	1	7	1	...

5 rows x 50 columns

```
1 data.tail()
```

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospita
101761	443847548	100162476	AfricanAmerican	Male	[70-80)	?	1	3	7	3
101762	443847782	74694222	AfricanAmerican	Female	[80-90)	?	1	4	5	5
101763	443854148	41088789	Caucasian	Male	[70-80)	?	1	1	7	1
101764	443857166	31693671	Caucasian	Female	[80-90)	?	2	3	7	10
101765	443867222	175429310	Caucasian	Male	[70-80)	?	1	1	7	6

5 rows × 50 columns

We can use the 'shape' attribute of the dataframe to know the shape of our dataset:

```
1 data.shape
(101766, 50)
```

From the above figure, we can say that our dataset has 101766 rows and 50 columns

Next, we will have to see the information pertaining to each of the 50 columns. For that, we will be using info() function:

```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 50 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   encounter_id              101766 non-null  int64
 1   patient_nbr               101766 non-null  int64
 2   race                      101766 non-null  object
 3   gender                    101766 non-null  object
 4   age                       101766 non-null  object
 5   weight                    101766 non-null  object
 6   admission_type_id         101766 non-null  int64
 7   discharge_disposition_id  101766 non-null  int64
 8   admission_source_id       101766 non-null  int64
 9   time_in_hospital          101766 non-null  int64
 10  payer_code                101766 non-null  object
 11  medical_specialty         101766 non-null  object
 12  num_lab_procedures        101766 non-null  int64
 13  num_procedures            101766 non-null  int64
 14  num_medications           101766 non-null  int64
 15  number_outpatient         101766 non-null  int64
 16  number_emergency          101766 non-null  int64
 17  number_inpatient          101766 non-null  int64
 18  diag_1                    101766 non-null  object
 19  diag_2                    101766 non-null  object
 20  diag_3                    101766 non-null  object
 21  number_diagnoses          101766 non-null  int64
 22  max_glu_serum             101766 non-null  object
```

Abstract:

This data has been prepared to analyze factors related to readmission as well as other outcomes pertaining to patients with diabetes.

Data Pre-processing:

We need to pre-process the collected data before gaining insights and building our model.

We need to clean the dataset properly in order to fetch good results. This activity includes handling null values and removing unnecessary columns.

Handling Null values and removing unnecessary columns:

Though the dataset seems to be completely free of null values, it is not so. We observe from the head and tail of data that a number of fields are filled with '?'. These are nothing but null values. So in order to get the null values count of each column, replace all '?' in data with np.nan and

then find the sum of null values.

```
1 data.replace('?', np.NaN, inplace=True)
```

```
1 data.isna().sum()
```

```
encounter_id                  0
patient_nbr                   0
race                       2273
gender                        0
age                           0
weight                    98569
admission_type_id             0
discharge_disposition_id      0
admission_source_id           0
time_in_hospital              0
payer_code                40256
medical_specialty         49949
num_lab_procedures            0
num_procedures                0
num_medications               0
number_outpatient             0
number_emergency              0
number_inpatient              0
diag_1                       21
diag_2                      358
diag_3                     1423
number_diagnoses              0
max_glu_serum                 0
A1Cresult                     0
metformin                     0
repaglinide                   0
nateglinide                   0
chlorpropamide                0
glimepiride                   0
acetohexamide                 0
glipizide                     0
glyburide                     0
tolbutamide                   0
pioglitazone                  0
rosiglitazone                 0
acarbose                      0
miglitol                      0
troglitazone                  0
tolazamide                    0
examide                       0
citoglipton                   0
insulin                       0
glyburide-metformin           0
glipizide-metformin           0
glimepiride-pioglitazone      0
metformin-rosiglitazone       0
metformin-pioglitazone        0
change                        0
diabetesMed                   0
readmitted                    0
dtype: int64
```

We observe that 3 columns - weight, payer_code and medical_speciality contain a huge number of null values. So we need to drop these columns.

Also, we need to check for columns that have a very large number of unique values and cannot be bucketed. For this purpose, we will use the nunique() function.

```
1 data.nunique()
```

```
encounter_id                101766
patient_nbr                  71518
race                             6
gender                           3
age                             10
weight                          10
admission_type_id                8
discharge_disposition_id        26
admission_source_id             17
time_in_hospital                14
payer_code                      18
medical_specialty               73
num_lab_procedures             118
num_procedures                   7
num_medications                 75
number_outpatient               39
number_emergency                33
number_inpatient                21
diag_1                         717
diag_2                         749
diag_3                         790
number_diagnoses                16
max_glu_serum                    4
A1Cresult                        4
metformin                        4
repaglinide                      4
nateglinide                      4
chlorpropamide                   4
glimepiride                      4
acetohexamide                    2
glipizide                        4
glyburide                        4
tolbutamide                      2
pioglitazone                     4
rosiglitazone                    4
acarbose                         4
miglitol                         4
troglitazone                     2
tolazamide                       3
examide                          1
citoglipton                      1
insulin                          4
glyburide-metformin              4
glipizide-metformin              2
glimepiride-pioglitazone         2
metformin-rosiglitazone          2
metformin-pioglitazone           2
change                           2
diabetesMed                      2
readmitted                       3
dtype: int64
```

From the above result, we can remove encounter_id and patient_nbr as they have a large amount of unique values. We can also remove examide and citoglipton as they have only 1 unique value and hence do not provide any information.

So, let us drop all of these columns using drop() method.

```
cols_to_drop = ['weight', 'payer_code','medical_specialty','encounter_id','patient_nbr','examide','citoglipton']
```

```
data = data.drop(cols_to_drop, axis=1)
```

```
data.shape
```

```
(101763, 43)
```

Let us remove the rows which have gender = Unknown/Invalid

```
data['gender'].value_counts()
```

```
Female             54708
Male               47055
Unknown/Invalid        3
Name: gender, dtype: int64
```

```
data = data[data['gender']!='Unknown/Invalid']
```

Now, let us drop all rows containing null values. Since the number is very less as compared to the total number of rows, it won't affect our model.

```
data.dropna(how='any',axis=0,inplace=True)
```

```
data.shape
```

```
(98052, 43)
```

Our dataset is now free from null values. The data now has 98052 rows and 43 columns. Before proceeding with visualisation, let us encode our target variable.

```
data['readmitted'] = data['readmitted'].replace('>30', 0)
data['readmitted'] = data['readmitted'].replace('<30', 1)
data['readmitted'] = data['readmitted'].replace('NO', 0)
```

Now we can proceed with visualising data.

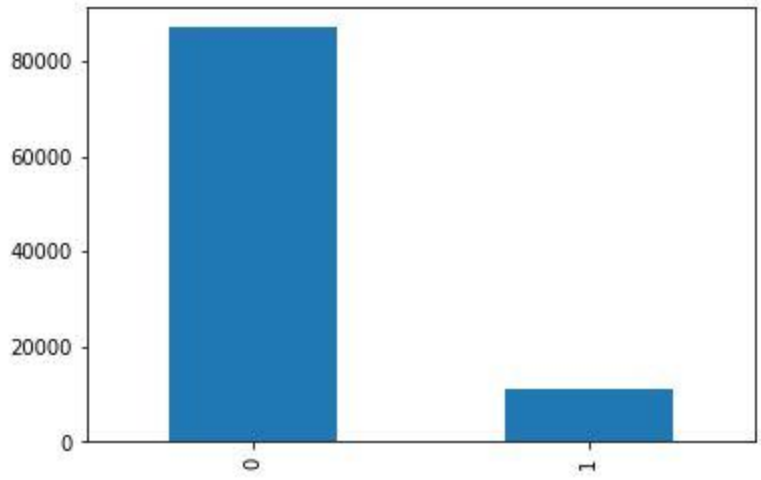
Data analysis and visualization:

Uni variate analysis:

In simple words, univariate analysis is understanding the data with a single feature.
Let us first plot the values of our target column - readmitted

```
1 data['readmitted'].value_counts().plot(kind='bar')
```

```
<AxesSubplot:>
```



From the above figure, it is observed that the target is quite imbalanced. So, before proceeding with model building, we will be balancing the target data.

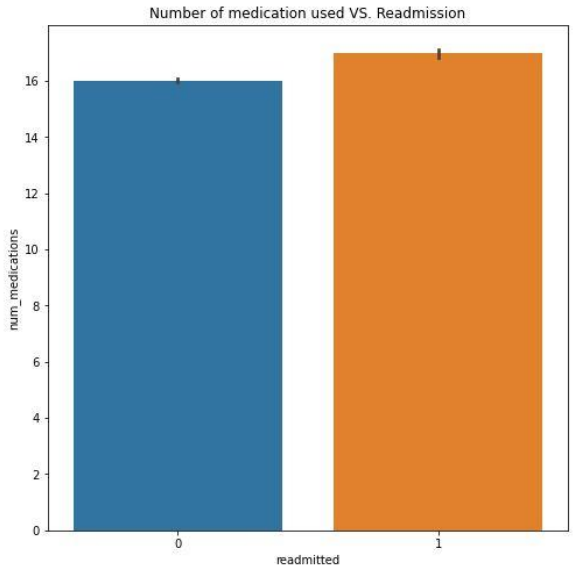
Bi variate analysis:

We use bivariate analysis to find the relation between two features. Here we are visualising the relationship of various features with respect to readmitted, which is our target variable.

- Number of medications used and readmitted

```
fig = plt.figure(figsize=(8,8))
sb.barplot(x = data['readmitted'], y = data['num_medications']).set_title("Number of medication used VS. Readmission")
```

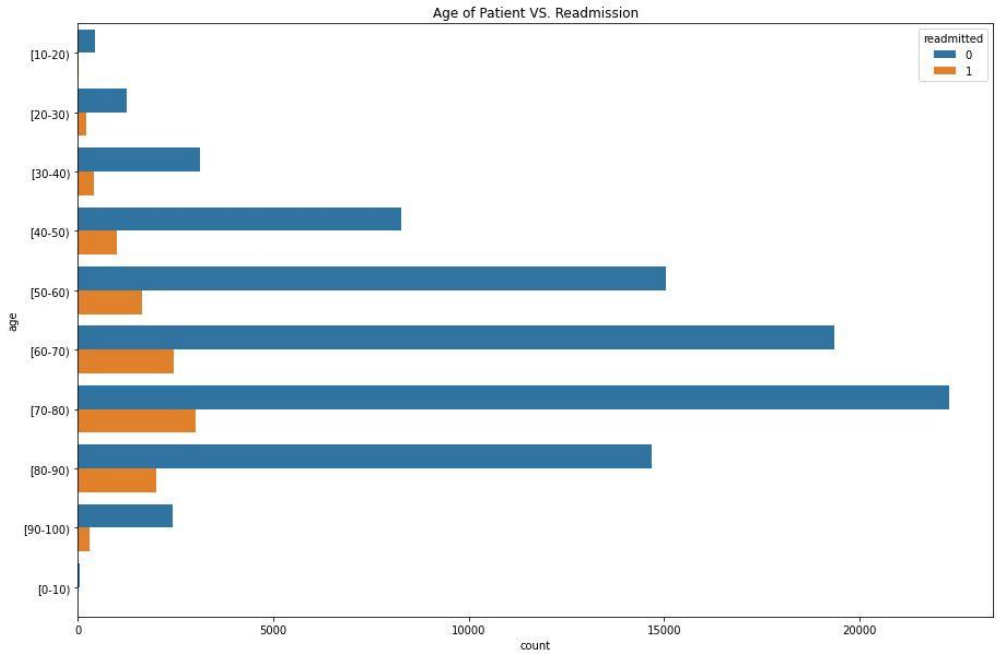
```
Text(0.5, 1.0, 'Number of medication used VS. Readmission')
```



- Age and readmitted

```
fig = plt.figure(figsize=(15,10))
sb.countplot(y= data['age'], hue = data['readmitted']).set_title('Age of Patient VS. Readmission')
```

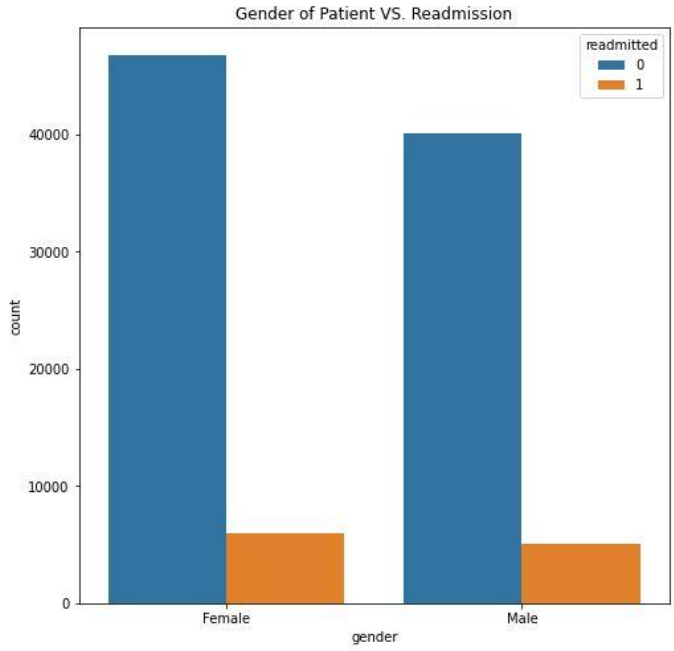
```
Text(0.5, 1.0, 'Age of Patient VS. Readmission')
```



- Gender and readmitted

```
1 fig = plt.figure(figsize=(8,8))
2 sb.countplot(x=data['gender'], hue = data['readmitted']).set_title("Gender of Patient VS. Readmission")
```

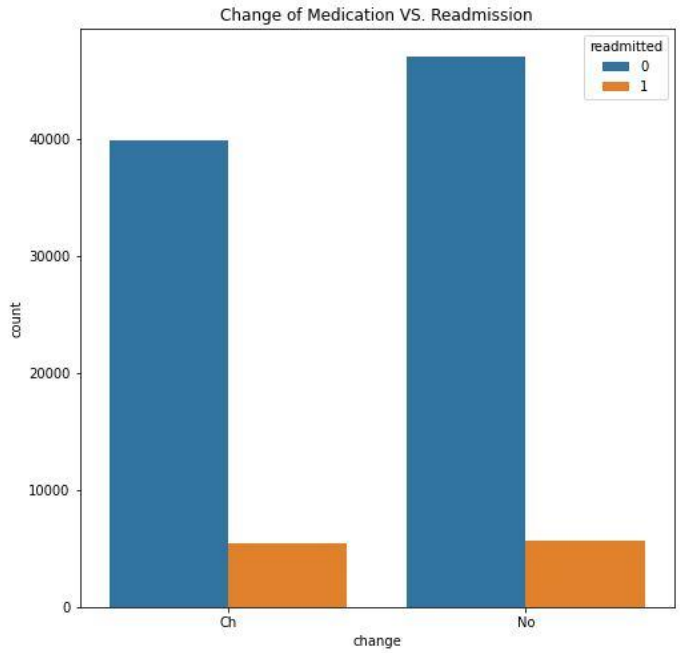
```
Text(0.5, 1.0, 'Gender of Patient VS. Readmission')
```



- Change of medication and readmitted

```
1 fig = plt.figure(figsize=(8,8))
2 sb.countplot(x=data['change'], hue = data['readmitted']).set_title('Change of Medication VS. Readmission')
```

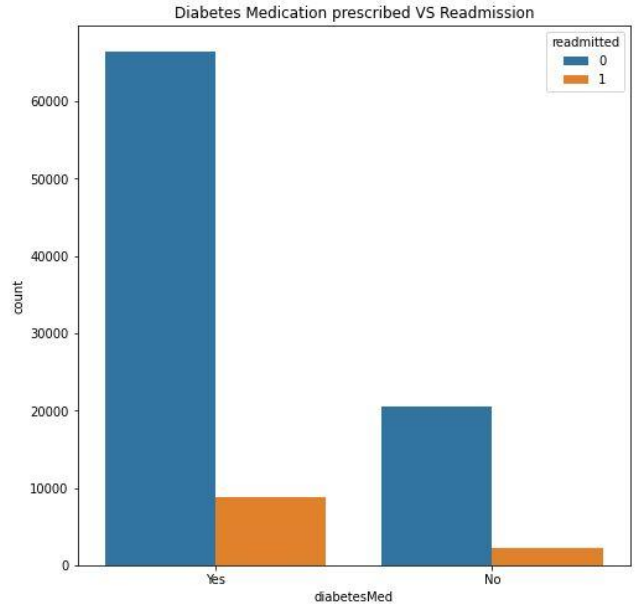
```
Text(0.5, 1.0, 'Change of Medication VS. Readmission')
```



- Diabetes Medication prescribed and readmitted

```
1 fig = plt.figure(figsize=(8,8))
2 sb.countplot(x=data['diabetesMed'], hue = data['readmitted']).set_title('Diabetes Medication prescribed VS Readmission')
```

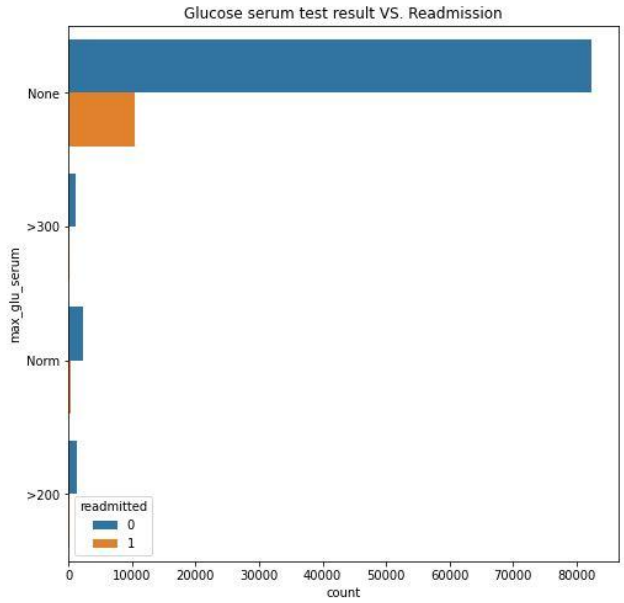
```
Text(0.5, 1.0, 'Diabetes Medication prescribed VS Readmission')
```



- Glucose serum test and readmitted

```
1 fig = plt.figure(figsize=(8,8))
2 sb.countplot(y = data['max_glu_serum'], hue = data['readmitted']).set_title('Glucose serum test result VS. Readmission')
```

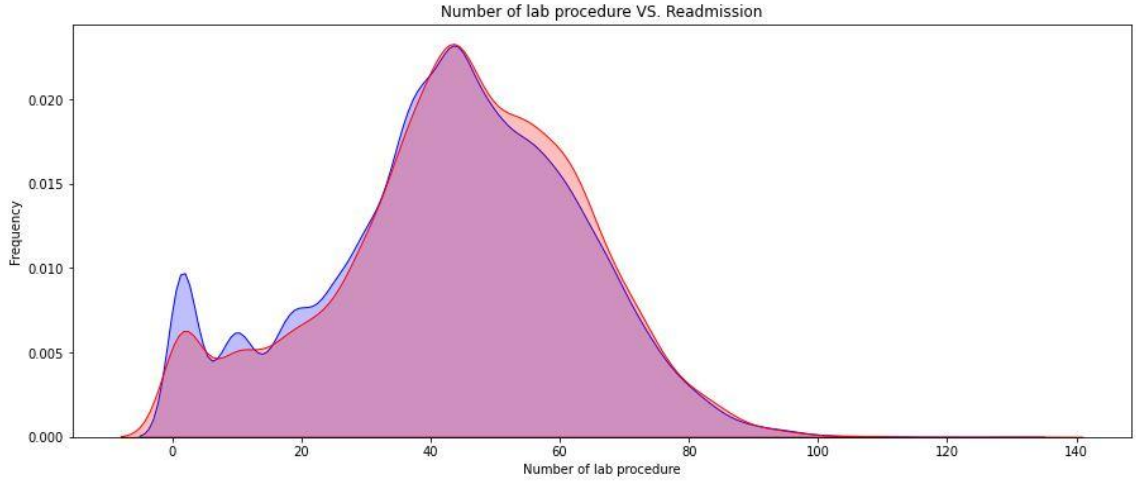
```
Text(0.5, 1.0, 'Glucose serum test result VS. Readmission')
```



- Number of lab procedures and readmitted

```
fig = plt.figure(figsize=(15,6),)
ax=sb.kdeplot(data.loc[(data['readmitted'] == 0),'num_lab_procedures'] , color='b',shade=True,label='Not readmitted')
ax=sb.kdeplot(data.loc[(data['readmitted'] == 1),'num_lab_procedures'] , color='r',shade=True, label='readmitted')
ax.set(xlabel='Number of lab procedure', ylabel='Frequency')
plt.title('Number of lab procedure VS. Readmission')
```

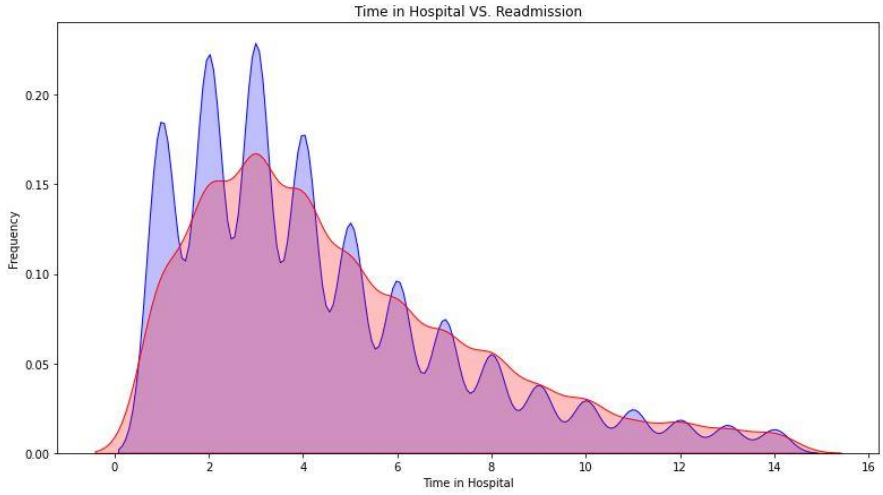
```
Text(0.5, 1.0, 'Number of lab procedure VS. Readmission')
```



- Time in hospital and readmitted

```
fig = plt.figure(figsize=(13,7),)
ax=sb.kdeplot(data.loc[(data['readmitted'] == 0),'time_in_hospital'] , color='b',shade=True,label='Not Readmitted')
ax=sb.kdeplot(data.loc[(data['readmitted'] == 1),'time_in_hospital'] , color='r',shade=True, label='Readmitted')
ax.set(xlabel='Time in Hospital', ylabel='Frequency')
plt.title('Time in Hospital VS. Readmission')
```

```
Text(0.5, 1.0, 'Time in Hospital VS. Readmission')
```



Descriptive analysis:

Descriptive analysis is to study the basic statistical features of data. We can achieve it by using the .describe() function. With this describe function we can understand the unique, top, and frequent values of categorical features. Also, we can find mean, std, min, max and percentile values of numerical features.

```
1 data.describe(include='all')
```

	race	gender	age	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	num_lab_procedures	num_procedures
count	98052	98052	98052	98052.000000	98052.000000	98052.000000	98052.000000	98052.000000	98052.00000(
unique	5	2	10	NaN	NaN	NaN	NaN	NaN	NaN
top	Caucasian	Female	[70-80)	NaN	NaN	NaN	NaN	NaN	NaN
freq	75079	52833	25305	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	2.025803	3.753396	5.776741	4.422011	43.148462	1.350712
std	NaN	NaN	NaN	1.450121	5.309412	4.071632	2.993070	19.711757	1.708475
min	NaN	NaN	NaN	1.000000	1.000000	1.000000	1.000000	1.000000	0.00000(
25%	NaN	NaN	NaN	1.000000	1.000000	1.000000	2.000000	31.000000	0.00000(
50%	NaN	NaN	NaN	1.000000	1.000000	7.000000	4.000000	44.000000	1.00000(
75%	NaN	NaN	NaN	3.000000	4.000000	7.000000	6.000000	57.000000	2.00000(
max	NaN	NaN	NaN	8.000000	28.000000	25.000000	14.000000	132.000000	6.00000(

11 rows × 43 columns

Model Building:

Handling Categorical Values:

As we can see our dataset has categorical data. Before training our model, we must convert the categorical data into a numeric form.

There are multiple encoding techniques to convert the categorical columns into numerical columns. For this project we will be encoding some features manually and some others using OrdinalEncoder()

Firstly, let us modify the values in the columns admission_type_id, discharge_disposition_id and admission_source_id with the help of IDs_mapping.csv file.

```
1 data = data.replace({'admission_type_id':{2:1,7:1,6:5,8:5}, 'discharge_disposition_id':{6:1,8:1,9:1,13:1,3:2,4:2,5:2,
2                     14:2,22:2,23:2,24:2,12:10,15:10,16:10,17:10,25:18,26:18}, 'admission_source_id':{2:1,3:1,5:4,6:4,
3                     10:4,22:4,25:4,15:9,17:9,20:9,21:9,13:11,14:11}})
```

Next, let us filter out drugs that are rarely prescribed and keep the ones which are prescribed more often. After that, we will encode the values of drugs prescribed.

```
drugs = ['metformin', 'repaglinide', 'glimepiride', 'glipizide',
                'glyburide', 'pioglitazone', 'rosiglitazone', 'insulin']
drugs_to_drop =["nateglinide", "chlorpropamide", "acetohexamide", "tolbutamide",
             "acarbose", "miglitol", "troglitazone", "tolazamide", "glyburide-metformin",
                "glipizide-metformin","glimepiride-pioglitazone", "metformin-rosiglitazone", "metformin-pioglitazone"]
```

```
for col in drugs:
    data[col] = data[col].replace('No', 0)
    data[col] = data[col].replace('Steady', 1)
    data[col] = data[col].replace('Up', 1)
    data[col] = data[col].replace('Down', 1)
```

```
data.drop(drugs_to_drop, axis=1,inplace=True)
```

Next, let us encode the values of columns - A1Cresult and max_glu_serum.

```
data['A1Cresult'] = data['A1Cresult'].replace('>7', 1)
data['A1Cresult'] = data['A1Cresult'].replace('>8', 1)
data['A1Cresult'] = data['A1Cresult'].replace('Norm', 0)
data['A1Cresult'] = data['A1Cresult'].replace('None', -99)

data['max_glu_serum'] = data['max_glu_serum'].replace('>200', 1)
data['max_glu_serum'] = data['max_glu_serum'].replace('>300', 1)
data['max_glu_serum'] = data['max_glu_serum'].replace('Norm', 0)
data['max_glu_serum'] = data['max_glu_serum'].replace('None', -99)
```

Now, let us convert age column into numeric by selecting the midpoint of the range given in the dataset

```
data['age'].value_counts()
```

```
[70-80)     25305
[60-70)     21809
[80-90)     16702
[50-60)     16697
[40-50)      9265
[30-40)      3548
[90-100)     2717
[20-30)      1478
[10-20)       466
[0-10)         65
Name: age, dtype: int64
```

```
for i in range(0,10):
    data['age'] = data['age'].replace('['+str(10*i)+'-'+str(10*(i+1))+')', (10*i+10*(i+1))//2)
data['age'].value_counts()
```

```
75    25305
65    21809
85    16702
55    16697
45     9265
35     3548
95     2717
25     1478
15      466
5        65
Name: age, dtype: int64
```

The columns diag_1, diag_2, diag_3 contain icd9 code values that represent the diagnosis.

The following abbreviations are used for particular icd9 codes: “circulatory” for icd9: 390–459, 785, “digestive” for icd9: 520–579, 787, “genitourinary” for icd9: 580–629, 788, “diabetes” for icd9: 250.xx, “injury” for icd9: 800–999, “musculoskeletal” for icd9: 710–739, “neoplasms” for icd9: 140–239, “respiratory” for icd9: 460–519, 786, and “other” for otherwise.

So, let us first create a function that categorises the type of diagnosis based on icd9 code value and apply it to all the 3 columns:

```
def conv_diag(icd9):
    try:
        n = float(icd9)
        if (n>=390 and n<=459) or (n==785):
            return "circulatory"
        elif (n>=520 and n<=579) or (n==787):
            return "digestive"
        elif (n>=580 and n<=629) or (n==788):
            return "genitourinary"
        elif np.trunc(n)==250:
            return "diabetes"
        elif (n>=800 and n<=999):
            return "injury"
        elif (n>=710 and n<=739):
            return "musculoskeletal"
        elif (n>140 and n<=239):
            return "neoplasms"
        elif (n>=460 and n<=519) or (n==786):
            return "respiratory"
        else :
            return "other"
    except:
        return "other"
```

```
data['diag_1'] = data['diag_1'].apply(conv_diag)
data['diag_2'] = data['diag_2'].apply(conv_diag)
data['diag_3'] = data['diag_3'].apply(conv_diag)
```

This is the end of encoding columns manually.

Before we continue with using OrdinalEncoder() for remaining categorical columns, let us first divide our data into features and target:

```
X = data.drop('readmitted', axis = 1)
y = data['readmitted']
```

Here X contains our features and y contains our target.

The next step is to apply ordinal encoding technique on categorical features. For that purpose, we will be using ColumnTransformer().

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder
```

```
ct =  ColumnTransformer([('oe',OrdinalEncoder(),['diag_1','diag_2','diag_3','race','change','gender','diabetesMed'])]
                        ,remainder='passthrough')
```

```
X = ct.fit_transform(X)
```

We need to save the column transformer instance so that we can use it during our model deployment.

```
import joblib
joblib.dump(ct,'feature_values')
```

After executing the above lines, ct will be saved in a file known as feature_values.

Splitting data into train and test sets:

For splitting the data into train and test sets, we are using the train_test_split() function from sklearn. As parameters, we are passing X, y,stratify, test_size, random_state.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    stratify=y,
                                                    test_size=0.3,random_state=20)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((68636, 29), (29416, 29), (68636,), (29416,))
```

Sampling data:

We will be using SMOTE algorithm for oversampling our minority class. To know in detail about this algorithm.

```
from imblearn.over_sampling import SMOTE
```

```
sm = SMOTE(random_state = 20)
X1_res, y1_res = sm.fit_sample(X_train,y_train)
```

```
X1_res = np.array(X1_res)
```

```
y1_res = np.array(y1_res)
```

```

X_train, X_val, y_train, y_val = train_test_split(X1_res,y1_res,test_size=0.3, random_state=20)
```

```
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

```
((85246, 29), (36534, 29), (85246,), (36534,))
```

Splitting train data into train and validation sets:

```
from imblearn.over_sampling import SMOTE
```

```
sm = SMOTE(random_state = 20)
X1_res, y1_res = sm.fit_sample(X_train,y_train)
```

```
X1_res = np.array(X1_res)
```

```
y1_res = np.array(y1_res)
```

```

X_train, X_val, y_train, y_val = train_test_split(X1_res,y1_res,test_size=0.3, random_state=20)
```

```
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

```
((85246, 29), (36534, 29), (85246,), (36534,))
```

Comparing performance of various models:

We will be considering multiple models to train our data and choose the one that performs the best. So, we need to import the necessary libraries and create a dictionary of our models.

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from xgboost.sklearn import XGBClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score, roc_curve, confusion_matrix, classification_report, auc
```

```
#Creating a dictionary of all models
model_dict = {}

model_dict['Logistic regression']= LogisticRegression(solver='liblinear',random_state=20)
model_dict['KNN Classifier'] = KNeighborsClassifier()
model_dict['Decision Tree Classifier'] = DecisionTreeClassifier(random_state=20)
model_dict['Random Forest Classifier'] = RandomForestClassifier(random_state=20)
model_dict['AdaBoost Classifier'] = AdaBoostClassifier(random_state=20)
model_dict['Gradient Boosting Classifier'] = GradientBoostingClassifier(random_state=20)
model_dict['XGB Classifier'] = XGBClassifier(random_state=20)
```

Next, we will define a function known as model_test() that accepts 6 parameters - X_train, X_test, y_train, y_test, model, model_name. We will obtain y_pred by using .predict() function and compute the training accuracy score for every model by iterating through the dictionary.

```
#function to print accuracy of all models
def model_test(X_train, X_test, y_train, y_test,model,model_name):
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    print('====================================={}====================================='.format(model_name))
    print('Score is : {}'.format(accuracy))

    print()
```

```
for model_name,model in model_dict.items():
    model_test(X_train, X_val, y_train, y_val, model, model_name)
```

```
=====================================Logistic regression=====================================
Score is : 0.6002080254009964

=====================================KNN Classifier=====================================
Score is : 0.7935621612744292

=====================================Decision Tree Classifier=====================================
Score is : 0.8801664203207971

=====================================Random Forest Classifier=====================================
Score is : 0.938632506706082

=====================================AdaBoost Classifier=====================================
Score is : 0.9139158044561231

=====================================Gradient Boosting Classifier=====================================
Score is : 0.9317621941205453

=====================================XGB Classifier=====================================
Score is : 0.9375376361745223
```

Similarly, let us find test accuracies:

```
for model_name, model in model_dict.items():
    p = model.predict(X_test)
    print('Testing accuracy of ',model_name,'=',accuracy_score(y_test,p))
```

```
Testing accuracy of  Logistic regression = 0.6484226271416916
Testing accuracy of  KNN Classifier = 0.5848517813434866
Testing accuracy of  Decision Tree Classifier = 0.7861028011966277
Testing accuracy of  Random Forest Classifier = 0.8871702474843622
Testing accuracy of  AdaBoost Classifier = 0.8744220832200164
Testing accuracy of  Gradient Boosting Classifier = 0.8873742181125918
Testing accuracy of  XGB Classifier = 0.8872382376937721
```

From the above results, it is clear that Random Forest Classifier provides the best accuracy. So let us create a different variable to fit and make predictions using the model.

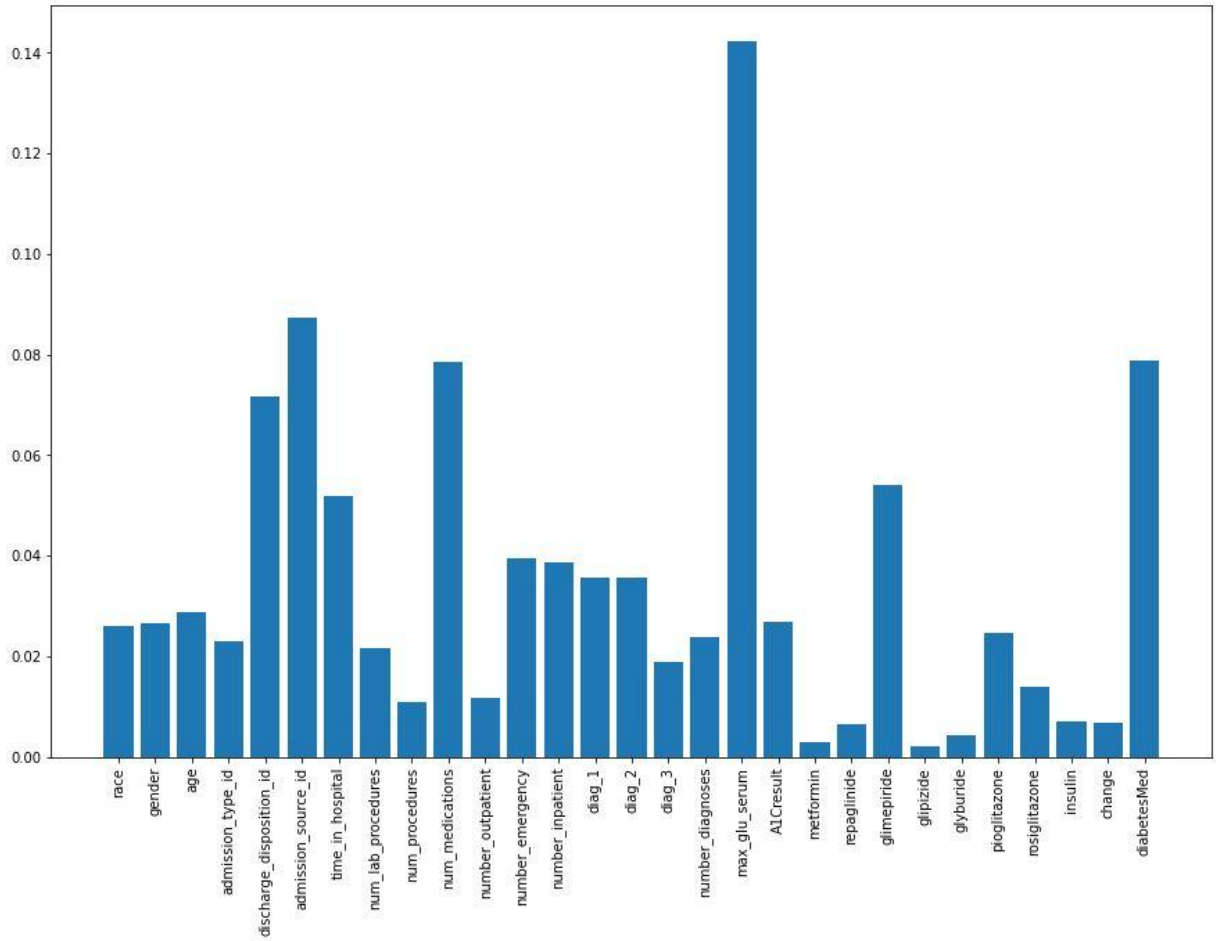
```
#Fitting data to Random Forest classifier
rfc = RandomForestClassifier(random_state=20)
rfc.fit(X_train,y_train)
pred_rfc = rfc.predict(X_val)
print('Training Accuracy of Random Forest=',accuracy_score(y_val,pred_rfc))
```

```
Training Accuracy of Random Forest= 0.938632506706082
```

Feature Selection:

We have trained our model with 29 features. But all these features may not be important for prediction. Hence we will select the features that contribute significantly to the model performance.

```
importance = rfc.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
print(cols)
plt.figure(figsize=(15,10))
plt.bar([cols[x] for x in range(len(importance))], importance)
plt.xticks(rotation=90)
plt.show()
```



Let us print the columns that are important for model performance:

```
imp_cols = []
for i,v in enumerate(importance):
    if (v>0.035):
        imp_cols.append(cols[i])
```

```
imp_cols
```

```
['discharge_disposition_id',
 'admission_source_id',
 'time_in_hospital',
 'num_medications',
 'number_emergency',
 'number_inpatient',
 'diag_1',
 'diag_2',
 'max_glu_serum',
 'glimepiride',
 'diabetesMed']
```

Below is the description of imp_cols:

- discharge_disposition_id : Integer identifier corresponding to 29 distinct values, for example, discharged to home, expired, and not available
- admission_source_id : Integer identifier corresponding to 21 distinct values, for example, physician referral, emergency room, and transfer from a hospital
- time_in_hospital : Integer number of days between admission and discharge
- num_medications : Number of distinct generic names administered during the encounter
- number_emergency : Number of emergency visits of the patient in the year preceding the encounter
- number_inpatient : Number of inpatient visits of the patient in the year preceding the encounter
- diag_1 : The primary diagnosis (coded as first three digits of ICD9); 848 distinct values
- diag_2 : The secondary diagnosis (coded as first three digits of ICD9); 923 distinct values
- max_glu_serum : Indicates the range of the result or if the test was not taken. Values: “>200,” “>300,” “normal,” and “none” if not measured
- glimepiride : glimepiride dosage - Values: “up” if the dosage was increased during the encounter, “down” if the dosage was decreased, “steady” if the dosage did not change, and “no” if the drug was not prescribed
- diabetesMed : Indicates if there was any diabetic medication prescribed. Values: “yes” and “no”

Repeat process from dividing data into train and test sets:

```
final_data = data[imp_cols+['readmitted']].copy()
```

```
final_data.shape
```

```
(98052, 12)
```

```
X1 = final_data.drop('readmitted', axis = 1)
y1 = final_data['readmitted']
```

```
ct =  ColumnTransformer([('oe',OrdinalEncoder(),['diag_1','diag_2','diabetesMed'])]
                        ,remainder='passthrough')
```

```
X1 = ct.fit_transform(X1)
```

```
X1.shape, y1.shape
```

```
((98052, 11), (98052,))
```

```
X_train, X_test, y_train, y_test = train_test_split(X1, y1,
                                                    stratify=y1,
                                                    test_size=0.3, random_state=20)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((68636, 11), (29416, 11), (68636,), (29416,))
```

```
from imblearn.over_sampling import SMOTE
```

```
sm = SMOTE(random_state = 20)
X1_res, y1_res = sm.fit_sample(X_train,y_train)
```

```
X1_res = np.array(X1_res)
```

```
y1_res = np.array(y1_res)
```

```

X_train, X_val, y_train, y_val = train_test_split(X1_res,y1_res,test_size=0.3, random_state=20)
```

```
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

```
((85246, 29), (36534, 29), (85246,), (36534,))
```

```

X_train, X_val, y_train, y_val = train_test_split(X1_res,y1_res,test_size=0.3, random_state=20)
```

```
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

```
((85246, 11), (36534, 11), (85246,), (36534,))
```

```
for model_name,model in model_dict.items():
    model_test(X_train, X_val, y_train, y_val, model, model_name)
```

```
====================================Logistic regression=====================================
Score is : 0.5861663108337439

====================================KNN Classifier=====================================
Score is : 0.7962719658400395

====================================Decision Tree Classifier=====================================
Score is : 0.8781956533639897

====================================Random Forest Classifier=====================================
Score is : 0.917309903103958

====================================AdaBoost Classifier=====================================
Score is : 0.8183336070509662

====================================Gradient Boosting Classifier=====================================
Score is : 0.8846280177369026

====================================XGB Classifier=====================================
Score is : 0.9267531614386599
```

```
for model_name, model in model_dict.items():
    p = model.predict(X_test)
    print('Testing accuracy of ',model_name,'=',accuracy_score(y_test,p))
```

```
Testing accuracy of  Logistic regression = 0.6683777536034811
Testing accuracy of  KNN Classifier = 0.6339067174326897
Testing accuracy of  Decision Tree Classifier = 0.7998368234974164
Testing accuracy of  Random Forest Classifier = 0.8569146042969812
Testing accuracy of  AdaBoost Classifier = 0.858784335055752
Testing accuracy of  Gradient Boosting Classifier = 0.8802352461245581
Testing accuracy of  XGB Classifier = 0.8861163992385096
```

```
for model_name,model in model_dict.items():
    model_test(X_train, X_val, y_train, y_val, model, model_name)
```

```
=====================================Logistic regression=====================================
Score is : 0.5861663108337439

=====================================KNN Classifier=====================================
Score is : 0.7962719658400395

=====================================Decision Tree Classifier=====================================
Score is : 0.8781956533639897

=====================================Random Forest Classifier=====================================
Score is : 0.917309903103958

=====================================AdaBoost Classifier=====================================
Score is : 0.8183336070509662

=====================================Gradient Boosting Classifier=====================================
Score is : 0.8846280177369026

=====================================XGB Classifier=====================================
Score is : 0.9267531614386599
```

```
for model_name, model in model_dict.items():
    p = model.predict(X_test)
    print('Testing accuracy of ',model_name,'=',accuracy_score(y_test,p))
```

```
Testing accuracy of  Logistic regression = 0.6683777536034811
Testing accuracy of  KNN Classifier = 0.6339067174326897
Testing accuracy of  Decision Tree Classifier = 0.7998368234974164
Testing accuracy of  Random Forest Classifier = 0.8569146042969812
Testing accuracy of  AdaBoost Classifier = 0.858784335055752
Testing accuracy of  Gradient Boosting Classifier = 0.8802352461245581
Testing accuracy of  XGB Classifier = 0.8861163992385096
```

Evaluating final model performance:

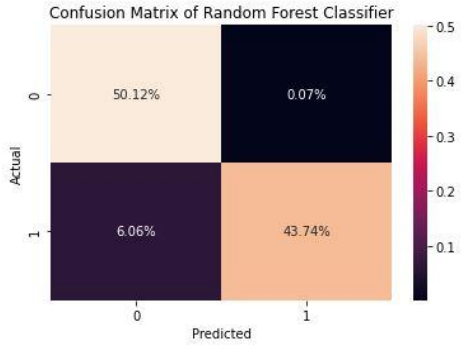
We will compare the confusion matrix, ROC curve and classification report for both models.

In order to obtain these, we will be using the confusion_matrix(),roc_curve() and classification_report() functions from sklearn.metrics.

Confusion matrix:

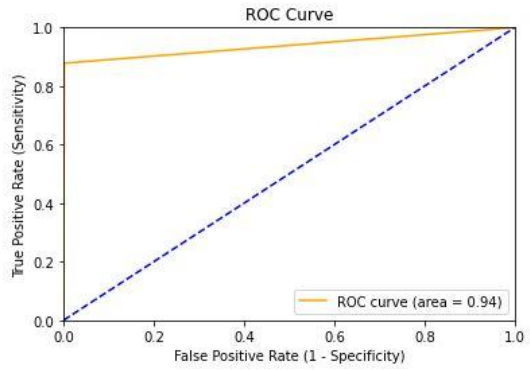
```
#Plotting confusion matrix
cf_matrix = confusion_matrix(y_val, pred_rfc)
sb.heatmap(cf_matrix/np.sum(cf_matrix), annot=True, fmt='.2%')
plt.title('Confusion Matrix of Random Forest Classifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
Text(33.0, 0.5, 'Actual')
```



ROC Curve:

```
#Plotting ROC curve
fpr_rfc, tpr_rfc, thresholds_rfc = roc_curve(y_val, pred_rfc)
roc_auc_rfc = metrics.auc(fpr_rfc, tpr_rfc)
plt.plot(fpr_rfc, tpr_rfc, color='orange', label='ROC curve (area = %0.2f)' % roc_auc_rfc)
plt.plot([0, 1], [0, 1], color='blue', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC Curve')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.legend(loc="lower right")
plt.show()
roc_curve(y_val, pred_rfc)
```



Classification report:

```
print(classification_report(y_val,pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     18338
           1       1.00      0.88      0.93     18196

    accuracy                           0.94     36534
   macro avg       0.95      0.94      0.94     36534
weighted avg       0.94      0.94      0.94     36534
```

Saving the final model:

The final step is saving our model. We can do it by using pickle.dump().

```
import pickle
pickle.dump(rfc,open('model.pkl','wb'))
```

Application Building:

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script

Building Html Pages:

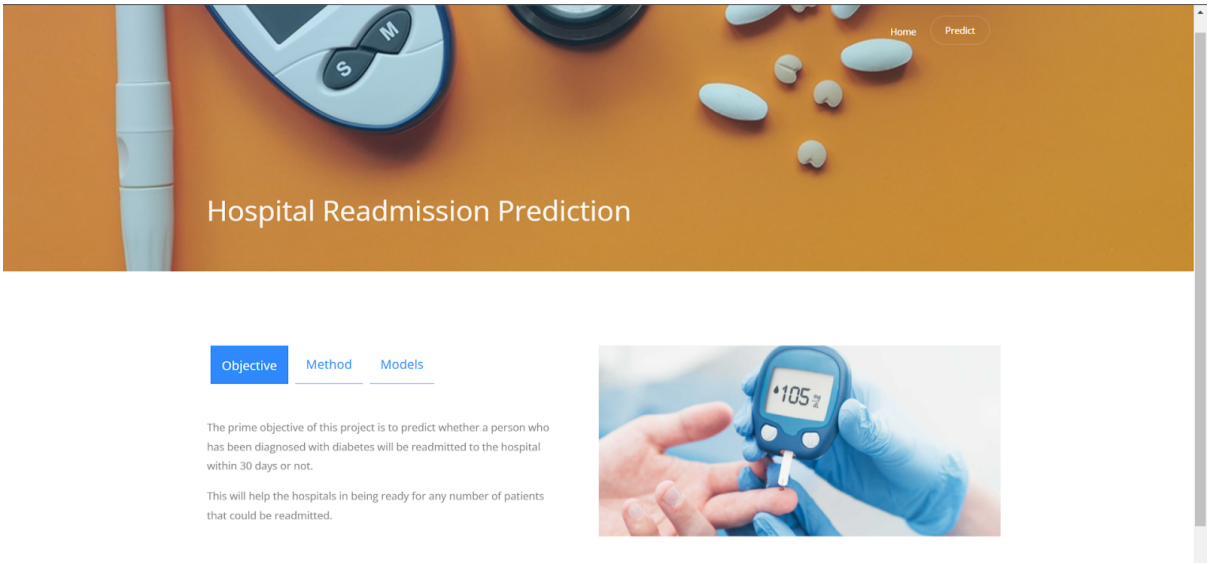
For this project create three HTML files namely

- home.html
- index.html
- output.html

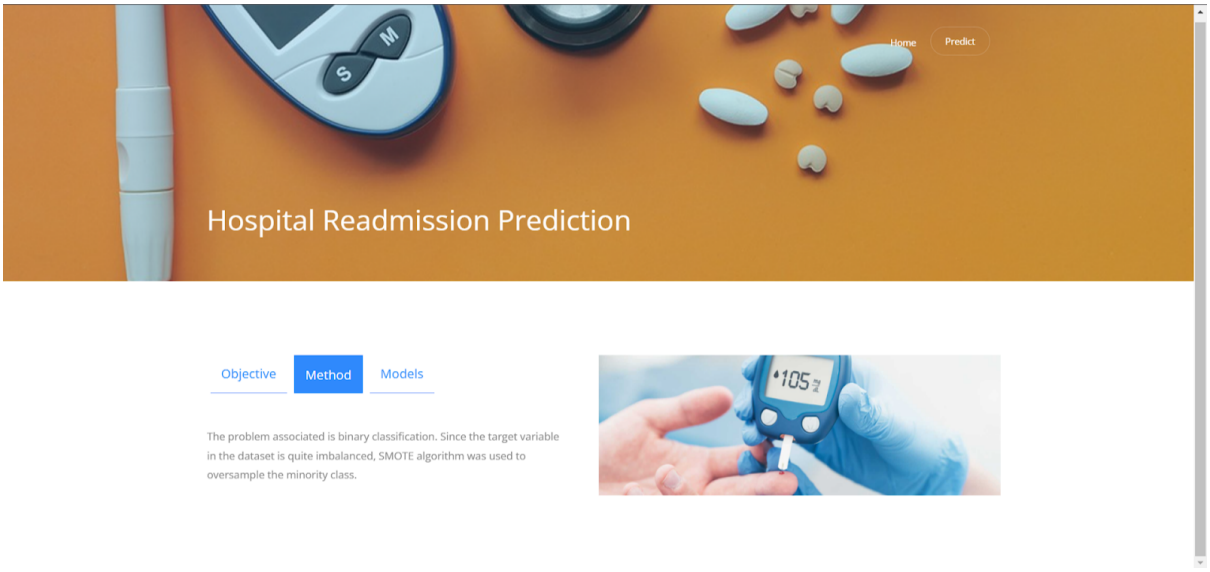
and save them in the templates folder.

Let's see how our home.html page looks like:

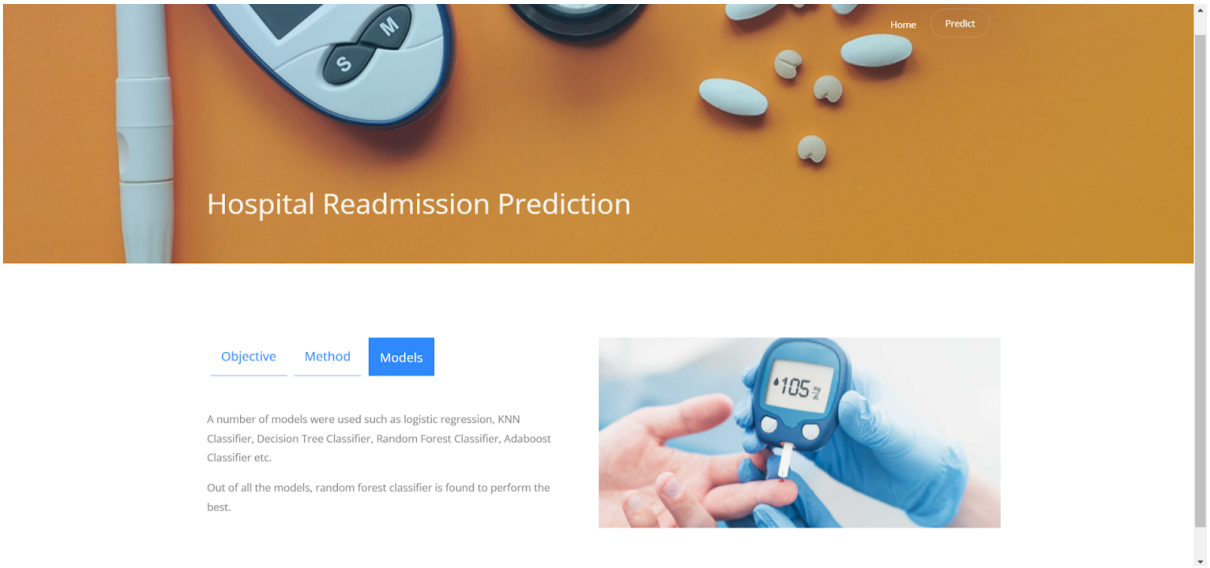
The objective tab describes the main objective of executing this project.



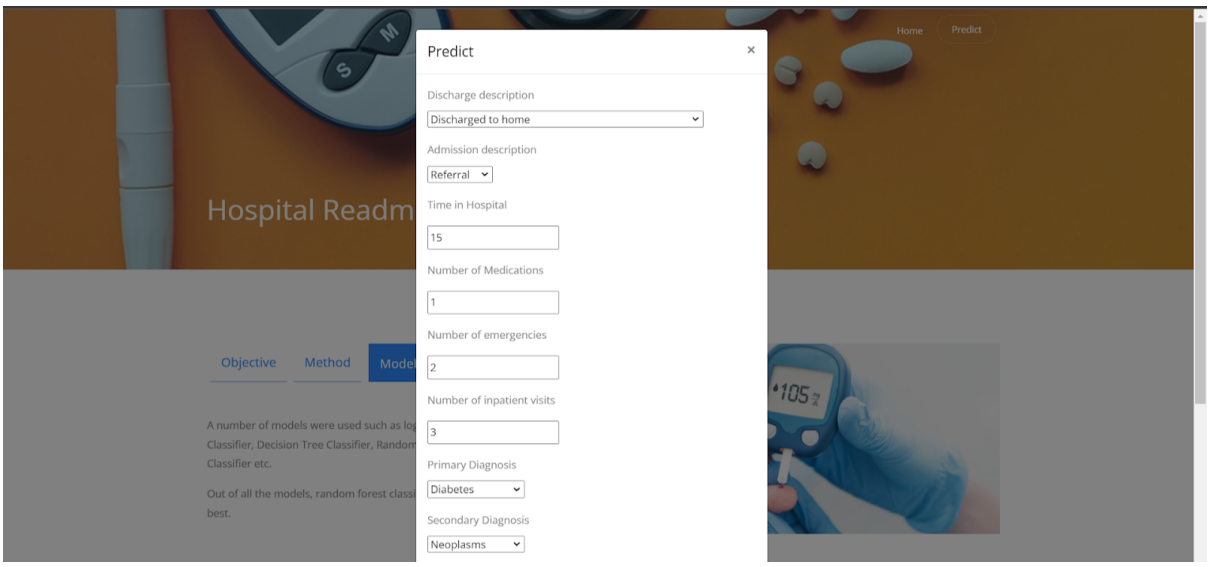
The method tab consists of the method used to solve the problem statement.

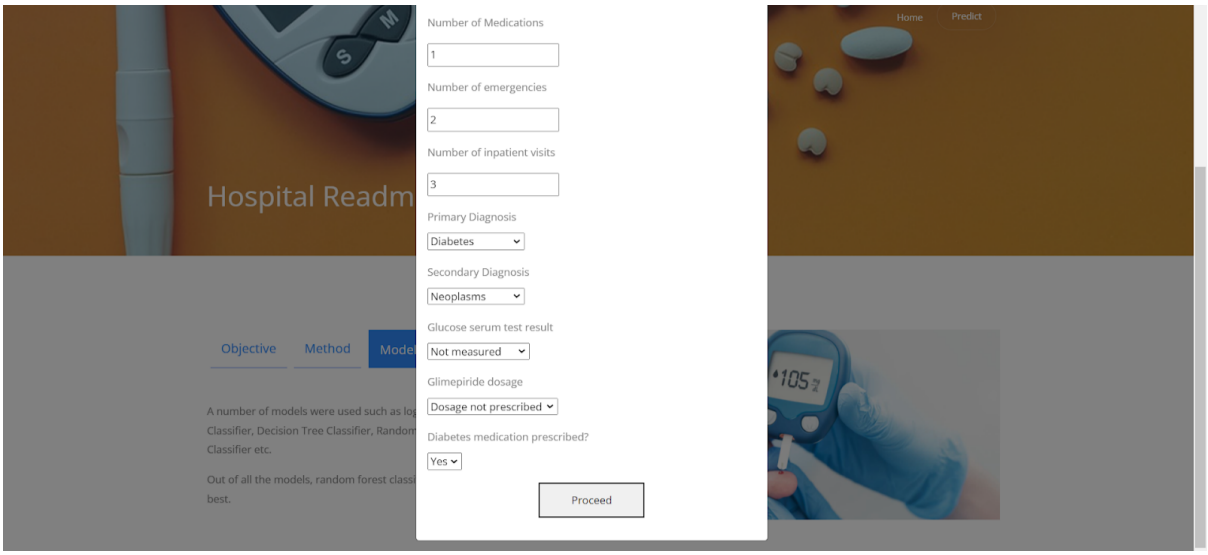


The models tab contains names of various models tested for the project.



Let us look how our index.html page looks like:





Now when you click on proceed button you will get redirected to output.html

Build Python code:

Import the required libraries and load model and ct

```
from flask import Flask, render_template, request
import pickle, joblib
import pandas as pd

app = Flask(__name__)

model = pickle.load(open("model.pkl","rb"))
ct = joblib.load('feature_values')
```

Render home.html and index.html pages:

```
@app.route('/')
def home():
    return render_template("home.html")

@app.route('/pred')
def predict():
    return render_template("index.html")
```

The values entered in can be retrieved using the POST Method.

Retrieves the value from UI:

```
@app.route('/out', methods =["POST"])
def output():
    discharge_disposition_id = request.form["discharge_disposition_id"]
    admission_source_id = request.form["admission_source_id"]
    time_in_hospital = request.form["time_in_hospital"]
    num_medications = request.form["num_medications"]
    number_emergency = request.form["number_emergency"]
    number_inpatient = request.form["number_inpatient"]
    diag_1 = request.form["diag_1"]
    diag_2 = request.form["diag_2"]
    max_glu_serum = request.form["max_glu_serum"]
    glimepiride = request.form["glimepiride"]
    diabetesMed = request.form["diabetesMed"]

    if max_glu_serum == '>200' or max_glu_serum=='>300':
        max_glu_serum=1
    elif max_glu_serum=='Norm':
        max_glu_serum=0
    else:
        max_glu_serum=-99

    if glimepiride == 'No':
        glimepiride = 0
    else:
        glimepiride=1



    data = [[discharge_disposition_id,admission_source_id,time_in_hospital,
             num_medications, number_emergency, number_inpatient,diag_1, diag_2,
             max_glu_serum, glimepiride, diabetesMed]]


    feature_cols = ['discharge_disposition_id', 'admission_source_id', 'time_in_hospital',
                    'num_medications','number_emergency', 'number_inpatient',
                    'diag_1', 'diag_2','max_glu_serum', 'glimepiride', 'diabetesMed']

    pred = model.predict(ct.transform(pd.DataFrame(data,columns=feature_cols)))
    pred = pred[0]
    if pred:
        return render_template("output.html",y="This patient will be readmitted ")
    else:
        return render_template("output.html",y="This patient will not be readmitted")

if __name__ == '__main__':
    app.run(debug = True)
```

Here we are routing our app to output() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the output.html page earlier.

Main Function:

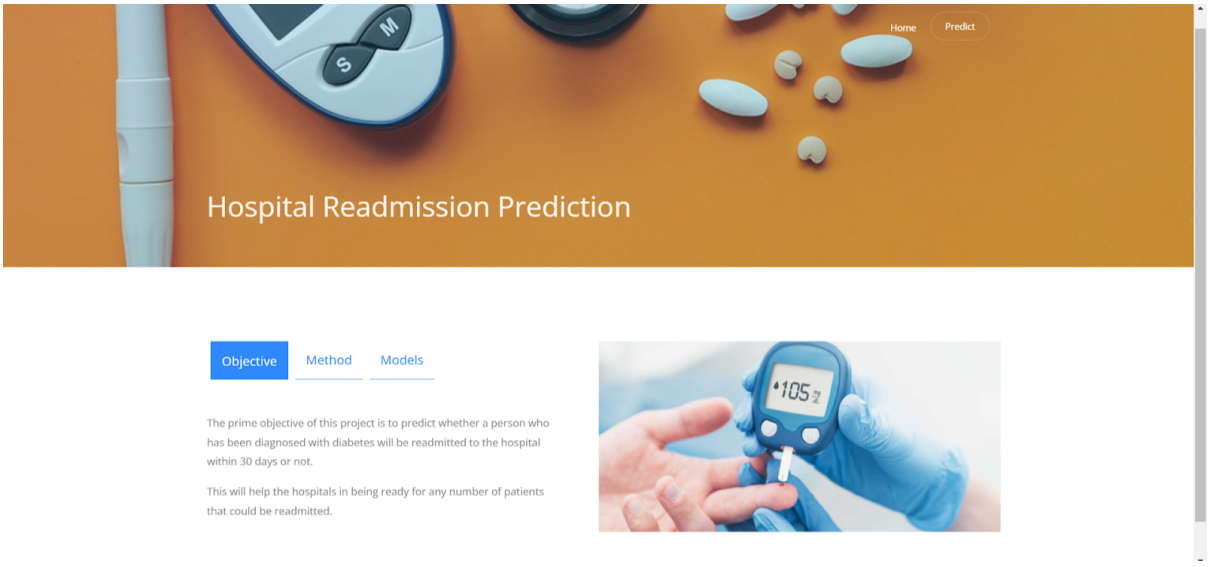
```
if __name__ == '__main__':
    app.run(debug = True)
```

Run the application:

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the proceed button, enter the inputs, click on the predict button, and see the result/prediction on the web.

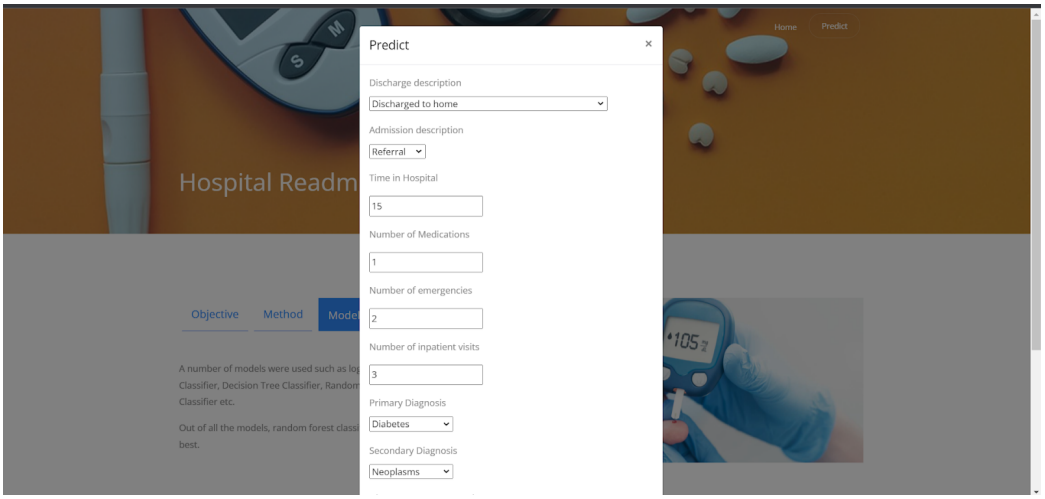
```
(env2) D:\SB_Projects\Hospital Readmission Prediction\flask>python app.py
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 843-846-462
```

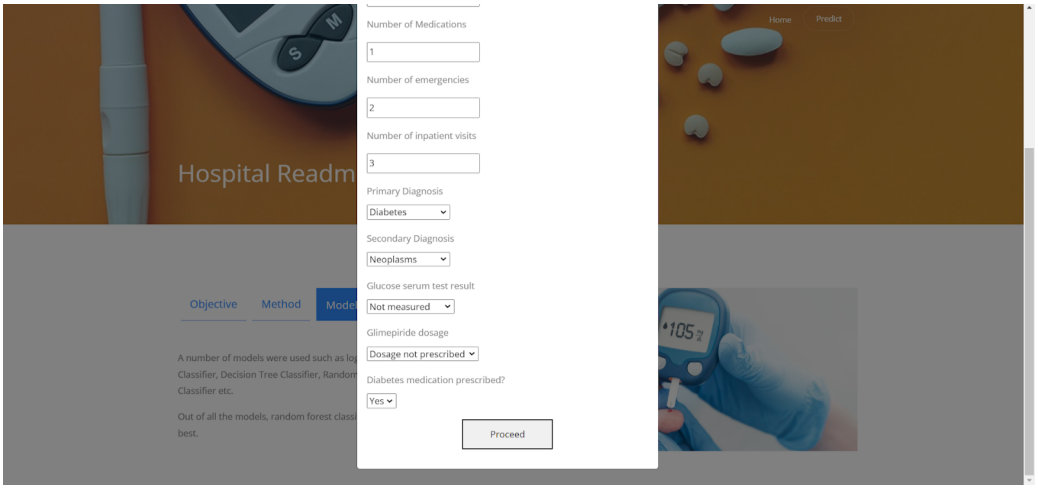
localhost: 5000 will redirect us to the below home page:



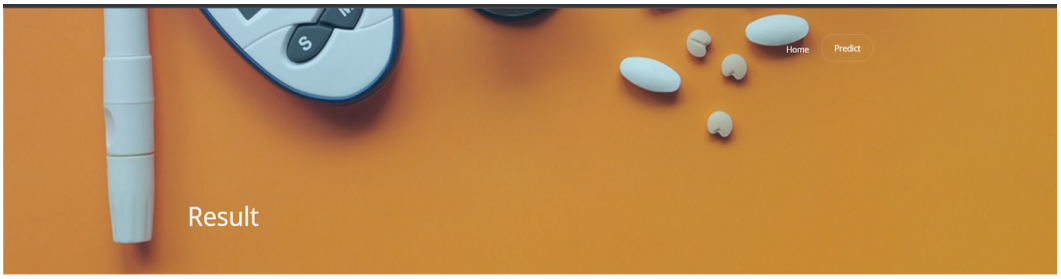
To make the predictions, the user has to click on the predict button at the top right corner. After clicking, it will display a popup window that contains a form to enter values for making prediction. The form looks like below:

Input 1:



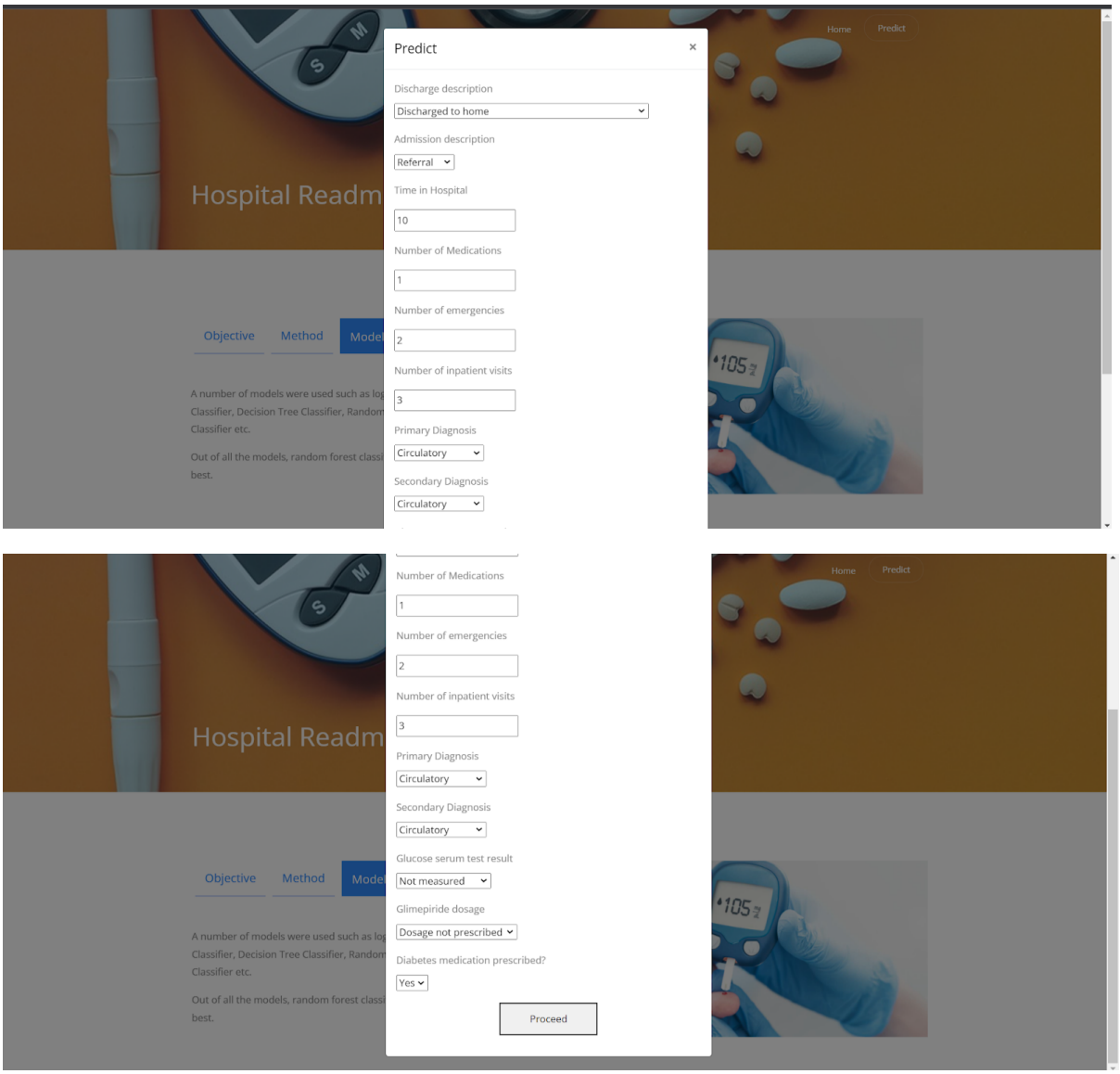


After clicking on Proceed, the result will be displayed in a different page:

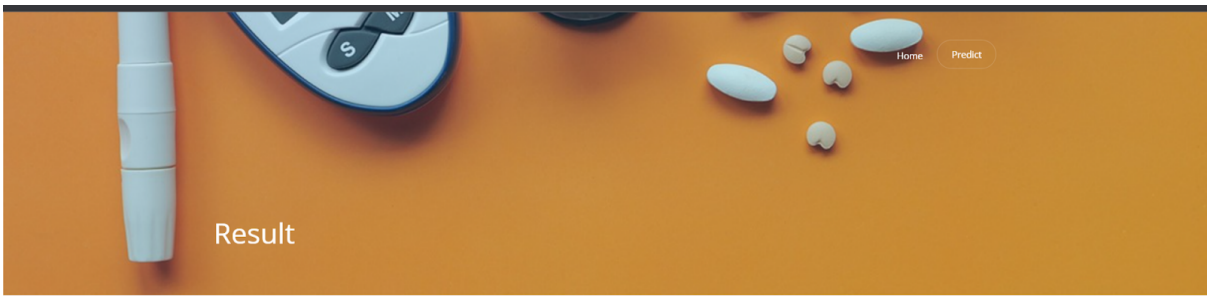


This patient will be readmitted

Input 2:



Output 2:



This patient will not be readmitted