Hospital Readmission Prediction Using ML

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, it 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.

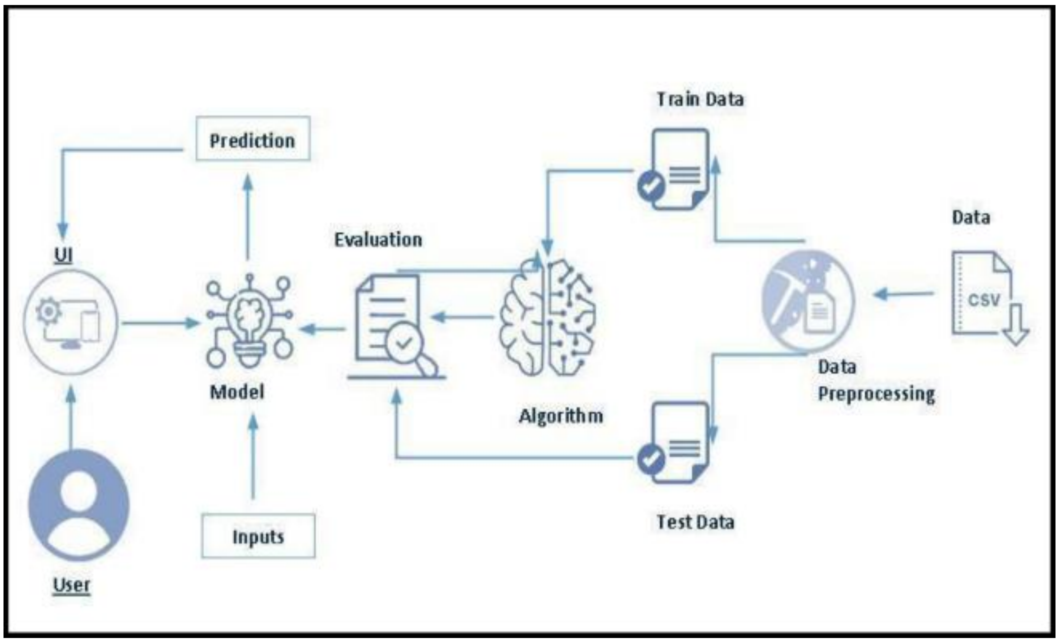
One patient population that is at increased risk of hospitalisation and readmission is that of diabetes. Diabetes is a medical condition that affects approximately 1 in 10 patients in the United States. So in this project, we will be focusing on hospital readmission prediction for patients who are having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

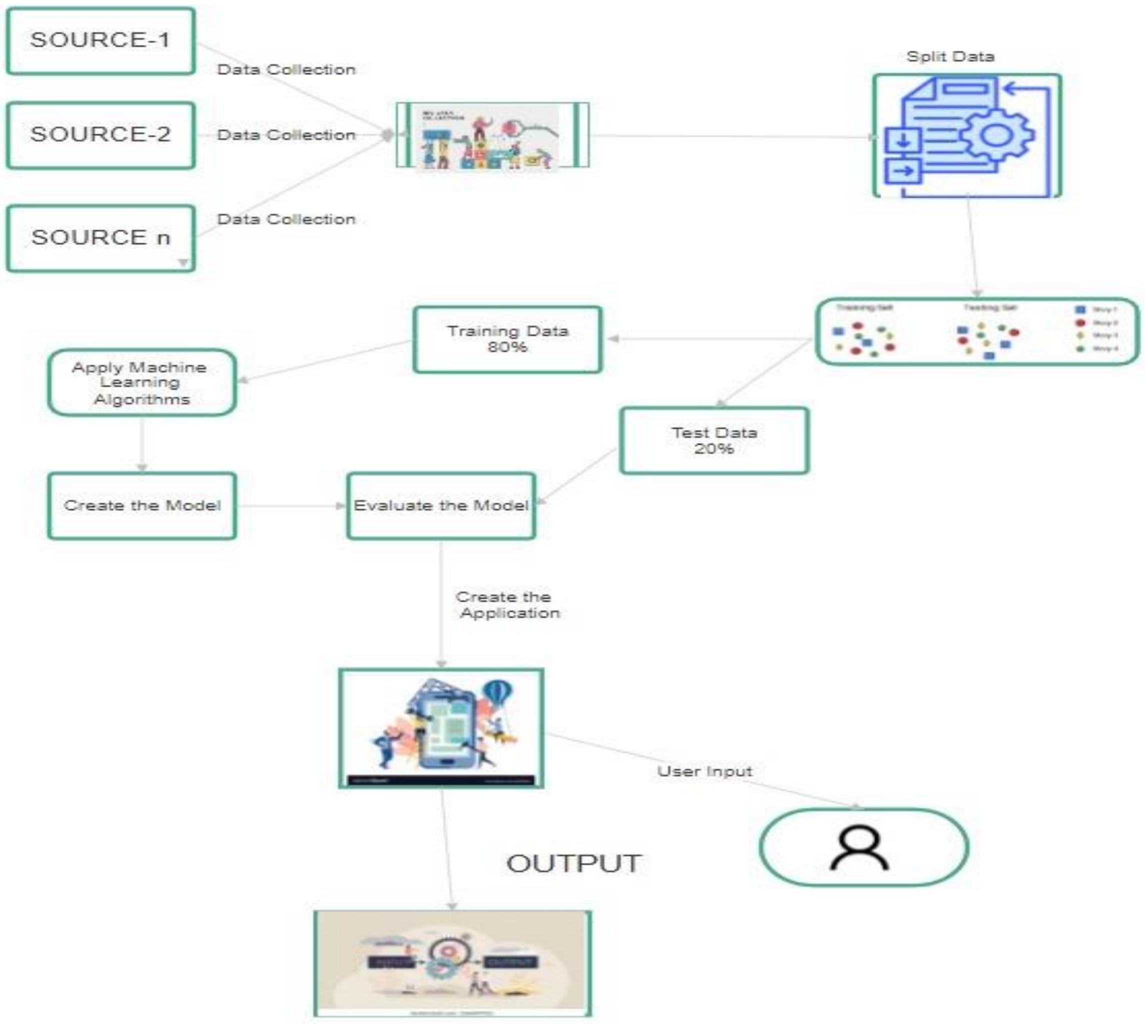
The main purpose of this project is to predict whether a person who is suffering with diabetes and consulting a specific hospital will be readmitted or not, based on multiple factors.

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:



Flow chart of model:



Pre requisites:

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

- Anaconda navigator:
    - o By referring some Youtube videos we downloaded anaconda navigator
- Python packages:
    - o Open anaconda prompt as administrator
    - o Type "pip install numpy" and click enter.
    - o Type "pip install pandas" and click enter.
    - o Type "pip install scikit-learn" and click enter.
    - o Type "pip install matplotlib" and click enter.
    - o Type "pip install scipy" and click enter.
    - o Type "pip install pickle-mixin" and click enter.
    - o Type "pip install seaborn" and click enter.
    - o Type "pip install Flask" and click enter.

Prior Knowledge:

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

- ML Concepts
    - o Supervised learning: https://www.javatpoint.com/supervised-machine-learning
    - o Unsupervised learning:https://www.javatpoint.com/unsupervised-machine-learning
    - o Regression and classification
        - ▪ Logistic regression:https://www.javatpoint.com/logistic-regression-in-machine-learning
        - ▪ Decision tree:https://www.javatpoint.com/machine-learning-decision-tree-classification
        - ▪ Random forest:https://www.javatpoint.com/machine-learning-random-forest-algorithm
        - ▪ KNN:https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

▪ AdaBoost:https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-begineers/

▪ Gradient Boost:https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-begineers/

▪ Evaluation metrics:https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evluation-metrics/

- Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Know how to deal with imbalanced target variables.
- Have knowledge on pre-processing the data/transformation techniques and some visualisation 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 analysed by the model which is integrated.
- The predictions made by the model is showcased on the UI

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

- Data collection
    - o Download the dataset
- Data pre-processing
    - o Handling null values and removing unnecessary columns

- Visualising and analysing data
    - o Univariate analysis
    - o Bivariate analysis
    - o Descriptive analysis
- Model building
    - o Handling categorical values
    - o Dividing data into train and test sets
    - o Sampling the data
    - o Dividing train data into train and validation sets
    - o Comparing performance of various models
    - o Feature Selection
    - o Repeat process from dividing data into train and test sets
    - o Evaluating final model performance
    - o Save the final model
- Application Building
    - o Building HTML pages
    - o Build python code

Milestone 1: Data Collection

Activity 1: 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:

Link:

https://archive.ics.uci.edu/ml/datasets/diabetes+130-us+hospitals+for+years+1999-2008

Load the dataset using read_csv() function:

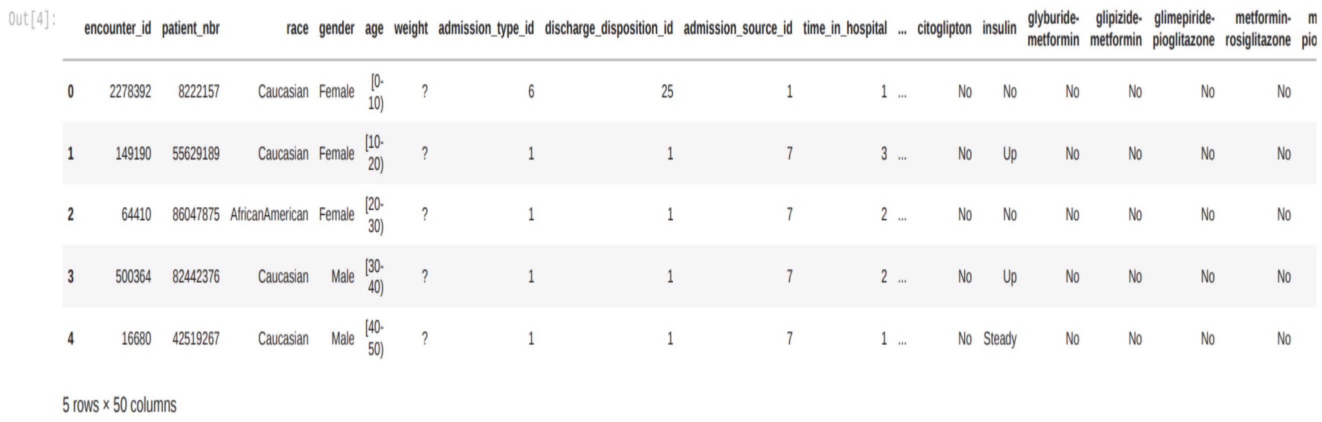
```python
import numpy as np
import pandas as pd
import os
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import StandardScaler

from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

```python
df = pd.read_csv("C:/Users/Dell/Downloads/diabetic_data.csv")
df.head()
```

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.

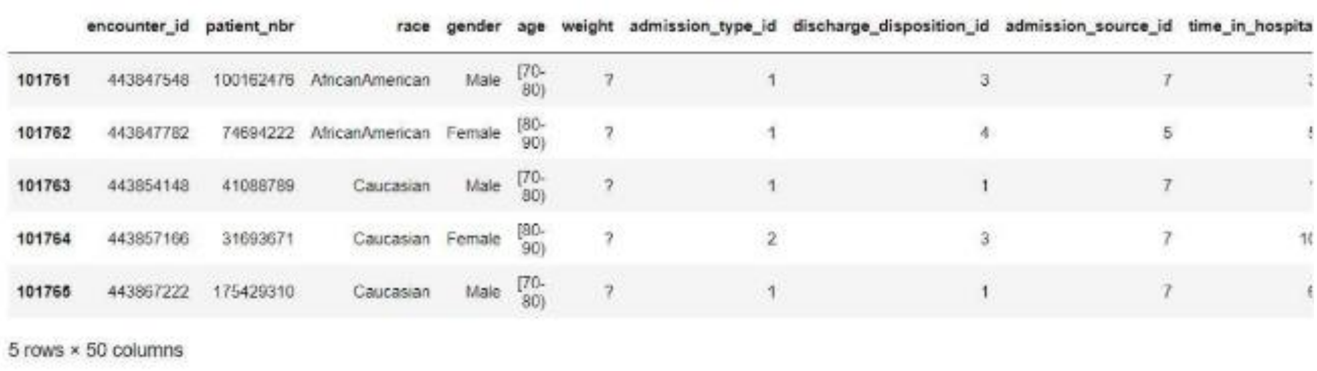


Out[4]:

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	...	citoglipton	insulin	glyburide-metformin	glipizide-metformin	glimepiride-pioglitazone	metformin-rosiglitazone	m pio
0	2278392	8222157	Caucasian	Female	[0-10)	?	6	25	1	1	...	No	No	No	No	No	No	
1	149190	55629189	Caucasian	Female	[10-20)	?	1	1	7	3	...	No	Up	No	No	No	No	
2	64410	86047875	AfricanAmerican	Female	[20-30)	?	1	1	7	2	...	No	No	No	No	No	No	
3	500364	82442376	Caucasian	Male	[30-40)	?	1	1	7	2	...	No	Up	No	No	No	No	
4	16680	42519267	Caucasian	Male	[40-50)	?	1	1	7	1	...	No	Steady	No	No	No	No	

5 rows × 50 columns

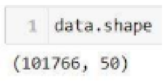
```python
data.tail()
```



	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	[illegible]
101762	443847782	74694222	AfricanAmerican	Female	[80-90)	?	1	4	5	[illegible]
101763	443854148	41088789	Caucasian	Male	[70-80)	?	1	1	7	[illegible]
101764	443857166	31693671	Caucasian	Female	[80-90)	?	2	3	7	[illegible]
101765	443867222	175429310	Caucasian	Male	[70-80)	?	1	1	7	[illegible]

5 rows × 50 columns

We can use the shape attribute of the data frame to know the shape of our dataset:

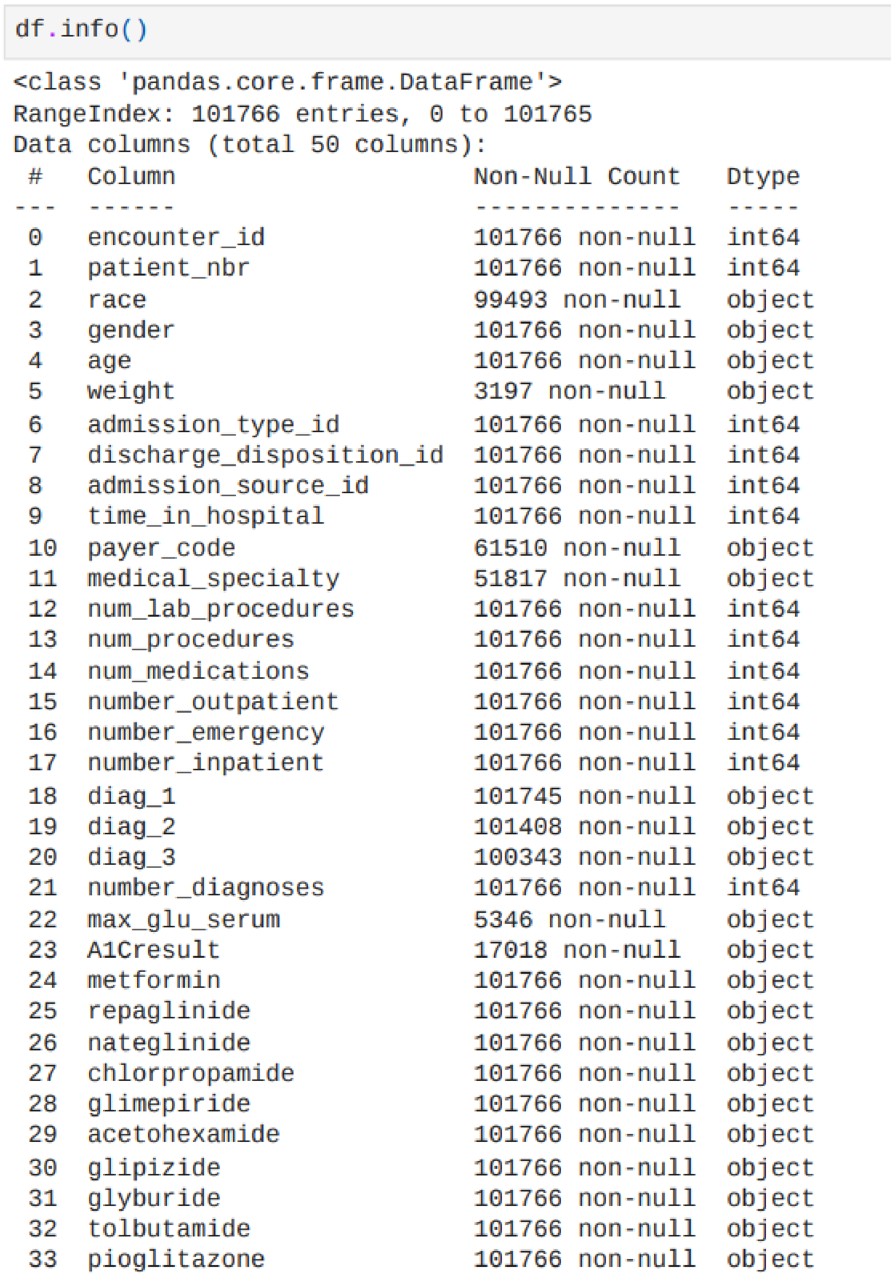


```
data.shape
```

```
(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:



```
df.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                      99493 non-null   object
 3   gender                    101766 non-null  object
 4   age                       101766 non-null  object
 5   weight                    3197 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                61510 non-null   object
 11  medical_specialty         51817 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                    101745 non-null  object
 19  diag_2                    101408 non-null  object
 20  diag_3                    100343 non-null  object
 21  number_diagnoses          101766 non-null  int64
 22  max_glu_serum             5346 non-null    object
 23  A1Cresult                 17018 non-null   object
 24  metformin                 101766 non-null  object
 25  repaglinide               101766 non-null  object
 26  nateglinide               101766 non-null  object
 27  chlorpropamide            101766 non-null  object
 28  glimepiride               101766 non-null  object
 29  acetohexamide             101766 non-null  object
 30  glipizide                 101766 non-null  object
 31  glyburide                 101766 non-null  object
 32  tolbutamide               101766 non-null  object
 33  pioglitazone              101766 non-null  object
 33  pioglitazone              101766 non-null  object
 34  rosiglitazone             101766 non-null  object
 35  acarbose                  101766 non-null  object
 36  miglitol                  101766 non-null  object
 37  troglitazone              101766 non-null  object
 38  tolazamide                101766 non-null  object
 39  examide                   101766 non-null  object
 40  citoglipton               101766 non-null  object
 41  insulin                   101766 non-null  object
 42  glyburide-metformin       101766 non-null  object
 43  glipizide-metformin       101766 non-null  object
 44  glimepiride-pioglitazone  101766 non-null  object
 45  metformin-rosiglitazone   101766 non-null  object
 46  metformin-pioglitazone    101766 non-null  object
 47  change                    101766 non-null  object
 48  diabetesMed               101766 non-null  object
 49  readmitted                101766 non-null  object
dtypes: int64(13), object(37)
memory usage: 38.8+ MB
```

Milestone 2: Data Pre-processing

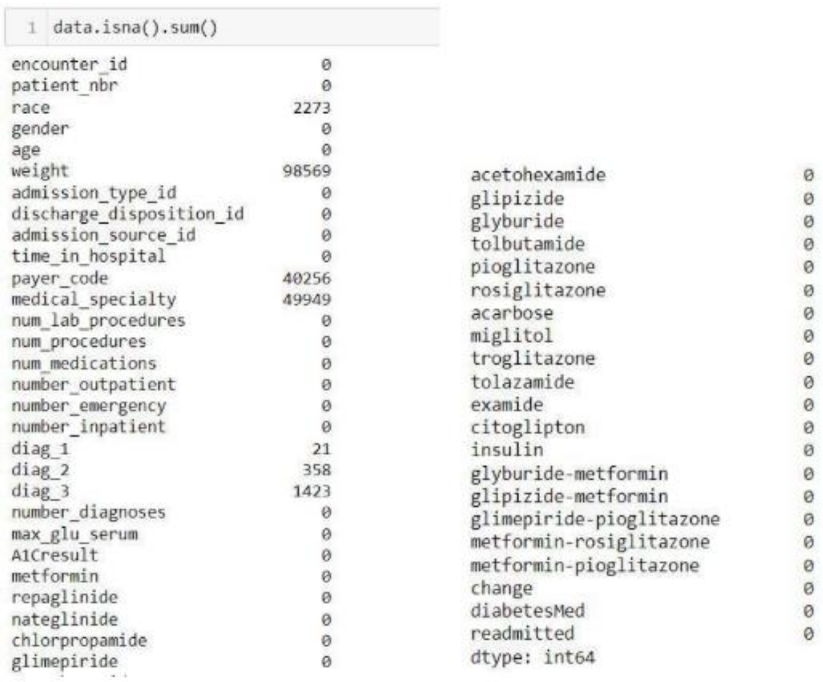
We need to pre-process the collected data before gaining insights and building our model. We need to clean the dataset properly in order to fetch good results. This activity includes handling null values and removing unnecessary columns.

Activity 1: 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.

```
df = df.replace("?",np.nan)
len(df.select_dtypes('O').columns)
```

```
37
```

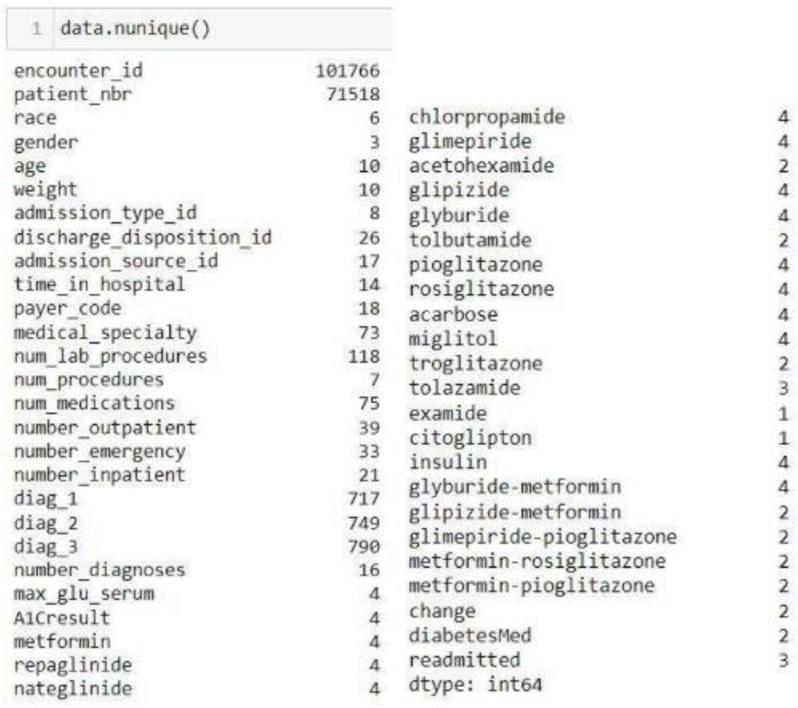


```
data.isna().sum()
```

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

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.

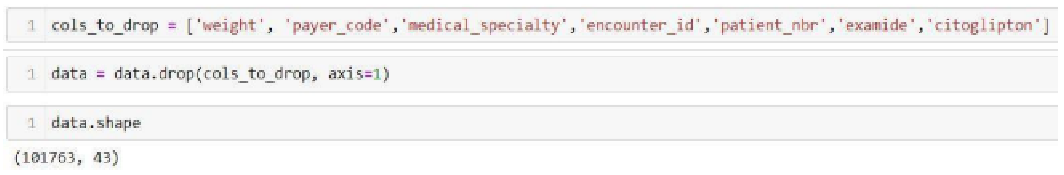


```
data.nunique()
```

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

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.



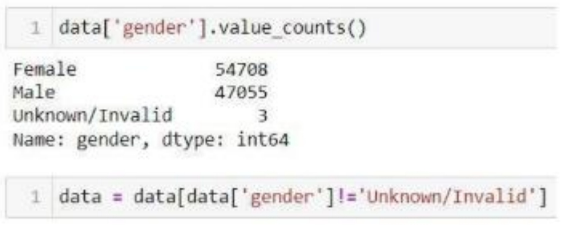
```
cols_to_drop = ['weight', 'payer_code','medical_specialty','encounter_id','patient_nbr','examide','citoglipton']
```

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

```
data.shape
```

```
(101763, 43)
```

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

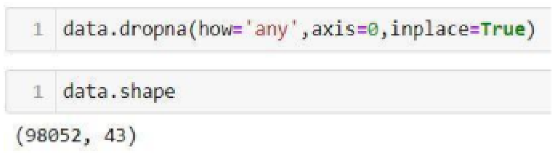


```
data['gender'].value_counts()
```

```
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.



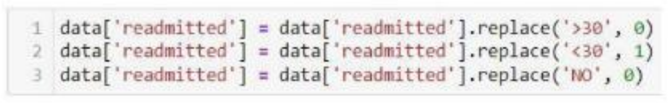
```
data.dropna(how='any',axis=0,inplace=True)
```

```
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.

Milestone 3: Data analysis and visualisation

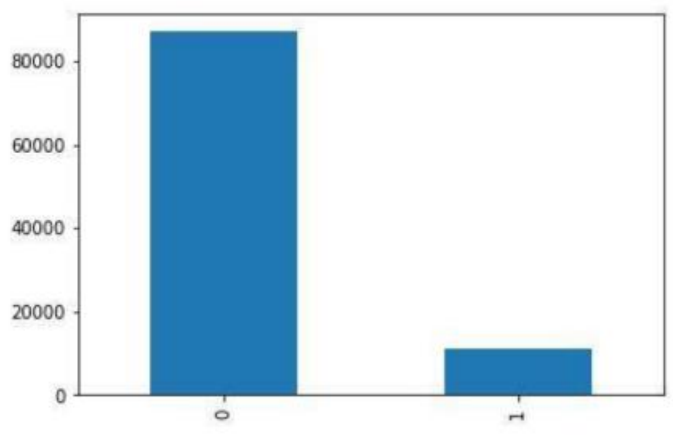
Activity 1: Univariate analysis

In simple words, univariate analysis is understanding the data with a single feature.

Let us first plot the values of our target column – readmitted

```
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.

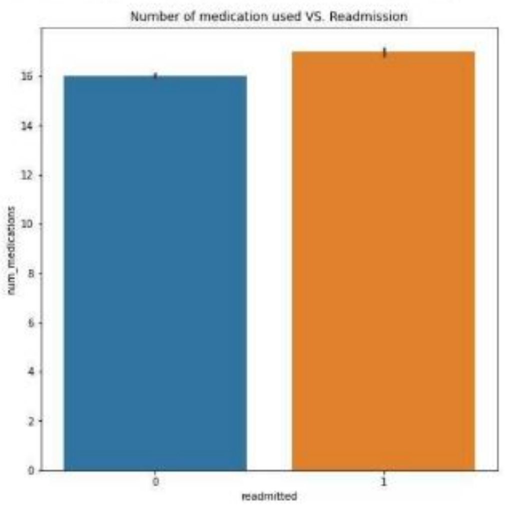
Activity 2: Bivariate analysis

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")
```

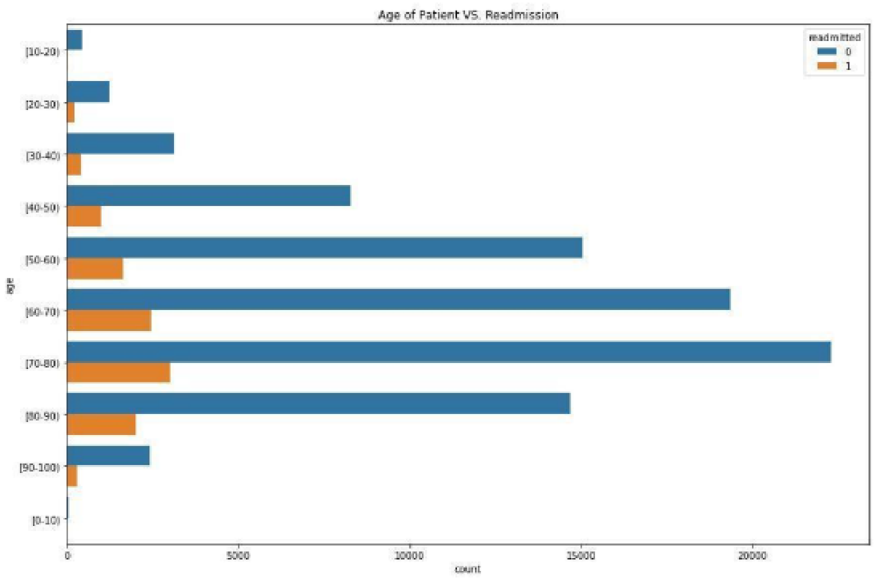
```
Text(0.5, 1.0, 'Number of medication used VS. Readmission')
```



- Age and readmitted

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

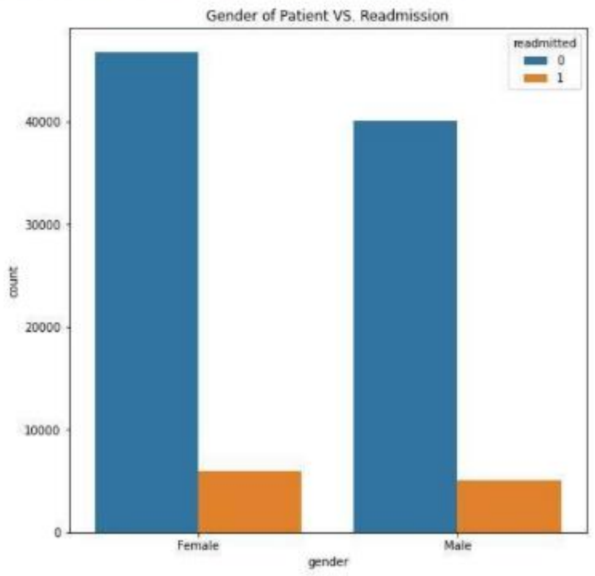
```
Text(0.5, 1.0, 'Age of Patient VS. Readmission')
```



- Gender and readmitted

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

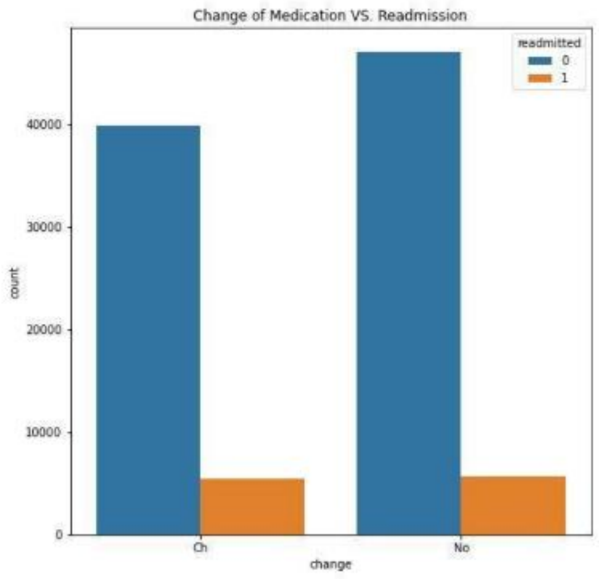
```
Text(0.5, 1.0, 'Gender of Patient VS. Readmission')
```



- Change of medication and readmitted

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

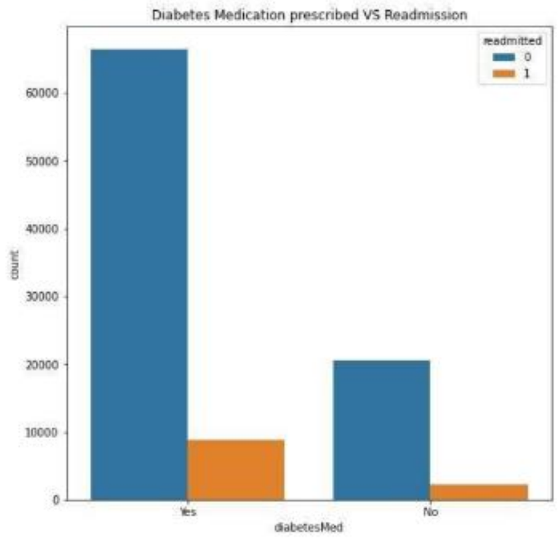
```
Text(0.5, 1.0, 'Change of Medication VS. Readmission')
```



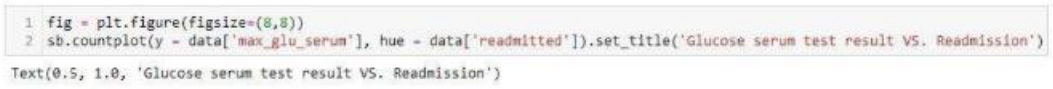
- Diabetes medication prescribed and readmitted

```
fig = plt.figure(figsize=(8,8))
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')
```

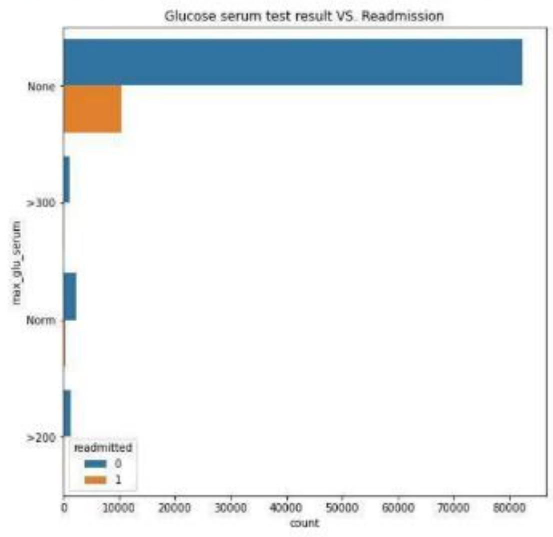


- Glucose serum test and readmitted

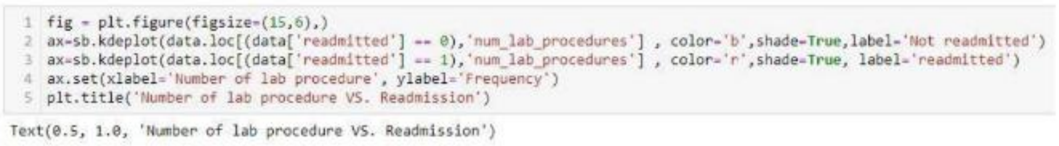


```
fig = plt.figure(figsize=(8,8))
sb.countplot(y = data['max_glu_serum'], hue = data['readmitted']).set_title('Glucose serum test result VS. Readmission')
```

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

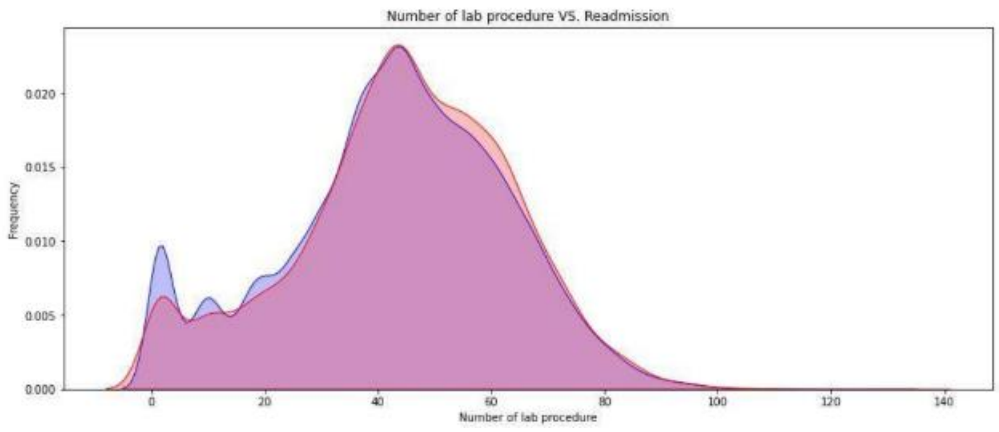


- Number of lab procedures and readmitted

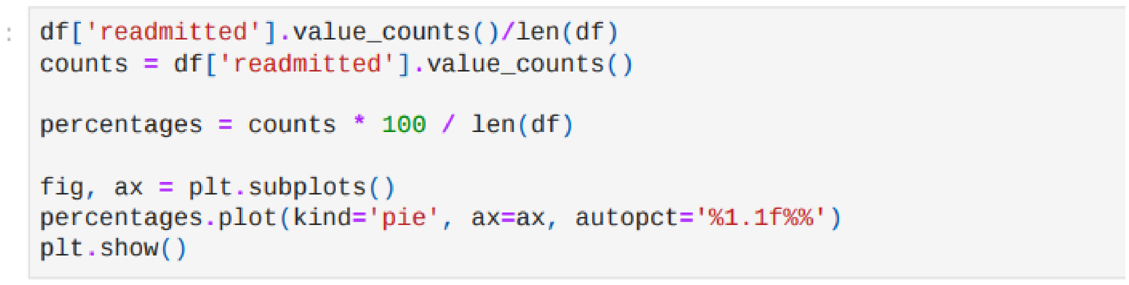


```
fig = plt.figure(figsize=(15,6),)
ax=sb.kdeplot(data.loc[(data['readmitted'] == 0),'num_lab_procedures'] , color='b',shade=True,label='Not readmitted')
ax=sb.kdeplot(data.loc[(data['readmitted'] == 1),'num_lab_procedures'] , color='r',shade=True, label='readmitted')
ax.set(xlabel='Number of lab procedure',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

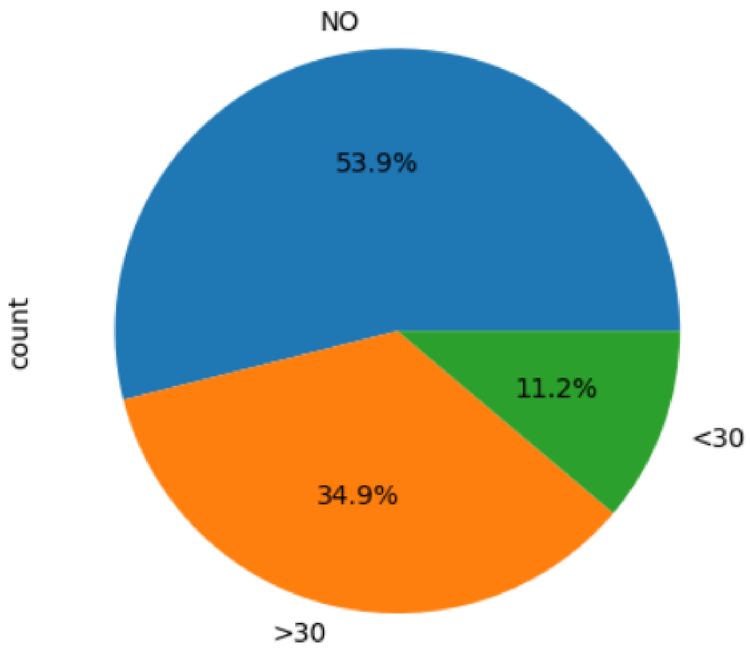
```
Text(0.5, 1.0, 'Number of lab procedure VS. Readmission')
```



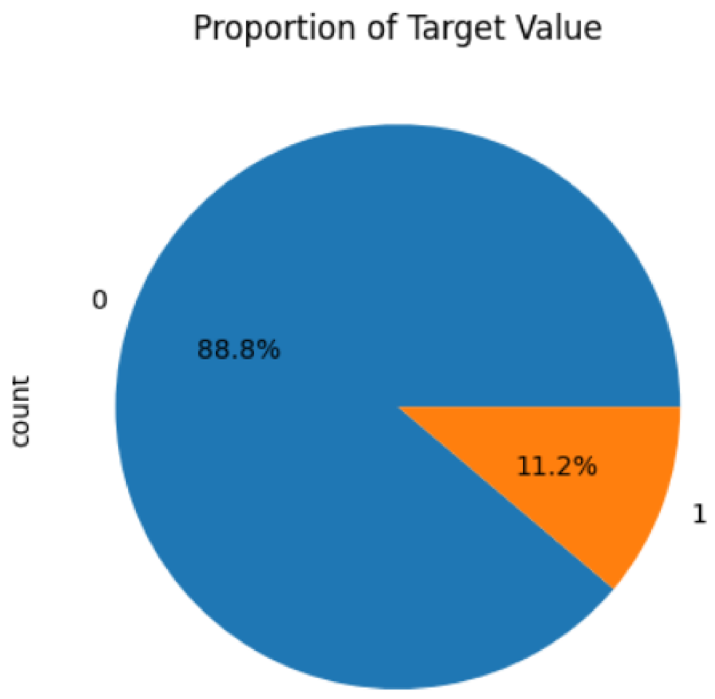
- Readmitted count



```
df['readmitted'].value_counts()/len(df)
counts = df['readmitted'].value_counts()

percentages = counts * 100 / len(df)

fig, ax = plt.subplots()
percentages.plot(kind='pie', ax=ax, autopct='%1.1f%%')
plt.show()
```



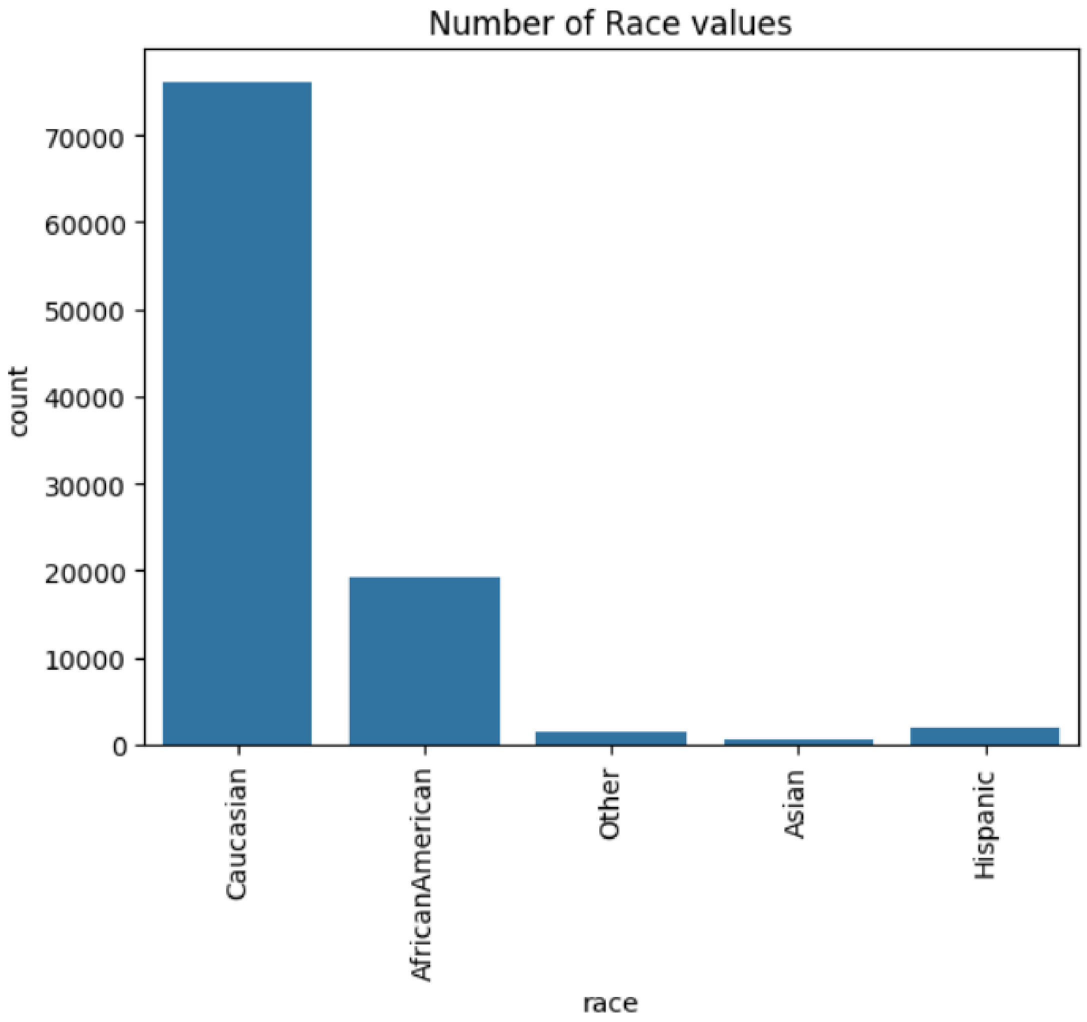
- Piechart showing proportion of target value



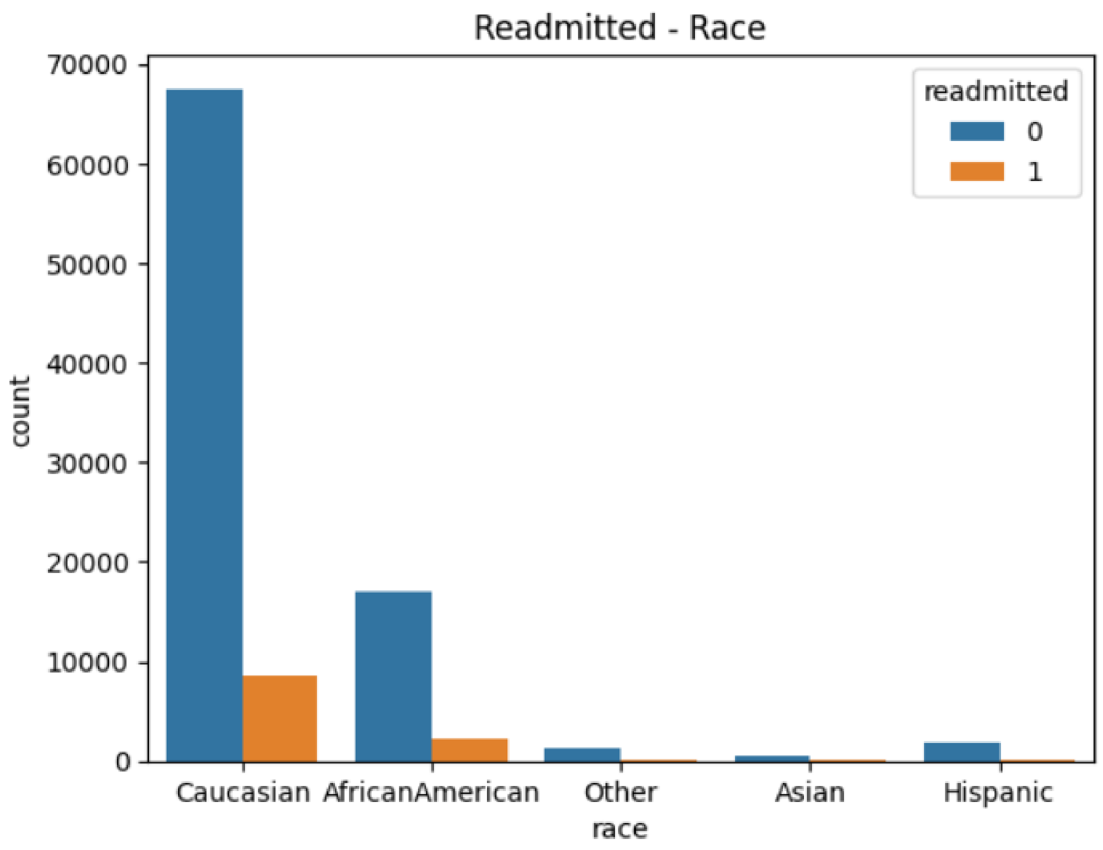
- Number of race values

```
print("Proportion of Race")
print(df.race.value_counts(normalize = True)*100)


sns.countplot(x=df.race, data = df)
plt.xticks(rotation=90)
plt.title("Number of Race values")
plt.show()
```

```
Proportion of Race
race
Caucasian          76.486788
AfricanAmerican    19.307891
Hispanic            2.047380
Other               1.513674
Asian               0.644266
Name: proportion, dtype: float64
```

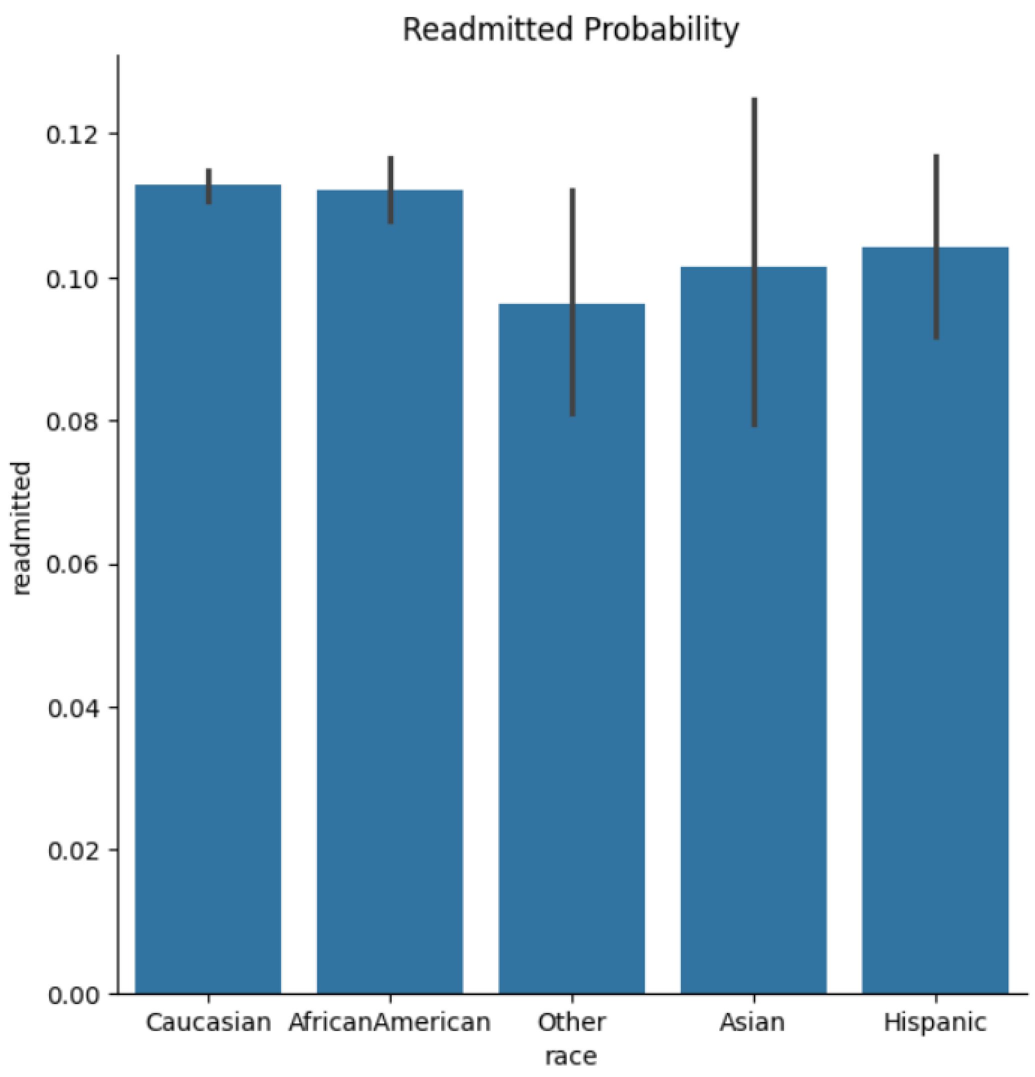


- Readmitted Race



- Readmitted probability

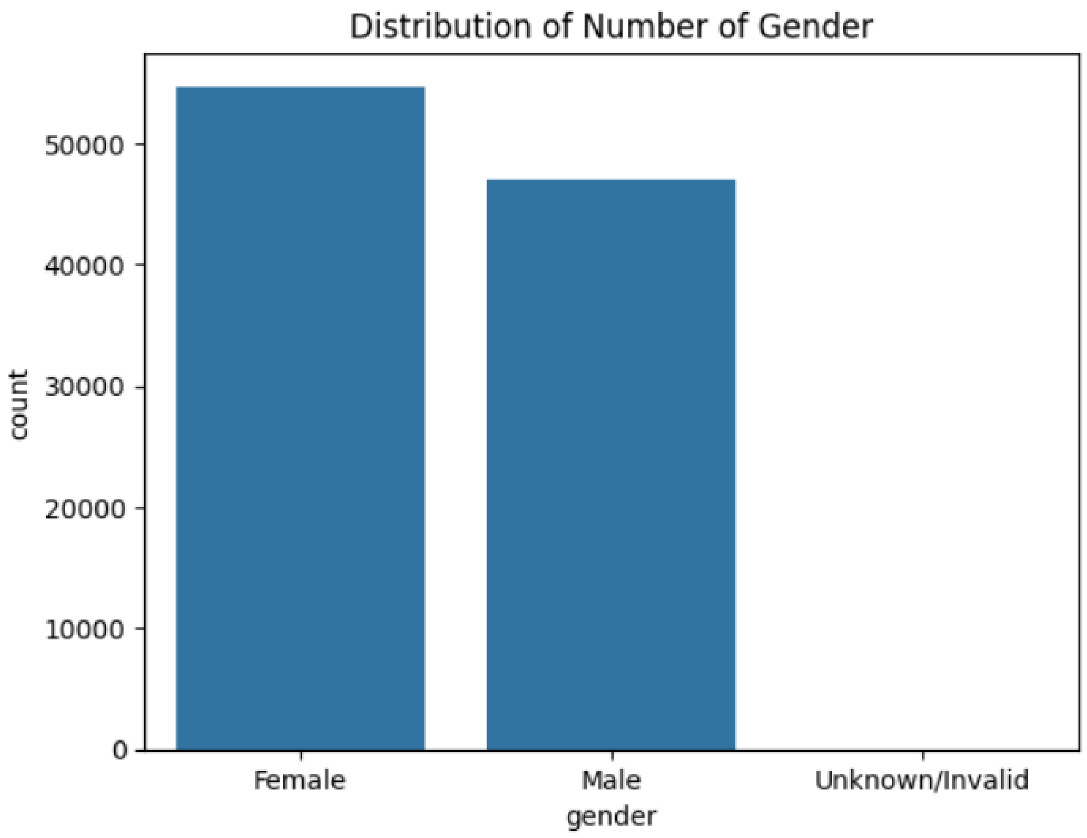
```
[15]: sns.catplot(x = "race", y = "readmitted",data = df, kind = "bar", height= 6)
      plt.title("Readmitted Probability")
      plt.show()
```



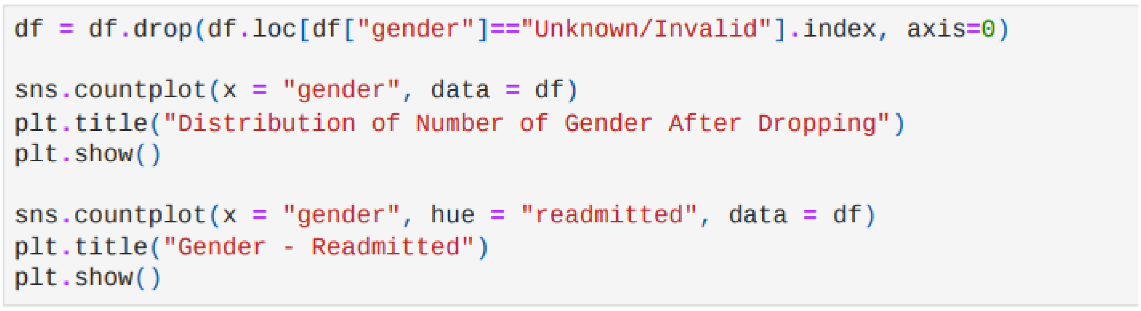
- Distribution of Number of Gender

```
6]: print("Proportions of Race Value")
    print(df.gender.value_counts(normalize = True))

    sns.countplot(x = "gender", data = df)
    plt.title("Distribution of Number of Gender")
    plt.show()
```

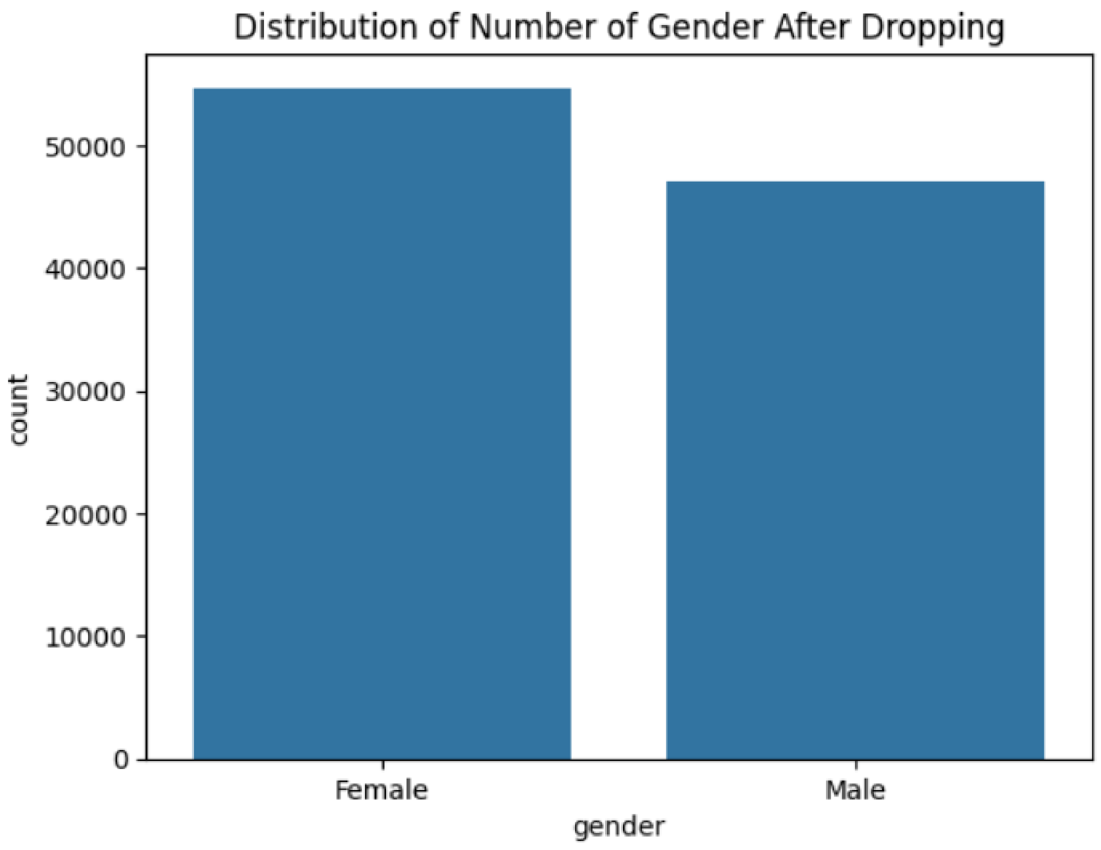
```
Proportions of Race Value
gender
Female             0.537586
Male               0.462384
Unknown/Invalid    0.000029
Name: proportion, dtype: float64
```



- Distribution of number of gender after dropping

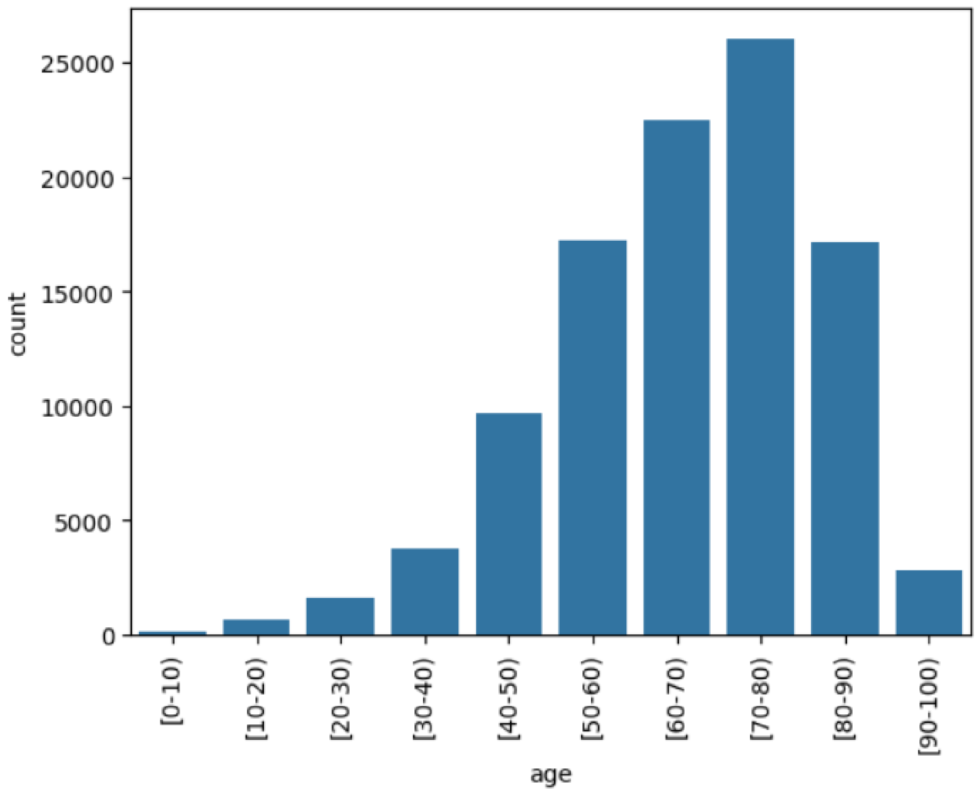


```
df = df.drop(df.loc[df["gender"]=="Unknown/Invalid"].index, axis=0)

sns.countplot(x = "gender", data = df)
plt.title("Distribution of Number of Gender After Dropping")
plt.show()

sns.countplot(x = "gender", hue = "readmitted", data = df)
plt.title("Gender - Readmitted")
plt.show()
```



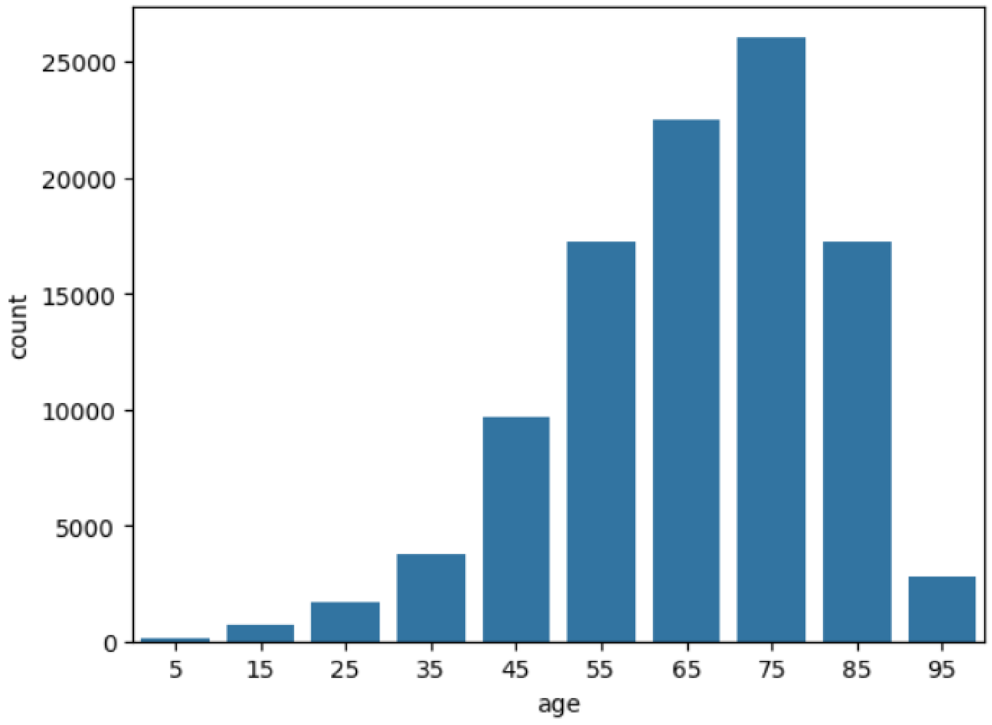
- Age

In [18]:
```
sns.countplot(x="age", data = df)
plt.xticks(rotation = 90)
plt.show()
```



- Replace by using average

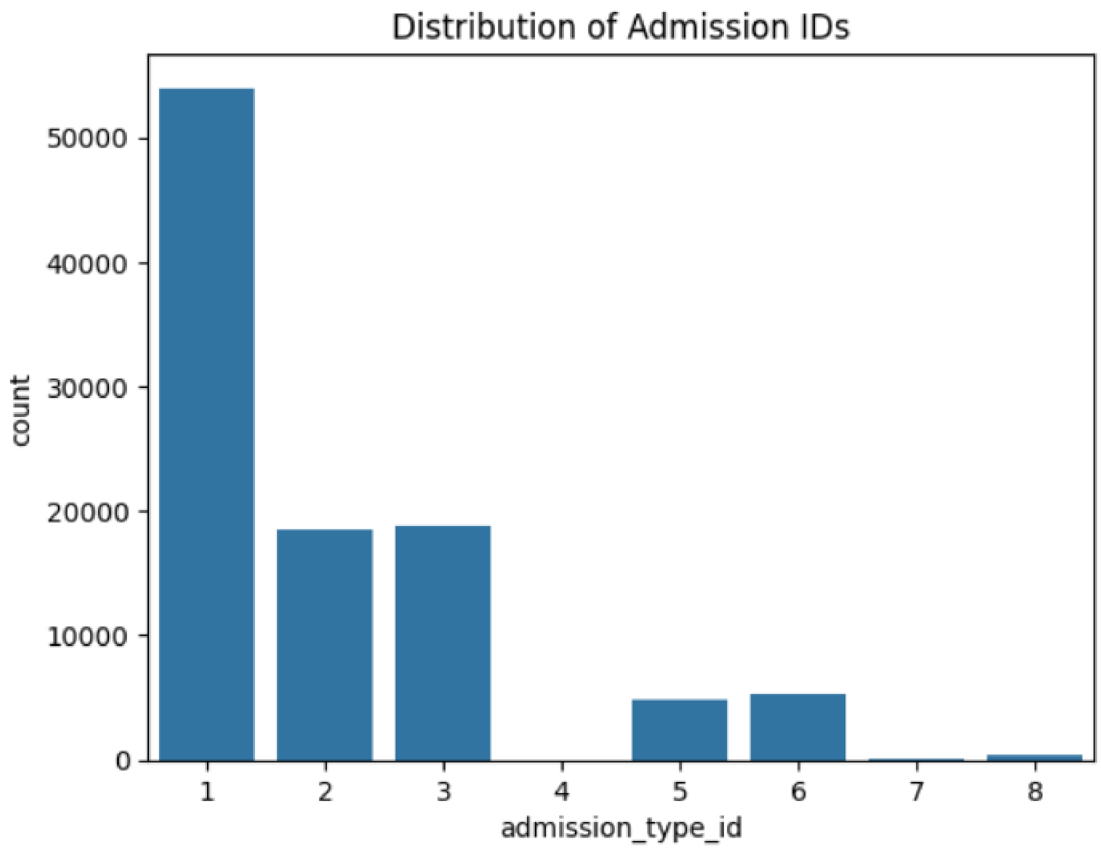
```
In [19]: df.age = df.age.replace({"[70-80)":75,
                                  "[60-70)":65,
                                  "[50-60)":55,
                                  "[80-90)":85,
                                  "[40-50)":45,
                                  "[30-40)":35,
                                  "[90-100)":95,
                                  "[20-30)":25,
                                  "[10-20)":15,
                                  "[0-10)":5})

         sns.countplot(x="age", data = df)
         plt.show()
```



- Distribution of Admission IDs

```
In [20]: print("Distribution of ID's")
         print(df.admission_type_id.value_counts())

         sns.countplot(x = "admission_type_id", data = df)
         plt.title("Distribution of Admission IDs")
         plt.show()
```

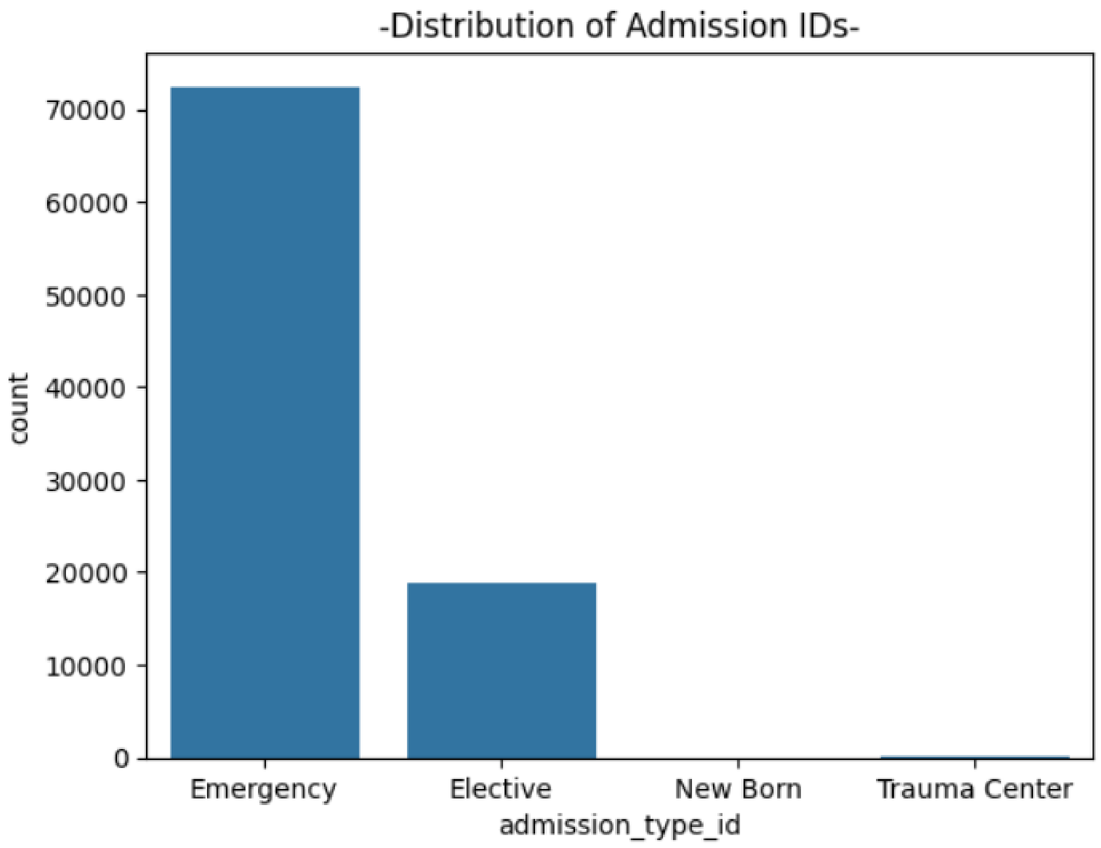
```
Distribution of ID's
admission_type_id
1    53988
3    18868
2    18480
6     5291
5     4785
8      320
7       21
4       10
Name: count, dtype: int64
```



- Distribution of Admission IDs

```
In [21]: mapped = {1.0:"Emergency",
                   2.0:"Emergency",
                   3.0:"Elective",
                   4.0:"New Born",
                   5.0:np.nan,
                   6.0:np.nan,
                   7.0:"Trauma Center",
                   8.0:np.nan}

         df.admission_type_id = df.admission_type_id.replace(mapped)

         print("-Distribution of ID's-")
         print(df.admission_type_id.value_counts())

         sns.countplot(x = "admission_type_id", data = df)
         plt.title("-Distribution of Admission IDs-")
         plt.show()
```

```
-Distribution of ID's-
admission_type_id
Emergency         72468
Elective          18868
Trauma Center        21
New Born             10
Name: count, dtype: int64
```



- Making necessary changes

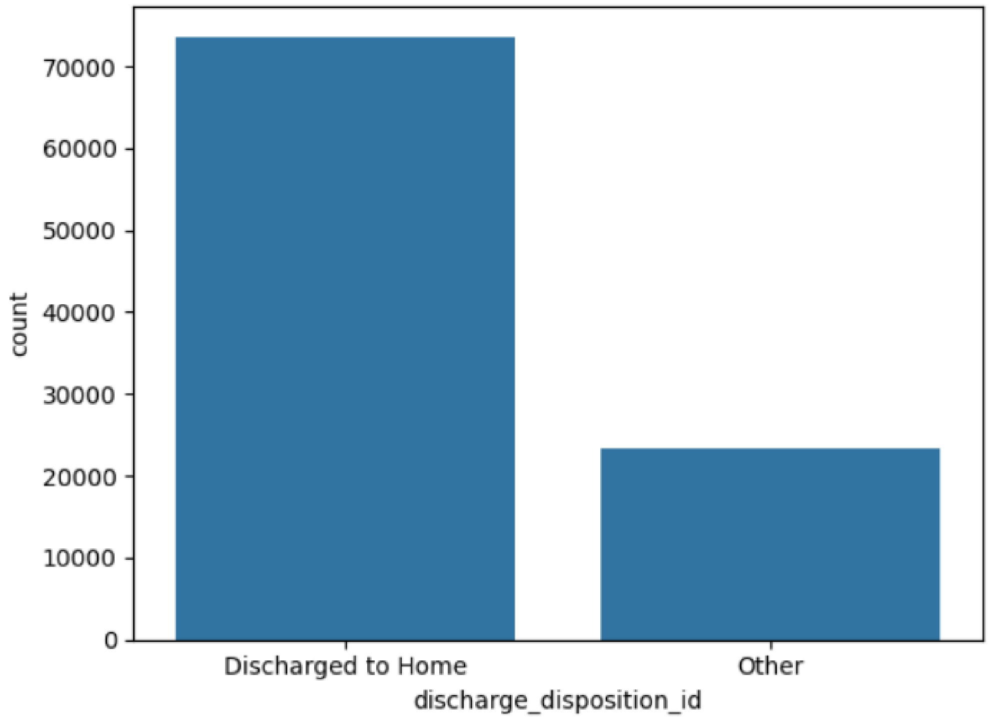
```
[ ]: mapped_discharge = {1:"Discharged to Home",
                         6:"Discharged to Home",
                         8:"Discharged to Home",
                         13:"Discharged to Home",
                         19:"Discharged to Home",
                         18:np.nan,25:np.nan,26:np.nan,
                         2:"Other",3:"Other",4:"Other",
                         5:"Other",7:"Other",9:"Other",
                         10:"Other",11:"Other",12:"Other",
                         14:"Other",15:"Other",16:"Other",
                         17:"Other",20:"Other",21:"Other",
                         22:"Other",23:"Other",24:"Other",
                         27:"Other",28:"Other",29:"Other",30:"Other"}

     df["discharge_disposition_id"] = df["discharge_disposition_id"].replace(mapped_discharge)
```

```
[ ]: df['discharge_disposition_id'].unique()
```

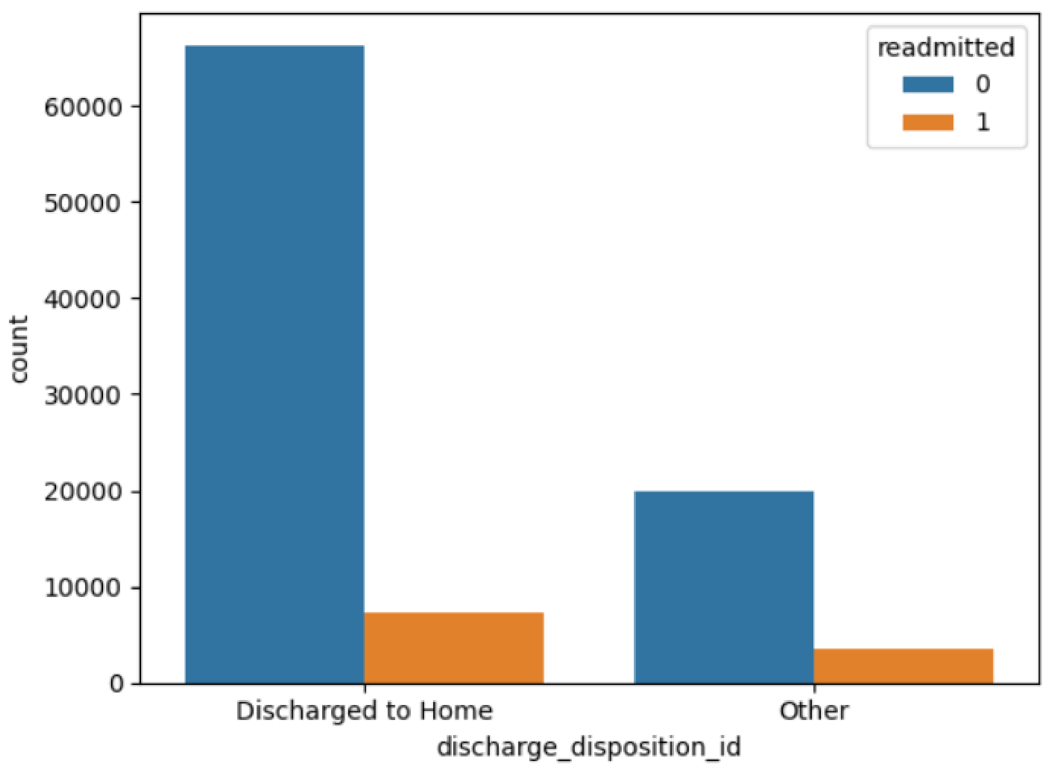
```
[ ]: array([nan, 'Discharged to Home', 'Other'], dtype=object)
```

```
4]: print("Proportions of ID's")
    print(df.discharge_disposition_id.value_counts())

    sns.countplot(x ="discharge_disposition_id", data = df)
    plt.show()

    sns.countplot(x ="discharge_disposition_id", hue = "readmitted", data = df)
    plt.show()
```

```
Proportions of ID's
discharge_disposition_id
Discharged to Home    73649
Other                 23434
Name: count, dtype: int64
```

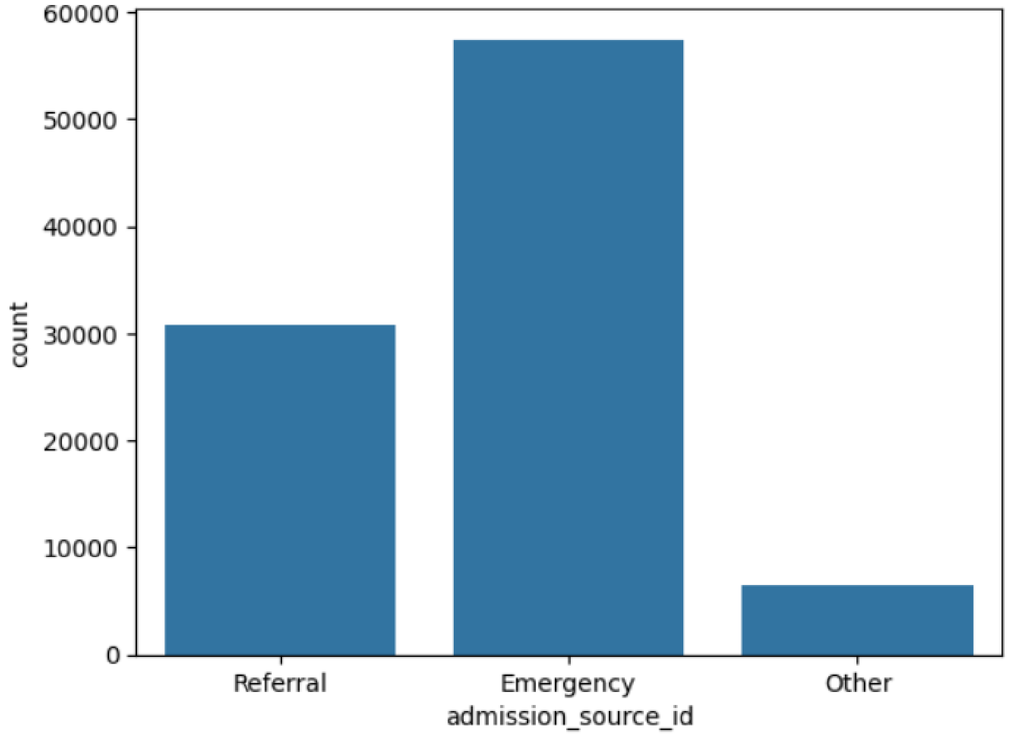


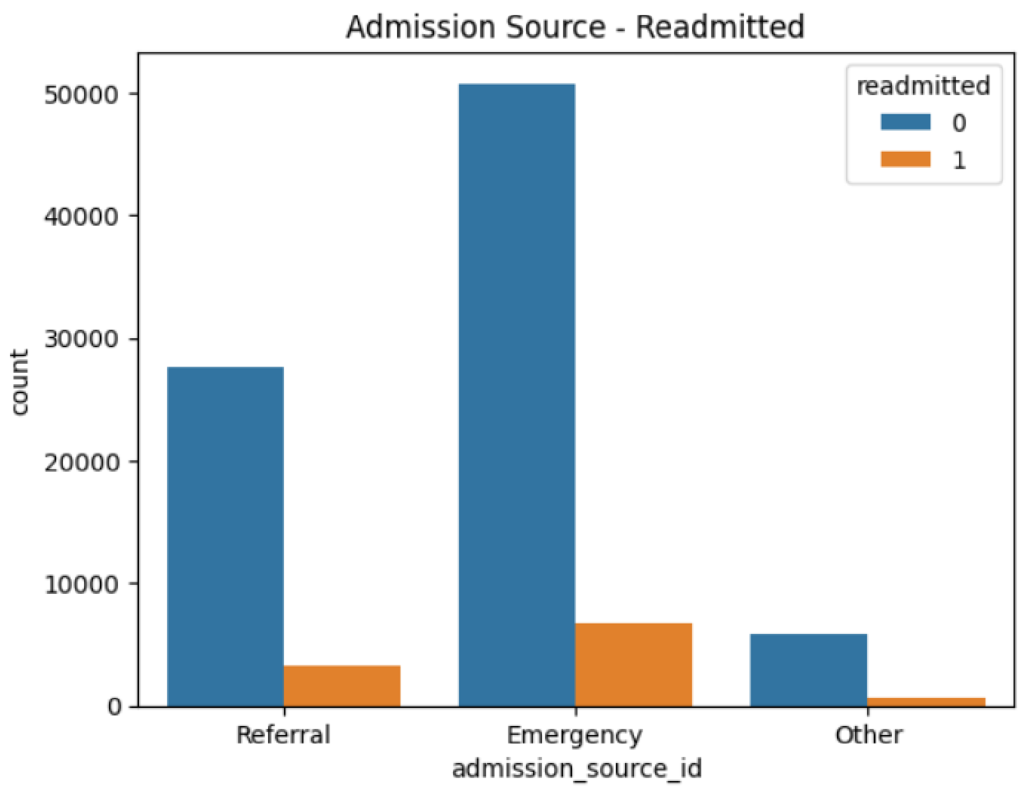
discharge_disposition_id



•

```
]: mapped_adm = {1:"Referral",2:"Referral",3:"Referral",
                4:"Other",5:"Other",6:"Other",10:"Other",22:"Other",25:"Other",
                9:"Other",8:"Other",14:"Other",13:"Other",11:"Other",
                15:np.nan,17:np.nan,20:np.nan,21:np.nan,
                7:"Emergency"}
df.admission_source_id = df.admission_source_id.replace(mapped_adm)
print(df.admission_source_id.value_counts())

sns.countplot(x = "admission_source_id", data = df)
plt.show()

sns.countplot(x = "admission_source_id", hue = "readmitted", data = df)
plt.title("Admission Source - Readmitted")
plt.show()
```

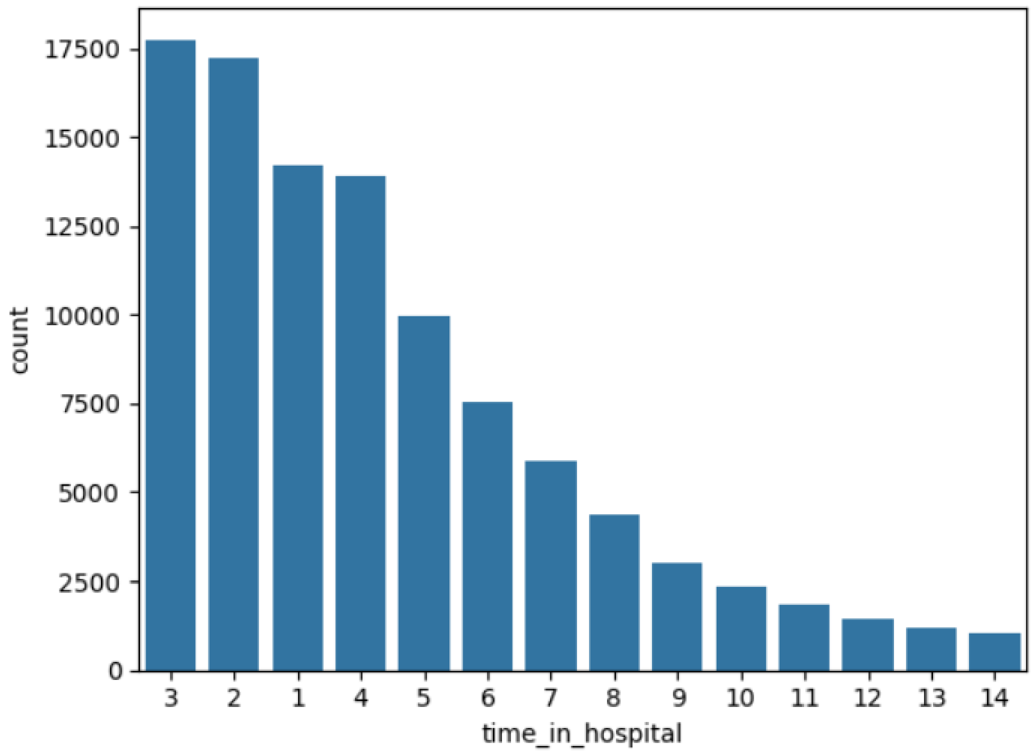
-
  ```
  admission_source_id
  Emergency    57492
  Referral     30855
  Other         6474
  Name: count, dtype: int64
  ```
  

- 

•

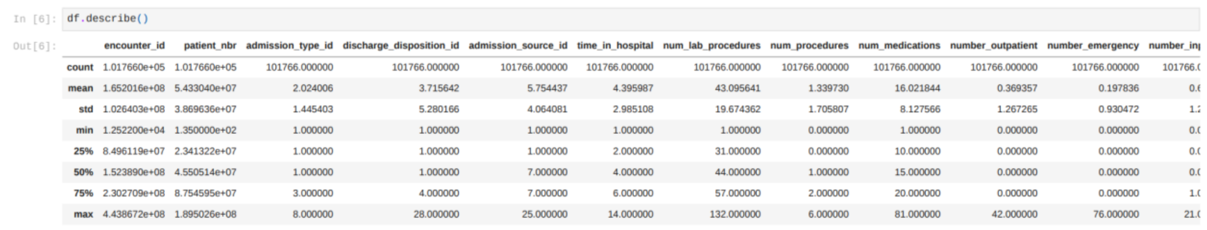
```
In [26]: print(df.time_in_hospital.value_counts())

         sns.countplot(x="time_in_hospital", data = df,
                       order = df.time_in_hospital.value_counts().index)
         plt.show()
```

```
time_in_hospital
3     17756
2     17224
1     14206
4     13924
5      9966
6      7539
7      5859
8      4390
9      3002
10     2342
11     1855
12     1448
13     1210
14     1042
Name: count, dtype: int64
```



Activity 3: 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.



```
In [6]: df.describe()
```

Out[6]:

	encounter_id	patient_nbr	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatient	number_emergency	number_in
count	1.017660e+05	1.017660e+05	101766.000000	101766.000000	101766.000000	101766.000000	101766.000000	101766.000000	101766.000000	101766.000000	101766.000000	101766.
mean	1.652016e+08	5.433040e+07	2.024006	3.715642	5.754437	4.395987	43.095641	1.339730	16.021844	0.369357	0.197836	0.
std	1.026403e+08	3.869636e+07	1.445403	5.280166	4.064081	2.985108	19.674362	1.705807	8.127566	1.267265	0.930472	1.
min	1.252200e+04	1.350000e+02	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.
25%	8.496119e+07	2.341322e+07	1.000000	1.000000	1.000000	2.000000	31.000000	0.000000	10.000000	0.000000	0.000000	0.
50%	1.523890e+08	4.550514e+07	1.000000	1.000000	7.000000	4.000000	44.000000	1.000000	15.000000	0.000000	0.000000	0.
75%	2.302709e+08	8.754595e+07	3.000000	4.000000	7.000000	6.000000	57.000000	2.000000	20.000000	0.000000	0.000000	1.
max	4.438672e+08	1.895026e+08	8.000000	28.000000	25.000000	14.000000	132.000000	6.000000	81.000000	42.000000	76.000000	21.

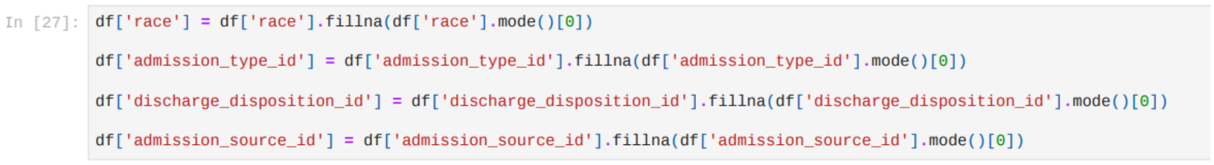
Milestone 4: Model Building

Activity 1: Handling categorical Values

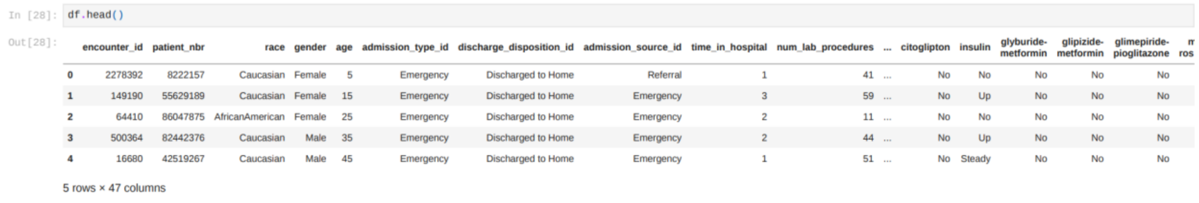
As we can see our dataset has categorical data. Before training our model, we must convert the categorical data into a numeric form.

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.



```
In [27]: df['race'] = df['race'].fillna(df['race'].mode()[0])

         df['admission_type_id'] = df['admission_type_id'].fillna(df['admission_type_id'].mode()[0])

         df['discharge_disposition_id'] = df['discharge_disposition_id'].fillna(df['discharge_disposition_id'].mode()[0])

         df['admission_source_id'] = df['admission_source_id'].fillna(df['admission_source_id'].mode()[0])
```

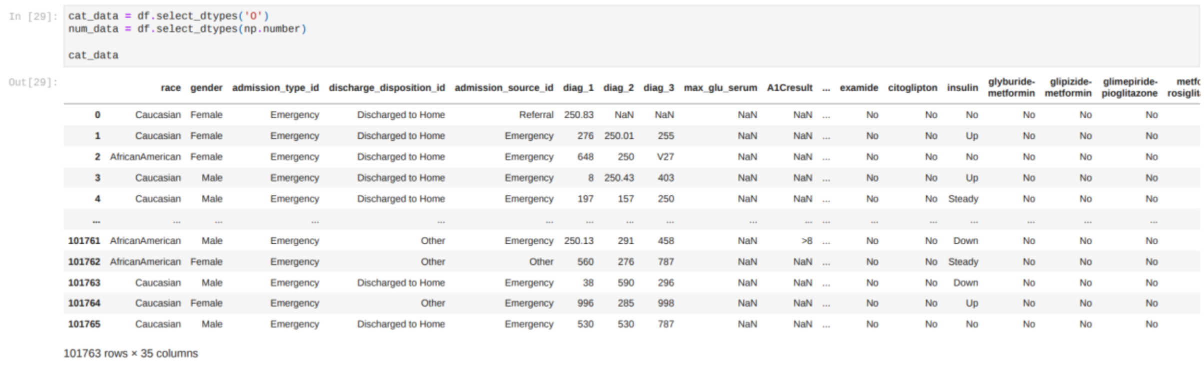


```
In [28]: df.head()
```

Out[28]:

	encounter_id	patient_nbr	race	gender	age	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	num_lab_procedures	...	citoglipton	insulin	glyburide-metformin	glipizide-metformin	glimepiride-pioglitazone	m ros
0	2278392	8222157	Caucasian	Female	5	Emergency	Discharged to Home	Referral	1	41	...	No	No	No	No	No	
1	149190	55629189	Caucasian	Female	15	Emergency	Discharged to Home	Emergency	3	59	...	No	Up	No	No	No	
2	64410	86047875	AfricanAmerican	Female	25	Emergency	Discharged to Home	Emergency	2	11	...	No	No	No	No	No	
3	500364	82442376	Caucasian	Male	35	Emergency	Discharged to Home	Emergency	2	44	...	No	Up	No	No	No	
4	16680	42519267	Caucasian	Male	45	Emergency	Discharged to Home	Emergency	1	51	...	No	Steady	No	No	No	

5 rows × 47 columns



```
In [29]: cat_data = df.select_dtypes('O')
         num_data = df.select_dtypes(np.number)

         cat_data
```

Out[29]:

	race	gender	admission_type_id	discharge_disposition_id	admission_source_id	diag_1	diag_2	diag_3	max_glu_serum	A1Cresult	...	examide	citoglipton	insulin	glyburide-metformin	glipizide-metformin	glimepiride-pioglitazone	metf rosiglit
0	Caucasian	Female	Emergency	Discharged to Home	Referral	250.83	NaN	NaN	NaN	NaN	...	No	No	No	No	No	No
1	Caucasian	Female	Emergency	Discharged to Home	Emergency	276	250.01	255	NaN	NaN	...	No	No	Up	No	No	No
2	AfricanAmerican	Female	Emergency	Discharged to Home	Emergency	648	250	V27	NaN	NaN	...	No	No	No	No	No	No
3	Caucasian	Male	Emergency	Discharged to Home	Emergency	8	250.43	403	NaN	NaN	...	No	No	Up	No	No	No
4	Caucasian	Male	Emergency	Discharged to Home	Emergency	197	157	250	NaN	NaN	...	No	No	Steady	No	No	No
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
101761	AfricanAmerican	Male	Emergency	Other	Emergency	250.13	291	458	NaN	>8	...	No	No	Down	No	No	No
101762	AfricanAmerican	Female	Emergency	Other	Other	560	276	787	NaN	NaN	...	No	No	Steady	No	No	No
101763	Caucasian	Male	Emergency	Discharged to Home	Emergency	38	590	296	NaN	NaN	...	No	No	Down	No	No	No
101764	Caucasian	Female	Emergency	Other	Emergency	996	285	998	NaN	NaN	...	No	No	Up	No	No	No
101765	Caucasian	Male	Emergency	Discharged to Home	Emergency	530	530	787	NaN	NaN	...	No	No	No	No	No	No

101763 rows × 35 columns

Activity 2: Splitting data into train and test data 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.

Lets us perform encoding first and then splitting

```
In [30]: from sklearn.preprocessing import LabelEncoder

         LE = LabelEncoder()

         for i in cat_data:
           cat_data[i] = LE.fit_transform(cat_data[i])
```

```
In [31]: data = pd.concat([num_data,cat_data],axis=1)
         data.head()
         data['glipizide'].unique()

Out[31]: array([1, 2, 3, 0])
```

```
In [32]: data.drop(['encounter_id','patient_nbr'],axis=1,inplace=True)
         data.head()
```

Out[32]:

	age	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatient	number_emergency	number_inpatient	number_diagnoses	readmitted	...	examide	citoglipton	insulin	glyburide-metformin	glipizide-metformin
0	5	1	41	0	1	0	0	0	1	0	...	0	0	1	1	0
1	15	3	59	0	18	0	0	0	9	0	...	0	0	3	1	0
2	25	2	11	5	13	2	0	1	6	0	...	0	0	1	1	0
3	35	2	44	1	16	0	0	0	7	0	...	0	0	3	1	0
4	45	1	51	0	8	0	0	0	5	0	...	0	0	2	1	0

5 rows × 45 columns

```
In [33]: X = data.drop('readmitted',axis=1)

         y = data['readmitted']
```

```
In [34]: X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

```
In [35]: SC = StandardScaler()

         X_train_scaled = pd.DataFrame(SC.fit_transform(X_train),columns=X_train.columns)

         X_test_scaled = pd.DataFrame(SC.transform(X_test),columns=X_test.columns)
```

```
In [36]: X_train.shape,X_test.shape,y_train.shape,y_test.shape

Out[36]: ((81410, 44), (20353, 44), (81410,), (20353,))
```

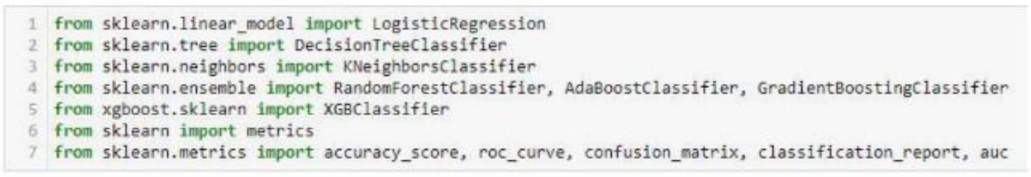
Activity 3: Splitting train data into train and validation sets

```
In [35]: SC = StandardScaler()

         X_train_scaled = pd.DataFrame(SC.fit_transform(X_train),columns=X_train.columns)

         X_test_scaled = pd.DataFrame(SC.transform(X_test),columns=X_test.columns)
```

```
In [36]: X_train.shape,X_test.shape,y_train.shape,y_test.shape

Out[36]: ((81410, 44), (20353, 44), (81410,), (20353,))
```

Activity 4: 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
```

```
:   LR = LogisticRegression()

    LR.fit(X_train_scaled,y_train)

    LR.score(X_train_scaled,y_train)

:   0.888134135855546
```

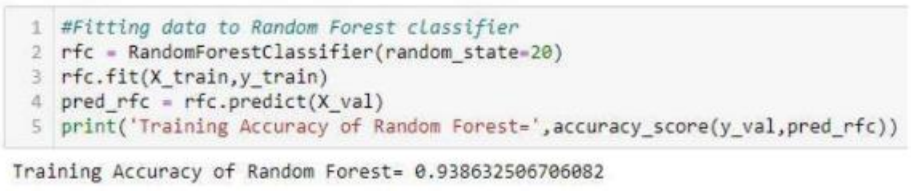
```
In [39]:  RF = RandomForestClassifier()

          RF.fit(X_train_scaled,y_train)

          RF.score(X_train_scaled,y_train)

Out[39]:  1.0
```

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.

```
In [38]:  LR.score(X_test_scaled,y_test)

Out[38]:  0.8882719992138751
```

```
In [40]:  RF.score(X_test_scaled,y_test)

Out[40]:  0.889156389721417
```

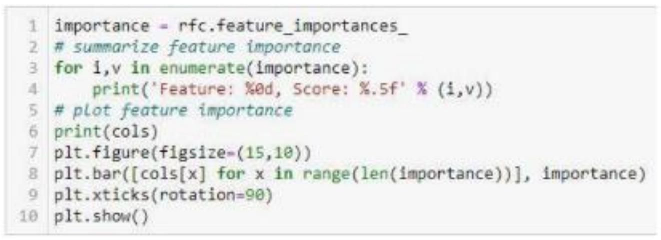
From the above results, it is clear that Random Forest Classifier provides the best accuracy. So let us create a different variable to fit and make predictions using the model.



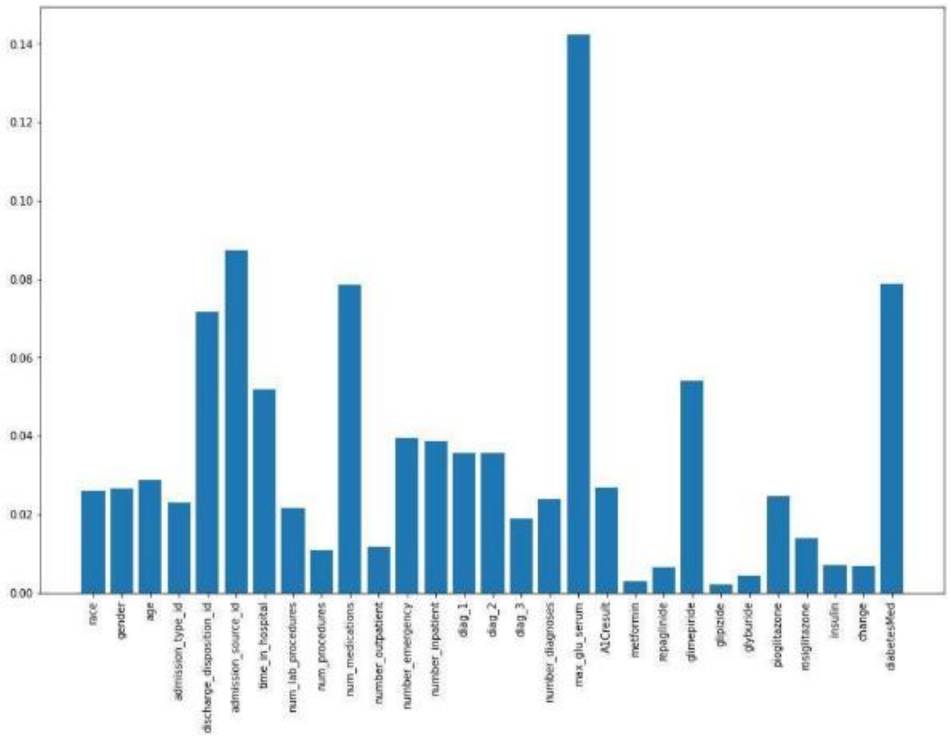
```
#Fitting data to Random Forest classifier
rfc = RandomForestClassifier(random_state=20)
rfc.fit(X_train,y_train)
pred_rfc = rfc.predict(X_val)
print('Training Accuracy of Random Forest=',accuracy_score(y_val,pred_rfc))

Training Accuracy of Random Forest= 0.938632506706082
```

Activity 5: 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.



```python
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()
```



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

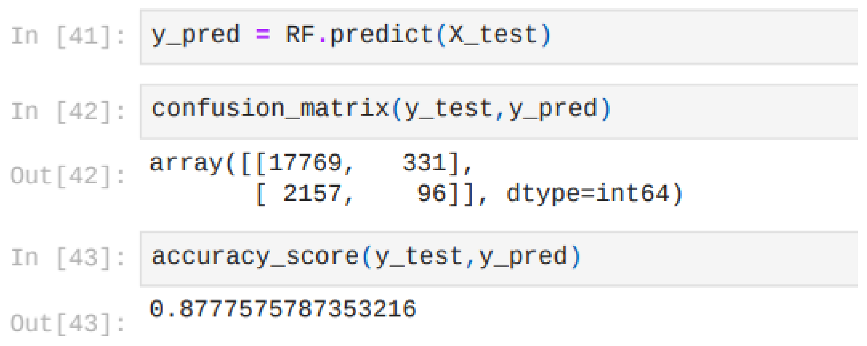
● glimepiride : glimepiride dosage - Values: “up” if the dosage was increased during the encounter, “down” if the dosage was decreased, “steady” if the dosage did not change, and “no” if the drug was not prescribed

● diabetesMed : Indicates if there was any diabetic medication prescribed. Values: “yes” and “no”

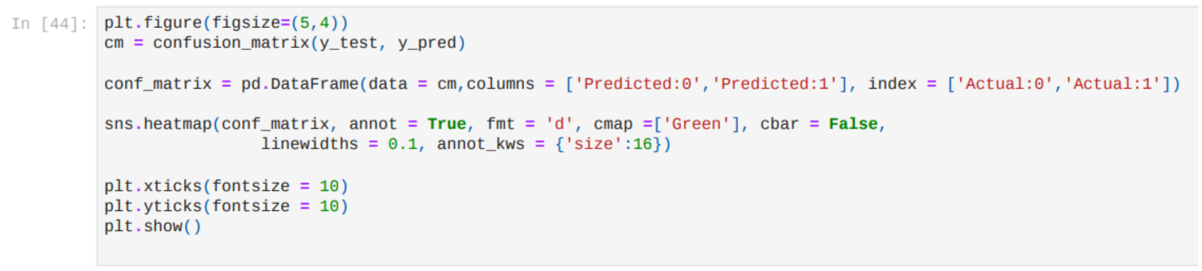
Activity 6: Evaluating final model performance

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

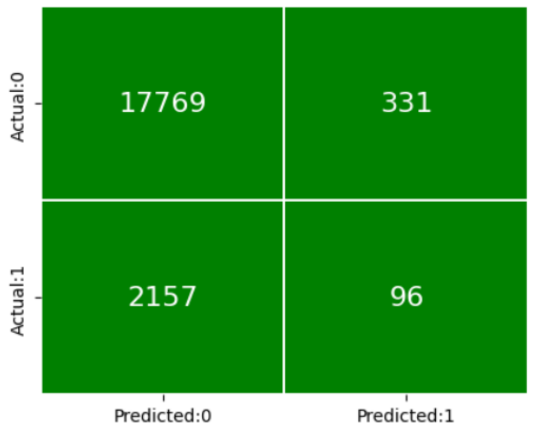
In order to obtain these, we will be using the confusion_matrix(),roc_curve() and classification_report() functions from sklearn.metrics.



```
In [41]: y_pred = RF.predict(X_test)

In [42]: confusion_matrix(y_test,y_pred)

Out[42]: array([[17769,   331],
                [ 2157,    96]], dtype=int64)

In [43]: accuracy_score(y_test,y_pred)

Out[43]: 0.8777575787353216
```

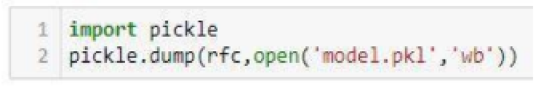


```
In [44]: plt.figure(figsize=(5,4))
         cm = confusion_matrix(y_test, y_pred)

         conf_matrix = pd.DataFrame(data = cm,columns = ['Predicted:0','Predicted:1'], index = ['Actual:0','Actual:1'])

         sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap =['Green'], cbar = False,
                         linewidths = 0.1, annot_kws = {'size':16})

         plt.xticks(fontsize = 10)
         plt.yticks(fontsize = 10)
         plt.show()
```



Activity 7: 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'))
```

Milestone 5: Application Building

In this section, we will be building a web application that is integrated to the model we built.

A UI is provided for the uses where he has to enter the values for predictions. The entered

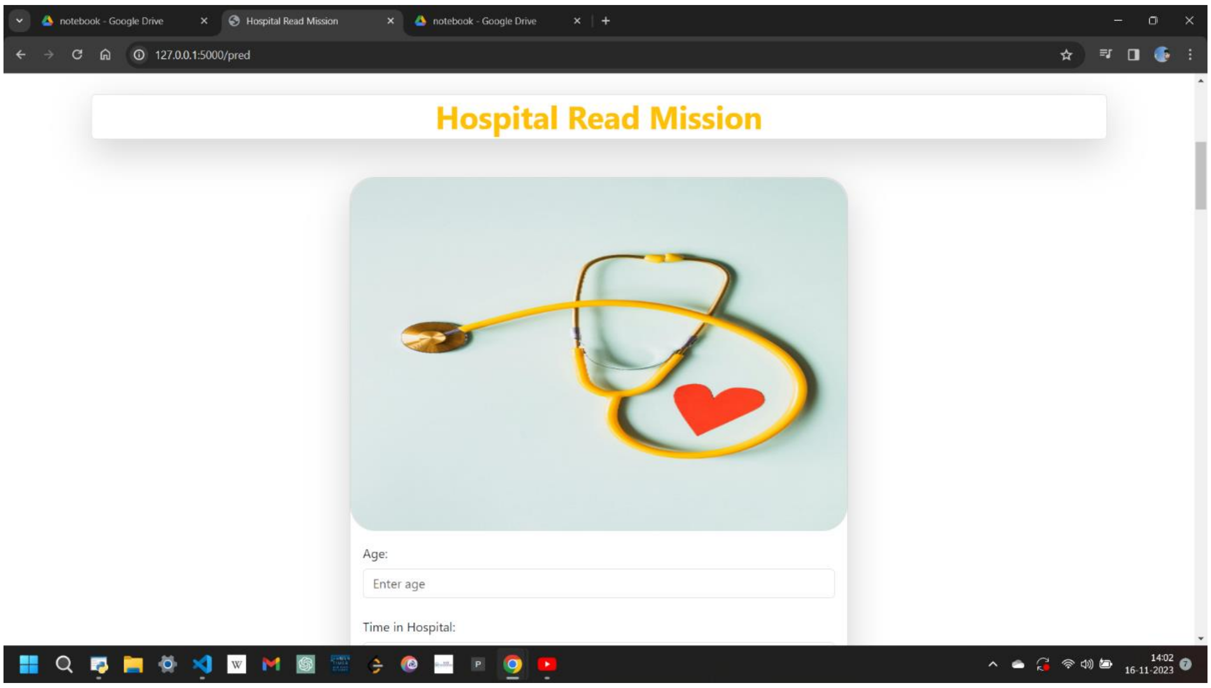
values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server side script

Activity1: Building Html Pages:

Lets see how our page looks like:

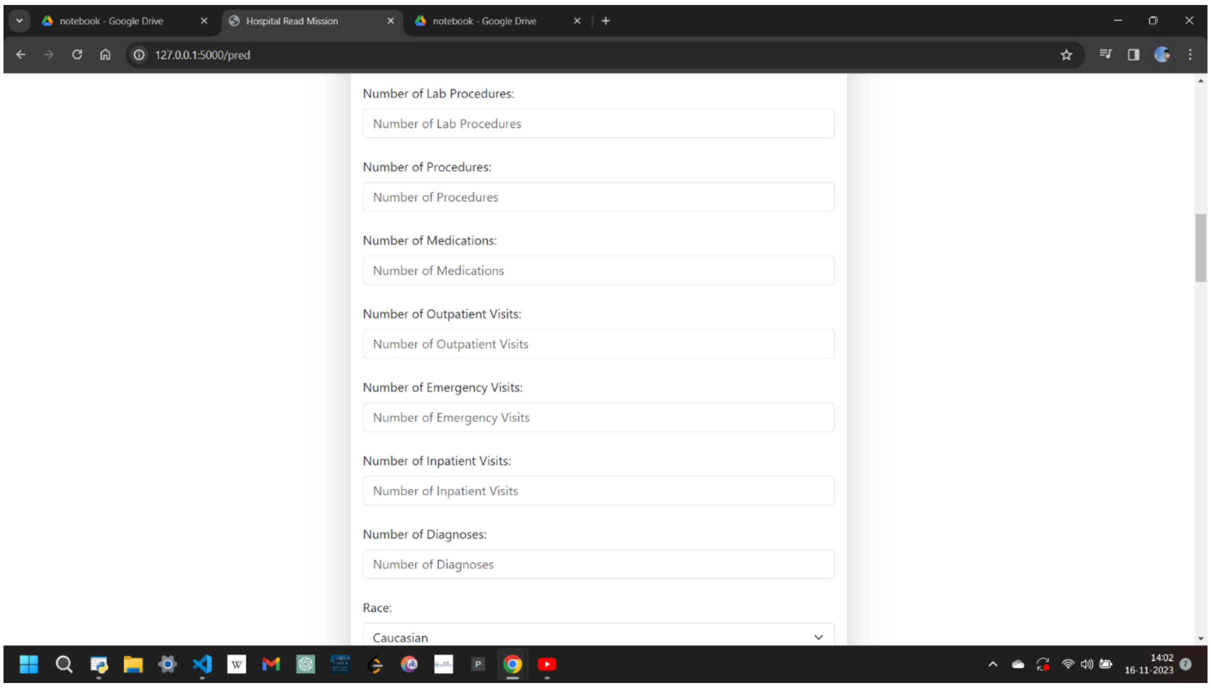


Hospital Read Mission

Age:

Enter age

Time in Hospital:



Number of Lab Procedures:

Number of Lab Procedures

Number of Procedures:

Number of Procedures

Number of Medications:

Number of Medications

Number of Outpatient Visits:

Number of Outpatient Visits

Number of Emergency Visits:

Number of Emergency Visits

Number of Inpatient Visits:

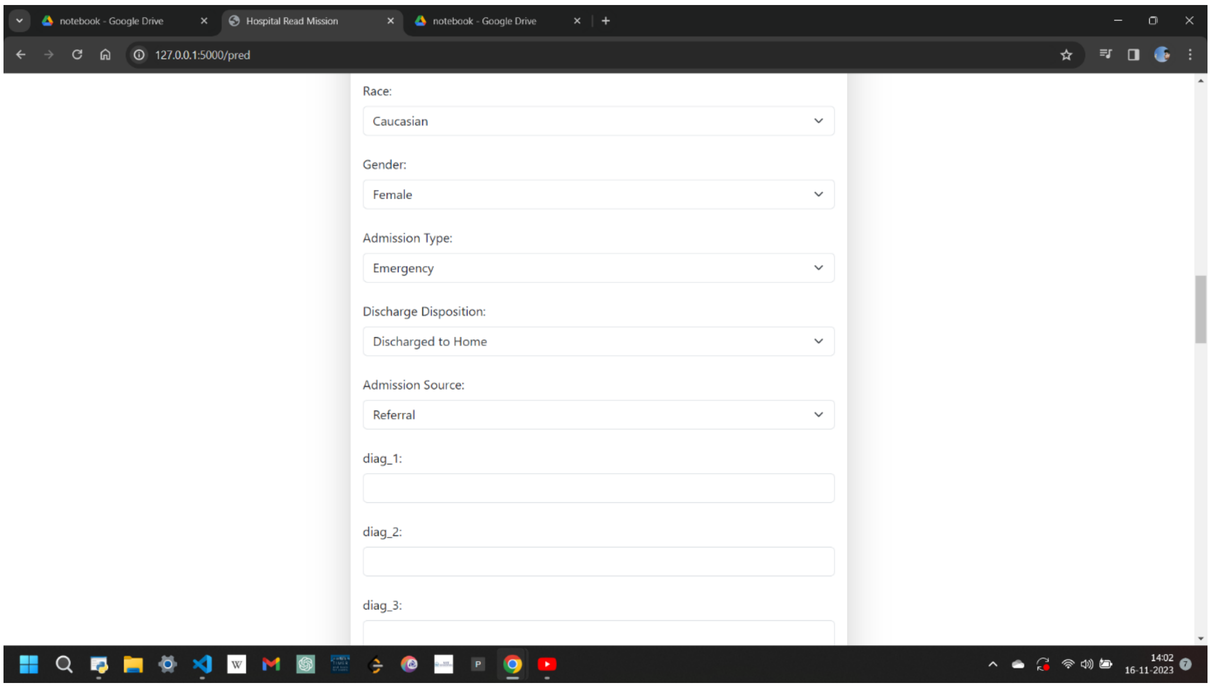
Number of Inpatient Visits

Number of Diagnoses:

Number of Diagnoses

Race:

Caucasian



notebook - Google Drive | Hospital Read Mission | notebook - Google Drive

127.0.0.1:5000/pred

Race: Caucasian

Gender: Female

Admission Type: Emergency

Discharge Disposition: Discharged to Home

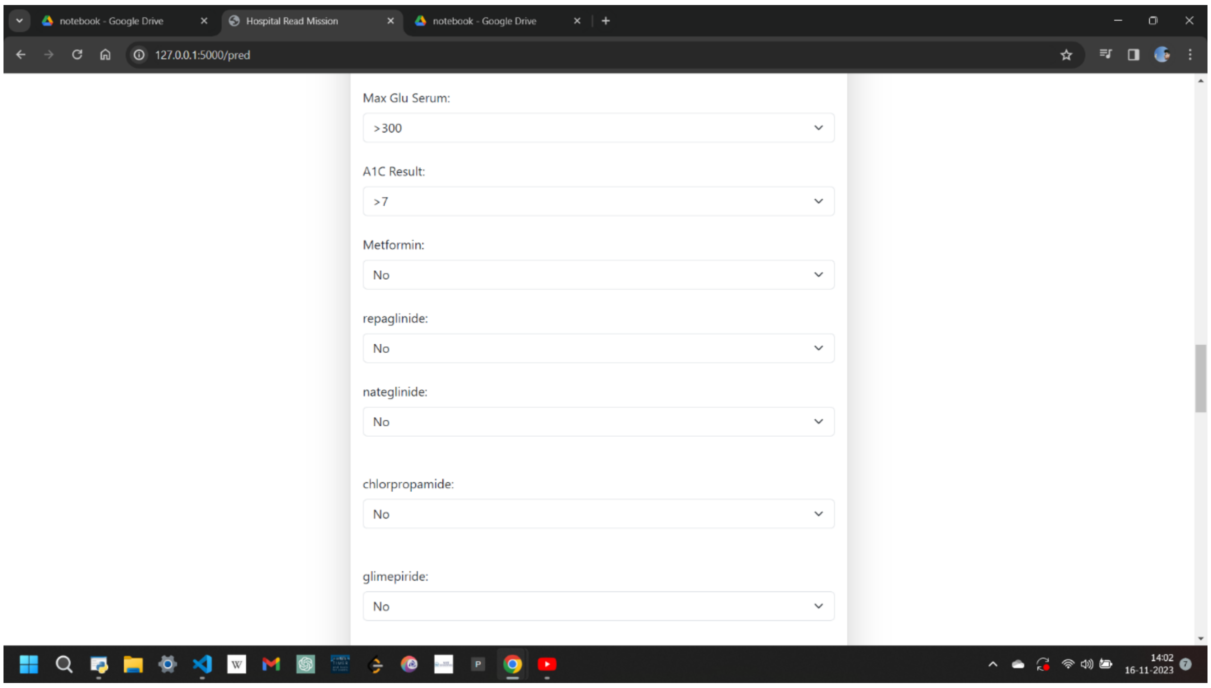
Admission Source: Referral

diag_1:

diag_2:

diag_3:

14:02 16-11-2023



notebook - Google Drive | Hospital Read Mission | notebook - Google Drive

127.0.0.1:5000/pred

Max Glu Serum: >300

A1C Result: >7

Metformin: No

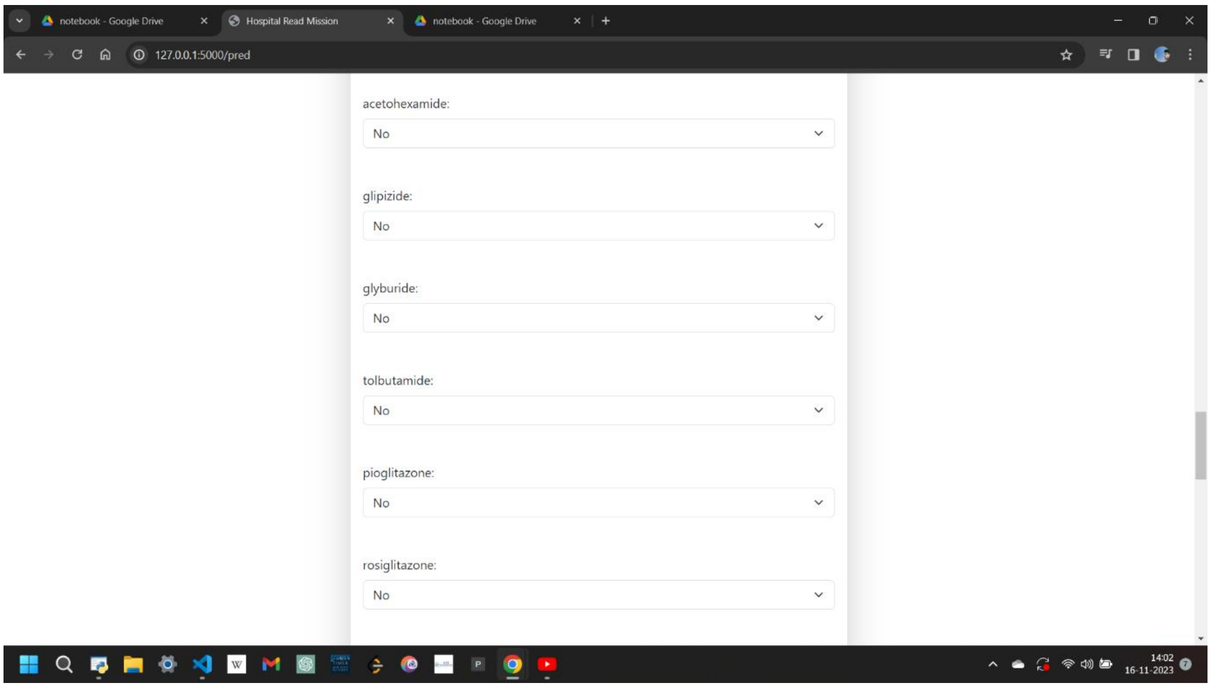
repaglinide: No

nateglinide: No

chlorpropamide: No

glimepiride: No

14:02 16-11-2023



acetohexamide: No

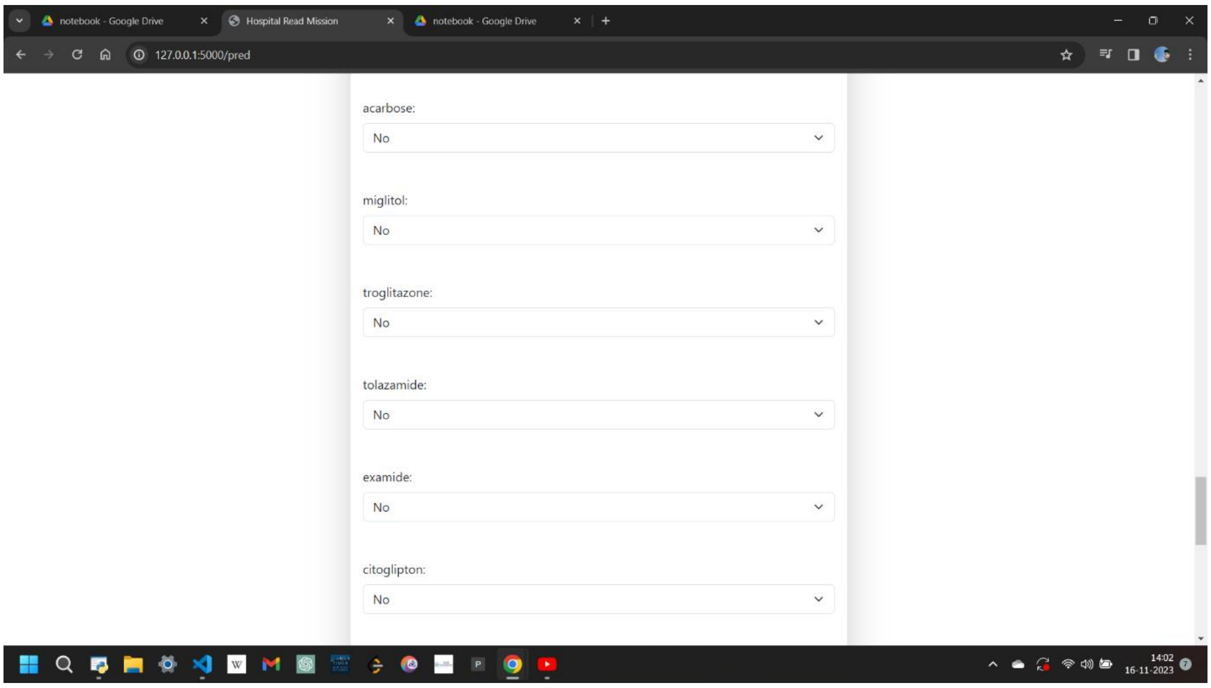
glipizide: No

glyburide: No

tolbutamide: No

pioglitazone: No

rosiglitazone: No



acarbose: No

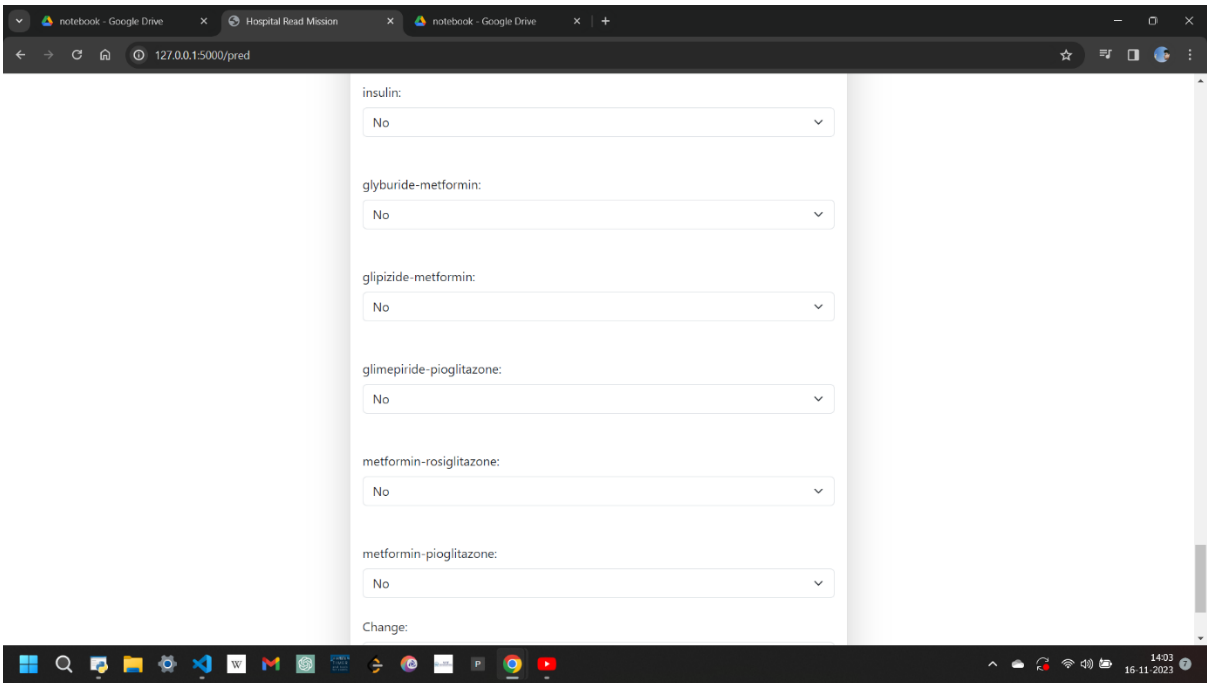
miglitol: No

troglitazone: No

tolazamide: No

examide: No

citoglipton: No



insulin: No

glyburide-metformin: No

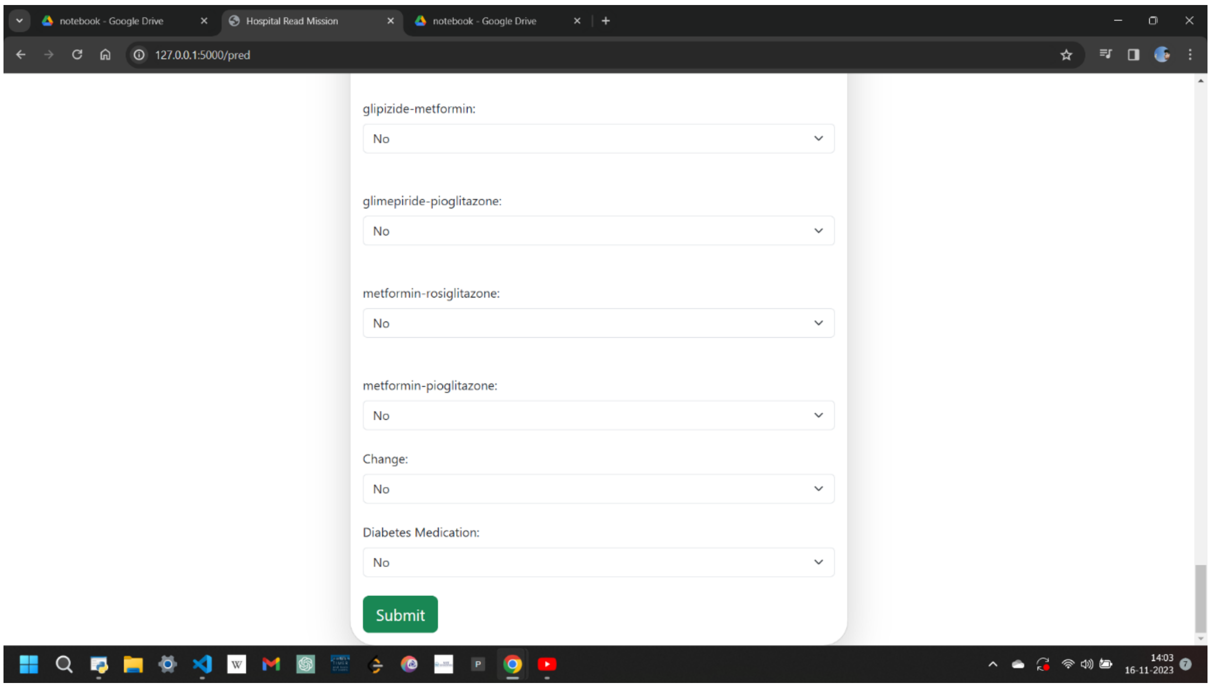
glipizide-metformin: No

glimepiride-pioglitazone: No

metformin-rosiglitazone: No

metformin-pioglitazone: No

Change:



glipizide-metformin: No

glimepiride-pioglitazone: No

metformin-rosiglitazone: No

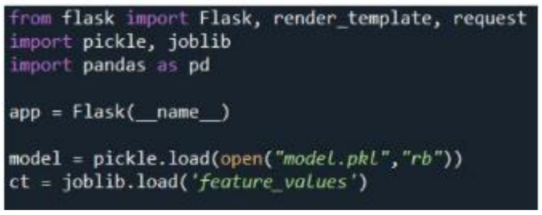
metformin-pioglitazone: No

Change: No

Diabetes Medication: No

Activity 2: 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')
```

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

Retrieves the value from UI:

```
# pip install flask

from flask import Flask,render_template,request
import pickle
import pandas as pd
import numpy as np
import os
# loading the label encoder
encoder=pickle.load(open('label_encoder.pkl','rb'))

# loading my mlr model
model=pickle.load(open("model.pkl",'rb'))

#loading Scaler
scalar=pickle.load(open("scaler.pkl",'rb'))

# Flask is used for creating your application
# render template is use for rendering the html page


app= Flask(__name__)  # your application


@app.route('/')  # default route
def home():
    return render_template('home.html') # rendering if your
home page.
```

```
@app.route('/pred',methods=['POST']) # prediction route
def predict1():
    a1=request.form["age"]
    a2=request.form["time_in_hospital"]
    a3=request.form["num_lab_procedures"]
    a4=request.form["num_procedures"]
    a5=request.form["num_medications"]
    a6=request.form["number_outpatient"]
    a7=request.form["number_emergency"]
    a8=request.form["number_inpatient"]
    a44=request.form["number_diagnoses"]
    a9=request.form["race"]
    a10=request.form["gender"]
    a11=request.form["admission_type_id"]
    a12=request.form["discharge_disposition_id"]
    a13=request.form["admission_source_id"]
    a14=request.form["diag_1"]
    a15=request.form["diag_2"]
    a16=request.form["diag_3"]
    a17=request.form["max_glu_serum"]
    a18=request.form["A1Cresult"]
    a19=request.form["metformin"]
    a20=request.form["repaglinide"]
    a21=request.form["nateglinide"]
    a22=request.form["chlorpropamide"]
    a23=request.form["glimepiride"]
    a24=request.form["acetohexamide"]
    a25=request.form["glipizide"]
    a26=request.form["glyburide"]
    a27=request.form["tolbutamide"]
    a28=request.form["pioglitazone"]
    a29=request.form["rosiglitazone"]
    a30=request.form["acarbose"]
    a31=request.form["miglitol"]
    a32=request.form["troglitazone"]
    a33=request.form["tolazamide"]
    a34=request.form["examide"]
    a35=request.form["citoglipton"]
    a36=request.form["insulin"]
    a37=request.form["glyburide-metformin"]
    a38=request.form["glipizide-metformin"]
    a39=request.form["glimepiride-pioglitazone"]
    a40=request.form["metformin-rosiglitazone"]
    a41=request.form["metformin-pioglitazone"]
    a42=request.form["change"]
    a43=request.form["diabetes_med"]
    t =
[a1,a2,a3,a4,a5,a6,a7,a8,a44,a9,a10,a11,a12,a13,a14,a15,a16,a
17,a18,a19,a20,a21,a22,a23,a24,a25,a26,a27,a28,a29,a30,a31,a3
2,a33,a34,a35,a36,a37,a38,a39,a40,a41,a42,a43]

    for i in range(0,len(t)):
        t[i]=float(t[i])

    t=np.reshape(t,[1,-1])
    output =model.predict(t)
    print(output)
    if output==0:
        return render_template("home.html", result = "You
Have to not admit")
    else:
        return render_template("home.html", result = "The
have to readmit")
```

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:

```
# running your application
if __name__ == "__main__":
    app.run()

#http://localhost:5000/ or localhost:5000
```

Activity 3: Run the application

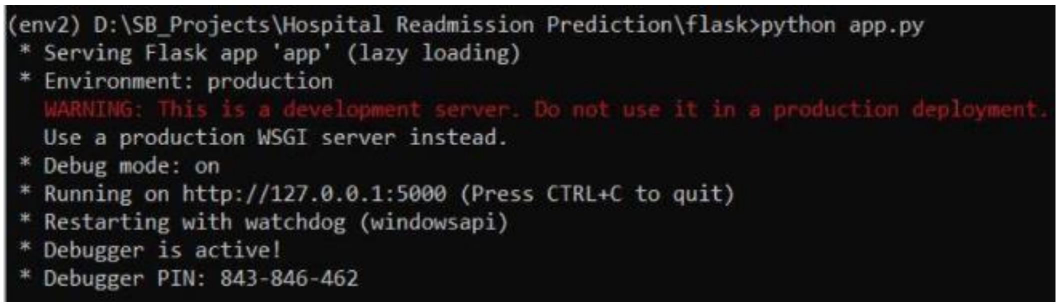
Open anaconda prompt from the start menu

● Navigate to the folder where your python script is.

● Now type “python app.py” command

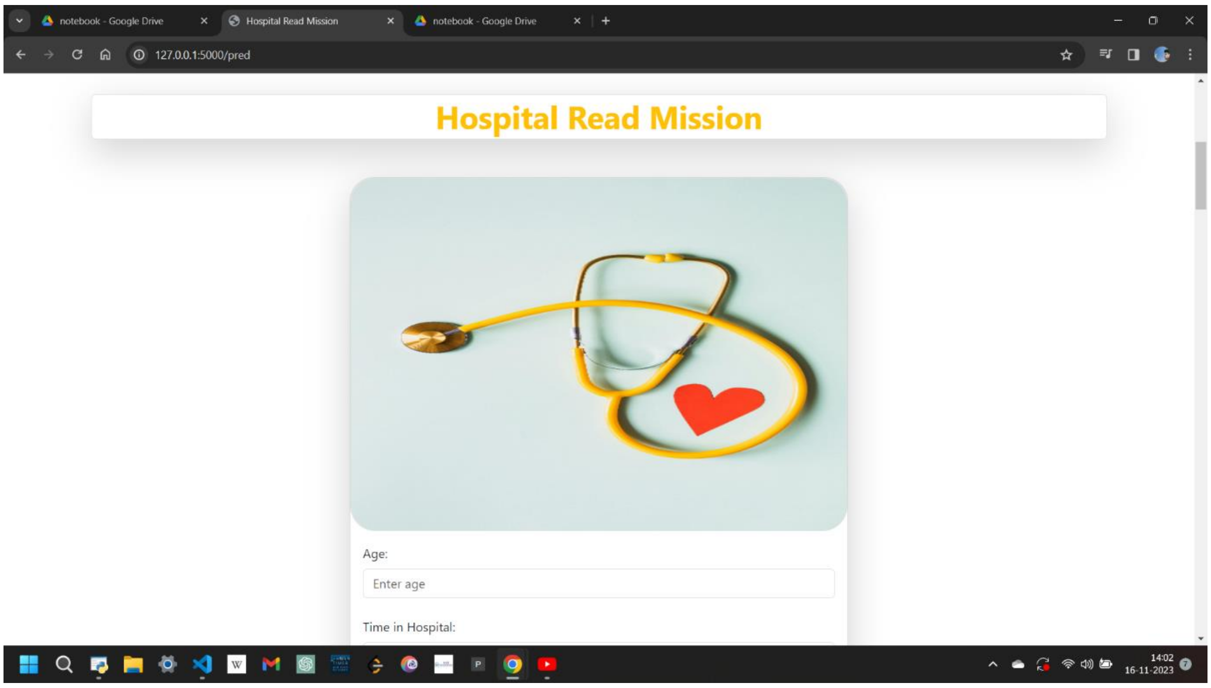
● Navigate to the localhost where you can view your web page.

● Click on the proceed button, enter the inputs, click on the predict button, and see the result/prediction on the web.



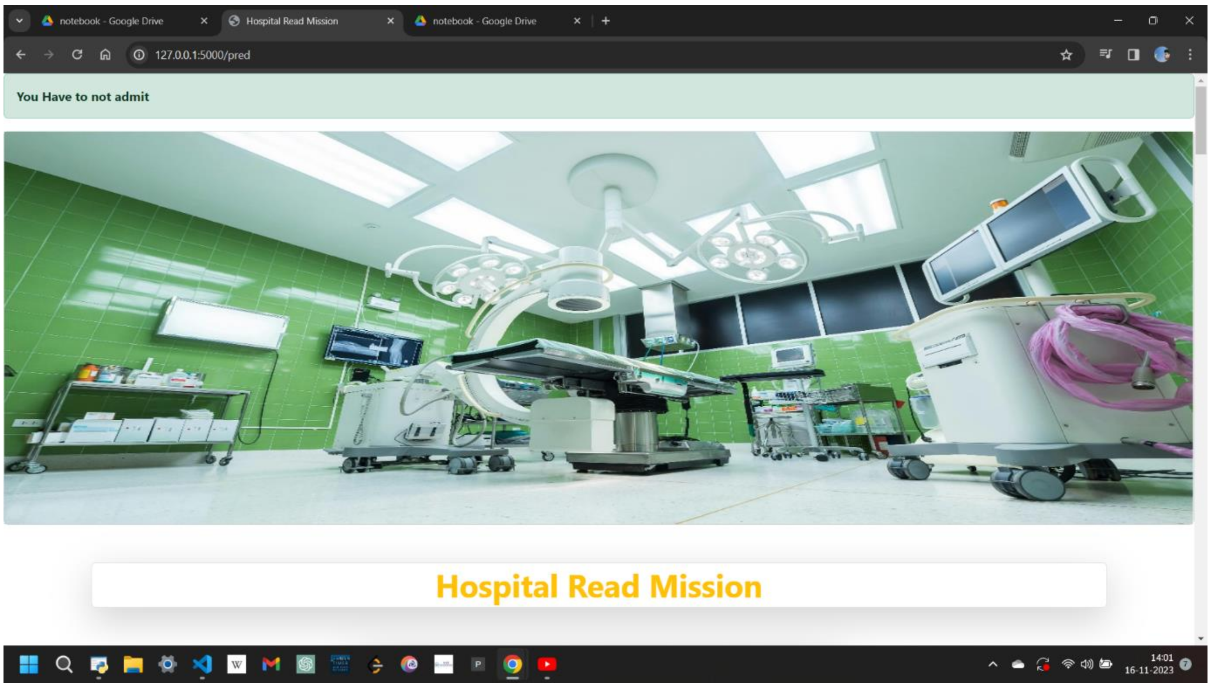
```
(env2) D:\SB_Projects\Hospital Readmission Prediction\flask>python app.py
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running 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.

Output:



You Have to not admit

Hospital Read Mission