A

Capstone Project Report (Final)

On

"DIABETIC PATIENTS' READMISSION PREDICTION"

Submitted in partial fulfillment of the requirements for the award of degree of

PGP in **Data Science and Engineering**

Affiliated to

Great Lakes Institute of Executive Learning

(Session 2021)

Guided by-

Mr. Sravan Malla

Submitted by-

Mohit Sahu

Rahul Mamgai

Ravi Chaubey

Sachin Pathania

Sushil

Candidate's Declaration

We hereby declare that the work, which is being presented in the **Capstone Project Report (Final)**, entitled **Diabetic Patients' Readmission Prediction** in partial fulfillment for the award of degree of **PGP** in **Data Science and Engineering** is a record of our own work/investigations carried under the guidance of Mr. Sravan Malla**.**

We have not submitted the matter presented in this Capstone Project Report anywhere for the award of any other Degree.

PGPDSE-GGN-JAN21-GROUP-4

(1) Mohit Sahu

(2) Rahul Mamgai

(3) Ravi Chaubey

(4) Sachin Pathania

(5) Sushil

ABSTRACT

Title: Diabetes Patients' Readmission Prediction

Project Description:

Hospital readmission is an indicator of the quality of care and is a driver for the increasing cost of healthcare. Like other chronic diseases, Diabetes is associated with a higher risk of hospital readmission. In this research, we evaluate several machine learning approaches to predict the probability of hospital re-admissions for diabetic patients. The data set used for this study contains more than 1 Lac+ diabetic patient data and 50 variables including length of stay, insulin, and in-patient visits from hospitals in the United States. We leverage several pre-processing techniques and investigate the performance of the various models. The significant variables contributing to the analysis are the number of in-patients, length of stay, number of medications, number of diagnoses, and age. The results demonstrate the viability of the techniques in providing a better understanding of factors influencing hospital re-admission.

Tools & Technologies used:

Programming Language: Python

Models: Logistic Regression, Decision Tree, Random Forest, LightGBM, XGBoost

Libraries: Numpy, Pandas, Seaborn, Matplotlib, Scikit-learn, Lightgbm, Xgboost, Warnings.

Visualization: Python (Matplotlib and Seaborn)

IDE: Jupyter Notebook

Table of Contents:

CHAPTER 1

Introduction to the project

1.1 Background

Diabetes Mellitus (DM) is a chronic disease where the blood has high sugar level. It can occur when the pancreas does not produce enough insulin, or when the body cannot effectively use the insulin it produces (WHO). Diabetes is a progressive disease that can lead to a significant number of health complications and profoundly reduce the quality of life. While many diabetic patients manage the health complication with diet and exercise, some require medications to control blood glucose level. As published by a research article named "The relationship between diabetes mellitus and 30-day readmission rates", it is estimated that 9.3% of the population in the United States have diabetes mellitus (DM), 28% of which are undiagnosed. In recent years, government agencies and healthcare systems have increasingly focused on 30-day readmission rates to determine the complexity of their patient populations and to improve quality. Thirty-day readmission rates for hospitalized patients with DM are reported to be between 14.4 and 22.7%, much higher than the rate for all hospitalized patients (8.5–13.5%).

1.2 Problem Statement

To identify the factors that lead to the high readmission rate of diabetic patients within 30 days post discharge and correspondingly to predict the high-risk diabetic-patients who are most likely to get readmitted within 30 days so that the quality of care can be improved along with improved patient's experience, health of the population and reduce costs by lowering readmission rates. Also, to identify the medicines that are the most effective in treating diabetes.

1.3 Impact on business

Hospital readmission is an important contributor to total medical expenditures and is an emerging indicator of quality of care. Diabetes, similar to other chronic medical conditions, is associated with increased risk of hospital readmission. As mentioned in the article "Correction to: Hospital Readmission of Patients with Diabetes", hospital readmission is a high-priority health care quality measure and target for cost reduction, particularly within 30 days of discharge. The burden of diabetes among hospitalized patients is substantial, growing, and costly, and readmissions contribute a significant portion of this burden. Reducing readmission rates among patients with diabetes has the potential to greatly reduce health care costs while simultaneously improving care. Our aim is to provide some insights into the risk factors for readmission and also to identify the medicines that are the most effective in treating diabetes.

CHAPTER 2

Dataset and Data Dictionary

2.1 Dataset

The data subset used for analysis covers 10 years of diabetes patient encounter data (1999 – 2008) among 130 US hospitals with over 100,000 diabetes patients. Moreover, all the encounters used for analysis satisfy five key criteria:

- It is a hospital admission.
- The inpatient was classified as diabetic (at least one of three initial diagnoses included diabetes).
- The length of stay was comprised from 1 to 14 days.
- The inpatient underwent laboratory testing.
- The inpatient received medication during its stay.

Data Source: **UCI Dataset Link**

2.2. Data Dictionary

Column Name	Datatype	Description	Information	Null Values
encounter_id	int64	ID assigned each time a patient arrives at the hospital.	101766 unique values	0%
patient_nbr	int64	Unique ID for every patient.	71518 values	0%
race	object	Race of the patients.	5 categories	2.23%
gender	object	Gender of the patients.	2 categories	0.0029%
age	object	Grouped in 10-year intervals: [0, 10), [10, 20), ..., [90, 100)	10 Bins of size 10	0%
weight	object	Weight in pounds. (Bins)	9 Bins of size 25	96.85%
admission_type_id	int64	Integer identifier corresponding to distinct values, for example, emergency, urgent, elective etc.	8 categories	0%
discharge_disposition_id	int64	Integer identifier corresponding to distinct values, for example, discharged to home etc.	26 categories	0%
admission_source_id	int64	Integer identifier corresponding to distinct values, for example, physician referral, emergency room etc.	17 categories	0%
time_in_hospital	int64	number of days between admission and discharge	1 – 14 Days	0%

payer_code	object	Integer identifier corresponding to distinct values, for example, Blue Cross\Blue Shield, Medicare etc.	23 categories	40%
medical_specialty	object	Integer identifier of a specialty of the admitting physician	73 categories	49%
num_lab_procedures	int64	Number of lab tests performed during the encounter	1 – 132 lab procedures	0%
num_procedures	int64	Number of procedures (other than lab tests) performed during the encounter	0 – 6 procedures	0%
num_medications	int64	Number of distinct generic names administered during the encounter	1 – 81 medications	0%
number_outpatient	int64	Number of outpatient visits of the patient in the year preceding the encounter	0 – 42 times	0%
number_emergency	int64	Number of emergency visits of the patient in the year preceding the encounter	0 – 76 times	0%
number_inpatient	int64	Number of inpatient visits of the patient in the year preceding the encounter	0 – 21 times	0%
diag_1	object	The primary diagnosis (coded as first three digits of ICD9)	717 unique valu es	0.2%
diag_2	object	Secondary diagnosis (coded as first three digits of ICD9)	749 unique values	0.35%
diag_3	object	Additional secondary diagnosis (coded as first three digits of ICD9)	790 unique values	1.3%
number_diagnoses	int64	Number of diagnoses entered to the system	1 – 16 diagnoses	0%
max_glu_serum	object	The Glucose level in a patient given in (mg/dL).	4 categories	0%
A1Cresult	object	The A1C test result of a patient in percentage.	4 categories	0%
metformin	object		4 categories	0%
repaglinide	object		4 categories	0%
nateglinide	object		4 categories	0%
chlorpropamide	object		4 categories	0%
glimepiride	object		4 categories	0%
acetohexamide	object		2 categories	0%
glipizide	object		4 categories	0%
glyburide	object		4 categories	0%

tolbutamide	object	(Types of medicines)	2 categories	0%
pioglitazone	object		4 categories	0%
Rosiglitazone	object		4 categories	0%
acarbose	object		4 categories	0%
Miglitol	object		4 categories	0%
troglitazone	object		2 categories	0%
Tolazamide	object		3 categories	0%
examide	object		1 category	0%
citoglipton	object		1 category	0%
insulin	object		4 categories	0%
glyburidemetformin	object		4 categories	0%
glipizidemetformin	object		2 categories	0%
glimepiridepioglitazone	object		2 categories	0%
metforminrosiglitazone	object		2 categories	0%
metforminpioglitazone	object		2 categories	0%
Change	object	Indicates if there was a change in diabetic medications	2 categories	0%
diabetesMed	object	Indicates if there was any diabetic medication prescribed.	2 categories	0%
readmitted	object	Days to inpatient readmission	3 categories	0%

CHAPTER 3

Data Exploration (EDA)

3.1 Relationship between the variables:

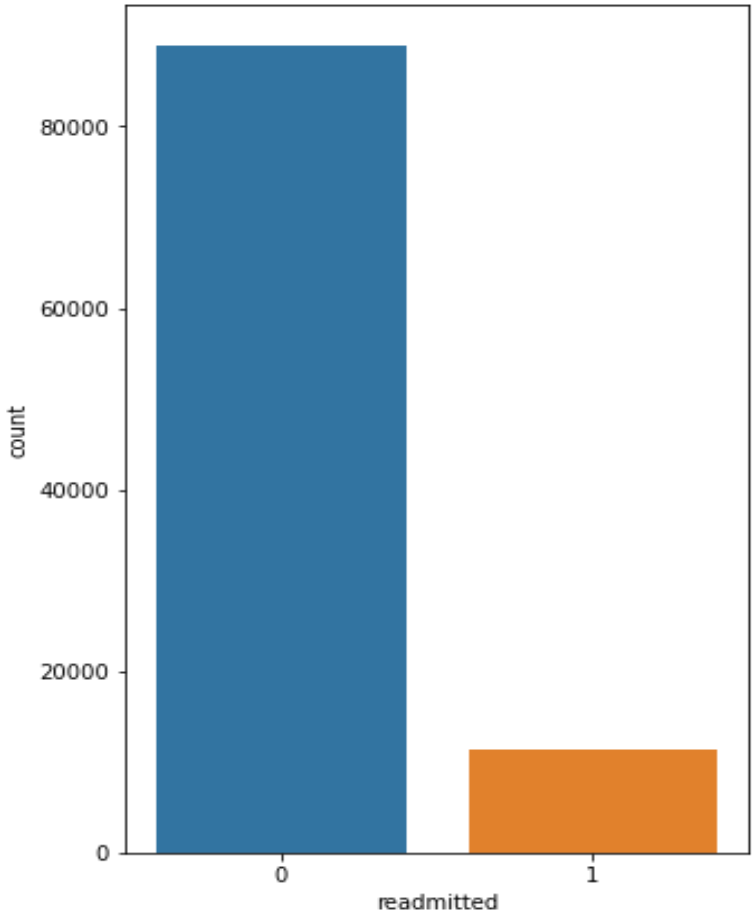
There are a total of 50 features, which will be further transformed during Data Processing

- Independent Features
- Re-admitted (Target dependent variable)

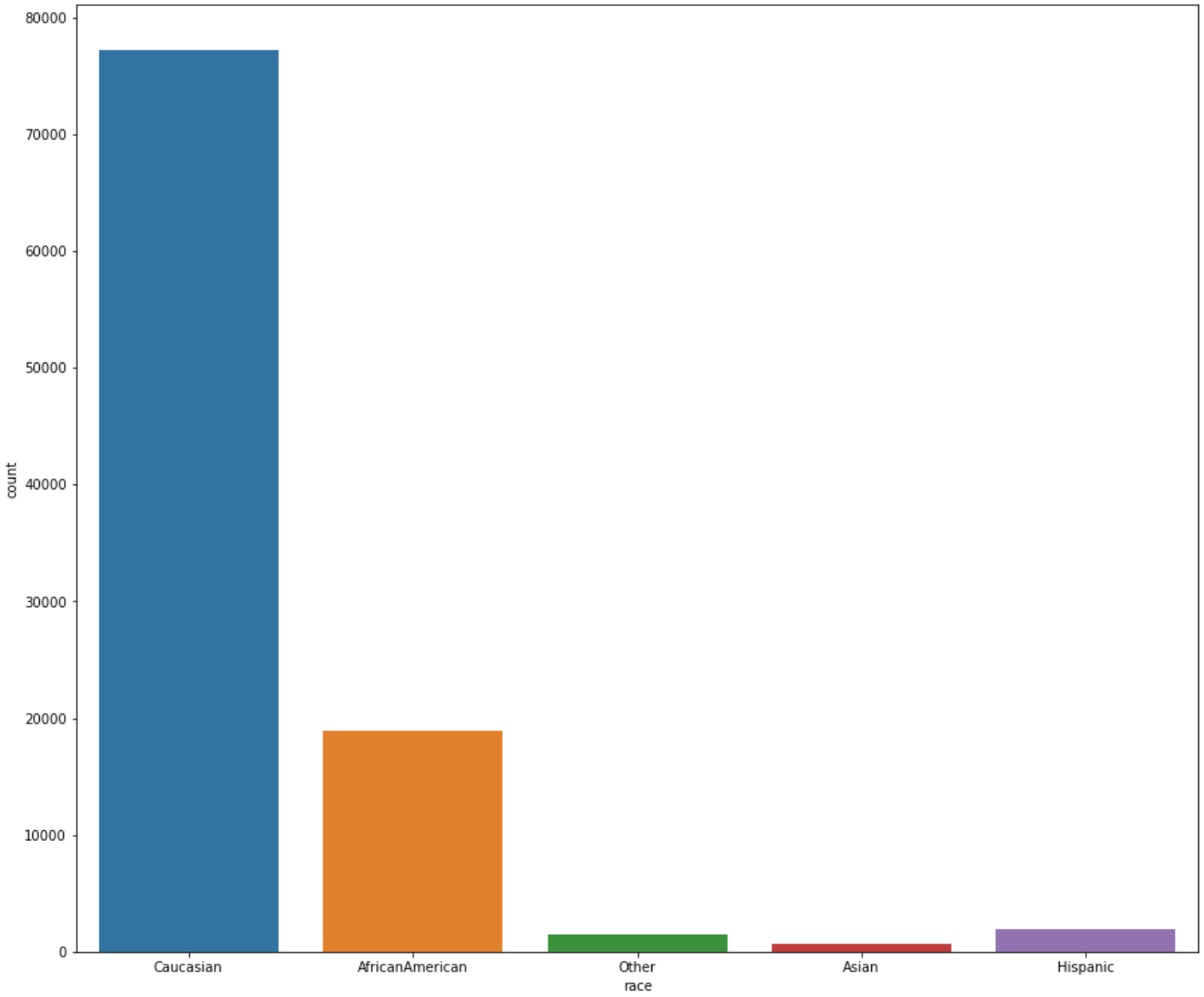
3.2 Univariate Analysis:

- **Count of Target column :**

In our target variable column there are 3 sub-categories present we have converted it into 2 categories, and these 2 categories have imbalance data. Here 0 represents 'Not Re-admitted' and 1 represents 'Re-admitted' patients.

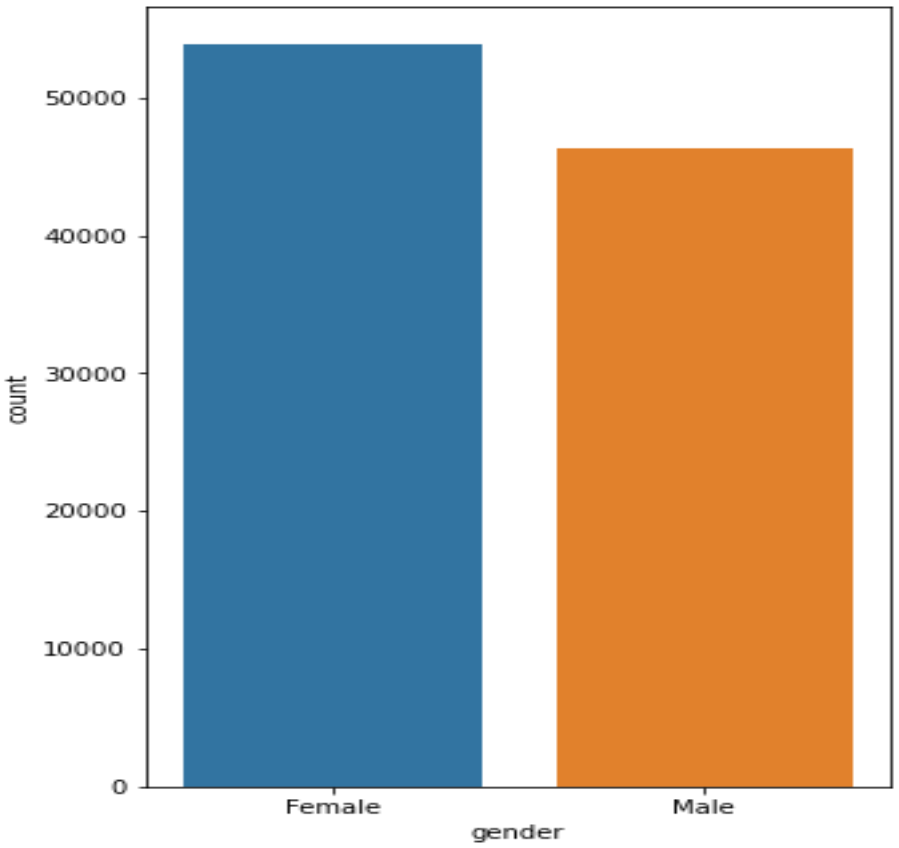


- **Count of Race Column:**



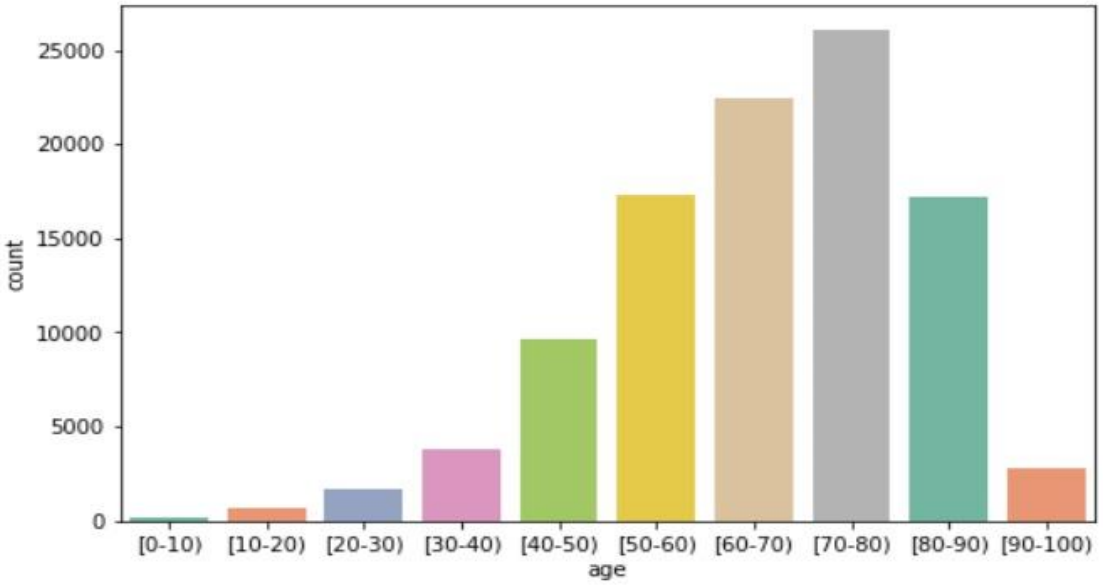
In column Race there are 5 sub-categories present, and those 5 categories have imbalancedata.

- **Count of Gender Column:**



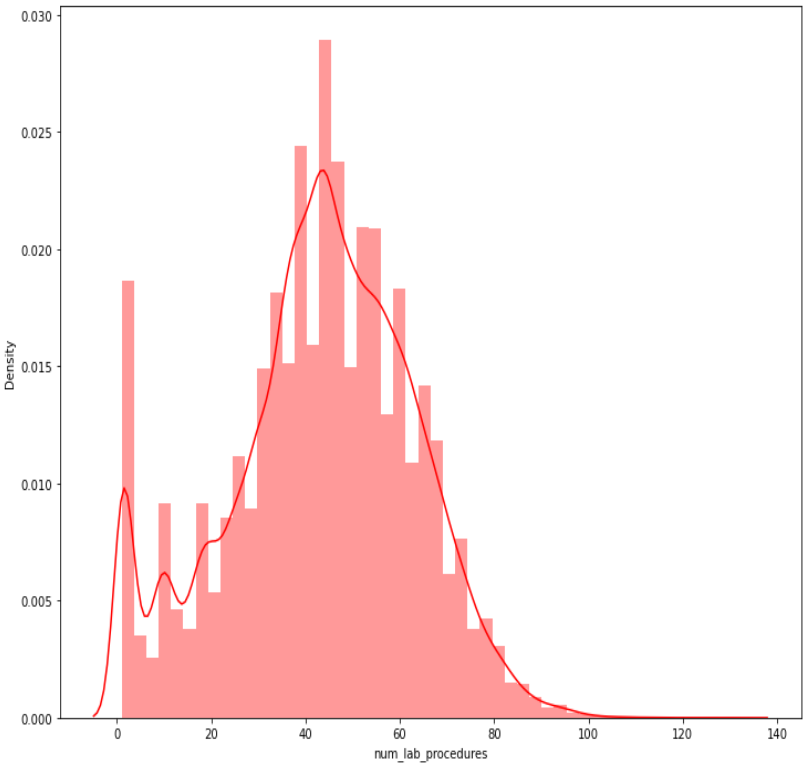
In column gender there are 2 sub-categories present.

- **Count of age Column:**



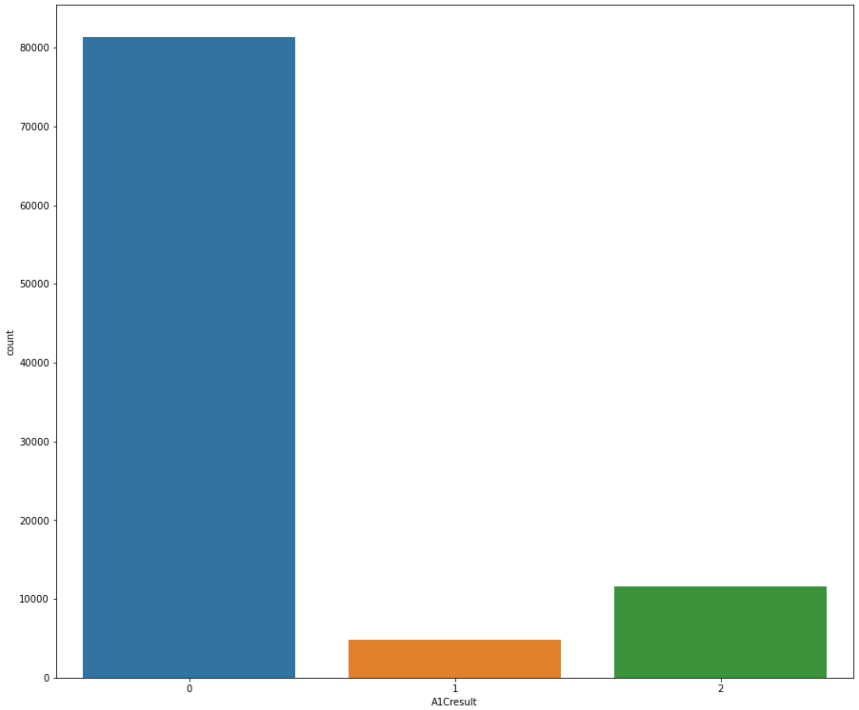
In column age there are 10 sub-categories present. As it is visible that the patient with the age greater than 50 are most prominent to Hospitalization.

- **Count of num_lab_procedures Column:**



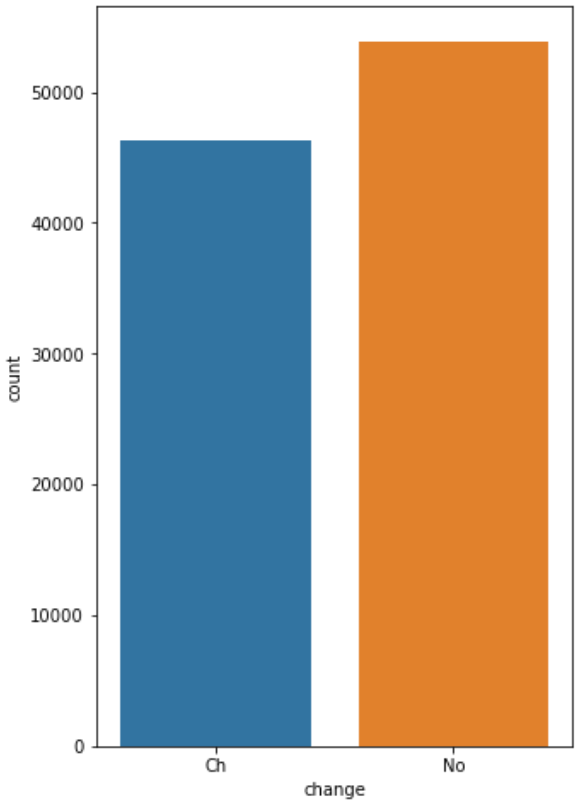
In columns num_lab_procedures, the distribution of the mentioned feature is centered around 43.

- **Count of A1C result Column:**



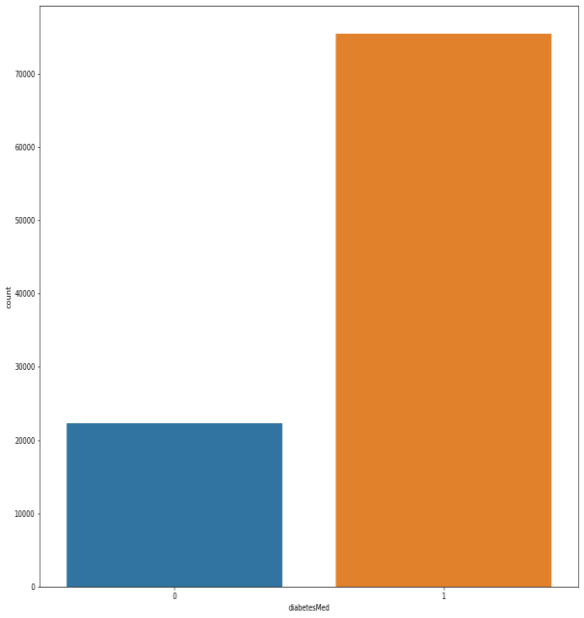
There 3 categories in A1Cresult which indicated the maximum percentage of the patients have not been tested for the A1C test. Where 0 represents the Test has not been performed, 1 represents that the result is Normal, whereas 2 represents a Pre-diabetic and Diabetic condition.

- **Count of Change Column:**



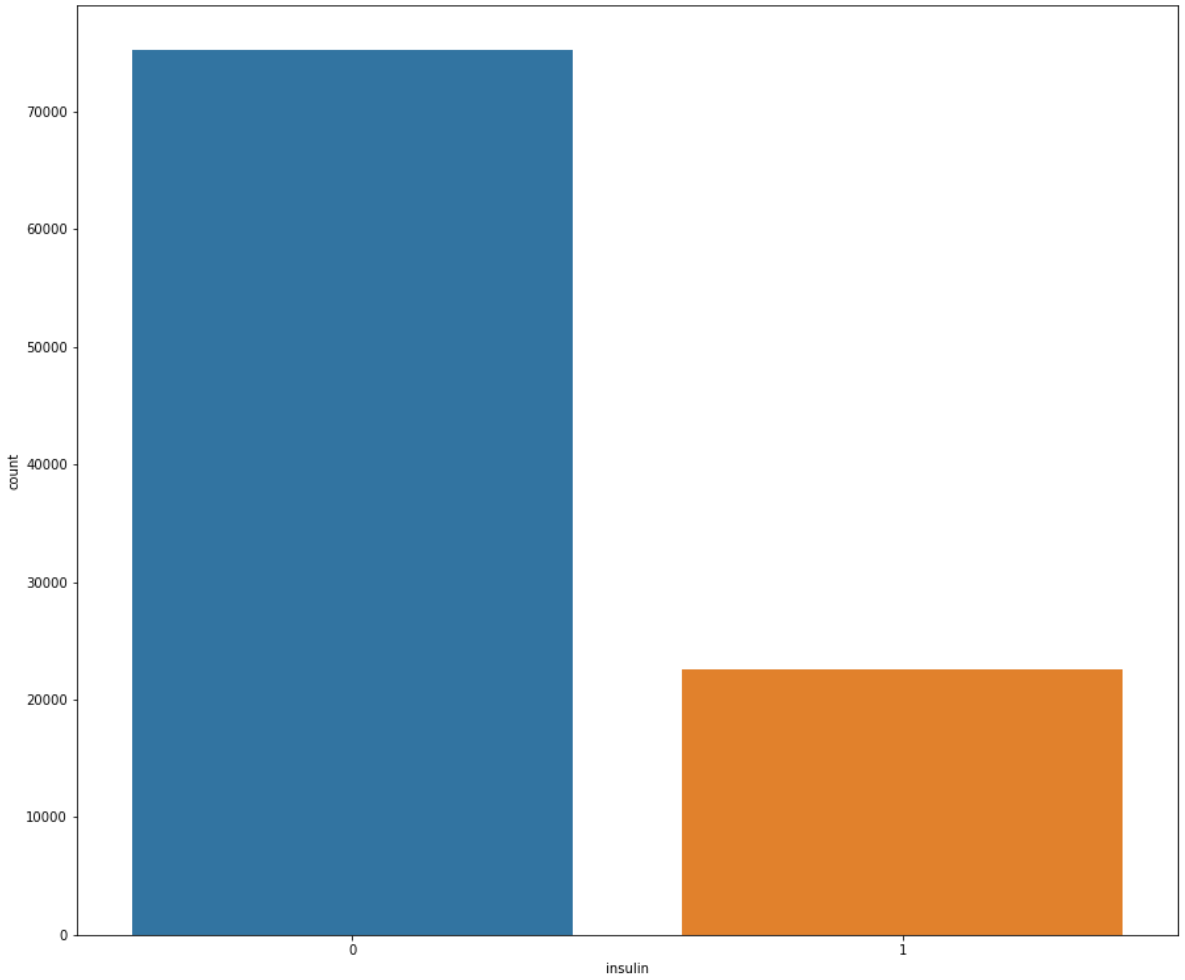
In the mentioned column the plot represents the change in the dosages of the prescribed medicine for the Diabetic Patient.

- **Count of diabeteMed Column:**



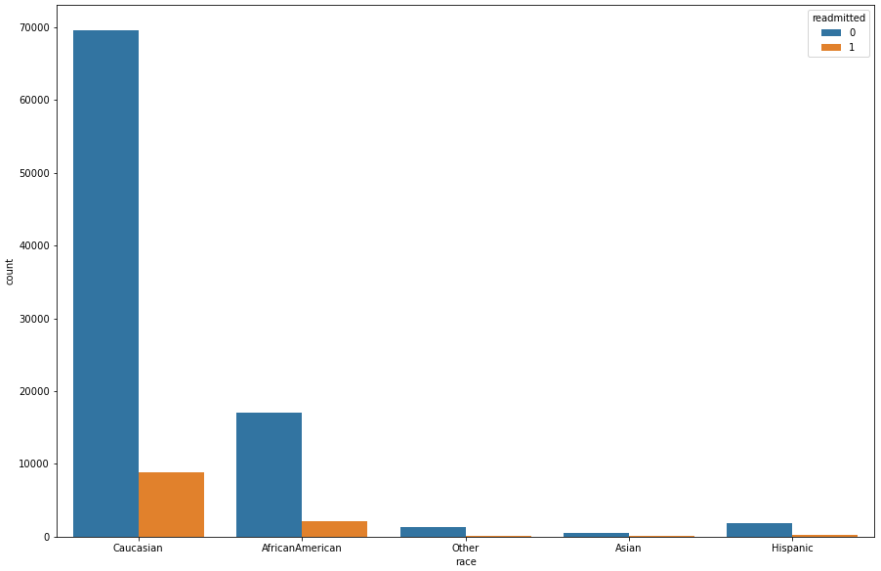
In the above mentioned column 0 denotes that no diabetes medicine is prescribed to the patient whereas 1 denotes that diabetes medicine has been prescribed to the patient.

- **Count of Insulin Column:**

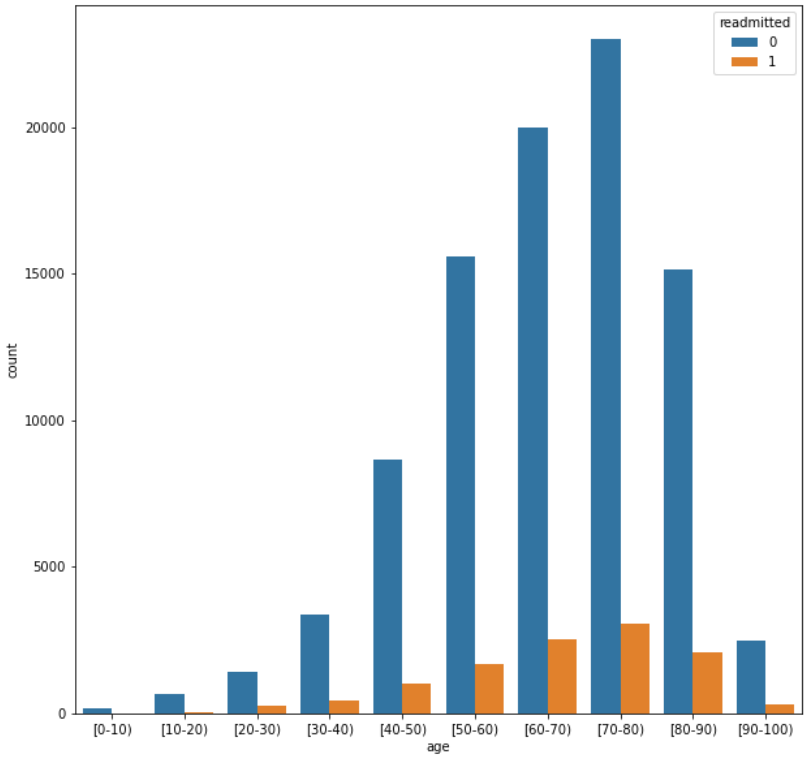


In the above mentioned plot the column 0 denotes that no Insulin is prescribed to the patient whereas 1 denotes that Insulin is prescribed to the patient.

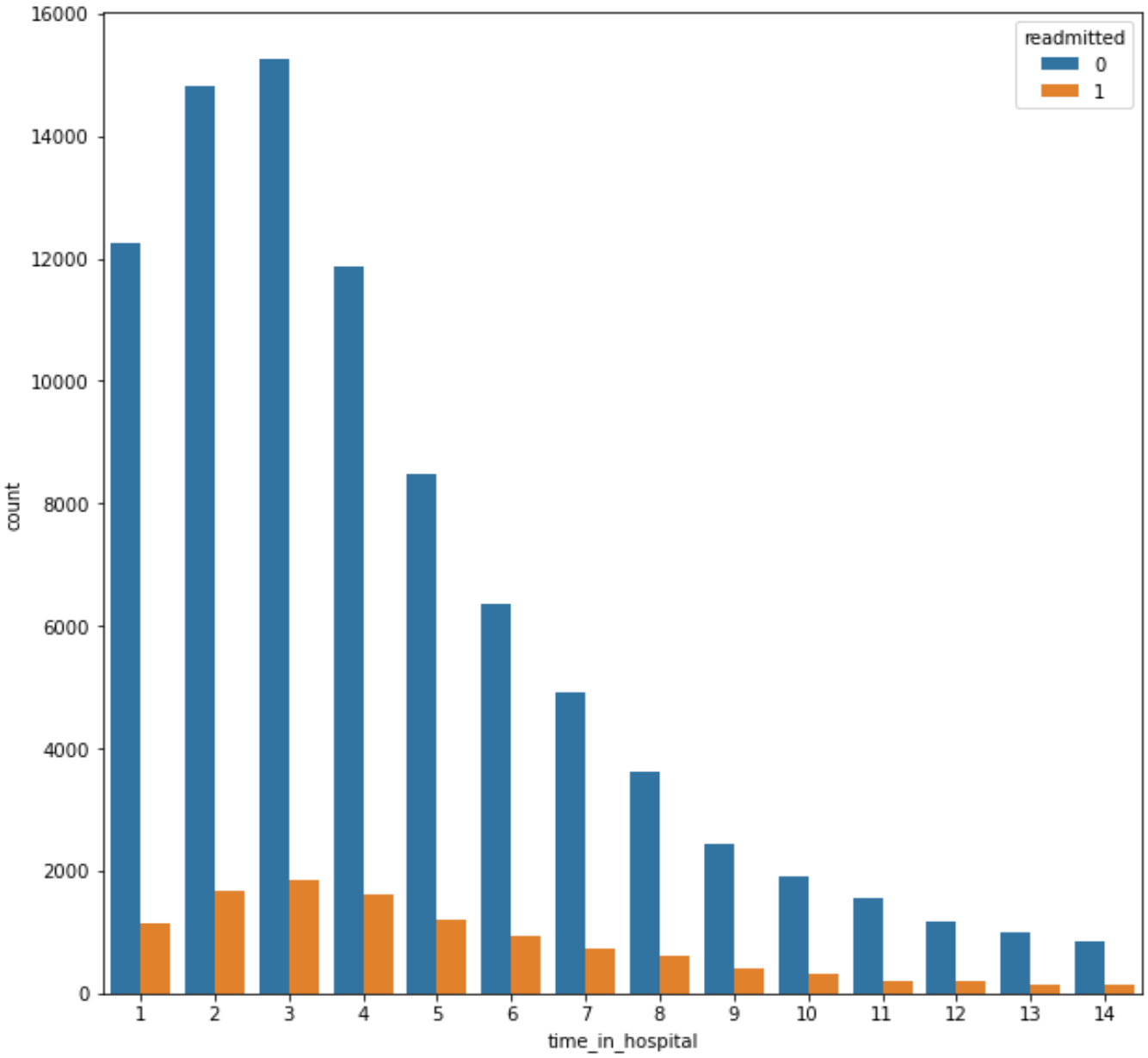
3.3 Bivariate Analysis:



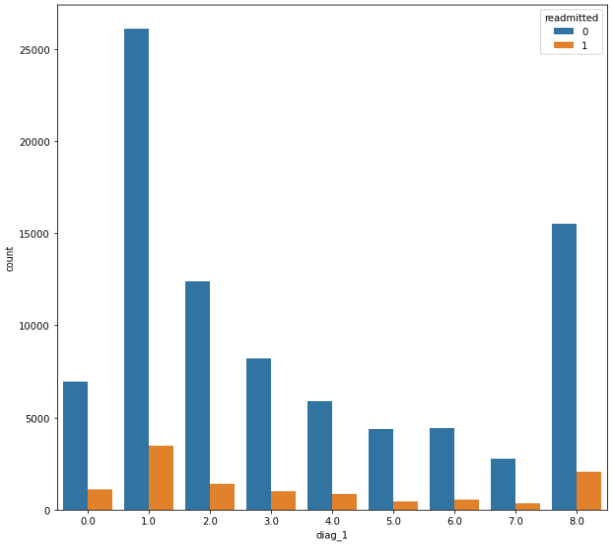
INFERENCE:- The Bar plot shows that the majority of the patient who are getting Re-admitted are Caucasian. Which can be due to the demographic distribution of the region.

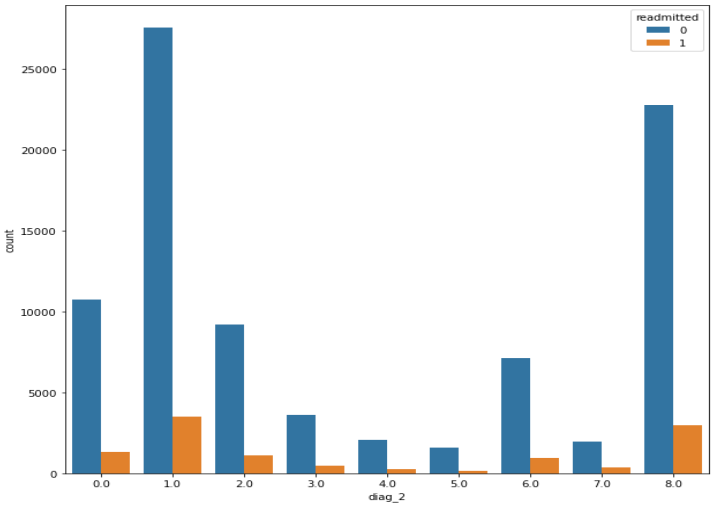


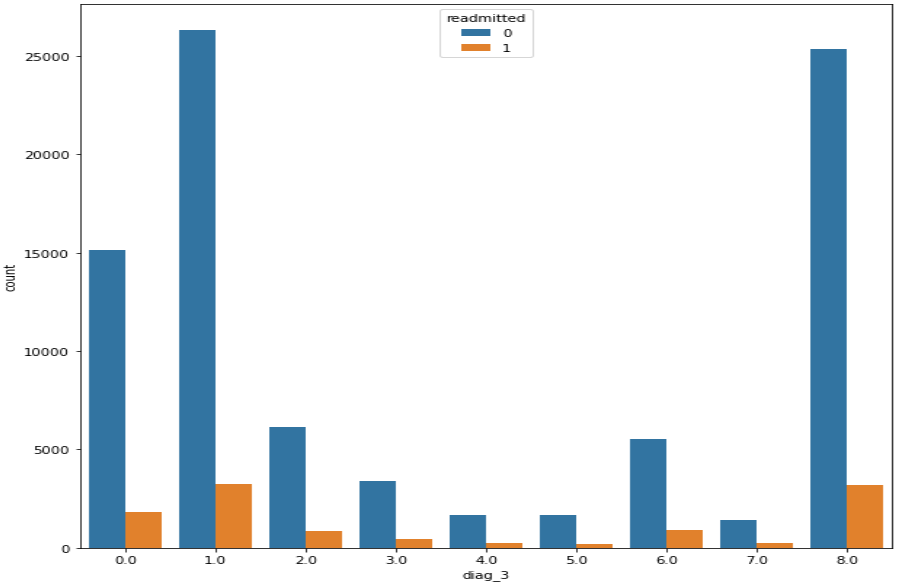
INFERENCE:- The Bar plot shows that the re-admission rate is higher in the senior patients than the patients in the lower age.



INFERENCE:- The plot shows that the maximum patients who are admitted for 3 days have greater chances to get Re-admitted again. As the day spent in the hospital increases the re-admission rate goes down.



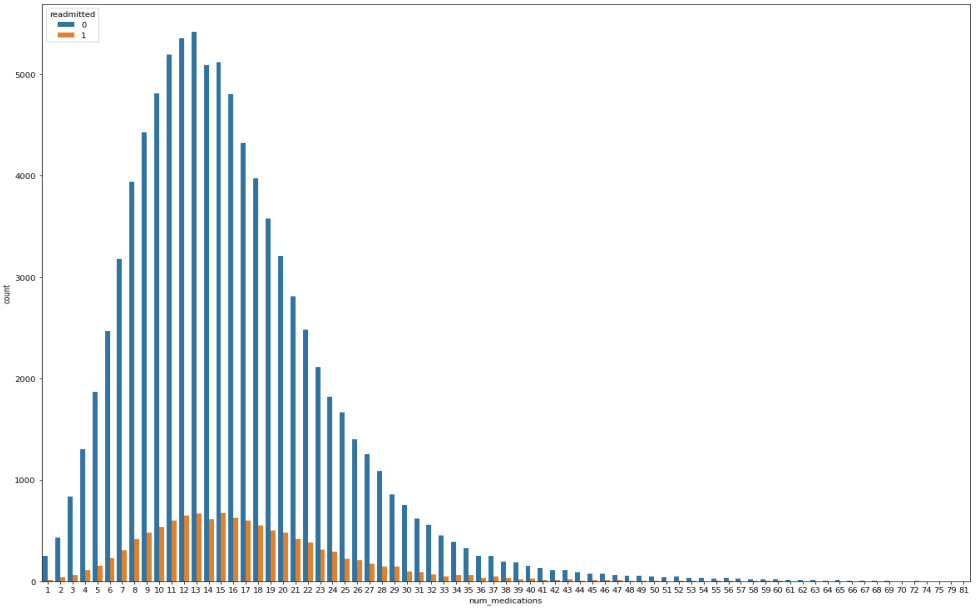




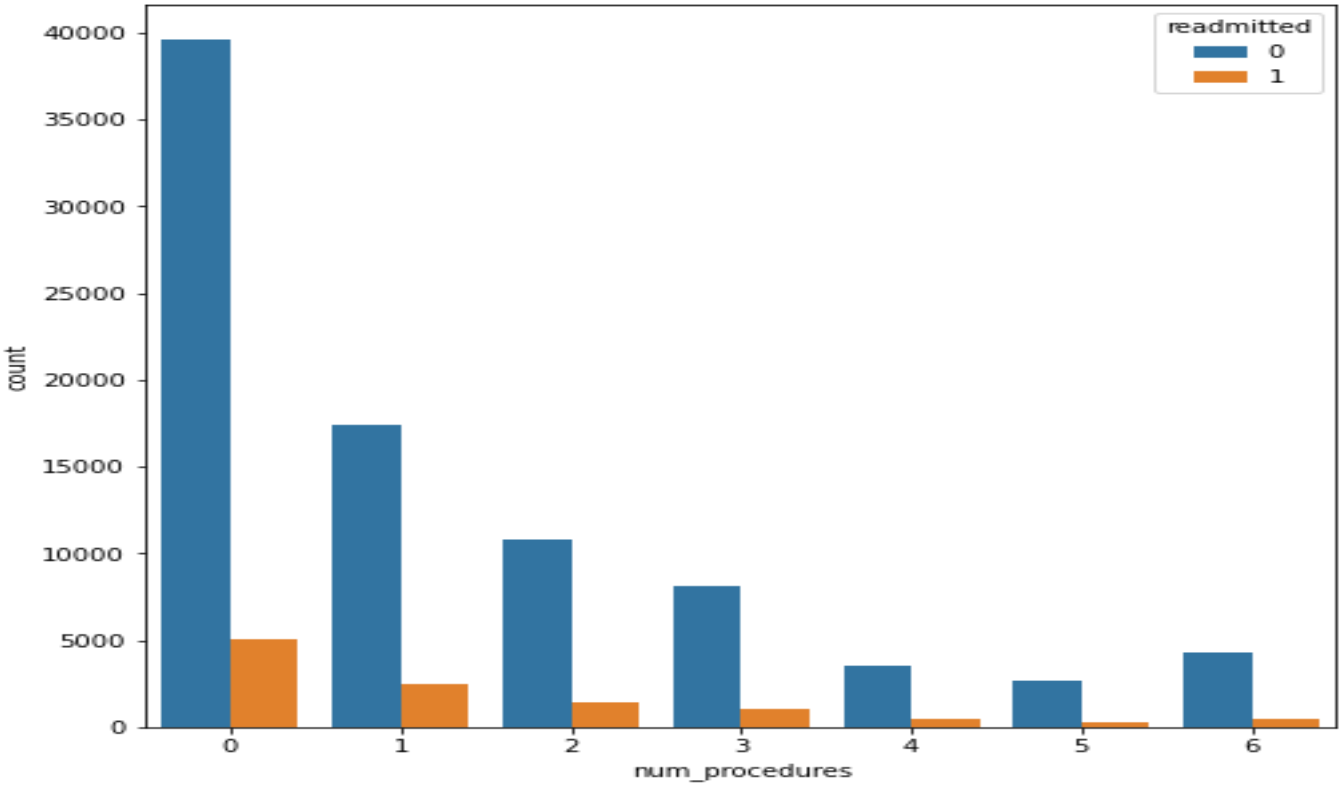
INFERENCE:- In the above plot the numbers in x-axis represents following categories.

Diabetes:0
Circulatory:1
Respiratory:2
Digestive:3
Injury:4
Musculoskeletal:5
Genitourinory:6
Neoplasms:7
Others:8

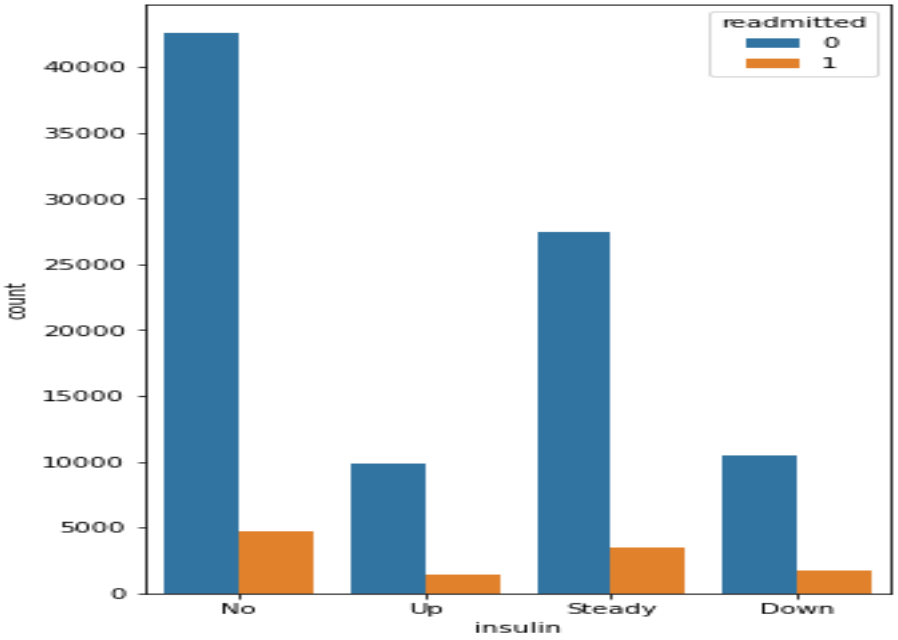
We can see that the patient diagnosis with Circulatory diseases have more chances to get Re-admitted.



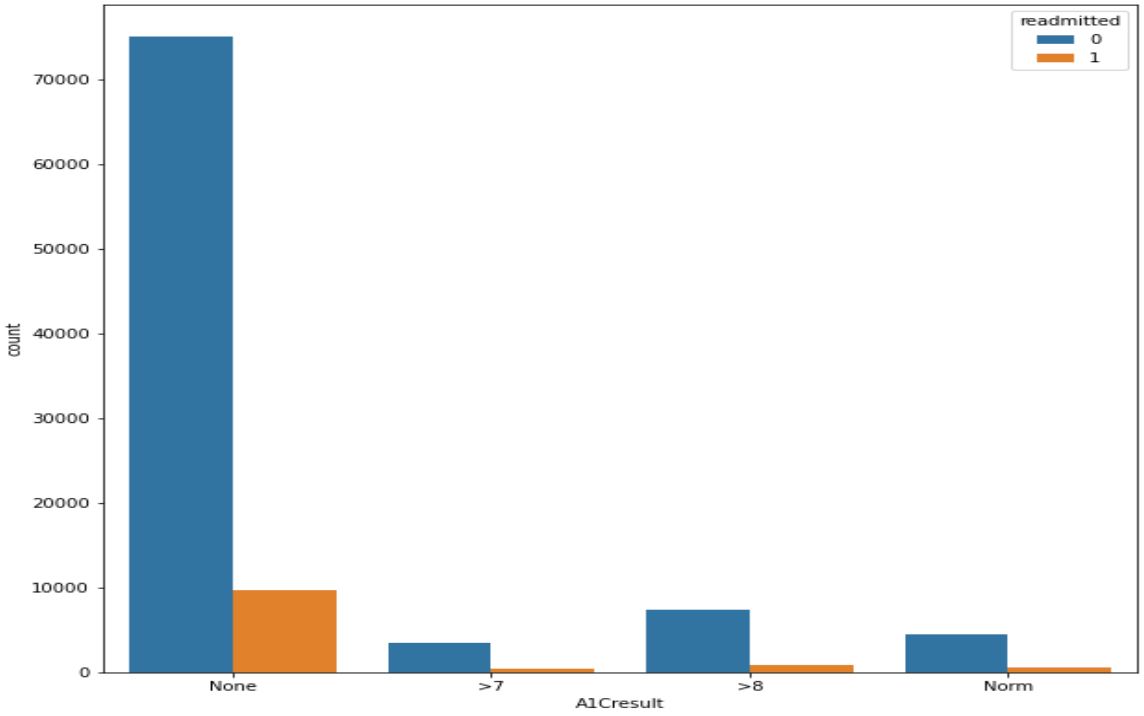
INFERENCE:- As num_medication increases from 1 to 15, the re-admission rate also increases, as if the num_medication increases further more than 15 the re-admission decreases.



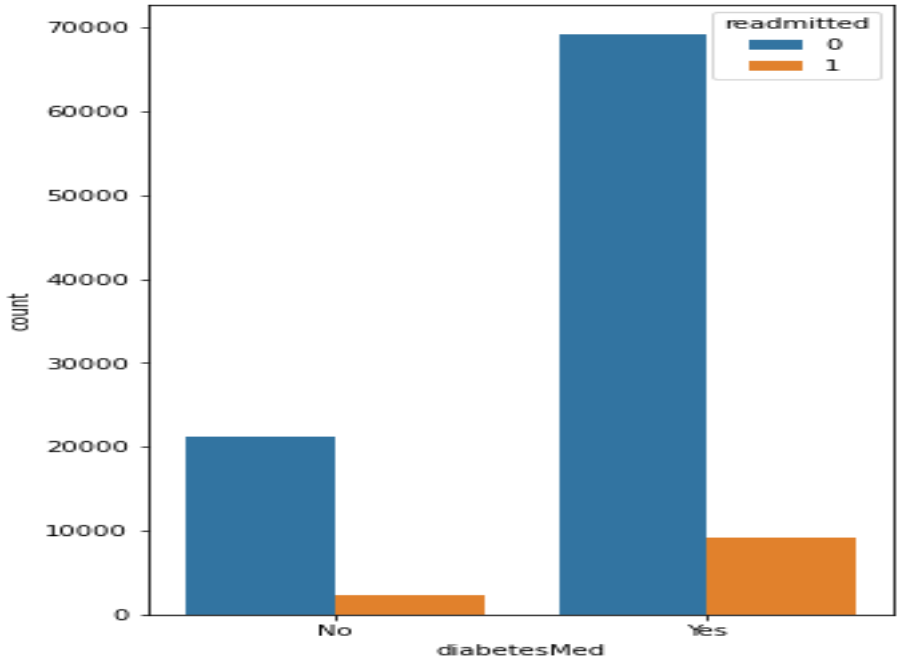
INFERENCE:- As the num_procedures increases the re-admission rate decreases.



INFERENCE:- The above plot shows the relationship between the Insulin dosages and the Re-admission rate.



INFERENCE:- The above plot shows the relationship between the A1Cresult test and the Re-admission rate.

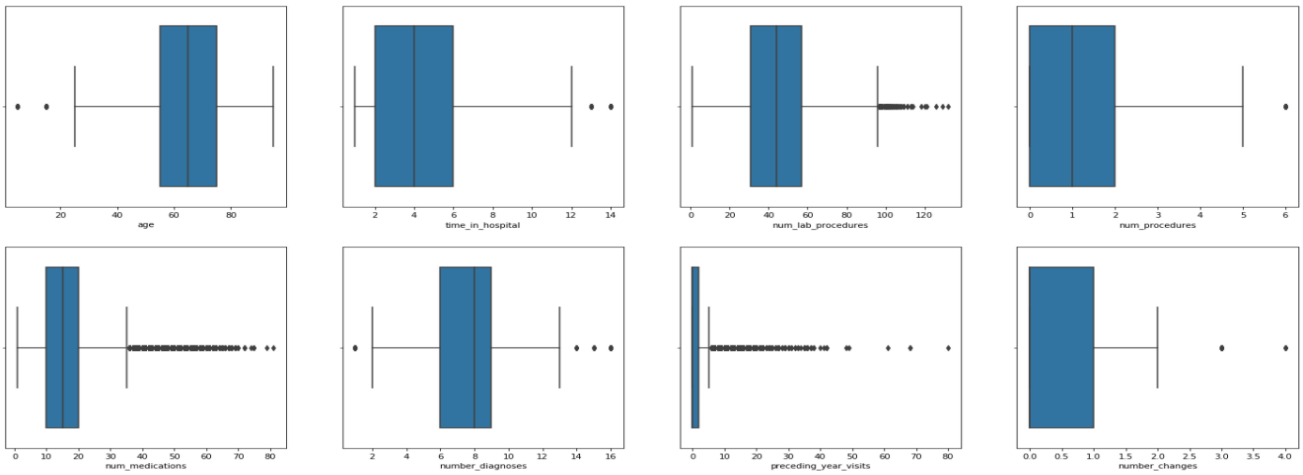


INFERENCE:- The above plot shows the relationship between the diabetes medication and the Re-admission rate.

3.4 Presence of outliers

Outliers, being the most extreme observations, may include the sample maximum or sample minimum, or both, depending on whether they are extremely high or low. However the sample maximum and minimum are not always outliers because they may not be unusually far from other observations.

Box plots show the overall spread of the data while plotting a data point for outliers. This physical point allows their specific values to be easily identified and compared among samples. We generally identify outliers with the help of boxplot, so here box plot shows some of the data points outside the range of the data.



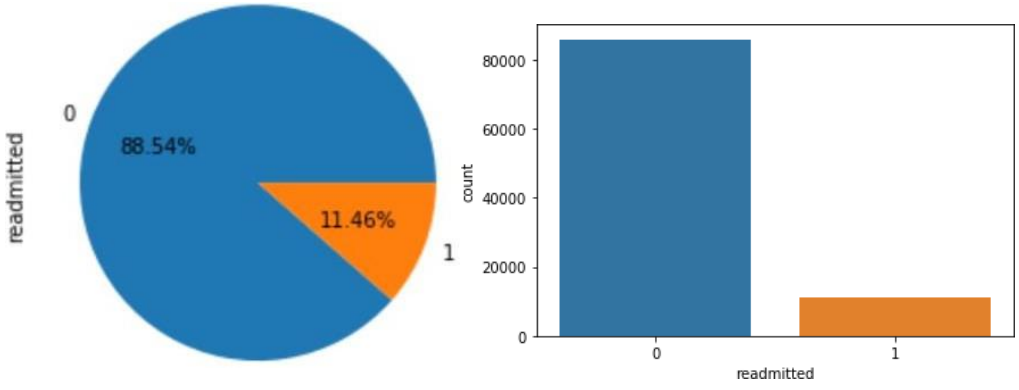
Outliers - Boxplot

- We can see that there are outliers in all of the plots above.
- Age has outliers having low value indicating that its distribution is left skewed.
- Time, number of lab procedures, number of procedures and preceding year visits have outliers with high values making the distribution right skewed.

3.5 Class imbalance and its treatment

Class imbalance: This is a scenario where the number of observations belonging to one class is significantly lower than those belonging to the other classes.

A standard machine learning algorithm like Decision Tree and Logistic Regression have a bias towards classes which have number of instances. They tend to only predict the majority class data. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class.



Class Imbalance

Imbalance Ratio ~ 1: 8

Treatment:

- **Oversampling:** Oversampling methods duplicate examples in the minority class or synthesize new examples from the examples in the minority class. Some of the more

widely used and implemented oversampling methods include Random Oversampling and Synthetic Minority Oversampling Technique (SMOTE). In some cases, seeking a balanced distribution for a severely imbalanced dataset can cause affected algorithms to overfit the minority class, leading to increased generalization error. The effect can be better performance on the training dataset, but worse performance on the holdout or test dataset. The increase in the number of examples for the minority class, especially if the class skew was severe, can also result in a marked increase in the computational cost when fitting the model, especially considering the model is seeing the same examples in the training dataset again and again.

- **Undersampling:** Undersampling methods delete or select a subset of examples from the majority class. Some of the more widely used and implemented undersampling methods include: Random Undersampling, Condensed Nearest Neighbor Rule (CNN), Near Miss Undersampling, Tomek Links Undersampling, Edited Nearest Neighbors Rule (ENN), One-Sided Selection (OSS), Neighborhood Cleaning Rule (NCR). A limitation of undersampling is that examples from the majority class are deleted that may be useful, important, or perhaps critical to fitting a robust decision boundary. Given that examples are deleted randomly, there is no way to detect or preserve "good" or more information-rich examples from the majority class.

- **Cost Sensitive Algorithms:** Cost-sensitive algorithms are modified versions of machine learning algorithms designed to take the differing costs of misclassification into account when fitting the model on the training dataset. These algorithms can be effective when used on imbalanced classification, where the cost of misclassification is configured to be inversely proportional to the distribution of examples in the training dataset. There are many cost-sensitive algorithms to choose from, although it might be practical to test a range of cost-sensitive versions of linear, nonlinear, and ensemble algorithms. Some examples of machine learning algorithms that can be configured using cost-sensitive training include: Logistic Regression, Decision Trees, Support Vector Machines, Random Forest, LightGBM, XGBoost.

Note: Keeping in mind the disadvantages of sampling techniques, we have used cost sensitive algorithms to deal with class imbalance.

CHAPTER-4

Pre-Processing Data Analysis

As real-world medical data are often noisy, a particular focus will be led on pre-processing task handling both missing data and inconsistencies but also by reducing the dataset and optimizing it for further model deployment.

4.1 Missing values

The first step in cleaning the data consist of handling missing values. Missing values refers to the absence, voluntary or not, of data in a record. In this data missing values are mainly in the form of question marks (‘? ‘).

- We will drop columns **‘Weight’**, **‘Payer code’** and **‘Medical specialty’** columns as they have more than 40% of missing values.

```
#dropping columns with large number of missing values
df = df.drop(['weight','payer_code','medical_specialty'], axis = 1)
```

- In the **‘Gender’** column, there are only three missing values hence we drop them.
- Missing values of the column **‘**diag_1’ is replace with ‘Missing’ and Missing values of the column ‘diag_2’ and ‘diag_3’ are replaced with ‘Not required’ if diag_1 and diag_2 value are given respectively. After that we were left with only 1 observation with missing diagnoses in all the 3, hence that observation was dropped.

```
#diag_1,diag_2,diag_3
df=df.dropna(subset=['diag_1', 'diag_2','diag_3'])
```

- In the ‘**Race’** column, we drop the missing values since we have only 2.23% of missing values in that particular column. Given the large dataset, we can afford to drop them. Moreover, dropping them gave better performance when compared with baseline model.

```
df['race'] = df['race'].fillna(df['race'].mode()[0]) #replacing with mode
```

```
df['race'].value_counts()
```

```
Caucasian          78372
AfricanAmerican    19210
Hispanic            2037
Other               1506
Asian                641
Name: race, dtype: int64
```

4.2 Inconsistencies

Data inconsistencies compromise data integrity and alter the performance of the algorithm. As a result, the third cleaning step resides in addressing such bias in data. Based on the body of literature this particular set of data has some specific inconsistent features to be addressed.

- After doing the above pre-processing, we found that 3 features, named, “examide”, “citoglipton”, “metformin-rosiglitazone” have the same observation (“No”) for every record in the dataset. Such features will, as a result, be dropped from the analysis.
- Discharge Disposition refers to the person's location or status after admission in the healthcare center. Patients who died during their admission have no probability to be readmitted and should be hence excluded from the analysis. Therefore, all records with expired discharge were deleted. Moreover, patients discharged to hospice, referring to terminally ill patients’ care, were also omitted for the same reason.
- We will drop the category ‘new born’ from admission type id because the age corresponding to it give contradictory information. Similarly, we will also drop the category ‘delivery’ from admission source id because the age and gender corresponding to it give contradictory information. Moreover, these observations have counts of up to 10 which is not much.

4.3 Encoding

- We converted all the **medicine** columns into numerical ones by encoding them as:
  ‘Not Given’: -2,
  ‘Down’: -1,
  ‘No Change’: 0,
  ‘Up’: 1.
- We label encoded ‘**gender’** column as **Female** : 0, **Male** : 1.
- We label encoded columns ‘**change’** and ‘**diabetesMed’** as follows:

```
df['change'] = df['change'].replace({'No' : 0, 'Ch' : 1})
```

```
# encoding diabetes med
df['diabetesMed'] = df['diabetesMed'].replace('Yes', 1)
df['diabetesMed'] = df['diabetesMed'].replace('No', 0)
```

Encoding Binary Columns

- **Age** was given in classes, so we replaced them with their respective means.

- **Readmission**: The outcome we are looking at is whether the patient gets readmitted to the hospital within 30 days or not. So, with respect to our problem we will convert this into 2 categories i.e. No readmission or readmission within 30 days.

```
#readmission converting multiclass to binary class
df['readmitted'] = df['readmitted'].replace({'>30' : 0, 'NO' : 0, '<30' : 1})
```

- **A1C_result:-** A1c test result Indicates the range of the result or if the test was not taken. Values: ">8" if the result was greater than 8%, ">7" if the result was greater than 7% but less than 8%, "normal" if the result was less than 7%, and "none" if not measured.

```
df["A1Cresult"] = df["A1Cresult"].replace({"None":0,"Norm":1,">7":2,">8":2})
```

4.4 Feature Engineering

There are three new features which have been created:

1. **'service_utilization:** 'number_outpatient', 'number_emergency', 'number_inpatient' were combined together by adding them up to get total visits of the patient in the previous year and then dropping the combining features.

```
df['service_utilization'] = df['number_outpatient'] + df['number_emergency'] + df['number_inpatient'] #year_visits
```

```
df.drop(['number_outpatient','number_emergency','number_inpatient'],axis=1,inplace=True)
```

2. encounter_id and patient_nbr: these are just identifiers and not useful variables so they can be dropped from our future analysis.

```
#droping not useful variables
df.drop(['encounter_id','patient_nbr'],axis=1,inplace=True)
```

3. Power Transformer:- Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like. This is useful for modeling issues related to heteroscedasticity (non-constant variance), or other situations where normality is desired.
   Currently, PowerTransformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood.We have performed power transformation on our dataset and developed multiple model on the power transformed data.

```
from sklearn.preprocessing from PowerTransformer

pt = PowerTransformer()
powertrain = pt.fit_transform(X_train)
powertest = pt.transform(X_test)
```

Chapter-5

Model Building:

In machine learning, there's something called the **"No Free Lunch"** theorem which states that no one algorithm works best for every problem, and it's especially relevant for supervised learning (i.e. predictive modelling).

For example, you can't say that neural networks are always better than decision trees or vice-versa. There are many factors at play, such as the size and structure of your dataset.

As a result, you should **try many different algorithms for your problem**, while using a hold-out "test set" of data to evaluate performance and select the winner.

Model selection is a method for setting a blueprint to analyse data and then using it to measure new data. Selecting a proper model allows you to generate accurate results when making a prediction.

5.1 Split the data to train and test

The train_test_split function is for splitting a single dataset for two different purposes: training and testing. The training subset is for building your model and the testing subset is for using the model on unknown data to evaluate the performance of the model.

Parameters:

X, y: The first parameter is the dataset you're selecting to use.

test_size: This parameter specifies the size of the testing dataset. The default state suits the training size. It will be set to 0.25 if the training size is set to default.

random_state: Controls the shuffling applied to the data before applying the split. The default mode performs a random split using np.random. Alternatively, you can add an integer using an exact number.

Splitting the data using sklearn.model_selection (train_test_split): 70% as train data, 30% as test data and random state = 10.

```
X=df1.drop(['readmitted'], axis=1)
y=df1['readmitted']
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.30, random_state=10)
```

5.2 Modeling

Our choice of models is governed primarily by our aim to understand the most important factors, along with their relative effects on medication change and readmission. The models that we implemented include the following.

1) Logistic Regression: Logistic regression is the classification counterpart to linear regression. Predictions are mapped to be between 0 and 1 through the logistic function, which means that predictions can be interpreted as class probabilities. The models themselves are still "linear," so they work well when your classes are linearly separable.

The logistic regression method assumes that:

- The outcome is a binary variable like yes vs no, positive vs negative or 1 vs 0.
- There is a linear relationship between the logit of the outcome and each predictor variables. (Recall that the logit function is logit(p) = log(p/(1-p)), where p is the probabilities of the outcome.)
- There are no influential values (extreme values or outliers) in the continuous predictors.
- There is no high intercorrelations (i.e. multicollinearity) among the predictors.

Strengths: Outputs have a nice probabilistic interpretation, and the algorithm can also be regularized by penalizing coefficients with a tunable penalty strength to avoid overfitting. Logistic models can be updated easily with new data using stochastic gradient descent.

Weaknesses: Logistic regression tends to underperform when there are multiple or non-linear decision boundaries. They are not flexible enough to naturally capture more complex relationships.

Logistic Regression (Base model on Imbalanced Data)

Logistic Regression

```
pipe = Pipeline((
("pt", PowerTransformer()),
("lr", LogisticRegression()),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

Above code has following information:

```
[[25954     0]
 [ 3388     0]]
              precision    recall  f1-score   support

           0       0.88      1.00      0.94     25954
           1       0.00      0.00      0.00      3388

    accuracy                           0.88     29342
   macro avg       0.44      0.50      0.47     29342
weighted avg       0.78      0.88      0.83     29342
```

Logistic Regression (Base model on Balanced Data)

Logistic Regression Balanced

```
pipe = Pipeline((
("pt", PowerTransformer()),
("lr", LogisticRegression(class_weight='balanced')),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

Above code has following information:

```
[[15518 10436]
 [ 1312  2076]]
              precision    recall  f1-score   support

           0       0.92      0.60      0.73     25954
           1       0.17      0.61      0.26      3388

    accuracy                           0.60     29342
   macro avg       0.54      0.61      0.49     29342
weighted avg       0.83      0.60      0.67     29342
```

2) **Decision Trees:** A Decision Tree is a simple representation for classifications and regression. It is a Supervised Machine Learning where the data is continuously split according to a certain parameter. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. In Decision Tree as we have no probabilistic model, but just binary split, we don't need to make any assumption at all.

Strengths: As with regression, classification tree ensembles also perform very well

in practice. They are robust to outliers, scalable, and able to naturally model non-linear decision boundaries thanks to their hierarchical structure.

Weaknesses: Unconstrained, individual trees are prone to overfitting, but this can be alleviated by ensemble methods.

Decision Tree Model (Base model on Imbalanced Data)

Decision Tree

```
pipe = Pipeline((
("pt", PowerTransformer()),
("dt", DecisionTreeClassifier(random_state=10)),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

Above code has following information:

```
[[22437  3517]
 [ 2762   626]]
              precision    recall  f1-score   support

           0       0.89      0.86      0.88     25954
           1       0.15      0.18      0.17      3388

    accuracy                           0.79     29342
   macro avg       0.52      0.52      0.52     29342
weighted avg       0.81      0.79      0.80     29342
```

Decision Tree Model (on Balanced Data)

Decision Tree Balanced

```
pipe = Pipeline((
("pt", PowerTransformer()),
("dt", DecisionTreeClassifier(random_state=10,class_weight='balanced')),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

Above code has following information:

```
[[23143  2811]
 [ 2919   469]]
              precision    recall  f1-score   support

           0       0.89      0.89      0.89     25954
           1       0.14      0.14      0.14      3388

    accuracy                           0.80     29342
   macro avg       0.52      0.52      0.52     29342
weighted avg       0.80      0.80      0.80     29342
```

3) **Random Forests:** Random Forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the "bagging" method. By considering more than one decision tree and then doing a majority voting, random forests helped in being more robust predictive representations than trees as in the previous case. It has no model underneath, and the only assumption that it relies is that sampling is representative. But this is usually a common assumption.

   Strengths: Random forest can solve both type of problems that is classification and regression and does a decent estimation at both fronts. It can handle thousands of input variables and identity most significant variables so it is considered as one of the dimensionality reduction method. Further, the model outputs importance of variable, which can be a very handy feature. It has an effective method for estimating missing data and maintains accuracy when large proportion of the data are missing. It has methods for balancing errors in data sets where classes are imbalanced.

   Weaknesses: It surely does a good job at classification but not as for regression problem as it does not give precise continuous nature prediction. In case of regression, it doesn't predict beyond the range in the training data, and that they may over fit data sets that are particularly noisy. Random forest can feel like a black box approach as we have very little control on what the model does. You can at best try different parameters and random seeds.

Random Forest Tree Model (Base model on Imalanced Data)

Random Forest

```
pipe = Pipeline((
("pt", PowerTransformer()),
("rf", RandomForestClassifier(random_state=10)),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

Above code has following information:

```
[[25948     6]
 [ 3371    17]]
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     25954
           1       0.74      0.01      0.01      3388

    accuracy                           0.88     29342
   macro avg       0.81      0.50      0.47     29342
weighted avg       0.87      0.88      0.83     29342
```

Random Forest Tree Model (On Balanced Data)

Random Forest Balanced

```
pipe = Pipeline((
("pt", PowerTransformer()),
("rf", RandomForestClassifier(random_state=10,class_weight='balanced')),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

Above code has following information:

```
[[25947     7]
 [ 3383     5]]
              precision    recall  f1-score   support

           0       0.88      1.00      0.94     25954
           1       0.42      0.00      0.00      3388

    accuracy                           0.88     29342
   macro avg       0.65      0.50      0.47     29342
weighted avg       0.83      0.88      0.83     29342
```

4) **XGBoost Classifier:** XGBoost stands for extreme Gradient Boosting.XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

Strengths: XGBoost is fast when compared to other implementations of gradient boosting. It dominates structured or tabular datasets on classification and regression predictive modeling problems.

Weaknesses: The implementation of XGBoost is relatively. Therefore, it lacks scalability. It also uses a lot of memory.

XG Boost Model (Base model on Imalanced Data)

XG Boost

```
pipe = Pipeline((
("pt", PowerTransformer()),
("gb", XGBClassifier()),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

Above code has following information:

```
[[25885    69]
 [ 3344    44]]
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     25954
           1       0.39      0.01      0.03      3388

    accuracy                           0.88     29342
   macro avg       0.64      0.51      0.48     29342
weighted avg       0.83      0.88      0.83     29342
```

XG Boost Model (On Balanced Data)

XG Boost Balanced

```
pipe = Pipeline((
("pt", PowerTransformer()),
("gb", XGBClassifier(scale_pos_weight=7.73)),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

Above code has following information:

```
[[18255  7699]
 [ 1747  1641]]
              precision    recall  f1-score   support

           0       0.91      0.70      0.79     25954
           1       0.18      0.48      0.26      3388

    accuracy                           0.68     29342
   macro avg       0.54      0.59      0.53     29342
weighted avg       0.83      0.68      0.73     29342
```

5) LightGBM Classifier: Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

Strengths:

- **Faster training speed and higher efficiency**: Light GBM use histogram-based algorithm i.e. it buckets continuous feature values into discrete bins which fasten the training procedure.
- **Lower memory usage:** Replaces continuous values to discrete bins which result in lower memory usage.
- **Better accuracy than any other boosting algorithm:** It produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy. However, it can sometimes lead to overfitting which can be avoided by setting the max_depth parameter.
- **Compatibility with Large Datasets:** It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST. Parallel learning supported.

Weaknesses: It can overfit if there are less than 10,000 observations in the dataset.

LightGBM Model (Base model on Imalanced Data)

LightGBM

```
pipe = Pipeline((
("pt", PowerTransformer()),
("lg", LGBMClassifier()),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

Above code has following information:

```
[[25945     9]
 [ 3375    13]]
              precision    recall  f1-score   support

           0       0.88      1.00      0.94     25954
           1       0.59      0.00      0.01      3388

    accuracy                           0.88     29342
   macro avg       0.74      0.50      0.47     29342
weighted avg       0.85      0.88      0.83     29342
```

LightGBM Model (On Balanced Data)

LightGBM Balanced

```
pipe = Pipeline((
("pt", PowerTransformer()),
("lg", LGBMClassifier(class_weight='balanced')),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

LightGBM Tuned

```
pipe = Pipeline((
("pt", PowerTransformer()),
("lg", LGBMClassifier(objective='binary',max_bin=7,num_leaves=7,max_depth=4, random_state=5,class_weight='balanced')),
))
pipe.fit(X_train,y_train)
print("Training Accuracy")
print(pipe.score(X_train,y_train))
print("Testing Accuracy")
print(pipe.score(X_test,y_test))
```

Above code has following information:

```
[[16852  9102]
 [ 1433  1955]]
              precision    recall  f1-score   support

           0       0.92      0.65      0.76     25954
           1       0.18      0.58      0.27      3388

    accuracy                           0.64     29342
   macro avg       0.55      0.61      0.52     29342
weighted avg       0.84      0.64      0.71     29342
```

Output after tuning

```
[[16080  9874]
 [ 1336  2052]]
              precision    recall  f1-score   support

           0       0.92      0.62      0.74     25954
           1       0.17      0.61      0.27      3388

    accuracy                           0.62     29342
   macro avg       0.55      0.61      0.50     29342
weighted avg       0.84      0.62      0.69     29342
```

CHAPTER-6

Model Summary

	Model_Name	Train_Accuracy	Test_Accuracy	Train_f1	Train_precision	Train_recall	Train_auc_roc	Test_f1	Test_precision	Test_recall	Test_auc_roc
0	Logistic Regression Base	0.885807	0.884534	0.000000	0.000000	0.000000	0.500000	0.000000	0.000000	0.000000	0.500000
1	Logistic Regression Balanced	0.592320	0.599618	0.255152	0.161224	0.611282	0.600653	0.261132	0.165921	0.612751	0.605327
2	Decision Tree	0.784760	0.786006	0.999936	1.000000	0.999872	0.999936	0.166246	0.151098	0.184770	0.524630
3	Decision Tree Balanced	0.803514	0.804717	0.999936	0.999872	1.000000	0.999992	0.140672	0.142988	0.138430	0.515061
4	Decision Tree Tuned	0.606123	0.620374	0.264433	0.169544	0.600537	0.610665	0.263440	0.169749	0.587957	0.606281
5	Random Forest	0.885763	0.884909	0.999872	1.000000	0.999744	0.999872	0.009968	0.739130	0.005018	0.502393
6	Random Forest Balanced	0.885705	0.884466	0.999808	0.999872	0.999744	0.999864	0.002941	0.416667	0.001476	0.500603
7	Random Forest Tuned	0.636724	0.671972	0.254357	0.171074	0.495651	0.593023	0.257617	0.174376	0.492916	0.594131
8	XG Boost	0.884551	0.883682	0.134460	0.964164	0.072269	0.535961	0.025136	0.389381	0.012987	0.505164
9	XG Boost Balanced	0.683712	0.678072	0.405011	0.273833	0.777437	0.755830	0.257857	0.175696	0.484357	0.593858
10	XG Boost Tuned	0.624220	0.628110	0.282664	0.181798	0.634945	0.633278	0.268044	0.173437	0.589728	0.611424
11	LightGBM	0.885865	0.884670	0.019494	0.939024	0.009849	0.504883	0.007625	0.590909	0.003837	0.501745
12	LightGBM Balanced	0.636431	0.640958	0.321068	0.208472	0.698133	0.678211	0.270682	0.176811	0.577037	0.613170
13	LightGBM Tuned	0.612287	0.617954	0.270469	0.172549	0.625352	0.619379	0.267990	0.172061	0.605667	0.612612

From the above tabular columns, we can infer that LightGBM model is performing well comparing to other models and giving 63% accuracy in training dataset and 64% accuracy in test dataset.

CHAPTER-7

Business Recommendations & Future enhancements

7.1 Conclusion

Hospital admission is an important factor that has high influence on economy, it could be disruptive to the patients and costly to healthcare system. The objective of this project was to develop a predictive risk model to identify patients with diabetes who are at a high risk of hospital readmission.

By analyzing key factors we can identify important factors or methods using machine learning that could help us to identify the medical condition of the diabetic patient, which impacts the all-purpose readmission of a patient with diabetes within 30 days of discharge and comparing different classification models that predict readmission and evaluating the best model. In this project, the problem of predicting the risk of readmission was framed as a binary classification problem and several available prediction models were developed and evaluated.

This study has assessed how various data preprocessing techniques such as feature selection, missing value imputation and class balancing techniques may impact the results of prediction modeling using readmission for patients with a diabetes diagnosis as the context for the analysis. Then, various predictive models like Logistic Regression and Decision Tree were applied to this improved dataset (after pre-processing) to obtain risk of readmission predictions accuracy. The impact of different pre-processing choices was assessed on various performance metrics like Area under Curve (AUC), Precision, Recall and Accuracy. This study offers empirical evidence that most proposed models with selected pre-processing techniques significantly outperform the baseline methods (without any pre-processing) with respect to selected evaluation criteria.

In this study, we evaluated various machine learning models to predict readmissions of high-risk patients. Extending prior research, we performed class balancing considering the skewness of data. We have observed that we are getting high accuracy on our LGBM model. Some of the key features that drove readmissions are number of preceding year visits, length of stay, number of medications and number of diagnosis. Extending this research, we plan to further investigate the performance of classifiers with the goal to improve the accuracy of prediction of readmission risk. Further research is also warranted to explore the relevant feature space particularly with the variability of findings across research studies.

7.2 Business Recommendations

To summarize, these are the recommendations proposed to the business client:

- Medical facilities can take precautionary measures with patients during their initial admission by making A1C and Maximum Glucose Serum test compulsory and providing the treatment accordingly.
- Providing extra attention and care to high-risk patients.
- A follow-up with the discharged patients should be done to keep a track of their health and to counsel them from time-to-time.
- High-risk patients' current medicines' regime should be re-evaluated and the most effective medicines should be considered.
- Most Effective Medications :- Metformin, Glipizide, Insulin.
- The annual plans, financials and infrastructure / inventory should be planned accordingly by taking into account the predicted readmissions.

7.3 Limitations/ Challenges

One of the biggest challenges is that our Data consist of different terms of missing values, incomplete or inconsistent records, and high dimensionality understood not only by number of features but also their complexity. After doing preprocessing and feature engineering, our data was converted into encoded form, so it is also
a limitation for us.

1. Huge amount of Data
2. Different Imputation methods
3. Imbalanced Data
4. Reductants Features
5. Hyper parameter tuning

7.4 Future Scope

This research study has only targeted patients with diabetes. Readmission prediction model needs to be generated for other key health conditions also such as Lung disease, Liver disease etc. in Healthcare system supporting Global standards. In the future studies, planned and unplanned (emergency) readmissions needs to be considered.

Some of the very imperative features like family history, emotional stabilization (depression), economic status, living lifestyle standards and season of readmission need to be collected and analyzed. Performing a more exhaustive exploration approach in the given dataset could be more relevant to predict the risk of readmission.

Also, we can collect Subjective data by interviewing the patient to know more about the background information related to their medical condition. Thus, collecting these features could add a more significant information or intelligence to our model.
We can also develop stacked model and drill more significant features that has high potential to impact the classification. We can also change the classification threshold to note down the affect it has on our model performance, more specifically reducing the false negative to improve recall score.

References (Data set source/Journals/articles)

- Diabetes 130-US hospitals for years 1999-2008 Data Set provided by Centre for Clinical and Translational Research, Virginia Commonwealth University.

  Website: **https://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+1999-2008**

- Diabetes 130 US hospitals for years 1999-2008 Diabetes – readmission.

  Website: **https://www.kaggle.com/brandao/diabetes**

- Correction to: Hospital Readmission of Patients with Diabetes

  Website : **https://link.springer.com/article/10.1007/s11892-018-0989-1**

- The relationship between diabetes mellitus and 30-day readmission rates.

  Website: **https://clindiabetesendo.biomedcentral.com/articles/10.1186/s40842-016-0040-x**

- Beata Strack, Jonathan P. DeShazo, Chris Gennings, Juan L. Olmo, Sebastian Ventura, Krzysztof J. Cios, and John N. Clore, "Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records," BioMed Research International, vol. 2014, Article ID 781670, 11 pages, 2014.

  Website: **https://www.hindawi.com/journals/bmri/2014/781670/**

- Predictors of 30-day unplanned hospital readmission among adult patients with diabetes mellitus: a systematic review with meta-analysis

  Website: **https://drc.bmj.com/content/8/1/e001227**

- Readmission and Comprehension of Diabetes Education at Discharge (ReCoDED Study)

  Website: **https://diabetes.diabetesjournals.org/content/67/Supplement_1/147-LB**