CAPSTONE PROJECT GROUP 5

Final Report

Title: Identifying High Risk Patients for Hospital Readmission

Submitted towards the partial fulfilment of the criteria for award of PGPDSE by GLIM.

Mentored By:
Mr. Sravan
Malla

Submitted By:
Ms. Anita H.
Ms. Mrunal
Dalvi
Ms. Shweta Chavan
Mr. Rhushikesh Hanjankar
Mr. Aroop Ajaykumar

Techniques: Data wrangling, Predictive modelling

Tools: Python

Domain: Medical domain

Index

1) Abstract

Hospital readmission is considered a key metric in order to assess health centre performances. Indeed, readmissions involve different consequences such as the patient's health condition, hospital operational efficiency but also cost burden from a wider perspective. As the healthcare system moves toward value-based care, Centre for Medicare and Medicaid Services (CMS) has created many programs to improve the quality of care of patients. One of these programs is called the Hospital Readmission Reduction Program (HRRP), which reduces reimbursement to hospitals with above average readmissions. For those hospitals which are currently penalized under this program, one solution is to create interventions to provide additional assistance to patients with increased risk of readmission. Prediction of 30-day readmission for diabetes patients is therefore of prime importance.

In order to deal with readmission prediction, this study will propose a Linear model and Ensemble techniques on data collected from 130 US hospitals. More specifically, the pre-processing technique includes comprehensive data cleaning, data reduction, and transformation. Random Forest algorithm for feature selection and SMOTE algorithm for data balancing are some example of methods used in the proposed pre-processing framework. The performance of the designed model was found, in this regard, particularly balanced across different metrics of interest with accuracy and F-1 score of 94% and close to the optimal recall of 99%.

2) Introduction

Diabetes is one of the chronic non-communicable diseases that are on the rise with massive urbanization and a drastic change of lifestyle in many countries. Chronic complications of Diabetes include retinopathy, neuropathy, nephropathy, an increased risk of cardiovascular disease and major cardiac events including myocardial infarction and stroke. The high prevalence of Diabetes and its complications makes it a common condition in hospitalized patients. This leads to frequent admissions for procedures and interventions during which patients with Diabetes are reported to have longer lengths of stay, increased hospital complications, and mortality. It is expected to turn into the seventh most prevalent mortality factor by 2030 and millions of deaths could be prevented each year through better analytics. Therefore, diabetes is on the health agenda of most developed and developing countries. Healthcare industry collects and process diabetes patient medical data in huge volume, diverse structure, and real-time flow of data. With the rise of technology, both from the diagnosis and monitoring, storage and analysis, novel solutions are now available to better address challenges like non-invasive screening, tailor-made treatment, and hospital readmissions.

When assessing the quality of care delivered by a health centre, readmission is the metric of choice. It measures the number of patients that need to come back to the hospital after their initial discharge. The readmission can be classified into three broad categories such as unavoidable, planned, and unplanned. The unavoidable readmission that is highly predictable mostly due to the nature of the pathology or patient's condition (i.e. cancer phase IV, metastasis). Secondly in the planned readmission which is directly prescribed by the healthcare professional to the patient (i.e. check-up, transfusion). Lastly, the unplanned is defined as readmission that shouldn't have happened given the practitioner's diagnosis and could have been avoided if proper care was given to the patient post-discharge. Unavoidable and planned readmissions already are highly anticipated. However, predicting unplanned readmission is of prime interest due to its inherent uncertainty.

Unplanned readmission is the most useful type when evaluating the quality of care of a hospital as it highlights a practitioner's diagnosis or treatment error. Beyond being a core indicator of the quality of care, unplanned readmissions also constitute a financial problem for nations. Therefore, with a predictive model to assess unplanned readmission risk could optimize the quality of hospital services and state Medicare.

According to, readmissions occurring after 30 days have less correlation with the quality of care from the health centre and might be an impact due to external factors such as complications or patient's lifestyle. Numerous researches highlighted clear interest in 30-day unplanned readmission prediction models based on diabetes complications. However, predictability performance is quite low when dealing with unplanned readmission rates. Moreover, several researchers have proposed a predictive model for readmission in healthcare for all types of diseases and only limited work are dedicated to diabetes. As different pathologies have different conditions and behaviours, prediction on specific pathology subset would highly benefit the prediction model's performance.

The purpose of this study is to propose a prediction model for 30-day unplanned readmission among diabetes patients in US hospitals. The analysis will be based on risk factors such as a patient's demographics, admission details, diagnosis, and medical data. In a broader sense, the goal of the study is to allow health centres to better anticipate and address unplanned readmissions while improving their quality of care and cost efficiency.

3) Problem Statement

Predict if a patient with diabetes will be readmitted to the hospital within 30 days. As, if there is a patient getting readmitted within 30 days, the hospital faces incentive issues and also increases the financial expenses of the patient. Taking this into consideration, the aim is to correctly predict the readmission of a diabetic patient so that hospital administration can take prescriptive measures to prevent financial loss of a hospital and patient.

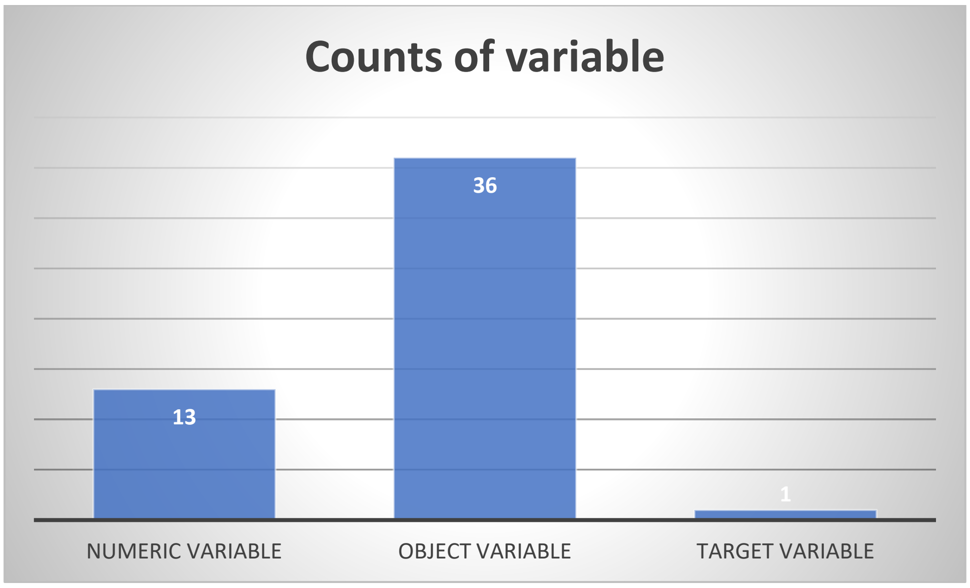
4) Literature Review

The data in this project is a classic example of Supervised Machine Learning, Classification Problem. In analytics, the classification problems are an important category where the outcome variable takes discrete variables. Its primary objective is Class Probability i.e. to predict the probability of an observation belonging to a particular class. Classification problems are generally of two types, namely Binary and Multinomial Classification. In this data, there are target observations belonging to different classes, thus is a multinomial classification. However, considering the problem statement and the business goal we convert it to a binary classification to further aid in correct model building.

As it is now a Binary Classification, we use Logistic Regression as base model. Then we move on to more complex models like Decision Tree, Random Forest, Bagging & Boosting along with Hyperparameter Tuning and further selecting the best from those depending on F1-score, accuracy, confusion matrix and other evaluation matrices.

5) Dataset information

The dataset contains the diabetic patient information who was admitted to a particular hospital. The data is of various patients who have been readmitted into the hospital numerously. Dataset comprises of 101766 rows and 50 columns. The data also consists of 13 integers and 37 object variables. The most important column we can see is readmitted, it tells us if a patient was readmitted to the hospital within 30 days, greater than 30 days or not readmitted. The data recorded over a span of 10 years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks.



Dataset Dictionary

List of features and their descriptions in the initial dataset. This dataset is also available at the website of Data Mining and Biomedical Informatics Lab at VCU (http://www.cioslab.vcu.edu/).

Sr.	Feature	Description
1	Encounter ID	Unique identifier of an encounter
2	Patient number	Unique identifier of a patient
3	Race	Values: Caucasian, Asian, African American, Hispanic, and other

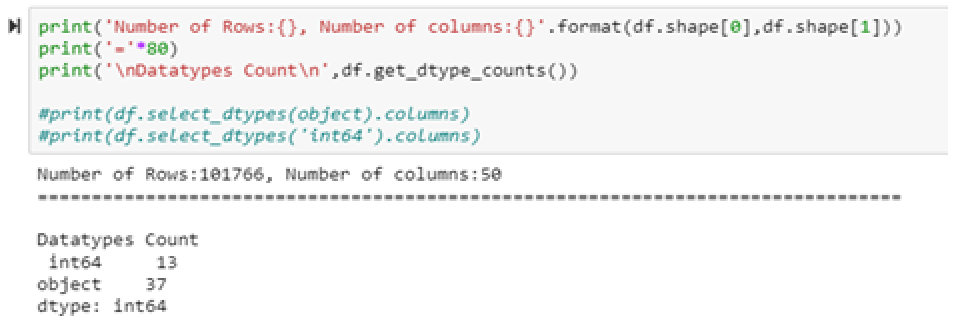
4	Gender	Values: male, female, and unknown/invalid
5	Age	Grouped in 10-year intervals: 0, 10), 10, 20), …, 90, 100)
6	Weight	Weight in pounds.
7	Admission type	Integer identifier corresponding to 9 distinct values, for example, emergency, urgent, elective, newborn, and not available
8	Discharge disposition	Integer identifier corresponding to 29 distinct values, for example, discharged to home, expired, and not available
9	Admission source	Integer identifier corresponding to 21 distinct values, for example, physician referral, emergency room, and transfer from a hospital
10	Time in hospital	Integer number of days between admission and discharge
11	Payer code	Integer identifier corresponding to 23 distinct values, for example, Blue Cross/Blue Shield, Medicare, and self-pay
12	Medical specialty	Integer identifier of a specialty of the admitting physician, corresponding to 84 distinct values, for example, cardiology, internal medicine, family/general practice, and surgeon
13	Number of lab procedures	Number of lab tests performed during the encounter
14	Number of procedures	Number of procedures (other than lab tests) performed during the encounter
15	Number of medications	Number of distinct generic names administered during the encounter
16	Number of outpatient visits	Number of outpatient visits of the patient in the year preceding the encounter
17	Number of emergency visits	Number of emergency visits of the patient in the year preceding the encounter
18	Number of inpatient visits	Number of inpatient visits of the patient in the year preceding the encounter
19	Diagnosis 1	The primary diagnosis (coded as first three digits of ICD9); 848 distinct values
20	Diagnosis 2	Secondary diagnosis (coded as first three digits of ICD9); 923 distinct values
21	Diagnosis 3	Additional secondary diagnosis (coded as first three digits of ICD9); 954 distinct values
22	Number of diagnoses	Number of diagnoses entered to the system
23	Glucose serum test result	Indicates the range of the result or if the test was not taken. Values: “>200,” “>300,” “normal,” and “none” if not measured
24	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.
25	Change of medications	Indicates if there was a change in diabetic medications (either dosage or generic name). Values: “change” and “no change”
26	Diabetes medications	Indicates if there was any diabetic medication prescribed. Values: “yes” and “no”
27	24 features for medications	For the generic names: metformin, repaglinide, nateglinide, chlorpropamide, glimepiride, acetohexamide, glipizide, glyburide, tolbutamide, pioglitazone, rosiglitazone, acarbose, miglitol, troglitazone, tolazamide, examide, sitagliptin, insulin, glyburide-metformin, glipizide-metformin, glimepiride-pioglitazone,

		metformin-rosiglitazone, and metformin-pioglitazone, the feature indicates whether the drug was prescribed or there was a change in the 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
28	Readmitted	Days to inpatient readmission. Values: “<30” if the patient was readmitted in less than 30 days, “>30” if the patient was readmitted in more than 30 days, and “No” for no record of readmission.

6) Data Exploration

6.1) Dimensions:

Dataset comprises of 101766 rows and 50 columns. With 13 integer datatype and 37 of object datatype.

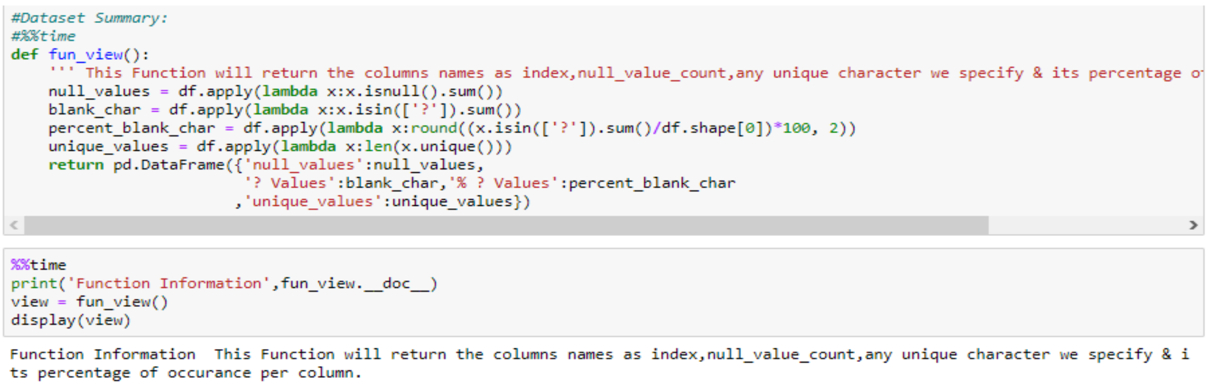


```
print('Number of Rows:{}, Number of columns:{}'.format(df.shape[0],df.shape[1]))
print('='*80)
print('\nDatatypes Count\n',df.get_dtype_counts())

#print(df.select_dtypes(object).columns)
#print(df.select_dtypes('int64').columns)
```

```
Number of Rows:101766, Number of columns:50
================================================================================

Datatypes Count
 int64     13
object    37
dtype: int64
```

6.2) Missing Value Analysis

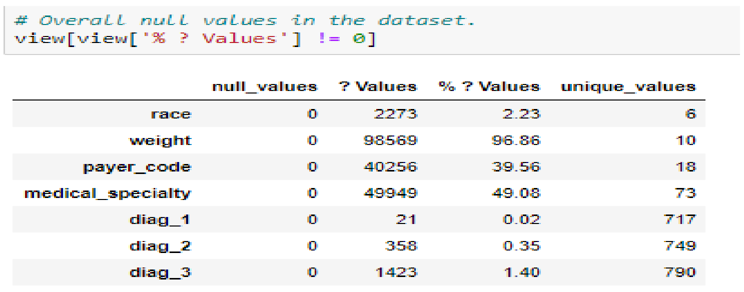
We created a function that will return the columns names as index, null value count, any unique character we specify & its percentage of occurrences, unique characters per column.



```
#Dataset Summary:
#%%time
def fun_view():
    ''' This Function will return the columns names as index,null_value_count,any unique character we specify & its percentage o
    null_values = df.apply(lambda x:x.isnull().sum())
    blank_char = df.apply(lambda x:x.isin(['?']).sum())
    percent_blank_char = df.apply(lambda x:round((x.isin(['?']).sum()/df.shape[0])*100, 2))
    unique_values = df.apply(lambda x:len(x.unique()))
    return pd.DataFrame({'null_values':null_values,
                         '? Values':blank_char,'% ? Values':percent_blank_char
                        ,'unique_values':unique_values})
```

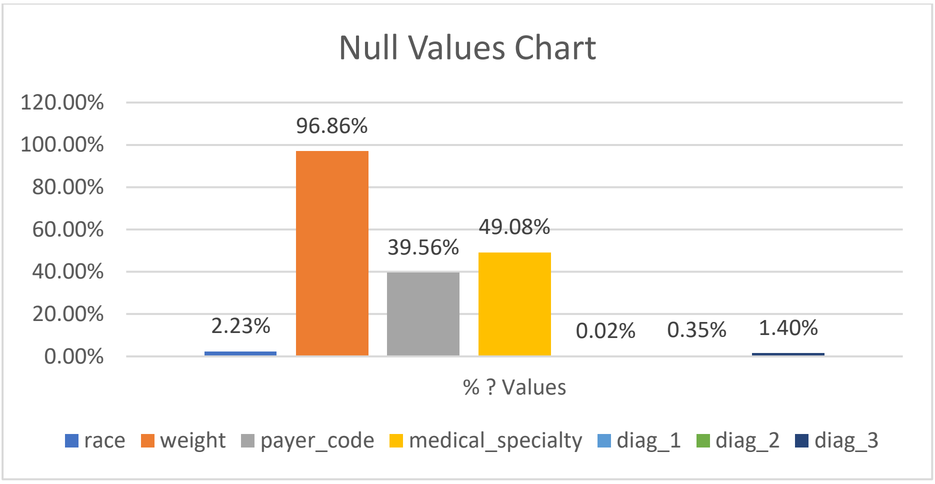
```
%%time
print('Function Information',fun_view.__doc__)
view = fun_view()
display(view)
```

```
Function Information  This Function will return the columns names as index,null_value_count,any unique character we specify & i
ts percentage of occurance per column.
```



```python
# Overall null values in the dataset.
view[view['% ? Values'] != 0]
```

	null_values	? Values	% ? Values	unique_values
race	0	2273	2.23	6
weight	0	98569	96.86	10
payer_code	0	40256	39.56	18
medical_specialty	0	49949	49.08	73
diag_1	0	21	0.02	717
diag_2	0	358	0.35	749
diag_3	0	1423	1.40	790



After initially looking at the dataset output it is observed that there are lot of "?" value in our dataset.

Columns weight, payer code and medical_speciality have the highest percentage of "?" values. We will drop weight column as it contains high number of null values.

The payer code is mode of payment, it will not affect the readmission rate hence we will drop this column. Medical specialty — It has 49% missing values (Medical specialty of the attendant surgeon), so we should consider this when making features. There are 2 ways we may either delete the whole column or replace null values with "unknown" label.

For the rest missing values of the features diag_1, diag_2, diag_3 the percentage is very low we can delete missing values in these rows.

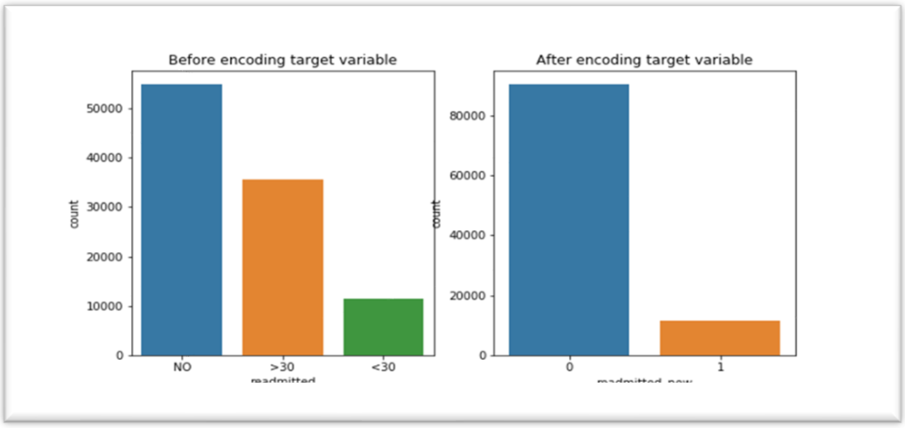
7) Exploratory data analysis

1) Target Variable:

Our target variable is "Readmitted". It contains values '<30', '>30', 'No'. As we are predicting the patients who might be readmitted within 30 days. We will encode the values ' >30' & 'No' with 0 and '<30' with 1.

```
# as we are predicting the readmittance within the 30 day, we will encode "<30" with 1 and ">30" & "No" with 0
re_dict = {'>30':0,'NO':0,'<30':1}
df['readmitted_new'] = df.readmitted.replace(re_dict)
```

```
figure,axes = plt.subplots(nrows=1,ncols=2,figsize=(10,5))
sns.countplot(df.readmitted,ax=axes[0]).set_title('Before encoding target variable')
sns.countplot(df.readmitted_new,ax=axes[1]).set_title('After encoding target variable')
```



From graphs it is clearly visible that the target variable is highly imbalanced. As target variable is imbalanced we will choose f1_score, cohen kappa score as our performance metric.

2) Checking the duplicate patient entries

```
# checking for duplicate entries
df.encounter_id.duplicated().sum()
```

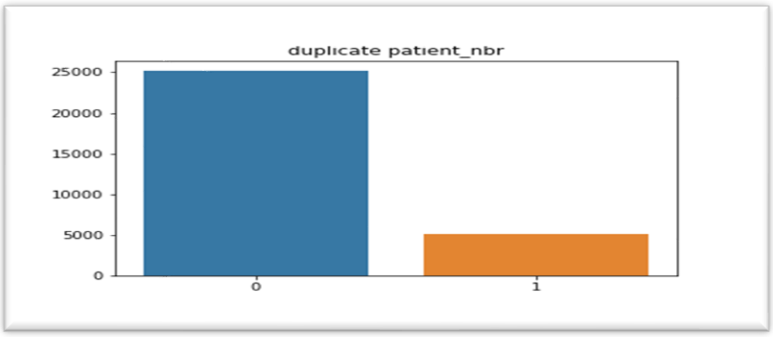
```
0
```

There are no duplicate values present in the Encounter id column.

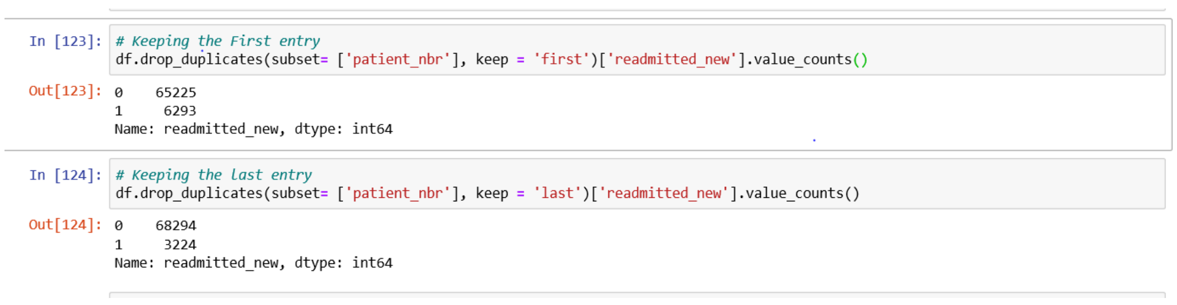
Now, we are checking for the patient_nbr column.

```
# checking for duplicate entries
df.patient_nbr.duplicated().sum()
```

```
30248
```



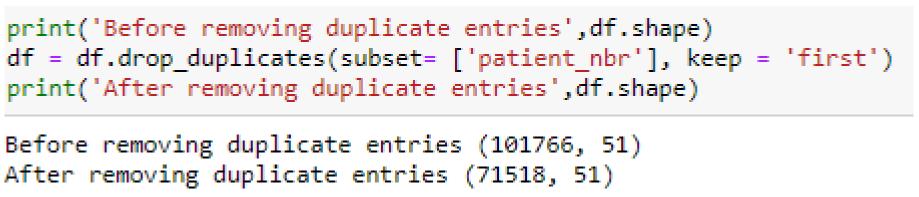
Target value distribution of the duplicate entries is just like the target variable distribution. Before removing the duplicate entries, we will see how it is affecting our target variable.



```
In [123]: # Keeping the First entry
          df.drop_duplicates(subset= ['patient_nbr'], keep = 'first')['readmitted_new'].value_counts()

Out[123]: 0    65225
          1     6293
          Name: readmitted_new, dtype: int64

In [124]: # Keeping the last entry
          df.drop_duplicates(subset= ['patient_nbr'], keep = 'last')['readmitted_new'].value_counts()

Out[124]: 0    68294
          1     3224
          Name: readmitted_new, dtype: int64
```

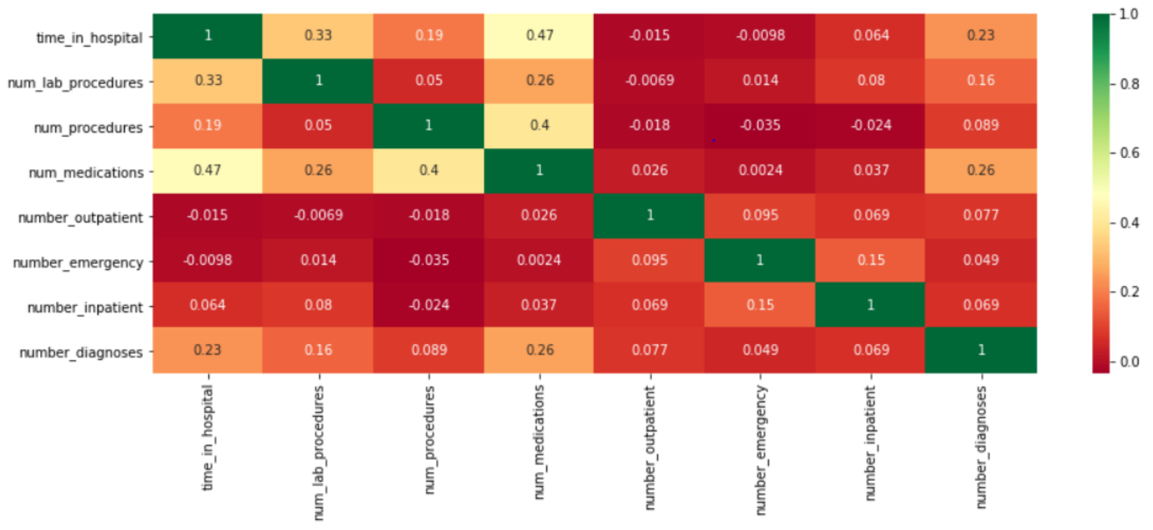
Collapsing of Multiple Encounters for the same patient. Some patients in the dataset had more than one encounter. We could not count them as independent encounters because of that bias the results towards those patients who had multiple encounters.

We then considered the first encounter and last encounter separately as possible representations of multiple encounters. However, last encounters gave extremely imbalanced data for readmissions and thus, we decided to keep first encounters of patients with multiple encounters.



```
print('Before removing duplicate entries',df.shape)
df = df.drop_duplicates(subset= ['patient_nbr'], keep = 'first')
print('After removing duplicate entries',df.shape)

Before removing duplicate entries (101766, 51)
After removing duplicate entries (71518, 51)
```

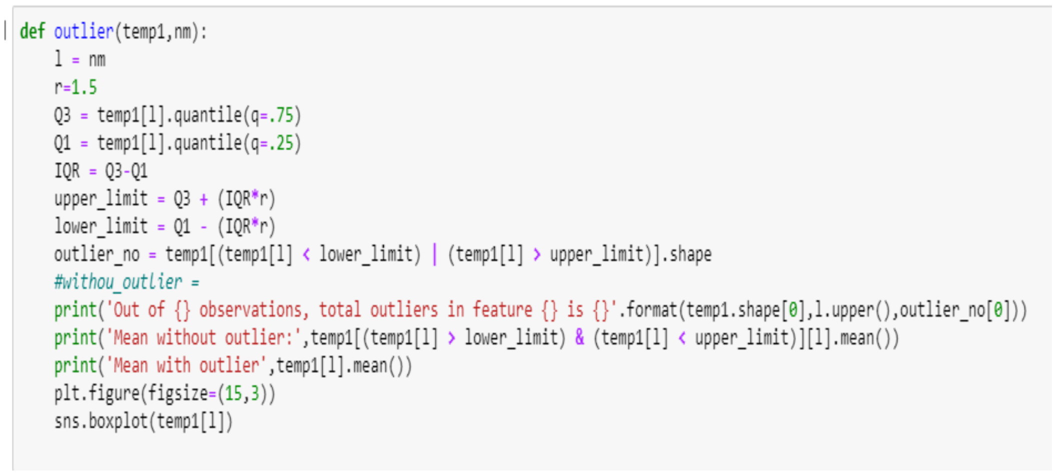
3) Heatmap for Numerical Variables



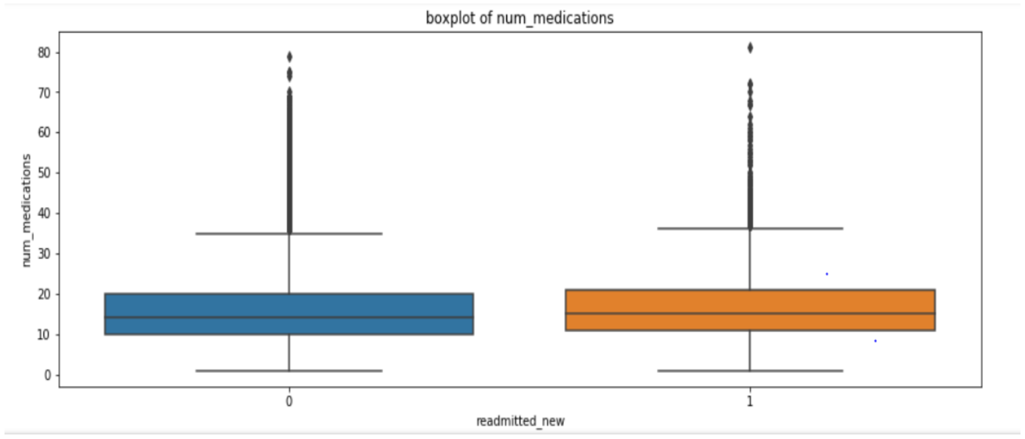
There are 12 independent numeric features in our dataset. As per the heatmap result, the numeric features are not correlated with each other. These numeric features are discrete in nature. They can be considered as categorical features too, given their unique values in the dataset.

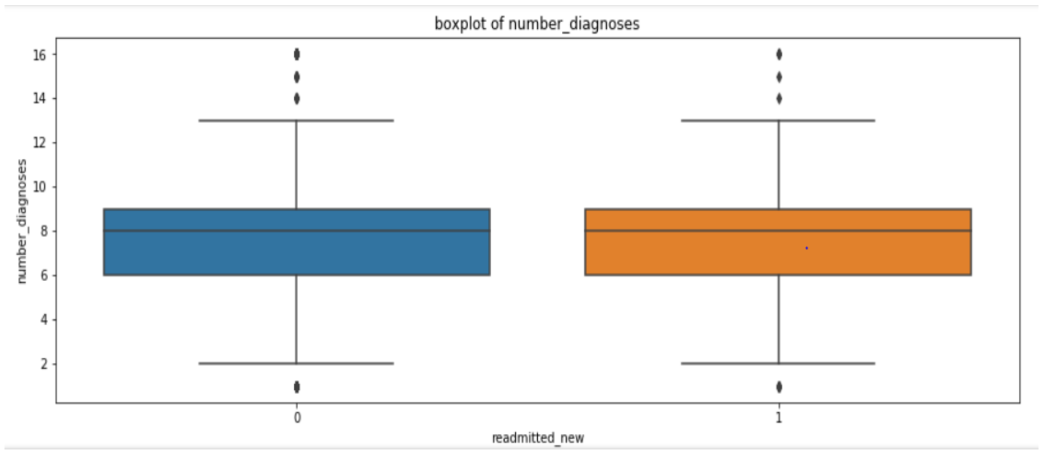
4) Outlier Detection

There are outliers present in the numerical features. As the medical conditions of the patient varies, the number of medicines, number of diagnosis each patient conduct will also vary. As we don't have the full medical history of the patient, we will not remove the outliers as they might be genuine. We will transform the numeric feature to remove the skewness and build model based on the obtained data for comparison.



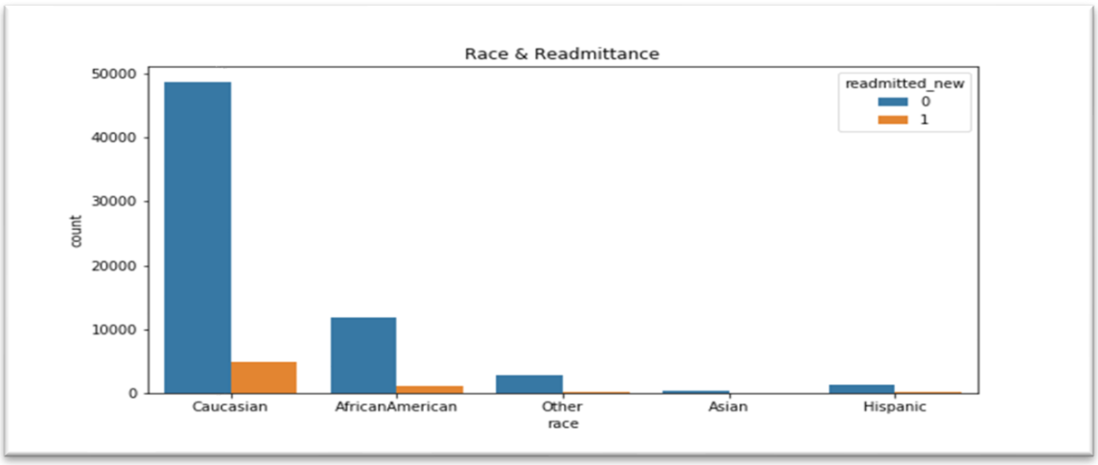
```
def outlier(temp1,nm):
    l = nm
    r=1.5
    Q3 = temp1[l].quantile(q=.75)
    Q1 = temp1[l].quantile(q=.25)
    IQR = Q3-Q1
    upper_limit = Q3 + (IQR*r)
    lower_limit = Q1 - (IQR*r)
    outlier_no = temp1[(temp1[l] < lower_limit) | (temp1[l] > upper_limit)].shape
    #withou_outlier =
    print('Out of {} observations, total outliers in feature {} is {}'.format(temp1.shape[0],l.upper(),outlier_no[0]))
    print('Mean without outlier:',temp1[(temp1[l] > lower_limit) & (temp1[l] < upper_limit)][l].mean())
    print('Mean with outlier',temp1[l].mean())
    plt.figure(figsize=(15,3))
    sns.boxplot(temp1[l])
```





5) Race Vs Readmitted

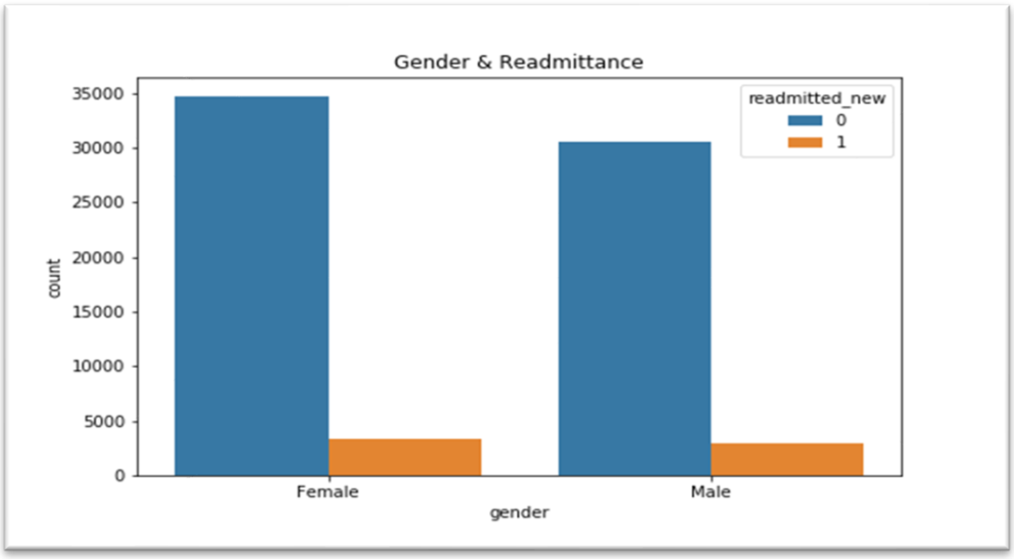
There are missing values '?' present in the race variable, so we had replace the missing values of race column with 'Other'.



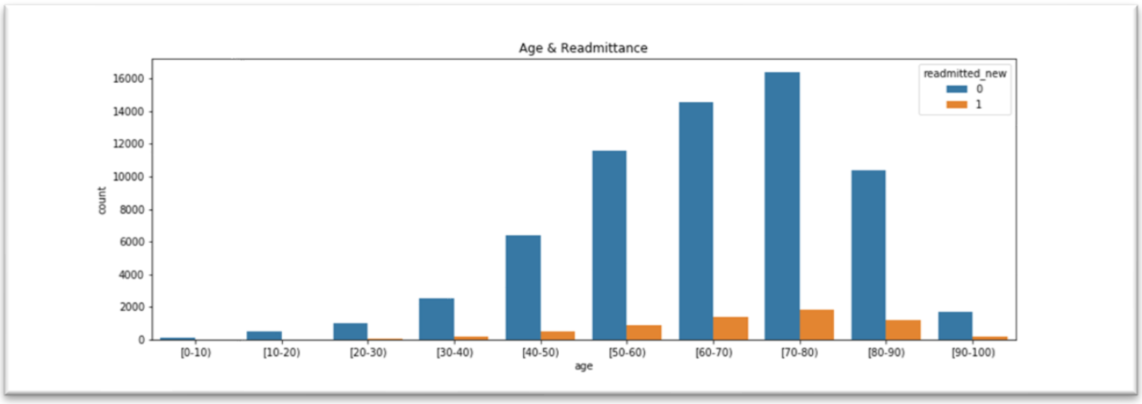
-

6) Gender Vs Readmitted

There are 3 unknown values present in the gender column. Our dataset contains the highest female patient data so we will replace null value of gender with 'Female' label.



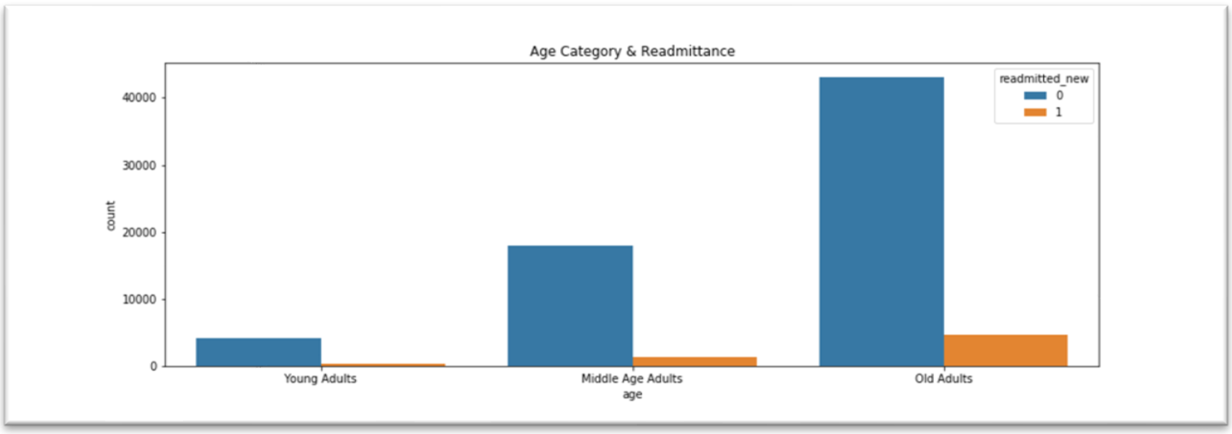
7) Age Vs Readmitted



Age feature is given in the range. We create the category of young, middle and old age from the age feature.

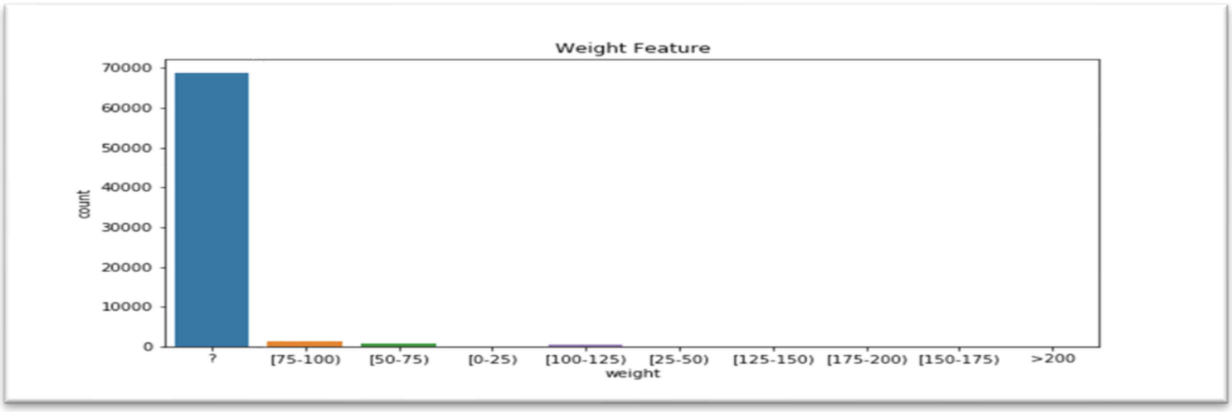
For the Age Group [0-40] have been encoded as Young Adults and similarly [40-60] is encoded by Middle age adults and above 60 have been encoded by old adults. To reduce the probability of creating the multiple variables after dummying the variable, we have grouped values of age feature.

```
age_imp = {'[0-10)':'Young Adults',
           '[10-20)':'Young Adults',
           '[20-30)':'Young Adults',
           '[30-40)':'Young Adults',
           '[40-50)':'Middle Age Adults',
           '[50-60)':'Middle Age Adults',
           '[60-70)':'Old Adults',
           '[70-80)':'Old Adults',
           '[80-90)':'Old Adults',
           '[90-100)':'Old Adults'}
df.age_category = df.age.replace(age_imp)
```



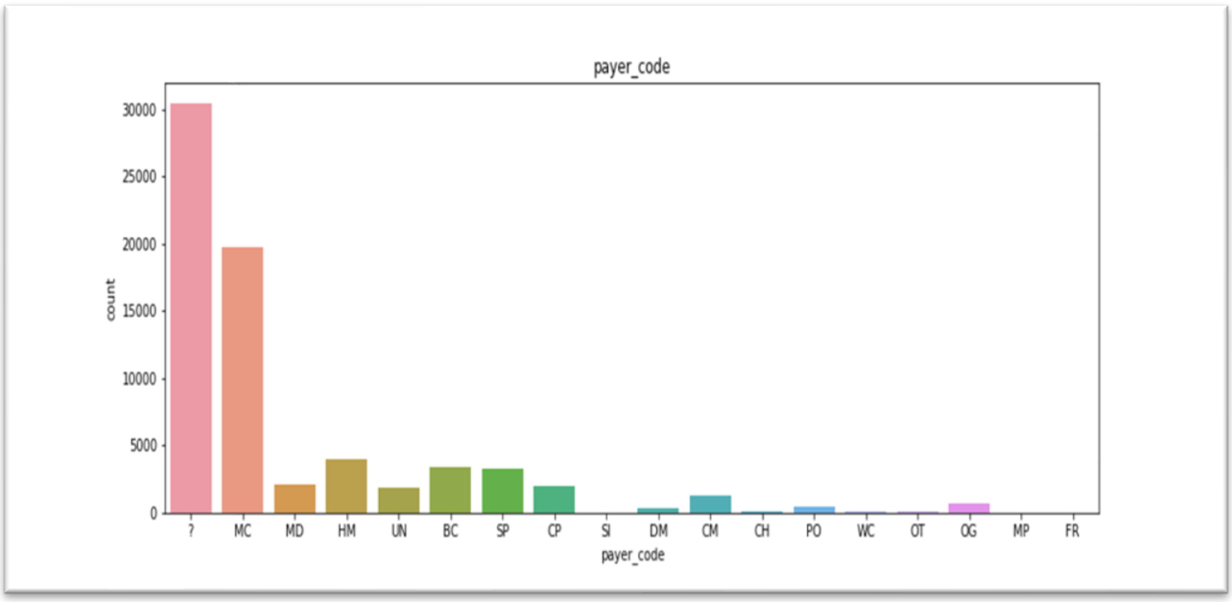
The old adults are getting readmitted more than the middle age and young adults. Also, data shows that old adults with high diabetes are readmitted more often.

8) Weight



There are 96% null values present in the weight column, we cannot predict this much null values hence we will delete this column. The presence of such huge amount of null values can be due to the fact that hospitals in early 2000 where not properly equipped in data collection of weight and recently they have started recording the weight of the patient.

9) Payer_code



There are 46% null values and this column represents the mode of payment. Readmittance cannot be decided by the mode of payment, hence we will delete this column.

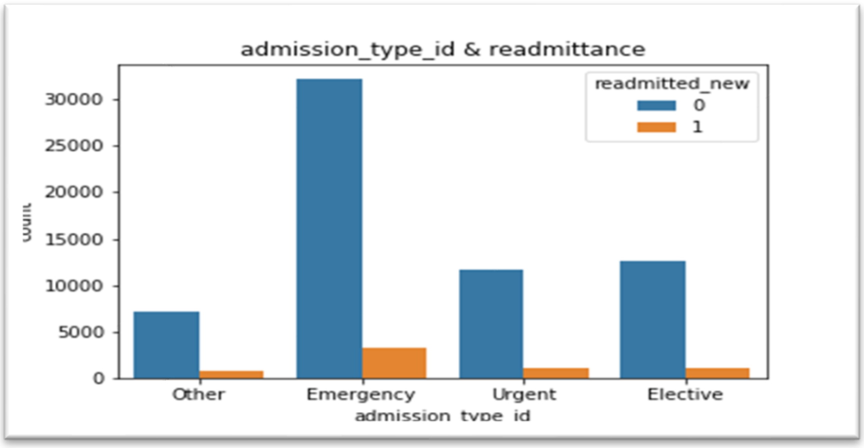
10) IDs Conversion:

We will impute the values in feature admission_type_id, discharge_disposition_id, admission_source_id.

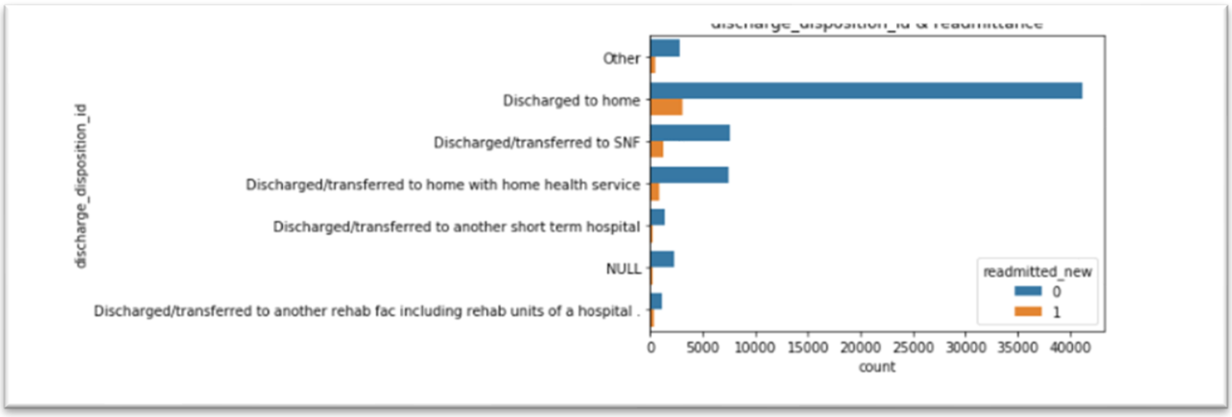
Discharge_disposition_id contains data like hospice, means a home providing care for the sick or terminally ill. As we are only concerned about the patients who were admitted to the hospital, we will remove hospice data. Additionally, we will remove the expired patients from the dataset as we are predicting readmittance within 30 days.

The admission_type_id, discharge_disposition_id, admission_source_id, are all IDs. We will further be grouped them based on their distribution with the target variable.

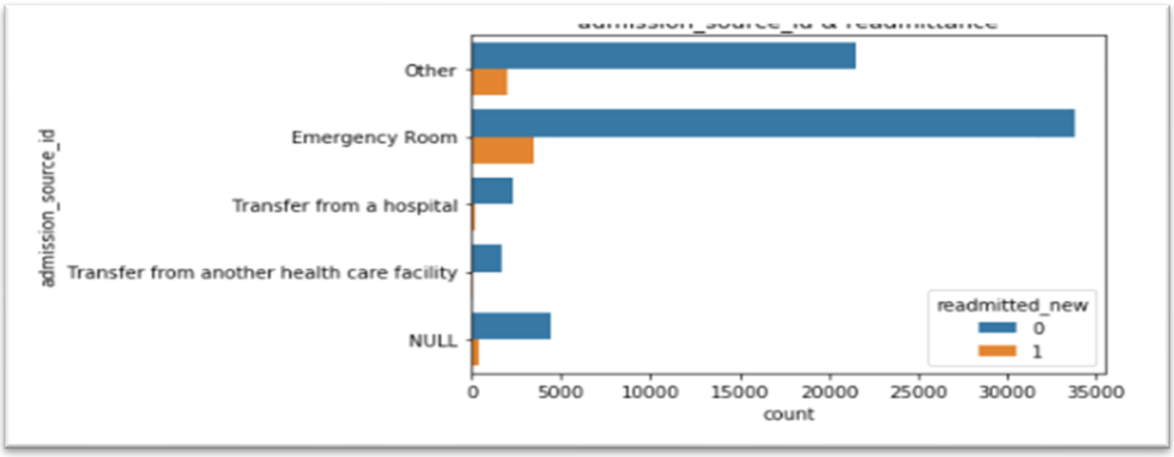
i) Admission_type_id Vs Redmitted



ii) Discharge_dispositin_id Vs Readmitted_new



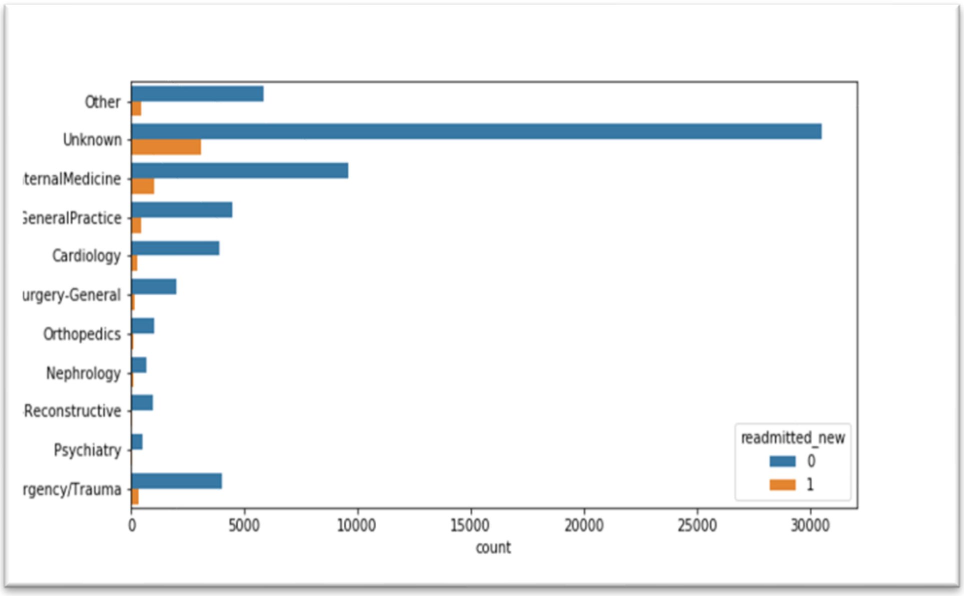
iii) Admission_source_id Vs Readmitted_new



11) medical_specialty:

Replaced the missing values '?' present in the medical_specialty with 'Unknown'.

We will only keep the top occurrence values and replace remaining with "Other" value.



12) diag_1, diag_2, diag_3

Drop the null values in the diag_1, diag_2, diag_3:

```
df.dropna(inplace=True)
df.shape
```

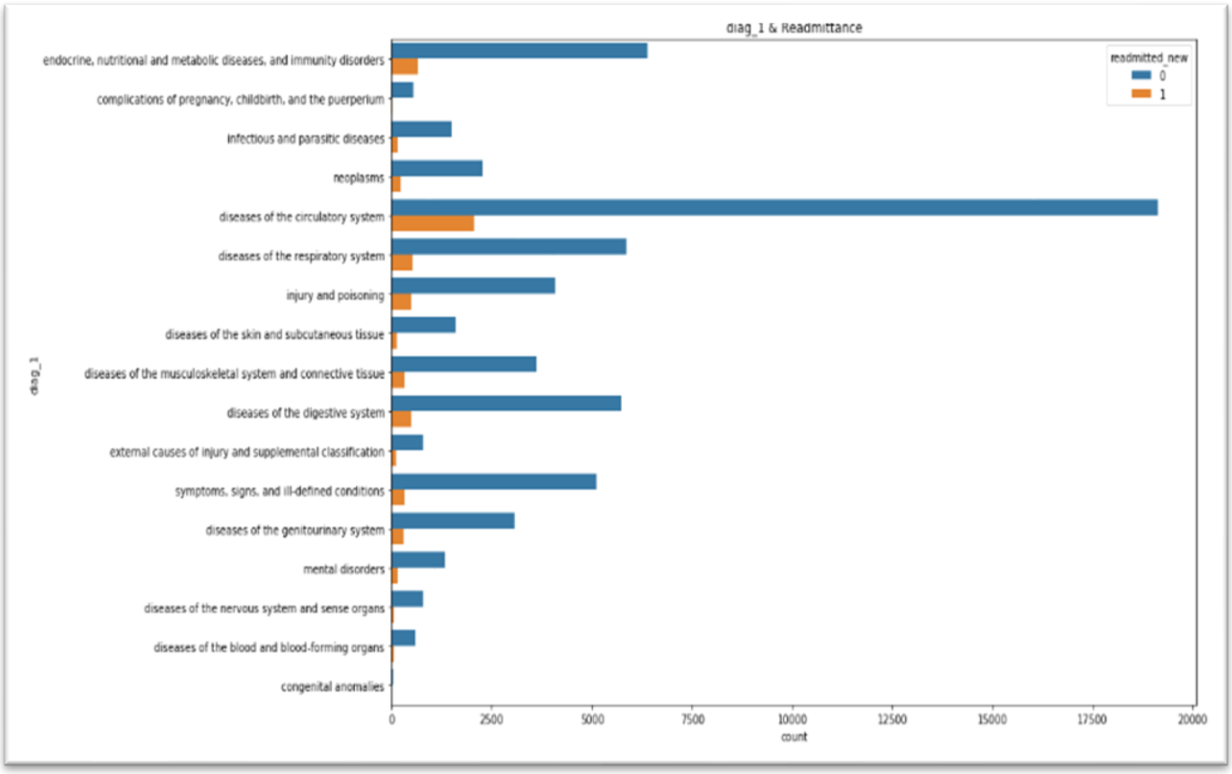
```
(68689, 47)
```

Here we first drop the null values, and then we encode the ICD-9 codes to our columns as per International Statistical Classification of Diseases and Related Health Problems standard names.

ICD-9 diagnosis codes contain three, four, or five digits. The three-digit code is the heading of a section of codes that are further divided by more detailed fourth and fifth digits. Here we have created dictionary for imputations and range of values from 1 to 139 have be recoded as infectious and parasitic disease and so on.

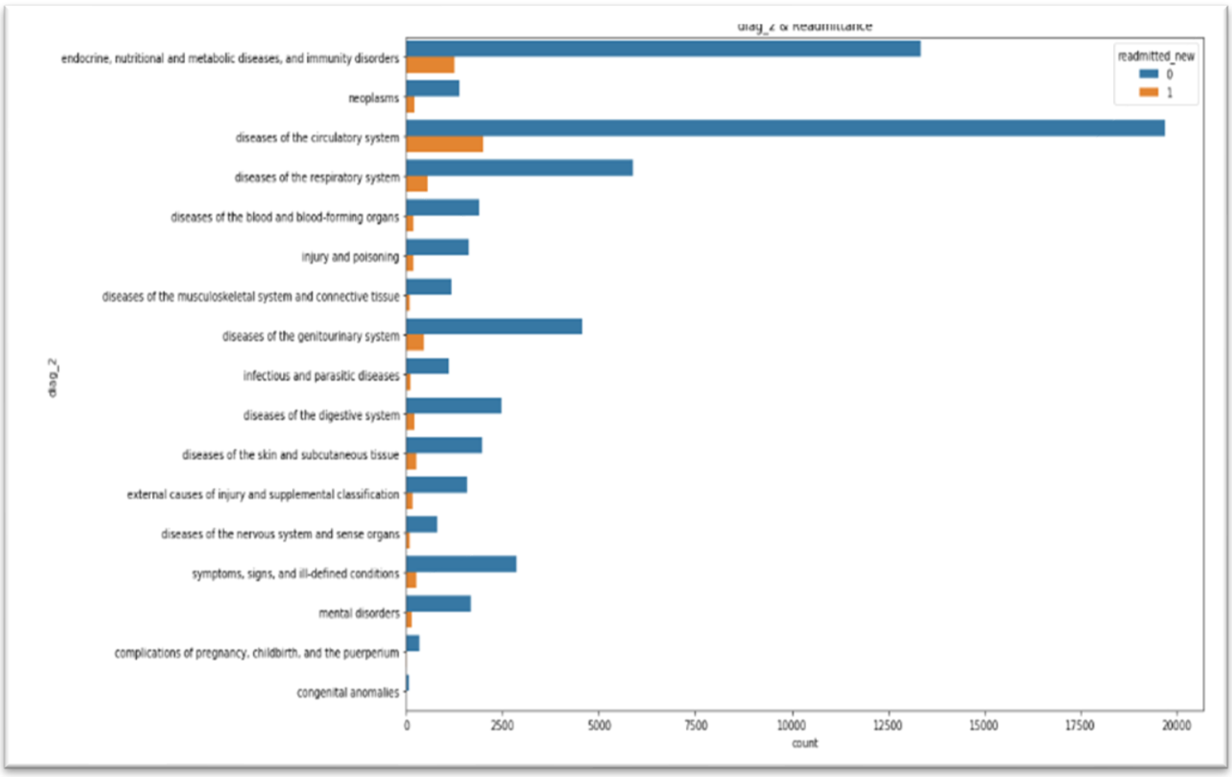
```
# Creating the dictionary for ICD-9 imputation
dict11 = {'infectious and parasitic diseases':(1,139)
          ,'neoplasms':(140,239)
          ,'endocrine, nutritional and metabolic diseases, and immunity disorders':(240,279)
          ,'diseases of the blood and blood-forming organs':(280,289)
          ,'mental disorders':(290,319)
          ,'diseases of the nervous system and sense organs':(320,389)
          ,'diseases of the circulatory system':(390,459)
          ,'diseases of the respiratory system':(460,519)
          ,'diseases of the digestive system':(520,579)
          ,'diseases of the genitourinary system':(580,629)
          ,'complications of pregnancy, childbirth, and the puerperium':(630,679)
          ,'diseases of the skin and subcutaneous tissue':(680,709)
          ,'diseases of the musculoskeletal system and connective tissue':(710,739)
          ,'congenital anomalies':(740,759)
          ,'certain conditions originating in the perinatal period':(760,779)
          ,'symptoms, signs, and ill-defined conditions':(780,799)
          ,'injury and poisoning':(800,999)
          ,'external causes of injury and supplemental classification':(1000,2000)}
```

i) Diag_1 Vs Readmitted_new:

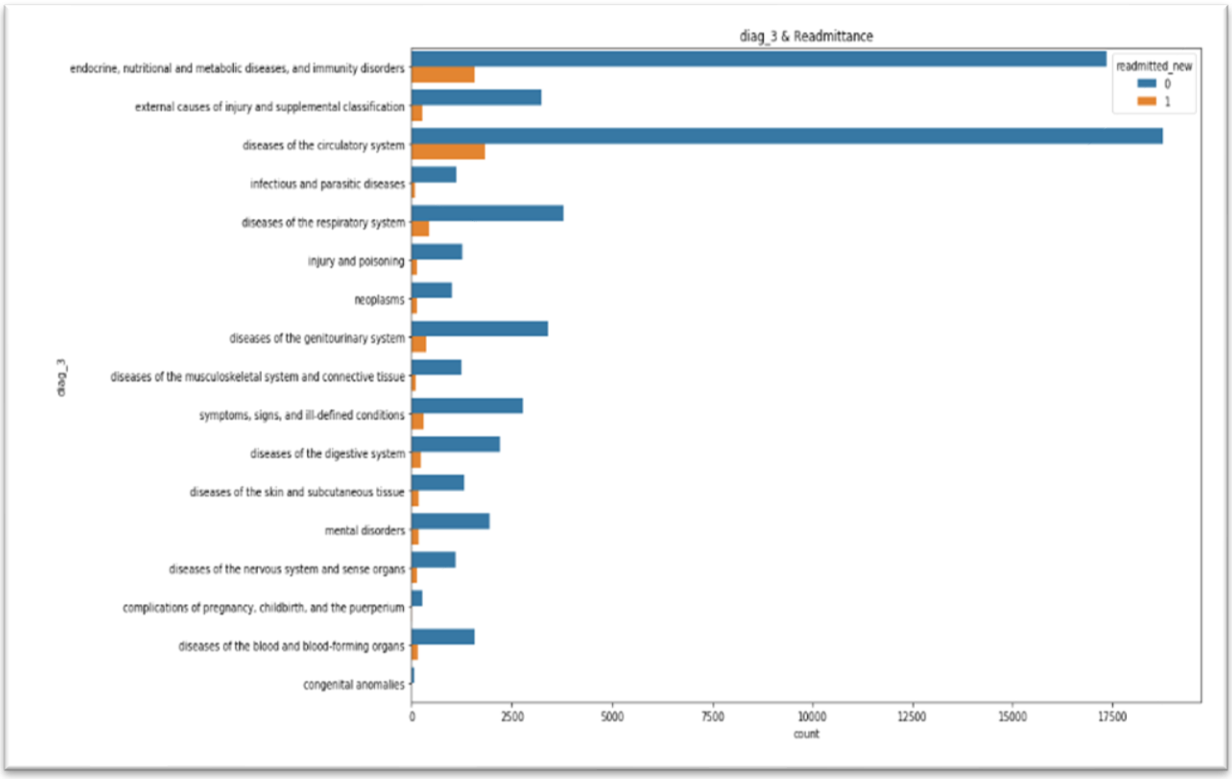


From the bar plot we can observe that patients who have conducted disease of circulatory system tests are highest number of patients who are readmitted in the hospital. We can infer that patients with high number of circulatory problems need a special care and treatment to save the readmittance cost of patient and hospital.

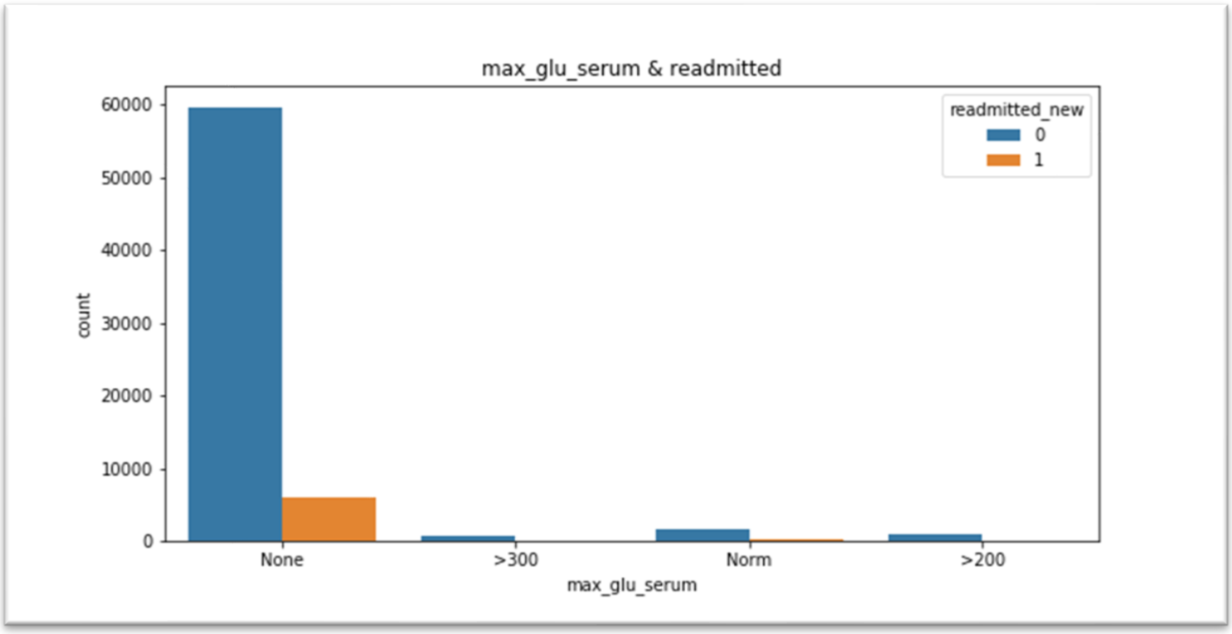
II) Diag_2 Vs Readmitted_new:



III) Diag_3 Vs Readmitted_new:



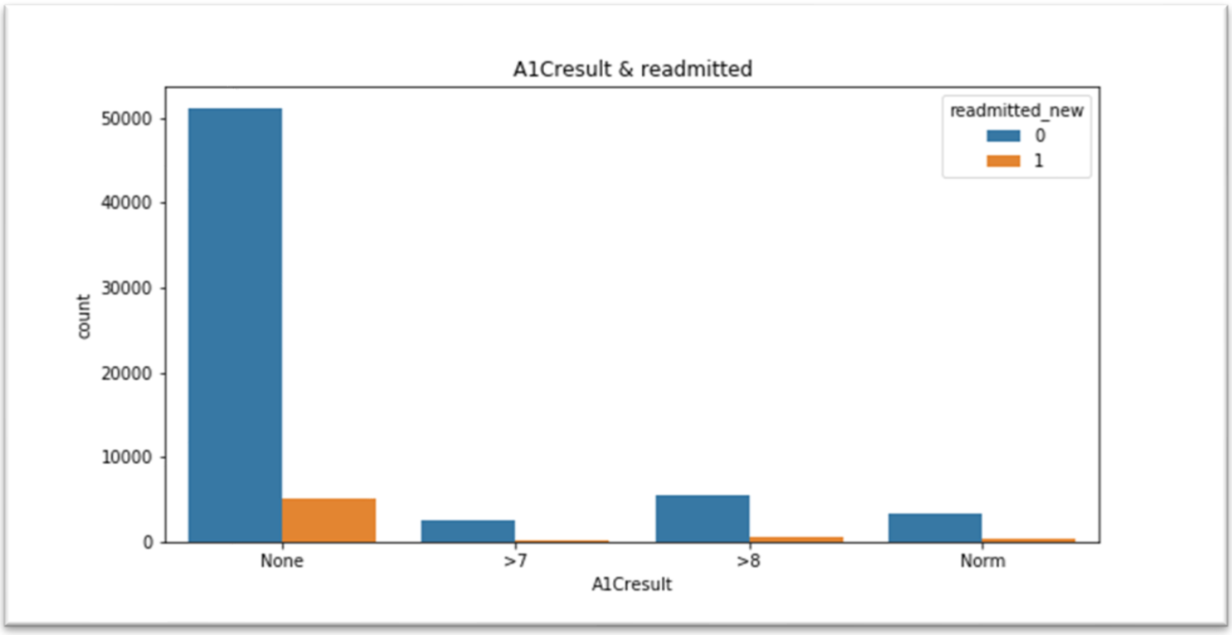
13) Max_glu_serum Vs Readmitted



The reference values for a "normal" random glucose test in an average adult are 80–140mg/dl (4.4–7.8 mmol/l), between 140-200mg/dl (7.8–11.1 mmol/l) is considered pre-diabetes, and ≥ 200 mg/dl is considered diabetes according to ADA guidelines.

The max_glu_serum is an ordinal datatype. We have ordered the values as none, norm, >200 and >300. As per this we will encode the data.

14) A1Cresult Vs Readmitted



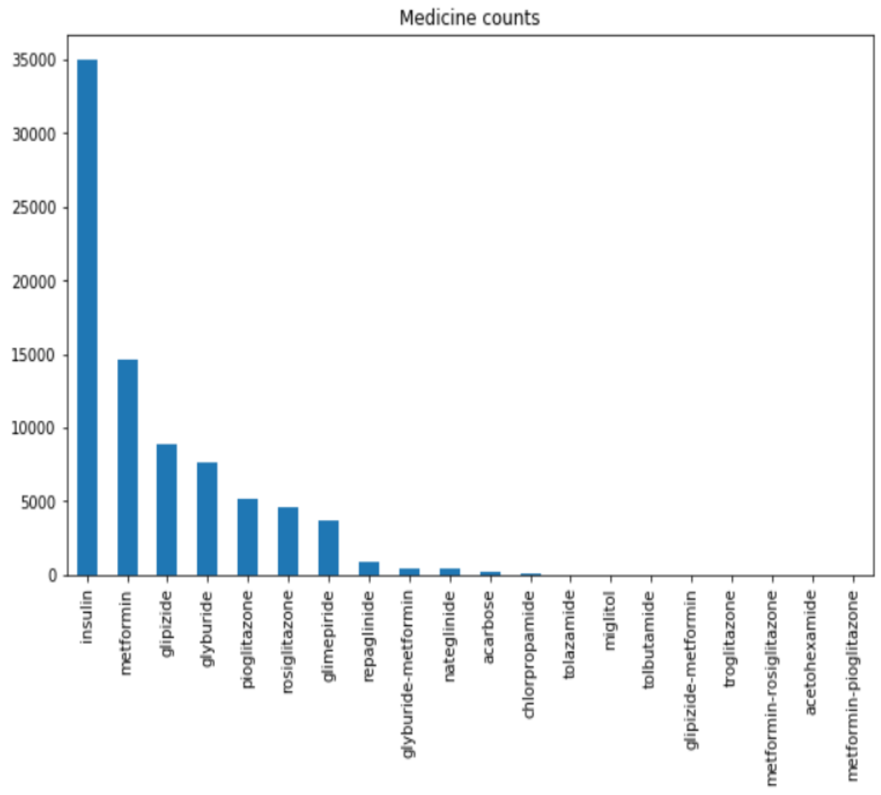
For people without diabetes, the normal range for the haemoglobin A1c level is between 4% and 5.6%. Haemoglobin A1c levels between 5.7% and 6.4% mean you have a higher chance of getting diabetes. Levels of 6.5% or higher mean you have diabetes. The A1Cresult is a ordinal datatype. We have ordered the above as none, norm, >7 and >8. As per this we will encode the data.

15) Medicine columns:

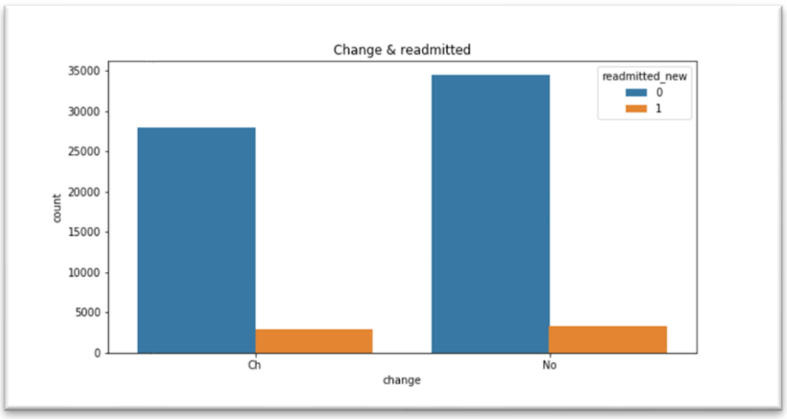
There are 23 medicines features in the dataset. This medicine dosage contains values like ‘None’, ‘Up’, ‘Down’, ‘Norm’. This values can be considered as the ordinal since we can order them. Medicine also represents the ordinal data that can be encoded as No with 0, down with 1, steady with 2, and Up with 3.

```
medicine_columns = ['metformin', 'repaglinide', 'nateglinide', 'chlorpropamide',
         'glimepiride', 'acetohexamide', 'glipizide', 'glyburide',
         'tolbutamide', 'pioglitazone', 'rosiglitazone', 'acarbose',
         'miglitol', 'troglitazone', 'tolazamide',
         'insulin', 'glyburide-metformin', 'glipizide-metformin',
         'glimepiride-pioglitazone', 'metformin-rosiglitazone',
         'metformin-pioglitazone','examide','citoglipton']
```

From above plots "citoglipthon" and "examide" & "glimepiride-pioglitazone" features have exactly 1 value so we will delete this column.

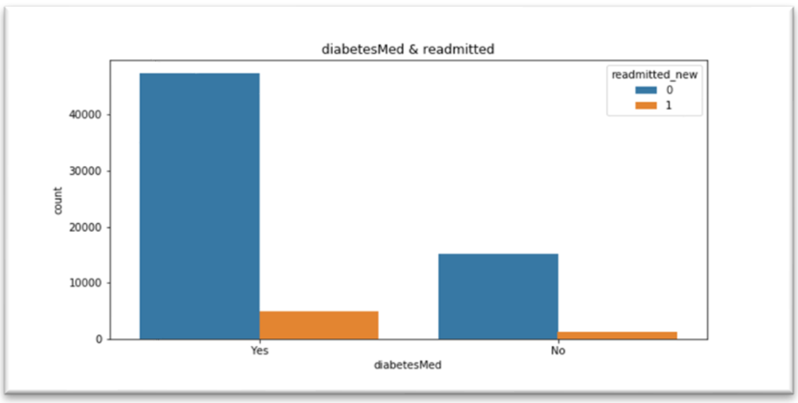


16) Change Vs Readmittance



Patients whose medicines were changed are getting readmitted within 30 days too. There is no biasness or whether medicine changed or not is not affecting readmittance.

17) DiabetesMed Vs Readmittance



Patients who were or were not administered with the diabetes medicine are getting readmitted within 30 days.

8) Data Preparation

Dropping the below columns:

I) age - we have created the category from the age hence we will drop the age column.

II) readmitted - this feature is encoded to create new column. hence will delete this column.

III) "citoglipthon" and "examide" & "glimepiride-pioglitazone" - These columns contain exactly 1 unique value, therefore these columns will not contribute towards prediction hence we will drop them too.

```
#droping the final columns
df.drop(['age','examide','citoglipton','glimepiride-pioglitazone','readmitted'],axis=1,inplace=True,errors='ignore')
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    display(df.head())
```

	race	gender	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	medical_specialty	num_lab_procedures	num_pr
1	Caucasian	Female	Emergency	Discharged to home	Emergency Room	3	Unknown	59	
2	AfricanAmerican	Female	Emergency	Discharged to home	Emergency Room	2	Unknown	11	
3	Caucasian	Male	Emergency	Discharged to home	Emergency Room	2	Unknown	44	

We are going to create 2 models. One with the base dataset and the another after transforming numerical variable with sqrt.

```
x = df.drop('readmitted_new',axis=1)
y = df.readmitted_new
x = pd.get_dummies(x)
```

Transforming data with the sqrt method.

```
x1 = df.drop('readmitted_new',axis=1)
y1 = df.readmitted_new
x1 = pd.get_dummies(x1)

for i in numeric_var:
    x1[i] = np.sqrt(x1[i])
```

9) Feature Selection

Feature selection is a technique where we choose those features in our data that contribute most to the target variable. In other words, we choose the best predictors for the target variable.

All the variables are not significant enough. We can do feature selection using statistical method like ANOVA F test, with the help of correlation, business understanding or we can use SelectKbest function. Here we are using SelectKbest for 20 best features. Statistical method is not used

```
from sklearn.feature_selection import SelectKBest, chi2
skb = SelectKBest(chi2).fit(x,y)
best_feature = pd.DataFrame((x.columns,skb.scores_)).T.sort_values(by=1,ascending=False)
best = list(best_feature[0].head(20))  ## BEST FEATURES 20 SELECTED FROM THIS QUERY

best_feature.head(30)
```

Using K-best we get the following features as significant

- Number_inpatients
- Num_lab_procedures
- Discharge_disposition_id
- Time_in_hospital
- num_medications
- diag_1
- repaglinide
- insulin
- diag_2
- metformin
- medical_speciality
- diag_3
- max_glu_serum
- race
- diabetes_med

The sub-categories are not included in the above list.

10) Performance Metric

Performance Metric: f1_score

Accuracy is used when the True Positives and True negatives are more important while F1-score is used when the False Negatives and False Positives are crucial. Accuracy can be used when the class distribution is similar while F1-score is a better metric when there are imbalanced classes as in the above case

What does f1_score tell you?

In statistical analysis of binary classification, the F1 score (also F-score or F-measure) is a measure of a test's accuracy. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0.

Why is f1 score Important?

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution.

11) Base Model Building:

Based on the data features with extreme null values such as weight were dropped and many features which were having many sub-categories were grouped based on frequency in order to build model with less dimension to avoid computational problems.

As we are working on binary classification problem. We will choose the logistic regression for our base model. Since the target variable is highly imbalanced, we will use parameter class weight which assigns weight to the class which are significantly lower than other classes and build model named Logistic Regression Balanced as shown below:

Choosing all the features:

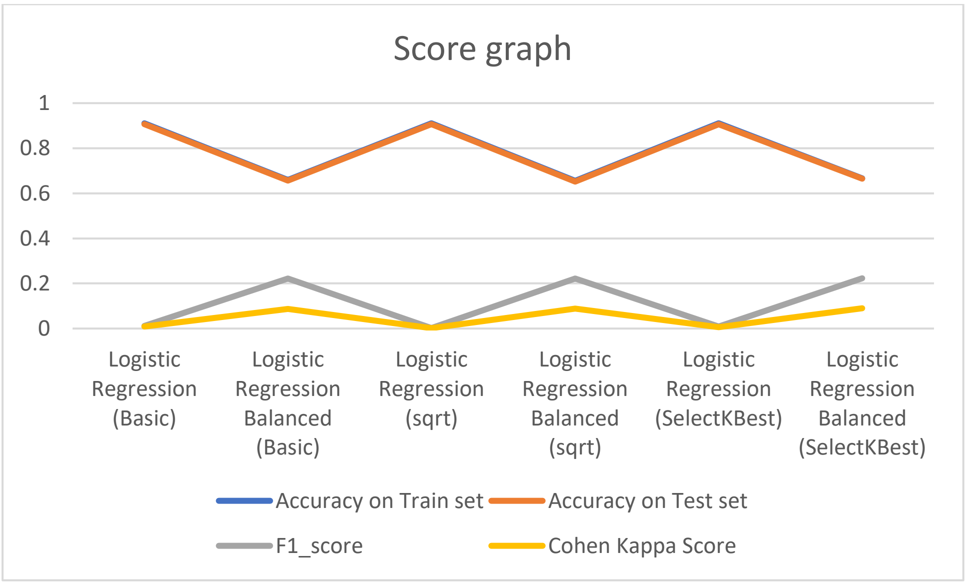
	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.910715	0.906828	0.011329	0.009331
1	Logistic Regression Balanced	0.659810	0.656185	0.221685	0.087452

Choosing all the features and applying square-root transformation to numerical features:

	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.910715	0.906828	0.002079	0.001789
1	Logistic Regression Balanced	0.655339	0.651914	0.222439	0.087780

Choosing the K-best features and building the model without transformation

	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.910736	0.906682	0.008252	0.006532
1	Logistic Regression Balanced	0.666112	0.665065	0.222923	0.089948



Inference:

With the model created using the balanced parameter we can see the sudden drop in the train & test score. The model in underfitted as it is learning less pattern. But at a same time f1_score have increased for the balanced logistic regression model.

There is no significant change in the f1_score or the accuracy score even after doing the sqrt transformation or choosing the kbest features.

12) Handling Data Imbalance:

What is Imbalanced Classification?

Imbalanced classification is a supervised learning problem where one class outnumbers other class by a large proportion. This problem is faced more frequently in binary classification problems than multi-level classification problems.

What are the methods to deal with imbalanced data sets?

The methods are widely known as ‘Sampling Methods’. Generally, these methods aim to modify an imbalanced data into balanced distribution using some mechanism. The modification occurs by altering the size of original data set and provide the same proportion of balance.

Below are the methods used to treat imbalanced datasets:

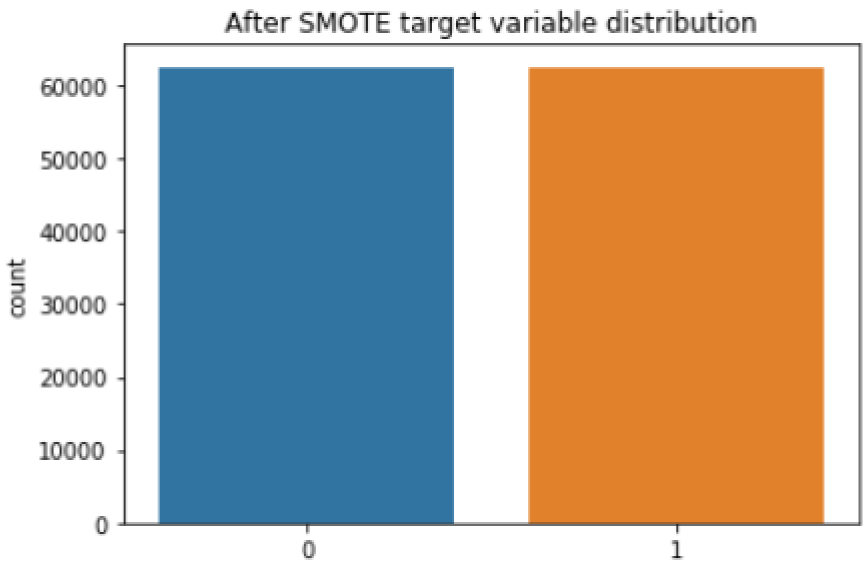
a. Under Sampling
b. Oversampling
c. SMOTE

a) Under Sampling
This method works with majority class. It reduces the number of observations from majority class to make the data set balanced. This method is best to use when the data set is huge and reducing the number of training samples helps to improve run time and storage troubles

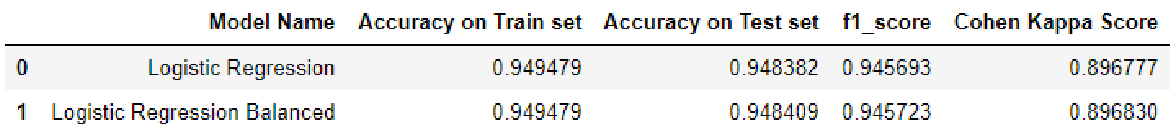
b) Over Sampling
This method works with minority class. It replicates the observations from minority class to balance the data. It is also known as *up sampling*.

c) SMOTE
SMOTE stands for Synthetic Minority Over-sampling Technique. Unlike Random Under Sampling, SMOTE creates new synthetic points in order to have an equal balance of the classes. This is another alternative for solving the "class imbalance problems".

We will not use Down sampling as it leads to large deletion of the data and Up sampling is not preferred as it leads to overfitting of the model. We will build our model using SMOTE.



Building Base Model using SMOTE



	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.949479	0.948382	0.945693	0.896777
1	Logistic Regression Balanced	0.949479	0.948409	0.945723	0.896830

Inference: After Balancing the Target variable using SMOTE, we can see that test score and other accuracy metrics have shown a drastic change. Since SMOTE does not leads to loss of useful information and it mitigates the problem of overfitting cause by random oversampling as synthetic examples are generated rather than replication of instances therefore we will use SMOTE in the future iteration for model building.

13) Model Building and Comparison:

I) Imbalanced Dataset

The models which where not been chosen to predict the readmitted class are:

- K-nearest Neighbors: While K-nearest neighbors provide decent predictions, they cannot help in deciding the features that contribute to this decision the most, since features are weighted equally (assuming we normalize them) based on simply their contribution to the proximity/distance function.
- Support Vector Machines: Support Vector Machines can help model linearly inseparable data, thus allowing us to explain complex non-linear relationships. However, because of high-dimensional structure and complexity, they are limited by

their interpretability to gain insights on how different features are weighted/assigned importance.

- Naive Bayes: Naive bayes uses time-sequence information of what came before (prior) and what came after (posterior) the variable being predicted. This could be useful but since we did not have data on which medicine was given before and which after, and the features available have little or no time-based ordering, using Naive Bayes would result in a model that is not interpretable.

Model which have chosen for prediction of readmittance are as follow:

- Logistic Regression: Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. To represent binary / categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function
- Decision Tree: Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

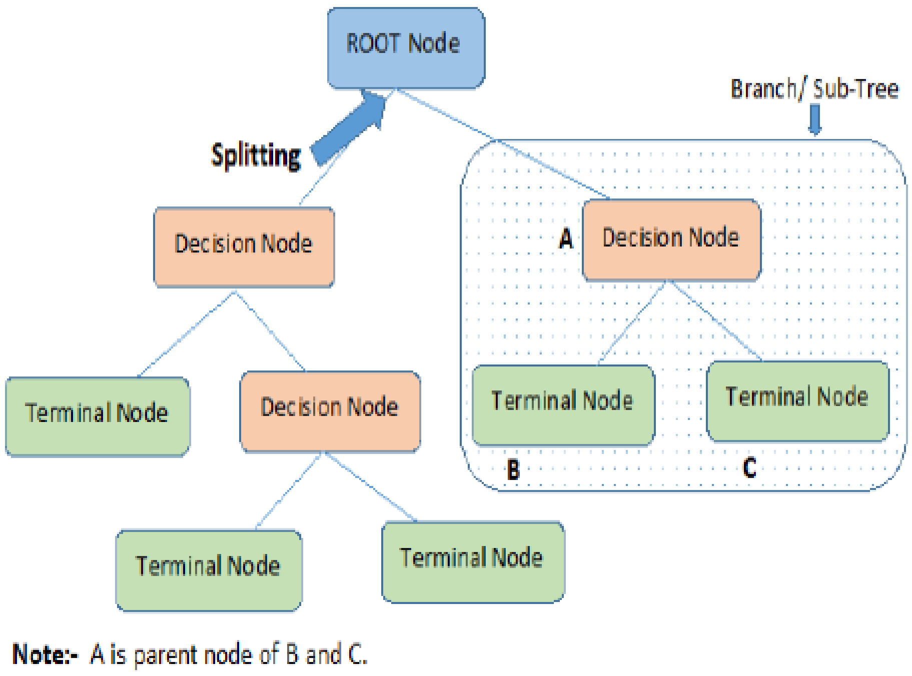


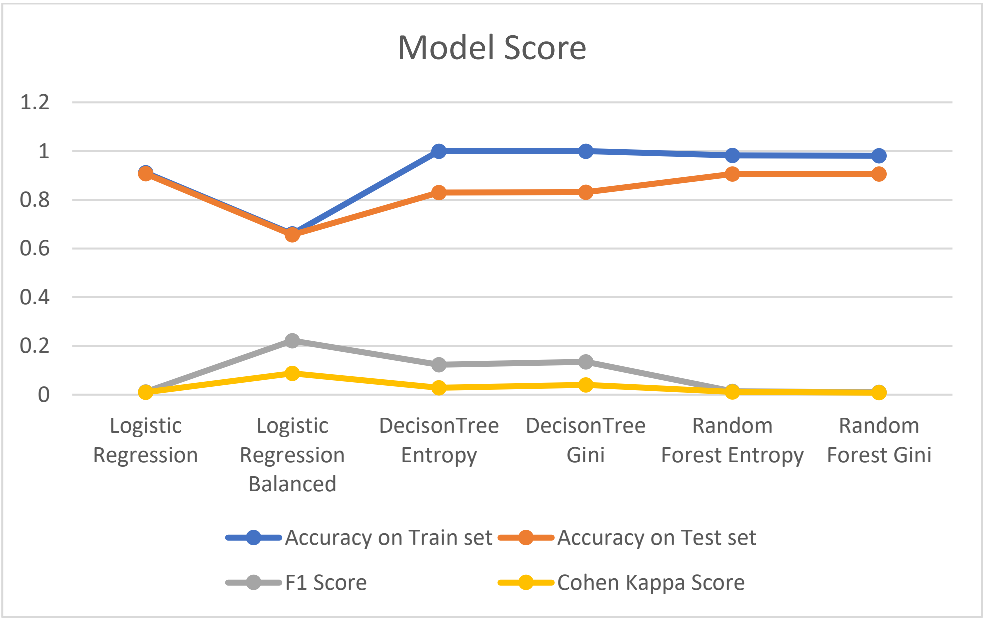
Fig:5 Decision Tree split

- Random Forest: Random Forest is the popular ensemble method in which several trees are developed using different strategy. One of the frequently used sampling strategy is Bootstrap Aggregating (Bagging). Bagging is a random sampling with replacement. A new observation is classified by using all the trees developed in the random forest and majority voting is used for deciding the class.

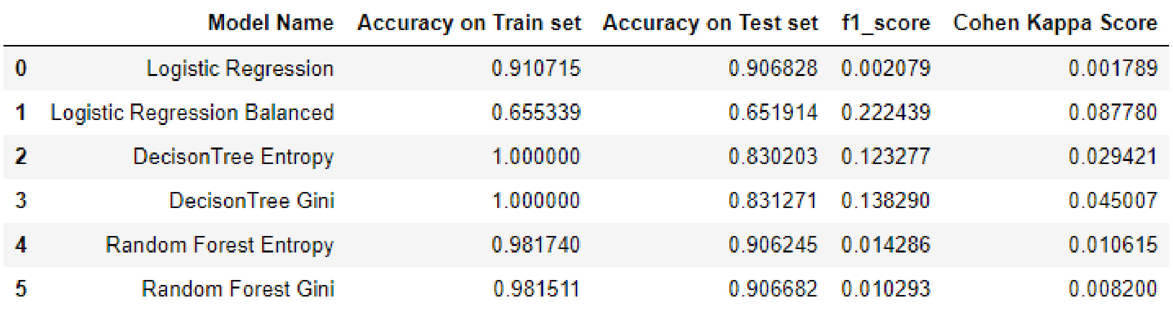
Building Models with and without square-root transformation, unbalanced dataset without Hyper-parameter tuning:

- Output of the Original data without square root transformation (unbalanced data)

	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.910715	0.906828	0.011329	0.009331
1	Logistic Regression Balanced	0.659810	0.656185	0.221685	0.087452
2	DecisonTree Entropy	1.000000	0.829524	0.122408	0.028170
3	DecisonTree Gini	1.000000	0.830543	0.134358	0.040672
4	Random Forest Entropy	0.981760	0.906343	0.014300	0.010815
5	Random Forest Gini	0.981552	0.906634	0.010288	0.008100



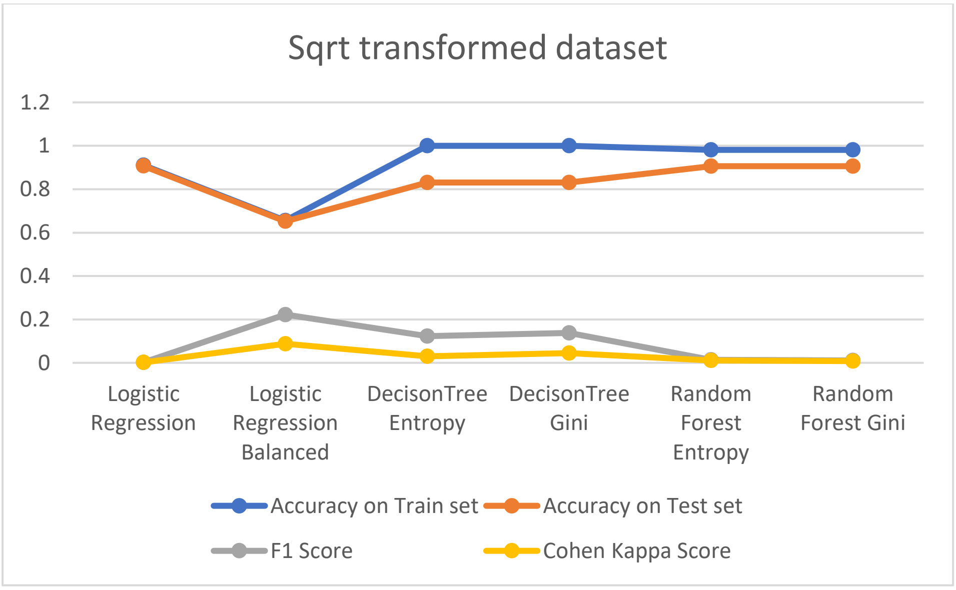
- Output of the Original data with square root transformation (unbalanced data)



	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.910715	0.906828	0.002079	0.001789
1	Logistic Regression Balanced	0.655339	0.651914	0.222439	0.087780
2	DecisonTree Entropy	1.000000	0.830203	0.123277	0.029421
3	DecisonTree Gini	1.000000	0.831271	0.138290	0.045007
4	Random Forest Entropy	0.981740	0.906245	0.014286	0.010615
5	Random Forest Gini	0.981511	0.906682	0.010293	0.008200

Building Models with square-root transformation, balanced dataset without Hyper-parameter tuning:

- Output using balanced data with square-root transformation



Inference: From the above observation we can observe that Random Forest and Logistics Regression are better in terms of Accuracy and F1-score while Decision Tree has slightly lesser accuracy than both of them.

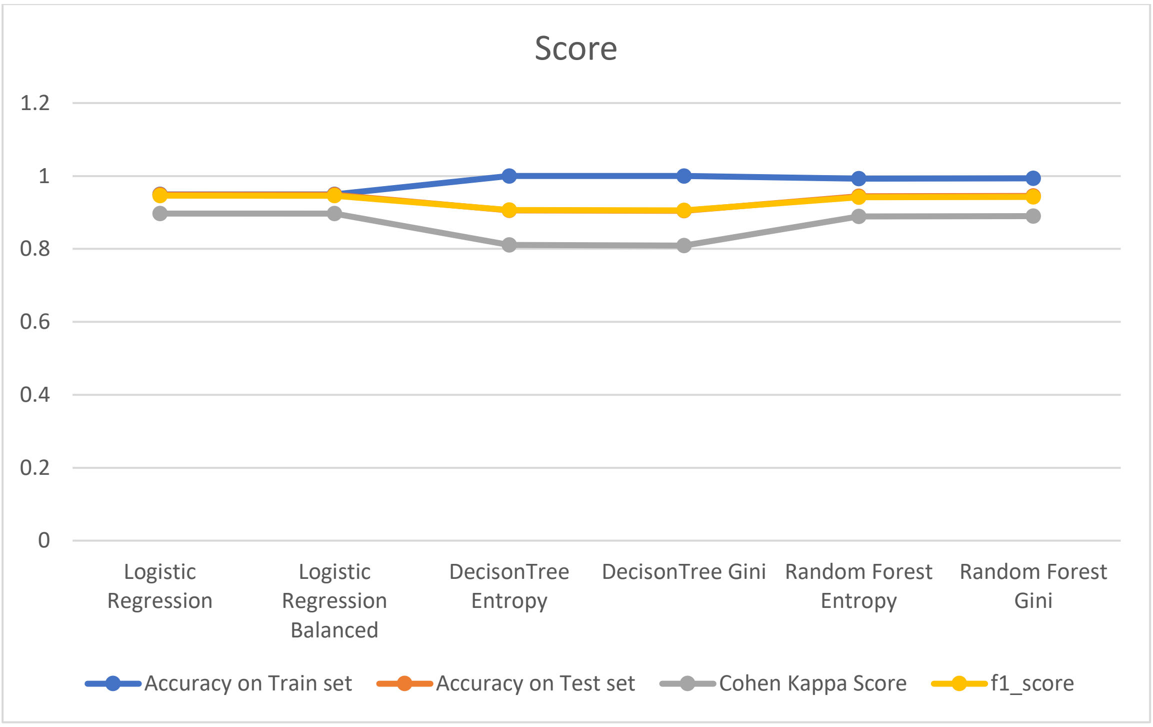
We have considered both Gini as well as Entropy as a splitting parameter while building Decision tree and Random forest. We can further improve the score using hyper-parameter tuning and select the best parameters and features.

II) Balancing the Target variable using smote:

i) Basic Model

	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.949479	0.948382	0.945693	0.896777
1	Logistic Regression Balanced	0.949479	0.948409	0.945723	0.896830
2	DecisonTree Entropy	1.000000	0.905301	0.906229	0.810597
3	DecisonTree Gini	1.000000	0.904554	0.905599	0.809102
4	Random Forest Entropy	0.993038	0.944541	0.941818	0.889094
5	Random Forest Gini	0.993392	0.944914	0.942252	0.889841

```
Wall time: 21 s
```



After balancing the target variable using smote, we are getting good f1_score, and cohen-kappa score. As the dataset is balanced parameter 'balanced' of Logistic regression is not giving much significant difference in score. As we can see from the graph the accuracy score and f1 score are overlapping. As the dataset is balanced, we can now choose our model based on the accuracy score on testing data.

Logistic regression, random forest with gini and entropy are giving similar output.

ii) Kbest 20 Feature

Select Kbest features on the Balanced dataset:

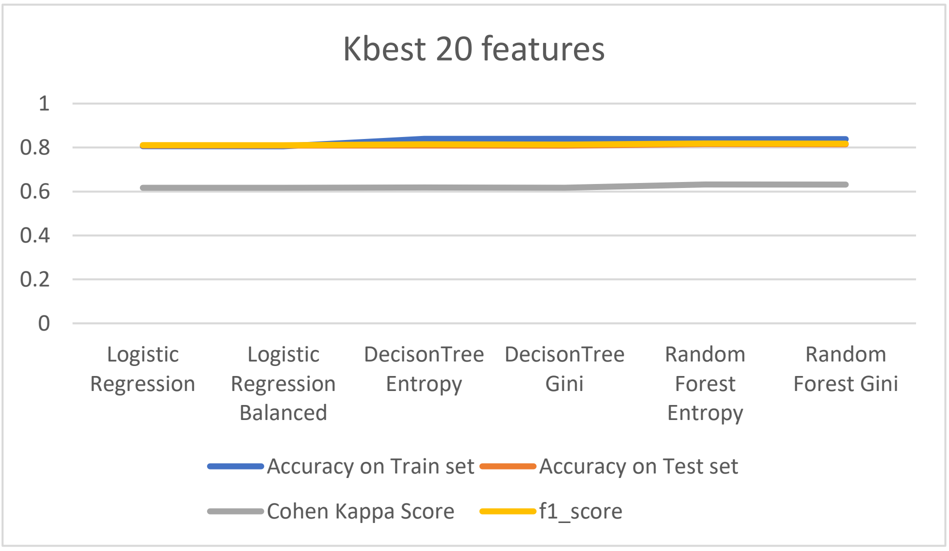
```
from sklearn.feature_selection import SelectKBest, chi2
skb = SelectKBest(chi2).fit(X_res,y_res)
best_feature = pd.DataFrame((X_res.columns,skb.scores_)).T.sort_values(by=1,ascending=False)
best = list(best_feature[0].head(20))
```

With Top 20 Features:

```
print('Top 20 features')
x4 = X_res[best]
y4 = y_res.copy()
x_train_new,x_test_new,y_train_new,y_test_new = train_test_split(x4,y4,train_size=.7,random_state=1)
model_outputs2 = fun_model(models,x_train_new,x_test_new,y_train_new,y_test_new)
display(model_outputs2)
```

```
Top 20 features
```

	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.805748	0.808680	0.811104	0.617349
1	Logistic Regression Balanced	0.805737	0.808680	0.811124	0.617349
2	DecisonTree Entropy	0.839612	0.808920	0.814483	0.617814
3	DecisonTree Gini	0.839612	0.808547	0.814207	0.617066
4	Random Forest Entropy	0.838206	0.815963	0.818614	0.631913
5	Random Forest Gini	0.838366	0.815669	0.818244	0.631327



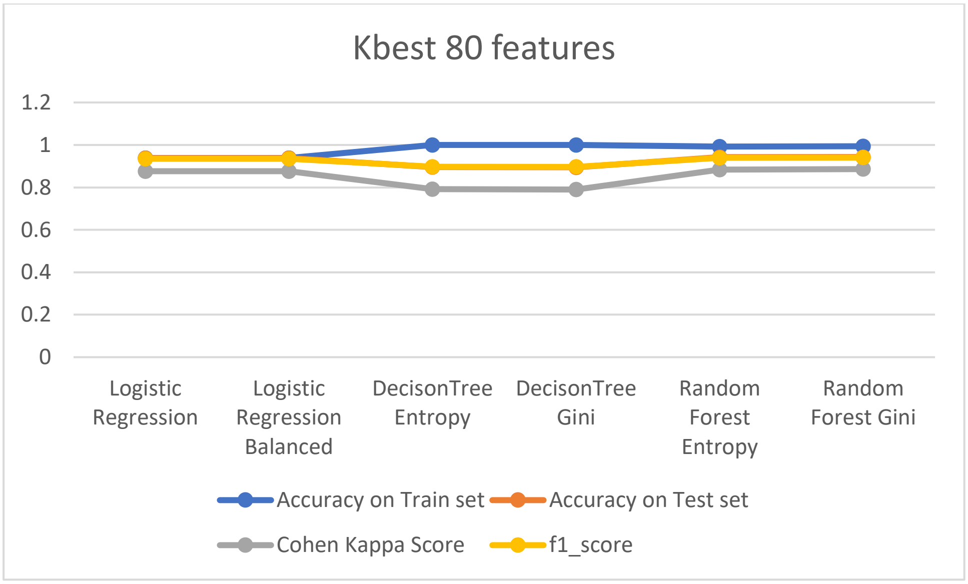
Using 20 features our overall model score has decreased. From the basic dataset we are getting almost 15 points drop in the testing score. Hence choose 20 features for model in not recommended.

iii) Kbest 80 Features

```
top = 80
print('Top {} features'.format(top))
best = list(best_feature[0].head(top))
x4 = X_res[best]
y4 = y_res.copy()
x_train_new,x_test_new,y_train_new,y_test_new = train_test_split(x4,y4,train_size=.7,random_state=1)
model_outputs2 = fun_model(models,x_train_new,x_test_new,y_train_new,y_test_new)
display(model_outputs2)
```

```
Top 80 features
```

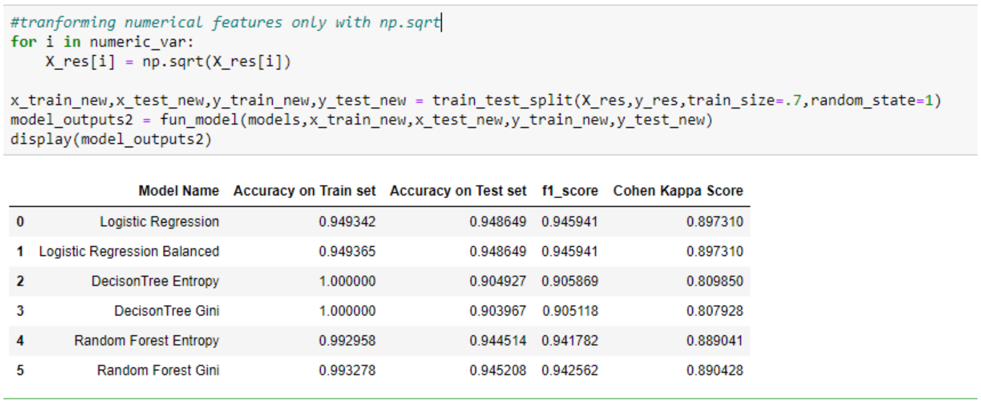
	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.938573	0.938405	0.935458	0.876824
1	Logistic Regression Balanced	0.938550	0.938379	0.935432	0.876771
2	DecisonTree Entropy	1.000000	0.895804	0.897437	0.791600
3	DecisonTree Gini	1.000000	0.894977	0.896610	0.789946
4	Random Forest Entropy	0.992637	0.941847	0.939215	0.883705
5	Random Forest Gini	0.993129	0.943074	0.940394	0.886160



After using the Kbest 80 features our model score is close enough to our basic dataset. Logistic regression, random forest with gini and entropy are giving similar output.

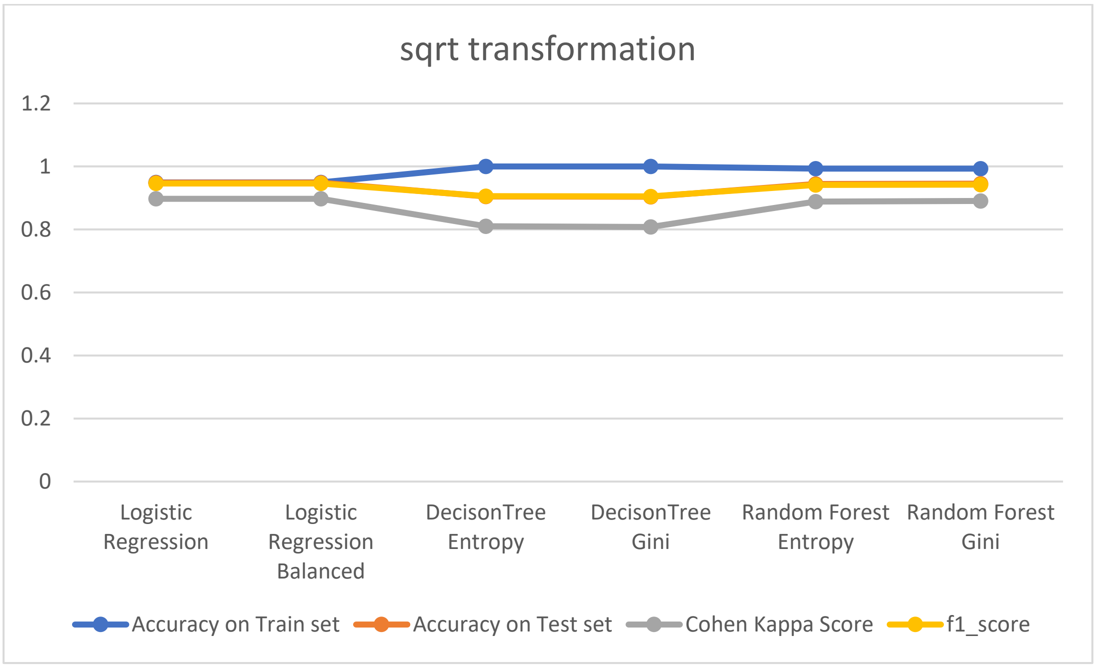
iv) square root transformation

Lets check by Transforming Numerical features only using square root transformation:



```
#tranforming numerical features only with np.sqrt
for i in numeric_var:
    X_res[i] = np.sqrt(X_res[i])

x_train_new,x_test_new,y_train_new,y_test_new = train_test_split(X_res,y_res,train_size=.7,random_state=1)
model_outputs2 = fun_model(models,x_train_new,x_test_new,y_train_new,y_test_new)
display(model_outputs2)
```

	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.949342	0.948649	0.945941	0.897310
1	Logistic Regression Balanced	0.949365	0.948649	0.945941	0.897310
2	DecisonTree Entropy	1.000000	0.904927	0.905869	0.809850
3	DecisonTree Gini	1.000000	0.903967	0.905118	0.807928
4	Random Forest Entropy	0.992958	0.944514	0.941782	0.889041
5	Random Forest Gini	0.993278	0.945208	0.942562	0.890428



Inference:

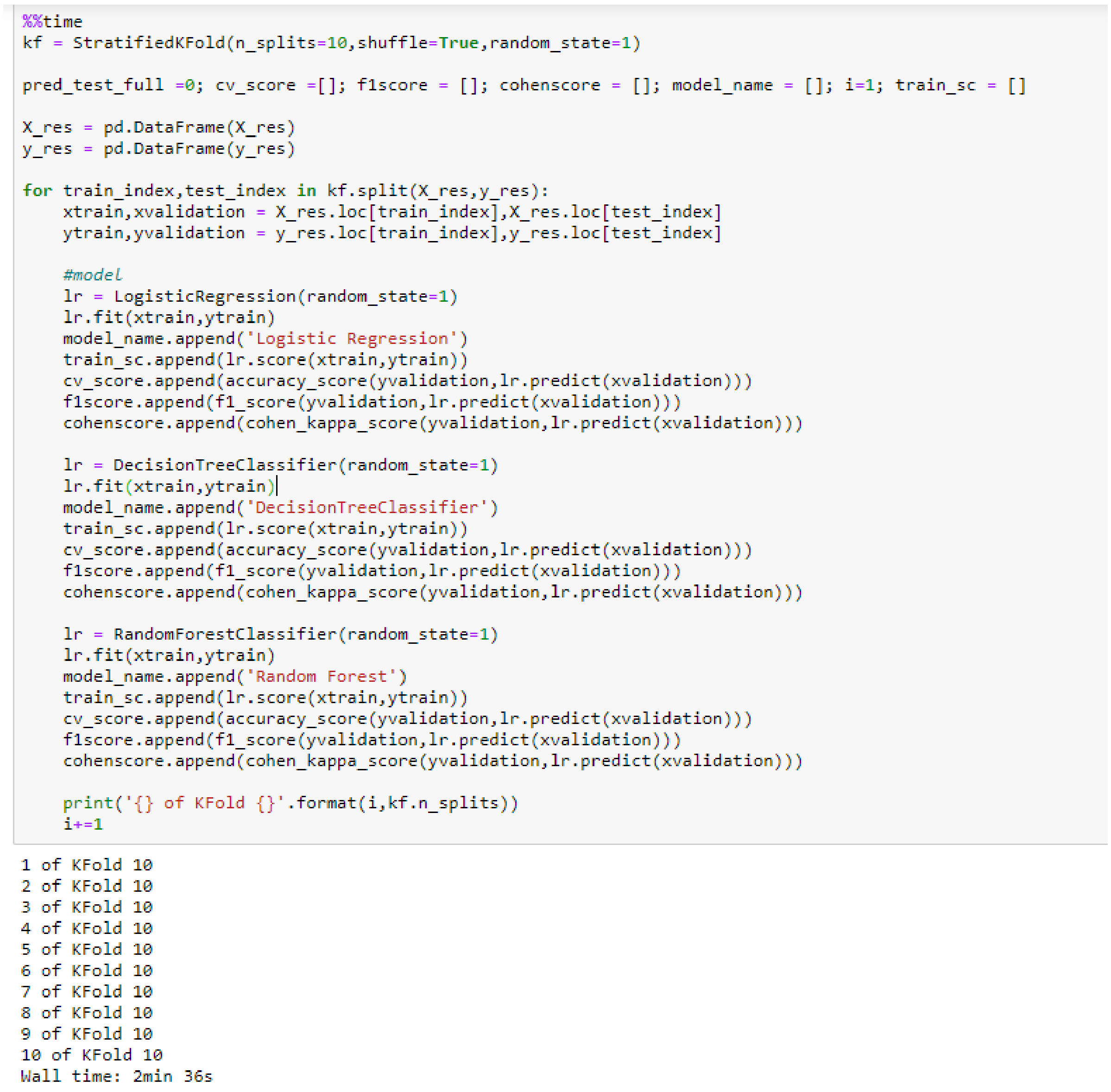
After performing the SMOTE to the original dataset (ie. without performing any transformation) we are getting good accuracy for the logistic model. but after the transformation on numeric data there is not much significant difference, hence we will choose the original dataset for further modelling.

v) Deciding Best Model based on the KFold cross validation result & boxplot

As we had got the good result after balancing the target variable, we will only use the balanced data which we obtained after performing SMOTE.

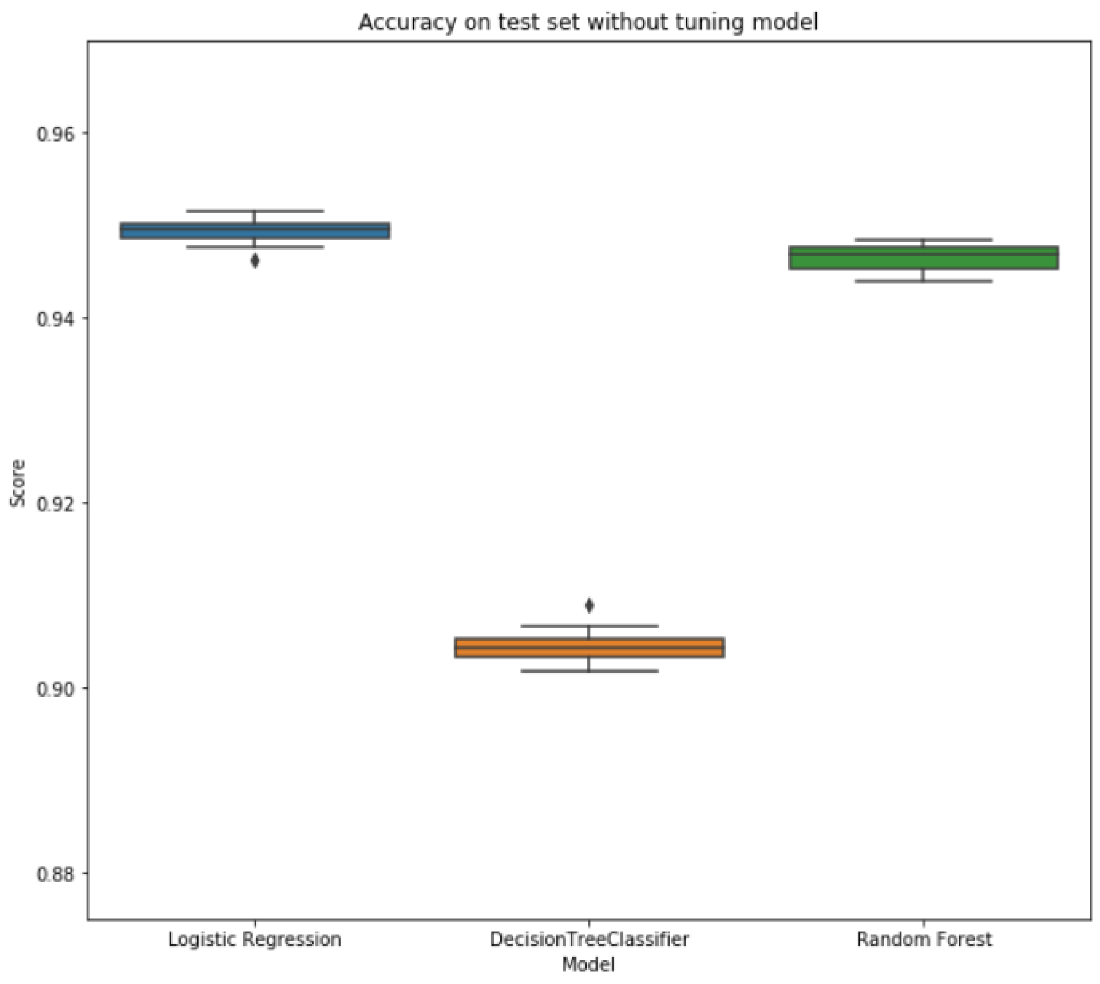
While finalizing the model only highest accuracy shouldn’t be considered. The consistency of the model is also one of the important factors. Here consistent model means model when fed with different train and test samples, should show less variance in obtained scores. To check this, we do k fold cross validation and do a box plot analysis on obtained scores.

Cross validation basically helps in learning all the rows and validating all the rows. We use cross validation to control bias and variance. Here we have split the data into 10 folds. This splitting of data happens randomly. In 10-fold it will train 9 part of data & validate 1 part of data for 10 times.



```
%%time
kf = StratifiedKFold(n_splits=10,shuffle=True,random_state=1)

pred_test_full =0; cv_score =[]; f1score = []; cohenscore = []; model_name = []; i=1; train_sc = []

X_res = pd.DataFrame(X_res)
y_res = pd.DataFrame(y_res)

for train_index,test_index in kf.split(X_res,y_res):
    xtrain,xvalidation = X_res.loc[train_index],X_res.loc[test_index]
    ytrain,yvalidation = y_res.loc[train_index],y_res.loc[test_index]

    #model
    lr = LogisticRegression(random_state=1)
    lr.fit(xtrain,ytrain)
    model_name.append('Logistic Regression')
    train_sc.append(lr.score(xtrain,ytrain))
    cv_score.append(accuracy_score(yvalidation,lr.predict(xvalidation)))
    f1score.append(f1_score(yvalidation,lr.predict(xvalidation)))
    cohenscore.append(cohen_kappa_score(yvalidation,lr.predict(xvalidation)))

    lr = DecisionTreeClassifier(random_state=1)
    lr.fit(xtrain,ytrain)
    model_name.append('DecisionTreeClassifier')
    train_sc.append(lr.score(xtrain,ytrain))
    cv_score.append(accuracy_score(yvalidation,lr.predict(xvalidation)))
    f1score.append(f1_score(yvalidation,lr.predict(xvalidation)))
    cohenscore.append(cohen_kappa_score(yvalidation,lr.predict(xvalidation)))

    lr = RandomForestClassifier(random_state=1)
    lr.fit(xtrain,ytrain)
    model_name.append('Random Forest')
    train_sc.append(lr.score(xtrain,ytrain))
    cv_score.append(accuracy_score(yvalidation,lr.predict(xvalidation)))
    f1score.append(f1_score(yvalidation,lr.predict(xvalidation)))
    cohenscore.append(cohen_kappa_score(yvalidation,lr.predict(xvalidation)))

    print('{} of KFold {}'.format(i,kf.n_splits))
    i+=1
```

```
1 of KFold 10
2 of KFold 10
3 of KFold 10
4 of KFold 10
5 of KFold 10
6 of KFold 10
7 of KFold 10
8 of KFold 10
9 of KFold 10
10 of KFold 10
Wall time: 2min 36s
```

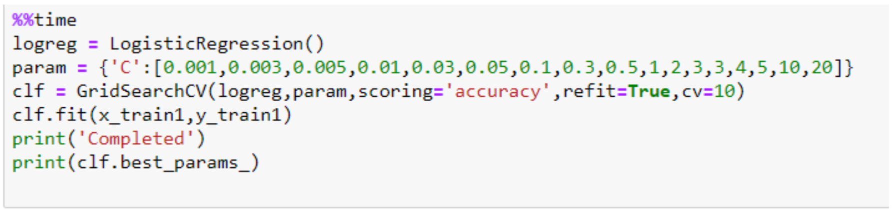


Inference: Here Logistic regression and decision tree score has the outlier. Also, the Random forest boxplot shows the variance in the score. We will not perform the grid search CV and use kfold cross validation and boxplot method for choosing the best model.

vi) Hyperparameter Tuning Using Grid Search:

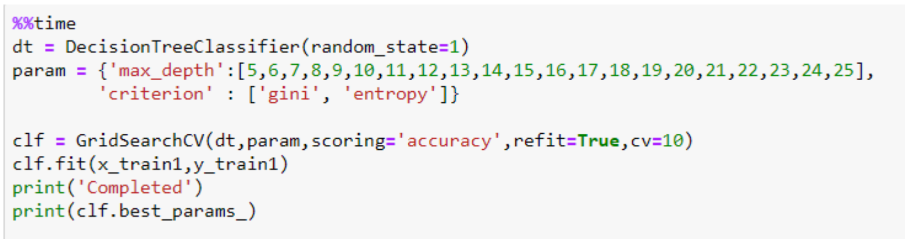
Grid-search is used to find the optimal *hyperparameters* of a model which results in the most 'accurate' predictions. Now let's tune the Parameters of the model and observe the results.

Grid Search CV Best Parameter - Logistic Regression:



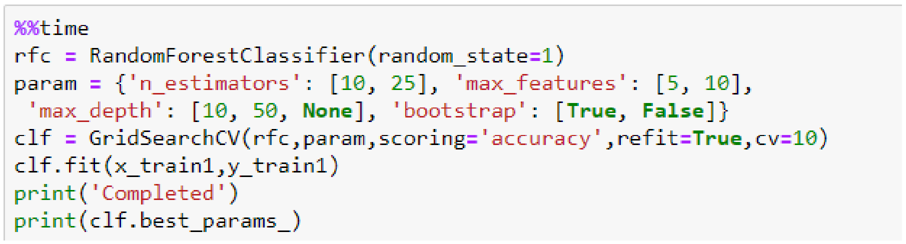
```
%%time
logreg = LogisticRegression()
param = {'C':[0.001,0.003,0.005,0.01,0.03,0.05,0.1,0.3,0.5,1,2,3,3,4,5,10,20]}
clf = GridSearchCV(logreg,param,scoring='accuracy',refit=True,cv=10)
clf.fit(x_train1,y_train1)
print('Completed')
print(clf.best_params_)
```

Grid Search CV Best Parameter - Decision Tree:



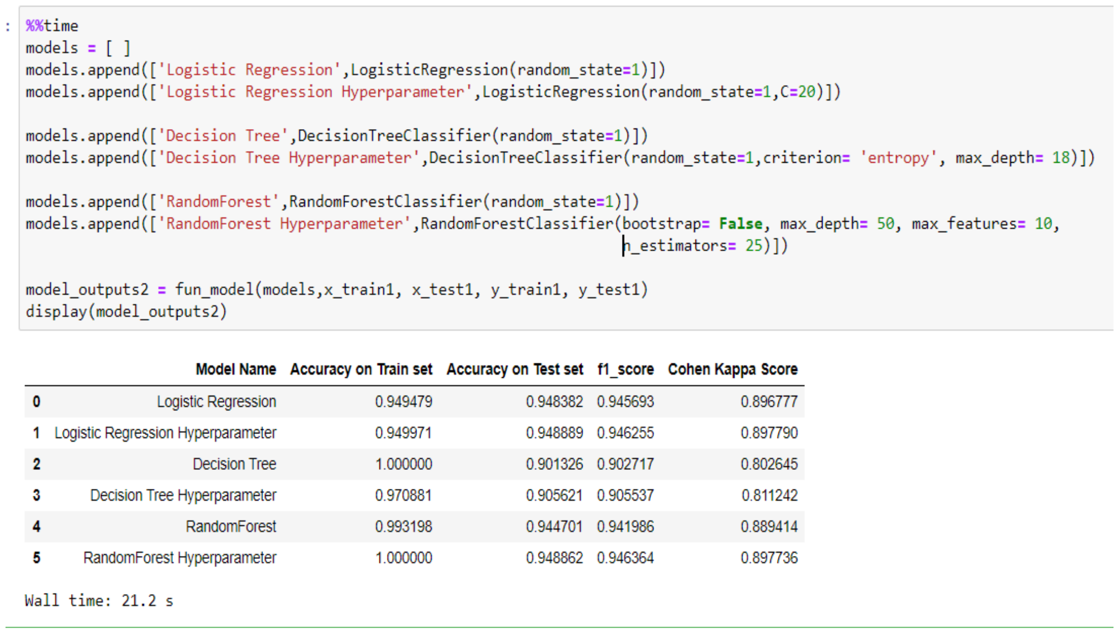
```
%%time
dt = DecisionTreeClassifier(random_state=1)
param = {'max_depth':[5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25],
        'criterion' : ['gini', 'entropy']}

clf = GridSearchCV(dt,param,scoring='accuracy',refit=True,cv=10)
clf.fit(x_train1,y_train1)
print('Completed')
print(clf.best_params_)
```

Grid Search CV Best Parameter - Random Forest:



```
%%time
rfc = RandomForestClassifier(random_state=1)
param = {'n_estimators': [10, 25], 'max_features': [5, 10],
 'max_depth': [10, 50, None], 'bootstrap': [True, False]}
clf = GridSearchCV(rfc,param,scoring='accuracy',refit=True,cv=10)
clf.fit(x_train1,y_train1)
print('Completed')
print(clf.best_params_)
```

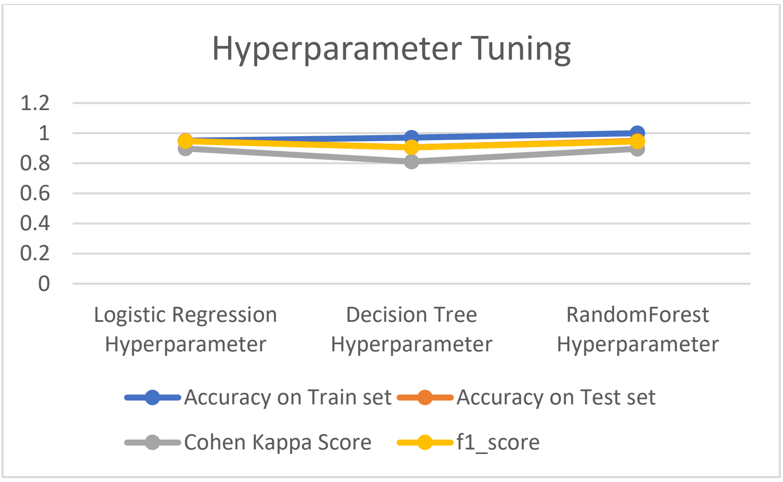
Predictions after adding hyperparameters:



```
%%time
models = [ ]
models.append(['Logistic Regression',LogisticRegression(random_state=1)])
models.append(['Logistic Regression Hyperparameter',LogisticRegression(random_state=1,C=20)])

models.append(['Decision Tree',DecisionTreeClassifier(random_state=1)])
models.append(['Decision Tree Hyperparameter',DecisionTreeClassifier(random_state=1,criterion= 'entropy', max_depth= 18)])

models.append(['RandomForest',RandomForestClassifier(random_state=1)])
models.append(['RandomForest Hyperparameter',RandomForestClassifier(bootstrap= False, max_depth= 50, max_features= 10,
                                                                    n_estimators= 25)])

model_outputs2 = fun_model(models,x_train1, x_test1, y_train1, y_test1)
display(model_outputs2)
```

	Model Name	Accuracy on Train set	Accuracy on Test set	f1_score	Cohen Kappa Score
0	Logistic Regression	0.949479	0.948382	0.945693	0.896777
1	Logistic Regression Hyperparameter	0.949971	0.948889	0.946255	0.897790
2	Decision Tree	1.000000	0.901326	0.902717	0.802645
3	Decision Tree Hyperparameter	0.970881	0.905621	0.905537	0.811242
4	RandomForest	0.993198	0.944701	0.941986	0.889414
5	RandomForest Hyperparameter	1.000000	0.948862	0.946364	0.897736

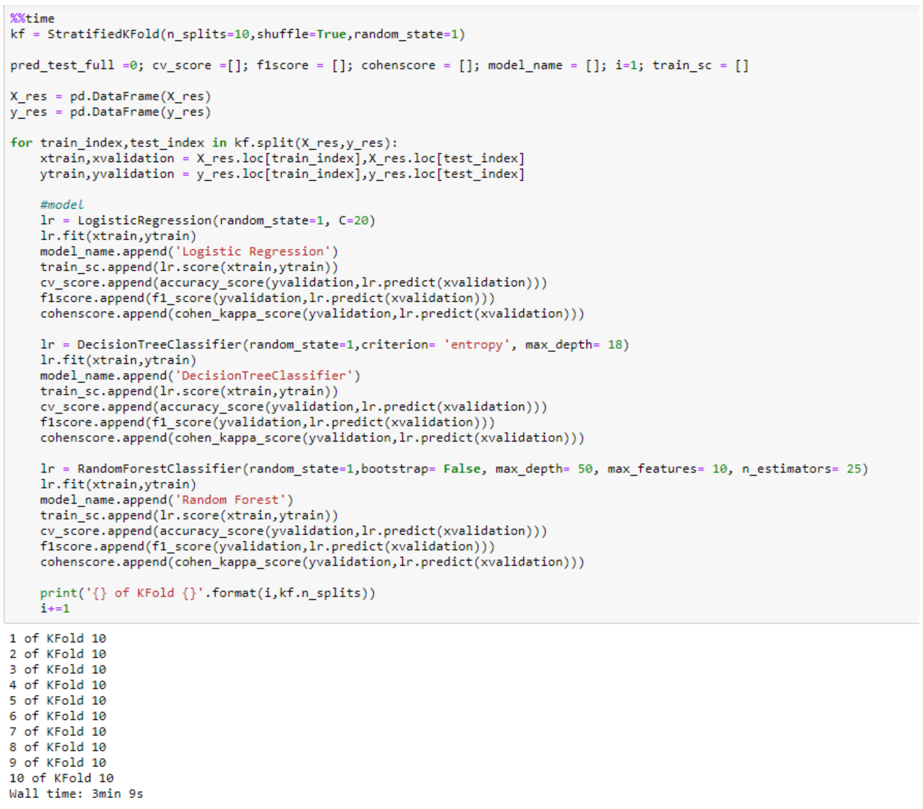
```
Wall time: 21.2 s
```



Here random forest and decision tree is giving similar score. We need to check if this models are consistent when fed with different train and testing dataset.

Deciding Best model based on best parameters and KFOLD:

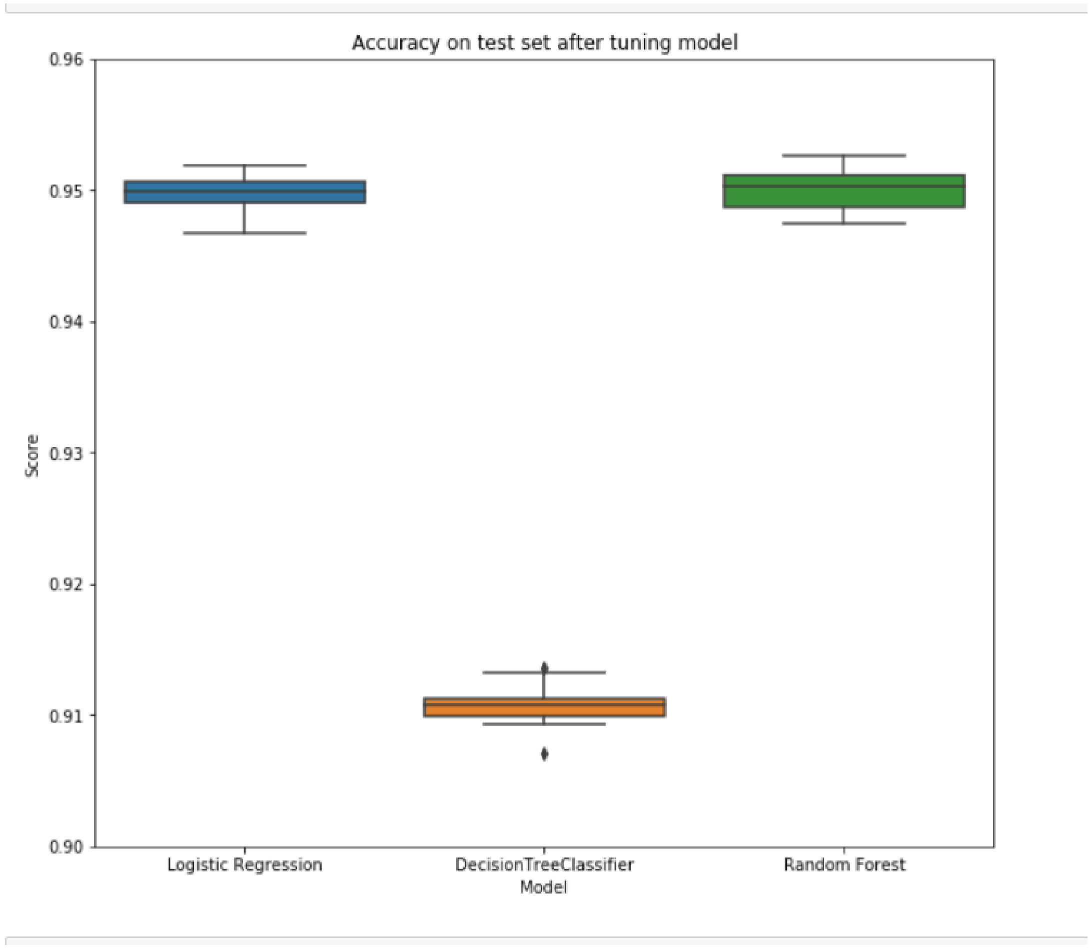
As we can observe from the above, by tuning the hyperparameter we are getting the increase in accuracy and overall scores. Now we will take the hyperparameter into consideration while choosing the best model.



```
%%time
kf = StratifiedKFold(n_splits=10,shuffle=True,random_state=1)

pred_test_full =0; cv_score =[]; f1score = []; cohenscore = []; model_name = []; i=1; train_sc = []

X_res = pd.DataFrame(X_res)
y_res = pd.DataFrame(y_res)

for train_index,test_index in kf.split(X_res,y_res):
    xtrain,xvalidation = X_res.loc[train_index],X_res.loc[test_index]
    ytrain,yvalidation = y_res.loc[train_index],y_res.loc[test_index]

    #model
    lr = LogisticRegression(random_state=1, C=20)
    lr.fit(xtrain,ytrain)
    model_name.append('Logistic Regression')
    train_sc.append(lr.score(xtrain,ytrain))
    cv_score.append(accuracy_score(yvalidation,lr.predict(xvalidation)))
    f1score.append(f1_score(yvalidation,lr.predict(xvalidation)))
    cohenscore.append(cohen_kappa_score(yvalidation,lr.predict(xvalidation)))

    lr = DecisionTreeClassifier(random_state=1,criterion= 'entropy', max_depth= 18)
    lr.fit(xtrain,ytrain)
    model_name.append('DecisionTreeClassifier')
    train_sc.append(lr.score(xtrain,ytrain))
    cv_score.append(accuracy_score(yvalidation,lr.predict(xvalidation)))
    f1score.append(f1_score(yvalidation,lr.predict(xvalidation)))
    cohenscore.append(cohen_kappa_score(yvalidation,lr.predict(xvalidation)))

    lr = RandomForestClassifier(random_state=1,bootstrap= False, max_depth= 50, max_features= 10, n_estimators= 25)
    lr.fit(xtrain,ytrain)
    model_name.append('Random Forest')
    train_sc.append(lr.score(xtrain,ytrain))
    cv_score.append(accuracy_score(yvalidation,lr.predict(xvalidation)))
    f1score.append(f1_score(yvalidation,lr.predict(xvalidation)))
    cohenscore.append(cohen_kappa_score(yvalidation,lr.predict(xvalidation)))

    print('{} of KFold {}'.format(i,kf.n_splits))
    i+=1
```

```
1 of KFold 10
2 of KFold 10
3 of KFold 10
4 of KFold 10
5 of KFold 10
6 of KFold 10
7 of KFold 10
8 of KFold 10
9 of KFold 10
10 of KFold 10
Wall time: 3min 9s
```

Displaying Best models with hyperparameter tuning using Box Plot:

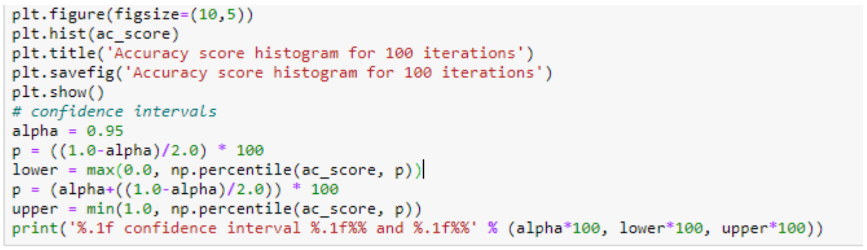


Inference: From the above three model we can see that Logistics regression have slightly less accuracy as compared to Random forest but the variance of the model Logistic Regression is low and hence, we can choose Logistics Regression for our Prediction due to its low variability.

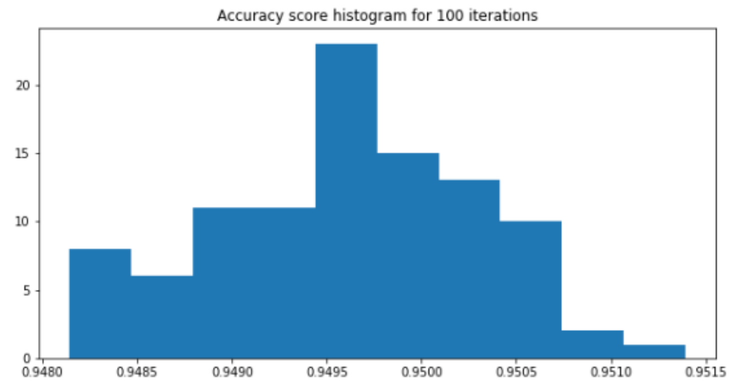
Logistic Regression Bootstrap:

In the case of Logistic Regression, the residual bootstrap and wild bootstrap both fail because the fitted value is a probability and the response value Yi = {0, 1}. Thus, the bootstrap sample from these approaches yields Y ∗ i ’s that might not be 0 or 1. But the empirical bootstrap still works. A robust way to calculate confidence intervals for machine learning algorithms is to use the bootstrap. This is a general technique for estimating statistics that can be used to calculate empirical confidence intervals, regardless of the distribution of skill scores

We will train our model on data provided on each bootstrap iteration and we will evaluate it on the out of bag samples(**OOB- some samples that are not included in the sample as we initially take sample size limited to 60%-80% for each iteration**) to give performance statistic, which can be collected and from which confidence intervals may be calculated



```
plt.figure(figsize=(10,5))
plt.hist(ac_score)
plt.title('Accuracy score histogram for 100 iterations')
plt.savefig('Accuracy score histogram for 100 iterations')
plt.show()
# confidence intervals
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, np.percentile(ac_score, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, np.percentile(ac_score, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))
```

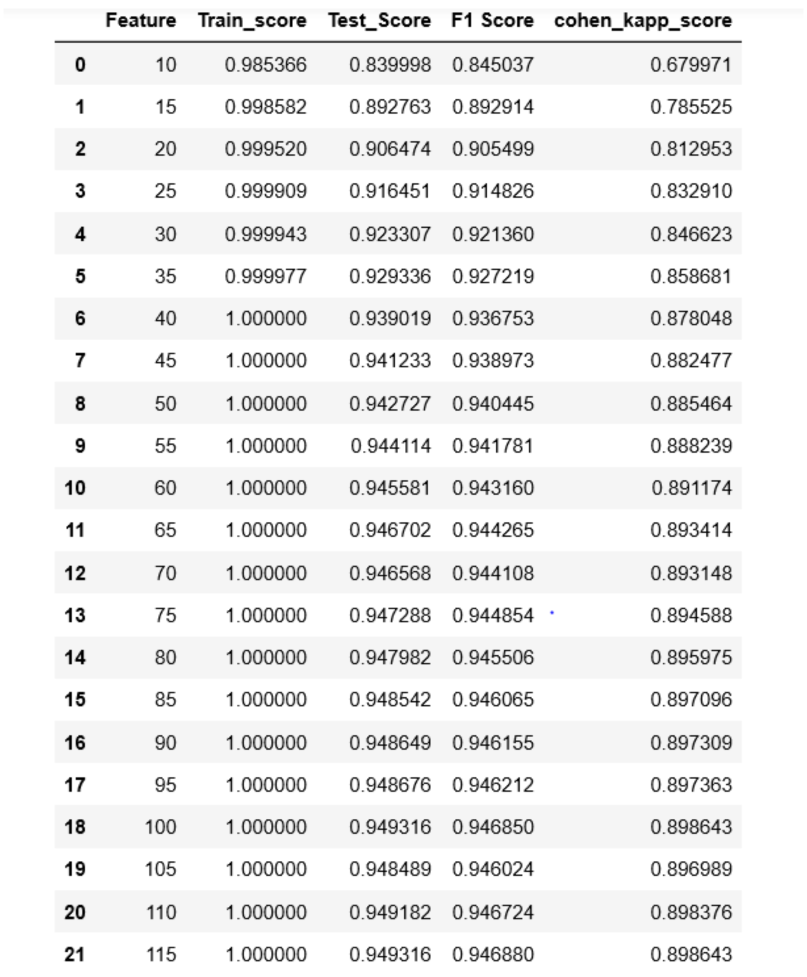


```
95.0 confidence interval 94.8% and 95.1%
```

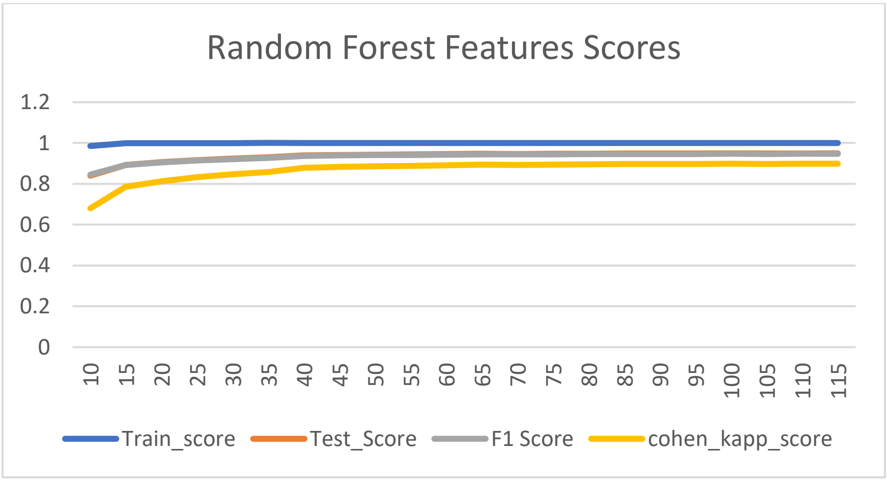
Point estimation require very detailed information which might not be the case when we don’t know about the data quality such as how it has been collected, how it is maintained etc. In that case it would not be appropriate to go with point estimate and hence interval estimate is can be used which gives us the range between which our value can lie with particular confidence limit.

In Logistic Regression we are 95% confidence that accuracy lies between 94.8% and 95.1%.

vii) Feature Selection Using Random Forest



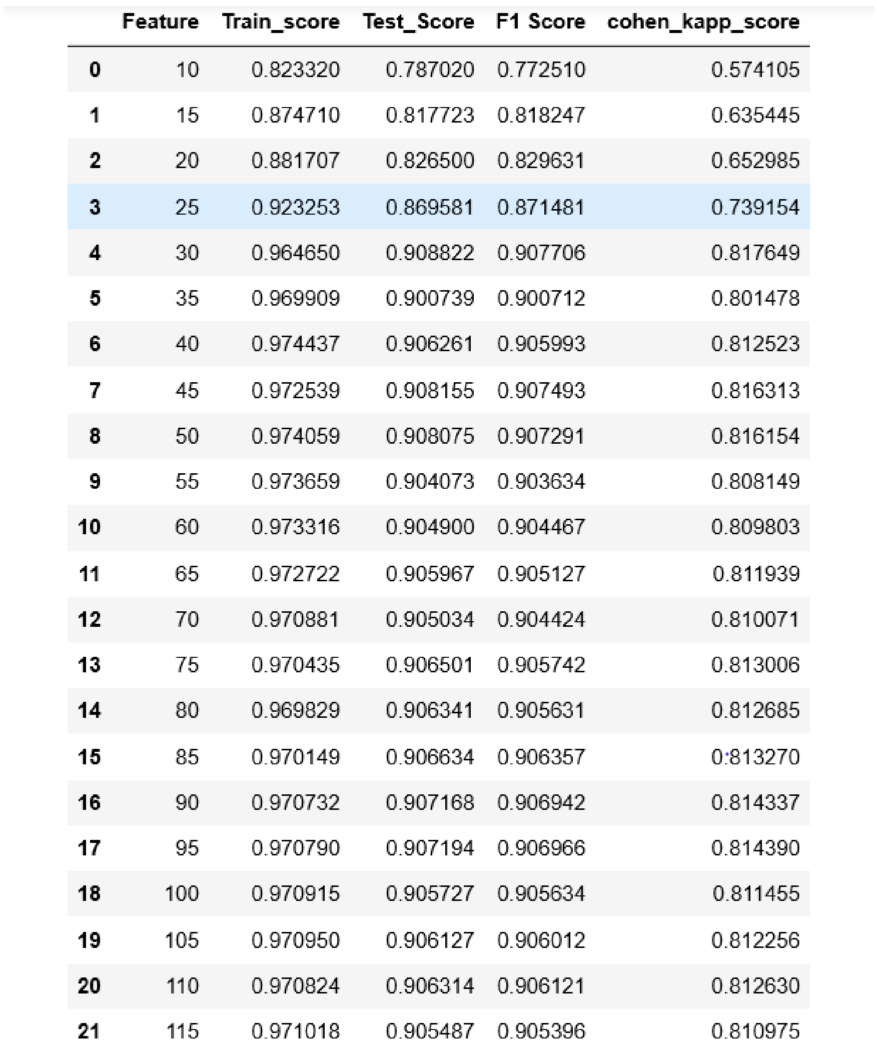
	Feature	Train_score	Test_Score	F1 Score	cohen_kapp_score
0	10	0.985366	0.839998	0.845037	0.679971
1	15	0.998582	0.892763	0.892914	0.785525
2	20	0.999520	0.906474	0.905499	0.812953
3	25	0.999909	0.916451	0.914826	0.832910
4	30	0.999943	0.923307	0.921360	0.846623
5	35	0.999977	0.929336	0.927219	0.858681
6	40	1.000000	0.939019	0.936753	0.878048
7	45	1.000000	0.941233	0.938973	0.882477
8	50	1.000000	0.942727	0.940445	0.885464
9	55	1.000000	0.944114	0.941781	0.888239
10	60	1.000000	0.945581	0.943160	0.891174
11	65	1.000000	0.946702	0.944265	0.893414
12	70	1.000000	0.946568	0.944108	0.893148
13	75	1.000000	0.947288	0.944854	0.894588
14	80	1.000000	0.947982	0.945506	0.895975
15	85	1.000000	0.948542	0.946065	0.897096
16	90	1.000000	0.948649	0.946155	0.897309
17	95	1.000000	0.948676	0.946212	0.897363
18	100	1.000000	0.949316	0.946850	0.898643
19	105	1.000000	0.948489	0.946024	0.896989
20	110	1.000000	0.949182	0.946724	0.898376
21	115	1.000000	0.949316	0.946880	0.898643



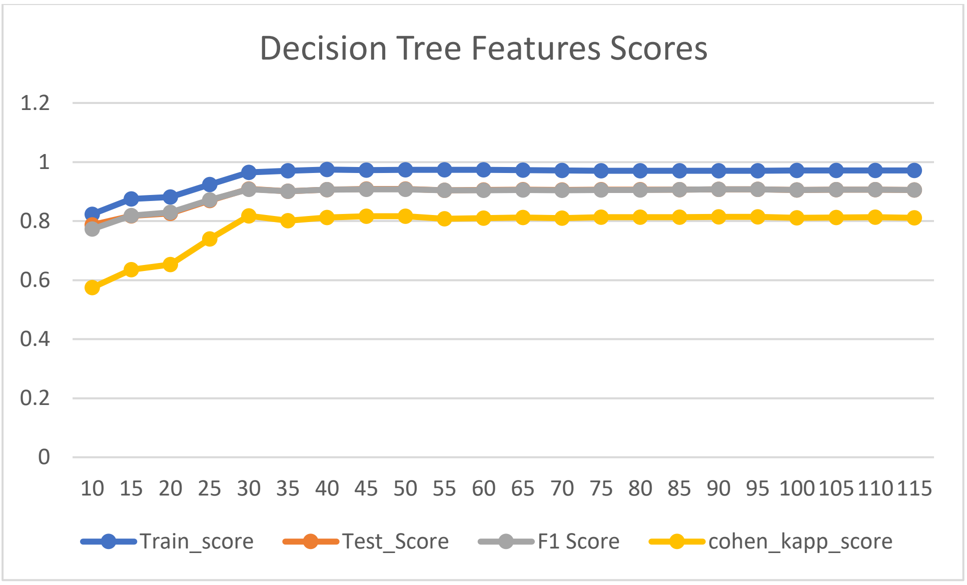
From the above table we can observe that building model with top 10 features give us a good predictability of model with an accuracy score 84% and that goes on increasing until top 50 features where we can see that change in accuracy did not have any change.

We can infer that features above 50 are mostly noise in the data and they don’t contribute much to the final predictions are not as significant than other features.

Feature Selection Using Decision Trees



	Feature	Train_score	Test_Score	F1 Score	cohen_kapp_score
0	10	0.823320	0.787020	0.772510	0.574105
1	15	0.874710	0.817723	0.818247	0.635445
2	20	0.881707	0.826500	0.829631	0.652985
3	25	0.923253	0.869581	0.871481	0.739154
4	30	0.964650	0.908822	0.907706	0.817649
5	35	0.969909	0.900739	0.900712	0.801478
6	40	0.974437	0.906261	0.905993	0.812523
7	45	0.972539	0.908155	0.907493	0.816313
8	50	0.974059	0.908075	0.907291	0.816154
9	55	0.973659	0.904073	0.903634	0.808149
10	60	0.973316	0.904900	0.904467	0.809803
11	65	0.972722	0.905967	0.905127	0.811939
12	70	0.970881	0.905034	0.904424	0.810071
13	75	0.970435	0.906501	0.905742	0.813006
14	80	0.969829	0.906341	0.905631	0.812685
15	85	0.970149	0.906634	0.906357	0.813270
16	90	0.970732	0.907168	0.906942	0.814337
17	95	0.970790	0.907194	0.906966	0.814390
18	100	0.970915	0.905727	0.905634	0.811455
19	105	0.970950	0.906127	0.906012	0.812256
20	110	0.970824	0.906314	0.906121	0.812630
21	115	0.971018	0.905487	0.905396	0.810975



In Decision Tree accuracy score increases from top 10 features and become quite stagnant after 30 features indicating that most of the features are not significant with target variable and should not be considered while prediction of the model.

14) Conclusion

As there is a strong urge for metric, we will go for Logistics Regression as the model as comparatively low variance as compared to other models. If we want to go for interpretability, we can go for Decision Tree which have low accuracy and F1 score as compared to Logistics regression.

As per Random Forest Feature importance the variables which are driving the most in hospital readmission are:

- Number of days Stayed
- Number of Procedures
- Number of medications
- Number of Diagnoses
- Age
- Discharge Disposition ID
- Diag_1
- Diag_3

15) Bibliography

- https://www.cms.gov/Medicare/Quality-Initiatives-Patient-Assessment-Instruments/Value-Based-Programs/HRRP/Hospital-Readmission-Reduction-Program
- https://en.wikipedia.org/wiki/List_of_ICD-9_codes
- https://www.researchgate.net/figure/Age-intervals-and-age-groups_tbl1_228404297
- https://machinelearningmastery.com/calculate-bootstrap-confidence-intervals-machine-learning-results-python/
- https://archive.ics.uci.edu/ml/datasets/diabetes+130-us+hospitals+for+years+1999-2008#
- https://www.hindawi.com/journals/bmri/2014/781670/