Case Study 2 – Diabetes

# Balaji Avvaru, Ravi Sivaraman, Apurv Mittal

**Abstract**

The objective of this study is to predict the re-admittance of the diabetes patients within 30 days to the medical facility after their previous stay. The patient admitted and re-admitted has various aspects about them like their demography, medication, stay in the hospital, diagnosis etc. which can play a vital role in predicting the probability of re-admittance of the patients within 30 days.

# 1. Introduction

The dataset represents 10 years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks. It includes over 50 features representing patient and hospital outcomes. The data includes the patient stay only if the following were true:

1. Stay in hospital for minimum 1 day to maximum of 14 days.
2. Patient diagnosed for diabetes
3. Lab tests were performed on the patient
4. Medication administered to the patient.

The data contains attributes such as patient number, race, gender, age, admission type, time in hospital, medical specialty of admitting physician, number of lab test performed, HbA1c test result, diagnosis, number of medications, diabetic medications, number of outpatient, inpatient, and emergency visits in the year before the hospitalization, etc.

The dataset provided has two files

1. *Diabetes\_data.csv* which contains 50 features extracted from 101,766 encounters with outcome of re-admittance.
2. *IDs\_mapping.csv* contains the codes for the IDs mentioned in the *Diabetes\_data.csv* file.

The main variables captured in the *Diabetes\_data.csv* file include:

| **Feature name** | **Description** |
| --- | --- |
| Race | Values: Caucasian, Asian, African American, Hispanic, and other |
| Gender | Values: male, female, and unknown/invalid |
| Age | Grouped in 10-year intervals: 0, 10), 10, 20), …, 90, 100) |
| Race | Values: Caucasian, Asian, African American, Hispanic, and other |
| Discharge disposition | Integer identifier corresponding to 29 distinct values, for example, discharged to home, expired, and not available |
| Admission source | Integer identifier corresponding to 21 distinct values, for example, physician referral, emergency room, and transfer from a hospital |
| Time in hospital | Integer number of days between admission and discharge |
| Payer code | Integer identifier corresponding to 23 distinct values, for example, Blue Cross*/*Blue Shield, Medicare, and self-pay |
| 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 |
| Number of lab procedures | Number of lab tests performed during the encounter |
| Number of procedures | Number of procedures (other than lab tests) performed during the encounter |
| Number of medications | Number of distinct generic names administered during the encounter |
| Number of outpatient visits | Number of outpatient visits of the patient in the year preceding the encounter |
| Number of emergency visits | Number of emergency visits of the patient in the year preceding the encounter |
| Number of inpatient visits | Number of inpatient visits of the patient in the year preceding the encounter |
| Diagnosis 1 | The primary diagnosis (coded as first three digits of ICD9); 848 distinct values |
| Diagnosis 2 | Secondary diagnosis (coded as first three digits of ICD9); 923 distinct values |
| Diagnosis 3 | Additional secondary diagnosis (coded as first three digits of ICD9); 954 distinct values |
| Number of diagnoses | Number of diagnoses entered to the system |
| 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 |
| 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. |
| Diabetes medications | Indicates if there was any diabetic medication prescribed. Values: “yes” and “no” |
| 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 |
| Readmitted (Target) | 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. |

Reference: <https://www.hindawi.com/journals/bmri/2014/781670/tab1/>

# 2. Methods

The data from the files were extracted and analyzed. First step of the data analysis is to look for any missing data. The data was identified to be incomplete, and some features were missing large percentage of data. The details of data analysis and imputation covered in the sections below.

## Missing Data

Following features has missing data (approx.):

1. Race – 2%
2. Weight – 97%
3. Payer Code – 40%
4. Medical Specialty – 50%
5. Diagnosis – 2%

## Imputation

### Race

Race is a categorical variable with values as Caucasian, Asian, African American, Hispanic, and Other. With 2% missing data, the data can be imputed with the mode of the values recorded.

Caucasian is about 76% of the data, so it appropriate to replace the missing 2% of the variables with Caucasian.

### Weight

Weight is missing 97% of the data. It’s also a categorical variable with range given for the weight of the patients in pounds. Since weight is traditionally considered to be important aspect with diabetes, it should be considered and imputed instead of ignoring or dropping the feature.

To impute the missing Weight data, the data is reviewed. The valid values captured as ranges as below:

1. 0 to 25 – Since the age associated with range is mostly of adults, it appropriate to consider this as misclassification and replace it as missing data
2. 25 to 50 – replaced this with the average of 25 and 50 as 37.5
3. 50 to 75 – replaced this with the average of 50 and 75 as 62.5
4. 75 to 100 – replaced this with the average of 75 and 100 as 87.5
5. 100 to 125 – replaced this with the average of 100 and 125 as 112.5
6. 125 to 150 – replaced this with the average of 125 and 150 as 137.5
7. 150 to 175 – replaced this with the average of 150 and 175 as 162.5
8. 175to 200 – replaced this with the average of 175 and 200 as 187.5
9. >200 – replaced this with a static value of 250

For the missing Weight data, the linear regression prediction was used. Age, Race, Gender can have direct impact on the weight of the patient. So, these variables were used to train a linear regression model and predict the missing values for Weight.

The predicted values of Weight were used for data imputation.

### medical\_specialty

This is integer variable; this will tell the specialty of the admitting physician. This variable has around 50 percent of data missing. We Imputed the missing values in medical\_specialty based on category ratios. There are 72 unique values for this variable. Instead of adding 73 new variables since some of them only have a few samples as an alternative, we can create a new variable that only has 9 options (the top 8 specialties and then another category).

### Payer\_code

This is object variable; this variable will tell you the insurance of the patient. This variable has 17 unique values. This variable has around 40 percent of data missing. We Imputed the missing values in payer\_code based on category ratios.

### discharge\_disposition\_id

There are 30 distinct values. This variable will tell whether the patient discharged to home or any other facility etc. We removed observations where the discharge disposition is related to death since these will not be readmitted. We can make all these observations with respect to discharge\_disposition\_id into two different categories Home and other. The categories 18-NULL 25-Not Mapped and 26-Unknown/Invalid are classified as NULL values. We need imputation for these categories. We decided to combine these categories with other categories.

### Admission Source ID

There are 26 distinct values. This variable will tell whether the patient admitted because of physician referral or clinic referral etc. We converted all observations with respect to admission source into three different categories like, Referral, Emergency or Other.

max\_glu\_serum is categorical column with values, None, >300, Norm, >200, so the column is one hot encoded.

A1cResult is categorical column with values, None, >7,>8 and Norm, so this column is one hot encoded.

number\_diagnoses, num\_medications, number\_outpatient , number\_emergency, number\_inpatient are all integers, so they are left as is.

### Diagnosis

diag\_1, diag\_2 and diag\_3 are categories with several unique values, as described in the page <https://www.hindawi.com/journals/bmri/2014/781670/tab2/>.

If these values are one-hot encoded, the number of columns exceeds over 2000 which makes running the model quite computationally expensive; number of unique values in diag\_1, diag\_2 and diag\_3 is 2256.

As described in the link above, the various diagnosis values are combined together into a fewer groups and then one-hot encoding them to keep the number of columns manageable. The missing values are imputed as Other and one-hot encoded.

For example, combining any values that are between 390–459 or 785 as Circulatory, and so on as described in the above-mentioned URL. Now, there are only 27 columns, makes the model run more efficiently.

The rest of the columns are metformin, repaglinide, nateglinide, chlorpropamide,

glimepiride, acetohexamide, glipizide, glyburide, tolbutamide, pioglitazone, rosiglitazone, acarbose, miglitol, troglitazone, tolazamide, examide,

citoglipton, insulin, glyburide-metformin, glipizide-metformin, glimepiride-pioglitazone, metformin-rosiglitazone, metformin-pioglitazone, change, diabetesMed are all medicines which indicates they have taken the dosage.

The features like medicines taken, change and diabetesMed are all categorical, they are all one-hot encoded.

## Target

Readmitted feature is the target variable, which has following values:

1. NO – 54%
2. > 30 – 35%
3. < 30 – 11%

The objective is to predict the probability of admittance within 30 days from initial hospital stay. To make binary classification, the values NO and >30 are combined as 0 (False) and <30 as 1 (True).

A picture containing shape

Description automatically generated

Figure 1:Distribution of readmitted (target variable)

The above graph indicates the target variable is heavily imbalanced, and the prediction could be poor. To overcome this issue, the class ‘balanced’ is used on LogisticRegression models.

### Data Wrangling

The attributes are normalized using StandardScaler to scale between 0 and 1 before running models.

Stratified k-fold validation algorithm is used for this model on which the original sample is randomly partitioned into *k* equal size subsamples in which each fold contains roughly the same proportions of class labels.

The advantage of Stratified k-fold method is that all observations are used for both training and validation, and each observation is used for validation exactly once.

Models were run on using 10k fold.

## Regression Models

The model used for this report is LogisticRegression, as the objective is a binary classification.

For classification model, *accuracy*, *precision* and *recall* are calculated.

The *precision* for the classification model represents the ratio of the patients that the model correctly identifies as a positive readmission out of all the patients the model identifies as positive. The *recall* (also referred to as *sensitivity*), represents the ability of the model to correctly identify true positive readmission. The accuracy represents how well the model classifies correctly both positive and negative readmission.

To identify the most efficient model, GridSearch algorithm is used. GridSearch run with various parameters and then compares the *accuracy* (for this report) and chooses the best parameters for the LogisticRegression.

Parameters were used for calculating the best model for *accuracy*, the following values are used:

* *penalty*: used to specify the norm used in the penalization (values used: 'l1', 'l2', 'elasticnet', 'none')
* *C*: Inverse of regularization strength (0.001, 0.01, 0.1, 1, 10, 100, 1000)
* *class\_weight*: The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data (value: balanced).
* *solver*: algorithm to use in the optimization problem ('newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga')

# 3. Results

The GridSearch produced 280 models of which the model with highest accuracy had following parameters:

|  |  |
| --- | --- |
| alpha | .001 |
| class weight | Balanced |
| Penalty | LASSO(l1) |
| Algorithm | Saga |

### Classification report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-Score | Support |
| 1 (Readmitted) | 0.18 | 0.54 | 0.26 | 11357 |
| 0 (Not Readmitted) | 0.92 | 0.68 | 0.78 | 88757 |
| Accuracy |  |  | 0.66 | 100114 |
| Macro avg | 0.55 | 0.61 | 0.52 | 100114 |
| Weighted avg | 0.84 | 0.66 | 0.72 | 100114 |

### ROC Curve

The ROC curve shows the relationship between the model's ability to correctly vs incorrectly classifies if readmitted or not with in 30 days. This could potentially make adjustments that could shift the ROC curve to have a higher true positive rate at the expense of increasing the false positive rate.

A picture containing diagram

Description automatically generated

Figure 2:ROC Curve (True Positive vs False Positive)

The ROC curve for the logistic regression classifier showed a mean area under the curve AUC of 0.65.

The accuracy and AUC are not very high. The F1-score is reasonable for category 0 (not readmitted), it is poor for category 1 (readmitted), and predicting readmittance is the objective of the study. This model will not be able to predict the probability of readmittance with high confidence. Other machine learning models might be required to investigate this further and get a better classification.

# Conclusion

### The features leading to higher readmittance within 30 days are (those with positive co-efficient) and the features that reduce the chance of readmittance within 30 days (those with negative co-efficient) are the most significant features.

| Features | Coefficients |
| --- | --- |
| Discharged\_to\_Home | -0.088361 |
| Diag1\_Respiratory | -0.022072 |
| diabetesMed\_No | -0.013376 |
| insulin\_No | -0.010145 |
| Diag1\_Circulatory | 0.001348 |
| metformin\_No | 0.002332 |
| num\_medications | 0.013085 |
| age | 0.013088 |
| diabetesMed\_Yes | 0.013376 |
| insulin\_Down | 0.01738 |
| number\_emergency | 0.019839 |
| time\_in\_hospital | 0.021283 |
| number\_diagnoses | 0.071881 |
| Discharged\_to\_Other | 0.088361 |
| number\_inpatient | 0.348032 |

Chart, bar chart

Description automatically generated

Figure 4:Feature Importance based on co-efficient

The above plot displays the features with their coefficient levels. These are all non-zero coefficient after L1 penalty.

# Appendix – Code

Nbviewer link: <https://nbviewer.org/github/ravisiv/QTW-CaseStudy2/blob/main/Case%20Study%202%20-%20Balaji-Apurv-Ravi.ipynb>

## Python Jupyter Notebook

# Case Study 2: Hospital Readmission[¶](#Case-Study-2-:-Hospital-Readmission)

The code for importing the data is combined with the initial loading of various analysis and visualization packages below.

In [ ]:

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from scipy import stats

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

from sklearn.feature\_selection import VarianceThreshold

from sklearn import linear\_model

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.model\_selection import KFold

from sklearn.model\_selection import GridSearchCV

from sklearn.feature\_selection import RFE

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression

In [ ]:

# read diabetic csv files

diabetic\_df = pd.read\_csv('/Users/ravis/Downloads/dataset\_diabetes/diabetic\_data.csv')

#diabetic\_df = pd.read\_csv('diabetic\_data.csv')

diabetic\_df.shape

In [ ]:

# Validate null values in the csv file

diabetic\_df.isnull().sum().sum()

There are no null values in the dataset, but null values are coded as ?, first replace observations value ? with nan

In [ ]:

# replace ? with nan

diabetic\_df = diabetic\_df.replace("?",np.nan)

In [ ]:

diabetic\_df.encounter\_id.nunique()

In [ ]:

# drop encounter\_id variable as it is unique id - https://www.hindawi.com/journals/bmri/2014/781670/tab1/

diabetic\_df.drop(['encounter\_id'], inplace=True, axis=1)

In [ ]:

# drop patient\_nbr Unique identifier of a patient - https://www.hindawi.com/journals/bmri/2014/781670/tab1/

diabetic\_df.drop(['patient\_nbr'], inplace=True, axis=1)

In [ ]:

# Validate null values in the csv file

print(diabetic\_df.isnull().sum())

# print columns with null values

missing\_data\_columns = diabetic\_df.columns[diabetic\_df.isnull().any()]

print("\n\nColumns with null values")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

missing\_data\_columns

In [ ]:

# percentage of missing values in each variable

diabetic\_df[missing\_data\_columns].isnull().sum()/len(diabetic\_df)\*100

In [ ]:

diabetic\_df.info()

##### Target variable[¶](#Target-variable)

In [ ]:

# Convert target variable to a categorical

diabetic\_df['readmitted'].value\_counts()

There are three different values for target variable. We decided to reduce these values to two based on whether readmitted ot not (0 means not admitted and 1 means admitted)

In [ ]:

diabetic\_df['readmitted'] = diabetic\_df['readmitted'].replace('>30', 0)

diabetic\_df['readmitted'] = diabetic\_df['readmitted'].replace('<30', 1)

diabetic\_df['readmitted'] = diabetic\_df['readmitted'].replace('NO', 0)

In [ ]:

diabetic\_df['readmitted'].value\_counts()

In [ ]:

sns.countplot(x = "readmitted", data = diabetic\_df)

plt.title("Distribution of Target Values")

plt.show()

In [ ]:

# Pie chart

diabetic\_df.readmitted.value\_counts().plot.pie(autopct = "%.1f%%")

plt.title("Proportion of Target Value")

plt.show()

These two diagrams show us that our target variable is not balanced.

##### Variables with Missing data[¶](#Variables-with-Missing-data)

In [ ]:

missing\_data\_columns

#### Analyse Variables[¶](#Analyse-Variables)

In [ ]:

data\_reduced = diabetic\_df[["race", "gender", "age", "weight"]]

In [ ]:

# Gender

data\_reduced["gender"] = [1 if x=="Male" else 0 for x in data\_reduced["gender"]]

In [ ]:

# race

data\_reduced['race'].value\_counts()

In [ ]:

# replacing the missing race values with Caucasian

data\_reduced['race'] = data\_reduced['race'].fillna("Caucasian")

In [ ]:

# One hot encoding Race

data\_reduced\_race = pd.get\_dummies(data\_reduced.race, prefix='race')

data\_reduced\_race=data\_reduced\_race.astype(float)

In [ ]:

# convert the age in numeric

age\_num = {'[0-10)': 5, '[10-20)': 15, '[20-30)': 25, '[30-40)': 35, '[40-50)': 45,

'[50-60)': 55, '[70-80)': 75, '[60-70)': 65, '[80-90)': 85, '[90-100)': 95 }

data\_reduced["age"].replace(age\_num, inplace=True)

print(data\_reduced["age"].value\_counts())

In [ ]:

#Concat

# creating DF of one hot encoded values

data = pd.concat([data\_reduced,data\_reduced\_race],axis=1)

In [ ]:

# drop race as its duplicate now

data=data.drop(["race"], axis=1)

In [ ]:

# checking values for weight

print("Weight values",data['weight'].value\_counts())

# replacinf Weight values with average of the range

data['weight'] = data['weight'].replace('[0-25)', np.nan)

data['weight'] = data['weight'].replace('[25-50)', 37.5)

data['weight'] = data['weight'].replace('[50-75)', 62.5)

data['weight'] = data['weight'].replace('[75-100)', 87.5)

data['weight'] = data['weight'].replace('[100-125)', 112.5)

data['weight'] = data['weight'].replace('[125-150)', 137.5)

data['weight'] = data['weight'].replace('[150-175)', 162.5)

data['weight'] = data['weight'].replace('[175-200)', 187.5)

data['weight'] = data['weight'].replace('>200', 250)

In [ ]:

# assigning all missing data rows to test data set

test\_data = data[data["weight"].isna()]

In [ ]:

# running regression to predict missing weight values

df\_wona = data.dropna(axis=0)

y\_train = df\_wona["weight"]

X\_train = df\_wona.drop("weight", axis=1)

X\_test = test\_data.drop("weight", axis=1)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

# reference: https://towardsdatascience.com/7-ways-to-handle-missing-values-in-machine-learning-1a6326adf79e

In [ ]:

X\_test['weight'] = y\_pred

df\_new = pd.concat([df\_wona, X\_test], axis=0)

df\_new.sort\_index(axis = 0)

diabetic\_df.drop(['weight','age', 'race', 'gender'], axis =1, inplace=True)

diabetic\_df = pd.concat([diabetic\_df, df\_new], axis=1)

In [ ]:

# Discharge Disposition ID

diabetic\_df['discharge\_disposition\_id'].describe()

There are 30 distinct values. This variable will tell whether the patient discharged to home or any other facility etc

* 1 Discharged to home
* 2 Discharged/transferred to another short term hospital
* 3 Discharged/transferred to SNF
* 4 Discharged/transferred to ICF
* 5 Discharged/transferred to another type of inpatient care institution
* 6 Discharged/transferred to home with home health service
* 7 Left AMA
* 8 Discharged/transferred to home under care of Home IV provider
* 9 Admitted as an inpatient to this hospital
* 10 Neonate discharged to another hospital for neonatal aftercare
* 11 Expired
* 12 Still patient or expected to return for outpatient services
* 13 Hospice / home
* 14 Hospice / medical facility
* 15 Discharged/transferred within this institution to Medicare approved swing bed
* 16 Discharged/transferred/referred another institution for outpatient services
* 17 Discharged/transferred/referred to this institution for outpatient services
* 18 NULL
* 19 Expired at home. Medicaid only, hospice.
* 20 Expired in a medical facility. Medicaid only, hospice.
* 21 Expired, place unknown. Medicaid only, hospice.
* 22 Discharged/transferred to another rehab fac including rehab units of a hospital .
* 23 Discharged/transferred to a long term care hospital.
* 24 Discharged/transferred to a nursing facility certified under Medicaid but not certified under Medicare.
* 25 Not Mapped
* 26 Unknown/Invalid
* 30 Discharged/transferred to another Type of Health Care Institution not Defined Elsewhere
* 27 Discharged/transferred to a federal health care facility.
* 28 Discharged/transferred/referred to a psychiatric hospital of psychiatric distinct part unit of a hospital
* 29 Discharged/transferred to a Critical Access Hospital (CAH).

In [ ]:

diabetic\_df['discharge\_disposition\_id'].value\_counts()

we remove observations where the discharge disposition is related to death since these will not add to the possibility of being readmitted.

In [ ]:

indexNames = diabetic\_df[diabetic\_df['discharge\_disposition\_id'].isin([11,19,20,21])].index

diabetic\_df.drop(indexNames, inplace=True)

We can make all these observations with respect to discharge\_disposition\_id into two different categories Home and other. The categories 18-NULL, 25-Not Mapped and 26-Unknown/Invalid are comes under NULL values. We need imputation for these categories. We decided to combine these categories with other categories.

In [ ]:

sns.countplot(x ="discharge\_disposition\_id", data = diabetic\_df)

plt.title("Original Distribution of Discharge Disposition ID Values")

plt.show()

In [ ]:

mapped\_discharge = {1:"Home",6:"Home",8:"Home",

18:"Other",25:"Other",26:"Other",

2:"Other",3:"Other",4:"Other",

5:"Other",7:"Other",9:"Other",

10:"Other",12:"Other",15:"Other",

13:"Other",14:"Other",

16:"Other",17:"Other",22:"Other",

23:"Other",24:"Other",27:"Other",

28:"Other",29:"Other",30:"Other"}

diabetic\_df["discharge\_disposition\_id"] = diabetic\_df["discharge\_disposition\_id"].replace(mapped\_discharge)

In [ ]:

sns.countplot(x ="discharge\_disposition\_id", data = diabetic\_df)

plt.title("Distribution of Discharge Disposition ID Values after conversion")

plt.show()

In [ ]:

print("Proportions of ID's")

print(diabetic\_df.discharge\_disposition\_id.value\_counts())

In [ ]:

sns.countplot(x ="discharge\_disposition\_id", hue = "readmitted", data = diabetic\_df)

plt.title("Discharge disposition id - Readmitted")

plt.show()

In [ ]:

diabetic\_df['discharge\_disposition\_id']

##### Admission Source ID[¶](#Admission-Source-ID)

In [ ]:

diabetic\_df['admission\_source\_id'].describe()

There are 26 distinct values. This variable will tell whether the patient admitted becuase of physician referral or clinic referral etc

* 1 Physician Referral
* 2 Clinic Referral
* 3 HMO Referral
* 4 Transfer from a hospital
* 5 Transfer from a Skilled Nursing Facility (SNF)
* 6 Transfer from another health care facility
* 7 Emergency Room
* 8 Court/Law Enforcement
* 9 Not Available
* 10 Transfer from critial access hospital
* 11 Normal Delivery
* 12 Premature Delivery
* 13 Sick Baby
* 14 Extramural Birth
* 15 Not Available
* 17 NULL
* 18 Transfer From Another Home Health Agency
* 19 Readmission to Same Home Health Agency
* 20 Not Mapped
* 21 Unknown/Invalid
* 22 Transfer from hospital inpt/same fac reslt in a sep claim
* 23 Born inside this hospital
* 24 Born outside this hospital
* 25 Transfer from Ambulatory Surgery Center
* 26 Transfer from Hospice

In [ ]:

sns.countplot(x ="admission\_source\_id", data = diabetic\_df)

plt.title("Original Distribution of Admission source ID Values")

plt.show()

We can make all observations with respect to admission source into three different categories like, Referral, Emergency or Other

In [ ]:

# map the admission source to three categories

mapped\_adm = {1:"Referral",2:"Referral",3:"Referral",

4:"Other",5:"Other",6:"Other",10:"Other",22:"Other",25:"Other",

9:"Other",8:"Other",14:"Other",13:"Other",11:"Other",

15:"Other",17:"Emergency",20:"Referral",21:"Other",

7:"Emergency"}

# replace the admisision values to three categories

diabetic\_df.admission\_source\_id = diabetic\_df.admission\_source\_id.replace(mapped\_adm)

In [ ]:

sns.countplot(x = "admission\_source\_id", data = diabetic\_df)

plt.title("Distribution of admission\_source ID ID Values after conversion")

plt.show()

In [ ]:

print(diabetic\_df.admission\_source\_id.value\_counts())

In [ ]:

sns.countplot(x = "admission\_source\_id", hue = "readmitted", data = diabetic\_df)

plt.title("Admission Source - Readmitted")

plt.show()

In [ ]:

diabetic\_df.columns

### max\_glu\_serum column[¶](#max_glu_serum-column)

In [ ]:

diabetic\_df["max\_glu\_serum"].unique()

In [ ]:

# Get one hot encoding of A1Cresult

one\_hot\_max\_glu\_s = pd.get\_dummies(diabetic\_df["max\_glu\_serum"], prefix="mgs")

# Drop column diabetic\_df as it is now encoded

diabetic\_df = diabetic\_df.drop('max\_glu\_serum',axis = 1)

# Join the encoded df

diabetic\_df = diabetic\_df.join(one\_hot\_max\_glu\_s)

### a1c[¶](#a1c)

In [ ]:

# Get one hot encoding of A1Cresult

one\_hot\_a1c = pd.get\_dummies(diabetic\_df["A1Cresult"], prefix="a1c")

# Drop column diabetic\_df as it is now encoded

diabetic\_df = diabetic\_df.drop('A1Cresult',axis = 1)

# Join the encoded df

diabetic\_df = diabetic\_df.join(one\_hot\_a1c)

### number\_diagnoses[¶](#number_diagnoses)

leaving them as int.

In [ ]:

diabetic\_df["number\_inpatient"].value\_counts()

### num\_medications[¶](#num_medications)

### number\_outpatient[¶](#number_outpatient)

### number\_emergency[¶](#number_emergency)

### number\_inpatient[¶](#number_inpatient)

In [ ]:

diabetic\_df["glipizide"].value\_counts()

##### time\_in\_hospital[¶](#time_in_hospital)

This is integer variable, this will tell how many days patient stayed in the hospital

In [ ]:

diabetic\_df['time\_in\_hospital'].describe()

In [ ]:

sns.countplot(x="time\_in\_hospital", data = diabetic\_df,

order = diabetic\_df.time\_in\_hospital.value\_counts().index)

plt.title("Distribution of Time in Hospital")

fig = plt.figure(figsize=(10,8))

plt.show()

In [ ]:

fig = plt.figure(figsize=(10,5))

#readmitted = 0

ax = sns.kdeplot(diabetic\_df.loc[(diabetic\_df.readmitted == 0), "time\_in\_hospital"],

color = "b", shade = True, label = "Not Readmitted")

ax = sns.kdeplot(diabetic\_df.loc[(diabetic\_df.readmitted == 1), "time\_in\_hospital"],

color = "r", shade = True, label = "Readmitted")

ax.legend(loc="upper right")

ax.set\_xlabel("Time in Hospital")

ax.set\_ylabel("Frequency")

ax.set\_title("Time in Hospital - Readmission")

plt.show()

##### payer\_code[¶](#payer_code)

This is object variable, this variable will tell you the insurance of the patient. This variable has 17 unique values

In [ ]:

print(diabetic\_df.payer\_code.value\_counts())

In [ ]:

sns.countplot(x = "payer\_code", data = diabetic\_df)

plt.title("Distribution of Payer Code")

plt.show()

Impute the missing values in payer\_code based on frequency

In [ ]:

s = diabetic\_df.payer\_code.value\_counts(normalize=True)

diabetic\_df['payer\_code\_impu'] = diabetic\_df['payer\_code']

diabetic\_df.loc[diabetic\_df.payer\_code.isna(), 'payer\_code\_impu'] = np.random.choice(s.index, p=s.values,

size=diabetic\_df.payer\_code.isna().sum())

diabetic\_df

In [ ]:

diabetic\_df.payer\_code\_impu.isna().sum()

In [ ]:

print(diabetic\_df.payer\_code\_impu.value\_counts())

##### medical\_specialty[¶](#medical_specialty)

This is integer variable, this will tell the specialty of the admitting physician. There are 72 unique values for this variable

In [ ]:

print(diabetic\_df.medical\_specialty.value\_counts())

In [ ]:

# Impute the missing values in medical\_specialty based on frequency

m = diabetic\_df.medical\_specialty.value\_counts(normalize=True)

diabetic\_df['medical\_specialty\_impu'] = diabetic\_df['medical\_specialty']

diabetic\_df.loc[diabetic\_df.medical\_specialty.isna(), 'medical\_specialty\_impu'] = np.random.choice(m.index, p=m.values,

size=diabetic\_df.medical\_specialty.isna().sum())

diabetic\_df

In [ ]:

print(diabetic\_df.medical\_specialty\_impu.value\_counts())

We can convert all these observations with respect to medical\_specialty into 9 or 10 different categories

In [ ]:

top\_10=['InternalMedicine','Family/GeneralPractice','Cardiology','Surgery-General',

"Nephrology","Orthopedics","Radiologist","Pulmonology"]

diabetic\_df.loc[~diabetic\_df['medical\_specialty\_impu'].isin(top\_10),'medical\_specialty\_impu']='Other'

In [ ]:

sns.countplot(x = "medical\_specialty\_impu", data = diabetic\_df)

plt.title("Distribution of Medical Specialty")

plt.xticks(rotation = 90)

plt.show()

In [ ]:

sns.countplot(x = "medical\_specialty\_impu",hue = "readmitted", data = diabetic\_df)

plt.title("Medical Specialty Source - Readmitted")

plt.xticks(rotation = 90)

plt.show()

##### num\_lab\_procedures[¶](#num_lab_procedures)

This is integer and this will tell number of lab tests performed during hospital stay

In [ ]:

sns.countplot(x = "num\_lab\_procedures", data = diabetic\_df)

plt.title("Distribution of Number of lab procedures")

plt.show()

In [ ]:

print("Proportions of Column")

print(diabetic\_df.num\_lab\_procedures.value\_counts().head(10))

In [ ]:

fig = plt.figure(figsize=(10,5))

#readmitted = 0

ax = sns.kdeplot(diabetic\_df.loc[(diabetic\_df.readmitted == 0), "num\_lab\_procedures"],

color = "b", shade = True,label = "Not Readmitted")

#readmitted = 1

ax = sns.kdeplot(diabetic\_df.loc[(diabetic\_df.readmitted == 1), "num\_lab\_procedures"],

color = "r", shade = True, label = "Readmitted")

ax.legend(loc="upper right")

ax.set\_xlabel("Number of Lab Procedures")

ax.set\_ylabel("Frequency")

ax.set\_title("Number of Lab Procedures - Readmission")

plt.show()

##### num\_procedures[¶](#num_procedures)

Number of procedures (other than lab tests) performed during the encounter

In [ ]:

sns.countplot(x = diabetic\_df.num\_procedures, order = diabetic\_df.num\_procedures.value\_counts().index)

plt.title("Distribution of Number of Procedures")

plt.show()

In [ ]:

print("Proportions of Values")

print(diabetic\_df.num\_procedures.value\_counts())

In [ ]:

sns.countplot(x = "num\_procedures", hue = "readmitted",

data = diabetic\_df, order = diabetic\_df.num\_procedures.value\_counts().index)

plt.title("Number of procedures - Readmitted")

plt.show()

##### glipizide[¶](#glipizide)

In [ ]:

print(diabetic\_df.glipizide.value\_counts())

print(diabetic\_df.glyburide.value\_counts())

print(diabetic\_df.tolbutamide.value\_counts())

print(diabetic\_df.pioglitazone.value\_counts())

print(diabetic\_df.rosiglitazone.value\_counts())

print(diabetic\_df.acarbose.value\_counts())

print(diabetic\_df.miglitol.value\_counts())

print(diabetic\_df.troglitazone.value\_counts())

print(diabetic\_df.tolazamide.value\_counts())

print(diabetic\_df.examide.value\_counts())

In [ ]:

drug\_cols = ['glipizide', 'glyburide', 'tolbutamide',

'pioglitazone', 'rosiglitazone', 'acarbose', 'miglitol', 'troglitazone',

'tolazamide', 'examide']

def explore\_drug(drugs):

for drug in drugs:

sns.countplot(x = drug, hue = "readmitted", data = diabetic\_df)

plt.show()

print(drug.upper())

print(diabetic\_df[f"{drug}"].value\_counts())

explore\_drug(drug\_cols)

In [ ]:

# convert diag\_ variables to Categorize based on https://www.hindawi.com/journals/bmri/2014/781670/tab2/

def convert\_diag(code):

try:

code = float(code)

except:

code = 0

# Circulatory

if code in range(390,460) or code == 785:

return("Circulatory")

# Respiratory

elif code in range(460,520) or code == 786:

return("Respiratory")

# Digestive

elif code in range(520,580) or code == 787:

return("Digestive")

# Diabetes

elif code >= 250 and code < 251:

return("Diabetes")

# Injury

elif code in range(800,1000):

return("Injury")

# Musculoskeletal

elif code in range(710,740):

return("Musculoskeletal")

# Genitourinary

elif code in range(580,630) or code == 788:

return("Genitourinary")

# Neoplasms

elif code in range(140,240):

return("Neoplasms")

elif code in range(780,783) or code == 784:

return("Neoplasms")

elif code in range(790,800):

return("Neoplasms")

elif code in range(240,250):

return("Neoplasms")

elif code in range(251,280):

return("Neoplasms")

elif code in range(680,710):

return("Neoplasms")

elif code in range(1,140):

return("Neoplasms")

elif code in range(290,320):

return("Neoplasms")

# Other

else:

return("Other")

In [ ]:

diabetic\_df["Diag1"] = diabetic\_df["diag\_1"].apply(convert\_diag)

diabetic\_df["Diag2"] = diabetic\_df["diag\_2"].apply(convert\_diag)

diabetic\_df["Diag3"] = diabetic\_df["diag\_3"].apply(convert\_diag)

In [ ]:

diabetic\_df = diabetic\_df.drop(columns=['diag\_1', 'diag\_2', 'diag\_3'])

In [ ]:

#balaji\_df = diabetic\_df[balaji\_cols]

df = diabetic\_df

In [ ]:

## removing variables "payer\_code" and "medical\_specialty" with large number of missing values and also not going to impact

## target "readmitted"

df = df.drop(['payer\_code', 'medical\_specialty'], axis = 1)

In [ ]:

## remove all variables with less than 6 percentage variance

#balaji\_df = balaji\_df.drop(['examide', 'glyburide', 'tolbutamide', 'pioglitazone', 'rosiglitazone',

# 'acarbose', 'miglitol', 'troglitazone', ], axis = 1)

In [ ]:

df.head()

Convert the categorical variables using onehot encoding

In [ ]:

df\_cleaned = pd.get\_dummies(df, columns=['discharge\_disposition\_id', 'admission\_source\_id', 'payer\_code\_impu',

'medical\_specialty\_impu', 'Diag1','Diag2','Diag3',

"metformin", "repaglinide", "nateglinide", "chlorpropamide",

"glimepiride", "acetohexamide", "glipizide", "glyburide",

"tolbutamide", "pioglitazone", "rosiglitazone", "acarbose",

"miglitol", "troglitazone", "tolazamide", "examide",

"citoglipton", "insulin", "glyburide-metformin",

"glipizide-metformin", "glimepiride-pioglitazone",

"metformin-rosiglitazone", "metformin-pioglitazone",

"change", "diabetesMed"],

prefix=["Discharged\_to", "Admission\_source","payer\_code", "medical\_specialty", "Diag1","Diag2",

"Diag3", "metformin", "repaglinide", "nateglinide", "chlorpropamide",

"glimepiride", "acetohexamide", "glipizide", "glyburide",

"tolbutamide", "pioglitazone", "rosiglitazone", "acarbose",

"miglitol", "troglitazone", "tolazamide","examide",

"citoglipton", "insulin", "glyburide-metformin",

"glipizide-metformin","glimepiride-pioglitazone",

"metformin-rosiglitazone", "metformin-pioglitazone",

"change", "diabetesMed"] )

In [ ]:

df\_cleaned.columns

In [ ]:

# Create X (Independent variables) and y(target) from the dataframe with all variables

X = df\_cleaned.drop(['readmitted'],axis=1)

ind\_columns = df\_cleaned.drop('readmitted',axis=1).columns

y = df\_cleaned['readmitted']

In [ ]:

pd.set\_option('display.max\_rows', 200)

print(X.dtypes)

We did normalize the attributes using StandardScaler() to scale them between 0 and 1 before running models.

In [ ]:

# Normalize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

We chose a stratified k-fold validation algorithm. In stratified k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples in which each fold contains roughly the same proportions of class labels. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.

The typical standard of 10 folds will be adequate for this dataset

In [ ]:

#Create Cross Validation Procedure

from sklearn.model\_selection import StratifiedKFold

cv = StratifiedKFold(n\_splits=10, random\_state=1234, shuffle=True)

#### Model 1: Logistic regression[¶](#Model-1:-Logistic-regression)

For our classification model we will initially track the accuracy, precision, and recall of the model as they are built. The precision for our classification model represents the ratio of the patients that the model correctly identifies as a positive readmission out of all the patients the model identifies as positive. The recall (also referred to as sensitivity), represents the ability of the model to correctly identify true positive readmission. The accuracy represents how well the model classifies correctly both positive and negative readmission.

In [ ]:

# Model Metrics

from sklearn.model\_selection import cross\_validate

from sklearn.model\_selection import cross\_val\_predict

from sklearn.metrics import classification\_report

def displayModel\_metrics(best\_model, grid\_model, features, target, cv):

metrics = cross\_validate(best\_model, features, y=target, cv=cv,

scoring=['accuracy','precision','recall'], return\_train\_score=True)

y\_predict = cross\_val\_predict(best\_model, features, target, cv=cv)

print('\nBest Accuracy with Grid Search : {:.3f}'.format(grid\_model.best\_score\_))

print('\nTraining data Metrics')

print('\n The average accuraccy : {:.3f}'.format(metrics['train\_accuracy'].mean()))

print(' The average precision : {:.3f}'.format(metrics['train\_precision'].mean()))

print(' The average recall : {:.3f}'.format(metrics['train\_recall'].mean()))

print('\nTest data Metrics')

print('\n The average accuracy : {:.3f}'.format(metrics['test\_accuracy'].mean()))

print(' The average precision : {:.3f}'.format(metrics['test\_precision'].mean()))

print(' The average recall : {:.3f}'.format(metrics['test\_recall'].mean()))

matrix = classification\_report(target, y\_predict, labels=[1,0])

print('\nClassification report\n')

print(matrix)

In [ ]:

# Reference https://github.com/jakemdrew/DataMiningNotebooks/blob/master/06.%20Classification.ipynb

# ROC curve plot

import seaborn as sns

from sklearn.preprocessing import label\_binarize

from sklearn import metrics as mt

def roc\_curve\_plot(model\_fit, features, target):

sns.set\_palette("dark")

yhat\_score = model\_fit.predict\_proba(features)

# Compute ROC curve for a subset of interesting classes

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in np.unique(target):

fpr[i], tpr[i], \_ = mt.roc\_curve(y, yhat\_score[:, i], pos\_label=i)

roc\_auc[i] = mt.auc(fpr[i], tpr[i])

for i in np.unique(target):

plt.plot(fpr[i], tpr[i], label= ('class %d (area = %0.2f)' % (i, roc\_auc[i])))

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.legend(loc="lower right")

plt.title('Receiver operating characteristic')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

GridSearch Parameters:

penalty: Used to specify the norm used in the penalization.

C: Inverse of regularization strength

max\_iter: Maximum number of iterations taken for the solvers to converge.

class\_weight: The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data .

solver:Algorithm to use in the optimization problem

In [ ]:

from sklearn.model\_selection import GridSearchCV

from sklearn.linear\_model import LogisticRegression

LR = LogisticRegression()

# define parameters

penalty\_LR = ['l1', 'l2', 'elasticnet', 'none']

C\_LR = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

max\_iter\_LR = [500, 1000]

class\_weight\_LR = ['balanced']

solver\_LR = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

# define grid search

param\_grid\_LR = dict(penalty=penalty\_LR, C=C\_LR, max\_iter=max\_iter\_LR, class\_weight=class\_weight\_LR, solver=solver\_LR)

grid\_search\_LR = GridSearchCV(estimator=LR, param\_grid=param\_grid\_LR, n\_jobs=7, cv=cv,

scoring='accuracy',error\_score=0)

In [ ]:

%%time

grid\_result\_LR = grid\_search\_LR.fit(X\_scaled, y)

# summarize results

print("Best: %f using %s" % (grid\_result\_LR.best\_score\_, grid\_result\_LR.best\_params\_))

means = grid\_result\_LR.cv\_results\_['mean\_test\_score']

stds = grid\_result\_LR.cv\_results\_['std\_test\_score']

params = grid\_result\_LR.cv\_results\_['params']

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param))

In [ ]:

#

results\_LR = pd.DataFrame(grid\_result\_LR.cv\_results\_['params'])

results\_LR['test\_score'] = grid\_result\_LR.cv\_results\_['mean\_test\_score']

results\_LR

In [ ]:

# The GridSearch algorithm determined the following optimal parameters

best\_Estimator\_LR =grid\_result\_LR.best\_estimator\_

best\_Estimator\_LR

In [ ]:

# Display model metrics

displayModel\_metrics(best\_Estimator\_LR, grid\_result\_LR, X\_scaled, y, cv)

#### Plot ROC[¶](#Plot-ROC)

The ROC curve shows the relationship between the model's ability to correctly vs incorrectly classify if readmmited or not with in 30 days. We could potentially make adjustments that could shift the ROC curve to have a higher true positive rate at the expense of increasing the false positive rate.

In [ ]:

# Plot ROC curve

roc\_curve\_plot(grid\_result\_LR, X\_scaled, y)

The ROC curve for the logistic regression classifier showed a mean area under the curve AUC of 0.66.

#### Learning Curves for Classifiers[¶](#Learning-Curves-for-Classifiers)

A learning curve shows the validation and training score of an estimator for varying numbers of training samples. It is a tool to find out how much we benefit from adding more training data and whether the estimator suffers more from a variance error or a bias error.

In [ ]:

from yellowbrick.model\_selection import LearningCurve

def Learning\_curve\_plot(model):

# Create the learning curve visualizer

sizes = np.linspace(0.3, 1.0, 10)

visualizer = LearningCurve(

model, cv=cv, scoring='accuracy', train\_sizes=sizes, n\_jobs=-1)

visualizer.fit(X\_scaled, y) # Fit the data to the visualizer

visualizer.show() # Finalize and render the figure

In [ ]:

# Learning curve for LR classifier

Learning\_curve\_plot(best\_Estimator\_LR)

With Logistic regression classifier, the training score and the validation score are converging with more data. With increasing training data, the validation score standard deviation is also changing a little which suggests logistic regression classifier will overfit with more data.

### Identify Important Features[¶](#Identify-Important-Features)

In [ ]:

feature\_importance\_df = pd.DataFrame(ind\_columns, columns=['features'])

feature\_importance\_df['feature\_coef'] = best\_Estimator\_LR.coef\_[0]

feature\_importance\_df.head()

feature\_importance\_df = feature\_importance\_df.sort\_values(by=['feature\_coef'])

The features leading to higher readmittance within 30 days are (those with positive co-efficients) and the features that reduce the chance of readmittance within 30 days (thos with negative co-efficients) are the most signficant features.

In [ ]:

feature\_importance\_df.loc[feature\_importance\_df['feature\_coef'] != 0].sort\_values(by='feature\_coef', ascending=True )

In [ ]:

important\_features = feature\_importance\_df.loc[feature\_importance\_df['feature\_coef'] != 0]

ax = sns.barplot(x ='feature\_coef', y = 'features', data=important\_features, orient= 'h')

ax.set\_title("Logistic Regression Feature Importance")

ax.set\_xlabel("Coefficients\n(z-score)")

ax.set\_ylabel("Features")

The bottom 10 features potentially show us those features that are most likely to decrease the chance of readmission. Having diabetic medication prescribed as "No" along with being primary and secondary diagnosis is Respiratory perhaps reduces the chance of readmission.

In [ ]:

print(important\_features)

In [ ]: