Case Study 2:
Predicting Hospital Readmittance of Patients with Diabetes

Kebur Fantahun, Eli Kravez, Halle Purdom

January 31, 2022

Introduction

Patients who have diabetes may be in and out of hospitals depending on different factors of their condition. Hospitals track a variety of factors related to the patient throughout their stay, which is what the data in this study is from (1). Driving the rate of readmission down could show an improvement in a hospital's overall effectiveness of treatment. Aside from the benefit of seeing patients do better, the refinement of the hospital's healthcare plans could also result in a lower cost for the hospital overall from less visits. Data from diabetes patients can be used to predict the readmittance of patients and to target the features that may be tuned to bring this rate down.

Methods

Data Exploration

The data is made up of information on patients with diabetes. There are 49 different features related to the patient and these will be used to predict the target variable, which is readmittance to the hospital. There was no duplicate data found, and all missing values in the dataset were marked with a question mark. All correlations between the numeric features were under 0.64 and therefore were not of concern in terms of multicollinearity. The examide and citoglipton columns all contained constant values and were dropped. The encounter ID column was unique to each row and was also dropped because it would not assist in predicting the target.

Missing Values and Imputation

There was a lot of missing data in the initial dataset, as can be seen in the table below:

	Type	Unique Values	Missing Percent (%)
Weight	Object	10	96.85
Medical_specialty	Object	73	49.08
Payer_code	Object	18	39.55
Race	Object	6	2.23
Diag_3	Object	790	1.39
Diag_2	Object	749	0.35
Diag_1	Object	717	0.02

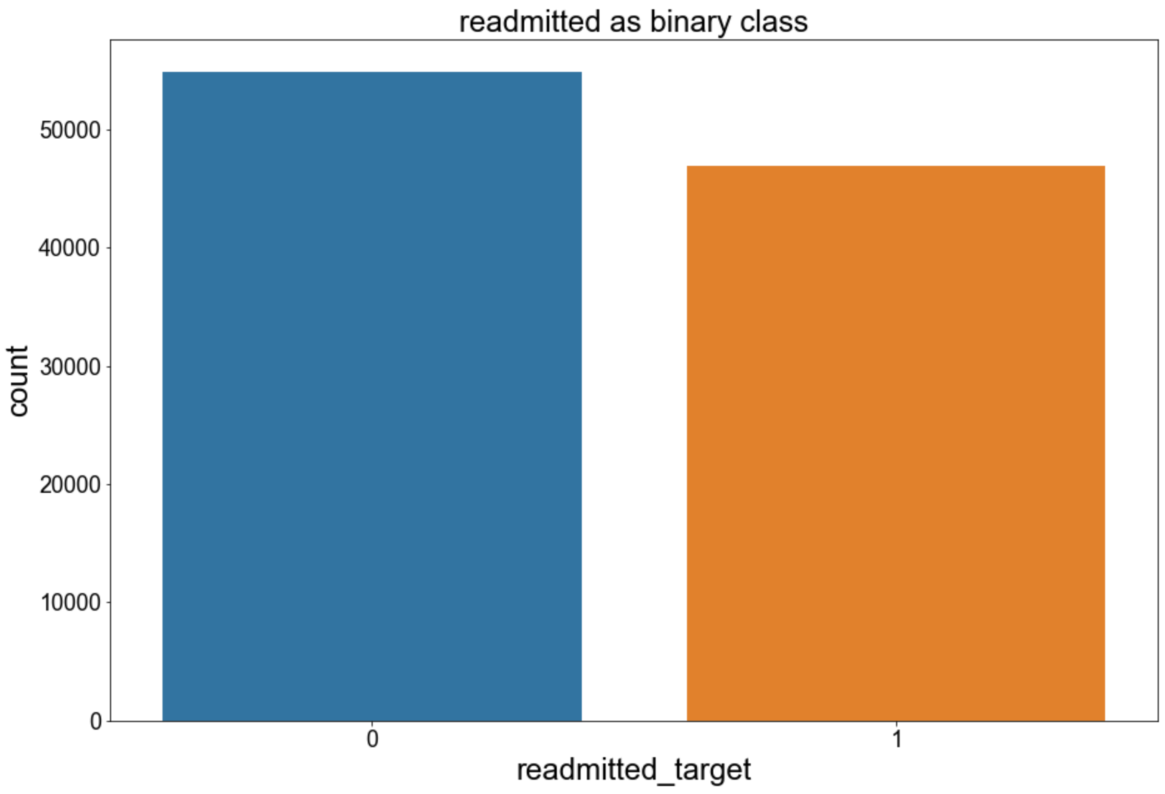
The weight column is 97% missing, because of this the column will be dropped. Different considerations for filling in the missing values of weight were considered, such as using the mode of the weight from the age group feature. Ultimately the column was dropped since only 3% of the data existed making the column of little use.

Medical specialty and payer code were 49% and 40% missing respectively. Methods of imputation from mode were considered as well as more complex imputation methods like multivariate logistic regression or knn imputer. Because these more complex methods take time to implement, as well as computation time and resources to run, they were not used. Specifically with knn, the categorical values would have to be mapped to become numerical which poses a large time commitment. Instead the missing values were added to a new category of their own. This has the benefit of maintaining the distribution of the original categories, as mode greatly disrupts the original distribution.

Race was 2% missing, and because most of the data from this feature was present, the method of imputation used was mode. Diag_1, diag_2, and diag_3 all represent diagnoses codes from the patients. They were all less than 1.5% missing and the mode was used to impute these codes. While the gender column did not have any missing values, it did have 3 'invalid' entries. These were kept in the dataset as a separate category.

Transforming Features

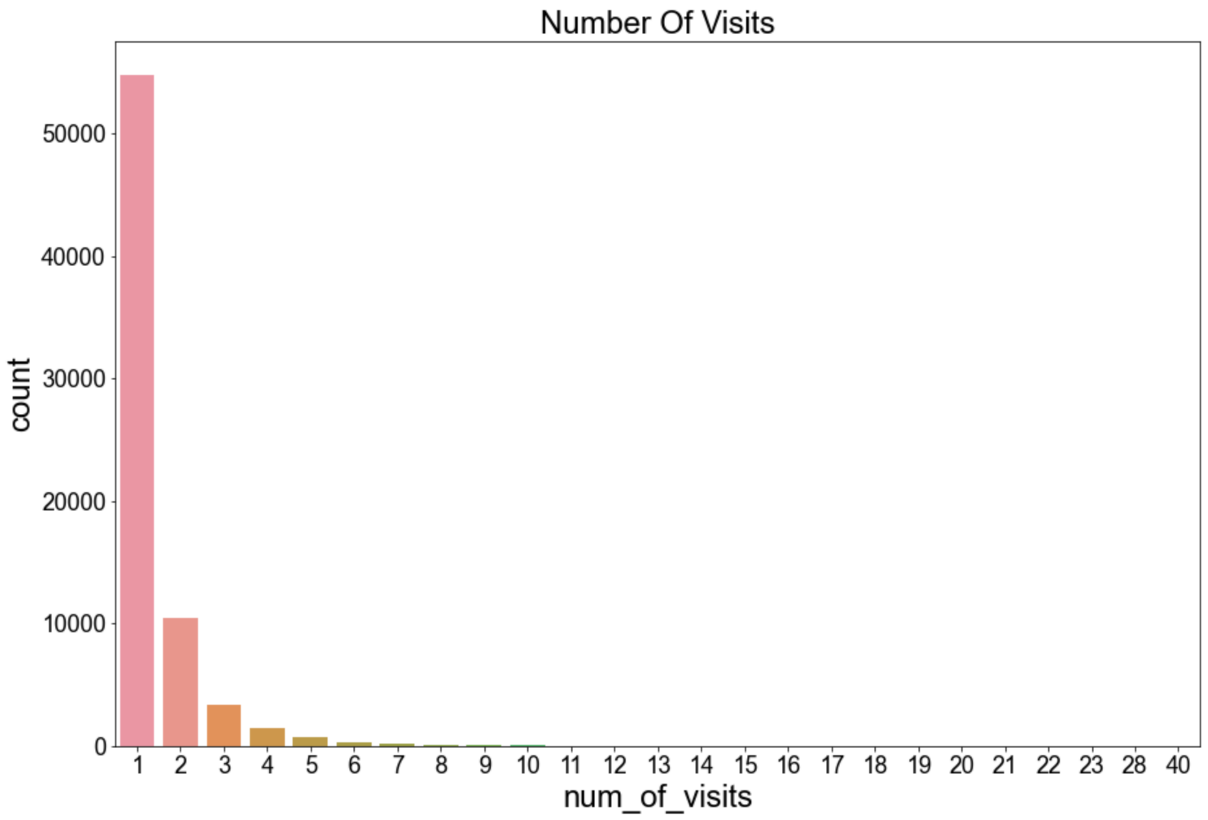
The target variable readmission contained values '>30', '<30', and 'no' representing if a patient was readmitted in over 30 days, in under 30 days, and if they were not readmitted. This column was transformed into binary values where 0 represents readmission and 1 represents no readmission. This is necessary with the logistic regression models used since they are binary predictors. After transforming, the two categories have a similar number of records as seen in the below graph. This makes accuracy a good metric to evaluate the logistic regression models, which would not be the case if the data were very skewed.



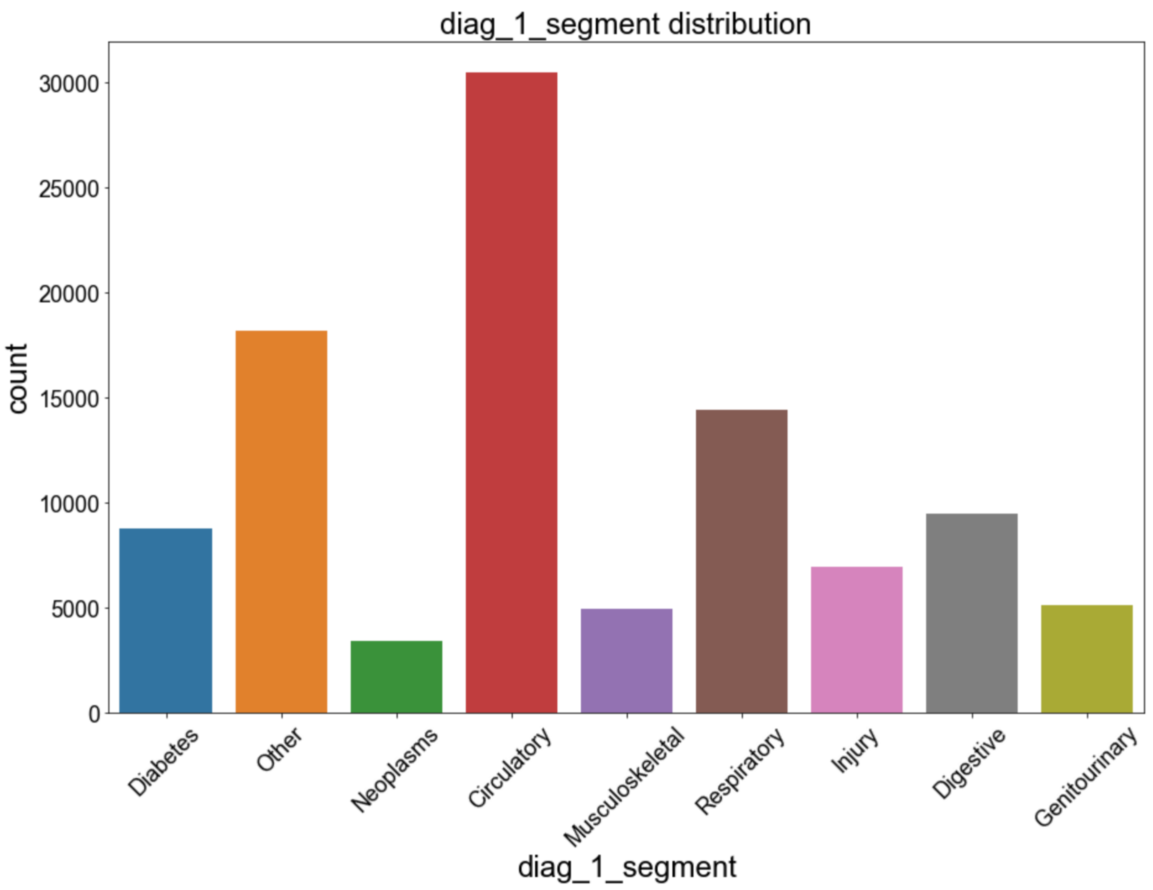
The age values were originally categorical entries that spanned 10 year ranges. In order to create an ordinal variable out of this, the ranges were dropped and the first value of the range was used (ex. [10-20] to 10). This transforms the previous categorical ranges into ordinal numeric values.

Three of the features including admission type ID, discharge disposition ID, and admission source ID were coded as integers but represented categories that were identified in the second data file. These features were transformed into categorical variables.

One of the main assumptions of logistic regression models assumes all observations are independent of each other. The patient_nbr column, representing the patient ID number, showed that some of the observations captured returning patients. This violates the independence assumption since multiple observations sometimes represent the same patient. To resolve this issue, this column was transformed into a count of the number of visits for a particular patient, and the original column was then dropped. Taking only the first visit of each patient was considered as another potential solution, however this was impossible with the data given since there was no date of visit recorded. As you can see in the graph below most of the patients visited only one time, but we can see the distribution is very right skewed. There is even an example of a patient who returned to the hospital 40 times.



The three diagnoses columns all represented codes and were therefore categorical. Because there were hundreds of codes in these columns, one-hot encoding is too computationally expensive to use in this case. These diagnoses codes could be categorized into 9 overarching groups (2). The variables were transformed down to these 9 categories, and their original columns were dropped. The below graph shows the resultant distribution of the diagnoses 1 column after transformation. The diagnoses 2 and 3 columns were about the same.



After all the data preparation, there were 11 numeric features and 37 categorical features. All the categorical features were one-hot encoded in order to be used in the logistic regressions. The numeric features were then scaled with StandardScaler(). The dummy variables from the one-hot encoding were not scaled. This is because they are sparse data with binary values, so the StandardScaler() tends to scale these variables much larger than the intended range around 0.

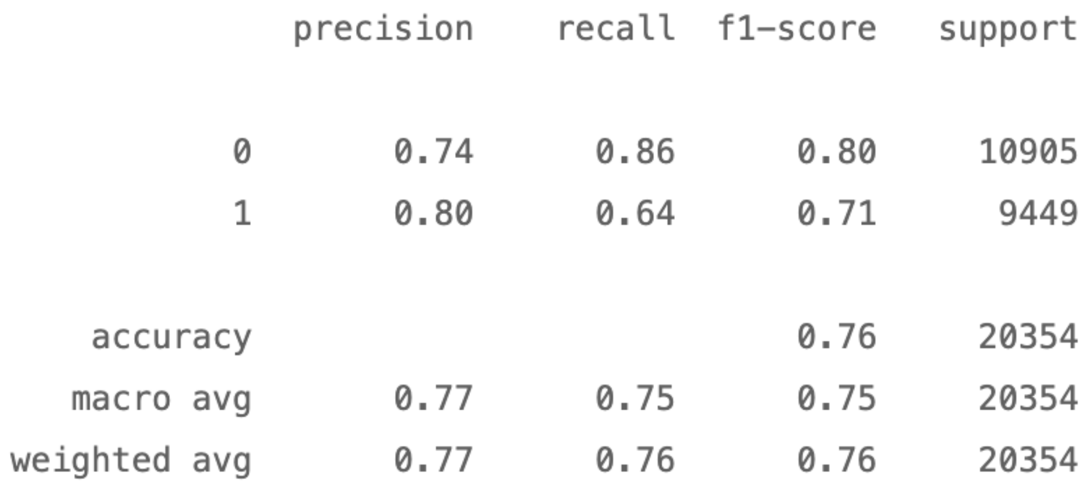
Logistic Regression Models

The data was split into a training set with 80% of the data and a testing set with the remaining 20% of the data. The training data is used for tuning the parameters with a shuffle split KFold cross validation using 5 splits. After looking into the 5 accuracies of the basic logistic regression model, there was not a significant difference between the values. This showed that the split of the data was relatively equal and would be sufficient for the L1 and L2 models.

Logistic regression models with L1 and L2 penalties were created. Their parameters were tuned on the training data using RandomizedSearchCV. After the best parameters for each model were found, the testing data was used to evaluate the final models.

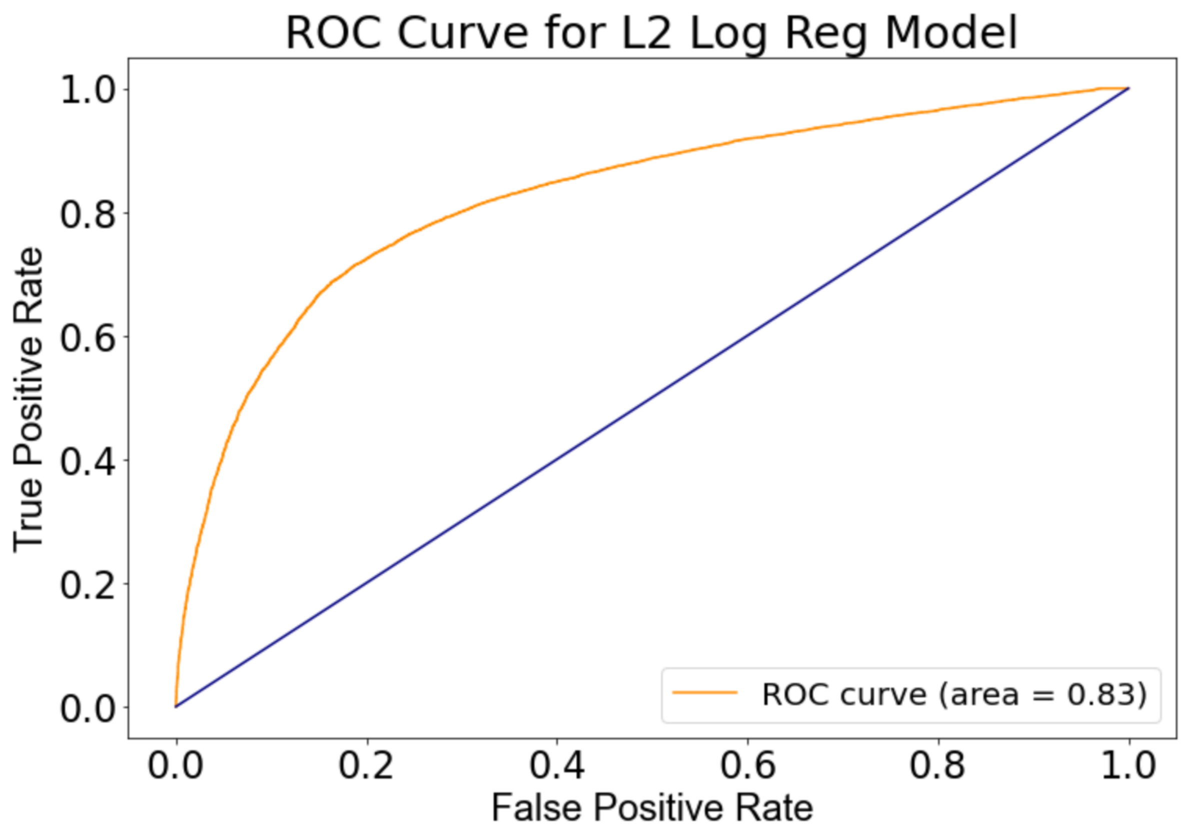
Results

L1 and L2 from a prediction perspective perform very similarly. Because of this only the L2 results are discussed below. The precision, recall, and accuracy are seen in the below table.



```
              precision    recall  f1-score   support

           0       0.74      0.86      0.80     10905
           1       0.80      0.64      0.71      9449

    accuracy                           0.76     20354
   macro avg       0.77      0.75      0.75     20354
weighted avg       0.77      0.76      0.76     20354
```

The area under the curve in the ROC graph is 0.83. This reasonable performance for a logistic regression model. Using more robust models like random forest would probably yield higher performance.



Important Features

Feature	**Coefficient**
Discharge disposition ID - expired	-7.83
Number of visits	2.68
Discharge disposition ID - hospice/medical facility	-2.26
Admission type ID - trauma center	-1.88
Medical specialty - Allergy and Immunology	1.79
Discharge disposition ID - Discharged/transferred within this institution to Medicare approved swing bed	1.51
Discharge disposition ID - Expired at home. Medicaid only, hospice.	-1.40

acarbose_Down	-1.36
Discharge disposition ID - hospice home	-1.34
Admission source 20 - not mapped	1.31

The most important feature determined by the model was discharge disposition ID - expired. More research into this feature indicated it meant the patient had passed away. The large negative coefficient makes sense because they would be unable to be readmitted. For future analysis, removing patients who passed away from the dataset could be considered. This would also include other discharge disposition ID categories such as ‘expired at home’. The second most important feature from the L2 model was number of visits. This also makes sense as more previous readmissions would indicate a patient is more likely to be readmitted in the future.

Ethics

One point to consider in this study is the ethics of studying and collecting data on the patients’ race. Some medical conditions are more prevalent among different races, which could be important in this study since the data is on diabetic patients. In looking at the most important features of the models, however, race was not present. In this particular case, the models indicate the data collected on the race of the patient was not highly important in predicting their readmission rates. From these results, one could make the decision not to continue collecting this data from the patients as it may not be particularly useful in trying to prevent readmission. In future work, consulting a medical professional would prove useful as they have domain knowledge on the data.

Conclusion

In conclusion, the L1 and L2 models performed very similarly. The final L2 model was able to predict readmittance of diabetic patients back into the hospital at a 76% accuracy, which is reasonably well. The model indicated the two most important features in determining readmittance were death and number of visits for a patient. Future work into this analysis would include exploring other more robust models to improve model performance. In addition, patients who have passed away would be removed from the dataset to provide a more useful model for patient insights into readmittance.

References

1. https://www.hindawi.com/journals/bmri/2014/781670/
2. https://www.hindawi.com/journals/bmri/2014/781670/tab2/

Case Study 2: Predicting Hospital Readmittance of Patients with Diabetes

Group: Kebur Fantahun, Eli Kravez, Halle Purdom

Your case study is to build a classifier using logistic regression to predict hospital readmittance. There is missing data that must be imputed. Once again, discuss variable importances as part of your submission.

In [ ]:

```
# Import necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns


import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import cross_val_score, StratifiedKFold, KFold, cro
from sklearn.metrics import precision_score, recall_score, confusion_matrix, acc
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.metrics import plot_confusion_matrix

import random
```

In [ ]:

```
# Visualization function for the columns
def visualize_counts(col_name, df, f_heigh = 15, f_width=10, title= "", f_rotati
    plt.figure(figsize=(f_heigh, f_width))
    axis_font = {'fontname':'Arial', 'size':'24'}
    ax = sns.countplot(x=col_name, data=df)
    plt.xlabel(col_name, **axis_font)
    plt.ylabel("count", **axis_font)
    plt.title(title, **axis_font)
    plt.xticks(rotation = f_rotation)

    for label in (ax.get_xticklabels() + ax.get_yticklabels()):
        label.set_fontname('Arial')
        label.set_fontsize(18)

    plt.show()
```

Data Preparation and Exploration

In [ ]:

```
# Reading in data, replacing all question marks with na values
data  = pd.read_csv("diabetic_data.csv", na_values=["?"])
```

```
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/IPython/core/intera
ctiveshell.py:3165: DtypeWarning: Columns (10) have mixed types.Specify dtype op
tion on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

In [ ]:
```
data.head()
```

Out[ ]:

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	discharg
0	2278392	8222157	Caucasian	Female	[0-10)	NaN	6	
1	149190	55629189	Caucasian	Female	[10-20)	NaN	1	
2	64410	86047875	AfricanAmerican	Female	[20-30)	NaN	1	
3	500364	82442376	Caucasian	Male	[30-40)	NaN	1	
4	16680	42519267	Caucasian	Male	[40-50)	NaN	1	

5 rows × 50 columns

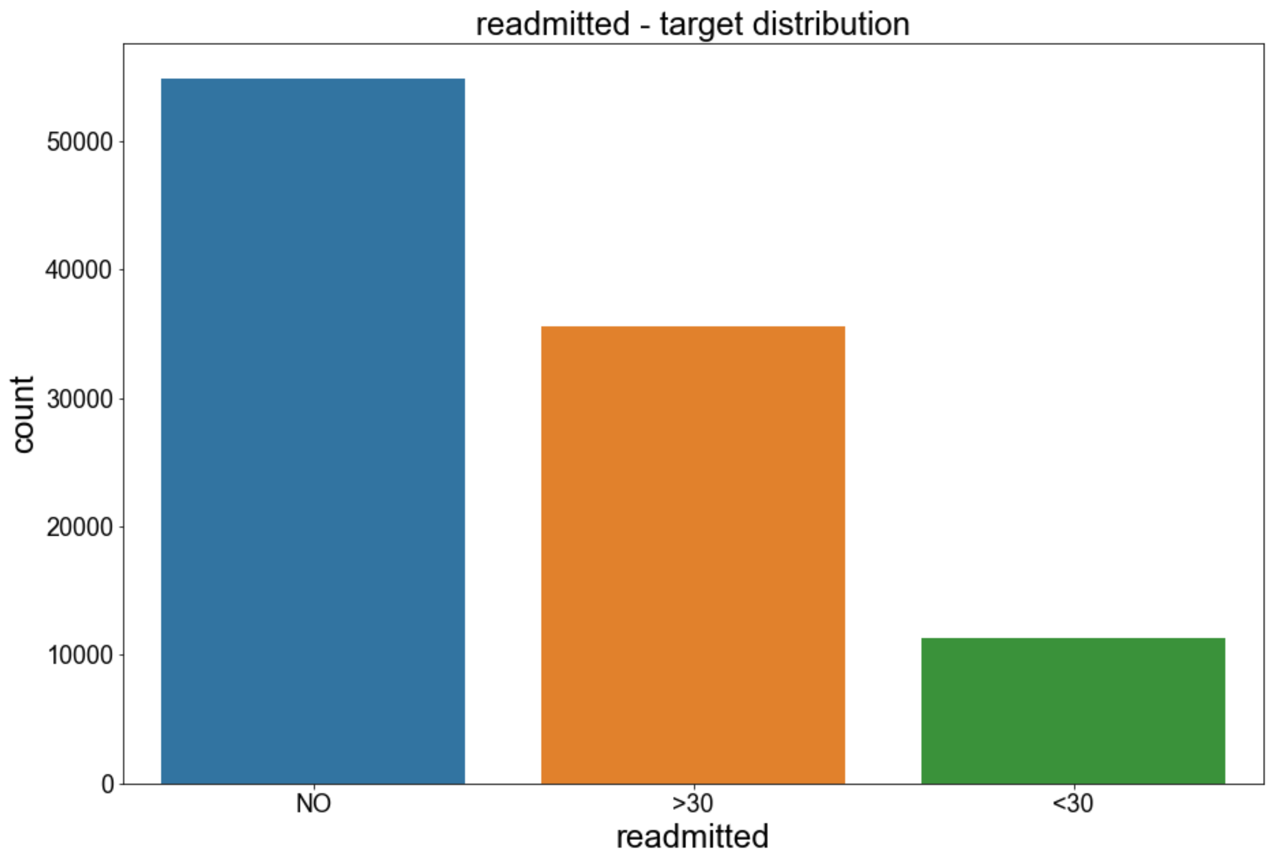
In [ ]:
```
shape = data.shape
print(f'data shape is : {shape}')
```

```
data shape is : (101766, 50)
```

In [ ]:
```
# Check for duplicates - no duplicatess found
data.duplicated().value_counts()
```

Out[ ]:
```
False    101766
dtype: int64
```

In [ ]:
```
# Visualize target variable: Readmitted -  switch >30 and <30 to 1, and No to 1
visualize_counts('readmitted', data, title='readmitted - target distribution')
```



In [ ]:

```
# Basic column info:
# Missing data - NA: (we will need to check other missing options)
#   weight - 96% data missing
#   payer_code - 39%
#   medical_specialty - 49%
#   race - 2.2% is missing
#   diag_3 - 1.3%
#   diag_2/diag_3 < 1%
# As well we can see that citoglipton, examide have on 1 constant value so we ca
# encounter_id - all the values are different - seems like ID coulmn which can b
# patient_nbr - paitent ID, there are some repeated values (returning patients)

data_types = data.dtypes
missing_value_stats = ((data.isnull().sum()/len(data)*100)) # .sort_values(ascen
unique_values = data.apply(lambda column: column.unique().shape[0])

basic_stats = pd.concat([data_types, unique_values, missing_value_stats], axis =
basic_stats.columns = ['type', 'unique values', 'missing percent']
basic_stats = basic_stats.sort_values('missing percent', ascending = False)
print(basic_stats)
```

```
                          type  unique values  missing percent
weight                  object             10        96.858479
medical_specialty       object             73        49.082208
payer_code              object             18        39.557416
race                    object              6         2.233555
diag_3                  object            790         1.398306
diag_2                  object            749         0.351787
diag_1                  object            717         0.020636
nateglinide             object              4         0.000000
num_lab_procedures       int64            118         0.000000
```

```
num_medications             int64          75         0.000000
num_procedures              int64           7         0.000000
number_diagnoses            int64          16         0.000000
number_emergency            int64          33         0.000000
number_inpatient            int64          21         0.000000
number_outpatient           int64          39         0.000000
patient_nbr                 int64       71518         0.000000
acetohexamide              object           2         0.000000
pioglitazone               object           4         0.000000
metformin-rosiglitazone    object           2         0.000000
readmitted                 object           3         0.000000
repaglinide                object           4         0.000000
rosiglitazone              object           4         0.000000
time_in_hospital            int64          14         0.000000
tolazamide                 object           3         0.000000
tolbutamide                object           2         0.000000
troglitazone               object           2         0.000000
miglitol                   object           4         0.000000
metformin                  object           4         0.000000
metformin-pioglitazone     object           2         0.000000
admission_source_id         int64          17         0.000000
admission_type_id           int64           8         0.000000
age                        object          10         0.000000
change                     object           2         0.000000
chlorpropamide             object           4         0.000000
citoglipton                object           1         0.000000
diabetesMed                object           2         0.000000
discharge_disposition_id    int64          26         0.000000
encounter_id                int64      101766         0.000000
examide                    object           1         0.000000
gender                     object           3         0.000000
glimepiride                object           4         0.000000
glimepiride-pioglitazone   object           2         0.000000
glipizide                  object           4         0.000000
glipizide-metformin        object           2         0.000000
glyburide                  object           4         0.000000
glyburide-metformin        object           4         0.000000
insulin                    object           4         0.000000
max_glu_serum              object           4         0.000000
acarbose                   object           4         0.000000
A1Cresult                  object           4         0.000000
```

In [ ]:

```
# Drop columns which have constant values and therefore do not contribute for pr
columns_to_drop = data.columns[data.nunique() <= 1]
print(columns_to_drop)
data_prcessed_df = data.drop(columns_to_drop, axis=1)

# Dropping ID coulmn as has only unique values per row
data_prcessed_df = data.drop("encounter_id", axis=1)
```

```
Index(['examide', 'citoglipton'], dtype='object')
```

Patient_nbr Column

In [ ]:

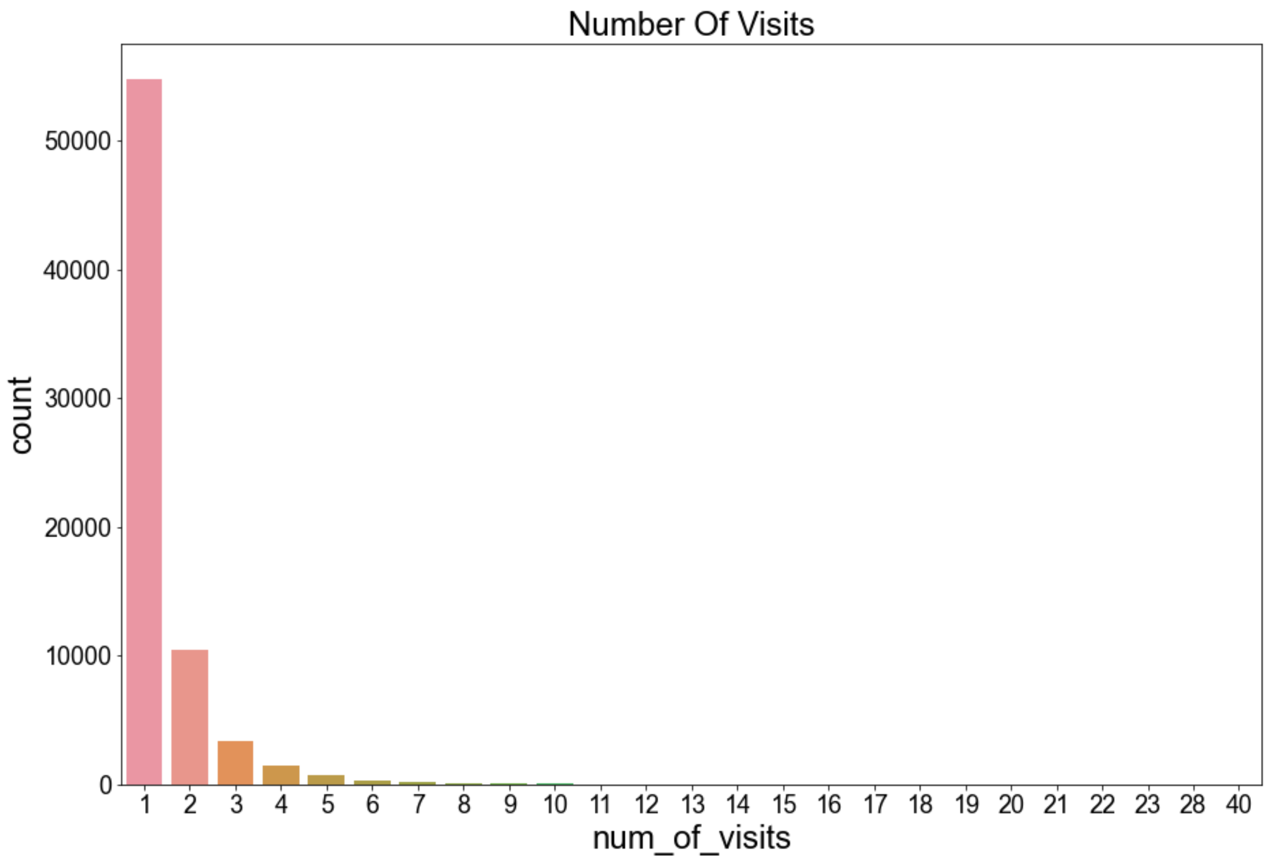
```
# Patient_nbr - some patients are returning to the hospital multiple times
# Create new column of how many times a specific patient returns to hospital 'nu
patient_nbr_db = data_prcessed_df.groupby(['patient_nbr']).size().reset_index(na
patient_nbr_db.head()
```

Out[ ]:

	patient_nbr	num_of_visits
0	135	2
1	378	1
2	729	1
3	774	1
4	927	1

In [ ]:

```python
# Visualize number of times patients visited hospital
visualize_counts('num_of_visits', patient_nbr_db, title='Number Of Visits')
```



In [ ]:

```python
data_prcessed_df = pd.merge(data_prcessed_df,patient_nbr_db,on='patient_nbr')
data_prcessed_df = data_prcessed_df.drop('patient_nbr', axis=1)
```

In [ ]:

```python
data_prcessed_df.head()
```

Out[ ]:

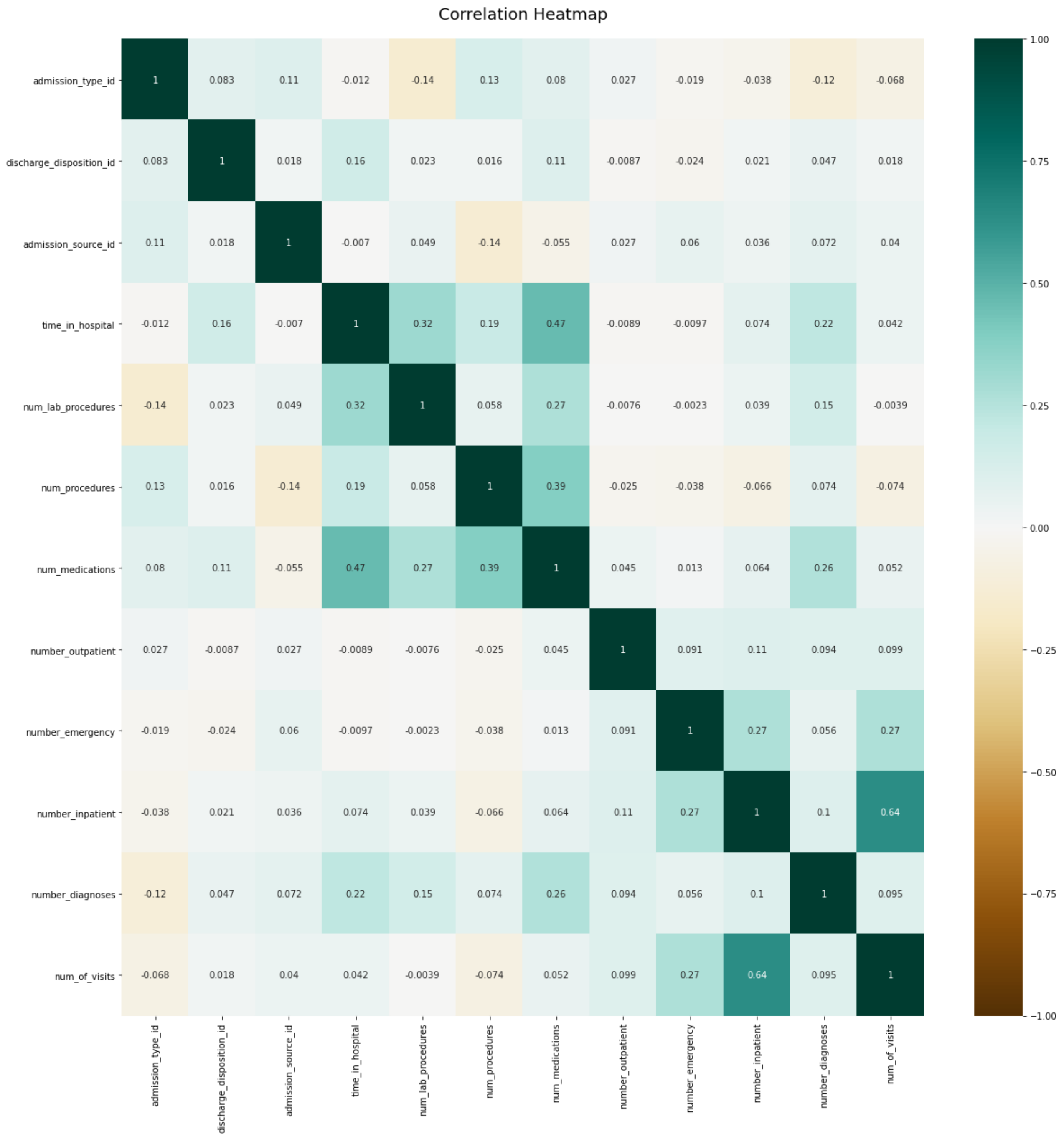
	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_
0	Caucasian	Female	[0-10)	NaN	6	25	
1	Caucasian	Female	[10-20)	NaN	1	1	

	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_
2	AfricanAmerican	Female	[20-30)	NaN	1	1	
3	Caucasian	Male	[30-40)	NaN	1	1	
4	Caucasian	Male	[40-50)	NaN	1	1	

5 rows × 49 columns

In [ ]:

```
# Looking at correlations between numeric columns
plt.figure(figsize=(20, 20))
heatmap = sns.heatmap(data_prcessed_df.corr(), vmin=-1, vmax=1, annot=True, cmap
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=22)
plt.show()
```



Missing Values Exploration and Imputation

In [ ]:
```
# Function to print missing values in column
def number_of_missing(col_name):
   missing_values = data_prcessed_df[col_name].isna().sum()
   print(f'{col_name} missing values: {missing_values}')
```

In [ ]:
```
# Race - Pattern of missing data = missing completely at random
# Race missing values - replace with mode
number_of_missing('race')
```

```
race missing values: 2273
```

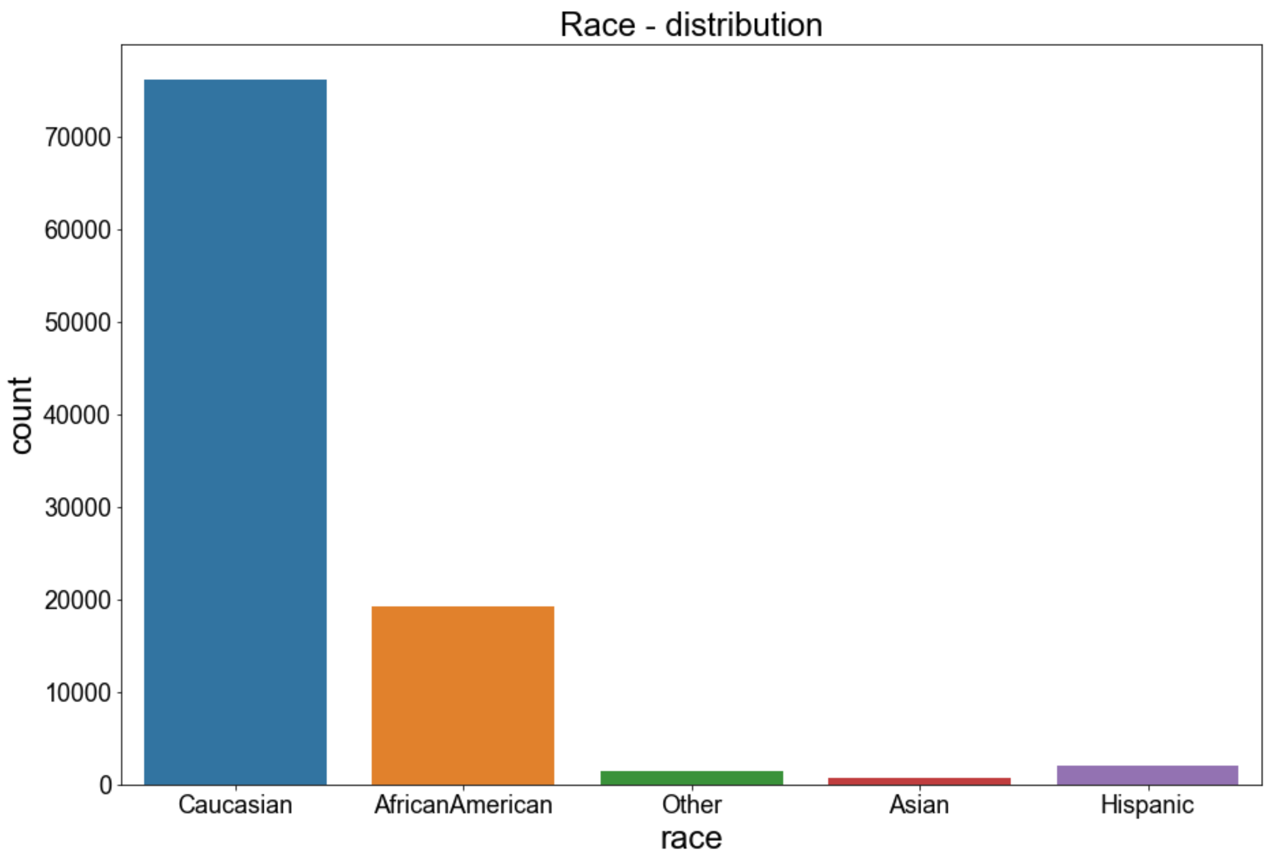
In [ ]:

```
race_segments = data_prcessed_df['race'].value_counts()
print(race_segments)
```

```
Caucasian          76099
AfricanAmerican    19210
Hispanic            2037
Other               1506
Asian                641
Name: race, dtype: int64
```

In [ ]:

```
# Visualize race distribution
visualize_counts('race', data_prcessed_df, title ='Race - distribution')
```



In [ ]:

```
# Replace missing values for 'race' with mode
data_prcessed_df['race'].fillna(data_prcessed_df['race'].mode()[0], inplace=True

# Make sure no more missing values for race:
number_of_missing('race')
```

```
race missing values: 0
```

In [ ]:

```
# Gender - 3 invalid values - we will not change this
data_prcessed_df['gender'].value_counts()
```

Out[ ]:

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

In [ ]:

```
# Look at Age ranges
```

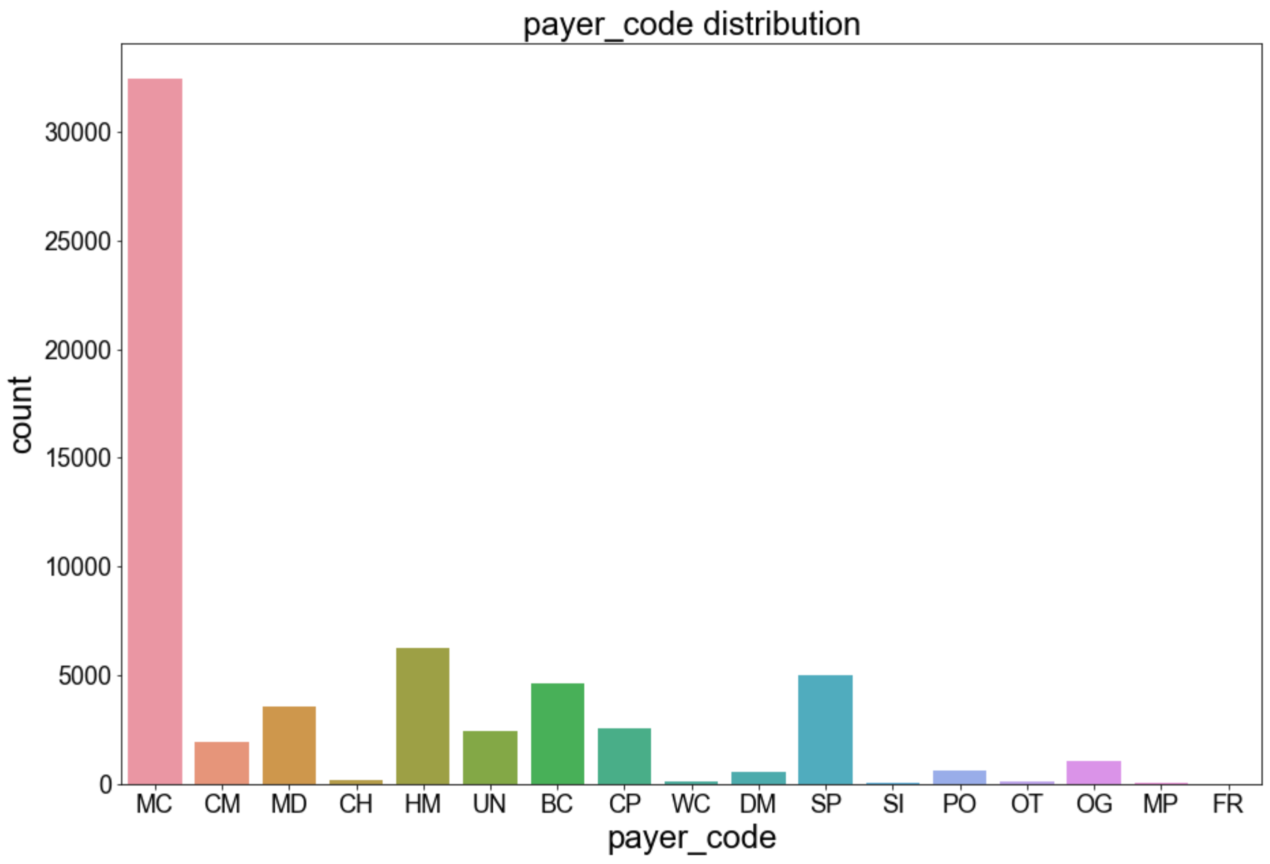
```
data_prcessed_df['age'].value_counts()
```

Out[ ]:
```
[70-80)      26068
[60-70)      22483
[50-60)      17256
[80-90)      17197
[40-50)       9685
[30-40)       3775
[90-100)      2793
[20-30)       1657
[10-20)        691
[0-10)         161
Name: age, dtype: int64
```

In [ ]:
```
# Weight - has 95% of missing values. Based on the class discussion - drop this
data_prcessed_df = data_prcessed_df.drop("weight", axis=1)
```

Payer_code Column

In [ ]:
```
visualize_counts('payer_code', data_prcessed_df, title = 'payer_code distributio
```



In [ ]:
```
# Significant amout of the missing values - replace with mode
number_of_missing('payer_code')
```

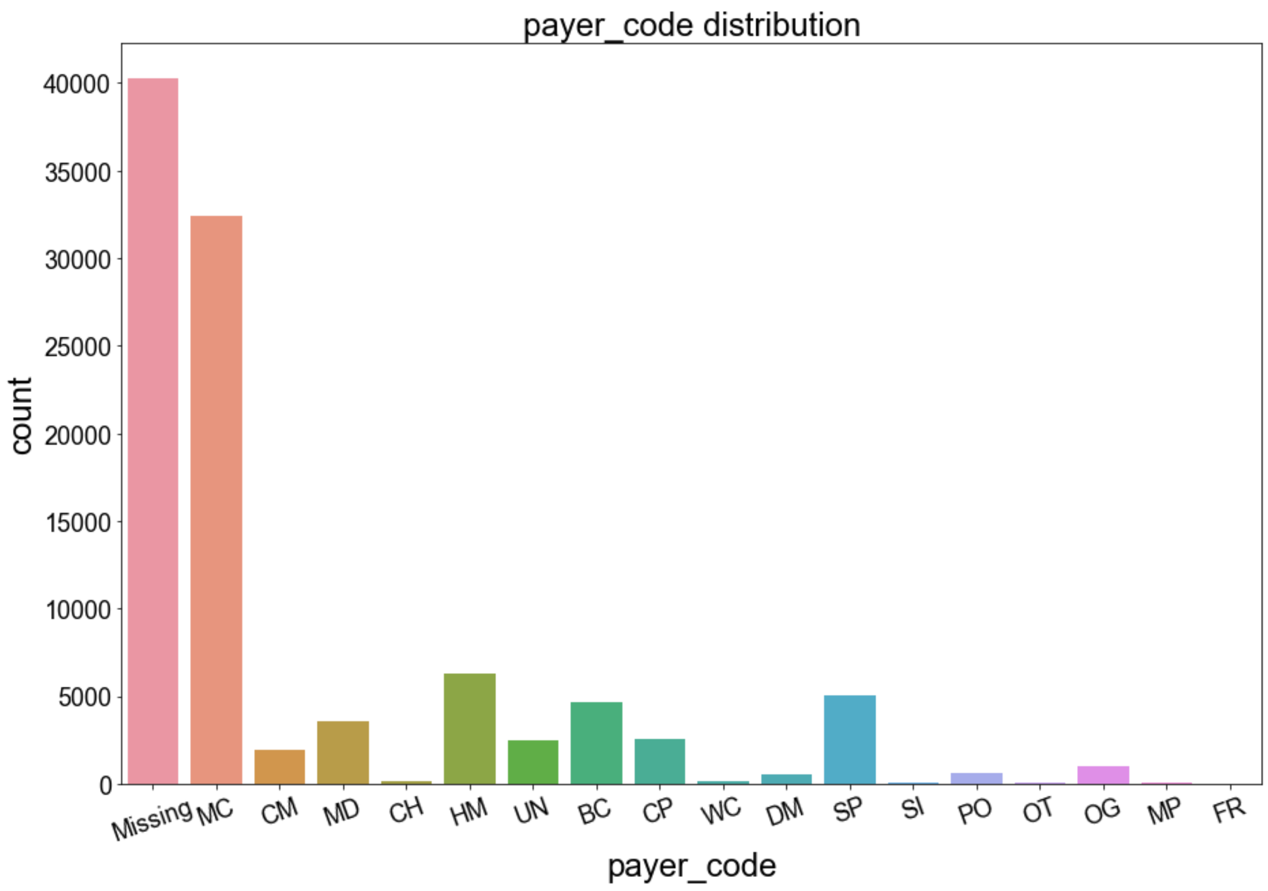
```
payer_code missing values: 40256
```

In [ ]:
```
# Replace missing values for 'payer_code' with mode
data_prcessed_df['payer_code'].fillna('Missing', inplace=True)
```

```
# Make sure no more missing values for payer_code:
number_of_missing('payer_code')
```

```
payer_code missing values: 0
```

In [ ]:
```
# Visualize distribution after adding new missing value feature to payer_code

visualize_counts('payer_code', data_prcessed_df,  title = 'payer_code distributi
```

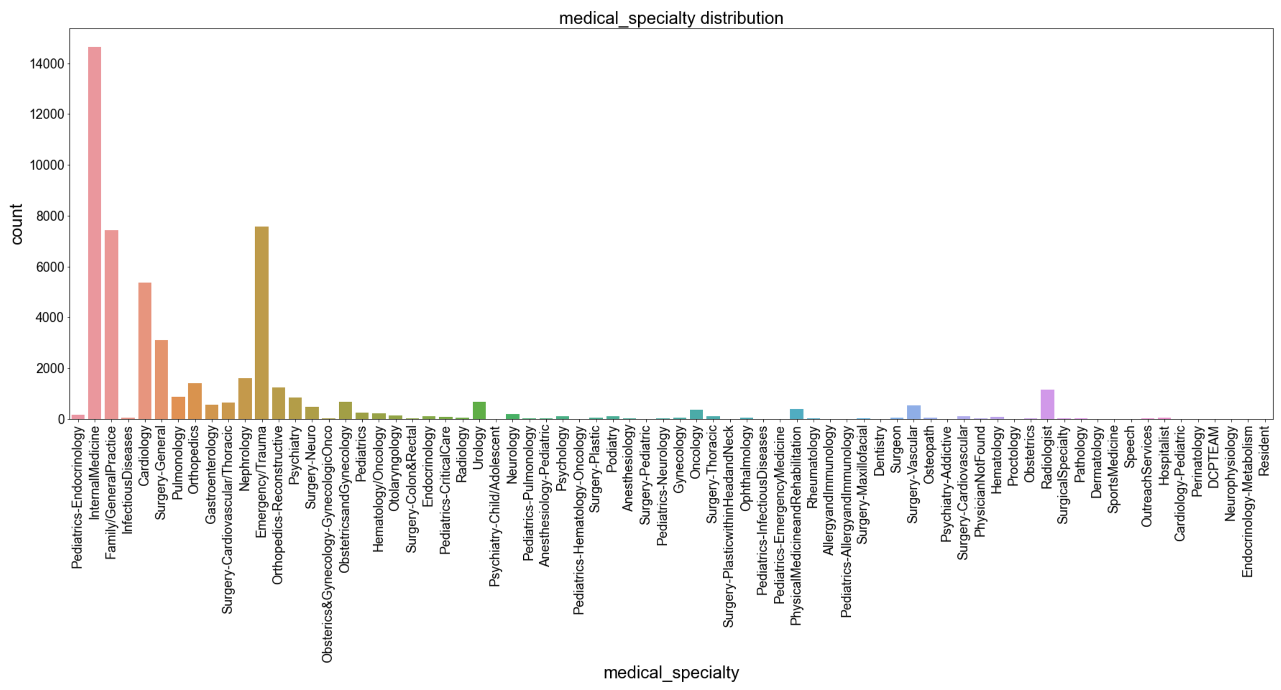


Medical_specialty Column

In [ ]:
```
number_of_missing('medical_specialty')
```

```
medical_specialty missing values: 49949
```

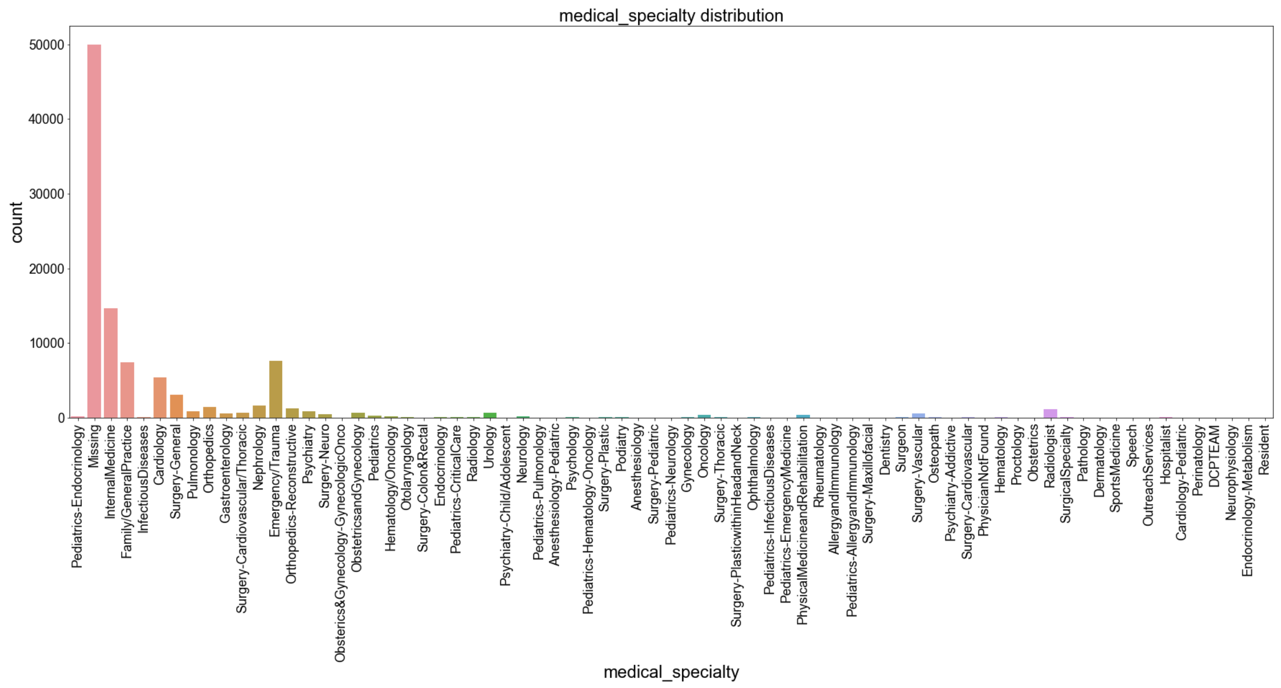
In [ ]:
```
visualize_counts('medical_specialty', data_prcessed_df, title = "medical_special
```



In [ ]:
```python
# Replace missing values for 'medical_specialty' with mode
data_prcessed_df['medical_specialty'].fillna('Missing', inplace=True)

# Make sure no more missing values for medical_specialty:
number_of_missing('medical_specialty')
```

```
medical_specialty missing values: 0
```

In [ ]:
```python
# Look at distribution after making missing values its own feature
visualize_counts('medical_specialty', data_prcessed_df, title = "medical_special
```



Diag_1, diag_2 diag_3 Columns

In [ ]:

```
# We have some percentage of the missing valus for the columns
number_of_missing('diag_1')
number_of_missing('diag_2')
number_of_missing('diag_3')
data_prcessed_df['diag_1'].fillna(data_prcessed_df['diag_1'].mode()[0], inplace=
data_prcessed_df['diag_2'].fillna(data_prcessed_df['diag_2'].mode()[0], inplace=
data_prcessed_df['diag_3'].fillna(data_prcessed_df['diag_3'].mode()[0], inplace=
print('evaluate na fix:')
number_of_missing('diag_1')
number_of_missing('diag_2')
number_of_missing('diag_3')
```

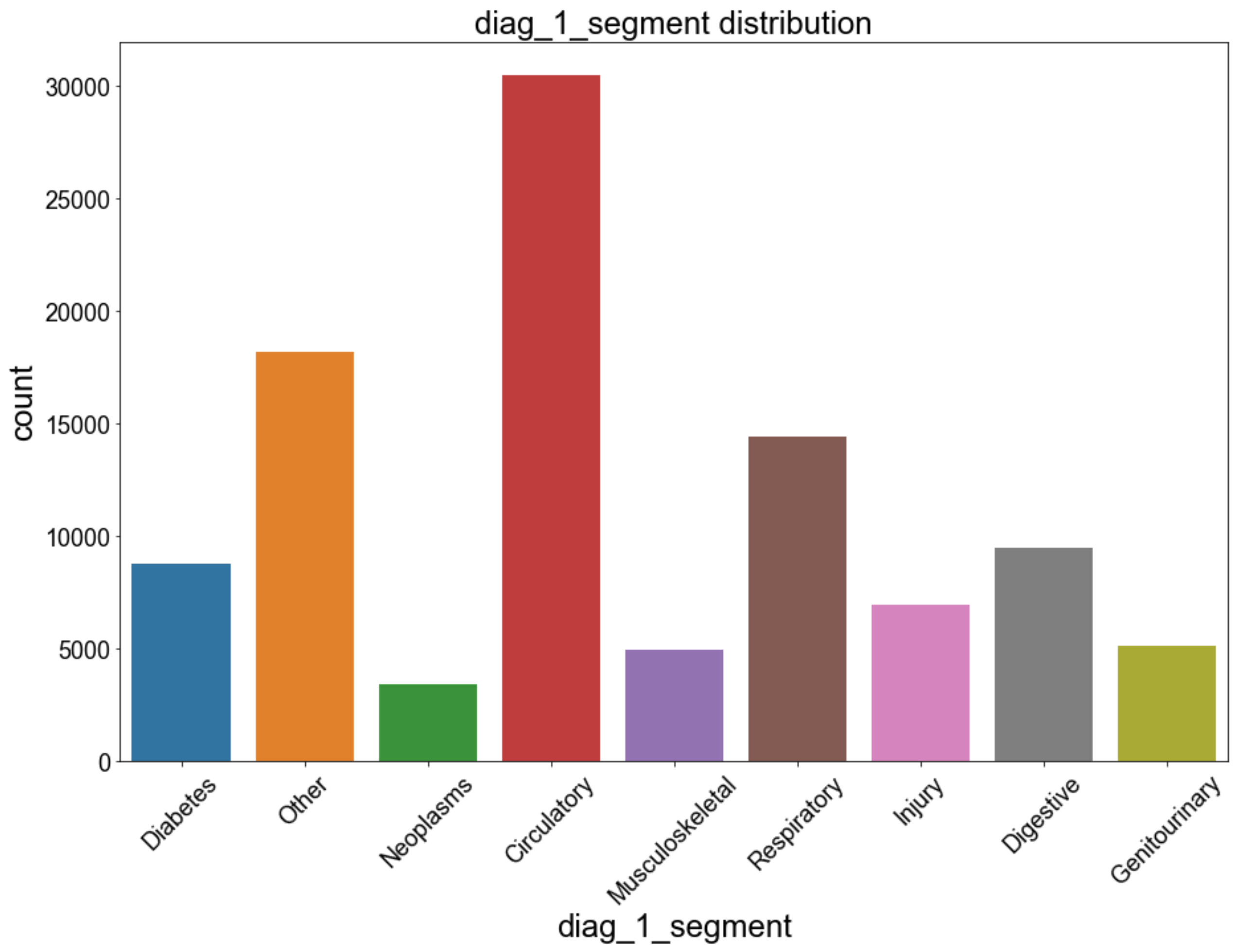
```
diag_1 missing values: 21
diag_2 missing values: 358
diag_3 missing values: 1423
evaluate na fix:
diag_1 missing values: 0
diag_2 missing values: 0
diag_3 missing values: 0
```

In [ ]:

```
# Function to put codes for diagnoses into categories based on https://www.hinda
for diag_col in ['diag_1','diag_2','diag_3']:
    new_col_name = diag_col+ '_segment'
    data_prcessed_df[new_col_name]= np.nan

    data_prcessed_df.loc[data_prcessed_df[diag_col].str.contains('V'), [diag_col
    data_prcessed_df.loc[data_prcessed_df[diag_col].str.contains('E'), [diag_col
    data_prcessed_df.loc[data_prcessed_df[diag_col].str.contains('250'), [diag_c

    data_prcessed_df[diag_col] = data_prcessed_df[diag_col].astype(float)

    # Update base on : https://www.hindawi.com/journals/bmri/2014/781670/tab2/

    data_prcessed_df.loc[(data_prcessed_df[diag_col]>=390) & (data_prcessed_df[d
    data_prcessed_df.loc[(data_prcessed_df[diag_col]>=460) & (data_prcessed_df[d
    data_prcessed_df.loc[(data_prcessed_df[diag_col]>=520) & (data_prcessed_df[d
    data_prcessed_df.loc[(data_prcessed_df[diag_col]==250),[new_col_name]]='Diab
    data_prcessed_df.loc[(data_prcessed_df[diag_col]>=800) & (data_prcessed_df[d
    data_prcessed_df.loc[(data_prcessed_df[diag_col]>=710) & (data_prcessed_df[d
    data_prcessed_df.loc[(data_prcessed_df[diag_col]>=580) & (data_prcessed_df[d
    data_prcessed_df.loc[(data_prcessed_df[diag_col]>=140) & (data_prcessed_df[d

    data_prcessed_df[new_col_name].fillna('Other', inplace=True)

data_prcessed_df.drop(['diag_1','diag_2','diag_3'], axis=1, inplace=True)
```

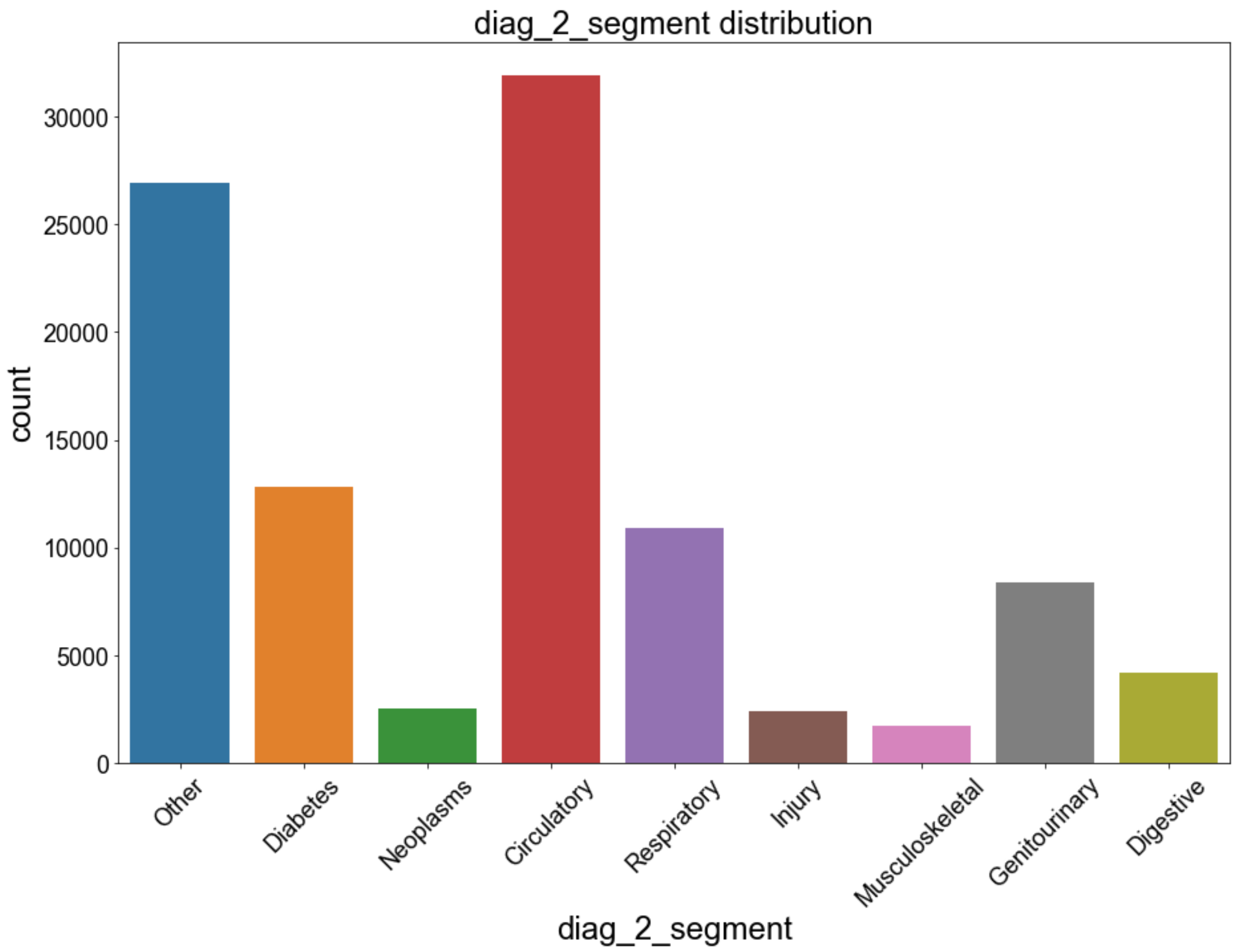
In [ ]:

```
# Visualize categorical transformation of diag_1 column
visualize_counts('diag_1_segment', data_prcessed_df, title='diag_1_segment distr
```



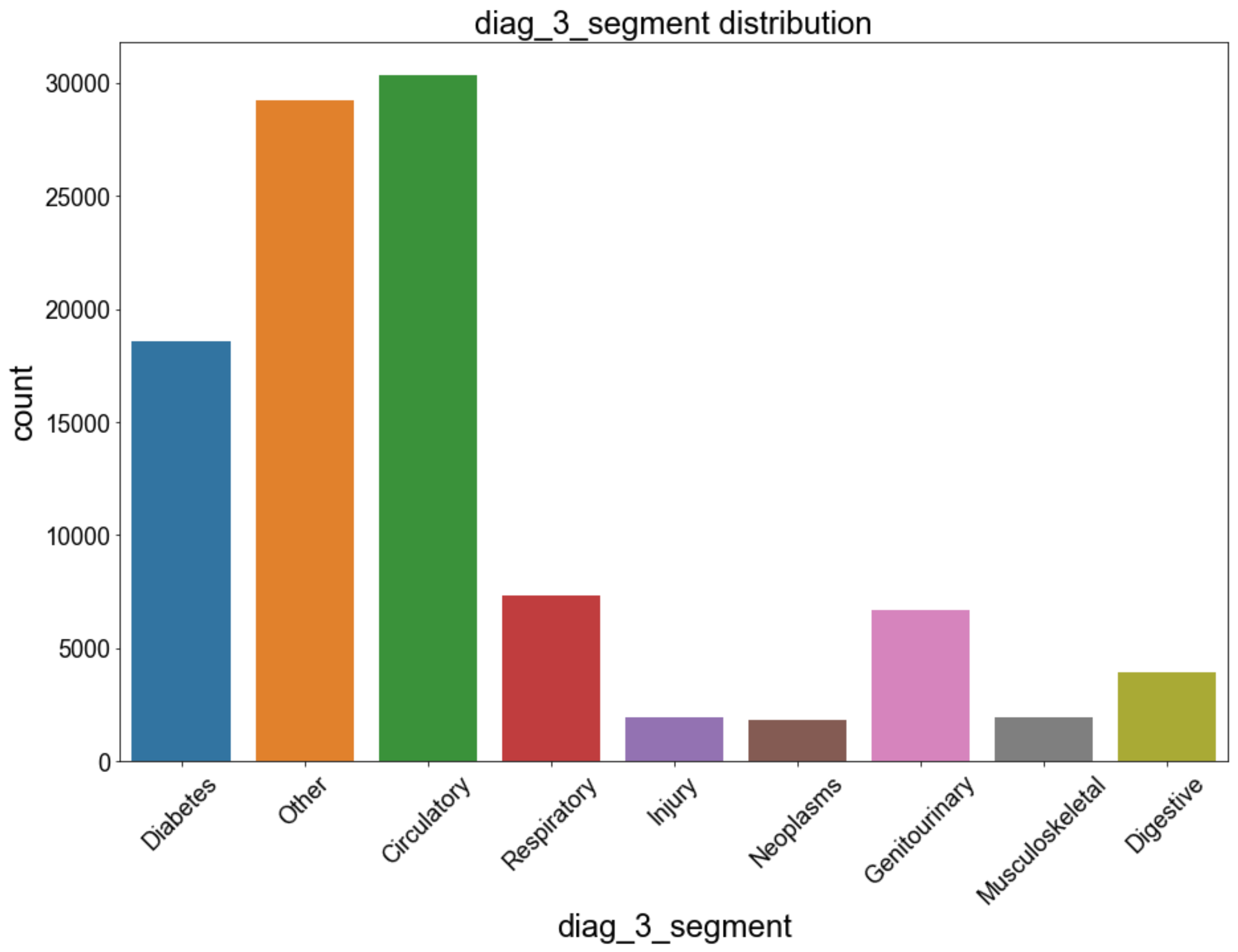
In [ ]:

```
# Diag_2 distribution
visualize_counts('diag_2_segment', data_prcessed_df, title='diag_2_segment distr
```



In [ ]:

```
# Diag_3 distribution
visualize_counts('diag_3_segment', data_prcessed_df, title='diag_3_segment distr
```



```
In [ ]:
# We do not see any more missing data issues
missing_value_stats = ((data_prcessed_df.isnull().sum()/len(data_prcessed_df)*10
missing_value_stats
```

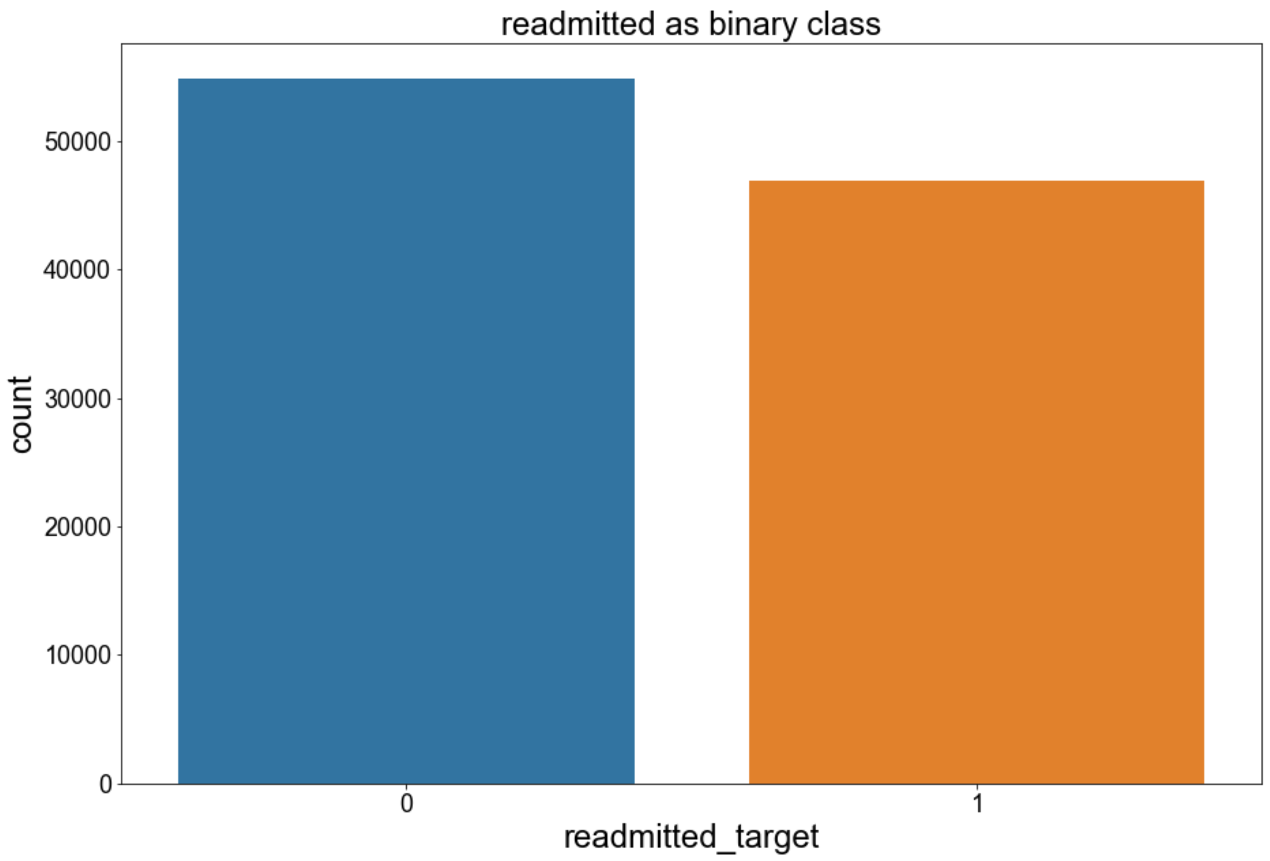
```
Out[ ]:
race                        0.0
gender                      0.0
age                         0.0
admission_type_id           0.0
discharge_disposition_id    0.0
admission_source_id         0.0
time_in_hospital            0.0
payer_code                  0.0
medical_specialty           0.0
num_lab_procedures          0.0
num_procedures              0.0
num_medications             0.0
number_outpatient           0.0
number_emergency            0.0
number_inpatient            0.0
number_diagnoses            0.0
max_glu_serum               0.0
A1Cresult                   0.0
metformin                   0.0
repaglinide                 0.0
nateglinide                 0.0
chlorpropamide              0.0
glimepiride                 0.0
acetohexamide               0.0
```

```
glipizide                   0.0
glyburide                   0.0
tolbutamide                 0.0
pioglitazone                0.0
rosiglitazone               0.0
acarbose                    0.0
miglitol                    0.0
troglitazone                0.0
tolazamide                  0.0
examide                     0.0
citoglipton                 0.0
insulin                     0.0
glyburide-metformin         0.0
glipizide-metformin         0.0
glimepiride-pioglitazone    0.0
metformin-rosiglitazone     0.0
metformin-pioglitazone      0.0
change                      0.0
diabetesMed                 0.0
readmitted                  0.0
num_of_visits               0.0
diag_1_segment              0.0
diag_2_segment              0.0
diag_3_segment              0.0
dtype: float64
```

Data Preparation for Logistic Models

In [ ]:

```
# Make sure our target is binary 0/1  NO = 1

data_model_df = data_prcessed_df.copy()

data_model_df['readmitted_target'] = np.where(data_model_df['readmitted']=='NO',

# Drop original readmitted column
data_model_df = data_model_df.drop("readmitted", axis=1)

visualize_counts('readmitted_target', data_model_df, title= 'readmitted as binar
```



```
In [ ]:
# Admission_type_id , discharge_disposition_id, admission_source_id - int in the
data_model_df['admission_type_id'] = data_model_df['admission_type_id'].astype('
data_model_df['discharge_disposition_id'] = data_model_df['discharge_disposition
data_model_df['admission_source_id'] = data_model_df['admission_source_id'].asty
```

```
In [ ]:
# Change age label to numeric
# Example: [10-20] to 10, [20-30] to 20, etc.

ordinal_col = ['age']

data_model_df['age_label'] = data_model_df['age'].apply(lambda x: int(x.split('-

data_model_df = data_model_df.drop(ordinal_col, axis=1)
```

```
In [ ]:
# Column types: numeric and categorical
# Split so we don't normalize on-hot encoded variables

numerical_cols = data_model_df._get_numeric_data().columns
categorical_cols=  [i for i in data_model_df.columns if data_model_df.dtypes[i]=


print(f"numerical columns: \n {numerical_cols}")
print(f"categorical columns: \n {categorical_cols}")
```

```
numerical columns:
 Index(['time_in_hospital', 'num_lab_procedures', 'num_procedures',
       'num_medications', 'number_outpatient', 'number_emergency',
       'number_inpatient', 'number_diagnoses', 'num_of_visits',
       'readmitted_target', 'age_label'],
      dtype='object')
categorical columns:
 ['race', 'gender', 'admission_type_id', 'discharge_disposition_id', 'admission_
source_id', 'payer_code', 'medical_specialty', 'max_glu_serum', 'A1Cresult', 'me
tformin', 'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride', 'acetoh
examide', 'glipizide', 'glyburide', 'tolbutamide', 'pioglitazone', 'rosiglitazon
e', 'acarbose', 'miglitol', 'troglitazone', 'tolazamide', 'examide', 'citoglipto
n', 'insulin', 'glyburide-metformin', 'glipizide-metformin', 'glimepiride-piogli
tazone', 'metformin-rosiglitazone', 'metformin-pioglitazone', 'change', 'diabete
sMed', 'diag_1_segment', 'diag_2_segment', 'diag_3_segment']
```

In [ ]:

```
# Normalize numeric variables with StandardScaler()
y = data_model_df['readmitted_target']

numerical_cols_df = data_model_df[numerical_cols].copy().drop('readmitted_target
scaler = StandardScaler()

numerical_cols_standard = scaler.fit_transform(numerical_cols_df)
numerical_cols_standard = pd.DataFrame(data = numerical_cols_standard, columns =
```

In [ ]:

```
numerical_cols_standard.head()
```

Out[ ]:

	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatient
0	-1.137649	-0.106517	-0.785398	-1.848268	-0.291461
1	-0.467653	0.808384	-0.785398	0.243390	-0.291461
2	-0.802651	-1.631351	2.145781	-0.371804	1.286748
3	-0.802651	0.045967	-0.199162	-0.002688	-0.291461
4	-1.137649	0.401761	-0.785398	-0.986997	-0.291461

In [ ]:

```
# Create dummy variables for categorical features - One hot encoding

categorical_features = pd.get_dummies(data_model_df[categorical_cols])
```

In [ ]:

```
categorical_features.head()
```

Out[ ]:

	race_AfricanAmerican	race_Asian	race_Caucasian	race_Hispanic	race_Other	gender_Female
0	0	0	1	0	0	1
1	0	0	1	0	0	1
2	1	0	0	0	0	1
3	0	0	1	0	0	0
4	0	0	1	0	0	0

5 rows × 260 columns

In [ ]:

```
model_features  = pd.concat([numerical_cols_standard, categorical_features], axi
In [ ]: model_features.head()
```

Out[ ]:

	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatient
0	-1.137649	-0.106517	-0.785398	-1.848268	-0.291461
1	-0.467653	0.808384	-0.785398	0.243390	-0.291461
2	-0.802651	-1.631351	2.145781	-0.371804	1.286748
3	-0.802651	0.045967	-0.199162	-0.002688	-0.291461
4	-1.137649	0.401761	-0.785398	-0.986997	-0.291461

5 rows × 270 columns

Basic Logistic Regression

```
In [ ]: # Train/test split
        X_train, X_test, y_train, y_test = train_test_split(model_features, y, test_size
```

```
In [ ]: my_model = LogisticRegression(max_iter = 1500)
```

```
In [ ]: my_model.fit(X_train, y_train)
        cross_res = cross_val_score(my_model, X_train, y_train, scoring = 'accuracy')
        print(cross_res)

[0.75987226 0.75772278 0.7605331  0.76170004 0.75869058]
```

L1 Logistic Regression Model

```
In [ ]: l1_model = LogisticRegression(penalty='l1', solver = 'saga', max_iter = 1500)

        split = KFold(n_splits = 5, shuffle = True)

        l1_param_grid = {
            "C": np.random.uniform(low=0, high=20, size=1000)
        }

        L1_cv = RandomizedSearchCV(
            l1_model, l1_param_grid, n_iter=30, cv=split, n_jobs=-1, scoring='accuracy'
        )
```

```
In [ ]: # L1_cv.fit(X_train, y_train) # Fit to train data
        #TODO: uncomment 3 cells later
```

```
In [ ]: # L1_cv.best_estimator_
```

```
In [ ]: # L1_cv.best_score_
```

In [ ]:

```
# Build model based on best parameters found above
feature_cols = X_train.columns
l1_model_best = LogisticRegression(penalty='l1', solver = 'saga', C=0.193, max_i
l1_model_best.fit(X_train, y_train)
```

Out[ ]:

```
LogisticRegression(C=0.193, max_iter=1500, penalty='l1', solver='saga')
```

In [ ]:

```
# Find most important features
l1_features_list = list(zip(feature_cols, l1_model_best.coef_[0]))
l1_features_df = pd.DataFrame(l1_features_list, columns = ['Feature Name', 'Coef
```

In [ ]:

```
l1_features_df['Coef_abs'] = l1_features_df['Coef'].abs()
l1_features_df.sort_values('Coef_abs', ascending=False).head(20)
```

Out[ ]:

	Feature Name	Coef	Coef_abs
36	discharge_disposition_id_11	-7.407892	7.407892
39	discharge_disposition_id_14	-2.783173	2.783173
8	num_of_visits	2.664316	2.664316
38	discharge_disposition_id_13	-1.941925	1.941925
49	discharge_disposition_id_25	-1.188847	1.188847
23	admission_type_id_6	1.102062	1.102062
66	admission_source_id_20	0.970363	0.970363
125	medical_specialty_Pediatrics-Endocrinology	-0.798709	0.798709
148	medical_specialty_Surgery-Cardiovascular/Thoracic	-0.770282	0.770282
43	discharge_disposition_id_18	-0.523367	0.523367
79	payer_code_Missing	0.486335	0.486335
73	payer_code_DM	0.462443	0.462443
112	medical_specialty_ObstetricsandGynecology	-0.435029	0.435029
6	number_inpatient	-0.430669	0.430669
152	medical_specialty_Surgery-Neuro	-0.402592	0.402592
116	medical_specialty_Orthopedics-Reconstructive	-0.402135	0.402135
30	discharge_disposition_id_5	0.385803	0.385803
46	discharge_disposition_id_22	0.380075	0.380075
117	medical_specialty_Osteopath	0.329228	0.329228
65	admission_source_id_17	-0.308556	0.308556

In [ ]:

```
# Evaluate model with test data
y_pred_l1 = l1_model_best.predict(X_test)
```

In [ ]:
```
precision = precision_score(y_true=y_test, y_pred=y_pred_l1)
recall = recall_score(y_true=y_test, y_pred=y_pred_l1)
accuracy = accuracy_score(y_true=y_test, y_pred=y_pred_l1)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2}")
```

```
Accuracy: 0.76
Precision: 0.81
Recall: 0.64
```

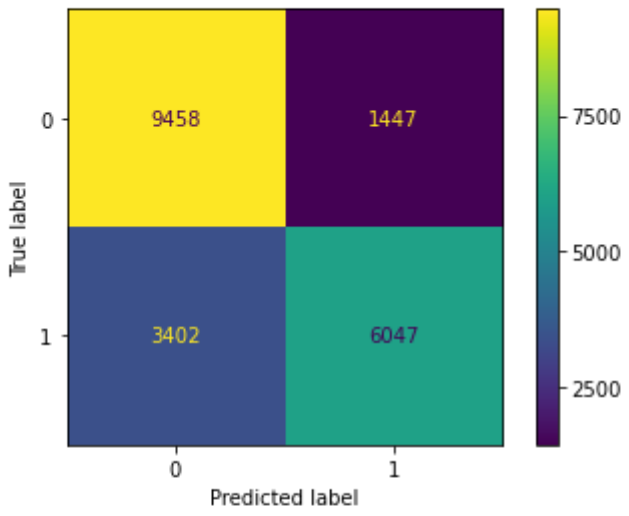
In [ ]:
```
confusion_matrix_l1 = confusion_matrix(y_true=y_test, y_pred=y_pred_l1)
print(confusion_matrix_l1)
```

```
[[9458 1447]
 [3402 6047]]
```

In [ ]:
```
print(classification_report(y_test, y_pred_l1))
```

```
              precision    recall  f1-score   support

           0       0.74      0.87      0.80     10905
           1       0.81      0.64      0.71      9449

    accuracy                           0.76     20354
   macro avg       0.77      0.75      0.75     20354
weighted avg       0.77      0.76      0.76     20354
```

In [ ]:
```
plot_confusion_matrix(l1_model_best, X_test, y_test)
plt.rcParams['font.size'] = '24'
plt.show()
```



In [ ]:
```
# Calculate probability for ROC curve
y_pred_l1_prob = cross_val_predict(l1_model_best, X_test, y_test, method = 'pred
```

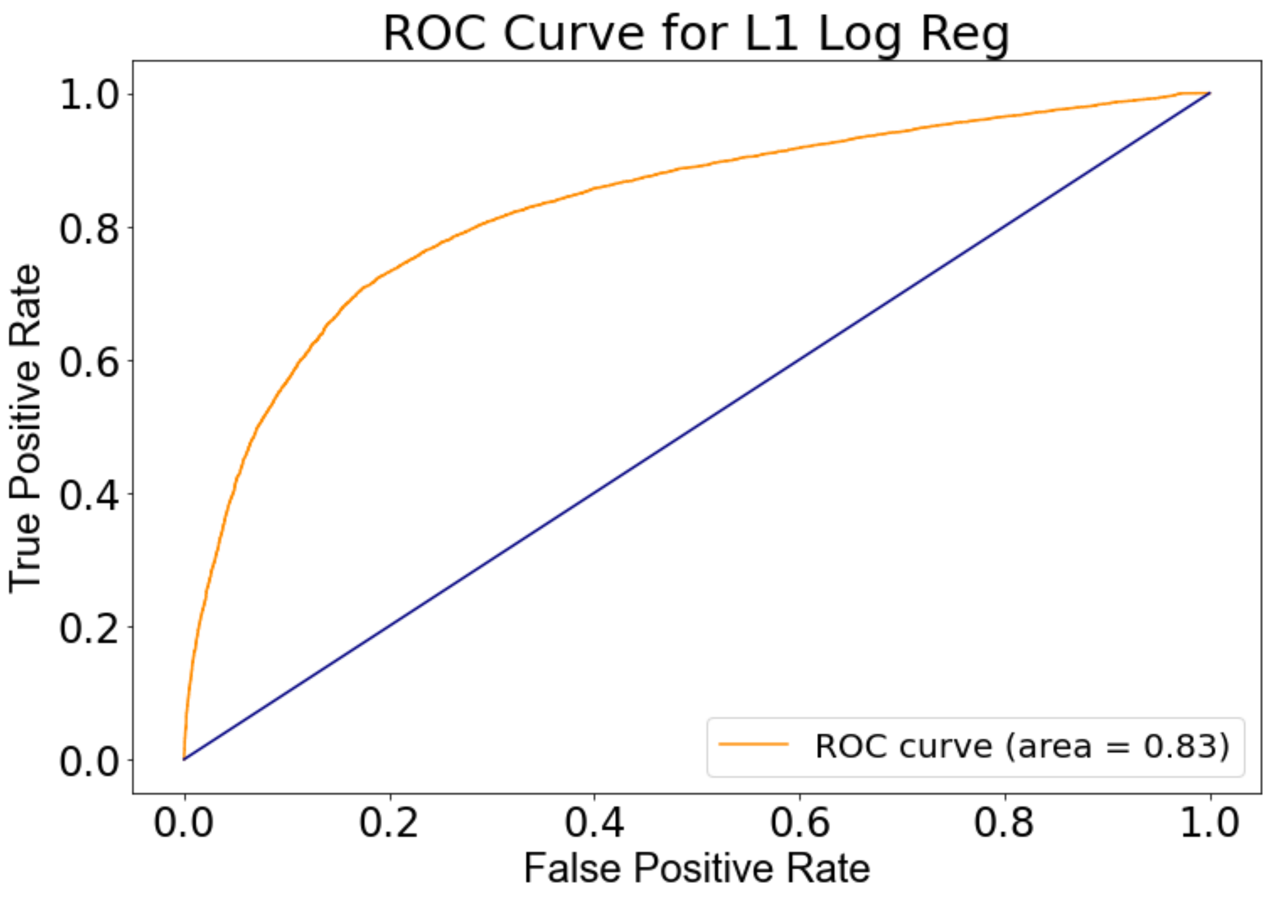
```
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef
```

```
_ did not converge
  warnings.warn("The max_iter was reached which means "
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef
_ did not converge
  warnings.warn("The max_iter was reached which means "
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef
_ did not converge
  warnings.warn("The max_iter was reached which means "
/Users/hallepurdom/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_mode
l/_sag.py:328: ConvergenceWarning: The max_iter was reached which means the coef
_ did not converge
  warnings.warn("The max_iter was reached which means "
```

In [ ]:

```
# ROC curve
n_classes = 2
fpr = dict()
tpr = dict()

roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i],_ = roc_curve(y_test, y_pred_l1_prob[:,1])
    roc_auc[i] = auc(fpr[i], tpr[i])
plt.figure()
plt.figure(figsize=(12, 8))

plt.plot( fpr[1], tpr[1], color = 'darkorange', label = "ROC curve (area = %0.2f
axis_font = {'fontname':'Arial', 'size':'24'}
plt.xlabel('False Positive Rate', **axis_font)
plt.ylabel('True Positive Rate', **axis_font)
plt.title('ROC Curve for L1 Log Reg')
plt.plot([0,1], [0,1], color="navy")
plt.legend(loc = 'lower right', prop={"size":20})
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



L2 Logistic Regression Model

In [ ]:
```
# L2 model tuning parameters
l2_model = LogisticRegression(penalty='l2',  max_iter = 1500)

split = KFold(n_splits = 5, shuffle = True)

l2_param_grid = {
    "C": np.random.uniform(low=0, high=20, size=1000)
}

L2_cv = RandomizedSearchCV(
    l2_model, l2_param_grid, n_iter=30, cv=split, n_jobs=-1, scoring='accuracy'
)
```

In [ ]:
```
# Train model on train data
#L2_cv.fit(X_train, y_train) #TODO: uncomment
```

In [ ]:
```
# TODO: uncomment
# L2_best_estimator = L2_cv.best_estimator_
# L2_best_score = L2_cv.best_score_

# print(f"L2_best_estimator: \n{L2_best_estimator}")
# print(f"L2_best_score:  {L2_best_score}")
```

In [ ]:
```python
# Use best parameters found above for final model
l2_model_best = LogisticRegression(penalty='l2', C=1.93, max_iter=1500,)
l2_model_best.fit(X_train, y_train)
```

Out[ ]:
```
LogisticRegression(C=1.93, max_iter=1500)
```

In [ ]:
```python
# Most Important features from L2
l2_features_list = list(zip(feature_cols, l2_model_best.coef_[0]))
l2_features_df = pd.DataFrame(l2_features_list, columns = ['Feature Name', 'Coef
l2_features_df['Coef_abs'] = l2_features_df['Coef'].abs()
l2_features_df.sort_values('Coef_abs', ascending=False).head(20)
```

Out[ ]:

	Feature Name	Coef	Coef_abs
36	discharge_disposition_id_11	-7.829860	7.829860
8	num_of_visits	2.682692	2.682692
39	discharge_disposition_id_14	-2.257118	2.257118
24	admission_type_id_7	-1.883936	1.883936
87	medical_specialty_AllergyandImmunology	1.785505	1.785505
40	discharge_disposition_id_15	1.506556	1.506556
44	discharge_disposition_id_19	-1.396625	1.396625
208	acarbose_Down	-1.356501	1.356501
38	discharge_disposition_id_13	-1.344573	1.344573
66	admission_source_id_20	1.312253	1.312253
51	discharge_disposition_id_28	1.287519	1.287519
30	discharge_disposition_id_5	1.195759	1.195759
46	discharge_disposition_id_22	1.185288	1.185288
117	medical_specialty_Osteopath	1.176884	1.176884
23	admission_type_id_6	1.156141	1.156141
125	medical_specialty_Pediatrics-Endocrinology	-1.144922	1.144922
37	discharge_disposition_id_12	1.083373	1.083373
227	glyburide-metformin_Down	-1.069509	1.069509
31	discharge_disposition_id_6	1.025688	1.025688
211	acarbose_Up	1.003144	1.003144

In [ ]:
```python
# Evaluate final model with test data
y_pred_l2 = l2_model_best.predict(X_test)

precision = precision_score(y_true=y_test, y_pred=y_pred_l2)
recall = recall_score(y_true=y_test, y_pred=y_pred_l2)
accuracy = accuracy_score(y_true=y_test, y_pred=y_pred_l2)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2}")
```

```
Accuracy: 0.76
Precision: 0.80
Recall: 0.64
```

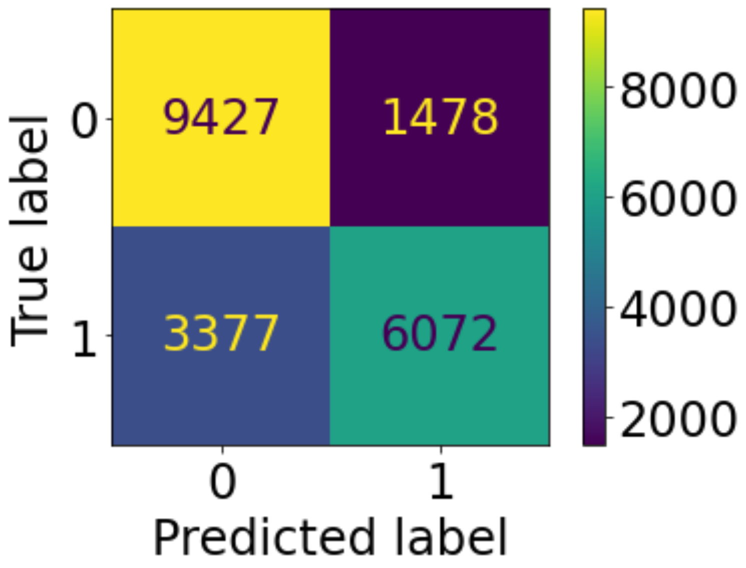
In [ ]:
```
confusion_matrix_l2 = confusion_matrix(y_true=y_test, y_pred=y_pred_l2)
print(confusion_matrix_l2)
```

```
[[9427 1478]
 [3377 6072]]
```

In [ ]:
```
print(classification_report(y_test, y_pred_l2))
```

```
              precision    recall  f1-score   support

           0       0.74      0.86      0.80     10905
           1       0.80      0.64      0.71      9449

    accuracy                           0.76     20354
   macro avg       0.77      0.75      0.75     20354
weighted avg       0.77      0.76      0.76     20354
```

In [ ]:
```
plot_confusion_matrix(l2_model_best, X_test, y_test)
plt.rcParams['font.size'] = '24'
plt.show()
```

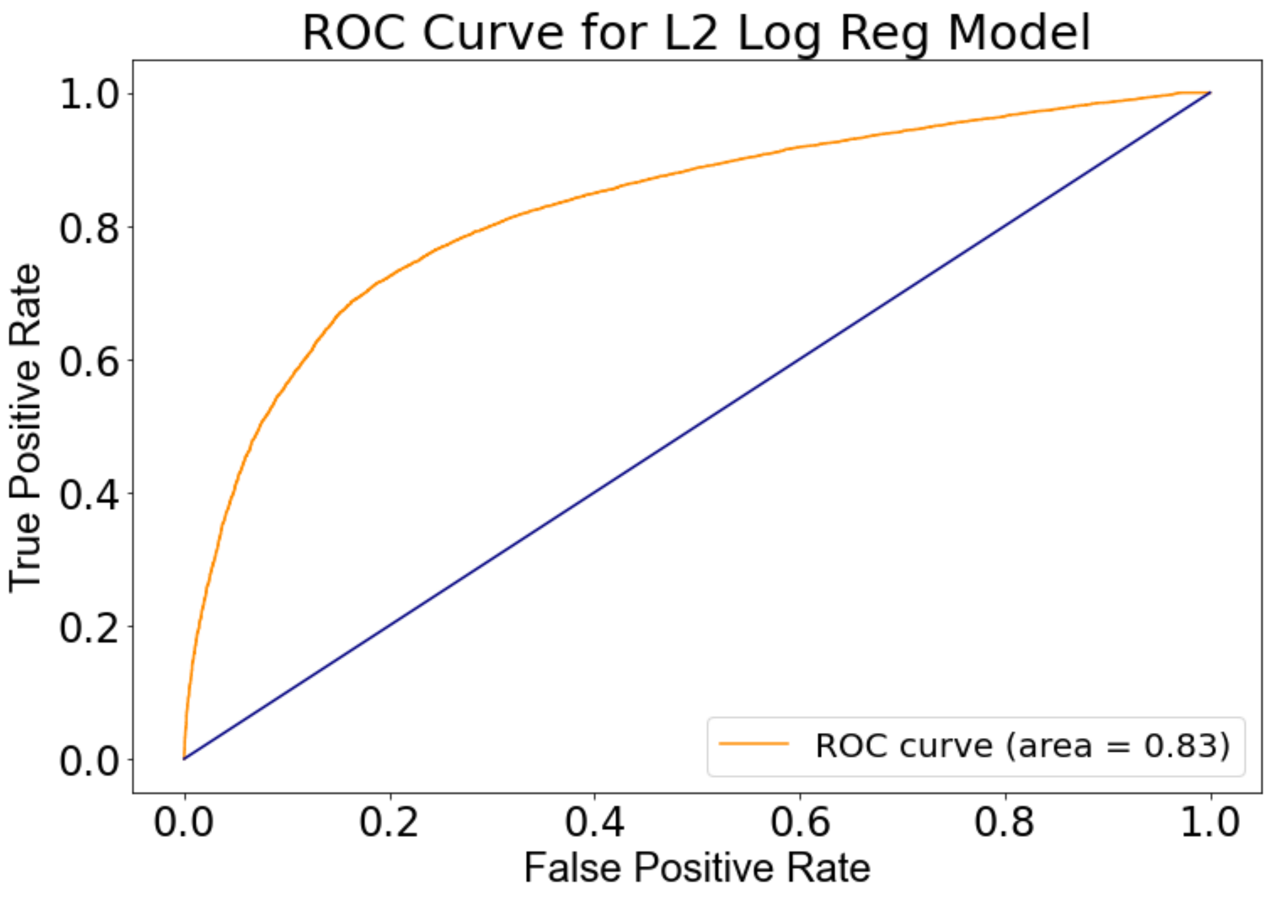


In [ ]:
```
y_pred_l2_prob = cross_val_predict(l2_model_best, X_test, y_test, method = 'pred
```

In [ ]:
```
# ROC Curve for L2 Model
n_classes = 2
fpr = dict()
tpr = dict()

roc_auc = dict()
```

```
for i in range(n_classes):
    fpr[i], tpr[i],_ = roc_curve(y_test, y_pred_l2_prob[:,1])
    roc_auc[i] = auc(fpr[i], tpr[i])
plt.figure()
plt.figure(figsize=(12, 8))

plt.plot( fpr[1], tpr[1], color = 'darkorange', label = "ROC curve (area = %0.2f
axis_font = {'fontname':'Arial', 'size':'24'}
plt.xlabel('False Positive Rate', **axis_font)
plt.ylabel('True Positive Rate', **axis_font)
plt.title('ROC Curve for L2 Log Reg Model')
plt.plot([0,1], [0,1], color="navy")
plt.legend(loc = 'lower right', prop={"size":20})
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```


ROC Curve for L2 Log Reg Model
True Positive Rate
False Positive Rate
ROC curve (area = 0.83)