Case Study II

Randy Kim, Kati Schuerger, Will Sherman

September 19, 2022

1 Introduction

Objective

The objective of this case study was to build a classifier using logistic regression to predict readmission of the patient within 30 days of initial hospitalization. In addition, our analysis sought to identify which variables carry the most importance in predicting hospital readmittance and address imputation of missing values in the provided dataset.

Dataset

The dataset was provided by the medical community, and the subject matter concerns details related to patients with diabetes. The goal of healthcare providers was to limit the number of patients that require readmission. The analysis in this case study may be useful in helping to identify patterns that make a patient more or less likely to be readmitted.

Data points include individual demographic, personal physical, and hospital record data. Hospital record data include such features as admission data, discharge data, length of stay, medical specialty, number of labs/procedures/medications, the results of these interventions, and clinical diagnoses.

2 Methods

Normalization

We used SKLEARN StandardScaler to transform the data to measure features on the same scale. Standard Scaler is a method of standardizing the data, transforming it so that each feature has mean of 0 and a standard deviation of 1.

Diagnoses

One feature we decided to work on initially was the diagnoses (*i.e.*, features *diag_1*, *diag_2*, and *diag_3*) from the original dataset. The feature spaces for these were very large: 716, 748, and 789 categories, respectively. Based on data from BioMed Research International[1], we were able to establish these diagnoses as corresponding to values from the International Classification of Diseases, Ninth Revision, Clinical Modification (ICD-9). We opted to transform these numeric and string vectors into grouped categories based on ICD-9 classifications[2].

[1] Strack, DeShazo, *et al.* Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records. BioMed Research International. Volume 2014 | Article ID 781670 | https://doi.org/10.1155/2014/781670.
[2] National Center for Health Statistics. Center for Disease Control and Prevention. https://www.cdc.gov/nchs/icd/icd9cm.htm.

Missing data

Upon initial review of the data, we identified multiple missing values. One reason for some of the missing data was that this information was gathered over 10 years ago, from several different hospitals, and it was known that the data have gaps.

To preserve as much information as possible, additional analysis was conducted to identify possible options for how we might be able to impute some (if not most) of these missing values.

Imputation of missing values

One of the primary considerations from the clinician's standpoint will likely be in-house labs and diagnoses. The features for labs (*i.e.*, *num_lab_procedures*, *num_procedures*, *num_medications*, *number_outpatient*, *number_emergency*, *number_inpatient*) did not include any missing values. However, the diagnoses—particularly secondary and additional secondary diagnoses—did have some missing information. For these missing values we opted to impute using the mode for each of the features.

Of note was that the diagnoses features did not appear to have correlation in their missingness. We expected that this feature set was missing completely at random or that we were unable to identify the relationship between missing values based on the given data.

The weight feature was missing in 96.86% of the collected data. Therefore, we opted to exclude this from the possible feature space, as this may have caused concerns about the confidence of features in the study. We also excluded the patient number as this should not impact modeling on readmission.

Payer code appeared to have values missing not at random. There was a high negative correlation between missing payer codes and encounter ID (Figure 1). The other features that included missing data were race, medical specialty, admission type, discharge disposition, and admission source. For these, mode imputation was chosen again based on underlying distribution of the data.

Ethical considerations regarding patient race

When conducting statistical analysis, it is important to be mindful of any possibilities for bias or potential discrimination against (or in favor of) a portion of the population being analyzed.

One such feature in this dataset was race. If this analysis were intended to build a model for setting insurance premiums, sorting individuals by race could certainly be considered as discriminatory or biased. For this case, however, we believe that we do not encounter this difficulty of bias. Here, we are talking about patients and diabetes in relationship to hospital readmission. The medical community is aware that diabetes affects different demographics in different ways. For this reason, race may be a useful feature in helping determine likelihood of the patient to return to the hospital within 30 days.

Logistic regression and the Sigmoid function

At a very high-level, logistic regression is a method used to build models where the task is one of categorization, meaning the options for the target variable is a discrete value (for example: yes or no, 0 or 1, color: red or white or blue).

Our target variable is categorical, with two options. Either yes, the patient is likely to be readmitted within 30 days; or no, the patient is not likely to be readmitted within 30 days. Classification will be based on the probability score generated by the model.

Logistic regression makes use of the Sigmoid function, which seeks to determine a probability score that the target variable belongs to one of the category options. The analyst then applies a threshold value to determine the cutoff for what determines which value-option the record belongs to.

Selecting threshold for category in logistic model

The go-to cutoff value is 50%, meaning that any observation scoring lower than 50% would belong to class 0, and anything over 50% belongs to class 1.

Train/Test/Validate split

Before we began modeling, we randomized the arrangement of the data samples and separated the data into 2 groups: a training dataset (to fit the model), and a validation dataset (holdout group of 20%) that was used to measure model performance on unseen data—evaluating whether the model generalizes well.

Scoring metric

The task was to create a logistic regression model for readmission predictions. We used accuracy scores in combination with 10-fold cross-validation to get model scores and compare models. Cross-validation is a way to assess different portions of the data using resampling to identify the model parameters. This was used with regularized logistic regression to prevent over-fitting problem. We iterated through the values of C (the penalty term) to generate optimal loss function prior to fitting the regularized logistic regression.

3 Results

Missing values

We identified several features with missing values. The payer_code feature had a distinct progression of missingness (Figure 1). The level of correlation between missingness and time series is difficult to quantify and beyond the scope of this particular project.

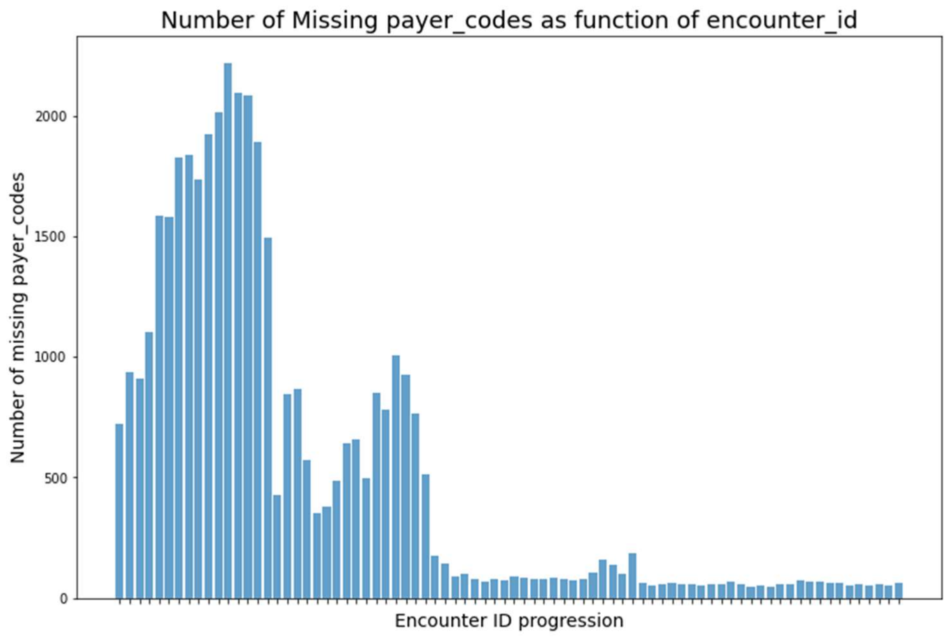


Figure 1. For increasing encounter_id, the number of missing payer_code values appears to decrease.

The diagnoses—particularly secondary and additional secondary diagnoses—had missing information (*diag_1* had 21, *diag_2* had 358, and *diag_3* had 1423 missing values). Due to the relatively small amount of missing data, we proceeded to do mode imputation for each of these: *diag_1* was circulatory disease, *diag_2* was metabolic disease, and *diag_3* was diabetes.

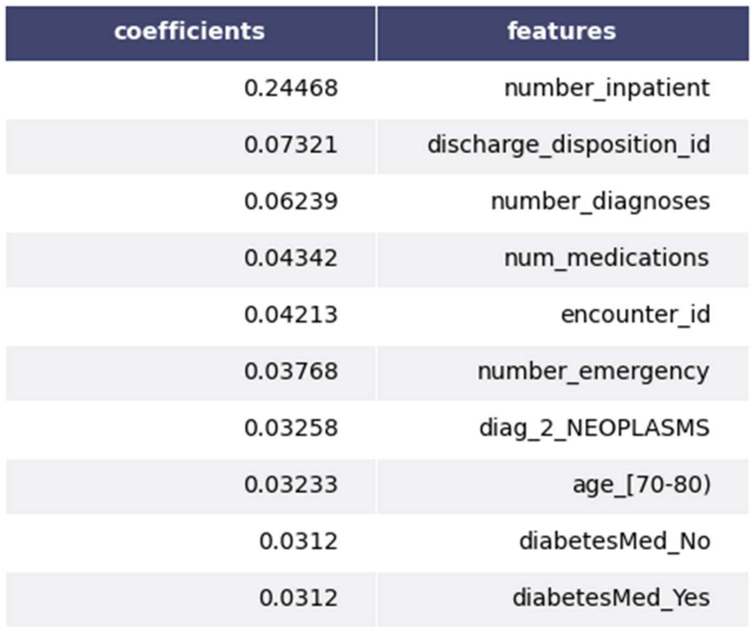
The optimization of the L2 penalty coefficient (*C* in this case) was generated by iterating over a 100 sample values in log space between -10 and 10. The optimal value for this hyperparameter was 0.00029.

The imputation for each of the following was also implemented using mode after distribution assessment: *race* as Caucasian, *medical specialty* as Internal Medicine, *admission type* as Emergency, *discharge disposition* as Discharged to home, and admission source

Feature importance

Based on the individual demographic, personal physical, and hospital record data that were provided, we pre-selected features of interest prior to model selection. Then, using L2 regularization, we identified the top ten features for the logistic model (Table 1).

Table 1. Model coefficients by weight.

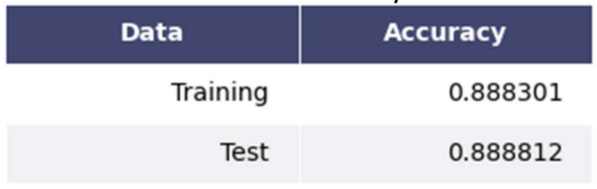


coefficients	features
0.24468	number_inpatient
0.07321	discharge_disposition_id
0.06239	number_diagnoses
0.04342	num_medications
0.04213	encounter_id
0.03768	number_emergency
0.03258	diag_2_NEOPLASMS
0.03233	age_[70-80)
0.0312	diabetesMed_No
0.0312	diabetesMed_Yes

Table 1. Provides the top 10 coefficients for the Logistic Regression model on diabetes readmission.

The most important feature appeared to be the number of inpatient visits by the patient in the year preceding the current encounter. The discharge disposition also appeared to have a significant impact along with the number of diagnoses and the number of medications the patient was receiving.

Table 2. Initial Accuracy Scores.

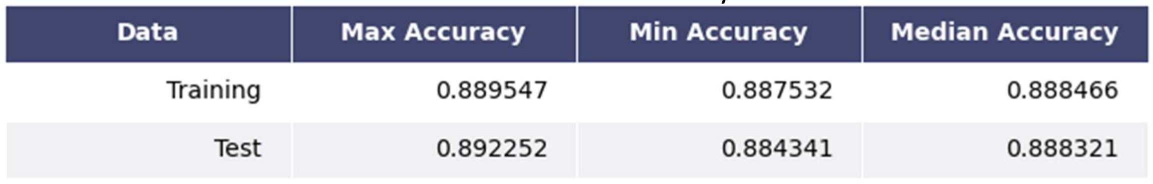


Data	Accuracy
Training	0.888301
Test	0.888812

Table 2. General accuracy scoring by the logistic model against the whole training dataset and the holdout (Test) dataset.

The accuracy score of the model against the training data was 88.83% accurate (Table 2). Based on the initial split, the model outperformed this accuracy on the smaller test set with a score of 88.88% accuracy. Assessment against other random seeds (43 randomly selected seeds) showed that the range of the accuracies in test performance were wider with median 88.43% (Table 3).

Table 3. Random Seed Accuracy Scores.



Data	Max Accuracy	Min Accuracy	Median Accuracy
Training	0.889547	0.887532	0.888466
Test	0.892252	0.884341	0.888321

Table 3. Range and behavior of accuracy scores based on random seed splitting of train/test split.

4 Conclusion

The regularized logistic regression model performed well for both training and unseen data with 88.83% accuracy score. We could improve this accuracy score by refining the variables such as diabetesMed_No and diabetesMed_Yes as these were given as coefficients. We can also try modeling by predictive mean match imputation rather than mode imputation. For example, we could predict missing data of medical_specialty by admission_type_id.

Future studies could try to divide patients by medical specialty or other categorical variables. This would need further investigation to see if there is evidence suggesting that categorical variables and readmission show correlations.

Finally, we would want to meet with the investigators to receive input on variable selection methods for further refinement of the model. Notably, *encounter_id* is included in the model. In a perfect world, when someone is being discharged, the date of their admission shouldn't dictate their health outcome (over the span of time this data was collected). However, that is under the assumption that *encounter_id* is linked to date information and not location information. Regardless, this variable is of particular interest and should be included with caution as there are also ethical concerns if this is location-based data. Namely, predicting readmission on locale if the demographics are different.

Code

```
# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn

# %%
### Will - read data ###
data_1 = pd.read_csv(r'C:\Users\sherm\Documents\Grad School - Classes\MSDS - 7333 -
Quantifying the World\Case Study 2\dataset_diabetes\diabetic_data.csv')
### Randy - read data ###
# data_1 = pd.read_csv("diabetic_data.csv")
### Kati - read data ###
# data_1 =

print(data_1.shape)
print(data_1.columns)
data_1.head(10)

# %%
data_1.isnull().any()

# %%
print(len(data_1[data_1['diag_1']=='?']))
print(len(data_1[data_1['diag_2']=='?']))
print(len(data_1[data_1['diag_3']=='?']))

# %%
from collections import Counter

diag_1 = Counter(list(data_1['diag_1'])).most_common(1)[0][0]
diag_2 = Counter(list(data_1['diag_2'])).most_common(1)[0][0]
diag_3 = Counter(list(data_1['diag_3'])).most_common(1)[0][0]
data_1['diag_1'] = data_1['diag_1'].apply(lambda x : diag_1 if x == '?' else x)
data_1['diag_2'] = data_1['diag_2'].apply(lambda x : diag_2 if x == '?' else x)
data_1['diag_3'] = data_1['diag_3'].apply(lambda x : diag_3 if x == '?' else x)

# credit for above code goes to Sanchin Ranveer
#      https://medium.com/analytics-vidhya/diabetes-130-us-hospitals-for-years-1999-2008-
e18d69beea4d

# %%
print(len(data_1[data_1['weight']=='?']), 'out of', len(data_1['weight']))
print('weight values missing at {}%'.format(round(98569/101766*100, 2)))

# %%
data_1 = data_1.drop(['weight'], axis=1)
data_1 = data_1.drop(['patient_nbr'], axis=1)

# %%
data_1 = data_1.replace('?', np.NaN)

# %%
data_1.isnull().any()

# %%
# List of vectors with np.NaN values that need to be imputed

vectors_w_NAs = data_1.columns[data_1.isnull().any()].tolist()
print(vectors_w_NAs)
```

```
# %%
print(len(data_1[data_1['gender']=='Unknown/Invalid']))
data_1_update = data_1.drop(index = data_1[data_1['gender']=='Unknown/Invalid'].index)

# %%
# Missing Not At Random for Payer Code
sorted_df = data_1_update.sort_values(by=['encounter_id'])
sorted_df['bins'] = pd.cut(data_1_update['encounter_id'], 80)
sorted_df['na_counts'] = sorted_df['payer_code'].isnull()

grouped = sorted_df.groupby(sorted_df['bins'], as_index=False).sum()
x = grouped['bins'].astype(str)
y = grouped['na_counts']

fig, ax = plt.subplots(figsize=(12,8))
plt.xlabel('Encounter ID progression', fontsize=14)
plt.ylabel('Number of missing payer_codes', fontsize=14)
plt.title('Number of Missing payer_codes as function of encounter_id', fontsize=18)
ax.set_xticklabels([])
plt.bar(x, y, alpha=.7)


# %%
data_1_update['diag_1'] = data_1_update['diag_1'].str.split('.').str[0]
data_1_update['diag_2'] = data_1_update['diag_2'].str.split('.').str[0]
data_1_update['diag_3'] = data_1_update['diag_3'].str.split('.').str[0]

# %%
def setall(d, keys, value):
    for k in keys:
        d[k] = value

numeric_codes = {}
setall(numeric_codes, range(1, 140), 'INFECTIOUS AND PARASITIC DISEASES')
setall(numeric_codes, range(140, 240), 'NEOPLASMS')
setall(numeric_codes, range(240, 280), 'ENDOCRINE, NUTRITIONAL AND METABOLIC DISEASES, AND
IMMUNITY DISORDERS')
setall(numeric_codes, range(280, 290), 'DISEASES OF THE BLOOD AND BLOOD-FORMING ORGANS')
setall(numeric_codes,  range(290,  320),  'MENTAL,  BEHAVIORAL  AND  NEURODEVELOPMENTAL
DISORDERS')
setall(numeric_codes, range(320, 390), 'DISEASES OF THE NERVOUS SYSTEM AND SENSE ORGANS')
setall(numeric_codes, range(390, 460), 'DISEASES OF THE CIRCULATORY SYSTEM')
setall(numeric_codes, range(460, 520), 'DISEASES OF THE RESPIRATORY SYSTEM')
setall(numeric_codes, range(520, 580), 'DISEASES OF THE DIGESTIVE SYSTEM')
setall(numeric_codes, range(580, 630), 'DISEASES OF THE GENITOURINARY SYSTEM')
setall(numeric_codes, range(630, 680), 'COMPLICATIONS OF PREGNANCY, CHILDBIRTH, AND THE
PUERPERIUM')
setall(numeric_codes, range(680, 710), 'DISEASES OF THE SKIN AND SUBCUTANEOUS TISSUE')
setall(numeric_codes,  range(710,  740),  'DISEASES  OF  THE  MUSCULOSKELETAL  SYSTEM  AND
CONNECTIVE TISSUE')
setall(numeric_codes, range(740, 760), 'CONGENITAL ANOMALIES')
setall(numeric_codes, range(760, 780), 'CERTAIN CONDITIONS ORIGINATING IN THE PERINATAL
PERIOD')
setall(numeric_codes, range(780, 800), 'SYMPTOMS, SIGNS, AND ILL-DEFINED CONDITIONS')
setall(numeric_codes, range(800, 1000), 'INJURY AND POISONING')

def new_setall(d, letter, keys, value):
    for k in keys:
        d[letter + str(k)] = value

extra_codes={250: 'DIABETES RELATED'}

new_setall(extra_codes,  'E',  range(800,999),  'OTHER  EXTERNAL  CAUSES  OF  INJURY  OR
POISONING')
new_setall(extra_codes, 'V0', range(1,10), 'HEALTH HAZARD - COMMUNICABLE DISEASE')
new_setall(extra_codes, 'V', range(10,40), 'FAMILY HISTORY OR REPRODUCTIVE DEVELOPMENT
ISSUE')
new_setall(extra_codes, 'V', range(40,50), 'NERVE/BRAIN ISSUES, ORGAN TRANSPLANT, MACHINE
SUPPORT, OTHER')
new_setall(extra_codes, 'V', range(50,60), 'POST-PROCEDURE CARE')
new_setall(extra_codes, 'V', range(60,70), 'ECONOMIC, FAMILY, CONVALESCENCE, PALLIATIVE
CARE OR FOLLOW-UP EXAM')
new_setall(extra_codes, 'V', range(70,90), 'GENETICS, HORMONAL, UNREPORTED DIAGNOSIS,
OTHER')
new_setall(extra_codes, 'V', range(90,99), 'MULTIPLE GESTATION PLACENTA STATUS')

# diagnoses_codes = numeric_codes | extra_codes

# %%
diagnoses_codes = {**numeric_codes, **extra_codes}

# %%
def transform_bad_vals(vector):
    for ind, code in vector.iteritems():
        try:
            code = int(code)
            if code not in diagnoses_codes.keys():
                raise ValueError('Incorrect key-value provided: {} {}'.format(ind, code))
            else:
                vector[ind] = diagnoses_codes[code]
        except:
            if code not in diagnoses_codes.keys():
                raise ValueError('Other error: {} {}'.format(ind, code))
            else:
                vector[ind] = diagnoses_codes[code]
    return vector

# %%
pd.options.mode.chained_assignment = None  # default='warn'

data_1_update['diag_1'] = transform_bad_vals(data_1_update['diag_1'])
data_1_update['diag_2'] = transform_bad_vals(data_1_update['diag_2'])
data_1_update['diag_3'] = transform_bad_vals(data_1_update['diag_3'])

data_1_update.loc[:, ['diag_1','diag_2','diag_3']].head()

# %%
set(data_1_update['admission_type_id'])

# %%
### Will - read data ###
data_2 = pd.read_csv(r'C:\Users\sherm\Documents\Grad School - Classes\MSDS - 7333 -
Quantifying the World\Case Study 2\dataset_diabetes\IDs_mapping.csv')
### Randy - read data ###
# data_2 = pd.read_csv("IDs_mapping.csv")
### Kati - read data ###
# data_2 =

pd.set_option('display.max_rows', 70)
print(data_2.shape)
print(data_2.columns)
data_2
### Frick this data it's messy *ugh* ###
### Original csv has weird formatting, extra spaces

# %%
admission_type_id = data_2.iloc[0:8, :]
discharge_disposition_id = data_2.iloc[10:40, :]
admission_source_id = data_2.iloc[42:, :]
```

```
# %%
admission_type_id.head(10)

# %%
discharge_disposition_id.rename(columns={'admission_type_id':'discharge_disposition_id'}
, inplace=True)
discharge_disposition_id.head(10)

# %%
discharge_disposition_id.rename(columns={'admission_type_id':'discharge_disposition_id'}
, inplace=True)
discharge_disposition_id.head(10)

# %%
admission_source_id.rename(columns={'admission_type_id':'admission_source_id'},
inplace=True)
admission_source_id

# %% [markdown]
# ### Still need to finish transforming
# ##### The *admission_type_id*, *discharge_disposition_id*, *admission_source_id* in the
original dataframe to categorical values
# ##### (See Above)

# %%
# # Randy
# what = []
# for i in range(0,len(data_1)):
#     try:
#                                               data_1["admission_type_id"].iloc[i]     =
admission_type_id["description"].iloc[int(data_1["admission_type_id"].iloc[i]-1)]
#                                              data_1["discharge_disposition_id"].iloc[i]
=discharge_disposition_id["description"].iloc[int(data_1["discharge_disposition_id"].ilo
c[i]-1)]
#                                             data_1["admission_source_id"].iloc[i]     =
admission_source_id["description"].iloc[int(data_1["admission_source_id"].iloc[i]-1)]
#     except:
#         what.append(i)
#         continue

# %%
# Randy
data_1_clean = data_1_update.copy()

# %%
data_1_clean['diag_1']

# %%
# Randy
colname = list(data_1_clean.columns)
mia = []

for i in range(len(data_1_clean.columns)):
  count = data_1_clean[data_1_clean.columns[i]].isna().sum()
  if count > 0:
    mia.append(data_1_clean.columns[i])
    print("Column '{col}' has {ct} NAs".format(col = colname[i], ct = count))

# %% [markdown]
# ### Still need to finish imputation
# ##### (See Below)

# %%
# Randy
for i in mia:
    plt.figure(figsize=(15,5))
    data_1_clean[i].value_counts().plot(kind='bar')
    plt.title(i)
    plt.show()

# %%
data_1_clean["race"] = data_1_clean["race"].fillna('Caucasian')
data_1_clean["admission_type_id"] = data_1_clean["admission_type_id"].fillna('Emergency')
data_1_clean["discharge_disposition_id"]                                              =
data_1_clean["discharge_disposition_id"].fillna('Discharged to home')
data_1_clean["medical_specialty"]                                                     =
data_1_clean["medical_specialty"].fillna('InternalMedicine')
data_1_clean["payer_code"] = data_1_clean["payer_code"].fillna('MC')

# %%
# Randy
colname = list(data_1_clean.columns)
mia = []

for i in range(len(data_1_clean.columns)):
  count = data_1_clean[data_1_clean.columns[i]].isna().sum()
  if count > 0:
    mia.append(data_1_clean.columns[i])
    print("Column '{col}' has {ct} NAs".format(col = colname[i], ct = count))

if mia:
    print("There are missing values")
else:
    print("All gucci")

# %%
data_1_clean.head(10)

# %% [markdown]
# ### MODELING (and Setup) ###

# %%
x0_data = data_1_clean.drop(['readmitted'], axis=1)
y0_data = data_1_clean.loc[:, 'readmitted']

column_cats = [col  for col, dt in x0_data.dtypes.items() if dt == object]
column_cats

model_data = pd.get_dummies(x0_data, prefix=column_cats)
print(len(model_data.columns))
model_data

# %%
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# %%
column_nums = [x for x in model_data.columns if x not in column_cats]

# column_nums
model_data[column_nums].head(2)

# %%
x_scaled = model_data.copy()
x_scaled[column_nums] = scaler.fit_transform(x_scaled[column_nums])
# x_scaled = pd.DataFrame(x_scaled, columns=x_data.columns)
x_scaled[column_nums].head()

# %%
```

```
x_scaled

# %%
y_data = y0_data.copy()

# %%
y_data.head()

# %%
# LABELS FOR OUTCOMES
y_data.loc[y_data == '<30'] = 1
y_data.loc[y_data == '>30'] = 0
y_data.loc[y_data == 'No'] = 0
y_data.loc[y_data == 'NO'] = 0
y_data = y_data.astype('int')

# %%
y_data.head(20)

# %% [markdown]
# ### Actual Modeling ###

# %%
# train/test/split
from sklearn.model_selection import train_test_split


X_train,  X_test,  y_train,  y_test  =  train_test_split(x_scaled,  y_data,  test_size=0.2,
random_state=1)

# %%
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, accuracy_score

folder = KFold(shuffle=True, n_splits=10)
log_reg = LogisticRegression(penalty = 'l2')

# %%
c_vals = np.logspace(-10,10,100)
cs_for_plots = []
best = -10000000000
best_c = 'error'

for i in c_vals:
    log_reg.C = i
    out  =  cross_val_score(log_reg,  X_train,  y_train,  scoring='accuracy',  cv=folder,
n_jobs=10).mean()
    cs_for_plots.append([i,out])

    if(out > best):
        best = out
        best_c = i

print('\tbest accuracy\t\tbest C\n',best, '\t', best_c)

# %%
model = LogisticRegression(penalty = 'l2', C=best_c).fit(X_train, y_train)
train_preds = model.predict(X_train)
train_score = accuracy_score(train_preds, y_train)
train_score

# %%
test_preds = model.predict(X_test)
```

```
test_score = accuracy_score(test_preds, y_test)
test_score

# %%
train_verify = []
test_verify = []
for state in range(100,745,15):
    X_train, X_test, y_train, y_test = train_test_split(x_scaled, y_data, test_size=0.2,
random_state=state)
    model = LogisticRegression(penalty = 'l2', C=best_c).fit(X_train, y_train)
    train_preds = model.predict(X_train)
    train_verify.append(accuracy_score(train_preds, y_train))
    test_preds = model.predict(X_test)
    test_verify.append(accuracy_score(test_preds, y_test))

# %%
import six

feature_df = pd.DataFrame({'coefficients':abs(model.coef_.flatten()),
                        'features':X_train.columns})
top10 = feature_df.sort_values('coefficients', ascending=False).head(10)
top10 = top10.round({'coefficients':5})

def render_mpl_table(data, col_width=3.0, row_height=0.625, font_size=14,
                     header_color='#40466e', row_colors=['#f1f1f2', 'w'], edge_color='w',
                     bbox=[0, 0, 1, 1], header_columns=0,
                     ax=None, **kwargs):
    if ax is None:
        size  =  (np.array(data.shape[::-1])  +  np.array([0,  1]))  *  np.array([col_width,
row_height])
        fig, ax = plt.subplots(figsize=size)
        ax.axis('off')
    mpl_table   =   ax.table(cellText=data.values,   bbox=bbox,   colLabels=data.columns,
**kwargs)
    mpl_table.auto_set_font_size(False)
    mpl_table.set_fontsize(font_size)

    for k, cell in mpl_table._cells.items():
        cell.set_edgecolor(edge_color)
        if k[0] == 0 or k[1] < header_columns:
            cell.set_text_props(weight='bold', color='w')
            cell.set_facecolor(header_color)
        else:
            cell.set_facecolor(row_colors[k[0]%len(row_colors) ])
    return ax.get_figure(), ax

fig,ax = render_mpl_table(top10, header_columns=0, col_width=4.0)
fig.savefig("table_mpl.png")

#  credit  to  volodymyr:  https://stackoverflow.com/questions/19726663/how-to-save-the-
pandas-dataframe-series-data-as-a-figure

# %%
L_features = abs(model.coef_.flatten())
L_features = (L_features > 0.000001).astype(int)
L_features = L_features.astype(np.bool)

model_metrics = pd.DataFrame({'Data':['Training','Test'],
                              'Accuracy':[train_score,test_score]})
# model_metrics
model_metrics = model_metrics.round({'Accuracy':6})
fig,ax = render_mpl_table(model_metrics, header_columns=0, col_width=3.0)
fig.savefig("metrics_tbl.png")

# %%
```

```
test_verify.sort()
len(test_verify)

# %%
from statistics import median

random_competition = pd.DataFrame({'Data':['Training','Test'],
                                   'Max Accuracy':[max(train_verify), max(test_verify)],
                                   'Min Accuracy':[min(train_verify), min(test_verify)],
                                   'Median               Accuracy':[median(train_verify),
median(test_verify)]})

random_competition = random_competition.round({'Max Accuracy':6,
                                               'Min Accuracy':6,
                                               'Median Accuracy':6})
fig,ax = render_mpl_table(random_competition, header_columns=0, col_width=3.0)
fig.savefig("final_tbl.png")
```