Topic: Analyzing Factors and Predicting Hospital Re-admission for Diabetes Patients

Problem Statement

The dataset being chosen is Diabetes 130-US hospitals for years 1999-2008 Data Set. It represents 10 years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks for diabetes patients. Over 50 features representing patient and hospital outcomes were recorded. There are two objectives that I have defined for the project:

- Data analysis: find, analyze, and establish correlation between different feature attributes
- Machine learning: build a model to predict hospital readmission

Dataset Analysis

Dataset creation This dataset is built using the Health Facts database, a national data warehouse that collects comprehensive clinical records across the hospital throughout the United States. The Health Facts data used was an extract representing 10 years (1999–2008) of clinical care at 130 hospitals and integrated delivery networks throughout the United States. The database consists of 41 tables in a fact-dimension schema and a total of 117 features. The database includes 74,036,643 unique encounters (visits) that correspond to 17,880,231 unique patients and 2,889,571 providers. To create the dataset, select information was extracted from this original database that satisfied the following criteria:

1. It is an inpatient encounter (a hospital admission)
2. It is a diabetic encounter, that is, one during which any kind of diabetes was entered to the system as a diagnosis
3. The length of stay was at least 1 day and at most 14 days
4. Laboratory tests were performed during the encounter
5. Medications were administered during the encounter

A total of 101,766 instances were identified to fulfill the above five criteria. Feature selection was performed by clinical experts and only attributes that were potentially relevant with diabetic condition were retained. A total of 55 (out of 114) attributes were selected describing the diabetic encounters, including demographics, diagnoses, diabetic medications, number of visits in the year preceding the encounter, and payer information. However, the dataset available on the UCI repository has only 50 features. The full list is shown below:

In [2]:
```
import pandas as pd
```

```
df = pd.read_csv('diabetic_data.csv')
print(f'There are a total of {len(df.columns)} features for every patient ac
print(f'Total number of datapoints are {len(df)}')
```

```
There are a total of 50 features for every patient admitted. The full list is
enumerated below ['encounter_id', 'patient_nbr', 'race', 'gender', 'age', 'we
ight', 'admission_type_id', 'discharge_disposition_id', 'admission_source_i
d', 'time_in_hospital', 'payer_code', 'medical_specialty', 'num_lab_procedure
s', 'num_procedures', 'num_medications', 'number_outpatient', 'number_emergen
cy', 'number_inpatient', 'diag_1', 'diag_2', 'diag_3', 'number_diagnoses', 'm
ax_glu_serum', 'A1Cresult', 'metformin', 'repaglinide', 'nateglinide', 'chlor
propamide', 'glimepiride', 'acetohexamide', 'glipizide', 'glyburide', 'tolbut
amide', 'pioglitazone', 'rosiglitazone', 'acarbose', 'miglitol', 'troglitazon
e', 'tolazamide', 'examide', 'citoglipton', 'insulin', 'glyburide-metformin',
'glipizide-metformin', 'glimepiride-pioglitazone', 'metformin-rosiglitazone',
'metformin-pioglitazone', 'change', 'diabetesMed', 'readmitted']
Total number of datapoints are 101766
```

Since the study conducted with this dataset were interested in factors that lead to early readmission, the readmission attribute (outcome) is defined as having two values: "readmitted", if the patient was readmitted within 30 days of discharge or "otherwise", which covers both readmission after 30 days and no readmission at all.

There were several features that could not be treated directly since they had a high percentage of missing values. These features were weight (97% values missing), payer code (40%), and medical specialty (47%). Weight attribute was considered to be too sparse and it was not included in further analysis. Medical specialty attribute was maintained, adding the value “missing” in order to account for missing values. Large percentage of missing values of the weight attribute can be explained by the fact that prior to the HITECH legislation of the American Reinvestment and Recovery Act in 2009 hospitals and clinics were not required to capture it in a structured format.

The preliminary dataset contained multiple inpatient visits for some patients and the observations could not be considered as statistically independent, an assumption of the logistic regression model. Thus, only one encounter per patient is used; in particular, we considered only the first encounter for each patient as the primary admission and determined whether or not they were readmitted within 30 days. Additionally, the authors removed all encounters that resulted in either discharge to a hospice or patient death were removed, to avoid biasing our analysis. After performing the above-described operations, we were left with 69,984 encounters that constituted the final dataset for analysis. The variables chosen to control for patient demographic and illness severity were gender, age, race, admission source, discharge disposition, primary diagnosis , medical specialty of the admitting physician, and time spent in the hospital.

Summary: To summarize, the dataset consists of hospital admissions of length between one and 14 days that did not result in a patient death or discharge to a hospice. Each encounter corresponds to a unique patient diagnosed with diabetes, although the primary diagnosis may be different. During each of the analyzed encounters, lab tests were ordered and medication was administered.

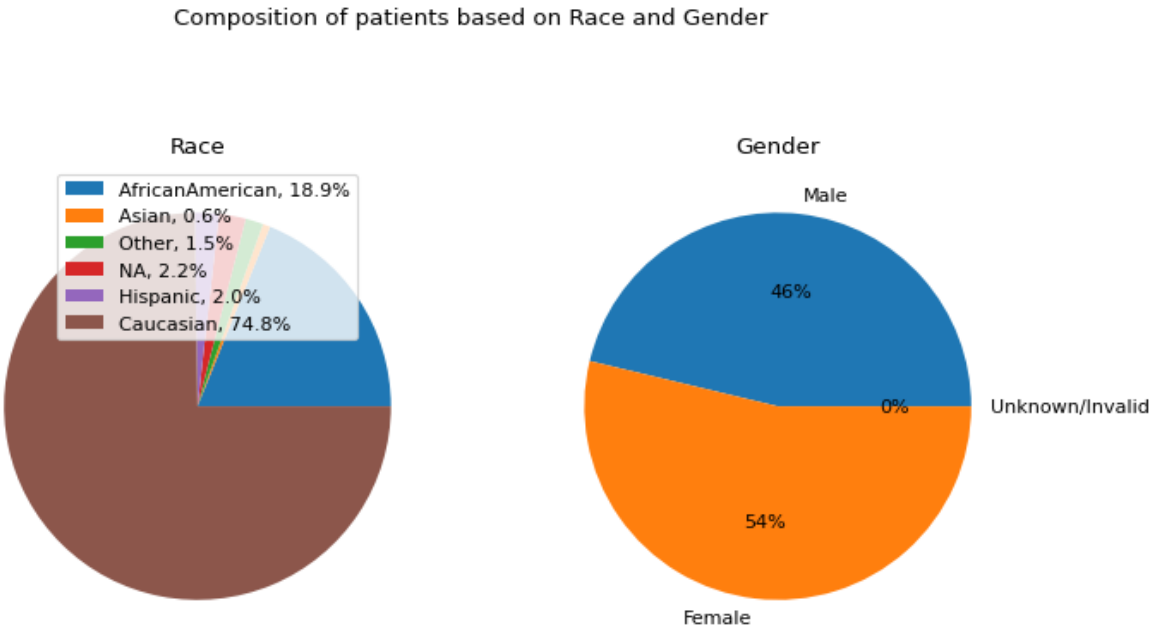
Analysis

#1 Composition of Race and Gender

It is evident from the race composition pie chart that the data is highly biased towards the 'white' people. The second most popular group is African-American while the other ethnicities are scarcely present. However, the data is fairly balanced in terms of gender, with near equal composition from both. With rising awareness of gender neutrality, the gender attribute should soon move away from the current binary system.

In [3]:
```
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import numpy as np

# fetch data specific to race of the patients
race_data = list(df.loc[:, 'race'])
race_dict = dict((x,race_data.count(x)) for x in set(race_data))
race_values = list(race_dict.values())
race_labels = list(race_dict.keys())
race_labels[race_labels.index('?')] = 'NA'

# initialize the figure
figure(figsize=(10, 6), dpi=80)

# calculate some values to make sure the labels are not overlapping each oth
sizes = np.array(race_values)/np.array(sum(race_values))*100
labels = [f'{l}, {s:0.1f}%' for l, s in zip(race_labels, sizes)]

# plotting data on pie chart
plt.subplot(1, 2, 1)
plt.title('Race')
plt.pie(race_values)
plt.legend(bbox_to_anchor=(0.85, 1), loc='upper right', labels=labels)

# fetch data specific to race of the patients
gender_data = list(df.loc[:, 'gender'])
gender_dict = dict((x,gender_data.count(x)) for x in set(gender_data))
gender_values = list(gender_dict.values())
gender_labels = list(gender_dict.keys())

# plotting data on pie chart
plt.subplot(1, 2, 2)
plt.title('Gender')
plt.pie(gender_values, labels=gender_labels, autopct='%.0f%%')
```

```
plt.suptitle('Composition of patients based on Race and Gender')
plt.show()
```



#2 HbA1c level v/s Readmission

A hemoglobin A1C (HbA1C) test is a blood test that shows what your average blood sugar (glucose) level was over the past two to three months. An A1C test can show your average glucose level for the past three months because:

- Glucose sticks to hemoglobin for as long as the red blood cells are alive.
- Red blood cells live about three months.

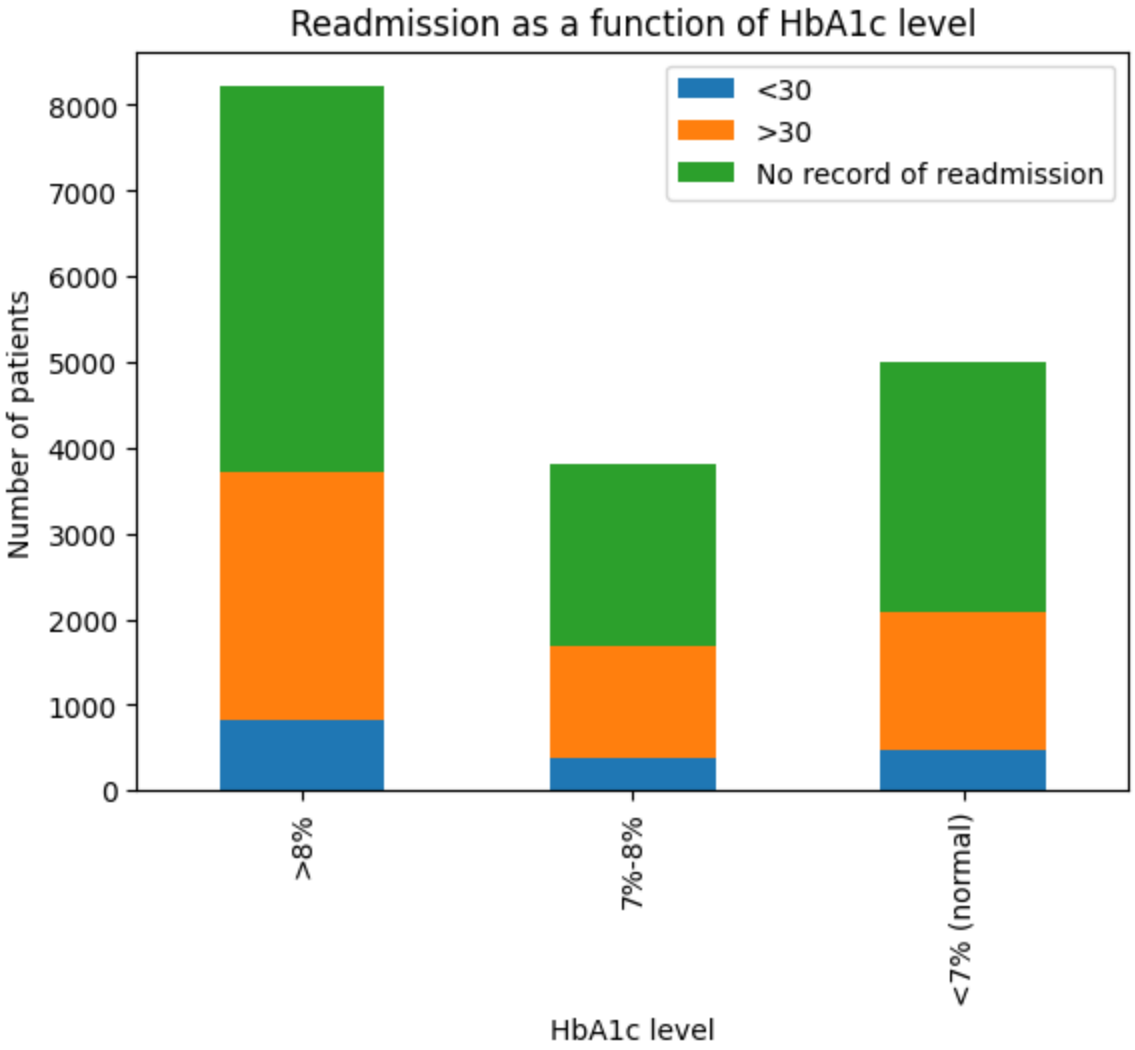
High A1C levels are a sign of high blood glucose from diabetes. It is an important measure of glucose control, which is widely applied to measure performance of diabetes care. A level >8% is considered borderline diabetis while <7% is considered normal. In an ideal situation, high HbA1c level should have more likelihood of readmission into the hospital.

In [4]:
```
# list of data points for patients that were readmitted
df_hb1ac_grt_8  = list(df['readmitted'][df.A1Cresult == '>8'])
df_hb1ac_bw_7_8 = list(df['readmitted'][df.A1Cresult == '>7'])
df_hb1ac_less_7 = list(df['readmitted'][df.A1Cresult == 'Norm'])
df_hb1ac_none   = list(df['readmitted'][df.A1Cresult == 'None'])

print(f'{len(df_hb1ac_none)} patients were not tested for HbA1c level.')

df_readmission_hb1ac = pd.DataFrame({
            '<30': [df_hb1ac_grt_8.count('<30'), df_hb1ac_bw_7_8.count('<30'
            '>30': [df_hb1ac_grt_8.count('>30'), df_hb1ac_bw_7_8.count('>30'
            'No record of readmission': [df_hb1ac_grt_8.count('NO'), df_hb1a
            index=['>8%', '7%-8%', '<7% (normal)'])

df_readmission_hb1ac.plot(kind='bar', stacked=True)
```

```
# # labels for x & y axis
plt.xlabel('HbA1c level')
plt.ylabel('Number of patients')

# title of plot
plt.title('Readmission as a function of HbA1c level')
```

```
84748 patients were not tested for HbA1c level.
```

Out[4]: Text(0.5, 1.0, 'Readmission as a function of HbA1c level')



About 83% of the patients were not tested for HbA1c level upon admission. Out of those tested, the above plots shows the readmission composition in each category of HbA1c level. Nearly 50% of the patients were readmitted for HbA1c level >7% but the readmission percentage is much lower for normal patients.

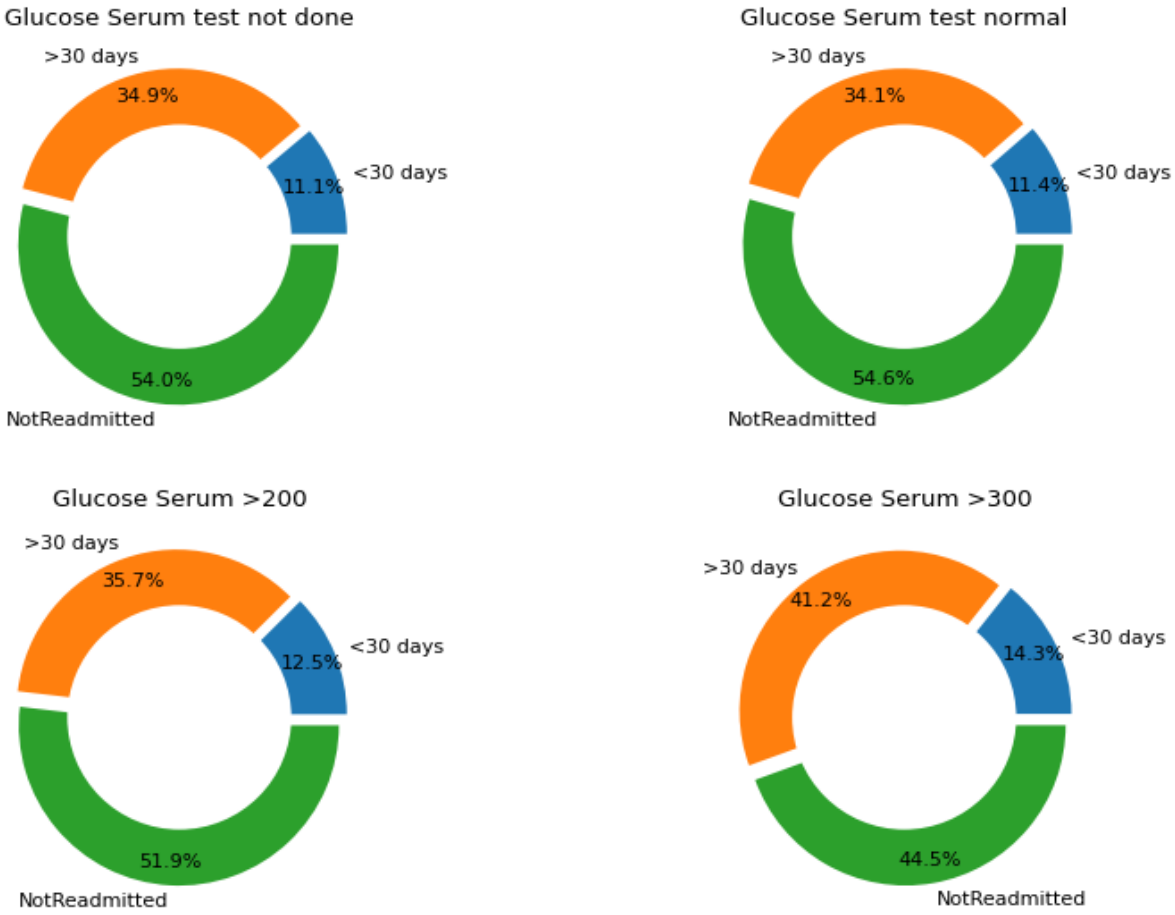
#3 Glucose Serum Test v/s Readmission

Glucose serum test measures the amount of sugar level in the blood and is yet another important metric for diabetes care, thus influencing readmission.

In [19]:
```
readm_categ = ['<30 days', '>30 days', 'NotReadmitted']
```

```
explode     = (0.05, 0.05, 0.05)

# initialize the figure
figure(figsize=(12, 8), dpi=80)

# chart for glucose level not taken
plt.subplot(2, 2, 1)
df_glucose_none  = list(df['readmitted'][df.max_glu_serum == 'None'])
data_glucose_none = [df_glucose_none.count('<30'), df_glucose_none.count('>3
plt.pie(data_glucose_none, labels=readm_categ, autopct='%1.1f%%', pctdistanc
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.title('Glucose Serum test not done')

# chart for glucose level normal
plt.subplot(2, 2, 2)
df_glucose_norm  = list(df['readmitted'][df.max_glu_serum == 'Norm'])
data_glucose_norm = [df_glucose_norm.count('<30'), df_glucose_norm.count('>3
plt.pie(data_glucose_norm, labels=readm_categ, autopct='%1.1f%%', pctdistanc
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.title('Glucose Serum test normal')


# chart for glucose level >200
plt.subplot(2, 2, 3)
df_glucose_grt_200  = list(df['readmitted'][df.max_glu_serum == '>200'])
data_glucose_grt_200 = [df_glucose_grt_200.count('<30'), df_glucose_grt_200.
plt.pie(data_glucose_grt_200, labels=readm_categ, autopct='%1.1f%%', pctdist
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.title('Glucose Serum >200')

# chart for glucose level >300
plt.subplot(2, 2, 4)
df_glucose_grt_300  = list(df['readmitted'][df.max_glu_serum == '>300'])
data_glucose_grt_300 = [df_glucose_grt_300.count('<30'), df_glucose_grt_300.
plt.pie(data_glucose_grt_300, labels=readm_categ, autopct='%1.1f%%', pctdist
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.title('Glucose Serum >300')

print(f'Number of people not tested for glucose serum are {len(df_glucose_nc
print(f'Number of people tested normal for glucose serum are {len(df_glucose
print(f'Number of people tested for glucose serum >200 are {len(df_glucose_g
print(f'Number of people tested for glucose serum >300 are {len(df_glucose_g

# Displaying Chart
plt.show()
```

```
Number of people not tested for glucose serum are 96420.
Number of people tested normal for glucose serum are 2597.
Number of people tested for glucose serum >200 are 1485.
Number of people tested for glucose serum >300 are 1264.
```



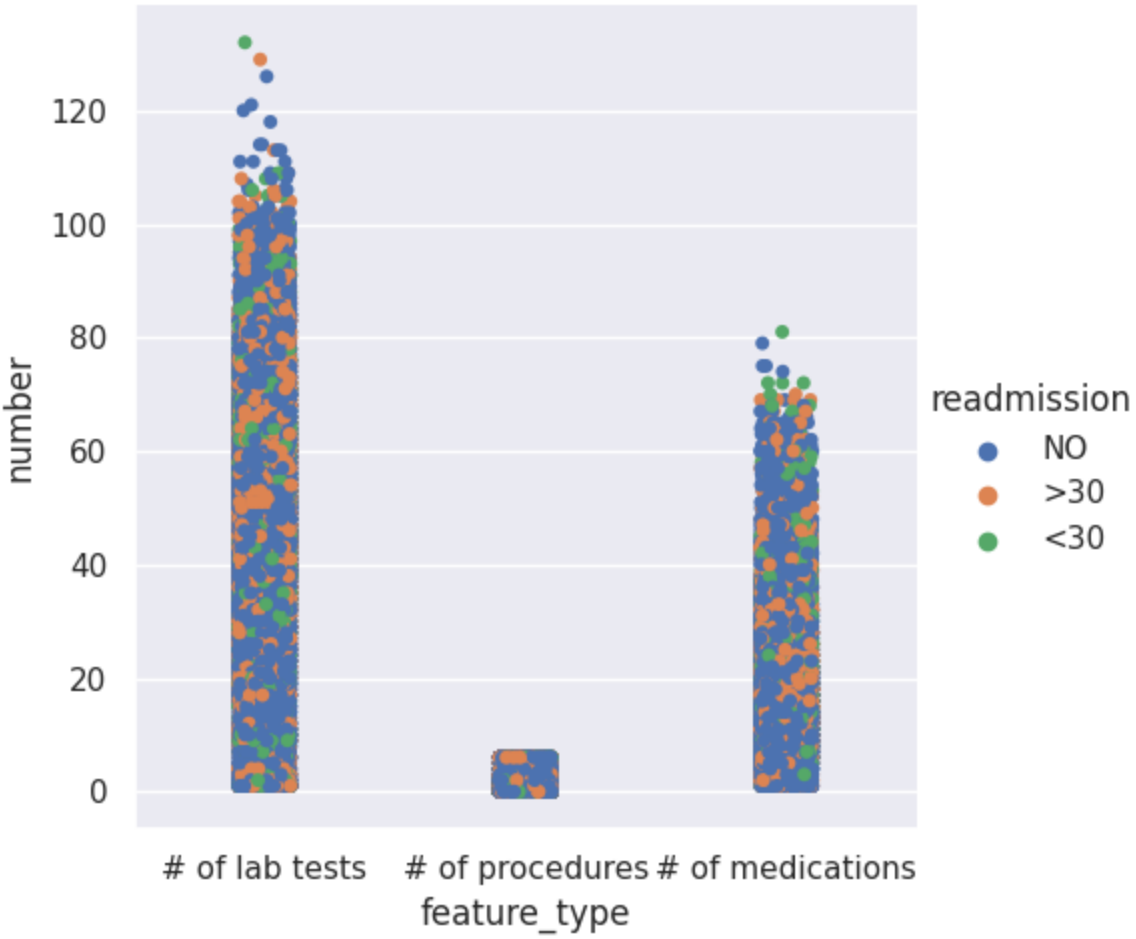
Like the HbA1c test, the glucose serum test was not done for nearly 95% of the patients. Out of those tested, it can be seen that the percentage of readmission increases from normal glucose level (45%) to level >200 (48%) or >300 (55%).

#4 No. of Lab Tests, No. of Procedures, and No. of Medications as a composition of Readmission

In [49]:
```
import seaborn as sns
sns.set_theme()

data_df = pd.DataFrame(columns = ['feature_type', 'number', 'readmission'])

# add number of lab tests data
feature_data  = pd.DataFrame(np.full((len(df), 1), '# of lab tests'), columr
lab_data      = df.loc[:, ['num_lab_procedures', 'readmitted']].rename(colum
lab_data_full = pd.concat([feature_data, lab_data], axis=1)
data_df       = pd.concat([data_df, lab_data_full], ignore_index=True)

# add number of procedures data
feature_data   = pd.DataFrame(np.full((len(df), 1), '# of procedures'), colu
proc_data      = df.loc[:, ['num_procedures', 'readmitted']].rename(columns=
proc_data_full = pd.concat([feature_data, proc_data], axis=1)
```

```
data_df        = pd.concat([data_df, proc_data_full], ignore_index=True)

# add number of medication data
feature_data   = pd.DataFrame(np.full((len(df), 1), '# of medications'), col
med_data       = df.loc[:, ['num_medications', 'readmitted']].rename(columns
med_data_full  = pd.concat([feature_data, med_data], axis=1)
data_df        = pd.concat([data_df, med_data_full], ignore_index=True)

sns.catplot(
    data=data_df, x="feature_type", y="number", hue="readmission",
    native_scale=True, zorder=1
)
```

Out[49]: <seaborn.axisgrid.FacetGrid at 0x7f668701e910>

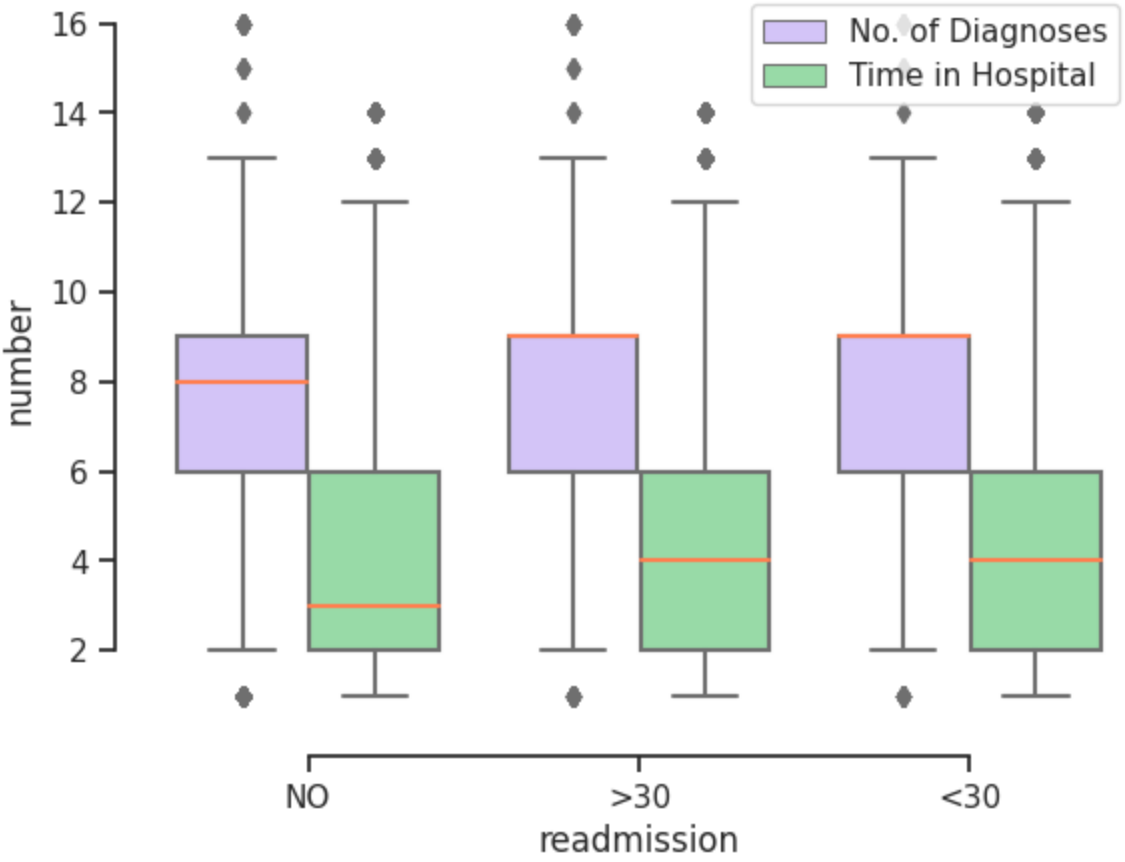


The above plot shows the scatter plots for three attributes: number of lab tests performed during the encounter, number of procedures (other than lab tests) performed during the encounter, and number of distint generic names administered to the patient. Each point is also labelled on type of readmission. It can be see that the number of the tests or medications do not influence the likelihood of readmission, since the points are scattered all over the column. Is it possible that the tests are performed without actually assessing the patient condition?

#5 No. of Diagnoses, and Time in Hospital as a function of Readmission

In [14]:
```
import seaborn as sns
sns.set_theme(style="ticks", palette="pastel")

data_df = pd.DataFrame(columns = ['feature_type', 'number', 'readmission'])

# add number of diagnoses data
feature_data    = pd.DataFrame(np.full((len(df), 1), 'No. of Diagnoses'), co
diagnoses_data  = df.loc[:, ['number_diagnoses', 'readmitted']].rename(colum
diagnoses_data_full = pd.concat([feature_data, diagnoses_data], axis=1)
data_df         = pd.concat([data_df, diagnoses_data_full], ignore_index=Tru

# add time in hospital data
feature_data       = pd.DataFrame(np.full((len(df), 1), 'Time in Hospital'),
time_in_hosp_data  = df.loc[:, ['time_in_hospital', 'readmitted']].rename(co
time_in_hosp_data_full = pd.concat([feature_data, time_in_hosp_data], axis=1
data_df            = pd.concat([data_df, time_in_hosp_data_full], ignore_inc



# Draw a nested boxplot
sns.boxplot(x="readmission", y="number",
            hue="feature_type", palette=["m", "g", "b"],
            data=data_df, medianprops={"color": "coral"})
sns.despine(offset=10, trim=True)
plt.legend(loc='upper right')
```

Out[14]: <matplotlib.legend.Legend at 0x7fb02a8c7d60>



The above box plot compares the number of diagnoses entered into the system for a patient and the time spent in the hospital in days, grouped on the basis of readmission. As expected, the median number of diagnoses for patients with no record of readmission (8) is less than that of those readmitted (9). Similarly, patients who weren't readmitted spent less average time in the hospital (3) as compared to those readmitted (4).

Data Cleaning and Pre-Processing

#1 Remove irrelevant attributes

The dataset in its original form is not ready to be used for training a machine learning model. First, the following two attributes are removed because they are patient specific and do not contribute in modelling patterns:

- Encounter ID
- Patient number

In [6]:
```
print(f'Number of attributes before processing are {len(df.columns)}.')
ml_data = df.drop(columns=['encounter_id', 'patient_nbr'])
print(f'Number of attributes are now {len(ml_data.columns)}.')
```

```
Number of attributes before processing are 50.
Number of attributes are now 48.
```

#2 Remove sparse attributes

Three other features are removed from the dataset because they are sparsely present:

- Weight (97% missing)
- Payer code (52% missing)
- Medical specialty (53% missing)

In [7]:
```
ml_data = ml_data.drop(columns=['weight', 'payer_code', 'medical_specialty'])
print(f'Number of attributes are now {len(ml_data.columns)}.')
```

```
Number of attributes are now 45.
```

#3 Remove rows with missing values

Not all row entries have complete data for all the attributes. Thus, the datapoints with any missing feature value are removed from the dataset.

In [8]:
```
print(f'Number of datapoints before cleaning are {len(ml_data)}.')
na_index_list = []

for index, row in ml_data.iterrows():
    row_data = list(row)
    if '?' in row_data:
        na_index_list.append(index)
```

```
ml_data.drop(na_index_list, inplace=True)
print(f'Number of datapoints after cleaning are {len(ml_data)}.')
```

```
Number of datapoints before cleaning are 101766.
Number of datapoints after cleaning are 98053.
```

#4 Remove additional data that lead to death or hospice

Additionally, all encounters that resulted in either discharge to a hospice, long-term care or patient death are removed, to avoid biasing in the analysis.

In [9]:
```
discharge_id_list = [11, 13, 14, 19, 20, 21, 22, 23]

ml_data = ml_data[ml_data.discharge_disposition_id.isin(discharge_id_list) =

print(f'The final processed dataset has {len(ml_data)} datapoints.')
```

```
The final processed dataset has 93303 datapoints.
```

#5 Convert categorical to numerical data

Most ML libraries cannot work with categorical data, thus, those attributes need to be converted to numerical values.

In [10]:
```
for column in ml_data.columns:
    if ml_data[column].dtype=='object' and column!='readmitted':
        ml_data[column] = ml_data[column].astype('category').cat.codes
```

#6 Turn the problem into binary classification

Since readmission <30 days points are very scare, thus the two labels (<30 and >30) are clubbed into label 'YES'. The problem is now of binary classification of patient readmission: YES and NO.

In [11]:
```
for idx, row in ml_data.iterrows():
    if row['readmitted']=='NO':
        ml_data.loc[idx, 'readmitted'] = 'NotReadmitted'
    else:
        ml_data.loc[idx, 'readmitted'] = 'Readmitted'

ml_data.to_csv('diabetes_data_cleaned_processed.csv', index=False)
```

In [12]:
```
labels = list(ml_data['readmitted'])
re_adm_yes = labels.count('Readmitted')
re_adm_no  = labels.count('NotReadmitted')
print(f'The number of readmissions and no readmission are {re_adm_yes, re_ac
```

```
The number of readmissions and no readmission are (44402, 48901) respectivel
y.
```

Machine Learning

First the data needs to be split into train and test sets. I also specify the random state for reproducibility across experiments.

In [14]:
```
from sklearn.model_selection import train_test_split
import pandas as pd

ml_data = pd.read_csv('diabetes_data_cleaned_processed.csv')
X = ml_data.iloc[:,0:-1]
Y = ml_data.iloc[:,-1]

# split data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, ra
```

#1 Support Vector Machine (SVM)

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. They are easy to implement, memory-efficient, effective in high dimensional spaces, and versatile

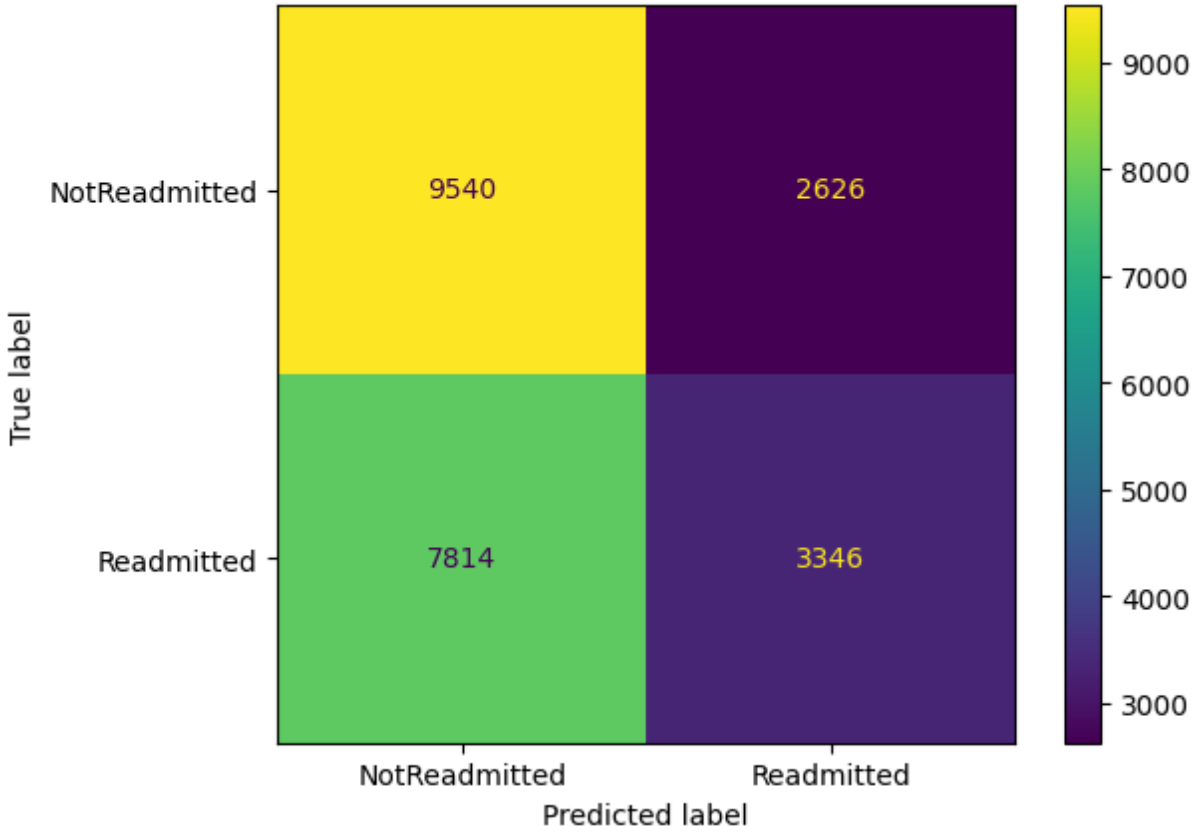
In [ ]:
```
from sklearn import svm
import pickle

svm_clf = svm.SVC()
svm_clf = svm_clf.fit(x_train, y_train)

svm_model_name = 'svm_model.pickle'
pickle.dump(svm_clf, open(svm_model_name, "wb"))
```

In [18]:
```
from sklearn.metrics import accuracy_score

# test the model
svm_model_name = 'svm_model.pickle'
model = pickle.load(open(svm_model_name, "rb"))
y_pred = model.predict(x_test)
y_true = y_test
accuracy = accuracy_score(y_pred, y_true)

print(f'The accuracy of the SVM model on the test set is {round(accuracy*100
```

```
The accuracy of the SVM model on the test set is 55.24%
```

In [28]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

svm_confusion_matrix = confusion_matrix(y_true, y_pred, labels=model.classes
disp = ConfusionMatrixDisplay(confusion_matrix=svm_confusion_matrix, display
disp.plot()
```

Out[28]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f8a696
90430>
```



In [29]:
```
precision = svm_confusion_matrix[1, 1]/(svm_confusion_matrix[1, 1] + svm_cor
recall    = svm_confusion_matrix[1, 1]/(svm_confusion_matrix[1, 1] + svm_cor

print(f'Precision is {round(precision*100,2)}% and recall is {round(recall*1
```

```
Precision is 56.03% and recall is 29.98%
```

Result discussion: The overall accuracy of the model is just 55%, which means that either the model is not able to learn the underlying patterns (underfitting the data) or the data itself is biased or has incorrect entries. Interesting to note that model's recall is significantly low compared to the accuracy or precision, which means that the model is not good at predicting readmission but better otherwise.

#2 Multi-layer Perceptron

A multi-layer perceptron is a fully connected class of feedforward artificial neural network. They can be trained on a variety of tasks (classification, regression) and a variety of input data (numerical, text, images, sound etc.). Theoretically, they are universal function approximators and generally trained using backpropagation. The initialized model has 4 hidden layers with 20, 15, 18, and 15 units respectively.

In [ ]:
```
from sklearn.neural_network import MLPClassifier
import pickle
```

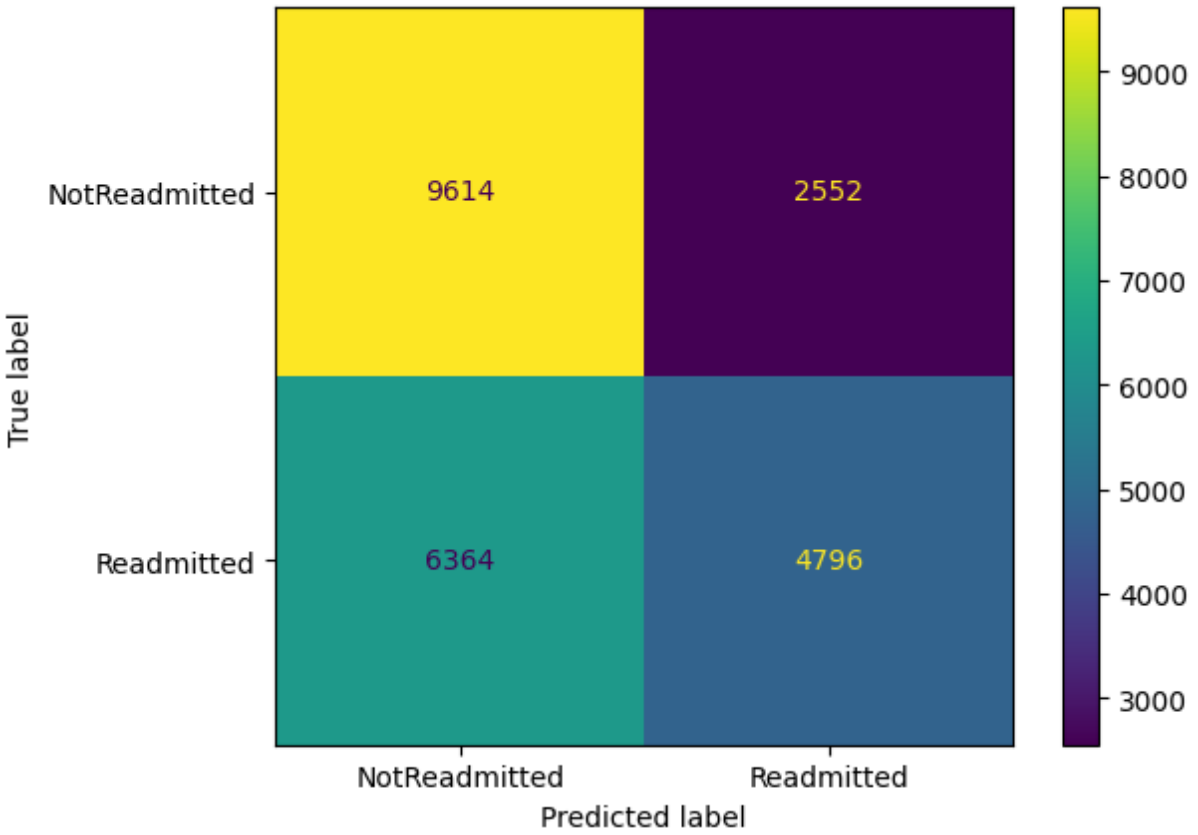
```
mlp_clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(20,
mlp_clf = mlp_clf.fit(x_train, y_train)

mlp_model_name = 'mlp_model.pickle'
pickle.dump(mlp_clf, open(mlp_model_name, "wb"))
```

In [15]:
```
from sklearn.metrics import accuracy_score
import pickle

# test the model
mlp_model_name = 'mlp_model.pickle'
model = pickle.load(open(mlp_model_name, "rb"))
y_pred = model.predict(x_test)
y_true = y_test
accuracy = accuracy_score(y_pred, y_true)

print(f'The accuracy of the MLP model on the test set is {round(accuracy*100
```

```
The accuracy of the MLP model on the test set is 61.78%
```

In [16]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

mlp_confusion_matrix = confusion_matrix(y_true, y_pred, labels=model.classes
disp = ConfusionMatrixDisplay(confusion_matrix=mlp_confusion_matrix, display
disp.plot()
```

Out[16]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fe8055
72340>
```



In [17]:
```
precision = mlp_confusion_matrix[1, 1]/(mlp_confusion_matrix[1, 1] + mlp_cor
```

```
recall    = mlp_confusion_matrix[1, 1]/(mlp_confusion_matrix[1, 1] + mlp_cor

print(f'Precision is {round(precision*100,2)}% and recall is {round(recall*1
```

```
Precision is 65.27% and recall is 42.97%
```

Result discussion: As expected, MLP has a higher accuracy, precision, and recall compared to SVM. Also, MLP's recall is significantly higher than that of SVM compared to other metrics.

In [ ]: