Hospital Readmission Case Study

Introduction

Hospital readmissions within 30 days of discharge are a significant concern for healthcare systems, as they often indicate a gap in patient care and contribute to increased healthcare costs. The objective of this project is to develop a predictive model that can identify patients at high risk of readmission. Such a tool can enable hospitals to implement targeted interventions, like enhanced post-discharge follow-up, to reduce readmission rates.

Dataset information

The dataset used is a simplified version of a dataset sourced from the UCI Machine Learning repository. UCI Repository Link: https://archive.ics.uci.edu/dataset/296/diabetes+130-us+hospitals+for+years+1999-2008

Kaggle Link: https://www.kaggle.com/datasets/dubradave/hospital-readmissions/data

Data Source and Acknowledgment

- Data was sourced from the UCI Machine Learning Repository and Kaggle.
- All patient data has been anonymized to ensure privacy and compliance with ethical data usage practices.

Data Dictionary

- "age" - age bracket of the patient
- "time_in_hospital" - days (from 1 to 14)
- "n_procedures" - number of procedures performed during the hospital stay
- "n_lab_procedures" - number of laboratory procedures performed during the hospital stay
- "n_medications" - number of medications administered during the hospital stay
- "n_outpatient" - number of outpatient visits in the year before a hospital stay
- "n_inpatient" - number of inpatient visits in the year before the hospital stay
- "n_emergency" - number of visits to the emergency room in the year before the hospital stay
- "medical_specialty" - the specialty of the admitting physician
- "diag_1" - primary diagnosis (Circulatory, Respiratory, Digestive, etc.)
- "diag_2" - secondary diagnosis
- "diag_3" - additional secondary diagnosis
- "glucose_test" - whether the glucose serum came out as high (> 200), normal, or not performed
- "A1Ctest" - whether the A1C level of the patient came out as high (> 7%), normal, or not performed
- "change" - whether there was a change in the diabetes medication ('yes' or 'no')
- "diabetes_med" - whether a diabetes medication was prescribed ('yes' or 'no')
- "readmitted" - if the patient was readmitted at the hospital ('yes' or 'no')

Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

import math
from scipy.stats import uniform

pd.options.display.max_colwidth = 100
pd.set_option('display.max_columns', None)

from numpy.random import seed
seed(42)

from sklearn.model_selection import train_test_split, RandomizedSearchCV, \
    GridSearchCV, cross_val_score, StratifiedKFold, KFold
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier, \
    GradientBoostingRegressor, RandomForestRegressor, AdaBoostRegressor, IsolationForest
from sklearn.svm import OneClassSVM
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,\
    recall_score, f1_score, roc_auc_score, ConfusionMatrixDisplay, roc_curve, auc, classification_report, \
    mean_absolute_error, r2_score, mean_squared_error
```

Data

In [2]:
```
df = pd.read_csv('../Data/hospital_readmissions.csv')
df.head()
```

Out[2]:

	age	time_in_hospital	n_lab_procedures	n_procedures	n_medications	n_outpatient	n_inpatient	n_emergency	medical_specialty	diag_1	diag_2	diag_3	g
0	[70-80)	8	72	1	18	2	0	0	Missing	Circulatory	Respiratory	Other	
1	[70-80)	3	34	2	13	0	0	0	Other	Other	Other	Other	
2	[50-60)	5	45	0	18	0	0	0	Missing	Circulatory	Circulatory	Circulatory	
3	[70-80)	2	36	0	12	1	0	0	Missing	Circulatory	Other	Diabetes	
4	[60-70)	1	42	0	7	0	0	0	InternalMedicine	Other	Circulatory	Respiratory	

In [3]:
```
# formating age
df['age'] = df['age'].str.replace('[','(',regex=False)
df.head()
```

Out[3]:

	age	time_in_hospital	n_lab_procedures	n_procedures	n_medications	n_outpatient	n_inpatient	n_emergency	medical_specialty	diag_1	diag_2	diag_3	g
0	(70-80)	8	72	1	18	2	0	0	Missing	Circulatory	Respiratory	Other	
1	(70-80)	3	34	2	13	0	0	0	Other	Other	Other	Other	
2	(50-60)	5	45	0	18	0	0	0	Missing	Circulatory	Circulatory	Circulatory	
3	(70-80)	2	36	0	12	1	0	0	Missing	Circulatory	Other	Diabetes	
4	(60-70)	1	42	0	7	0	0	0	InternalMedicine	Other	Circulatory	Respiratory	

Exploratory Data Analysis

In [71]:
```
df.shape
```

Out[71]:
```
(25000, 17)
```

In [72]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   age                25000 non-null  object
 1   time_in_hospital   25000 non-null  int64 
 2   n_lab_procedures   25000 non-null  int64 
 3   n_procedures       25000 non-null  int64 
 4   n_medications      25000 non-null  int64 
 5   n_outpatient       25000 non-null  int64 
 6   n_inpatient        25000 non-null  int64 
 7   n_emergency        25000 non-null  int64 
 8   medical_specialty  25000 non-null  object
 9   diag_1             25000 non-null  object
 10  diag_2             25000 non-null  object
 11  diag_3             25000 non-null  object
 12  glucose_test       25000 non-null  object
 13  A1Ctest            25000 non-null  object
 14  change             25000 non-null  object
 15  diabetes_med       25000 non-null  object
 16  readmitted         25000 non-null  object
dtypes: int64(7), object(10)
memory usage: 3.2+ MB
```

In [73]:
```
df.isnull().sum()
```

Out[73]:
```
age                 0
time_in_hospital    0
n_lab_procedures    0
n_procedures        0
n_medications       0
n_outpatient        0
n_inpatient         0
n_emergency         0
medical_specialty   0
diag_1              0
diag_2              0
diag_3              0
glucose_test        0
A1Ctest             0
change              0
diabetes_med        0
readmitted          0
dtype: int64
```

In [4]:
```
# Numerical
numerical_features = df.select_dtypes(exclude='object').columns
numerical_features
```

Out[4]:
```
Index(['time_in_hospital', 'n_lab_procedures', 'n_procedures', 'n_medications',
       'n_outpatient', 'n_inpatient', 'n_emergency'],
      dtype='object')
```

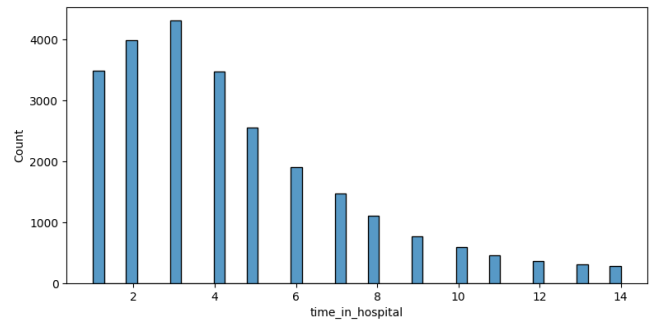
In [75]:
```
df.describe()
```

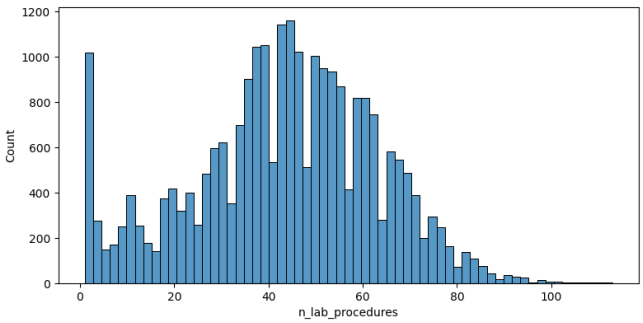
Out[75]:

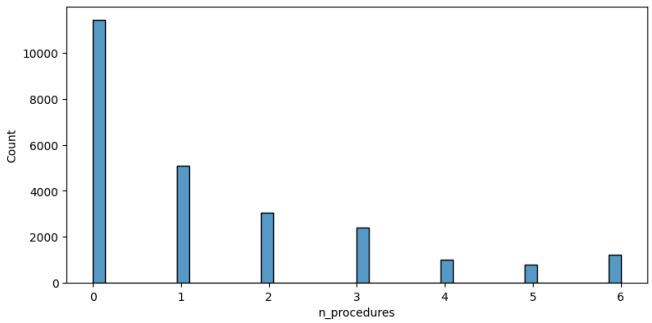
	time_in_hospital	n_lab_procedures	n_procedures	n_medications	n_outpatient	n_inpatient	n_emergency
count	25000.00000	25000.00000	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000
mean	4.45332	43.24076	1.352360	16.252400	0.366400	0.615960	0.186600
std	3.00147	19.81862	1.715179	8.060532	1.195478	1.177951	0.885873
min	1.00000	1.00000	0.000000	1.000000	0.000000	0.000000	0.000000
25%	2.00000	31.00000	0.000000	11.000000	0.000000	0.000000	0.000000
50%	4.00000	44.00000	1.000000	15.000000	0.000000	0.000000	0.000000
75%	6.00000	57.00000	2.000000	20.000000	0.000000	1.000000	0.000000
max	14.00000	113.00000	6.000000	79.000000	33.000000	15.000000	64.000000

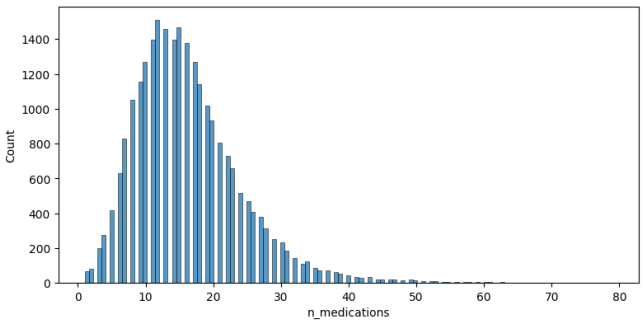
In [76]:
```
plt.figure(figsize = (20, 20))

plt.subplot(4, 2, 1)
sns.histplot(x = df['time_in_hospital'], kde = False)

plt.subplot(4, 2, 2)
sns.histplot(x = df['n_lab_procedures'], kde = False)

plt.subplot(4, 2, 3)
sns.histplot(x = df['n_procedures'], kde = False)

plt.subplot(4, 2, 4)
sns.histplot(x = df['n_medications'], kde = False)

plt.subplot(4, 2, 5)
sns.histplot(x = df['n_outpatient'], kde = False)

plt.subplot(4, 2, 6)
sns.histplot(x = df['n_inpatient'], kde = False)

plt.subplot(4, 2, 7)
sns.histplot(x = df['n_emergency'], kde = False)
```

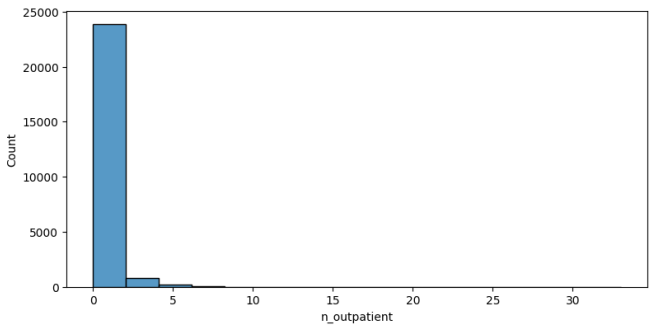
Out[76]:
```
<Axes: xlabel='n_emergency', ylabel='Count'>
```

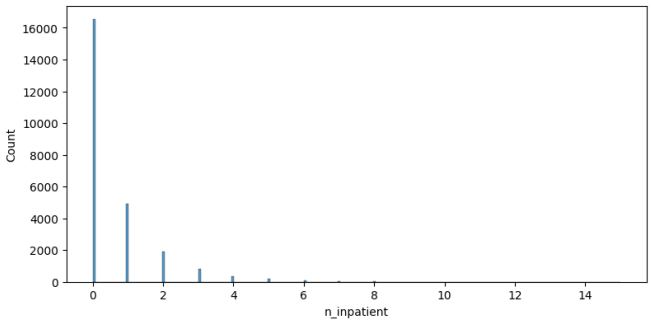


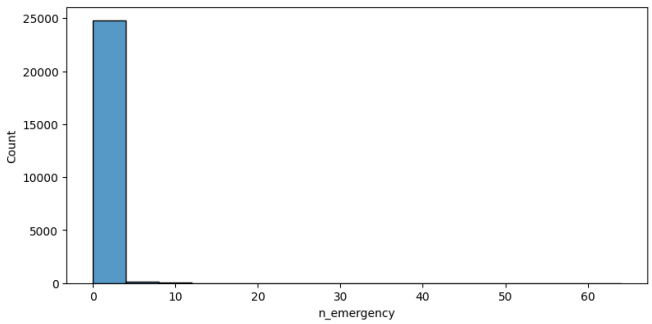




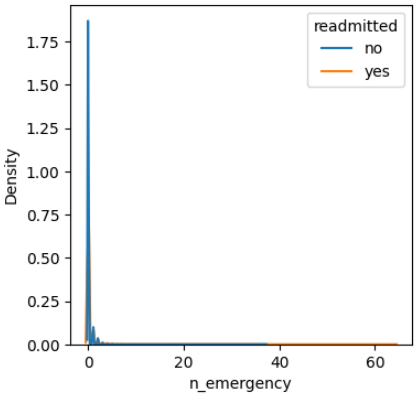


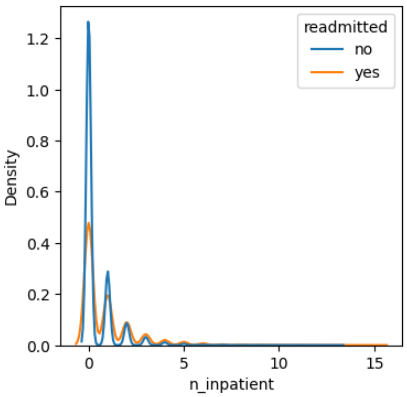


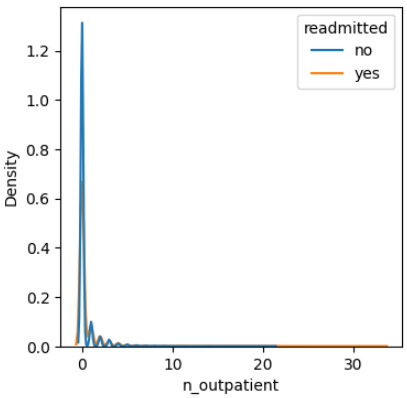


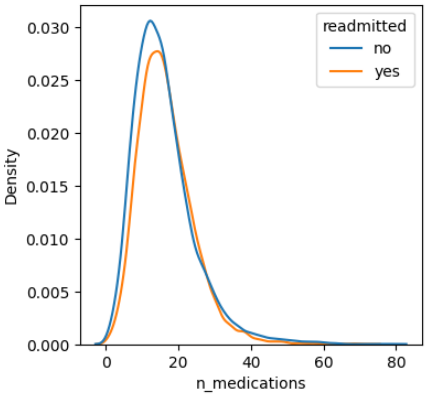


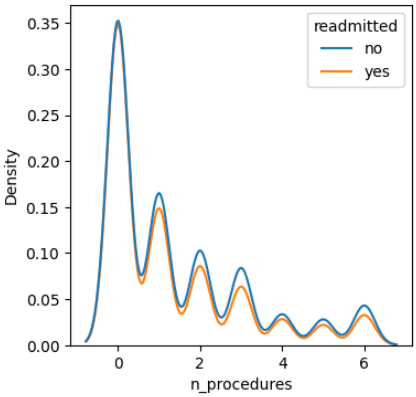
In [84]:
```
# KDE plots
num_feat = numerical_features[::-1]
for feature in num_feat:
    plt.figure(figsize=(4,4))
    sns.kdeplot(data=df, x=feature, hue='readmitted')
    plt.show()
```

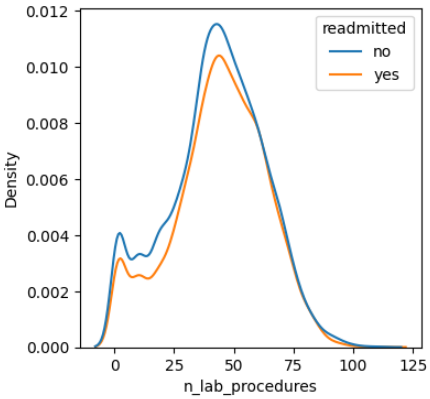

readmitted
no
yes
Density
n_emergency

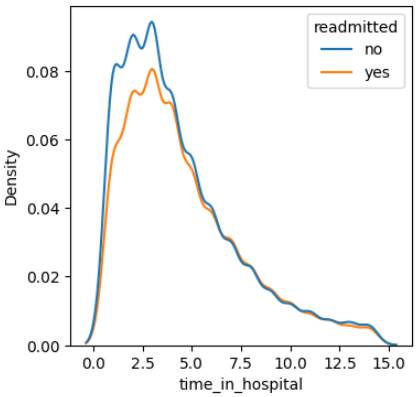

readmitted
no
yes
Density
n_inpatient


readmitted
no
yes
Density
n_outpatient


readmitted
no
yes
Density
n_medications


readmitted
no
yes
Density
n_procedures





Several attributes are positively skewed and need to be transformed.

In [85]:
```
df['n_inpatient'].value_counts()
```

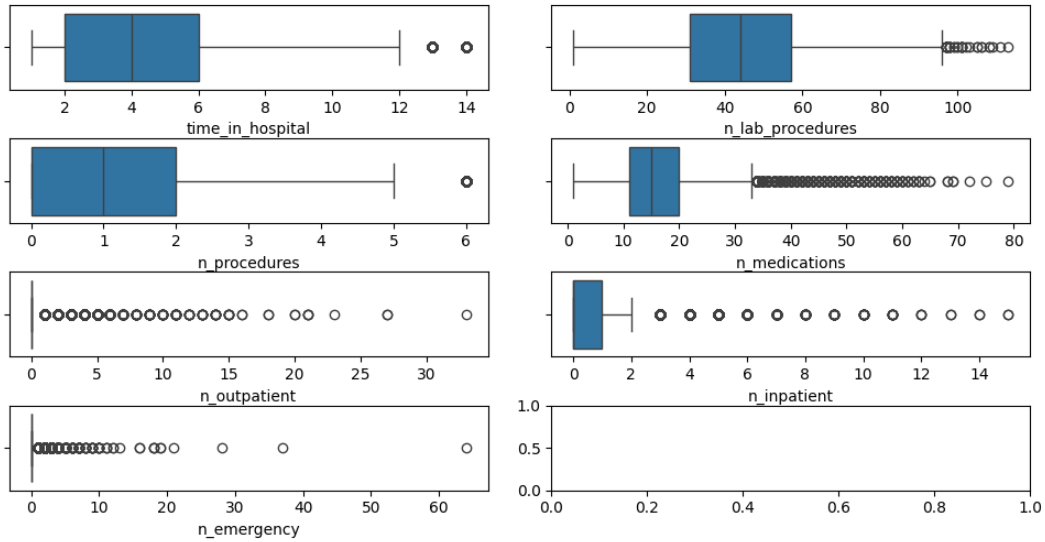
Out[85]:
```
n_inpatient
0     16537
1      4926
2      1909
3       833
4       358
5       211
6       104
7        47
8        26
9        20
10       12
11        8
12        3
14        2
15        2
13        2
Name: count, dtype: int64
```

In [86]:
```
df['n_emergency'].value_counts()
```

Out[86]:
```
n_emergency
0     22272
1      1842
2       525
3       167
4        83
5        40
7        18
6        18
9         6
8         6
10        6
18        3
11        3
12        2
16        2
19        2
28        1
37        1
21        1
13        1
64        1
Name: count, dtype: int64
```

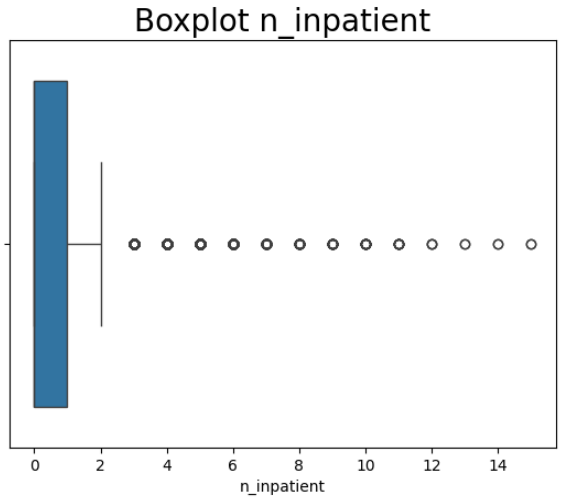
In [87]:
```
plt.subplots_adjust(hspace=0.3, wspace = 0.7)
fig, axs = plt.subplots(4, 2, figsize=(10, 5))
fig.tight_layout()

sns.boxplot(data = df, x = 'time_in_hospital', ax = axs[0, 0])
sns.boxplot(data = df, x = 'n_lab_procedures', ax = axs[0, 1])
sns.boxplot(data = df, x = 'n_procedures', ax = axs[1, 0])
sns.boxplot(data = df, x = 'n_medications', ax = axs[1, 1])
sns.boxplot(data = df, x = 'n_outpatient', ax = axs[2, 0])
sns.boxplot(data = df, x = 'n_inpatient', ax = axs[2, 1])
sns.boxplot(data = df, x = 'n_emergency', ax = axs[3, 0])

plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

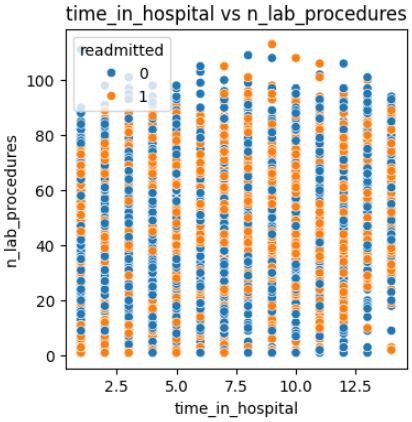


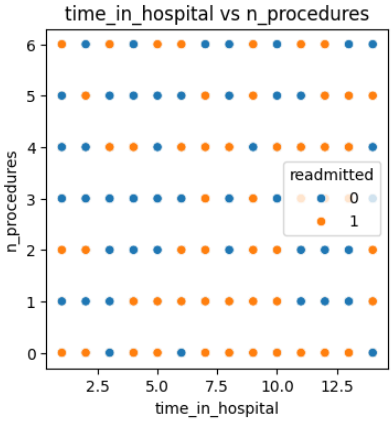
In [88]:
```
plt.title("Boxplot n_inpatient", fontdict = {'fontsize': 20})
sns.boxplot(x=df["n_inpatient"])
```

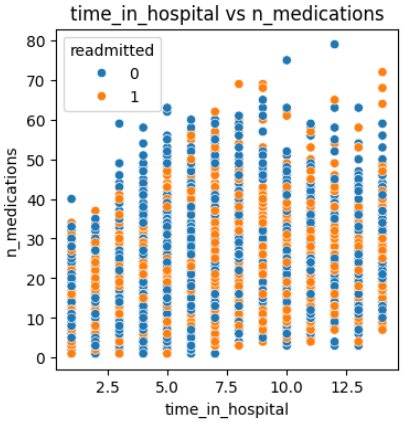
Out[88]: <Axes: title={'center': 'Boxplot n_inpatient'}, xlabel='n_inpatient'>

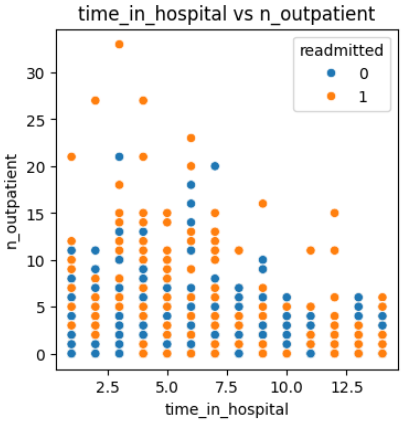


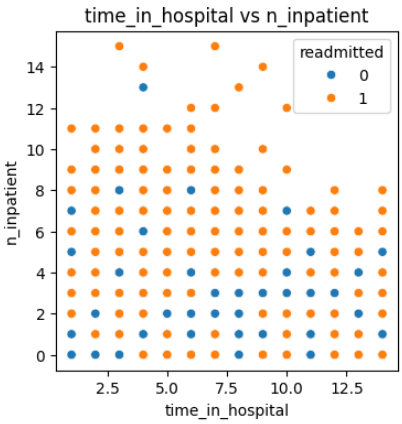
In [113...
```
# scatter plot

def check_pair(pair, pair_dict):
    f1, f2 = pair
    # If f1 already has f2 recorded or vice versa → duplicate
    if f1 in pair_dict and f2 in pair_dict[f1]:
        return False
    if f2 in pair_dict and f1 in pair_dict[f2]:
        return False
    return True

def append_pair(pair, pair_dict):
    f1, f2 = pair
    pair_dict.setdefault(f1, []).append(f2)
    pair_dict.setdefault(f2, []).append(f1)

pair_dict = dict()

for f1 in numerical_features:
  for f2 in numerical_features:
    if f1 == f2:
      continue
    else:
      pair = (f1, f2)
      if check_pair(pair, pair_dict):
        append_pair(pair, pair_dict)
        plt.figure(figsize=(4,4))
        sns.scatterplot(data=df, x=f1, y=f2, hue='readmitted')
        plt.title(f'{f1} vs {f2}')
        plt.show()
```

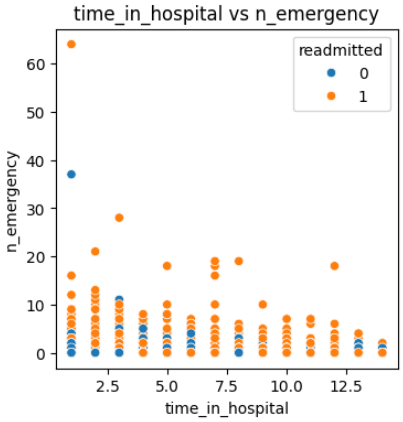

time_in_hospital vs n_lab_procedures
readmitted
0
1
n_lab_procedures
time_in_hospital

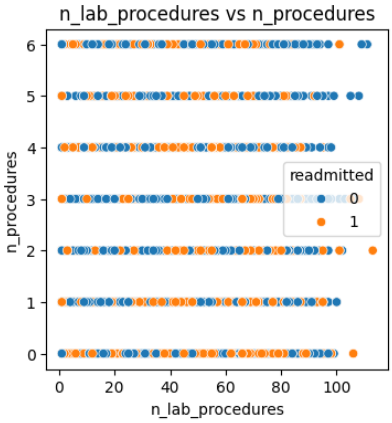

time_in_hospital vs n_procedures
readmitted
0
1
n_procedures
time_in_hospital

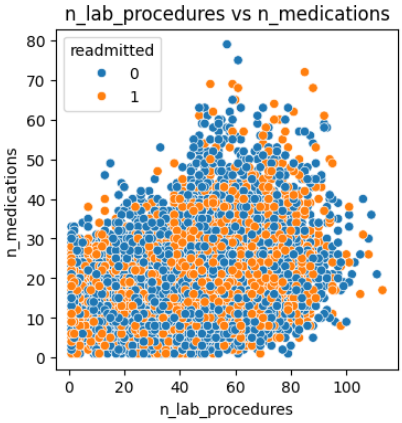

time_in_hospital vs n_medications
readmitted
0
1
n_medications
time_in_hospital

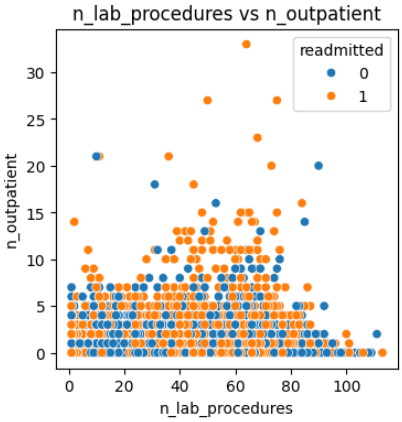

time_in_hospital vs n_outpatient
readmitted
0
1
n_outpatient
time_in_hospital

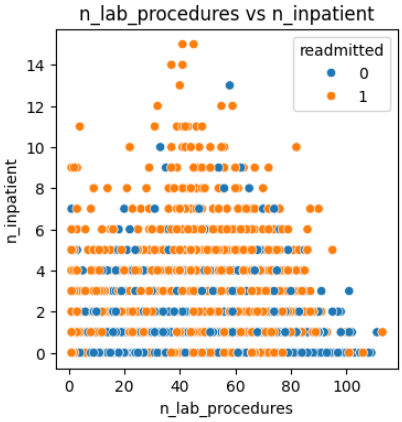

time_in_hospital vs n_inpatient
readmitted
0
1
n_inpatient
time_in_hospital

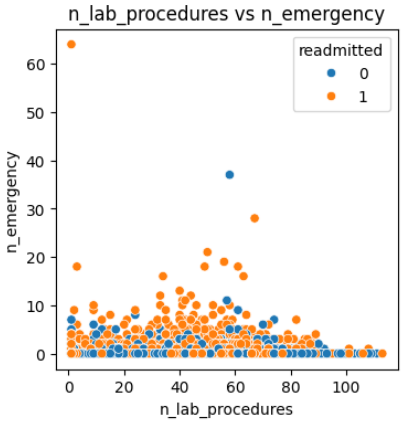

time_in_hospital vs n_emergency
readmitted
0
1
n_emergency
time_in_hospital

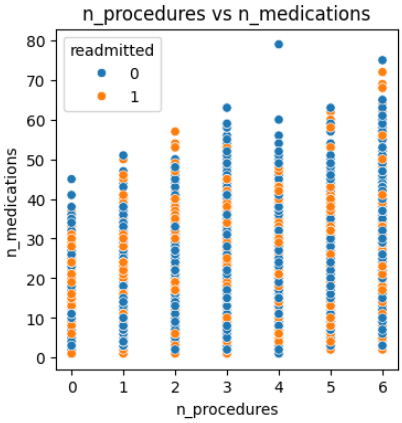

n_lab_procedures vs n_procedures
readmitted
0
1
n_procedures
n_lab_procedures

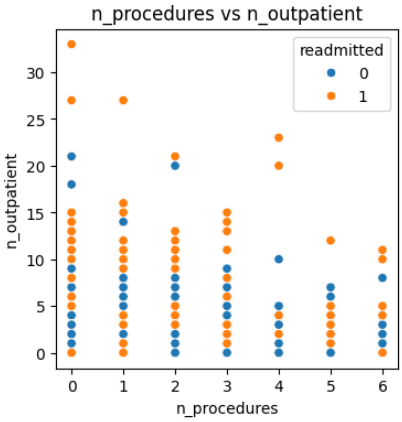

n_lab_procedures vs n_medications
readmitted
0
1
n_medications
n_lab_procedures

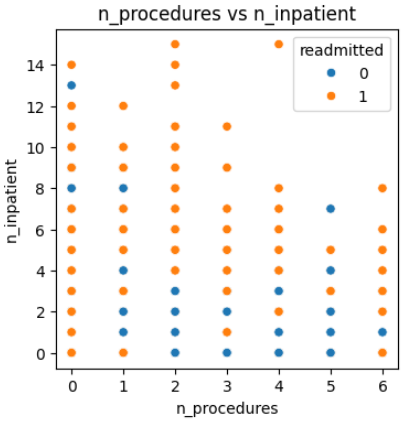

n_lab_procedures vs n_outpatient
readmitted
0
1
n_outpatient
n_lab_procedures

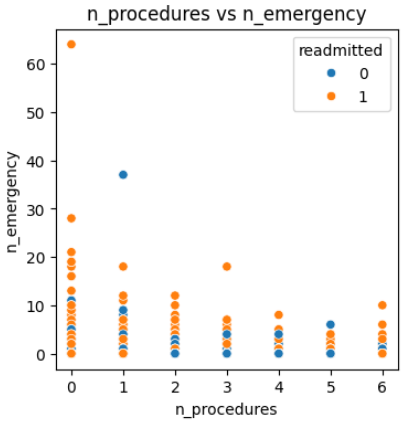

n_lab_procedures vs n_inpatient
readmitted
0
1
n_inpatient
n_lab_procedures

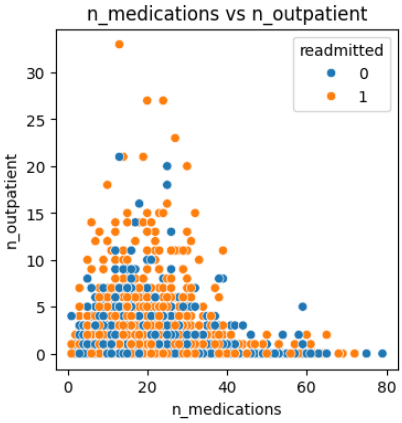

n_lab_procedures vs n_emergency
readmitted
0
1
n_emergency
n_lab_procedures

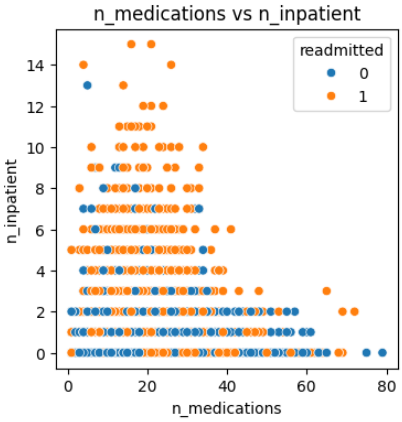

n_procedures vs n_medications
readmitted
0
1
n_medications
n_procedures

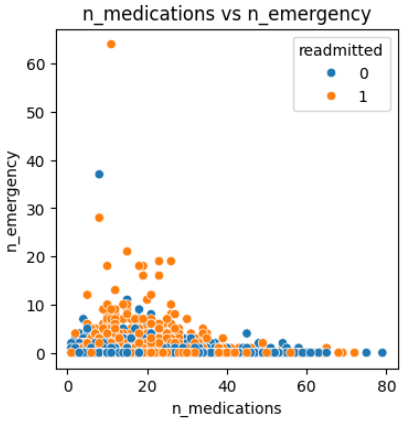

n_procedures vs n_outpatient
readmitted
0
1
n_outpatient
n_procedures

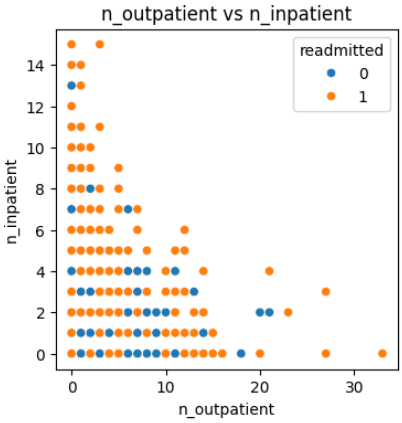

n_procedures vs n_inpatient
readmitted
0
1
n_inpatient
n_procedures

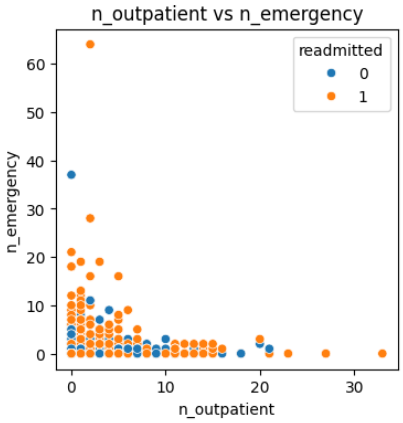

n_procedures vs n_emergency
readmitted
0
1
n_emergency
n_procedures

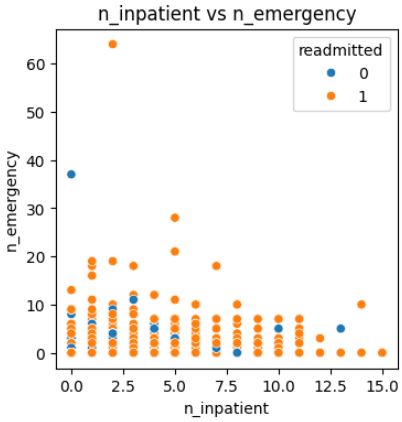

n_medications vs n_outpatient
readmitted
0
1
n_outpatient
n_medications


n_medications vs n_inpatient
readmitted
0
1
n_inpatient
n_medications
0
2
4
6
8
10
12
14
0
20
40
60
80


n_medications vs n_emergency
readmitted
0
1
n_emergency
n_medications
0
10
20
30
40
50
60
0
20
40
60
80

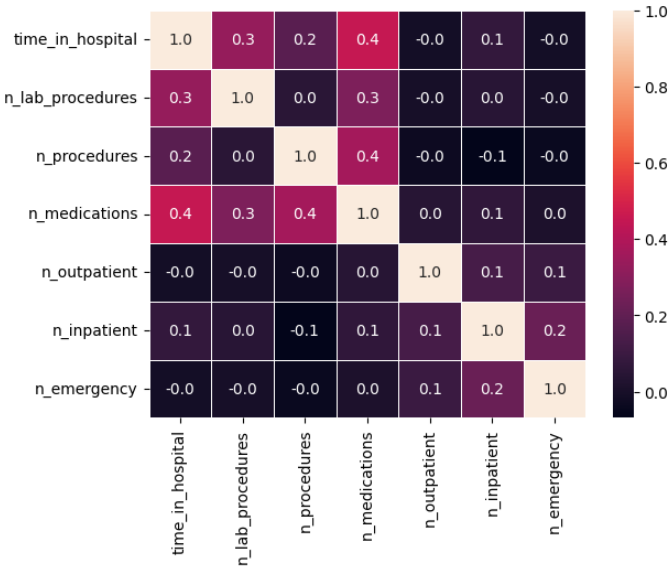

n_outpatient vs n_inpatient
readmitted
0
1
n_inpatient
n_outpatient
0
2
4
6
8
10
12
14
0
10
20
30


n_outpatient vs n_emergency
readmitted
0
1
n_emergency
n_outpatient
0
10
20
30
40
50
60
0
10
20
30



In [89]:
```
sns.heatmap(df[numerical_features].corr(), annot=True, linewidth=.5, fmt=".1f")
```

Out[89]: <Axes: >



In [ ]:
```
# categorical
df.select_dtypes(include = 'object').describe()
```

Out[ ]:

	age	medical_specialty	diag_1	diag_2	diag_3	glucose_test	A1Ctest	change	diabetes_med	readmitted
count	25000	25000	25000	25000	25000	25000	25000	25000	25000	25000
unique	6	7	8	8	8	3	3	2	2	2
top	(70-80)	Missing	Circulatory	Other	Other	no	no	no	yes	no
freq	6837	12382	7824	9056	9107	23625	20938	13497	19228	13246

In [91]:
```
df.select_dtypes(include='object').columns
```

Out[91]:
```
Index(['age', 'medical_specialty', 'diag_1', 'diag_2', 'diag_3',
       'glucose_test', 'A1Ctest', 'change', 'diabetes_med', 'readmitted'],
      dtype='object')
```

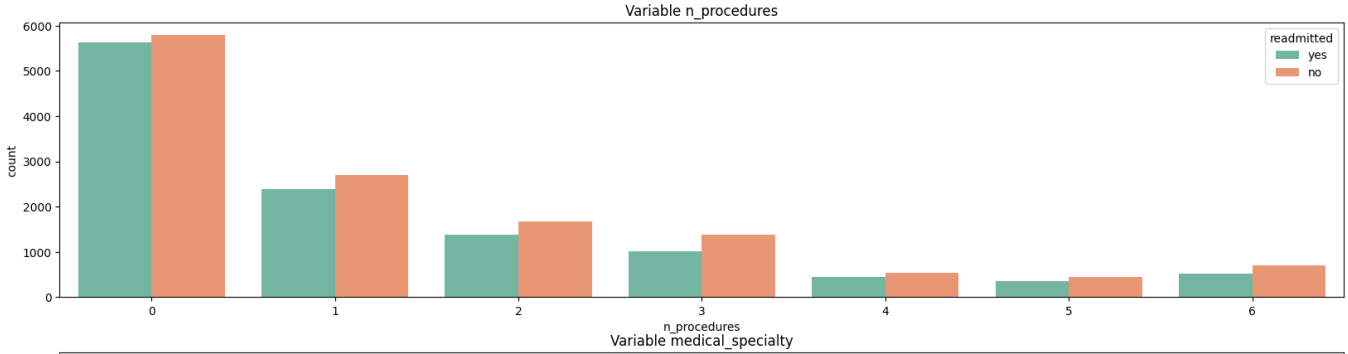
In [92]:
```
categorical_features = df.select_dtypes(include='object').columns
for i in categorical_features:
    print(i,':',df[i].unique())
```

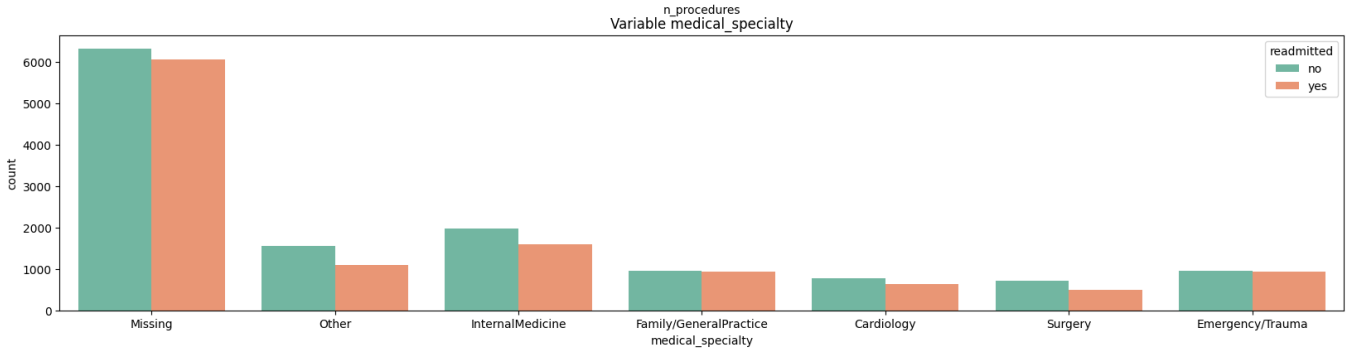
```
age : ['(70-80)' '(50-60)' '(60-70)' '(40-50)' '(80-90)' '(90-100)']
medical_specialty : ['Missing' 'Other' 'InternalMedicine' 'Family/GeneralPractice'
 'Cardiology' 'Surgery' 'Emergency/Trauma']
diag_1 : ['Circulatory' 'Other' 'Injury' 'Digestive' 'Respiratory' 'Diabetes'
 'Musculoskeletal' 'Missing']
diag_2 : ['Respiratory' 'Other' 'Circulatory' 'Injury' 'Diabetes' 'Digestive'
 'Musculoskeletal' 'Missing']
diag_3 : ['Other' 'Circulatory' 'Diabetes' 'Respiratory' 'Injury' 'Musculoskeletal'
 'Digestive' 'Missing']
glucose_test : ['no' 'normal' 'high']
A1Ctest : ['no' 'normal' 'high']
change : ['no' 'yes']
diabetes_med : ['yes' 'no']
readmitted : ['no' 'yes']
```

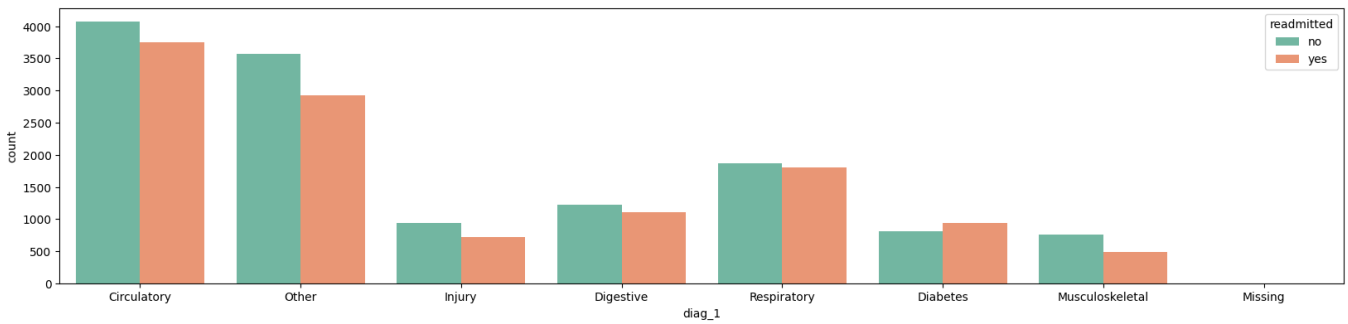
In [93]:
```
plt.figure(figsize = (20, 25))
plt.suptitle("Analysis Of Variable readmitted",fontweight="bold", fontsize=20)

plt.subplot(5, 1, 1)
plt.gca().set_title('Variable n_procedures')
sns.countplot(x = 'n_procedures', hue = 'readmitted', palette = 'Set2', data = df)

plt.subplot(5, 1, 2)
plt.gca().set_title('Variable medical_specialty')
sns.countplot(x = 'medical_specialty', hue = 'readmitted', palette = 'Set2', data = df)

plt.subplot(5, 1, 3)
sns.countplot(x = 'diag_1', hue = 'readmitted', palette = 'Set2', data = df)

plt.subplot(5, 1, 4)
sns.countplot(x = 'diag_2', hue = 'readmitted', palette = 'Set2', data = df)

plt.subplot(5, 1, 5)
sns.countplot(x = 'diag_3', hue = 'readmitted', palette = 'Set2', data = df)
```

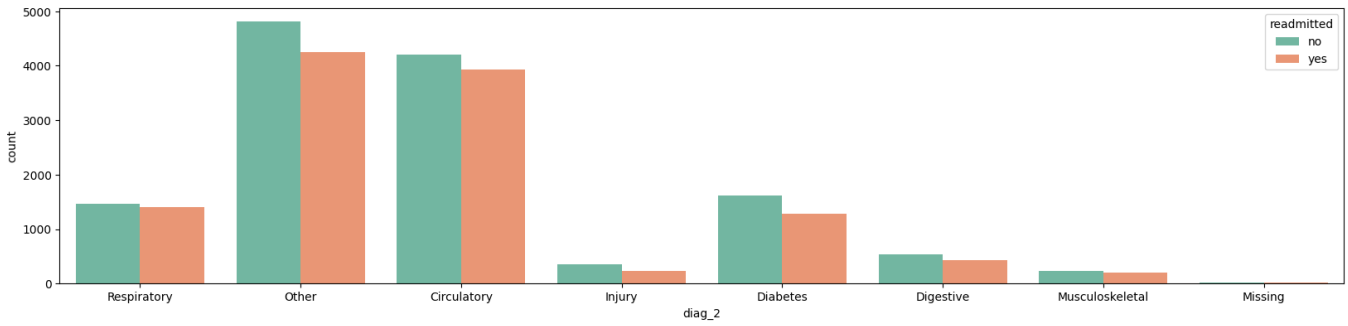
Out[93]:
```
<Axes: xlabel='diag_3', ylabel='count'>
```

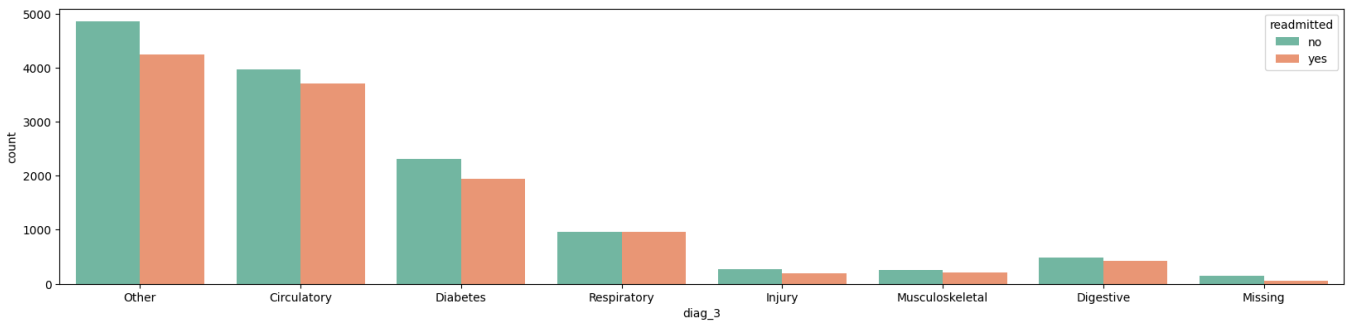
Analysis Of Variable readmitted







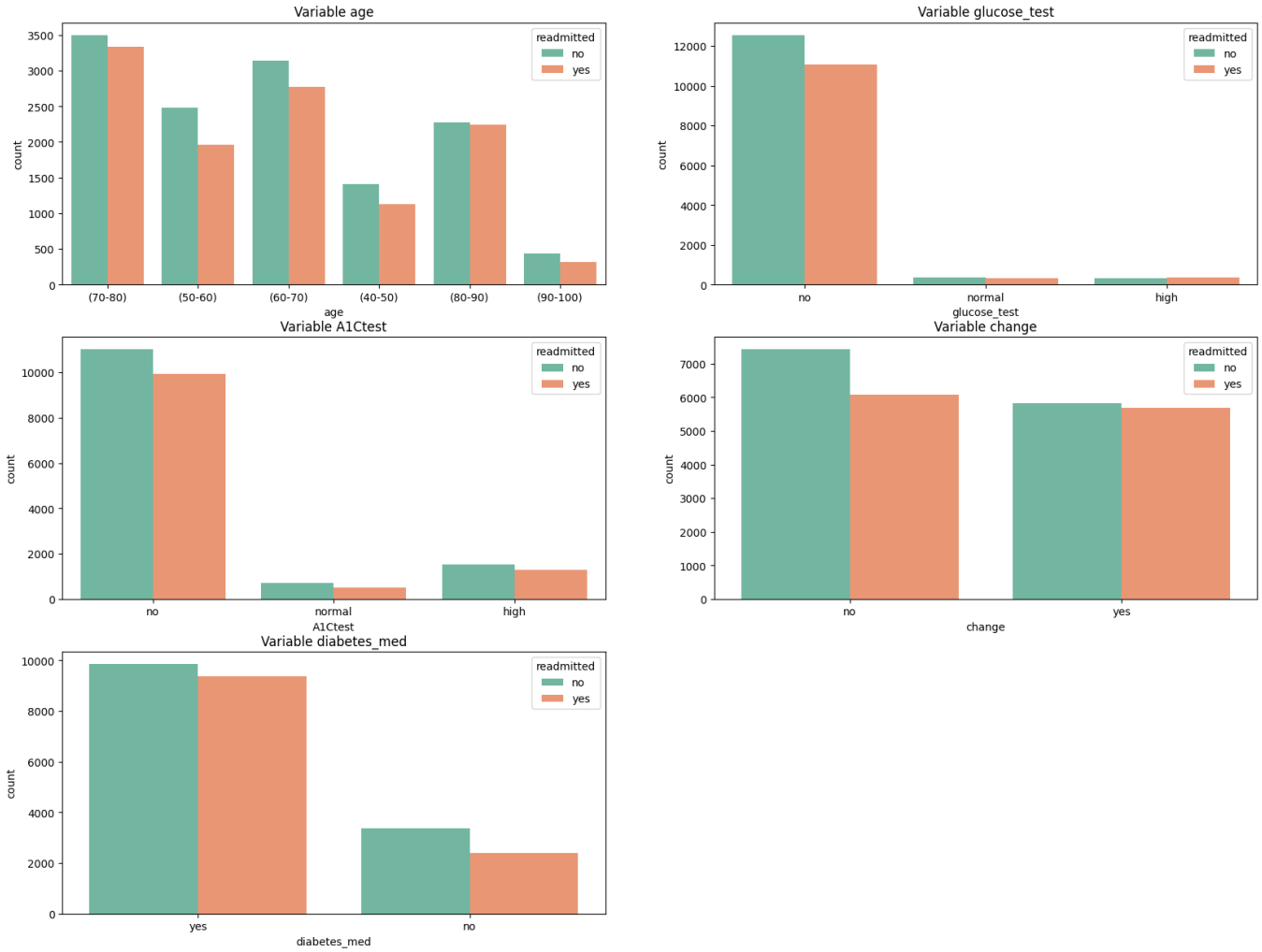




In [94]:
```
plt.figure(figsize = (20, 15))

plt.subplot(3, 2, 1)
plt.gca().set_title('Variable age')
sns.countplot(x = 'age', hue = 'readmitted', palette = 'Set2', data = df)

plt.subplot(3, 2, 2)
plt.gca().set_title('Variable glucose_test')
sns.countplot(x = 'glucose_test', hue = 'readmitted', palette = 'Set2', data = df)

plt.subplot(3, 2, 3)
plt.gca().set_title('Variable A1Ctest')
sns.countplot(x = 'A1Ctest', hue = 'readmitted', palette = 'Set2', data = df)

plt.subplot(3, 2, 4)
```

```
plt.gca().set_title('Variable change')
sns.countplot(x = 'change', hue = 'readmitted', palette = 'Set2', data = df)

plt.subplot(3, 2, 5)
plt.gca().set_title('Variable diabetes_med')
sns.countplot(x = 'diabetes_med', hue = 'readmitted',  palette = 'Set2', data = df)
```

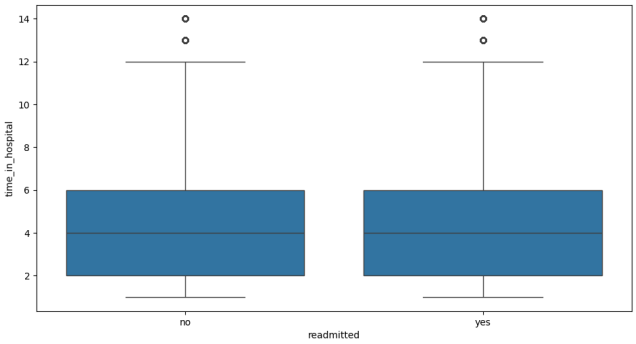
Out[94]: <Axes: title={'center': 'Variable diabetes_med'}, xlabel='diabetes_med', ylabel='count'>

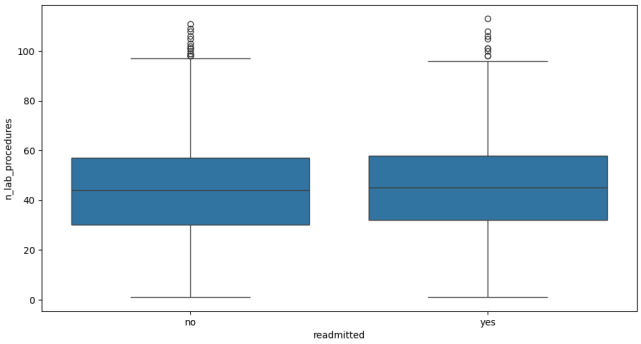


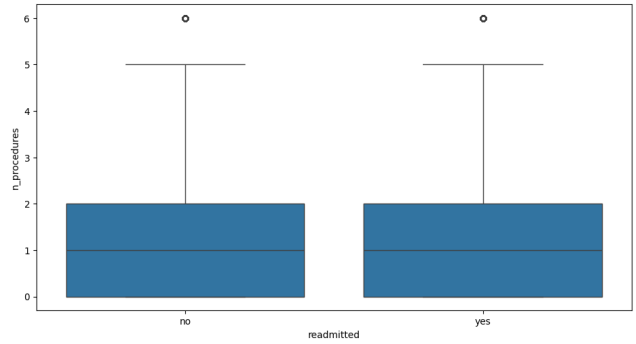
In [95]:
```
plt.figure(figsize = (25, 20))
plt.suptitle("Analysis Of Variable readmitted",fontweight="bold", fontsize=20)

plt.subplot(3,2,1)
sns.boxplot(x="readmitted", y="time_in_hospital", data=df)

plt.subplot(3,2,2)
sns.boxplot(x="readmitted", y="n_lab_procedures", data=df)

plt.subplot(3,2,3)
sns.boxplot(x="readmitted", y="n_procedures", data=df)

plt.subplot(3,2,4)
sns.boxplot(x="readmitted", y="n_medications", data=df)

plt.subplot(3,2,5)
sns.boxplot(x="readmitted", y="n_outpatient", data=df)

plt.subplot(3,2,6)
sns.boxplot(x="readmitted", y="n_emergency", data=df)
```

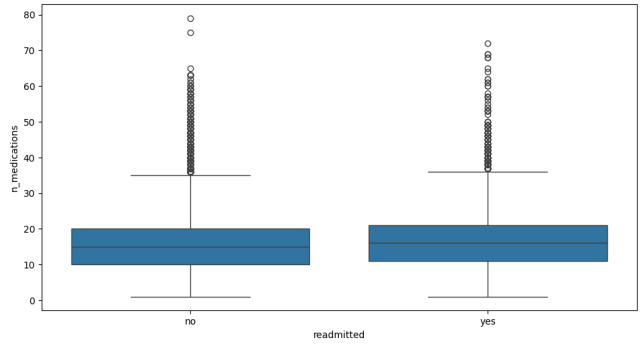
Out[95]: <Axes: xlabel='readmitted', ylabel='n_emergency'>

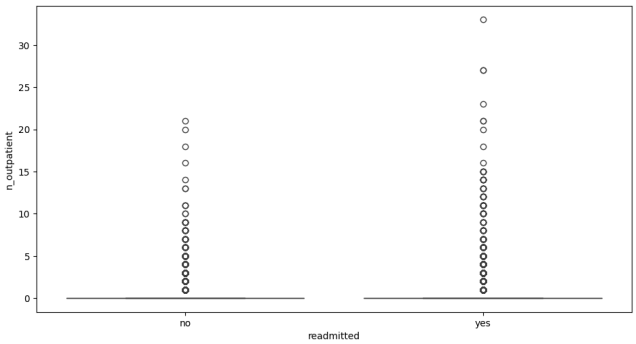
Analysis Of Variable readmitted

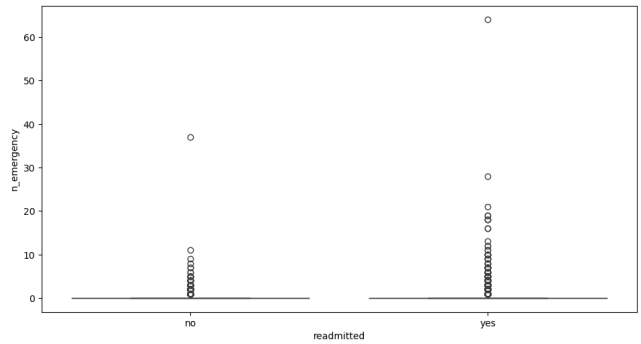












In [96]:
```
df['diag_1'].value_counts() / df.shape[0]
```

Out[96]:
```
diag_1
Circulatory        0.31296
Other              0.25992
Respiratory        0.14720
Digestive          0.09316
Diabetes           0.06988
Injury             0.06664
Musculoskeletal    0.05008
Missing            0.00016
Name: count, dtype: float64
```

In [97]:
```
df['diag_2'].value_counts() / df.shape[0]
```

Out[97]:
```
diag_2
Other              0.36224
Circulatory        0.32536
Diabetes           0.11624
Respiratory        0.11488
Digestive          0.03892
Injury             0.02364
Musculoskeletal    0.01704
Missing            0.00168
Name: count, dtype: float64
```

In [98]:
```
df['diag_3'].value_counts() / df.shape[0]
```

Out[98]:
```
diag_3
Other              0.36428
Circulatory        0.30744
Diabetes           0.17044
Respiratory        0.07660
Digestive          0.03664
Injury             0.01856
Musculoskeletal    0.01820
Missing            0.00784
Name: count, dtype: float64
```

In [99]:
```
categorical_features
```

Out[99]:
```
Index(['age', 'medical_specialty', 'diag_1', 'diag_2', 'diag_3',
       'glucose_test', 'A1Ctest', 'change', 'diabetes_med', 'readmitted'],
      dtype='object')
```

In [5]:
```
binaryCols = ['change','diabetes_med','readmitted']
for i in binaryCols:
    df[i] = df[i].apply(lambda x: 0 if x == 'no' else 1)

noNormalHighCols = ['glucose_test','A1Ctest']
```

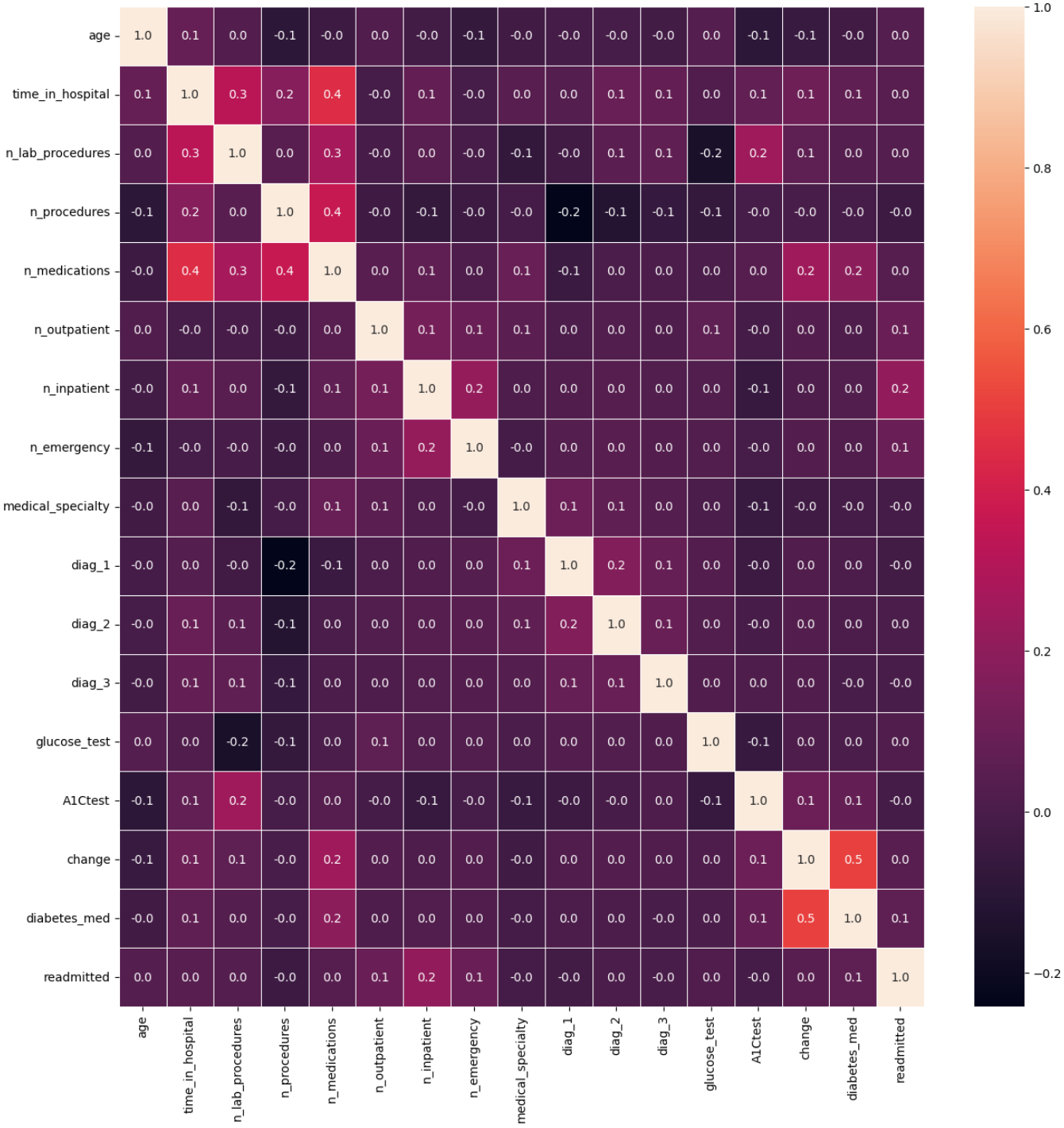
```
for i in noNormalHighCols:
    df[i] = df[i].apply(lambda x: 0 if x == 'no' else 1 if x == 'normal' else 2)


def label_encode_cat_features(data, cat_features):
    label_encoder = LabelEncoder()
    data_encoded = data.copy()

    for col in cat_features:
        data_encoded[col] = label_encoder.fit_transform(data[col])

    data = data_encoded
    return data

col_cat = ['age', 'medical_specialty', 'diag_1', 'diag_2', 'diag_3']
df = label_encode_cat_features(df, col_cat)

df.head()
```

Out[5]:

	age	time_in_hospital	n_lab_procedures	n_procedures	n_medications	n_outpatient	n_inpatient	n_emergency	medical_specialty	diag_1	diag_2	diag_3	glucose_test	A
0	3	8	72	1	18	2	0	0	4	0	7	6	0	
1	3	3	34	2	13	0	0	0	5	6	6	6	0	
2	1	5	45	0	18	0	0	0	4	0	0	0	0	
3	3	2	36	0	12	1	0	0	4	0	6	1	0	
4	2	1	42	0	7	0	0	0	3	6	0	7	0	

In [ ]:
```
# Correlation matrix
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(), annot=True, linewidth=.5, fmt=".1f")
```

Out[ ]: <Axes: >



In [102…
```
df['readmitted'].value_counts() / df.shape[0]
```

Out[102…
```
readmitted
0    0.52984
1    0.47016
Name: count, dtype: float64
```

In [6]:
```
quantCols = df.select_dtypes(include=[int,float]).columns
df_features = df[quantCols]
df_features.head()
```

Out[6]:

	age	time_in_hospital	n_lab_procedures	n_procedures	n_medications	n_outpatient	n_inpatient	n_emergency	medical_specialty	diag_1	diag_2	diag_3	glucose_test	A
0	3	8	72	1	18	2	0	0	4	0	7	6	0	
1	3	3	34	2	13	0	0	0	5	6	6	6	0	
2	1	5	45	0	18	0	0	0	4	0	0	0	0	
3	3	2	36	0	12	1	0	0	4	0	6	1	0	
4	2	1	42	0	7	0	0	0	3	6	0	7	0	

In [17]:
```
#Transforming all colimns that have noticeable skewness
transformCols = [
    'time_in_hospital','n_lab_procedures','n_procedures',
    'n_medications','n_outpatient','n_inpatient','n_emergency'
]

# checking if all column datatype is numeric
all([pd.api.types.is_numeric_dtype(df_features[col]) for col in transformCols])
```

Out[17]:
```
True
```

In [19]:
```
# Perform a log transformation of the data to unskew the data
df_sqrt = np.sqrt(df_features)

#Check how many INF readings we have in the data
np.isinf(df_sqrt).sum()
```

Out[19]:
```
age                 0
time_in_hospital    0
n_lab_procedures    0
n_procedures        0
n_medications       0
n_outpatient        0
n_inpatient         0
n_emergency         0
medical_specialty   0
diag_1              0
diag_2              0
diag_3              0
glucose_test        0
A1Ctest             0
change              0
diabetes_med        0
readmitted          0
dtype: int64
```

In [20]:
```
#Change INF values to mean for each feature
for i in transformCols:
    df_sqrt[i].replace([np.inf, -np.inf], np.nan, inplace=True)
    df_sqrt[i] = df_sqrt[i].fillna(df_sqrt[i].mean())

np.isinf(df_sqrt).sum()
```

Out[20]:
```
age                 0
time_in_hospital    0
n_lab_procedures    0
n_procedures        0
n_medications       0
n_outpatient        0
n_inpatient         0
n_emergency         0
medical_specialty   0
diag_1              0
diag_2              0
diag_3              0
glucose_test        0
A1Ctest             0
change              0
diabetes_med        0
readmitted          0
dtype: int64
```

In [21]:
```
df_sqrt.shape
```

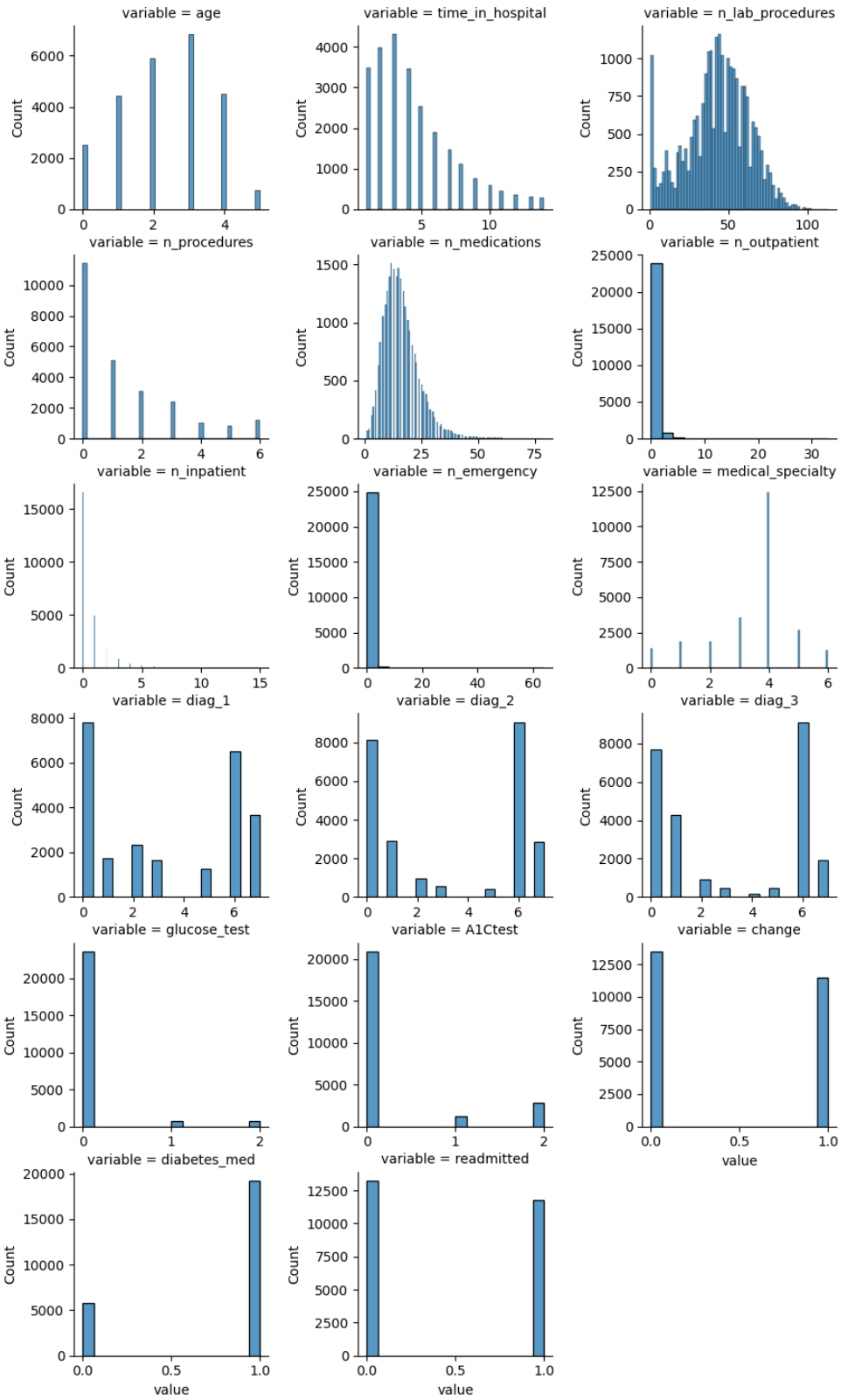
Out[21]:
```
(25000, 17)
```

In [179…
```
df_sqrt.head()
```

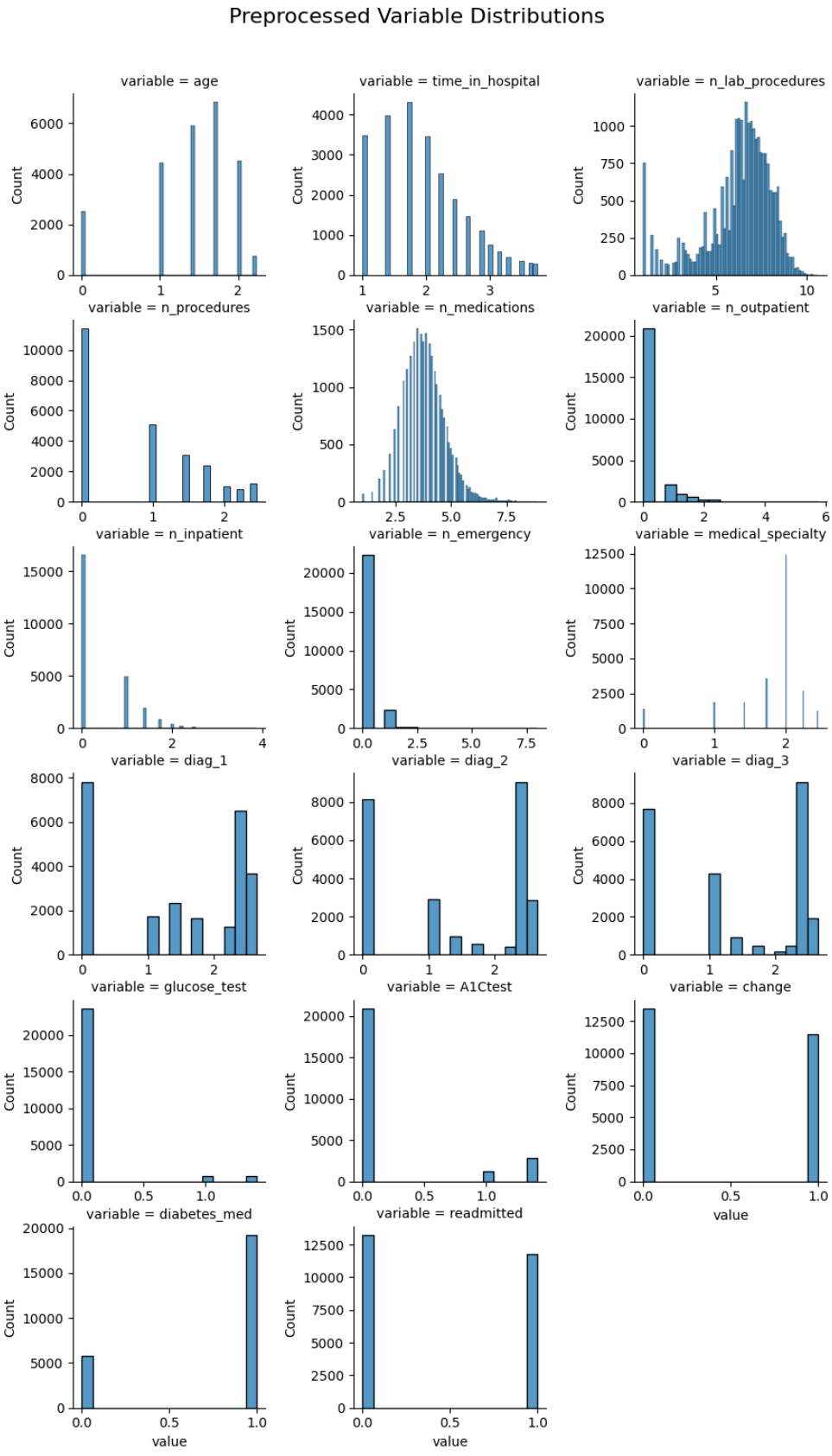
Out[179…

	age	time_in_hospital	n_lab_procedures	n_procedures	n_medications	n_outpatient	n_inpatient	n_emergency	medical_specialty	diag_1	diag_2	diag_3	gluc
0	1.732051	2.828427	8.485281	1.000000	4.242641	1.414214	0.0	0.0	2.000000	0.00000	2.645751	2.449490	
1	1.732051	1.732051	5.830952	1.414214	3.605551	0.000000	0.0	0.0	2.236068	2.44949	2.449490	2.449490	
2	1.000000	2.236068	6.708204	0.000000	4.242641	0.000000	0.0	0.0	2.000000	0.00000	0.000000	0.000000	
3	1.732051	1.414214	6.000000	0.000000	3.464102	1.000000	0.0	0.0	2.000000	0.00000	2.449490	1.000000	
4	1.414214	1.000000	6.480741	0.000000	2.645751	0.000000	0.0	0.0	1.732051	2.44949	0.000000	2.645751	

In [180…
```
g = sns.FacetGrid(
    df_features.melt(),
    col='variable',
    sharey=False,
    sharex=False,
    col_wrap = 3
)

g.map(sns.histplot, "value")
g.figure.subplots_adjust(top=0.8)
g.figure.suptitle("Unprocessed Variable Distributions", fontsize=16,y=.85)
plt.show()
```



In [22]:
```
df_sqrt['readmitted'] = df_features['readmitted']

g = sns.FacetGrid(
    df_sqrt.melt(),
    col="variable",
    col_wrap = 3,
    sharey=False,
    sharex=False
)
g.map(sns.histplot, "value")
g.figure.subplots_adjust(top=0.8)
g.figure.suptitle("Preprocessed Variable Distributions", fontsize=16,y=.85)
plt.show()
```



Model for Re-admission Classification

The following models are being compared for the classification task:

- Logistic Regression: A fundamental linear model that calculates the probability of a binary outcome. It serves as a strong baseline for classification problems.
- Decision Tree Classifier: A non-linear model that makes predictions by following a tree-like structure of decisions. It is easy to interpret but can be prone to overfitting.
- Random Forest Classifier: An ensemble model that builds multiple decision trees and averages their predictions to improve accuracy and reduce overfitting.
- Gradient Boosting Classifier: A powerful ensemble technique that builds a series of weak models sequentially. Each new model corrects the errors of its predecessor, leading to a highly accurate final model.
- AdaBoost Classifier: Another ensemble method that focuses on misclassified samples from previous iterations, giving them higher weight to improve the final model's performance.

Performance metrics:Accuracy, Precision, Recall, F1 score, ROC-AUC score and confusion metrics

In [102...

```
df.head()
```

Out[182…

	age	time_in_hospital	n_lab_procedures	n_procedures	n_medications	n_outpatient	n_inpatient	n_emergency	medical_specialty	diag_1	diag_2	diag_3	glucose_test	A
0	3	8	72	1	18	2	0	0	4	0	7	6	0	
1	3	3	34	2	13	0	0	0	5	6	6	6	0	
2	1	5	45	0	18	0	0	0	4	0	0	0	0	
3	3	2	36	0	12	1	0	0	4	0	6	1	0	
4	2	1	42	0	7	0	0	0	3	6	0	7	0	

In [183…
```
df_sqrt.head()
```

Out[183…

	age	time_in_hospital	n_lab_procedures	n_procedures	n_medications	n_outpatient	n_inpatient	n_emergency	medical_specialty	diag_1	diag_2	diag_3	gluc
0	1.732051	2.828427	8.485281	1.000000	4.242641	1.414214	0.0	0.0	2.000000	0.00000	2.645751	2.449490	
1	1.732051	1.732051	5.830952	1.414214	3.605551	0.000000	0.0	0.0	2.236068	2.44949	2.449490	2.449490	
2	1.000000	2.236068	6.708204	0.000000	4.242641	0.000000	0.0	0.0	2.000000	0.00000	0.000000	0.000000	
3	1.732051	1.414214	6.000000	0.000000	3.464102	1.000000	0.0	0.0	2.000000	0.00000	2.449490	1.000000	
4	1.414214	1.000000	6.480741	0.000000	2.645751	0.000000	0.0	0.0	1.732051	2.44949	0.000000	2.645751	

In [23]:
```
X = df.drop(columns=['readmitted'], axis=1)
X_processed = df_sqrt.drop(columns=['readmitted'], axis=1)
y = df_sqrt['readmitted']

X.shape, X_processed.shape, y.shape
```

Out[23]: ((25000, 16), (25000, 16), (25000,))

In [24]:
```
seed = 42
# train test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.10, random_state=seed)

X_train_processed, X_test_processed, y_train_processed, y_test_processed = train_test_split(
    X_processed, y, test_size=0.10, random_state=seed)

X_train.shape, X_test.shape, X_train_processed.shape, X_test_processed.shape
```

Out[24]: ((22500, 16), (2500, 16), (22500, 16), (2500, 16))

In [ ]:
```
# Performing PCA to reduce the dimensionality of the dataset
pca = PCA(n_components=None)
pipeline = make_pipeline(StandardScaler(), pca)

# Fit on train, transform both
X_train_trans = pipeline.fit_transform(X_train)
X_test_trans  = pipeline.transform(X_test)

X_train_trans_processed = pipeline.fit_transform(X_train_processed)
X_test_trans_processed  = pipeline.transform(X_test_processed)

X_train_trans.shape, X_train_trans_processed.shape
```

Out[ ]: ((22500, 16), (22500, 16))

In [187…
```
def roc_auc_display(test_result, model_name):
    plt.figure(figsize=(4,4))
    for name in model_name:
        fpr, tpr, _ = roc_curve(test_result['y_test'], test_result[name] )
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], 'r--', label='Random Guess')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curves for Two Models')
    plt.legend()
    plt.show()

def evaluate_model_classifer(model_list, model_name, X_train, y_train, X_test, y_test):
    pred_log = {'y_test': y_test}
    model_dict = {}
    results_list = []

    for model, name in zip(model_list, model_name):
        # train & predict
        model_base = model
        model_base.fit(X_train, y_train)
        y_pred = model_base.predict(X_test)

        # save predictions and model
        pred_log[name] = y_pred
        model_dict[name] = model_base

        # compute metrics
        acc = accuracy_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred)
        rec = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)
        roc = roc_auc_score(y_test, y_pred)
        cm = confusion_matrix(y_test, y_pred)

        # append to results list for DataFrame
        results_list.append({
            'Name': name,
            'Accuracy': acc,
            'Precision': prec,
            'Recall': rec,
            'F1 score': f1,
            'ROC_AUC': roc,
            'Confusion_Matrix': cm
        })

    # build DataFrame and print a rounded view for neat output
    results_df = pd.DataFrame(results_list)
    display_df = results_df.copy()
    numeric_cols = ['Accuracy', 'Precision', 'Recall', 'F1 score', 'ROC_AUC']
    display_df[numeric_cols] = display_df[numeric_cols].round(4)
    print(display_df.to_string(index=False))
```

```
    return pred_log, model_dict, results_df
```

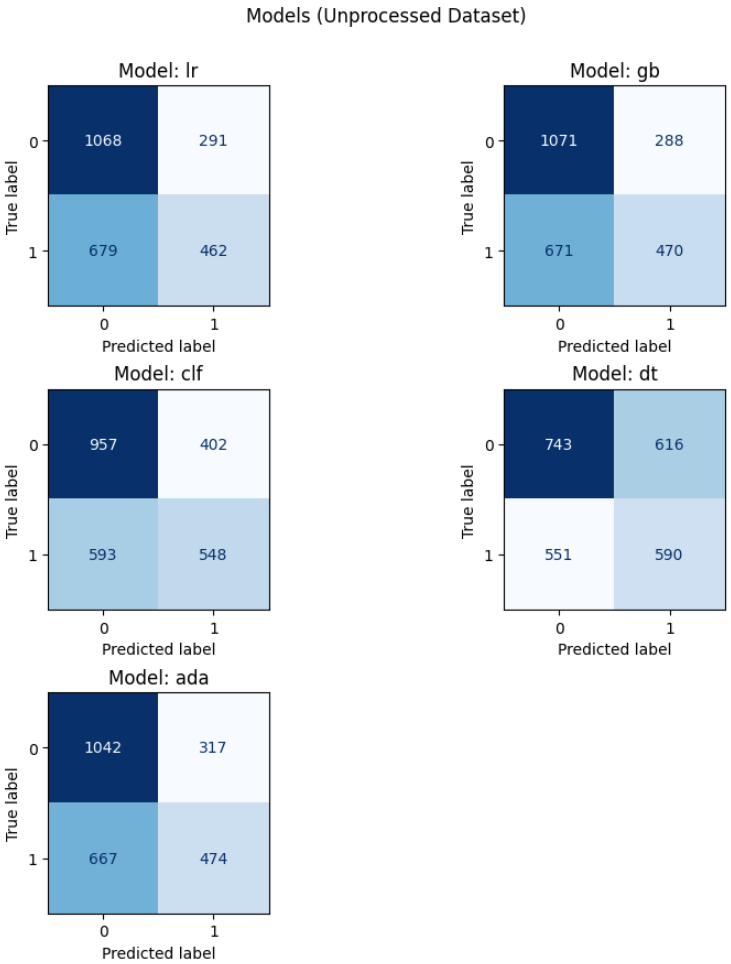
In [ ]:
```
lr = LogisticRegression(tol=1e-4, max_iter=1000, random_state=seed)

space = dict(C=uniform(loc=0, scale=5),
                    penalty=['l2', 'l1'],
                    solver= ['liblinear'])

search = RandomizedSearchCV(lr,
                        space,
                        random_state=seed,
                        cv = 5,
                        scoring='f1')

rand_search = search.fit(X_train, y_train)
print('Best Hyperparameters: %s' % rand_search.best_params_)

params = rand_search.best_params_
lr = LogisticRegression(**params)
lr.fit(X_train, y_train)
print(classification_report(y_test, lr.predict(X_test)))
```

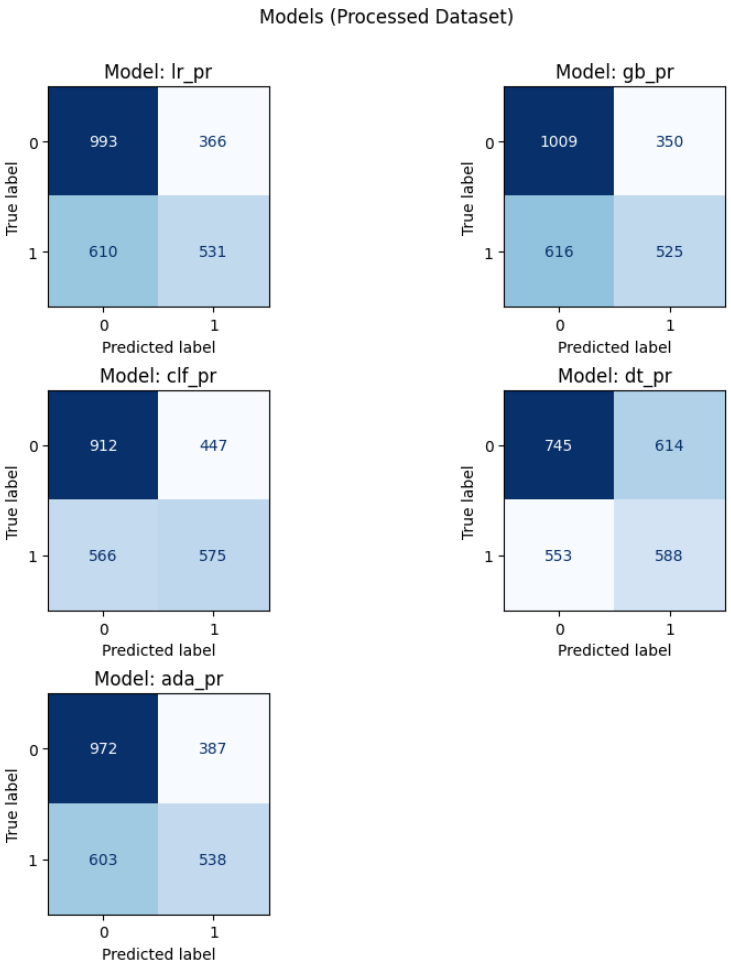
In [213...
```
# models for readmission prediction
seed = 42
model_lr = LogisticRegression()
model_gb = GradientBoostingClassifier(random_state=seed)
model_clf = RandomForestClassifier(random_state=seed)
model_dt = DecisionTreeClassifier(random_state=seed)
model_ada = AdaBoostClassifier(random_state=seed)

model_list = [model_lr, model_gb, model_clf, model_dt, model_ada]
model_name = ['lr', 'gb', 'clf', 'dt', 'ada']
model_name_pr = ['lr_pr', 'gb_pr', 'clf_pr', 'dt_pr', 'ada_pr']

pred_log, model_dict, results_df = evaluate_model_classifer(
    model_list, model_name, X_train_trans, y_train, X_test_trans, y_test
)
print()

pred_log_pr, model_dict_pr, results_df_pr = evaluate_model_classifer(
    model_list, model_name_pr, X_train_trans_processed, y_train_processed, X_test_trans_processed, y_test_processed
)
```

```
Name  Accuracy  Precision  Recall  F1 score  ROC_AUC          Confusion_Matrix
  lr    0.6120     0.6135  0.4049    0.4879   0.5954 [[1068, 291], [679, 462]]
  gb    0.6164     0.6201  0.4119    0.4950   0.6000 [[1071, 288], [671, 470]]
 clf    0.6020     0.5768  0.4803    0.5242   0.5922  [[957, 402], [593, 548]]
  dt    0.5332     0.4892  0.5171    0.5028   0.5319  [[743, 616], [551, 590]]
 ada    0.6064     0.5992  0.4154    0.4907   0.5911 [[1042, 317], [667, 474]]

  Name  Accuracy  Precision  Recall  F1 score  ROC_AUC          Confusion_Matrix
 lr_pr    0.6096     0.5920  0.4654    0.5211   0.5980  [[993, 366], [610, 531]]
 gb_pr    0.6136     0.6000  0.4601    0.5208   0.6013 [[1009, 350], [616, 525]]
clf_pr    0.5948     0.5626  0.5039    0.5317   0.5875  [[912, 447], [566, 575]]
 dt_pr    0.5332     0.4892  0.5153    0.5019   0.5318  [[745, 614], [553, 588]]
ada_pr    0.6040     0.5816  0.4715    0.5208   0.5934  [[972, 387], [603, 538]]
```

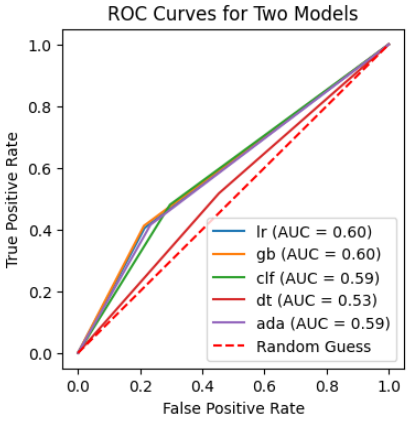
Processing the dataset to correct for skewness if not affecting the performance in any significant way. However, the overall F1 score and recall of the models trained on processed dataset is higher, hence proceeding for hyperparameter tuning on processed dataset.

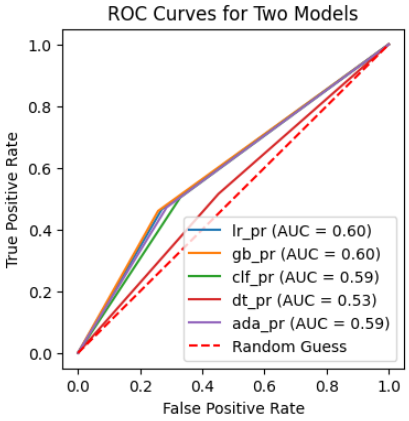
In [214...
```
def plot_cm(results_df, n_cols, plot_super_title):
    n_models = len(results_df)
    n_rows = math.ceil(n_models / n_cols)

    fig, axes = plt.subplots(n_rows, n_cols, figsize=(9, 3 * n_rows))  # adjust width & height
    axes = axes.flatten()  # flatten in case of multiple rows

    for i, (_, row) in enumerate(results_df.iterrows()):
        name = row['Name']
        cm = row['Confusion_Matrix']

        disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
        disp.plot(ax=axes[i], cmap="Blues", values_format="d", colorbar=False)
        axes[i].set_title(f'Model: {name}')

    # Hide any unused subplots if models < n_cols * n_rows
    for j in range(i+1, len(axes)):
        fig.delaxes(axes[j])

    fig.tight_layout()
    fig.subplots_adjust(top=0.9)
    fig.suptitle(plot_super_title)
    plt.show()

plot_cm(results_df, 2, 'Models (Unprocessed Dataset)')
print()
plot_cm(results_df_pr, 2, 'Models (Processed Dataset)')
```





In [190…
```
roc_auc_display(pred_log, model_name)
print()
roc_auc_display(pred_log_pr, model_name_pr)
```





Hyperparameter Tuning

In [193…
```
seed = 42
lr = LogisticRegression(tol=1e-4, max_iter=1000, random_state=seed)

space = dict(C=uniform(loc=0, scale=5),
                    penalty=['l2', 'l1'],
                    solver= ['liblinear'])

search = RandomizedSearchCV(lr,
                        space,
                        random_state=seed,
                        cv = 5,
                        scoring='f1')

rand_search = search.fit(X_train, y_train)
print('Best Hyperparameters: %s' % rand_search.best_params_)

params = rand_search.best_params_
lr = LogisticRegression(**params)
lr.fit(X_train_trans_processed, y_train)
print(classification_report(y_test, lr.predict(X_test_processed)))
```

```
Best Hyperparameters: {'C': 0.9171739493308819, 'penalty': 'l1', 'solver': 'liblinear'}
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1359
           1       0.46      1.00      0.63      1141

    accuracy                           0.46      2500
   macro avg       0.23      0.50      0.31      2500
weighted avg       0.21      0.46      0.29      2500
```

In [197…
```
# Models
seed=42
model_lr  = LogisticRegression(random_state=seed, max_iter=2000)
model_gb  = GradientBoostingClassifier(random_state=seed)
model_clf = RandomForestClassifier(random_state=seed)
model_dt  = DecisionTreeClassifier(random_state=seed)
model_ada = AdaBoostClassifier(random_state=seed)

# Common CV setup
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
scoring = 'f1'

# Pipelines
pipe_lr  = Pipeline([('scaler', StandardScaler()), ('clf', model_lr)])
pipe_gb  = Pipeline([('scaler', StandardScaler()), ('clf', model_gb)])
pipe_rf  = Pipeline([('scaler', StandardScaler()), ('clf', model_clf)])
pipe_dt  = Pipeline([('scaler', StandardScaler()), ('clf', model_dt)])
pipe_ada = Pipeline([('scaler', StandardScaler()), ('clf', model_ada)])

# Parameter grids
param_grids = {
    'lr': {
        'clf__penalty': ['l1', 'l2'],
        'clf__C': [0.01, 0.1, 1.0, 10.0],
        'clf__solver': ['saga'],
        'clf__max_iter': [2000]
    },
    'gb': {
        'clf__n_estimators': [100, 300],
        'clf__learning_rate': [0.01, 0.05, 0.1],
        'clf__max_depth': [3, 5],
        'clf__subsample': [0.6, 0.8, 1.0],
        'clf__min_samples_split': [2, 5]
```

```
    },
    'rf': {
        'clf__n_estimators': [100, 300],
        'clf__max_depth': [None, 10, 20],
        'clf__min_samples_split': [2, 5],
        'clf__max_features': ['sqrt', 'log2', 0.3]
    },
    'dt': {
        'clf__criterion': ['gini', 'entropy'],
        'clf__max_depth': [None, 5, 10, 20],
        'clf__min_samples_split': [2, 5, 10],
        'clf__min_samples_leaf': [1, 2, 5]
    },
    'ada': {
        'clf__n_estimators': [50, 100, 200],
        'clf__learning_rate': [0.01, 0.1, 1.0],
        'clf__algorithm': ['SAMME.R', 'SAMME']
    }
}

# Mapping pipelines
pipes = {
    'lr': pipe_lr,
    'gb': pipe_gb,
    'rf': pipe_rf,
    'dt': pipe_dt,
    'ada': pipe_ada
}

# Run GridSearchCV for each model and collect results
results_tuned = []
for name in ['lr', 'gb', 'rf', 'dt', 'ada']:
    print(f"\nRunning GridSearch for: {name}")
    grid = GridSearchCV(
        estimator=pipes[name],
        param_grid=param_grids[name],
        scoring=scoring,
        cv=cv,
        n_jobs=-1,
        verbose=0,
        return_train_score=False
    )
    grid.fit(X_train_trans_processed, y_train)
    best = grid.best_estimator_
    best_params = grid.best_params_
    best_score = grid.best_score_
    print(f"Best {scoring} for {name}: {best_score:.4f}")
    print("Best params:", best_params)

    results_tuned.append({
        'model': name,
        'best_score': best_score,
        'best_params': best_params,
        'best_estimator': best
    })

# Convert to DataFrame for a quick summary
results_df_tuned = pd.DataFrame(results_tuned).sort_values('best_score', ascending=False)
results_df_tuned
```

```
Running GridSearch for: lr
Best f1 for lr: 0.5280
Best params: {'clf__C': 1.0, 'clf__max_iter': 2000, 'clf__penalty': 'l2', 'clf__solver': 'saga'}

Running GridSearch for: gb
Best f1 for gb: 0.5501
Best params: {'clf__learning_rate': 0.05, 'clf__max_depth': 5, 'clf__min_samples_split': 2, 'clf__n_estimators': 300, 'clf__subsample': 0.6}

Running GridSearch for: rf
Best f1 for rf: 0.5629
Best params: {'clf__max_depth': None, 'clf__max_features': 'sqrt', 'clf__min_samples_split': 2, 'clf__n_estimators': 300}

Running GridSearch for: dt
Best f1 for dt: 0.5191
Best params: {'clf__criterion': 'entropy', 'clf__max_depth': 10, 'clf__min_samples_leaf': 5, 'clf__min_samples_split': 2}

Running GridSearch for: ada
```

```
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
```

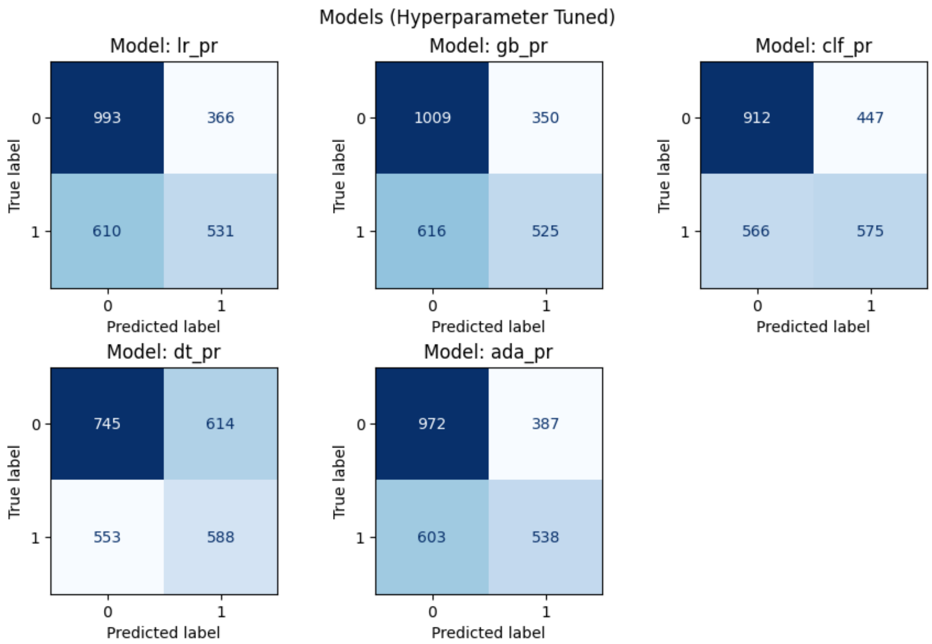
```
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
/opt/miniconda3/envs/envTorch/lib/python3.11/site-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The parameter 'algorithm' is deprecated
in 1.6 and has no effect. It will be removed in version 1.8.
  warnings.warn(
Best f1 for ada: 0.5231
Best params: {'clf__algorithm': 'SAMME', 'clf__learning_rate': 1.0, 'clf__n_estimators': 100}
```

	model	best_score	best_params	best_estimator
2	rf	0.562919	{'clf__max_depth': None, 'clf__max_features': 'sqrt', 'clf__min_samples_split': 2, 'clf__n_estim...	(StandardScaler(), (DecisionTreeClassifier(max_features='sqrt', random_state=1608637542), Decisi...
1	gb	0.550065	{'clf__learning_rate': 0.05, 'clf__max_depth': 5, 'clf__min_samples_split': 2, 'clf__n_estimator...	(StandardScaler(), ([DecisionTreeRegressor(criterion='friedman_mse', max_depth=5,\n ...
0	lr	0.528001	{'clf__C': 1.0, 'clf__max_iter': 2000, 'clf__penalty': 'l2', 'clf__solver': 'saga'}	(StandardScaler(), LogisticRegression(max_iter=2000, random_state=42, solver='saga'))
4	ada	0.523070	{'clf__algorithm': 'SAMME', 'clf__learning_rate': 1.0, 'clf__n_estimators': 100}	(StandardScaler(), (DecisionTreeClassifier(max_depth=1, random_state=1608637542), DecisionTreeCl...
3	dt	0.519150	{'clf__criterion': 'entropy', 'clf__max_depth': 10, 'clf__min_samples_leaf': 5, 'clf__min_sample...	(StandardScaler(), DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_leaf=5,...

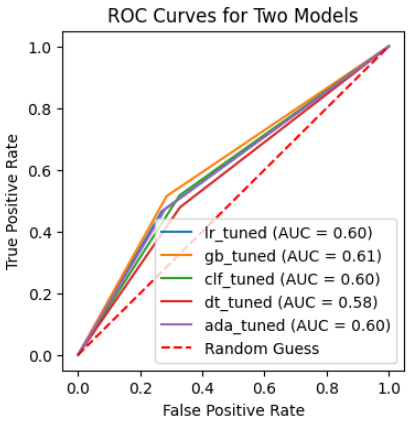
```python
best_pipelines = {r['model']: r['best_estimator'] for r in results_tuned}

model_list = [
    best_pipelines['lr'],
    best_pipelines['gb'],
    best_pipelines['rf'],
    best_pipelines['dt'],
    best_pipelines['ada']
]
model_name = ['lr_tuned', 'gb_tuned', 'clf_tuned', 'dt_tuned', 'ada_tuned']

pred_log_tuned, model_dict_tuned, results_df_tuned = evaluate_model_classifer(
    model_list, model_name, X_train_trans_processed, y_train_processed, X_test_trans_processed, y_test_processed
)
```

```
     Name  Accuracy  Precision  Recall  F1 score  ROC_AUC        Confusion_Matrix
 lr_tuned    0.6096     0.5920  0.4654    0.5211   0.5980 [[993, 366], [610, 531]]
 gb_tuned    0.6236     0.6027  0.5145    0.5551   0.6148 [[972, 387], [554, 587]]
clf_tuned    0.6020     0.5706  0.5171    0.5425   0.5952 [[915, 444], [551, 590]]
 dt_tuned    0.5836     0.5504  0.4785    0.5120   0.5752 [[913, 446], [595, 546]]
ada_tuned    0.6068     0.5859  0.4724    0.5230   0.5960 [[978, 381], [602, 539]]
```

```python
plot_cm(results_df_pr, 3, 'Models (Hyperparameter Tuned)')
```




```python
roc_auc_display(pred_log_tuned, ['lr_tuned', 'gb_tuned', 'clf_tuned', 'dt_tuned', 'ada_tuned'])
```

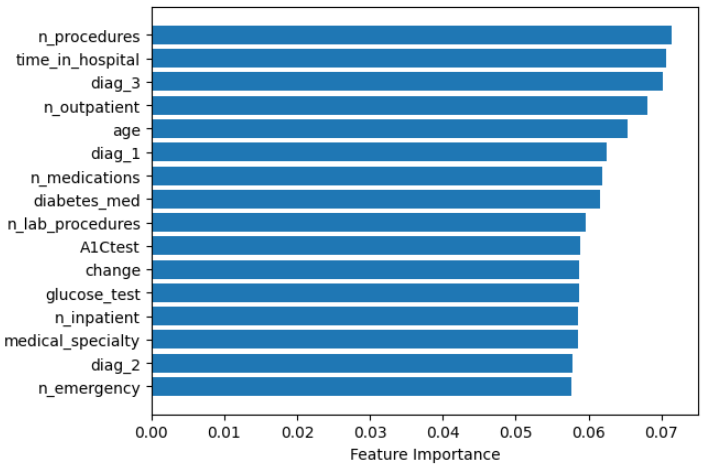


In [ ]:
```
# Feature importance
print("Feature importance (Logistic Regression) (Coefficient and Odds Ratio):: ")

# Coefficients and Odds Ratios
log_reg = best_pipelines['lr'].named_steps['clf']
coefficients = log_reg.coef_[0]
odds_ratios = np.exp(coefficients)

feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': coefficients,
    'Odds Ratio': odds_ratios
})
print(feature_importance.sort_values(by='Coefficient', ascending=False))
```

```
Feature importance (Logistic Regression) (Coefficient and Odds Ratio)::
              Feature  Coefficient  Odds Ratio
3        n_procedures     0.330980    1.392332
1    time_in_hospital     0.280817    1.324211
4       n_medications     0.200812    1.222395
0                 age     0.177896    1.194701
5        n_outpatient     0.147357    1.158767
2    n_lab_procedures     0.086154    1.089974
12       glucose_test     0.056940    1.058592
14             change     0.038906    1.039672
13            A1Ctest     0.012011    1.012083
15       diabetes_med     0.008468    1.008504
8   medical_specialty     0.002118    1.002120
10             diag_2    -0.015176    0.984938
7         n_emergency    -0.050506    0.950748
9              diag_1    -0.072368    0.930188
6         n_inpatient    -0.072618    0.929956
11             diag_3    -0.105117    0.900219
```

In [209...
```
# Feature importance
print("Feature importance (Random Forest) (Coefficient and Odds Ratio):: ")
clf = best_pipelines['rf'].named_steps['clf']
importances = clf.feature_importances_
feature_names = X.columns
sorted_idx = importances.argsort()

plt.barh(range(len(importances)), importances[sorted_idx])
plt.yticks(range(len(importances)), feature_names[sorted_idx])
plt.xlabel("Feature Importance")
plt.show()
```

```
Feature importance (Random Forest) (Coefficient and Odds Ratio)::
```



In [238...
```
importances = clf.feature_importances_
feature_names = X.columns
for i in range(len(feature_names)):
    print(f"{feature_names[i]} : {importances[i]}")
```

```
age : 0.06543477986033111
time_in_hospital : 0.07060294005399916
n_lab_procedures : 0.059558450810273354
n_procedures : 0.07145417428242885
n_medications : 0.061888537902753775
n_outpatient : 0.06803925523030398
n_inpatient : 0.058621728562216084
n_emergency : 0.057604907907453415
medical_specialty : 0.05852173206154494
diag_1 : 0.06244013867991918
diag_2 : 0.057762515060332344
diag_3 : 0.07018356165118397
glucose_test : 0.05866969721755237
A1Ctest : 0.05888998872779137
change : 0.05878620956015825
diabetes_med : 0.061532372431765424
```

The following observations can be made from the above graph:

- Patients undergoing more procedures are at higher risk.
- Longer hospital stays indicate complexity or comorbidity.
- Specific diagnosis codes impact readmission risk.
- Older patients are more likely to be readmitted.
- Frequent outpatient visits may indicate underlying health issues.

Model for Predicting the Length of Stay

The following models were used for the regression task to predict the length of hospital stay:

- **Linear Regression:** A foundational statistical model that predicts a continuous outcome based on a linear relationship with the input features.
- **Decision Tree Regressor:** A model that partitions the data based on feature values to predict a continuous outcome. It's similar to its classification counterpart but is used for regression tasks.
- **Random Forest Regressor:** An ensemble model that aggregates the predictions of multiple decision trees to improve accuracy and generalization for regression problems.
- **Gradient Boosting Regressor:** A powerful ensemble technique for regression that sequentially builds models to correct the errors of previous models.
- **AdaBoost Regressor:** An ensemble method that combines multiple weak regressors to create a strong predictor, with a focus on improving performance on difficult samples.

Performance Metrics:

- Mean Absolute Error
- Mean Absolute Error (After Rounding off predictions)
- Mean Squared Error
- Root Mean Squared Error
- R2 Score

In [198…
```
df_log.head()
```

Out[198…

	age	time_in_hospital	n_lab_procedures	n_procedures	n_medications	n_outpatient	n_inpatient	n_emergency	medical_specialty	diag_1	diag_2	diag_3	glucc
0	1.386294	2.197225	4.290459	0.693147	2.944439	1.098612	0.0	0.0	1.609438	0.00000	2.079442	1.945910	
1	1.386294	1.386294	3.555348	1.098612	2.639057	0.000000	0.0	0.0	1.791759	1.94591	1.945910	1.945910	
2	0.693147	1.791759	3.828641	0.000000	2.944439	0.000000	0.0	0.0	1.609438	0.00000	0.000000	0.000000	
3	1.386294	1.098612	3.610918	0.000000	2.564949	0.693147	0.0	0.0	1.609438	0.00000	1.945910	0.693147	
4	1.098612	0.693147	3.761200	0.000000	2.079442	0.000000	0.0	0.0	1.386294	1.94591	0.000000	2.079442	

In [217…
```
# Features from your transformed dataset
X_pd = df_sqrt.drop(columns=['readmitted', 'time_in_hospital'], axis=1)  # drop old classification + target
y_pd = df['time_in_hospital']  # use original target

X_train_pd, X_test_pd, y_train_pd, y_test_pd = train_test_split(X_pd, y_pd, test_size=0.10, random_state=42)

X_train_pd.shape, X_test_pd.shape
```

Out[217… ((22500, 15), (2500, 15))

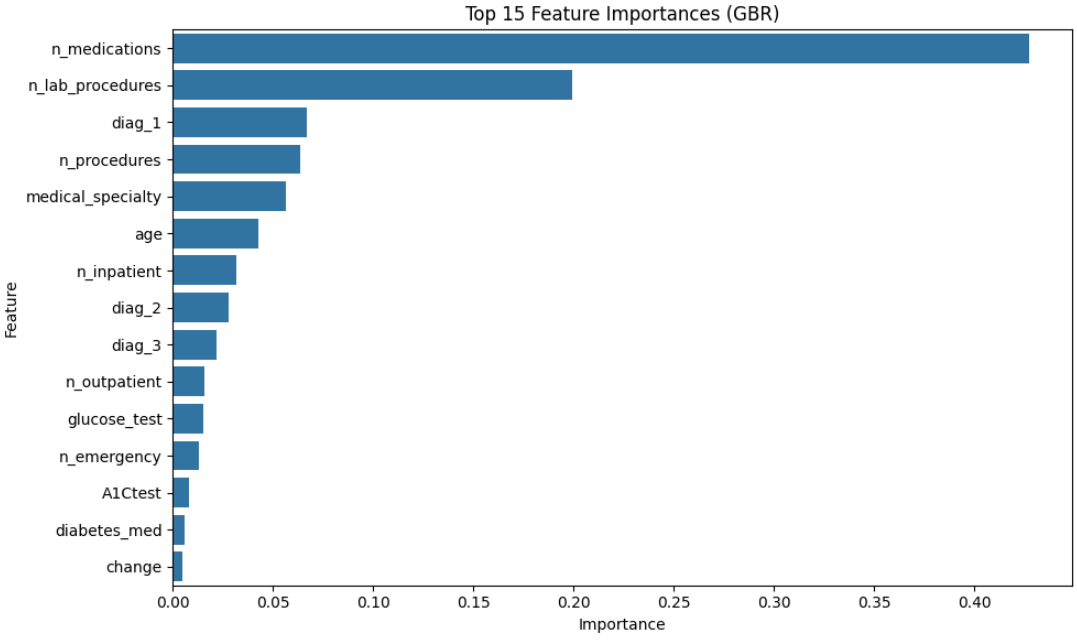
In [ ]:
```
seed = 42

# pipeline
pipe_lr_pd = Pipeline([('scaler', StandardScaler()), ('reg', LinearRegression())])
pipe_gbr_pd = Pipeline([('reg', GradientBoostingRegressor(random_state=seed))])
pipe_rf_pd = Pipeline([('reg', RandomForestRegressor(random_state=seed))])
pipe_dt_pd = Pipeline([('reg', DecisionTreeRegressor(random_state=seed))])
pipe_ada_pd = Pipeline([('reg', AdaBoostRegressor(random_state=seed))])

pipeline_dict_pd = {
    'lr': pipe_lr_pd,
    'gbr': pipe_gbr_pd,
    'rf': pipe_rf_pd,
    'dt': pipe_dt_pd,
    'ada': pipe_ada_pd
}

# paramter grid
param_grids_pd = {
    'lr': {
    },
    'gbr': {
        'reg__n_estimators': [100, 300],
        'reg__learning_rate': [0.01, 0.05, 0.1],
        'reg__max_depth': [3, 5],
        'reg__subsample': [0.6, 0.8, 1.0]
    },
    'rf': {
        'reg__n_estimators': [100, 300],
        'reg__max_depth': [None, 10, 20],
        'reg__min_samples_split': [2, 5],
        'reg__max_features': ['sqrt', 0.3]
    },
    'dt': {
        'reg__max_depth': [None, 5, 10, 20],
        'reg__min_samples_split': [2, 5, 10],
        'reg__min_samples_leaf': [1, 2, 4]
    },
```

```
    'ada': {
        'reg__n_estimators': [50, 100, 200],
        'reg__learning_rate': [0.01, 0.1, 1.0]

    }
}

# CV
cv_pd = KFold(n_splits=5, shuffle=True, random_state=seed)
scoring_pd = 'neg_mean_absolute_error'

results_pd = []
for name, pipe in pipeline_dict_pd.items():
    print(f"\n=== Grid search for {name} ===")
    grid = GridSearchCV(
        estimator=pipe,
        param_grid=param_grids_pd.get(name, {}),
        scoring=scoring_pd,
        cv=cv_pd,
        n_jobs=-1,
        verbose=0,
        refit=True
    )

    grid.fit(X_train_pd, y_train_pd)

    best = grid.best_estimator_
    best_params = grid.best_params_
    best_cv_mae = -grid.best_score_

    # Evaluate on test set
    y_pred = best.predict(X_test_pd)
    test_mae = mean_absolute_error(y_test_pd, y_pred)
    test_mse = mean_squared_error(y_test_pd, y_pred)
    test_rmse = np.sqrt(test_mse)
    test_r2 = r2_score(y_test_pd, y_pred)

    # Rounded/clipped predictions for integer day error
    y_pred_rounded = np.clip(np.round(y_pred), 1, 14).astype(int)
    test_mae_rounded = mean_absolute_error(y_test_pd, y_pred_rounded)

    print(f"Best CV MAE: {best_cv_mae:.4f}")
    print("Best params:", best_params)
    print(f"Test MAE (raw preds): {test_mae:.4f}")
    print(f"Test MAE (rounded & clipped to 1–14): {test_mae_rounded:.4f}")
    print(f"Test MSE: {test_mse} \nTest RMSE: {test_rmse:.4f} \nTest R2: {test_r2:.4f}\n")

    results_pd.append({
        'model': name,
        'best_params': best_params,
        'cv_mae': best_cv_mae,
        'test_mae': test_mae,
        'test_mae_rounded': test_mae_rounded,
        'test_mse': test_mse,
        'test_rmse': test_rmse,
        'test_r2': test_r2,
        'estimator': best
    })

summary_pd = pd.DataFrame(results_pd).sort_values('test_mae')
print("\n===== Summary (sorted by test MAE) =====")
print(summary_pd[['model','cv_mae','test_mae','test_mae_rounded','test_rmse','test_r2']])
```

```
=== Grid search for lr ===
Best CV MAE: 1.9699
Best params: {}
Test MAE (raw preds): 1.9607
Test MAE (rounded & clipped to 1–14): 1.9328
Test MSE: 6.506078728061827
Test RMSE: 2.5507
Test R2: 0.2587


=== Grid search for gbr ===
Best CV MAE: 1.8292
Best params: {'reg__learning_rate': 0.05, 'reg__max_depth': 5, 'reg__n_estimators': 300, 'reg__subsample': 0.6}
Test MAE (raw preds): 1.8267
Test MAE (rounded & clipped to 1–14): 1.8136
Test MSE: 5.81850094244545
Test RMSE: 2.4122
Test R2: 0.3371


=== Grid search for rf ===
Best CV MAE: 1.8705
Best params: {'reg__max_depth': 20, 'reg__max_features': 0.3, 'reg__min_samples_split': 5, 'reg__n_estimators': 300}
Test MAE (raw preds): 1.8593
Test MAE (rounded & clipped to 1–14): 1.8368
Test MSE: 5.945861157843448
Test RMSE: 2.4384
Test R2: 0.3226


=== Grid search for dt ===
Best CV MAE: 1.9681
Best params: {'reg__max_depth': 5, 'reg__min_samples_leaf': 4, 'reg__min_samples_split': 2}
Test MAE (raw preds): 1.9466
Test MAE (rounded & clipped to 1–14): 1.9584
Test MSE: 6.5047273042320715
Test RMSE: 2.5504
Test R2: 0.2589


=== Grid search for ada ===
Best CV MAE: 2.0069
Best params: {'reg__learning_rate': 0.01, 'reg__n_estimators': 50}
Test MAE (raw preds): 1.9761
Test MAE (rounded & clipped to 1–14): 1.9744
Test MSE: 6.629451760719629
Test RMSE: 2.5748
Test R2: 0.2447


===== Summary (sorted by test MAE) =====
  model    cv_mae  test_mae  test_mae_rounded  test_rmse   test_r2
1   gbr  1.829239  1.826706            1.8136   2.412157  0.337080
2    rf  1.870481  1.859267            1.8368   2.438414  0.322569
3    dt  1.968051  1.946614            1.9584   2.550437  0.258896
0    lr  1.969946  1.960656            1.9328   2.550702  0.258742
4   ada  2.006893  1.976067            1.9744   2.574772  0.244686
```

In [224…

```python
# feature importance
gbr_model = summary_pd.loc[summary_pd['model'] == 'gbr', 'estimator'].values[0]
gbr_reg = gbr_model.named_steps['reg']

feature_importances = pd.DataFrame({
    'Feature': X_train_pd.columns,
    'Importance': gbr_reg.feature_importances_
})

feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feature_importances)
plt.title('Top 15 Feature Importances (GBR)')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```



In [235…

```python
feature_importances
```

Out[235...

	Feature	Importance
3	n_medications	0.427435
1	n_lab_procedures	0.199480
8	diag_1	0.066956
2	n_procedures	0.063471
7	medical_specialty	0.056438
0	age	0.042415
5	n_inpatient	0.031486
9	diag_2	0.028017
10	diag_3	0.021760
4	n_outpatient	0.015759
11	glucose_test	0.015166
6	n_emergency	0.012868
12	A1Ctest	0.007949
14	diabetes_med	0.005963
13	change	0.004837

Observations drawn from the graph

- Most important predictors:
  - n_medications (number of medications)
  - n_lab_procedures (number of lab tests)
  - These dominate the prediction of hospital stay length, suggesting more treatments and lab investigations correlate with longer stays.
- Moderately important predictors:
  - diag_1 (primary diagnosis)
  - n_procedures (number of procedures)
  - medical_specialty (specialty of the admitting physician)
  - These show that the type of illness and medical care also significantly affect hospital stay.
- Smaller contributions:
  - age
  - n_inpatient, n_outpatient, n_emergency (prior healthcare visits)
  - Patient demographics and past healthcare utilization have some impact, but less than treatment intensity.
- Least important predictors:
  - A1Ctest, glucose_test
  - change (change in diabetes medication)
  - diabetes_med (whether diabetes medication prescribed)
  - Diabetes-specific measures contribute very little to predicting hospital stay length.
- Overall takeaway:
  - Intensity of care (medications, labs, procedures) is the main driver of hospital stay length.
  - Patient history and comorbidities play a secondary role.

Anomaly Detection

The following models were used for anomaly detection to identify unusual patient cases:

- **One-Class SVM:** An unsupervised learning algorithm that is trained on a dataset with only one class of data (the "normal" data). The model learns to identify a boundary that separates the normal data points from any outliers or anomalies.
- **Isolation Forest:** An efficient algorithm that detects anomalies by isolating them from the rest of the data. It builds a forest of random trees and measures the number of splits required to isolate a data point. Anomalies, being few and different, are isolated in fewer steps.

Performance Metrics: Precision, Recall, F1 score

In [25]:
```
# Features (exclude target)
X = df_features.drop(columns=['readmitted'])

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Target
y = df_features['readmitted']  # 1 = readmitted, 0 = normal
```

In [8]:
```
# Assume 1 = anomaly (readmitted), -1 = normal
iso_forest = IsolationForest(contamination=y.mean(), random_state=42)  # contamination ~ proportion of anomalies
iso_forest.fit(X_scaled)

# Predict anomalies (-1 = normal, 1 = anomaly) -> map to 0/1
y_pred_iso = iso_forest.predict(X_scaled)
y_pred_iso = [1 if p == -1 else 0 for p in y_pred_iso]  # flip because IsolationForest marks outliers as -1

# Evaluate
precision_iso = precision_score(y, y_pred_iso)
recall_iso = recall_score(y, y_pred_iso)
f1_iso = f1_score(y, y_pred_iso)

print("Isolation Forest - Precision:", precision_iso)
print("Isolation Forest - Recall:", recall_iso)
print("Isolation Forest - F1-score:", f1_iso)
```

```
Isolation Forest - Precision: 0.5024672451931258
Isolation Forest - Recall: 0.5024672451931258
Isolation Forest - F1-score: 0.5024672451931258
```

In [9]:
```
# Fit One-Class SVM on the majority class only (normal patients)
X_normal = X_scaled[y == 0]
oc_svm = OneClassSVM(nu=y.mean(), kernel='rbf', gamma='scale')
oc_svm.fit(X_normal)

# Predict anomalies (-1 = anomaly, 1 = normal)
```

```
y_pred_svm = oc_svm.predict(X_scaled)
y_pred_svm = [1 if p == -1 else 0 for p in y_pred_svm]  # map -1 -> anomaly

# Evaluate
precision_svm = precision_score(y, y_pred_svm)
recall_svm = recall_score(y, y_pred_svm)
f1_svm = f1_score(y, y_pred_svm)

print("One-Class SVM - Precision:", precision_svm)
print("One-Class SVM - Recall:", recall_svm)
print("One-Class SVM - F1-score:", f1_svm)
```

```
One-Class SVM - Precision: 0.4984300780935512
One-Class SVM - Recall: 0.5267143100221201
One-Class SVM - F1-score: 0.512182006204757
```

In [26]:
```
# Hyperparameter grid
contamination = [y.mean(), 0.05, 0.1, 0.15]  # proportion of anomalies
n_estimators = [100, 300, 500]
max_samples = ['auto', 0.5, 0.75]
max_features = [1.0, 0.8, 0.5]

best_f1 = 0
best_params_iso = {}

for c in contamination:
    for n in n_estimators:
        for s in max_samples:
            for f in max_features:
                iso = IsolationForest(
                    n_estimators=n,
                    max_samples=s,
                    contamination=c,
                    max_features=f,
                    random_state=42
                )
                iso.fit(X_train)

                y_pred = iso.predict(X_test)
                y_pred = [1 if p == -1 else 0 for p in y_pred]  # convert -1 -> anomaly

                f1 = f1_score(y_test, y_pred)
                if f1 > best_f1:
                    best_f1 = f1
                    best_params_iso = {'n_estimators': n, 'max_samples': s, 'contamination': c, 'max_features': f}

print("Best Isolation Forest F1:", best_f1)
print("Best params:", best_params_iso)
```

```
Best Isolation Forest F1: 0.4976118106817195
Best params: {'n_estimators': 300, 'max_samples': 'auto', 'contamination': 0.47016, 'max_features': 1.0}
```

In [11]:
```
# Hyperparameter grid
nus = [y.mean(), 0.05, 0.1, 0.15]  # expected anomaly proportion
gammas = ['scale', 'auto', 0.01, 0.05, 0.1]
kernels = ['rbf', 'poly', 'sigmoid']

best_f1_svm = 0
best_params_svm = {}

# Fit on majority class only (normal patients)
X_train_normal = X_scaled[y == 0]

for nu in nus:
    for gamma in gammas:
        for kernel in kernels:
            svm = OneClassSVM(nu=nu, kernel=kernel, gamma=gamma)
            svm.fit(X_train_normal)

            y_pred = svm.predict(X_scaled)
            y_pred = [1 if p == -1 else 0 for p in y_pred]  # -1 = anomaly

            f1 = f1_score(y, y_pred)
            if f1 > best_f1_svm:
                best_f1_svm = f1
                best_params_svm = {'nu': nu, 'gamma': gamma, 'kernel': kernel}

print("Best One-Class SVM F1:", best_f1_svm)
print("Best params:", best_params_svm)
```

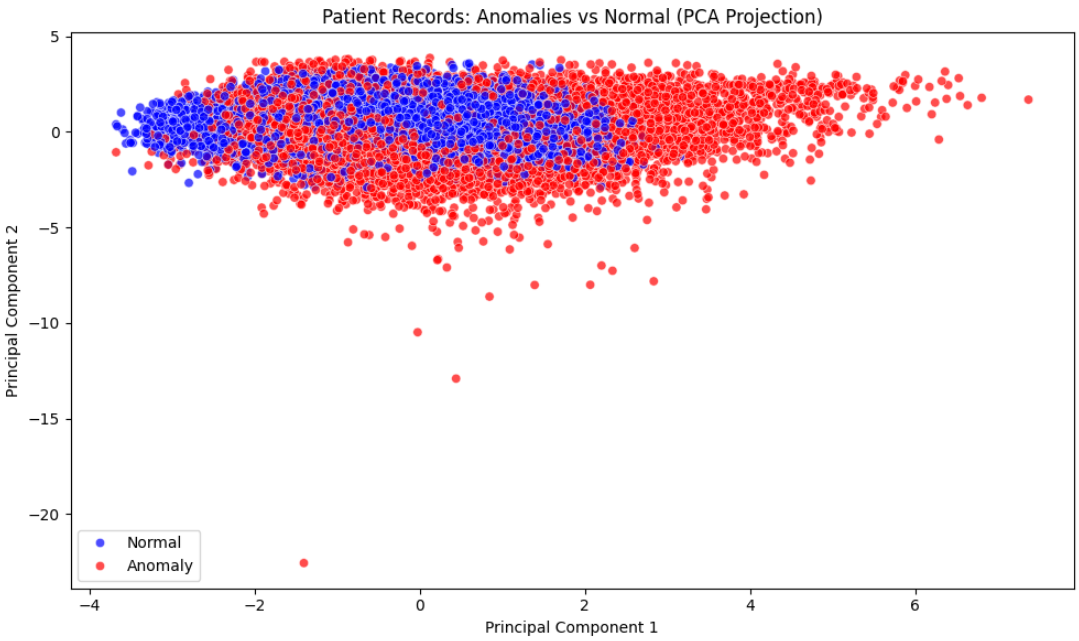
```
Best One-Class SVM F1: 0.5306422618074849
Best params: {'nu': 0.47016, 'gamma': 0.05, 'kernel': 'sigmoid'}
```

In [12]:
```
# Fit One-Class SVM on the majority class only (normal patients)
X_normal = X_scaled[y == 0]
oc_svm = OneClassSVM(nu=0.47016, kernel='sigmoid', gamma=0.05)
oc_svm.fit(X_normal)

# Predict anomalies (-1 = anomaly, 1 = normal)
y_pred_svm = oc_svm.predict(X_scaled)
y_pred_svm = [1 if p == -1 else 0 for p in y_pred_svm]  # map -1 -> anomaly

# Evaluate
precision_svm = precision_score(y, y_pred_svm)
recall_svm = recall_score(y, y_pred_svm)
f1_svm = f1_score(y, y_pred_svm)

print("One-Class SVM - Precision:", precision_svm)
print("One-Class SVM - Recall:", recall_svm)
print("One-Class SVM - F1-score:", f1_svm)
```

```
One-Class SVM - Precision: 0.5104543310800189
One-Class SVM - Recall: 0.5524927684192615
One-Class SVM - F1-score: 0.5306422618074849
```

In [13]:
```
# Anomaly plot
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Create a DataFrame for plotting
df_plot = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df_plot['Anomaly'] = y_pred_svm  # or y_pred_svm for One-Class SVM

# Map 0->Normal, 1->Anomaly
df_plot['Anomaly_Label'] = df_plot['Anomaly'].map({0:'Normal', 1:'Anomaly'})

plt.figure(figsize=(10,6))
```

```
sns.scatterplot(
    x='PC1', y='PC2',
    hue='Anomaly_Label',
    data=df_plot,
    palette={'Normal':'blue', 'Anomaly':'red'},
    alpha=0.7
)
plt.title("Patient Records: Anomalies vs Normal (PCA Projection)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend()
plt.tight_layout()
plt.show()
```



Recommended Strategies for Hospitals

1. Targeted Follow-Up Care
    - Who: High-risk patients based on model predictions (e.g., older patients, long stays, multiple procedures).
    - What: Post-discharge follow-up calls, home visits, telemedicine check-ins.
    - Impact: Early intervention can prevent complications, reducing readmissions.
2. Enhanced Discharge Planning
    - Ensure patients understand medications, wound care, and follow-up appointments.
    - Use predictive risk models to personalize discharge instructions.
    - Coordinate with primary care physicians for smooth transitions.
3. Medication Management
    - Review high-risk patients' medications to reduce polypharmacy complications.
    - Provide education on adherence and potential side effects.
4. Specialty-Specific Interventions
    - Focus on patients in high-risk medical specialties (identified by medical_specialty and diagnosis codes).
    - Tailor care plans for chronic diseases like diabetes (A1C, glucose tests) or cardiovascular conditions.
5. Outpatient Care Coordination
    - Track patients with frequent outpatient visits (n_outpatient) and provide integrated care plans to reduce unnecessary admissions.

Impact on Patient Outcomes and Hospital Efficiency

- Patient Outcomes
    - Lower readmissions -> fewer complications -> improved quality of life.
    - Personalized care reduces errors and improves adherence to treatment plans.
- Hospital Efficiency
    - Reduced readmissions -> lower costs and penalties under healthcare policies.
    - Optimized resource allocation (staffing, bed management, discharge planning).

Challenges of Deploying ML in Healthcare

1. Data Privacy
    - Healthcare data is sensitive (HIPAA/GDPR).
    - Must anonymize data and use secure storage and transfer protocols.
2. Ethical Use
    - Predictions should aid, not replace, clinician judgment.
    - Risk of bias: older patients or minority groups may be flagged more frequently — models must be audited for fairness.
3. Integration
    - Models must integrate into existing EHR systems without disrupting workflow.
    - Clinicians need interpretable explanations (why the patient is high-risk).
4. Data Quality
    - Missing or inconsistent entries (diagnosis codes, lab results) can reduce model reliability.
    - Continuous monitoring and retraining are required.

5. Patient Communication
    - Explain interventions to patients clearly to ensure cooperation.
    - Avoid stigma from being labeled high-risk.

Conclusions and Future Work

This project successfully demonstrated the feasibility of using machine learning to predict hospital readmissions and length of stay. Along with this it explore the anomalies present. The model can serve as a valuable decision support tool for clinicians and hospital administrators. For clinical integration, key considerations include:

1. Data Privacy: Strict adherence to regulations like HIPAA.
2. Ethical Use: Ensuring the model is audited for bias and does not unfairly flag certain patient groups.
3. Integration: The model must be seamlessly integrated into existing EHR systems without disrupting workflow.
4. Patient Communication: Providing clear explanations to patients to ensure cooperation with post-discharge plans and avoid the stigma of being labeled as "high-risk."

Future Work

The current project serves as a strong foundation, and several key areas can be explored to improve the model's performance, robustness, and clinical utility. Potential future work includes:

- Advanced Feature Engineering: Moving beyond the current dataset, we could engineer more complex features from detailed electronic health records (EHRs), such as specific medication dosages, full medical history, and clinical notes (using Natural Language Processing).
- Deep Learning Models: Investigating the use of deep neural networks, particularly for handling unstructured data like clinical notes or patient-generated data. Recurrent Neural Networks (RNNs) could be used to model sequences of patient visits over time.
- Advanced Anomaly Detection: Applying the anomaly detection models (One-Class SVM and Isolation Forest) to proactively flag unusual patient cases that may require closer review, such as patients with an atypical combination of diagnoses or an unusually high number of procedures.
- Hyperparameter Optimization: Implementing more rigorous hyperparameter tuning techniques (e.g., Grid Search, Random Search, or Bayesian Optimization) to find the optimal settings for each model and further improve predictive performance.
- Dynamic Risk Prediction: Developing a model that provides a real-time risk score that changes throughout a patient's stay, allowing for more dynamic and timely interventions.
- Integration with Wearable Device Data: Exploring the potential of integrating data from patient wearable devices (e.g., smartwatches) to monitor activity levels, heart rate, and other health metrics after discharge. This could provide valuable post-hospitalization insights to predict and prevent readmission.