Predicting Diabetes Patient Readmission

April 30, 2024

1 Defining the Questions

The overall question we would like to answer is if we can predict whether or not a patient is readmitted within 30 days of discharge, and, if so, what features are most important in this prediction. In order to answer this, we will be answering the following sub-questions:

1. **Comparative Analysis:** How do readmission rates differ among different demographic groups (age, gender, race)?
   - This explores whether there is a disparity in readmission rates among different patient demographics, which could guide targeted interventions.
   - By determining how readmission rates differ among different demographic groups, doctors could tailor their treatments to better suit their patients
2. **Treatment and Testing Variables:** What role do the number of lab tests performed during the stay, changes in medication, and the results of HbA1c tests play in predicting readmissions?
3. **Healthcare Utilization and Outcomes:** How do the number of emergency visits, inpatient stays, and outpatient visits in the year prior to the hospitalization correlate with readmission rates?
4. **Feature Importance Analysis:** Which factors are most predictive of readmission for diabetic patients?
   - This question aims to utilize predictive modeling to identify variables that significantly influence the likelihood of a patient being readmitted again after discharge.
   - We aim to identify the most important features in predicting whether a patient will be readmitted in order to equip medical personal with the tools needed to asses patients' situations more thoroughly.

2 Data Collection

2.1 About the Dataset

This dataset can be found here. It contains data from 10 years (1999-2008) of patient records, collected across 130 different hospitals. The dataset contains features about patients who have been diagnosed with diabetes and stayed at the hospital for up to 14 days, and whether the patient was readmitted within 30 days of discharge. This data was collected by UCI and provided by HuggingFace.

[1]:
```
from datasets import load_dataset
%matplotlib inline
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import numpy as np
import pandas as pd
import statsmodels.api as sm
import plotly.graph_objects as go
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

data_files = {"train": "dataset/train.csv", "test": "dataset/test.csv"}
dataset = load_dataset("csv", data_files=data_files)
train_df = pd.DataFrame(dataset['train'])
test_df = pd.DataFrame(dataset['test'])
```

3 Data Cleaning and Preparation

3.1 Dataset Features

This dataset contains 47 features, but we will be narrowing it down to this analysis. Our analysis will focus primarily on information about the patient such as age, weight, and gender, along with some information about the treatment they recieved while in the hospital. Specific features are below:

- age
- gender
- race
- time_in_hospital: The number of days spent in the hospital between admission and discharge
- num_procedures: Number of procedures performed during the patient's stay, excluding lab procedures
- num_lab_procedures: Number of lab procedures performed during the patient's stay
- num_medications: Number of distinct medications given during the stay
- num_outpatient: Number of outpatient visits of the patient in the year preceding the encounter
- number_inpatient: Number of inpatient visits of the patient in the year preceding the encounter
- number_emergency: Number of emergency visits of the patient in the year preceding the encounter
- A1Cresult: A1C is a measure of the level of blood sugar in a patient. This variable indicates whether a user's A1C was >8%, >7%, normal, or not measured.
- readmitted: Whether the patient was readmitted after discharge

The other features in the dataset would likely be useful for future exploration in collaboration with field experts, as many of them are quite technical.

3.2 Filtering to Features of interest

[2]:
```
features_of_interest = [
    'age:[0-10)',
    'age:[10-20)',
    'age:[20-50)',
    'age:[50-70)',
    'age:70+',
    'gender:Female',
    'gender:Male',
    'race:AfricanAmerican',
    'race:Asian',
    'race:Caucasian',
    'race:Hispanic',
    'race:Other',
    'time_in_hospital',
    'num_procedures',
    'num_lab_procedures',
    'num_medications',
    'number_outpatient',
    'number_inpatient',
    'number_emergency',
    'A1Cresult:>7',
    'A1Cresult:>8',
    'A1Cresult:Norm',
    'A1Cresult:None',
    'readmitted'
]

train_df = train_df.loc[:, features_of_interest]
test_df = test_df.loc[:, features_of_interest]

print(train_df.head())
print(test_df.head())
```

```
   age:[0-10)  age:[10-20)  age:[20-50)  age:[50-70)  age:70+  gender:Female  \
0         0.0          0.0          0.0          0.0      1.0            0.0
1         0.0          0.0          0.0          0.0      1.0            1.0
2         0.0          0.0          1.0          0.0      0.0            1.0
3         0.0          0.0          0.0          0.0      1.0            0.0
4         0.0          0.0          0.0          0.0      1.0            0.0

   gender:Male  race:AfricanAmerican  race:Asian  race:Caucasian  ...  \
0          1.0                   0.0         0.0             1.0  ...
1          0.0                   0.0         0.0             1.0  ...
2          0.0                   1.0         0.0             0.0  ...
3          1.0                   0.0         0.0             1.0  ...
4          1.0                   0.0         0.0             1.0  ...
```

```
   num_lab_procedures  num_medications  number_outpatient  number_inpatient  \
0                38.0             27.0                0.0               2.0
1                48.0             11.0                0.0               0.0
2                28.0             15.0                0.0               4.0
3                44.0             10.0                0.0               0.0
4                54.0              8.0                0.0               0.0

   number_emergency  A1Cresult:>7  A1Cresult:>8  A1Cresult:Norm  \
0               1.0           0.0           0.0             0.0
1               0.0           0.0           0.0             1.0
2               3.0           0.0           0.0             0.0
3               0.0           0.0           0.0             0.0
4               0.0           0.0           0.0             0.0

   A1Cresult:None  readmitted
0             1.0           0
1             0.0           0
2             1.0           1
3             1.0           0
4             1.0           0

[5 rows x 24 columns]
   age:[0-10)  age:[10-20)  age:[20-50)  age:[50-70)  age:70+  gender:Female  \
0         0.0          0.0          0.0          0.0      1.0            1.0
1         0.0          0.0          1.0          0.0      0.0            1.0
2         0.0          0.0          0.0          1.0      0.0            1.0
3         0.0          0.0          1.0          0.0      0.0            1.0
4         0.0          0.0          0.0          1.0      0.0            0.0

   gender:Male  race:AfricanAmerican  race:Asian  race:Caucasian  ...  \
0          0.0                   0.0         0.0             1.0  ...
1          0.0                   0.0         0.0             1.0  ...
2          0.0                   0.0         0.0             1.0  ...
3          0.0                   0.0         0.0             1.0  ...
4          1.0                   0.0         0.0             0.0  ...

   num_lab_procedures  num_medications  number_outpatient  number_inpatient  \
0                66.0             18.0                0.0               1.0
1                48.0             15.0                4.0               0.0
2                21.0             23.0                1.0               2.0
3                38.0              5.0                0.0               0.0
4                 6.0              6.0                0.0               0.0

   number_emergency  A1Cresult:>7  A1Cresult:>8  A1Cresult:Norm  \
0               0.0           0.0           0.0             0.0
1               0.0           0.0           0.0             0.0
2               0.0           0.0           0.0             0.0
```

```
3                0.0           0.0           1.0             0.0
4                0.0           0.0           0.0             0.0

   A1Cresult:None  readmitted
0             1.0           1
1             1.0           0
2             1.0           0
3             0.0           0
4             1.0           1

[5 rows x 24 columns]
```

3.3 Checking for missing values

Our data was already cleaned by HuggingFace for missing values, but just to be sure, we went ahead and checked for any null values for all rows and columns.

[3]:
```
# Check for missing values in each column of the training dataset
missing_values_train = train_df.isnull().sum()
missing_values_train

# Check for missing values in each row of the training dataset
missing_values_per_row_train = train_df.isnull().sum(axis=1)
missing_values_per_row_train

# Check for missing values in each column of the testing dataset
missing_values_test = test_df.isnull().sum()
missing_values_test

# Check for missing values in each row of the testing dataset
missing_values_per_row_test = test_df.isnull().sum(axis=1)
missing_values_per_row_test
```

[3]:
```
age:[0-10)              0
age:[10-20)             0
age:[20-50)             0
age:[50-70)             0
age:70+                 0
gender:Female           0
gender:Male             0
race:AfricanAmerican    0
race:Asian              0
race:Caucasian          0
race:Hispanic           0
race:Other              0
time_in_hospital        0
num_procedures          0
num_lab_procedures      0
```

```
num_medications         0
number_outpatient       0
number_inpatient        0
number_emergency        0
A1Cresult:>7            0
A1Cresult:>8            0
A1Cresult:Norm          0
A1Cresult:None          0
readmitted              0
dtype: int64
```

[3]:
```
0        0
1        0
2        0
3        0
4        0
        ..
81405    0
81406    0
81407    0
81408    0
81409    0
Length: 81410, dtype: int64
```

[3]:
```
age:[0-10)              0
age:[10-20)             0
age:[20-50)             0
age:[50-70)             0
age:70+                 0
gender:Female           0
gender:Male             0
race:AfricanAmerican    0
race:Asian              0
race:Caucasian          0
race:Hispanic           0
race:Other              0
time_in_hospital        0
num_procedures          0
num_lab_procedures      0
num_medications         0
number_outpatient       0
number_inpatient        0
number_emergency        0
A1Cresult:>7            0
A1Cresult:>8            0
A1Cresult:Norm          0
A1Cresult:None          0
```

```
readmitted              0
dtype: int64
```

[3]:
```
0        0
1        0
2        0
3        0
4        0
        ..
20348    0
20349    0
20350    0
20351    0
20352    0
Length: 20353, dtype: int64
```

4 Data Analysis

4.1 Comparative Analysis

How do readmission rates differ among different demographic groups (age, gender, race)? - This explores whether there is a disparity in readmission rates among different patient demographics, which could guide targeted interventions.

Race Demographic Data

This code calculates the sum of data points for each racial demographic within the two datasets, aiming to facilitate a comparative analysis. The objective is to discern potential trends in racial demographics concerning hospital readmissions attributed to diabetes.

[4]:
```
def race_patients():
    import matplotlib.pyplot as plt
    import pandas as pd

    # Summing up the race demographic counts
    race_africanAmerican_sum = train_df['race:AfricanAmerican'].sum()
    race_asian_sum = train_df['race:Asian'].sum()
    race_caucasian_sum = train_df['race:Caucasian'].sum()
    race_hispanic_sum = train_df['race:Hispanic'].sum()
    race_other_sum = train_df['race:Other'].sum()

    sums = [race_asian_sum, race_caucasian_sum, race_hispanic_sum,␣
↪race_africanAmerican_sum, race_other_sum]
    labels = ['Asian', 'Caucasian', 'Hispanic', 'African American', 'Other']

    plt.figure(figsize=(10, 7))
    plt.pie(sums, labels=labels, autopct='%1.1f%%', startangle=140,␣
↪pctdistance=0.85)
```

```
    plt.title('Race Demographics of Patients')
    plt.legend(labels, title="Races", loc="best")
    plt.show()
```

[5]:
```
def show_race_demographics_pie_chart():
    import matplotlib.pyplot as plt

    # Sum of 'readmitted' values for each race demographic of the patients in␣
↪the training dataset
    train_readmittedAA_sum = train_df[train_df['race:AfricanAmerican'] ==␣
↪1]['readmitted'].sum()
    train_readmittedA_sum = train_df[train_df['race:Asian'] == 1]['readmitted'].
↪sum()
    train_readmittedC_sum = train_df[train_df['race:Caucasian'] ==␣
↪1]['readmitted'].sum()
    train_readmittedH_sum = train_df[train_df['race:Hispanic'] ==␣
↪1]['readmitted'].sum()
    train_readmittedO_sum = train_df[train_df['race:Other'] == 1]['readmitted'].
↪sum()

    sums = [train_readmittedA_sum ,
            train_readmittedC_sum ,
            train_readmittedH_sum ,
            train_readmittedAA_sum ,
            train_readmittedO_sum ]

    labels = ['Asian', 'Caucasian', 'Hispanic', 'African American', 'Other']

    plt.figure(figsize=(10, 7))
    plt.pie(sums, labels=labels, autopct='%1.1f%%', startangle=140,␣
↪pctdistance=0.85)
    plt.title('Race Demographics of Readmitted Patients')
    plt.legend(labels, title="Races", loc="best")
    plt.show()
```

Gender Demographic Data

This code computes the sum of data points for the gender demographic across the two datasets, with the goal of enabling a comparative analysis. The objective is to explore whether gender plays a role in readmission rates.

[6]:
```
def gender_patients():
    import matplotlib.pyplot as plt
    import pandas as pd

    # Assuming 'train' is your DataFrame and it contains columns 'gender:Female'␣
↪and 'gender:Male' with binary      values (1 for presence, 0 for absence)
```

```
    # Summing up the gender counts
    gender_female_sum = train_df['gender:Female'].sum()
    gender_male_sum = train_df['gender:Male'].sum()

    sums = [gender_female_sum, gender_male_sum]
    labels = ['Female', 'Male']

    # The colors for each section of the pie chart
    colors = ['#ff9999','#66b3ff']

    plt.figure(figsize=(5, 5))
    plt.pie(sums, labels=labels, autopct='%1.1f%%', startangle=90, pctdistance=0.
↪85)
    plt.title('Gender Demographics of Patients')
    plt.legend(labels, title="Gender", loc="best")
    plt.show()
```

[7]:
```
def show_gender_readmission_demographics_pie_chart():

    # Sum of 'readmitted' values where 'gender:Female' equals 1 for the training␣
↪dataset
    train_readmittedF_sum = train_df[train_df['gender:Female'] ==␣
↪1]['readmitted'].sum()
    train_readmittedM_sum = train_df[train_df['gender:Male'] == 1]['readmitted'].
↪sum()

    # Labels for the sections of our pie chart
    labels = 'Female', 'Male'

    # The values for each section of the pie chart
    sizes = [train_readmittedF_sum, train_readmittedM_sum]

    # The colors for each section of the pie chart
    colors = ['#ff9999','#66b3ff']

    # Plotting the pie chart
    plt.figure(figsize=(5, 5))
    plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%',␣
↪startangle=90)
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a␣
↪circle.
    plt.title('Gender Demographics of Readmitted Patients')
    plt.show()
```

Age Demographic Data

This code computes the sum of data points for the age demographic across the two datasets to compare the readmission rates in each group.

[8]:
```
def age_patients():
    import matplotlib.pyplot as plt

    # Summing up the age demographic counts
    age_70plus_sum = train_df['age:70+'].sum()
    age_less10_sum = train_df['age:[0-10)'].sum()
    age_10to19_sum = train_df['age:[10-20)'].sum()
    age_20to49_sum = train_df['age:[20-50)'].sum()
    age_50to69_sum = train_df['age:[50-70)'].sum()

    # Combining the 0-10 and 10-20 age groups because their percentages do not␣
 ↪have a significant difference        between them
    sums = [(age_less10_sum + age_10to19_sum), age_20to49_sum, age_50to69_sum,␣
 ↪age_70plus_sum]
    age_labels = ['[0,20)', '[20,50)', '[50,70)', '70+']

    plt.figure(figsize=(10, 7))
    plt.pie(sums, labels=age_labels, autopct='%1.1f%%')
    plt.title('Age Demographics of Patients')
    plt.legend()
    plt.show()
```

[9]:
```
def show_age_demographics_pie_chart():

    # Sum of 'readmitted' values for each age group for the training dataset
    train_readmitted70plus_sum = train_df[train_df['age:70+'] ==␣
 ↪1]['readmitted'].sum()
    train_readmittedLess10_sum = train_df[train_df['age:[0-10)'] ==␣
 ↪1]['readmitted'].sum()
    train_readmitted10to19_sum = train_df[train_df['age:[10-20)'] ==␣
 ↪1]['readmitted'].sum()
    train_readmitted20to49_sum = train_df[train_df['age:[20-50)'] ==␣
 ↪1]['readmitted'].sum()
    train_readmitted50to69_sum = train_df[train_df['age:[50-70)'] ==␣
 ↪1]['readmitted'].sum()

    # Combining the 0-10 and 10-20 age groups because their percentages do not␣
 ↪have a significant difference between them
    age_demographics_sums = [
        train_readmittedLess10_sum + train_readmitted10to19_sum,
        train_readmitted20to49_sum,
        train_readmitted50to69_sum,
        train_readmitted70plus_sum
    ]

    age_labels = ['[0,20)', '[20,50)', '[50,70)', '70+']
```

```
    plt.figure(figsize=(10, 7))
    plt.pie(age_demographics_sums, labels=age_labels, autopct='%1.1f%%')
    plt.title('Age Demographics of Readmitted Patients')
    plt.legend()
    plt.show()
```

4.2 Treatment and Testing Variables

What role do the number of lab tests performed during the stay, changes in medication, and the results of HbA1c tests play in predicting readmissions?

We introduced a new variable, A1Cresult_combined, to simplify the dataset by aggregating specific A1C results into broader categories, allowing us to better analyze the relationship between A1C results and patient readmission rates.

[10]:
```
# Create an A1Cresult_combined variable based on the specific A1Cresult columns
train_df['A1Cresult_combined'] = 'Unknown'  # Default value

# Update the A1Cresult_combined based on specific conditions
train_df.loc[train_df['A1Cresult:>7'] == 1, 'A1Cresult_combined'] = 'Normal'
train_df.loc[train_df['A1Cresult:>8'] == 1, 'A1Cresult_combined'] = 'Normal'
train_df.loc[train_df['A1Cresult:None'] == 1, 'A1Cresult_combined'] = 'Abnormal'
train_df.loc[train_df['A1Cresult:Norm'] == 1, 'A1Cresult_combined'] = 'Abnormal'

# Aggregate data for visualization
agg_lab = train_df.groupby(['num_lab_procedures', 'readmitted']).size().
 ↪reset_index(name='count')
agg_med = train_df.groupby(['num_medications', 'readmitted']).size().
 ↪reset_index(name='count')
agg_hba1c = train_df.groupby(['A1Cresult_combined', 'readmitted']).size().
 ↪reset_index(name='count')

# find mode for later use
mode_agg_lab = agg_lab.groupby('num_lab_procedures')['count'].sum().idxmax()
mode_agg_med = agg_med.groupby('num_medications')['count'].sum().idxmax()
mode_agg_hba1c = agg_hba1c.groupby('A1Cresult_combined')['count'].sum().idxmax()

# Define a function to calculate mean while dropping outliers
def calculate_mean_without_outliers(data, column_name, readmitted_status,␣
 ↪lower_percentile=5, upper_percentile=95):
    # Calculate the percentile cutoffs
    filtered_data = data[data['readmitted'] == readmitted_status]
    lower_cutoff = np.percentile(filtered_data[column_name], lower_percentile)
    upper_cutoff = np.percentile(filtered_data[column_name], upper_percentile)

    # Filter data to exclude outliers
```

```
    filtered_data = filtered_data[(filtered_data[column_name] >= lower_cutoff) &␣
↪(filtered_data[column_name] <= upper_cutoff)]

    # Calculate mean of the filtered data
    mean_value = filtered_data[column_name].mean()

    return mean_value
```

4.2.1 Number of Lab Tests Performed

[11]:
```
from scipy.stats import ttest_ind
# Lab Procedures vs. Readmission
# Calculate mean of 'num_lab_procedures' while dropping outliers
mean_lab_procedures_no = calculate_mean_without_outliers(train_df,␣
↪'num_lab_procedures', 0, lower_percentile=5, upper_percentile=95)
mean_lab_procedures_yes = calculate_mean_without_outliers(train_df,␣
↪'num_lab_procedures', 1, lower_percentile=5, upper_percentile=95)

def filter_data_without_outliers(data, column_name, readmitted_status,␣
↪lower_percentile=5, upper_percentile=95):
    # Calculate the percentile cutoffs
    filtered_data = data[data['readmitted'] == readmitted_status]
    lower_cutoff = np.percentile(filtered_data[column_name], lower_percentile)
    upper_cutoff = np.percentile(filtered_data[column_name], upper_percentile)

    # Filter data to exclude outliers
    filtered_data = filtered_data[(filtered_data[column_name] >= lower_cutoff) &␣
↪(filtered_data[column_name] <= upper_cutoff)]

    return filtered_data

# t-test for Lab Procedures vs. Readmission
# Filter data for 'num_lab_procedures' by readmission status
filtered_data_no = filter_data_without_outliers(train_df, 'num_lab_procedures',␣
↪0)
filtered_data_yes = filter_data_without_outliers(train_df, 'num_lab_procedures',␣
↪1)

# Extract the relevant column for the t-test
data_no = filtered_data_no['num_lab_procedures']
data_yes = filtered_data_yes['num_lab_procedures']

# Perform t-test
t_statistic, p_value = ttest_ind(data_no, data_yes)


def labProcedures_v_readmission_chart():
```

```
    fig_lab = go.Figure()
    for readmitted_status in agg_lab['readmitted'].unique():
        if readmitted_status == 0:
            label = 'Not Readmitted'
        else:
            label = 'Readmitted'
        filtered_data = agg_lab[agg_lab['readmitted'] == readmitted_status]
        fig_lab.add_trace(go.Bar(x=filtered_data['num_lab_procedures'],␣
↪y=filtered_data['count'], name=label))

    fig_lab.update_layout(title='Lab Procedures vs. Readmission',␣
↪xaxis_title='Number of Lab Procedures', yaxis_title='Count', barmode='group')
    fig_lab.show()
```

4.2.2 Changes in Medication

[12]:
```
# Medications vs. Readmission
def Medications_v_readmission_chart():
    fig_med = go.Figure()
    for readmitted_status in agg_med['readmitted'].unique():
        if readmitted_status == 0:
            label = 'Not Readmitted'
        else:
            label = 'Readmitted'
        filtered_data = agg_med[agg_med['readmitted'] == readmitted_status]
        fig_med.add_trace(go.Bar(x=filtered_data['num_medications'],␣
↪y=filtered_data['count'], name=label))

    fig_med.update_layout(title='Medications vs. Readmission',␣
↪xaxis_title='Number of Medications',         yaxis_title='Count',␣
↪barmode='group')
    fig_med.show()
```

4.2.3 HbA1c Test Results

[13]:
```
# HbA1c Test Results vs. Readmission
def HbA1c_v_readmission_chart():
    fig_hba1c = go.Figure()
    for readmitted_status in agg_hba1c['readmitted'].unique():
        if readmitted_status == 0:
            label = 'Not Readmitted'
        else:
            label = 'Readmitted'
        filtered_data = agg_hba1c[agg_hba1c['readmitted'] == readmitted_status]
        fig_hba1c.add_trace(go.Bar(x=filtered_data['A1Cresult_combined'],␣
↪y=filtered_data['count'],             name=label))
```

```
    fig_hba1c.update_layout(title='HbA1c Test Results vs. Readmission',␣
↪xaxis_title='A1C Result Combined',      yaxis_title='Count', barmode='group')
    fig_hba1c.show()
```

4.3 Healthcare Utilization and Outcomes

How do the number of emergency visits, inpatient stays, and outpatient visits in the year prior to the hospitalization correlate with readmission rates?

4.3.1 Overview of Stays

[14]:
```
# Set the aesthetic style of the plots
sns.set_style("whitegrid")

# Create box and whisker plots to visualize the relationships
plt.figure(figsize=(18, 6))

# Create a copy of the dataframe and map the 'readmitted' values on the copy
plot_data = train_df.copy()
plot_data['readmitted'] = plot_data['readmitted'].map({0: 'No', 1: 'Yes'})

plt.subplot(1, 3, 1)
sns.boxplot(x='readmitted', y='number_emergency', data=plot_data,␣
 ↪palette="muted")
plt.title('Emergency Visits vs. Readmission')

plt.subplot(1, 3, 2)
sns.boxplot(x='readmitted', y='number_outpatient', data=plot_data,␣
 ↪palette="muted")
plt.title('Outpatient Visits vs. Readmission')

plt.subplot(1, 3, 3)
sns.boxplot(x='readmitted', y='number_inpatient', data=plot_data,␣
 ↪palette="muted")
plt.title('Inpatient Visits vs. Readmission')

plt.show()
```

[14]:
```
<Figure size 1800x600 with 0 Axes>
```

[14]:
```
<Axes: >
```

```
/var/folders/dy/_pzrctxj2fdd91qltk8lbfhr0000gn/T/ipykernel_51151/644874333.py:12
: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```

```
  sns.boxplot(x='readmitted', y='number_emergency', data=plot_data,
palette="muted")
```

[14]: <Axes: xlabel='readmitted', ylabel='number_emergency'>

[14]: Text(0.5, 1.0, 'Emergency Visits vs. Readmission')

[14]: <Axes: >

```
/var/folders/dy/_pzrctxj2fdd91qltk8lbfhr0000gn/T/ipykernel_51151/644874333.py:16
: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x='readmitted', y='number_outpatient', data=plot_data,
palette="muted")
```

[14]: <Axes: xlabel='readmitted', ylabel='number_outpatient'>

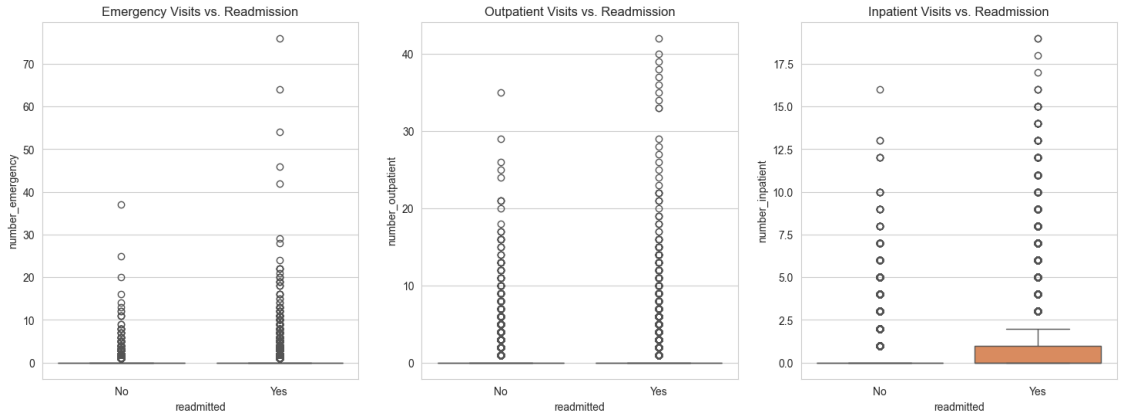
[14]: Text(0.5, 1.0, 'Outpatient Visits vs. Readmission')

[14]: <Axes: >

```
/var/folders/dy/_pzrctxj2fdd91qltk8lbfhr0000gn/T/ipykernel_51151/644874333.py:20
: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x='readmitted', y='number_inpatient', data=plot_data,
palette="muted")
```

[14]: <Axes: xlabel='readmitted', ylabel='number_inpatient'>

[14]: Text(0.5, 1.0, 'Inpatient Visits vs. Readmission')



Since the vast majority of people didn't have any emergency visits, outpatient visits, or inpatient visits, lets remove them from the plots.

[15]:
```
# Set the aesthetic style of the plots
sns.set_style("whitegrid")

# Create box and whisker plots to visualize the relationships without showing␣
 ↪outliers
plt.figure(figsize=(18, 12))

cleaned_data = train_df[['number_emergency', 'number_outpatient',␣
 ↪'number_inpatient', 'readmitted']].copy()

cleaned_data = cleaned_data[cleaned_data['number_emergency'] > 0]
cleaned_data = cleaned_data[cleaned_data['number_outpatient'] > 0]
cleaned_data = cleaned_data[cleaned_data['number_inpatient'] > 0]
cleaned_data['readmitted'] = cleaned_data['readmitted'].map({0: 'No', 1: 'Yes'})

plt.subplot(2, 3, 1)
sns.boxplot(x='readmitted', y='number_emergency', data=cleaned_data,␣
 ↪palette="muted")
plt.title('Emergency Visits vs. Readmission')

plt.subplot(2, 3, 2)
sns.boxplot(x='readmitted', y='number_outpatient', data=cleaned_data,␣
 ↪palette="muted")
plt.title('Outpatient Visits vs. Readmission')

plt.subplot(2, 3, 3)
sns.boxplot(x='readmitted', y='number_inpatient', data=cleaned_data,␣
 ↪palette="muted")
plt.title('Inpatient Visits vs. Readmission')
```

```
plt.subplot(2, 3, 4)
sns.boxplot(x='readmitted', y='number_emergency', data=cleaned_data,␣
 ↪palette="muted", showfliers=False)
plt.title('Emergency Visits vs. Readmission (Outliers Removed)')

plt.subplot(2, 3, 5)
sns.boxplot(x='readmitted', y='number_outpatient', data=cleaned_data,␣
 ↪palette="muted", showfliers=False)
plt.title('Outpatient Visits vs. Readmission (Outliers Removed)')

plt.subplot(2, 3, 6)
sns.boxplot(x='readmitted', y='number_inpatient', data=cleaned_data,␣
 ↪palette="muted", showfliers=False)
plt.title('Inpatient Visits vs. Readmission (Outliers Removed)')

plt.show()
```

[15]: <Figure size 1800x1200 with 0 Axes>

[15]: <Axes: >

```
/var/folders/dy/_pzrctxj2fdd91qltk8lbfhr0000gn/T/ipykernel_51151/3828681882.py:1
5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x='readmitted', y='number_emergency', data=cleaned_data,
palette="muted")
```

[15]: <Axes: xlabel='readmitted', ylabel='number_emergency'>

[15]: Text(0.5, 1.0, 'Emergency Visits vs. Readmission')

[15]: <Axes: >

```
/var/folders/dy/_pzrctxj2fdd91qltk8lbfhr0000gn/T/ipykernel_51151/3828681882.py:1
9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x='readmitted', y='number_outpatient', data=cleaned_data,
palette="muted")
```

```
[15]: <Axes: xlabel='readmitted', ylabel='number_outpatient'>

[15]: Text(0.5, 1.0, 'Outpatient Visits vs. Readmission')

[15]: <Axes: >

      /var/folders/dy/_pzrctxj2fdd91qltk8lbfhr0000gn/T/ipykernel_51151/3828681882.py:2
      3: FutureWarning:

      Passing `palette` without assigning `hue` is deprecated and will be removed in
      v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
      effect.

        sns.boxplot(x='readmitted', y='number_inpatient', data=cleaned_data,
      palette="muted")

[15]: <Axes: xlabel='readmitted', ylabel='number_inpatient'>

[15]: Text(0.5, 1.0, 'Inpatient Visits vs. Readmission')

[15]: <Axes: >

      /var/folders/dy/_pzrctxj2fdd91qltk8lbfhr0000gn/T/ipykernel_51151/3828681882.py:2
      7: FutureWarning:

      Passing `palette` without assigning `hue` is deprecated and will be removed in
      v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
      effect.

        sns.boxplot(x='readmitted', y='number_emergency', data=cleaned_data,
      palette="muted", showfliers=False)

[15]: <Axes: xlabel='readmitted', ylabel='number_emergency'>

[15]: Text(0.5, 1.0, 'Emergency Visits vs. Readmission (Outliers Removed)')

[15]: <Axes: >

      /var/folders/dy/_pzrctxj2fdd91qltk8lbfhr0000gn/T/ipykernel_51151/3828681882.py:3
      1: FutureWarning:

      Passing `palette` without assigning `hue` is deprecated and will be removed in
      v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
      effect.

        sns.boxplot(x='readmitted', y='number_outpatient', data=cleaned_data,
      palette="muted", showfliers=False)

[15]: <Axes: xlabel='readmitted', ylabel='number_outpatient'>
```

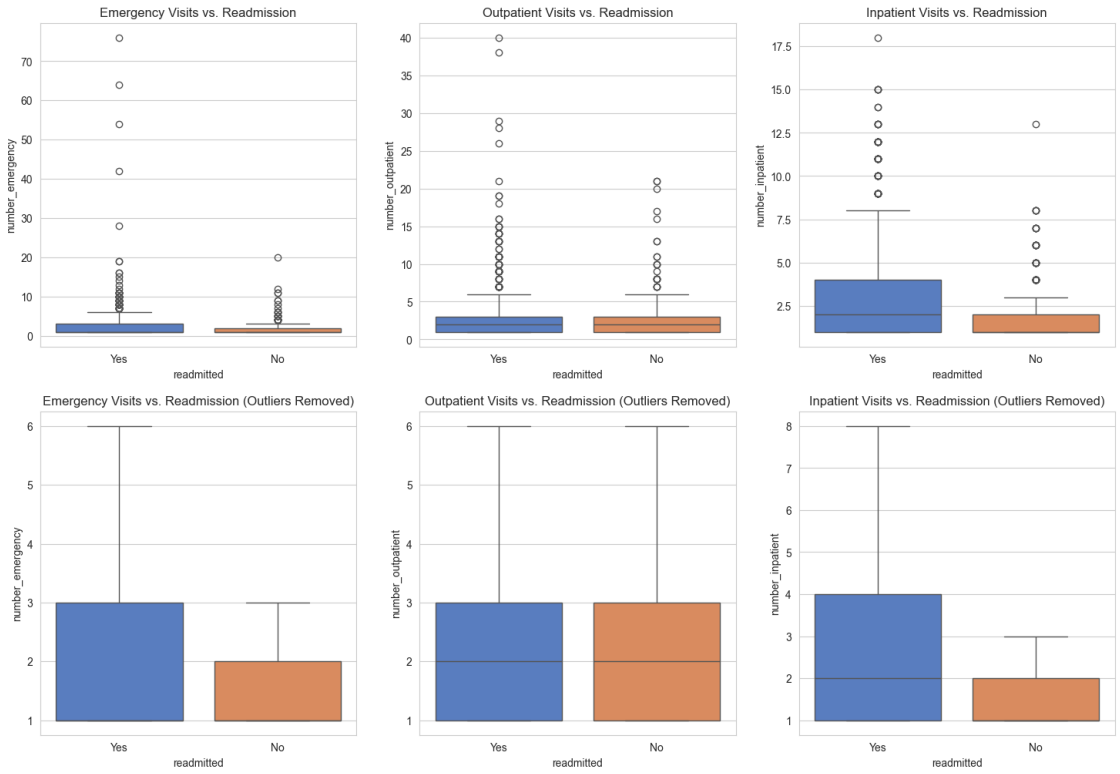
[15]: Text(0.5, 1.0, 'Outpatient Visits vs. Readmission (Outliers Removed)')

[15]: <Axes: >

```
/var/folders/dy/_pzrctxj2fdd91qltk8lbfhr0000gn/T/ipykernel_51151/3828681882.py:3
5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x='readmitted', y='number_inpatient', data=cleaned_data,
palette="muted", showfliers=False)
```

[15]: <Axes: xlabel='readmitted', ylabel='number_inpatient'>

[15]: Text(0.5, 1.0, 'Inpatient Visits vs. Readmission (Outliers Removed)')



There still doesn't seem to be lots of correlation, but we'll fit a Logistic Regression model and a Random Forest Classifier model to be sure.

1. How emergency visits correlate with readmission
2. How inpatient stays correlate with readmission
3. How outpatient visits correlate with readmission

4. How all three correlate

4.3.2 Emergency Visits vs. Readmission Models

The logistic regression model reveals a moderate accuracy of approximately 57% in predicting hospital readmissions based on emergency visits, suggesting a partial but not strong predictive relationship. The confusion matrix highlights that the model is more effective at identifying non-readmissions with a recall of 0.92 for class 0, yet it significantly underperforms in predicting actual readmissions, with a recall of only 0.15 for class 1. This indicates that emergency visits alone may not be a robust predictor for readmissions.

The random forest classifier demonstrates an improved accuracy of approximately 63% in predicting hospital readmissions based on emergency visits, indicating a slightly better correlation than the logistic regression model. The confusion matrix reveals that the model has a reasonable balance in predicting both non-readmissions and readmissions, with recall rates of 0.74 and 0.52 for classes 0 and 1, respectively. These results suggest that the random forest model, while better at balancing prediction across classes compared to logistic regression, still shows room for improvement, particularly in accurately identifying readmissions.

[16]:
```
X_train = train_df[['number_emergency']]
y_train = train_df['readmitted']

X_test = test_df[['number_emergency']]
y_test = test_df['readmitted']

# Initializing the Logistic Regression model
emer_log_reg = LogisticRegression()

# Fitting the model with the training data
emer_log_reg.fit(X_train, y_train)

# Predicting the test set results
y_pred = emer_log_reg.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Displaying the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

[16]:
```
LogisticRegression()
```

```
Accuracy: 0.5717093303198546
Confusion Matrix:
 [[10189   827]
```

```
 [ 7890  1447]]
Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.92      0.70     11016
           1       0.64      0.15      0.25      9337

    accuracy                           0.57     20353
   macro avg       0.60      0.54      0.47     20353
weighted avg       0.60      0.57      0.49     20353
```

[17]:
```python
from sklearn.ensemble import RandomForestClassifier

# Initializing the Random Forest Classifier model
emer_rf = RandomForestClassifier()

# Fitting the model with the training data
emer_rf.fit(X_train, y_train)

# Predicting the test set results
y_pred = emer_rf.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Displaying the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

[17]:
```
RandomForestClassifier()
```

```
Accuracy: 0.5717584631258291
Confusion Matrix:
 [[10190   826]
 [ 7890  1447]]
Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.93      0.70     11016
           1       0.64      0.15      0.25      9337

    accuracy                           0.57     20353
   macro avg       0.60      0.54      0.47     20353
weighted avg       0.60      0.57      0.49     20353
```

4.3.3 Inpatient Visits vs. Readmission Models

Both models achieved an identical accuracy of 61.75%, suggesting a moderate ability to predict readmission based on the number of inpatient visits. Although each model demonstrated similar precision, recall, and f1-scores across the classes, it's notable that neither method significantly outperformed the other, indicating that the predictive power of the number of inpatient visits on readmission likelihood might be limited, regardless of the model used.

[18]:
```
X_train = train_df[['number_inpatient']]
y_train = train_df['readmitted']

X_test = test_df[['number_inpatient']]
y_test = test_df['readmitted']

# Initializing the Logistic Regression model
inpat_log_reg = LogisticRegression()

# Fitting the model with the training data
inpat_log_reg.fit(X_train, y_train)

# Predicting the test set results
y_pred = inpat_log_reg.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Displaying the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

[18]:
```
LogisticRegression()
```

```
Accuracy: 0.6175011054881344
Confusion Matrix:
 [[8376 2640]
 [5145 4192]]
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.76      0.68     11016
           1       0.61      0.45      0.52      9337

    accuracy                           0.62     20353
   macro avg       0.62      0.60      0.60     20353
```

```
weighted avg       0.62      0.62      0.61     20353
```

[19]:
```
X_train = train_df[['number_inpatient']]
y_train = train_df['readmitted']

X_test = test_df[['number_inpatient']]
y_test = test_df['readmitted']

# Initializing the Random Forest Classifier model
inpat_rf = RandomForestClassifier()

# Fitting the model with the training data
inpat_rf.fit(X_train, y_train)

# Predicting the test set results
y_pred = inpat_rf.predict(X_test)

# Evaluating the model
accuracy_rf = accuracy_score(y_test, y_pred)
conf_matrix_rf = confusion_matrix(y_test, y_pred)
class_report_rf = classification_report(y_test, y_pred)

# Displaying the results
print("Accuracy:", accuracy_rf)
print("Confusion Matrix:\n", conf_matrix_rf)
print("Classification Report:\n", class_report_rf)
```

[19]:
```
RandomForestClassifier()
```

```
Accuracy: 0.6175011054881344
Confusion Matrix:
 [[8376 2640]
 [5145 4192]]
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.76      0.68     11016
           1       0.61      0.45      0.52      9337

    accuracy                           0.62     20353
   macro avg       0.62      0.60      0.60     20353
weighted avg       0.62      0.62      0.61     20353
```

4.3.4 Outpatient visits vs. readmission models

The Logistic Regression model achieved an accuracy of approximately 55.7%, with a notably high recall of 94% for predicting non-readmissions, suggesting it is conservative in predicting readmis-

sions. Conversely, the Random Forest model slightly outperformed Logistic Regression in overall accuracy with about 57.2% and demonstrated a more balanced performance across the metrics with a recall of 21% for readmissions. This suggests that the Random Forest model, while still limited, may provide a more nuanced understanding of the factors influencing readmissions compared to the Logistic Regression model. Both models, however, show moderate effectiveness, indicating a more complex relationship.

[20]:
```
X_train = train_df[['number_outpatient']]
y_train = train_df['readmitted']

X_test = test_df[['number_outpatient']]
y_test = test_df['readmitted']

# Initializing the Logistic Regression model
outpat_log_reg = LogisticRegression()

# Fitting the model with the training data
outpat_log_reg.fit(X_train, y_train)

# Predicting the test set results
y_pred = outpat_log_reg.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Displaying the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

[20]:
```
LogisticRegression()
```

```
Accuracy: 0.5572151525573625
Confusion Matrix:
 [[10375   641]
 [ 8371   966]]
Classification Report:
              precision    recall  f1-score   support

           0       0.55      0.94      0.70     11016
           1       0.60      0.10      0.18      9337

    accuracy                           0.56     20353
   macro avg       0.58      0.52      0.44     20353
weighted avg       0.58      0.56      0.46     20353
```

[21]:
```
# Initializing the Random Forest Classifier model
outpat_rf = RandomForestClassifier()

# Fitting the model with the training data
outpat_rf.fit(X_train, y_train)

# Predicting the test set results
y_pred = outpat_rf.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Displaying the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

[21]:
```
RandomForestClassifier()
```

```
Accuracy: 0.5720041271557018
Confusion Matrix:
 [[9675 1341]
 [7370 1967]]
Classification Report:
               precision    recall  f1-score   support

           0       0.57      0.88      0.69     11016
           1       0.59      0.21      0.31      9337

    accuracy                           0.57     20353
   macro avg       0.58      0.54      0.50     20353
weighted avg       0.58      0.57      0.52     20353
```

4.3.5 All Three Visits vs. Readmission Models

The modest accuracy scores from both models (approximately 61-62%) indicate that while there is a correlation between the number of visits and readmissions, it is not strong enough to predict readmissions with high reliability.

Furthermore, the distribution of performance metrics like precision, recall, and f1-score across both models suggests variability in how well different types of visits predict readmissions. For instance, the generally lower recall for predicting readmissions (particularly noted in the Logistic Regression model) implies that many actual readmissions are not being captured by the models.

[22]:
```
X_train = train_df[['number_inpatient', 'number_outpatient', 'number_emergency']]
y_train = train_df['readmitted']

X_test = test_df[['number_inpatient', 'number_outpatient', 'number_emergency']]
y_test = test_df['readmitted']

# Initializing the Logistic Regression model
all_log_reg = LogisticRegression()

# Fitting the model with the training data
all_log_reg.fit(X_train, y_train)

# Predicting the test set results
y_pred = all_log_reg.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Displaying the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

[22]:
```
LogisticRegression()
```

```
Accuracy: 0.6123421608608067
Confusion Matrix:
 [[9508 1508]
 [6382 2955]]
Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.86      0.71     11016
           1       0.66      0.32      0.43      9337

    accuracy                           0.61     20353
   macro avg       0.63      0.59      0.57     20353
weighted avg       0.63      0.61      0.58     20353
```

[23]:
```
# Initializing the Random Forest Classifier model
all_rf = RandomForestClassifier()

# Fitting the model with the training data
all_rf.fit(X_train, y_train)
```

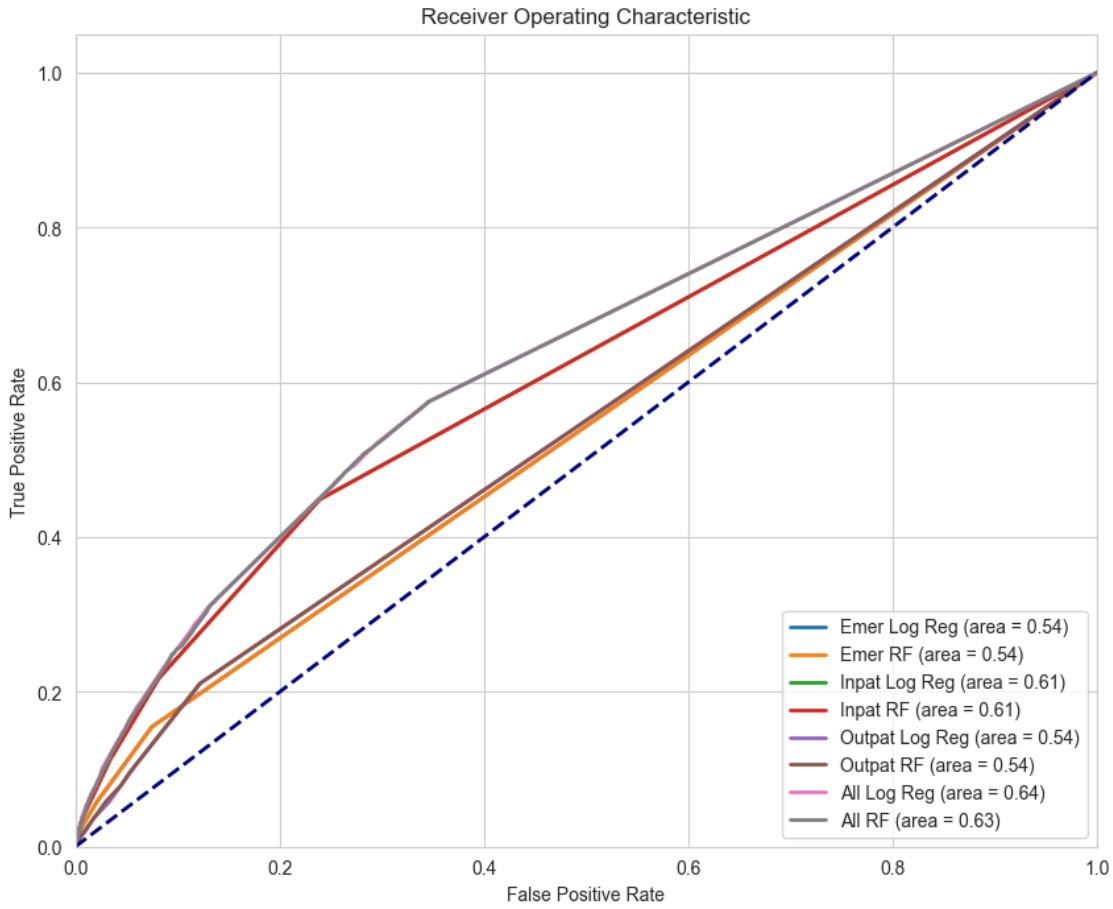
```
# Predicting the test set results
y_pred = all_rf.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Displaying the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

[23]:
```
RandomForestClassifier()
```

```
Accuracy: 0.6195155505330909
Confusion Matrix:
 [[8210 2806]
 [4938 4399]]
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.75      0.68     11016
           1       0.61      0.47      0.53      9337

    accuracy                           0.62     20353
   macro avg       0.62      0.61      0.61     20353
weighted avg       0.62      0.62      0.61     20353
```

[24]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

def plot_roc_curve(models, X_tests, y_test, model_names):
    plt.figure(figsize=(10, 8))
    for model, X_test, name in zip(models, X_tests, model_names):
        y_score = model.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_score)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, lw=2, label=f'{name} (area = {roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
```

```
    plt.show()

models = [emer_log_reg, emer_rf, inpat_log_reg, inpat_rf, outpat_log_reg,␣
 ↪outpat_rf, all_log_reg, all_rf]
X_tests = [test_df[['number_emergency']], test_df[['number_emergency']],␣
 ↪test_df[['number_inpatient']], test_df[['number_inpatient']],␣
 ↪test_df[['number_outpatient']], test_df[['number_outpatient']],␣
 ↪test_df[['number_inpatient', 'number_outpatient', 'number_emergency']],␣
 ↪test_df[['number_inpatient', 'number_outpatient', 'number_emergency']]]
model_names = ['Emer Log Reg', 'Emer RF', 'Inpat Log Reg', 'Inpat RF', 'Outpat␣
 ↪Log Reg', 'Outpat RF', 'All Log Reg', 'All RF']

plot_roc_curve(models, X_tests, y_test, model_names)
```



4.4 Feature Importance Analysis

Which factors are most predictive of readmission for diabetic patients? - This question aims to utilize predictive modeling to identify variables that significantly influence the likelihood of a patient being

readmitted again after discharge.

Here, we'll train the models used for the feature importance analysis later.

4.4.1 Logistic Regression

We'll use a LASSO logistic regression to get a basic model that shrinks unimportant features to 0. Since we have so many features, this will be helpful in narrowing down which ones are most important.

[25]:
```
from sklearn.preprocessing import StandardScaler

X_train = train_df.drop(columns=['A1Cresult_combined', 'readmitted'])
y_train = train_df['readmitted']

X_test = test_df.drop(columns=['readmitted'])
y_test = test_df['readmitted']

logistic_regression_all = LogisticRegression(penalty='l1', solver='saga')
logistic_regression_all.fit(X_train, y_train)

# Predicting the test set results
y_pred = logistic_regression_all.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Displaying the results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
/Users/jacksonstone/PycharmProjects/stat-315-final/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
```

[25]:
```
LogisticRegression(penalty='l1', solver='saga')
```

```
Accuracy: 0.6167149805925416
Confusion Matrix:
 [[9061 1955]
 [5846 3491]]
Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.82      0.70     11016
           1       0.64      0.37      0.47      9337
```

```
    accuracy                           0.62     20353
   macro avg       0.62      0.60      0.59     20353
weighted avg       0.62      0.62      0.60     20353
```

4.4.2 Random Forest

We will tune the parameters on the random forest model by comparing accuracies across different n_estimators and different max_depth

[26]:
```
accuracies = {}
# Random Forest model
# Finding n_estimators
for i in range(10, 101, 20):
    rf = RandomForestClassifier(n_estimators=i, max_depth=10)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    accuracies[i] = accuracy_score(y_test, y_pred)

plt.title("Number of Estimators vs. Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Number of Estimators")
sns.lineplot(data=accuracies)
plt.xlim(0, 100)
plt.ylim(0, 1)
plt.show()

accuracies = {}
# finding max_depth
for i in range(5, 21, 5):
    rf = RandomForestClassifier(n_estimators=60, max_depth=i)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    accuracies[i] = accuracy_score(y_test, y_pred)

plt.title("Max Depth vs. Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Max depth")
sns.lineplot(data=accuracies)
plt.xlim(0, 20)
plt.xticks(range(2, 21, 2))
plt.ylim(0, 1)
plt.show

rf = RandomForestClassifier(n_estimators=50, max_depth=8)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

```
# Evaluating the model
accuracy_best = accuracy_score(y_test, y_pred)
conf_matrix_best = confusion_matrix(y_test, y_pred)
class_report_best = classification_report(y_test, y_pred)

# Displaying the results with the best parameters
print("Accuracy:", accuracy_best)
print("Confusion Matrix:\n", conf_matrix_best)
print("Classification Report:\n", class_report_best)

random_forest_all = rf
```

```
[26]: RandomForestClassifier(max_depth=10, n_estimators=10)
```

```
[26]: RandomForestClassifier(max_depth=10, n_estimators=30)
```

```
[26]: RandomForestClassifier(max_depth=10, n_estimators=50)
```

```
[26]: RandomForestClassifier(max_depth=10, n_estimators=70)
```

```
[26]: RandomForestClassifier(max_depth=10, n_estimators=90)
```

```
[26]: Text(0.5, 1.0, 'Number of Estimators vs. Accuracy')
```

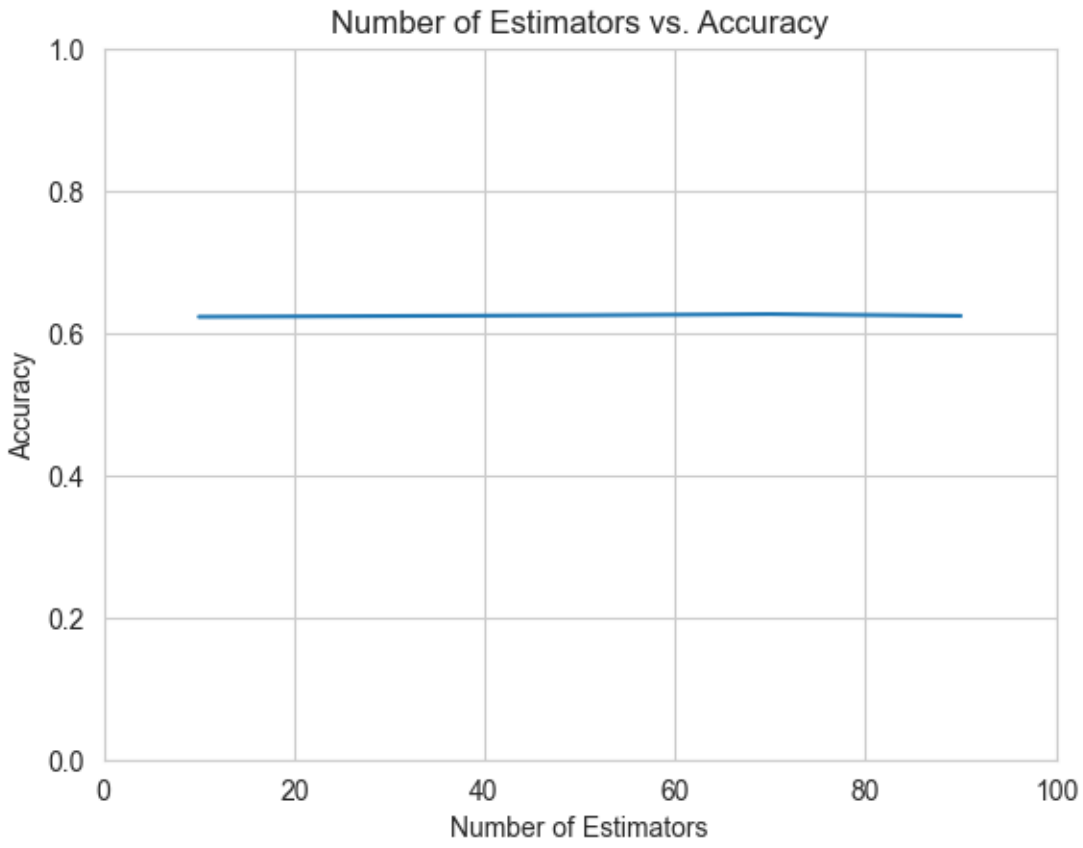
```
[26]: Text(0, 0.5, 'Accuracy')
```

```
[26]: Text(0.5, 0, 'Number of Estimators')
```

```
[26]: <Axes: title={'center': 'Number of Estimators vs. Accuracy'}, xlabel='Number of
      Estimators', ylabel='Accuracy'>
```

```
[26]: (0.0, 100.0)
```

```
[26]: (0.0, 1.0)
```



[26]: RandomForestClassifier(max_depth=5, n_estimators=60)

[26]: RandomForestClassifier(max_depth=10, n_estimators=60)

[26]: RandomForestClassifier(max_depth=15, n_estimators=60)

[26]: RandomForestClassifier(max_depth=20, n_estimators=60)

[26]: Text(0.5, 1.0, 'Max Depth vs. Accuracy')

[26]: Text(0, 0.5, 'Accuracy')

[26]: Text(0.5, 0, 'Max depth')

[26]: <Axes: title={'center': 'Max Depth vs. Accuracy'}, xlabel='Max depth',
ylabel='Accuracy'>

[26]: (0.0, 20.0)

[26]: ([<matplotlib.axis.XTick at 0x31d2f8f80>,
<matplotlib.axis.XTick at 0x31d2f8f20>,

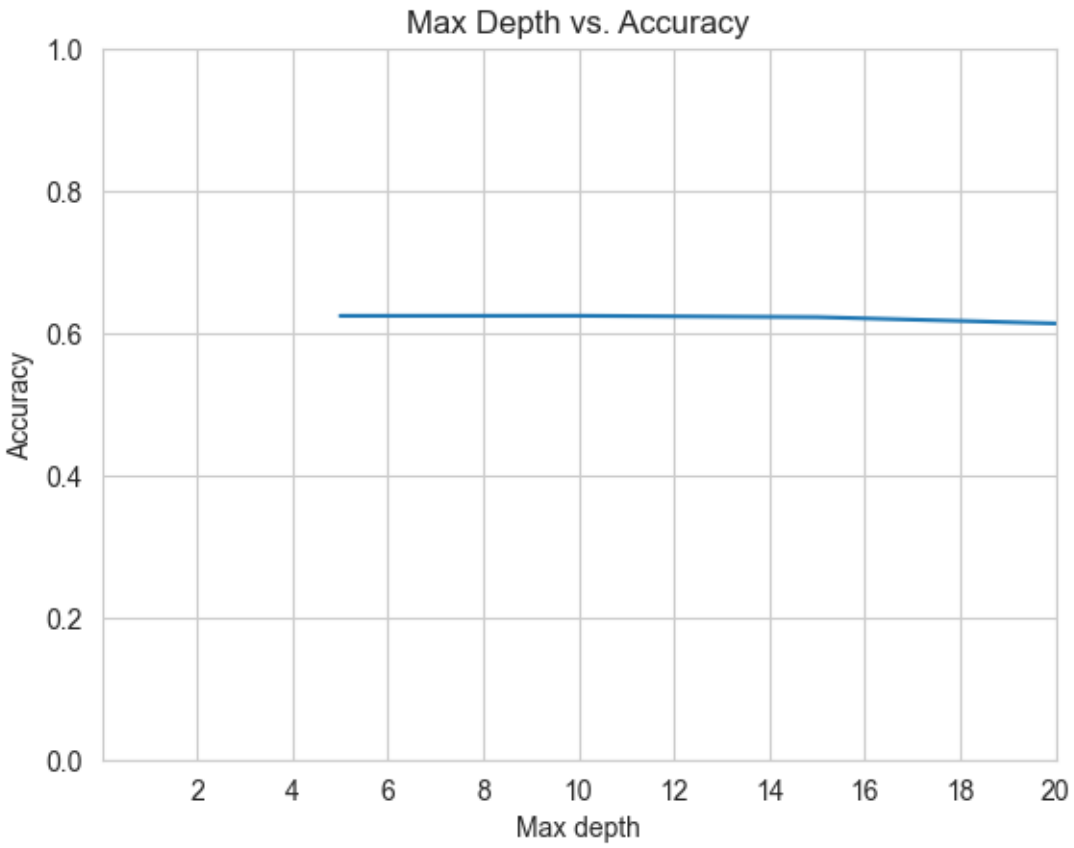
```
  <matplotlib.axis.XTick at 0x31d29ef90>,
  <matplotlib.axis.XTick at 0x31d243b30>,
  <matplotlib.axis.XTick at 0x31eb302c0>,
  <matplotlib.axis.XTick at 0x31eb30b60>,
  <matplotlib.axis.XTick at 0x31d2fbce0>,
  <matplotlib.axis.XTick at 0x31eb304a0>,
  <matplotlib.axis.XTick at 0x31eb31760>,
  <matplotlib.axis.XTick at 0x31eb31fd0>],
 [Text(2, 0, '2'),
  Text(4, 0, '4'),
  Text(6, 0, '6'),
  Text(8, 0, '8'),
  Text(10, 0, '10'),
  Text(12, 0, '12'),
  Text(14, 0, '14'),
  Text(16, 0, '16'),
  Text(18, 0, '18'),
  Text(20, 0, '20')])
```

[26]: (0.0, 1.0)

[26]: <function matplotlib.pyplot.show(close=None, block=None)>

[26]: RandomForestClassifier(max_depth=8, n_estimators=50)

```
Accuracy: 0.6250184248022405
Confusion Matrix:
 [[8203 2813]
 [4819 4518]]
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.74      0.68     11016
           1       0.62      0.48      0.54      9337

    accuracy                           0.63     20353
   macro avg       0.62      0.61      0.61     20353
weighted avg       0.62      0.63      0.62     20353
```



It seems that increasing number of estimators has little effect on accuracy, and the max depth trends upwards until around 15.

Using all variables available, our logistic regression and parameter-tuned random forest models both have an accuracy of 62%-63%

5 Interpretation and Reporting

5.1 Can We Predict the Readmission of a Patient?

5.1.1 Logistic Regression

[27]:
```
y_prob_rf = logistic_regression_all.predict_proba(X_test)[:, 1]

mask = y_test == 1
probabilities_1 = y_prob_rf[mask]
probabilities_0 = y_prob_rf[~mask]

mean_1 = np.mean(probabilities_1)
mean_0 = np.mean(probabilities_0)
```

```
iqr_1 = np.percentile(probabilities_1, 75) - np.percentile(probabilities_1, 25)
iqr_0 = np.percentile(probabilities_0, 75) - np.percentile(probabilities_0, 25)

plt.figure(figsize=(12, 6))
plt.scatter(probabilities_1, np.ones_like(probabilities_1) * 1, alpha=0.5,␣
↪color='blue', label='Actual 1')
plt.scatter(probabilities_0, np.zeros_like(probabilities_0), alpha=0.5,␣
↪color='red', label='Actual 0')
plt.errorbar(mean_1, 1, xerr=iqr_1/2, fmt='o', color='orange', capsize=5,␣
↪label='Mean & IQR (Actual 1)')
plt.errorbar(mean_0, 0, xerr=iqr_0/2, fmt='o', color='yellow', capsize=5,␣
↪label='Mean & IQR (Actual 0)')
plt.xlabel('Predicted Probability of Readmission')
plt.ylabel('Actually Readmitted')
plt.title('Scatter Plot of Predicted Probabilities vs Actuals (Logistic␣
↪Regression)')
plt.legend()
plt.show()
```

[27]: <Figure size 1200x600 with 0 Axes>

[27]: <matplotlib.collections.PathCollection at 0x31eb00e00>

[27]: <matplotlib.collections.PathCollection at 0x31eb01610>

[27]: <ErrorbarContainer object of 3 artists>

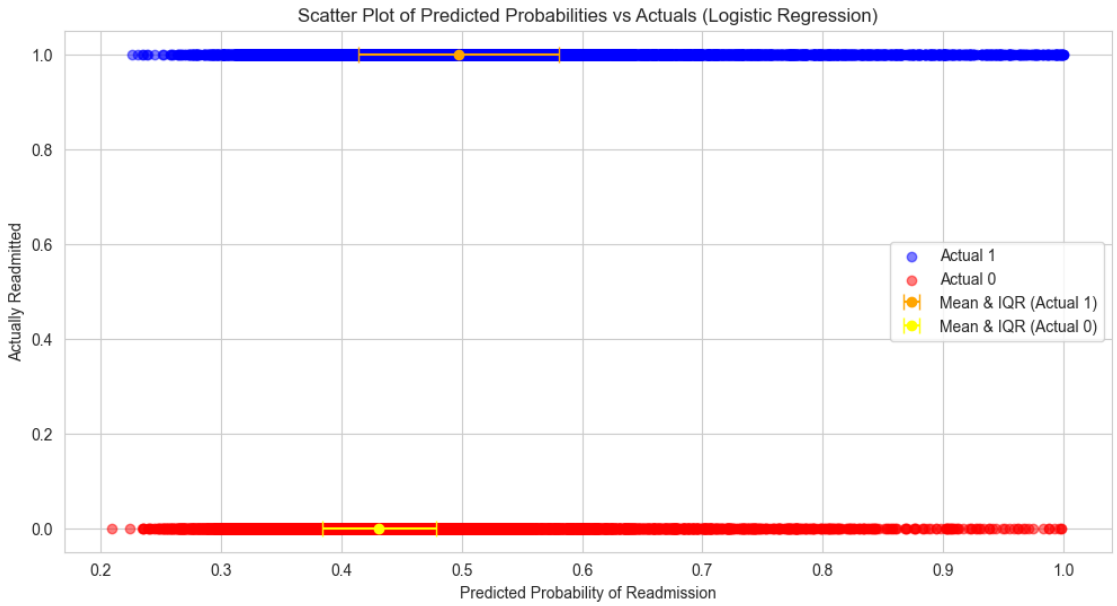
[27]: <ErrorbarContainer object of 3 artists>

[27]: Text(0.5, 0, 'Predicted Probability of Readmission')

[27]: Text(0, 0.5, 'Actually Readmitted')

[27]: Text(0.5, 1.0, 'Scatter Plot of Predicted Probabilities vs Actuals (Logistic Regression)')

[27]: <matplotlib.legend.Legend at 0x32fb43380>



[28]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.63      0.74      0.68     11016
           1       0.62      0.48      0.54      9337

    accuracy                           0.63     20353
   macro avg       0.62      0.61      0.61     20353
weighted avg       0.62      0.63      0.62     20353
```

[29]:
```
from sklearn.metrics import mean_squared_error

print(mean_squared_error(y_test, y_pred))
```

```
0.37498157519775954
```

Our logistic regression model has demonstrated the capability to predict patient readmission with an accuracy of 63%. This level of accuracy indicates that the model is significantly better than random guessing, which would have an accuracy of 50% in a balanced binary classification problem. However, there is still room for improvement. The model's performance could potentially be enhanced by incorporating more features, applying feature engineering, or experimenting with different modeling techniques.

5.1.2 Random Forest

Lets see if using a more sophisticated modeling technique with some parameter tuning will yield better results

[30]:
```
y_prob_rf = random_forest_all.predict_proba(X_test)[:, 1]

mask = y_test == 1
probabilities_1 = y_prob_rf[mask]
probabilities_0 = y_prob_rf[~mask]

mean_1 = np.mean(probabilities_1)
mean_0 = np.mean(probabilities_0)

iqr_1 = np.percentile(probabilities_1, 75) - np.percentile(probabilities_1, 25)
iqr_0 = np.percentile(probabilities_0, 75) - np.percentile(probabilities_0, 25)

plt.figure(figsize=(12, 6))
plt.scatter(probabilities_1, np.ones_like(probabilities_1) * 1, alpha=0.5,␣
↪color='blue', label='Actual 1')
plt.scatter(probabilities_0, np.zeros_like(probabilities_0), alpha=0.5,␣
↪color='red', label='Actual 0')
plt.errorbar(mean_1, 1, xerr=iqr_1/2, fmt='o', color='orange', capsize=5,␣
↪label='Mean & IQR (Actual 1)')
plt.errorbar(mean_0, 0, xerr=iqr_0/2, fmt='o', color='yellow', capsize=5,␣
↪label='Mean & IQR (Actual 0)')
plt.xlabel('Predicted Probability of Readmission')
plt.ylabel('Actually Readmitted')
plt.title('Scatter Plot of Predicted Probabilities vs Actuals (Random Forest)')
plt.legend()
plt.show()
```

[30]: <Figure size 1200x600 with 0 Axes>

[30]: <matplotlib.collections.PathCollection at 0x31d25a8d0>

[30]: <matplotlib.collections.PathCollection at 0x31eb311f0>

[30]: <ErrorbarContainer object of 3 artists>

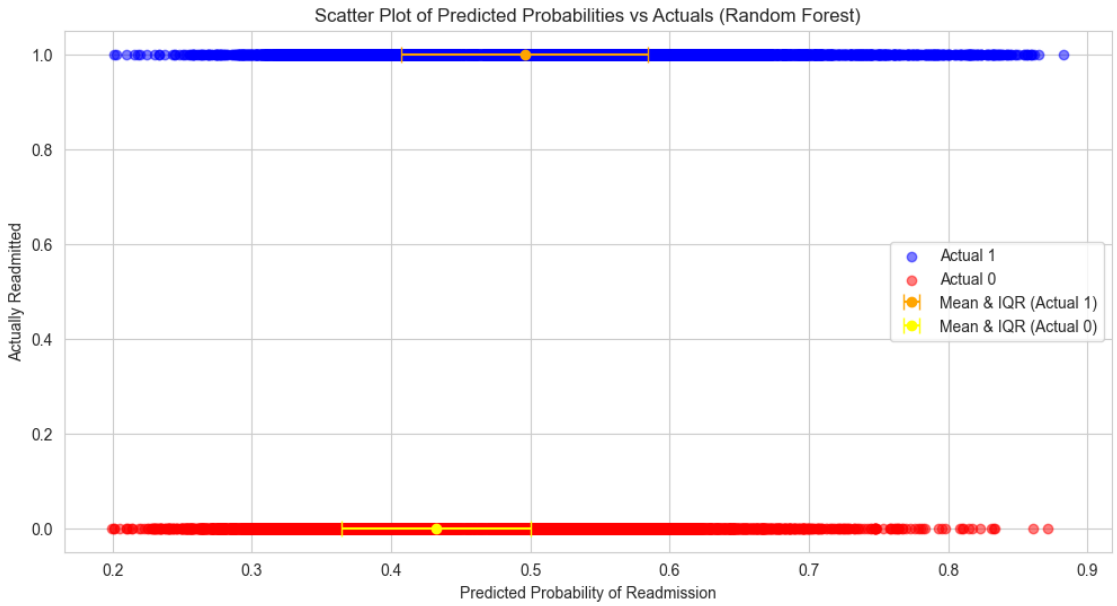
[30]: <ErrorbarContainer object of 3 artists>

[30]: Text(0.5, 0, 'Predicted Probability of Readmission')

[30]: Text(0, 0.5, 'Actually Readmitted')

[30]: Text(0.5, 1.0, 'Scatter Plot of Predicted Probabilities vs Actuals (Random Forest)')

[30]: <matplotlib.legend.Legend at 0x31d26e330>



[31]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.63      0.74      0.68     11016
           1       0.62      0.48      0.54      9337

    accuracy                           0.63     20353
   macro avg       0.62      0.61      0.61     20353
weighted avg       0.62      0.63      0.62     20353
```

[32]:
```
print(mean_squared_error(y_test, y_pred))
```

```
0.37498157519775954
```

This model has about the same accuracy compared to the logistic regression.

5.1.3 Random Forest vs Logistic Regression Conclusion

In our attempts to improve the predictive performance of our models on the dataset, we turned to Random Forest, a more sophisticated ensemble learning method.

Why We Used Parameter Tuning For the Random Forest model, we engaged in parameter tuning to optimize its performance for several reasons:

1. **Complexity of the Model**: Random Forest introduces additional complexity with multiple hyperparameters such as the number of trees in the forest (`n_estimators`) and the depth

of the trees (`max_depth`). Tuning these parameters is crucial for balancing the bias-variance tradeoff and preventing overfitting.

2. **Improvement over Baseline Models**: We aimed to surpass the performance of simpler models, such as logistic regression, by leveraging Random Forest's ability to capture complex nonlinear relationships between features. Parameter tuning is essential to unlock this potential fully.

The Outcome Despite our efforts in parameter tuning, the Random Forest model achieved around the same level of accuracy as the Logistic Regression. This outcome can be attributed to several factors:

1. **Model Complexity vs. Data Complexity**: While Random Forest can capture complex relationships, it might also introduce unnecessary complexity if the underlying relationships in the data are not as complex. In such cases, simpler models like logistic regression can perform better due to their simplicity and transparency.

2. **Dimensionality and Feature Representation**: Logistic regression might have benefited from the way features were represented or engineered in our dataset. In contrast, Random Forest, despite its capability to handle high-dimensional data, might not have leveraged the feature space as effectively as logistic regression.

3. **Hyperparameter Tuning Limitations**: There's always a possibility that the range of hyperparameters explored during the tuning process was not extensive enough or that the optimal combination was not found due to limitations in computational resources or time constraints.

4. **Inherent Randomness**: Random Forest introduces a level of randomness in model construction, which can sometimes lead to variability in performance. This randomness, while beneficial in reducing overfitting, can also lead to less stable performance across different runs or subsets of the data.

Conclusion The experience underscores the importance of understanding both the strengths and limitations of sophisticated machine learning models like Random Forest and the nuances of the dataset being modeled. It also highlights that more complex models do not always guarantee better performance and that sometimes simpler models can be more effective, depending on the context and nature of the data.

However, both models still show much better predictive power than random guessing, and could be useful for medical personnel to evaulate a patient's risk for readmission.

5.2 What Features are Most Important in Prediction?

5.2.1 Random Forest Importances

[33]:
```
# Get the indices of the 10 features with the greatest importance
top_10_indices =  np.argsort(np.abs(random_forest_all.feature_importances_))[-10:
 ↪]

features = train_df.drop(columns=['A1Cresult_combined']).columns
```

```
# Plotting
plt.figure(figsize=(10, 8))
plt.title('Top 10 Feature Importances')
plt.barh(range(10), [random_forest_all.feature_importances_[i] for i in␣
 ↪top_10_indices], color='skyblue', align='center')
plt.yticks(range(10), features[top_10_indices])
plt.xlabel('Relative Importance')
plt.ylabel('Features')
plt.show()
```

```
[33]: <Figure size 1000x800 with 0 Axes>
```

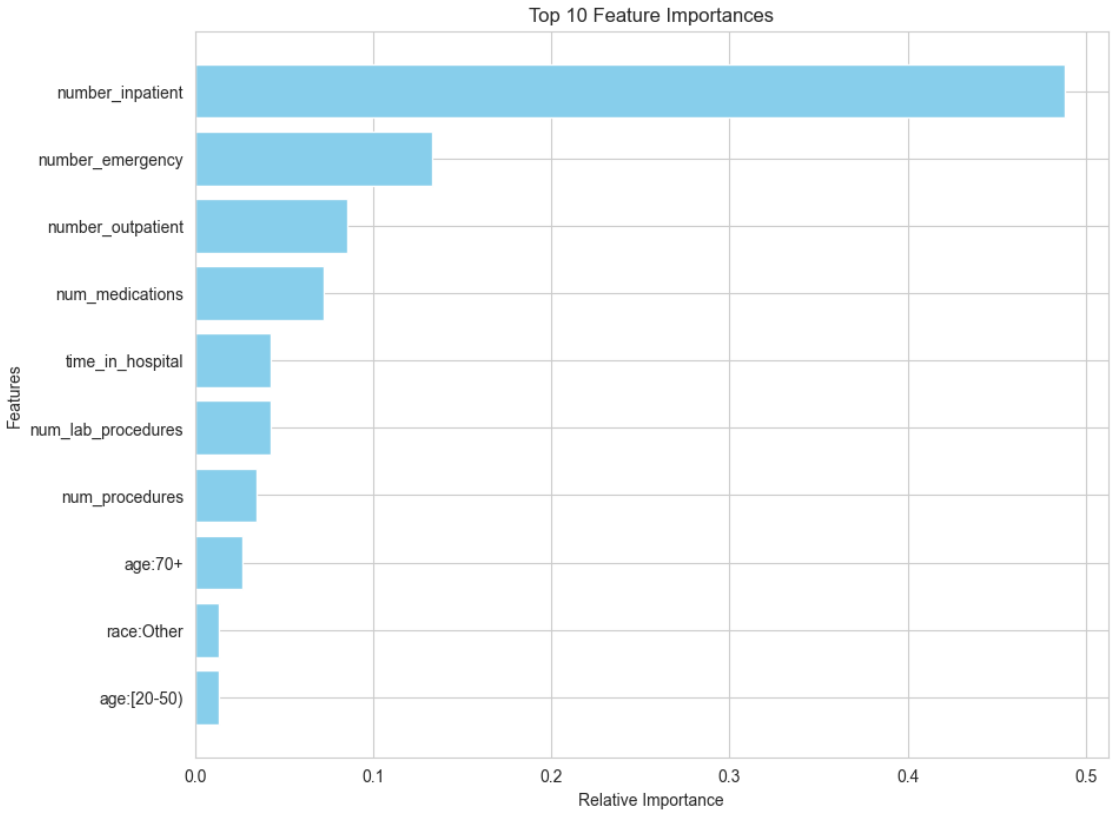
```
[33]: Text(0.5, 1.0, 'Top 10 Feature Importances')
```

```
[33]: <BarContainer object of 10 artists>
```

```
[33]: ([<matplotlib.axis.YTick at 0x328fc4e60>,
        <matplotlib.axis.YTick at 0x324c0ad50>,
        <matplotlib.axis.YTick at 0x31eb02120>,
        <matplotlib.axis.YTick at 0x328fc7710>,
        <matplotlib.axis.YTick at 0x324c68200>,
        <matplotlib.axis.YTick at 0x324c69dc0>,
        <matplotlib.axis.YTick at 0x324c6a690>,
        <matplotlib.axis.YTick at 0x328fc7a70>,
        <matplotlib.axis.YTick at 0x324c6ad20>,
        <matplotlib.axis.YTick at 0x324c6b6b0>],
       [Text(0, 0, 'age:[20-50)'),
        Text(0, 1, 'race:Other'),
        Text(0, 2, 'age:70+'),
        Text(0, 3, 'num_procedures'),
        Text(0, 4, 'num_lab_procedures'),
        Text(0, 5, 'time_in_hospital'),
        Text(0, 6, 'num_medications'),
        Text(0, 7, 'number_outpatient'),
        Text(0, 8, 'number_emergency'),
        Text(0, 9, 'number_inpatient')])
```

```
[33]: Text(0.5, 0, 'Relative Importance')
```

```
[33]: Text(0, 0.5, 'Features')
```



The model indicates that the number of inpatient visits a patient had prior to their current stay is by far the most predictive feature, accounting for approximately 45% of the model's predictive power. This is a substantial percentage, underscoring the critical role of prior inpatient visits in forecasting readmissions.

Following the number of inpatient visits, the next two most important features are related to the number of visits as well - emergency and outpatient visits, respectively. These findings suggest a clear pattern: patients with a higher number of prior hospital visits, regardless of the type, are more likely to be readmitted. This pattern highlights the importance of these features in predicting readmissions and suggests that they should be key considerations in patient care and monitoring strategies.

The significance of these features points towards the necessity for healthcare providers to focus on patients with a history of multiple hospital visits. By identifying these patients early, doctors and medical personnel can implement targeted interventions and monitoring strategies to potentially reduce the risk of readmission. This could include more rigorous follow-up care, personalized patient education programs, and enhanced support for managing chronic conditions.

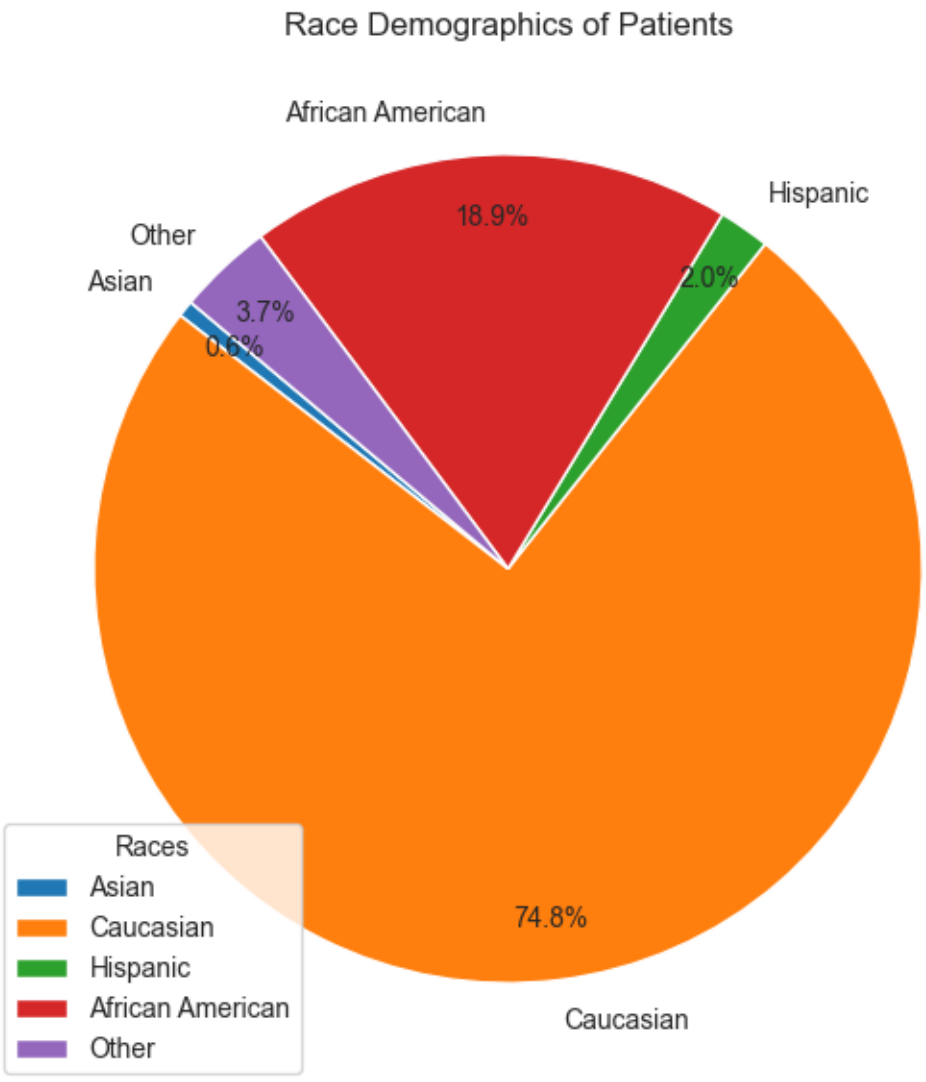
In summary, the insights from our random forest model emphasize the importance of prior hospital visits in predicting readmissions. Healthcare providers should take these findings into account when developing strategies to improve patient outcomes and reduce readmission rates. By focusing on patients with a high number of prior visits, medical personnel can better allocate resources and tailor care to those who are most at risk of readmission.

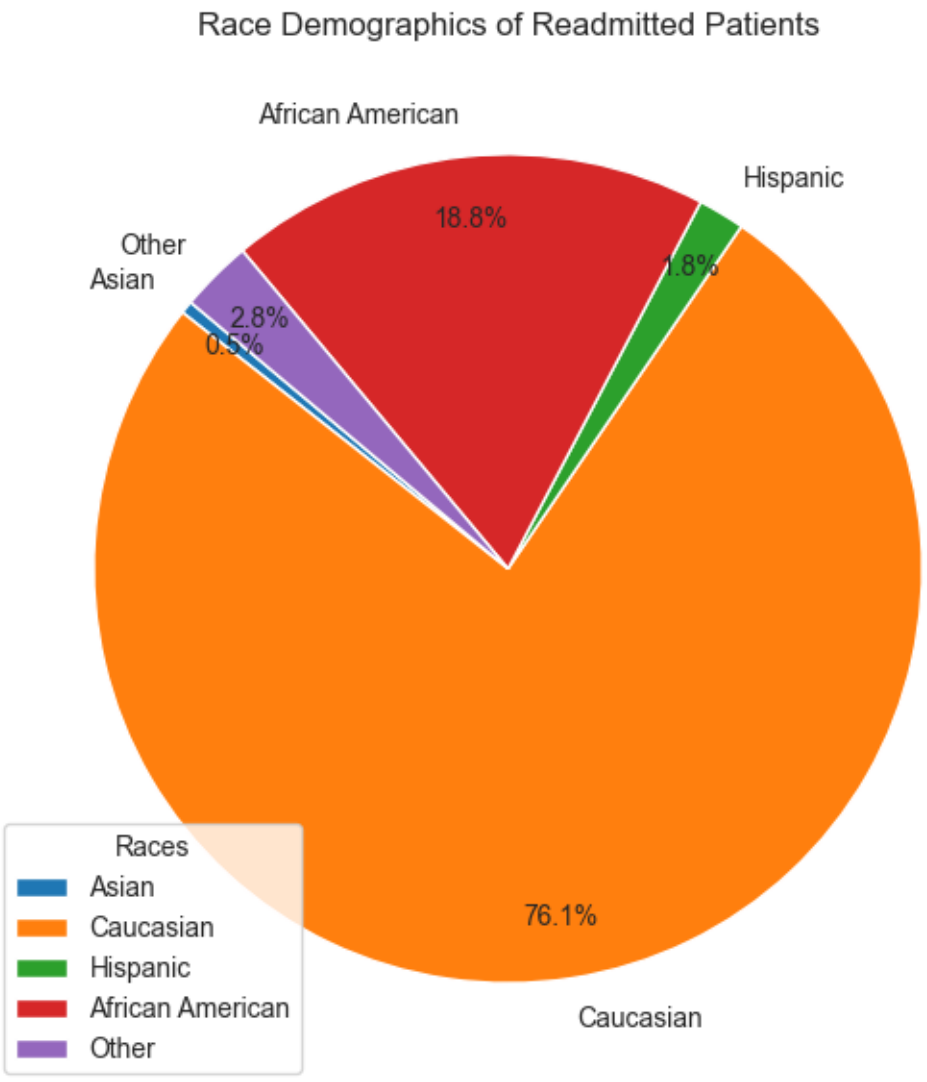
5.3 How Does Readmission Differ Among Different Demographics?

When conducting a comparative analysis of readmission rates among different demographic groups - observing that the readmission rates closely mirror the overall totals of each demographic - we are presented with both challenges and insights. On one hand, it suggests that the distribution of readmissions within each demographic group is proportionate to their representation in the overall population. However, on the other hand, it raises questions about whether this alignment truly reflects equitable healthcare outcomes or if it conceals underlying disparities.

This demonstration of this mirroring effect on the data indicates that readmission rates are consistent across demographic groups relative to their population size. In such cases, it becomes challenging to pinpoint specific demographic groups that may be disproportionately affected by higher readmission rates. Without clear deviations from this trend, it's difficult to ascertain whether disparities exist and which groups require targeted interventions. Therefore, it's imperative to interpret these findings cautiously and consider the broader context of healthcare access, quality, and social determinants of health.

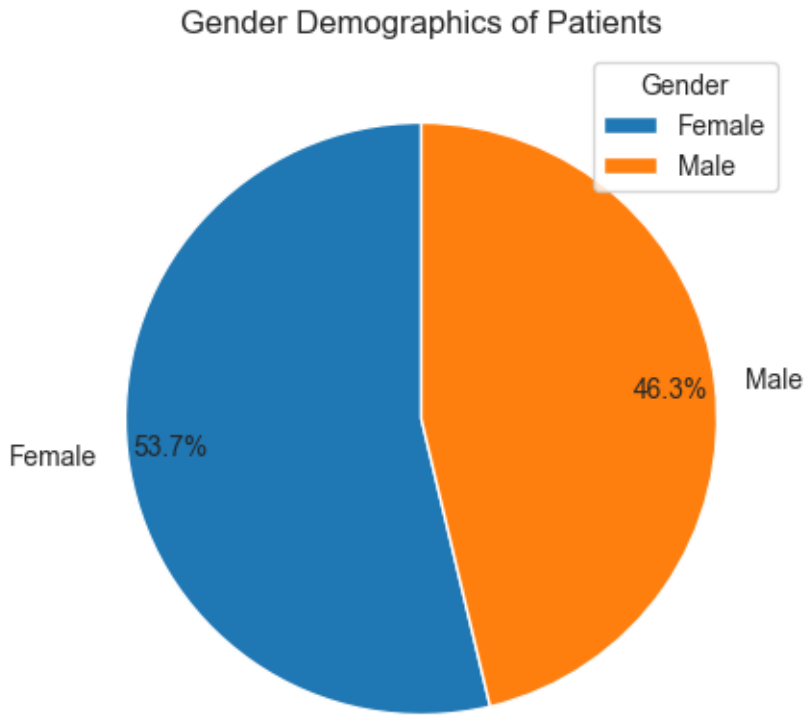
[34]:
```
race_patients()
show_race_demographics_pie_chart()
```

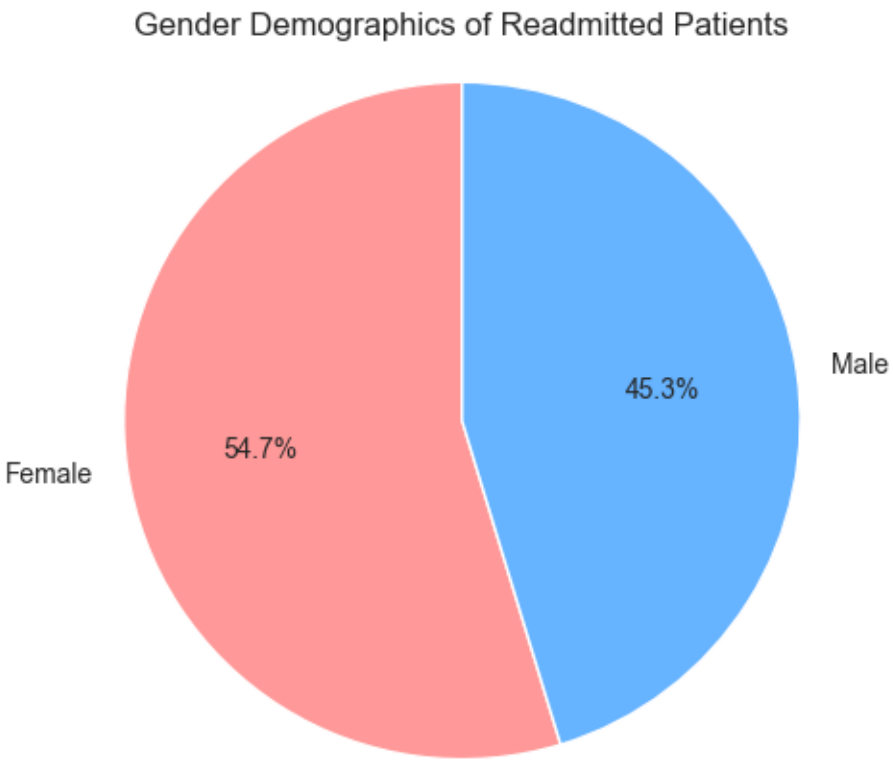

Race Demographics of Patients
African American
Hispanic
Other
Asian
18.9%
2.0%
3.7%
0.6%
74.8%
Caucasian
Races
Asian
Caucasian
Hispanic
African American
Other



This pie chart visualizes the distribution of readmissions among different demographic groups, specifically focusing on race. This visualization helps in understanding how readmission rates vary across different racial demographics. The prominence of the Caucasian demographic in this readmission analysis is evident. Nevertheless, it's crucial to acknowledge that our dataset primarily comprises patients from this demographic, potentially introducing significant bias into the data.

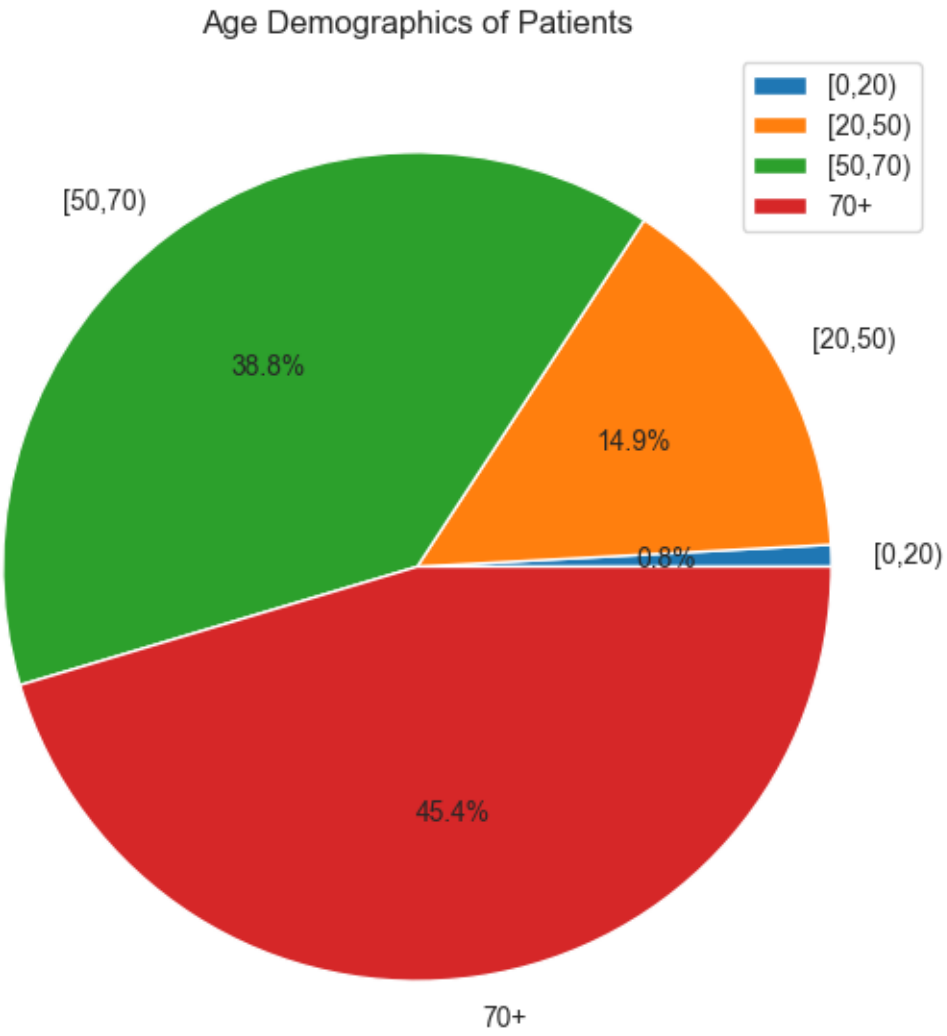
[35]:
```
gender_patients()
show_gender_readmission_demographics_pie_chart()
```

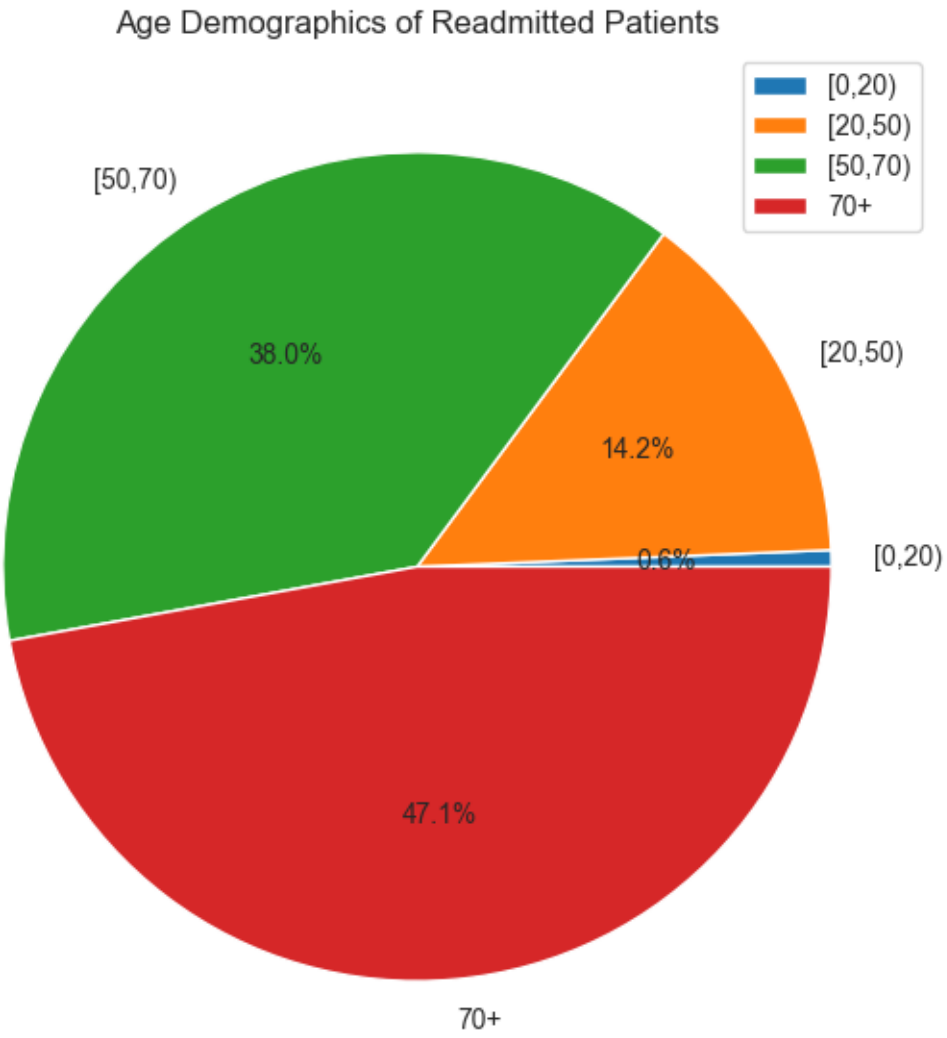

Gender Demographics of Patients
Gender
Female
Male
46.3%
Male
Female
53.7%



There seems to be no significance in the gender demographics.

[36]:
```
age_patients()
show_age_demographics_pie_chart()
```

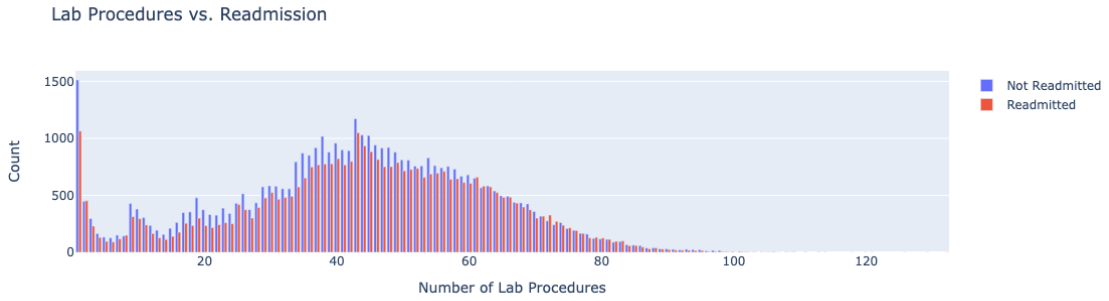

Age Demographics of Patients
[0,20)
[20,50)
[50,70)
70+
[50,70)
38.8%
[20,50)
14.9%
0.8%
[0,20)
45.4%
70+



In this chart, the age groups of 10 years or less and 10 to 20 years were merged due to their similar percentages, enhancing readability. Upon examining this chart, it becomes evident that individuals aged 50 and above exhibit a tendency towards higher rates of readmission to the hospital.

5.4 What Role Does the Number of Tests, Changes in Medications, and Test Results Play in Predicting readmission?

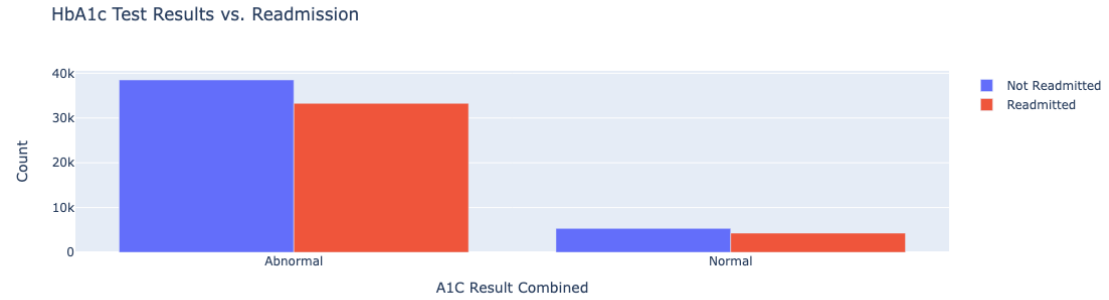
[37]:
```
labProcedures_v_readmission_chart()
```



The graph depicting the relationship between the number of medications and the count of readmissions seems to be normal with the exception of some outliers. After removing the outliers, we discover a mean of around 42.40 medications for hospital patients not readmitted and a mean of around 44.39 medications for hospital patients that are readmitted. For each graph, as the number of medications deviates from this central range, the count of readmissions tends to decrease.

1. **Insights from Statistical Analysis**: After conducting a t-test, accounting for the outliers in the dataset, we get a T-statistic of -16.71 and a p-value of 1.40e-62. Given our large absolute T-statistic and a very small p-value that is close to zero, this suggests a highly significant difference in the means of num_lab_procedures between hospital patients that are not readmitted vs. hospital patients that are readmitted. Overall, the result of this test suggests that num_lab_procedures may be an important factor related to readmission status.
2. **Implications for Healthcare**: The observed difference in mean number of lab procedures between patients who are readmitted versus those who are not highlights the potential importance of monitoring and managing the utilization of lab procedures for hospital patients. A higher number of lab procedures may indicate more complex health conditions or greater healthcare needs, suggesting that patients with a higher number of procedures may be at increased risk of readmission. It may highlight that patients who need more lab procedures done should require greater careful monitoring to prevent readmissions.

[38]:
```
HbA1c_v_readmission_chart()
```

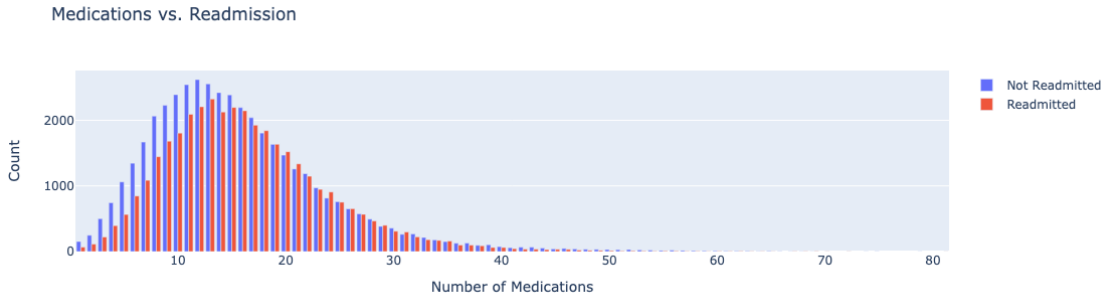


The graph presented above illustrates the relationship between HbA1c levels and hospital readmission rates. HbA1c, a marker for long-term blood glucose control, can provide insights into the management of diabetes in patients. The graph suggests that glycemic control, as indicated by HbA1c levels, plays a significant role in the likelihood of hospital readmission.

1. **Distribution of Patients**: We see that there are higher readmission rates in the abnormal category, which includes patients with elevated HbA1c levels (">7" or ">8"). This indicates that patients with poorly controlled diabetes are more prone to complications requiring readmission. Lower readmission rates in the "Normal" category suggests that well-managed diabetes, reflected by normal HbA1c levels, is associated with fewer complications leading to readmissions.

2. **Implications for Healthcare**: Although it's important to consider that while HbA1c levels provide valuable information about long-term glucose control, the direct cause of readmission can be multifactorial, involving additional aspects such as the adherence to treatment and access to healthcare services. Therefore, while the graph provides critical insights, it should be interpreted within the broader context of each patient's health status and healthcare system factors.

[39]:
```
Medications_v_readmission_chart()
```



The graph depicting the relationship between the number of medications and the count of readmissions seems to be right-skewed with a mode of 12 medications for hospital patients not readmitted and a mode of 13 medications for hospital patients that are readmitted. For each graph, as the number of medications deviates from this central range, the count of readmissions tends to decrease.

We can see that the curve for hospital patients that are readmitted seems to be shifted to the right more than the curve for hospital patients that are not readmitted. The observation that the chart for the number of patients versus not readmitted patients is more to the left compared to the bell curve for the number of patients versus readmitted patients suggests a few key interpretations:

1. **Distribution of Patients**: The higher mode value (13 medications) among readmitted patients compared to non-readmitted patients (12 medications) could indicate that patients who were readmitted tend to have a slightly higher medication burden on average compared to those who were not readmitted. This observation suggests that patients with more complex medical conditions requiring more medications may have a higher likelihood of readmission.

2. **Implications for Healthcare**: This distribution implies that interventions aimed at moving patients to the left of the curve could potentially reduce readmission rates. It may highlight that patients who need to be provided more medications due to having more compelx health needs should require greater careful monitoring and management to prevent readmissions.

5.5 How do the number of emergency visits, inpatient stays, and outpatient visits in the year prior to the hospitalization correlate with readmission rates?

The models for for inpatient visits, outpatient visits, emergency visits, and all combined show similar results: there is some correlation with readmission, but it is not highly correlated, as the models only had around 60% maximum accuracy.

Outpatient visits have the least predictive power, while inpatient visits or all combined have the most.

[40]:
```
np.exp(emer_log_reg.coef_)
```

[40]:
```
array([[1.53105233]])
```

[41]:
```
np.exp(outpat_log_reg.coef_)
```

[41]:
```
array([[1.1660991]])
```

[42]:
```
np.exp(inpat_log_reg.coef_)
```

[42]:
```
array([[1.55070072]])
```

Our analysis using logistic regression models has provided insightful findings on the factors influencing hospital readmissions. Specifically, we observed that the number of prior visits to the hospital plays a significant role in predicting the likelihood of a patient being readmitted. The models revealed the following:

- **Inpatient Visits**: For each inpatient visit a patient had prior to the current admission, their odds of being readmitted increase by 53%. This indicates a strong association between the frequency of inpatient visits and the risk of readmission, highlighting the need for targeted interventions for patients with frequent inpatient stays.
- **Outpatient Visits**: Each outpatient visit prior to admission is associated with a 17% increase in the odds of readmission. Although the impact is less pronounced than that of inpatient visits, it nonetheless signifies that outpatient visit history is an important factor to consider in readmission risk models.
- **Emergency Visits**: The data shows a 55% increase in the odds of readmission for each emergency visit a patient had before their current admission. This underscores the critical nature of emergency visits as a predictor of readmission, possibly reflecting the severity of health issues that necessitate emergency care.

These findings underscore the importance of considering the patient's entire healthcare journey, including inpatient, outpatient, and emergency visits, in efforts to predict and prevent hospital

readmissions. By identifying patients at higher risk of readmission, healthcare providers can tailor follow-up care and interventions to address the specific needs of these individuals, ultimately improving patient outcomes and reducing the burden of readmissions on the healthcare system.

6 Contribution Report

6.1 Kimberly Chen

I focused mostly on creating the visualizations for the Treatment and Testing Variables portion of the assignment as well as doing some statistical analysis on the results and interpreting the resulting graphs. Additionally, I created a random forest model for Random Forest Importances to showcase our top ten most influential features.

6.2 Nikki Rad

I worked on writing a few of the interpretations, specifically in the Healthcare Utilization and Outcomes section as well as in Interpretation and Reporting. I also did the final run through, making sure that formatting was inline with the rubric, visualizations were properly named and displayed, code was organized, and results were accurate.

6.3 Jackson Stone

I primarily worked on creating the logistic regression and random forest models for the Feature Importance Analysis and interpretation of it, including visualizing the models' performance. I also worked on creating the models for the Healthcare Utilization and Outcomes and interpreting these models. I also created the Github repository and Docker image.

6.4 Rhiannon Stracener

I tackled the Comparative Anaylsis portion of the assignment. I created the code to develop the charts for each demographic section we were wanting to compare. I would also go through and help with various issues with other group members code such as creating the A1Cresult_combined variable and the printing of some of the graphs in the conclusion.