COVID-19 Hospital Stay Predictor

I. Technical Summary

In this project, I create a predictive classification model geared towards a hospital's analytics team seeking to make informed decisions for resources and personnel allocation based on predicted inpatient length of stay. The target variable is classified into three labels: `"Short-Term"`, `"Medium-Term"`, and `"Long-Term"` based on the number of days a patient is predicted to stay. The model uses fourteen features to predict the length of stay -- with `Type of Admission`, `Severity of Illness`, `Bed Grade`, and `Ward Type` having the most influence. The final model produced is a tuned `XGBoost model` that is able to predict target labels with `54% accuracy`. With further research and implementation, classifiers like these could be used in emergency situations to properly allocate resources based on predictive knowledge of how long a patient is expected to admitted for.

II. The Motivation

It goes without saying that the onset of the COVID-19 virus has had a drastic impact on the world as we know it today. At the beginning of the pandemic, hospitals were overrun with patients needing medical attention and it became clear that hospital infrastructure was not equipped for this unprecedented event.

The motivation for this project came from a conversation I had with my roommate who is a nurse here in San Francisco who worked on a COVID unit during the pandemic. She told me that one of the biggest factors leading to inadequate care was the lack of hospital resources. Without proper resources, sick people were not able to receive proper care. One of major contributing factors to this came from not knowing how long a patient would be hospitalized for when they are admitted. With such a huge influx of more long-term residential patients, hospitals were quickly drained of resources such as PPE, certain medications, and even beds themselves.

Predictive modeling has many applications in the healthcare field and I wanted to use this project as an opportunity to show how machine learning could be used to help alleviate the burden of this type of issue in the future. In this project, I build a classifier model that can be used to predict the length of stay (LOS) of a COVID-19 patient given certain predictive features. With further research and implementation, classifiers like these could be used in emergency situations to properly allocate resources based on predictive knowledge of how long a patient is expected to admitted for.

III. Business Understanding

Stakeholder: Healthcare analytics team for a group of hospitals looking to predict length of stay (LOS) estimates for COVID-19 inpatients.

Problem: Effectively predict the length of stay of a COVID-19 patient given similar past patient profiles in order to better allocate hospital resources and personnel.

Target: Length of stay (LOS)

- Length of stay is defined as a categorical variable in this dataset broken into 10 day increments.
- In order to fit with my model, this was encoded using a `LabelEncoder` as follows:

Label	# of Days
0	0-10 Days
1	11-20 Days
2	21-30 Days
3	31-40 Days
4	41-50 Days
5	51-60 Days
6	61-70 Days
7	71-80 Days
8	81-90 Days
9	91-100 Days
10	>100 Days

- Upon further analysis of the data, I decided to group the labels into three categories: `"Short-Term"`, `"Medium-Term"`, and "Long-Term" by binning the data as follows:

Label	Type of Stay	# of Days
0	Short-Term	0-20 Days
1	Medium-Term	21-50 Days
2	Long-Term	> 51 Days

My data came from a series of analytic healthcare data provided on Kaggle by Vidhya Healthcare Analytics to be used as a means of building and sharing predictive models for COVID-19 related projects.

The data was collected anonymously in order to protect the identity of the individuals and hospitals.

Imports

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pylab as pl
%matplotlib inline
sns.set(color_codes=True)

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
from sklearn.tree import DecisionTreeClassifier
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, cross_val_
from xgboost import XGBClassifier, plot_tree
```

IV. Data Understanding

The Dataset

In [2]:

```python
# Read file and initial observations of data

df_train = pd.read_csv('data/COVID-19_Train.csv')
df_train.head()
```

Out[2]:

	case_id	Hospital_code	Hospital_type_code	City_Code_Hospital	Hospital_region_code	Availal Ex Roo Hospi
0	1	8	c	3	Z	
1	2	2	c	5	Z	
2	3	10	e	1	X	
3	4	26	b	2	Y	
4	5	26	b	2	Y	

In [3]:

```python
# Check shape --> there are 318,438 patient entries and 18 feature columns inclu

df_train.shape
```

Out[3]: (318438, 18)

In [4]:

```python
# Get info --> quick summary of what I am working with

df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 318438 entries, 0 to 318437
Data columns (total 18 columns):
 #   Column                             Non-Null Count   Dtype
---  ------                             --------------   -----
 0   case_id                            318438 non-null  int64
 1   Hospital_code                      318438 non-null  int64
 2   Hospital_type_code                 318438 non-null  object
 3   City_Code_Hospital                 318438 non-null  int64
 4   Hospital_region_code               318438 non-null  object
 5   Available Extra Rooms in Hospital  318438 non-null  int64
 6   Department                         318438 non-null  object
 7   Ward_Type                          318438 non-null  object
 8   Ward_Facility_Code                 318438 non-null  object
 9   Bed Grade                          318325 non-null  float64
 10  patientid                          318438 non-null  int64
```

```
 11   City_Code_Patient                  313906 non-null   float64
 12   Type of Admission                  318438 non-null   object
 13   Severity of Illness                318438 non-null   object
 14   Visitors with Patient              318438 non-null   int64
 15   Age                                318438 non-null   object
 16   Admission_Deposit                  318438 non-null   float64
 17   Stay                               318438 non-null   object
dtypes: float64(3), int64(6), object(9)
memory usage: 43.7+ MB
```

In [5]:
```
# Get data types --> int, float, and objects, some preprocessing will be necessa

df_train.dtypes
```

Out[5]:
```
case_id                               int64
Hospital_code                         int64
Hospital_type_code                   object
City_Code_Hospital                    int64
Hospital_region_code                 object
Available Extra Rooms in Hospital     int64
Department                           object
Ward_Type                            object
Ward_Facility_Code                   object
Bed Grade                           float64
patientid                             int64
City_Code_Patient                   float64
Type of Admission                    object
Severity of Illness                  object
Visitors with Patient                 int64
Age                                  object
Admission_Deposit                   float64
Stay                                 object
dtype: object
```

In [6]:
```
# Check for duplicates

df_train.duplicated().sum()
```

Out[6]: 0

Feature Analysis

Below is an overview of the features, a description, and important notes pertaining to how the data points were recorded in the dataset:

Feature	Description	Notes
case_ID	Case ID registered in Hospital	
Hospital_code	Unique code for the Hospital	Codes 1-30 for 30 Hospitals included in data
Hospital_type_code	Unique code for the type of Hospital	Codes a-g for 7 types of hospitals included (e.g. general, research, etc.)
City_Code_Hospital	City Code of the Hospital	Codes 1-13 for 13 cities included in data
Hospital_region_code	Region Code of the Hospital	Codes X, Y, Z for three regions included in data
Available Extra Rooms in Hospital	Number of extra rooms available in the Hospital	
Department	Department overlooking the case	

Feature	Description	Notes
Ward_Type	Unique code for the Ward type	Unique code P-U for type of Ward
Ward_Facility_Code	Unique code for the Ward facility	Unique code A-F for Ward facility
Bed Grade	Condition of bed in the Ward	Scale grade 1-4
patientid	Unique patient ID number	
City_Code_Patient	City code for the patient	Codes 1-38 for 38 cities included in data
Type of Admission	Admission type registered by the Hospital	Categorized as Trauma, Emergency, or Urgent
Severity of Illness	Severity of the illness recorded at the time of admission	Categorized as Minor, Moderate, or Extreme
Visitors with Patient	Number of visitors with the patient	
Age	Age of patient	
Admission_Deposit	Deposit made at time of admission	

Since my model is intended to predict length of stay at time of admission in order to place a patient effectively, I am going to remove features that do not have predictive applications:

- `case_ID` , `patientid` --> simply information about unique patient code
- `Visitors with Patient` --> more visitors could mean longer stay but this is not predictive
- `Hospital_region_code` --> region is covered by `City_Code_Hospital` and provides more detailed geographic importance.

In [7]:
```
df_train.drop(['case_id', 'patientid', 'Hospital_region_code', 'Visitors with Pa
df_train.head()
```

Out[7]:

	Hospital_code	Hospital_type_code	City_Code_Hospital	Available Extra Rooms in Hospital	Department	Ward_Type	W
0	8	c	3	3	radiotherapy	R	
1	2	c	5	2	radiotherapy	S	
2	10	e	1	2	anesthesia	S	
3	26	b	2	2	radiotherapy	R	
4	26	b	2	2	radiotherapy	S	

In [8]:
```
# Summary statistics for the numerical variables

df_train.describe()
```

Out[8]:

	Hospital_code	City_Code_Hospital	Available Extra Rooms in Hospital	Bed Grade	City_Code_Patient	Ad
count	318438.000000	318438.000000	318438.000000	318325.000000	313906.000000	
mean	18.318841	4.771717	3.197627	2.625807	7.251859	
std	8.633755	3.102535	1.168171	0.873146	4.745266	
min	1.000000	1.000000	0.000000	1.000000	1.000000	
25%	11.000000	2.000000	2.000000	2.000000	4.000000	
50%	19.000000	5.000000	3.000000	3.000000	8.000000	
75%	26.000000	7.000000	4.000000	3.000000	8.000000	
max	32.000000	13.000000	24.000000	4.000000	38.000000	

Some features seem to have some skew and the low minumum values should be checked as outliers. Let's start out by doing a little data cleaning before jumping into my analysis.

Data Cleaning: First Steps

In [9]:
```
# Check missing values --> Bed Grade & City_Code_Patient

df_train.isna().sum()
```

Out[9]:
```
Hospital_code                        0
Hospital_type_code                   0
City_Code_Hospital                   0
Available Extra Rooms in Hospital    0
Department                           0
Ward_Type                            0
Ward_Facility_Code                   0
Bed Grade                          113
City_Code_Patient                 4532
Type of Admission                    0
Severity of Illness                  0
Age                                  0
Admission_Deposit                    0
Stay                                 0
dtype: int64
```

In [10]:
```
# Fill 'Bed Grade' and 'City_Code_Patient' missing values with mode due to outli

df_train['Bed Grade'].fillna(df_train['Bed Grade'].mode()[0], inplace=True)
df_train['City_Code_Patient'].fillna(df_train['City_Code_Patient'].mode()[0], in

# Check and make sure these are now filled

df_train.isna().sum()
```

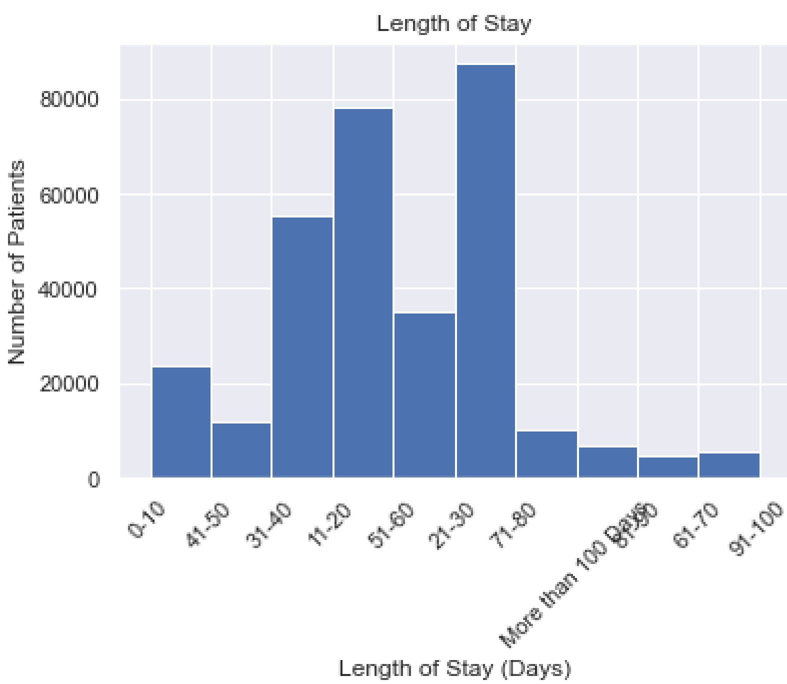
Out[10]:
```
Hospital_code                        0
Hospital_type_code                   0
City_Code_Hospital                   0
Available Extra Rooms in Hospital    0
Department                           0
Ward_Type                            0
Ward_Facility_Code                   0
Bed Grade                            0
City_Code_Patient                    0
```

```
Type of Admission                    0
Severity of Illness                  0
Age                                  0
Admission_Deposit                    0
Stay                                 0
dtype: int64
```

Data Cleaning: Target Analysis

Let's first take a look at the target variable 'Stay' in the data set to see the distribution of values:

In [11]:
```python
# Plot histogram for target 'Stay'

plt.hist(df_train['Stay'])
plt.xticks(rotation=45)
plt.xlabel('Length of Stay (Days)')
plt.ylabel('Number of Patients')
plt.title('Length of Stay')
plt.show()

# Value counts for LOS

df_train['Stay'].value_counts()
```



Out[11]:
```
21-30                 87491
11-20                 78139
31-40                 55159
51-60                 35018
0-10                  23604
41-50                 11743
71-80                 10254
More than 100 Days     6683
81-90                  4838
91-100                 2765
61-70                  2744
Name: Stay, dtype: int64
```

We can see that the vast majority of patients are hospitalized for less than 60 days. In the context of my problem, I want to help hospitals allocate resources and personnel effectively. In doing so, it may not be necessary to have 10 different prediction labels and instead am going to separate into three types of stay.

I am going to bin the target label into three categories based on length of stay:

- **Short-Term**: less than 20 days
  - Quick turnaround for hospital
  - Less personnel & resources
  - Lower risk for long-term complications and/or death

- **Medium-Term**: 21 to 60 days
  - More intensive care
  - Moderate use of personnel & resources
  - Increased risk for long-term complications and/or death

- **Long-Term**: more than 60 days
  - Most intensive care
  - Heavy use of personnel & resources
  - Significant risk for long-term complications and/or death

In [12]:

```
# Bin target 'Stay' into three types - short (0) medium (1) and long (2)

df_train['Stay']= df_train['Stay'].replace({'0-10':0, '11-20':0,
                                           '21-30':1, '31-40':1, '41-50':1,
                                           '51-60':2,'61-70':2,'71-80':2,'81-90':2

df_train['Stay']
```

Out[12]:

```
0         0
1         1
2         1
3         1
4         1
         ..
318433    0
318434    1
318435    0
318436    0
318437    0
Name: Stay, Length: 318438, dtype: int64
```

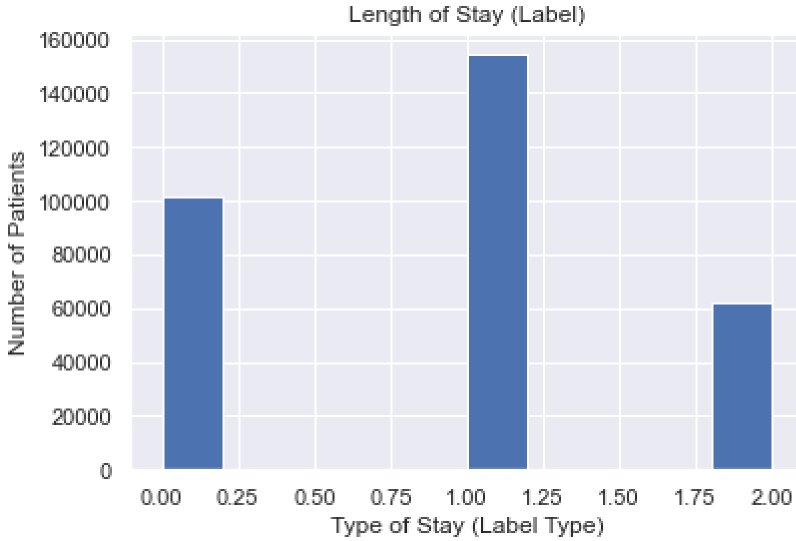
Let's check out the new distributions:

In [13]:

```
# Plot histogram of new distributions

plt.hist(df_train['Stay'],)
df_train['Stay'].value_counts()
plt.xlabel('Type of Stay (Label Type)')
plt.ylabel('Number of Patients')
plt.title('Length of Stay (Label)')

# Check new value counts

df_train['Stay'].value_counts()
```

```
1    154393
```

```
Out[13]: 0     101743
         2      62302
         Name: Stay, dtype: int64
```



We can see with the new distributions that there is much less class imbalance which will allow my model to have an easier time predicting the minority classes. This works with the context of the problem as well since 10 day increments aren't necessarily the best for a length of stay analysis.

Exploratory Data Analysis (EDA)

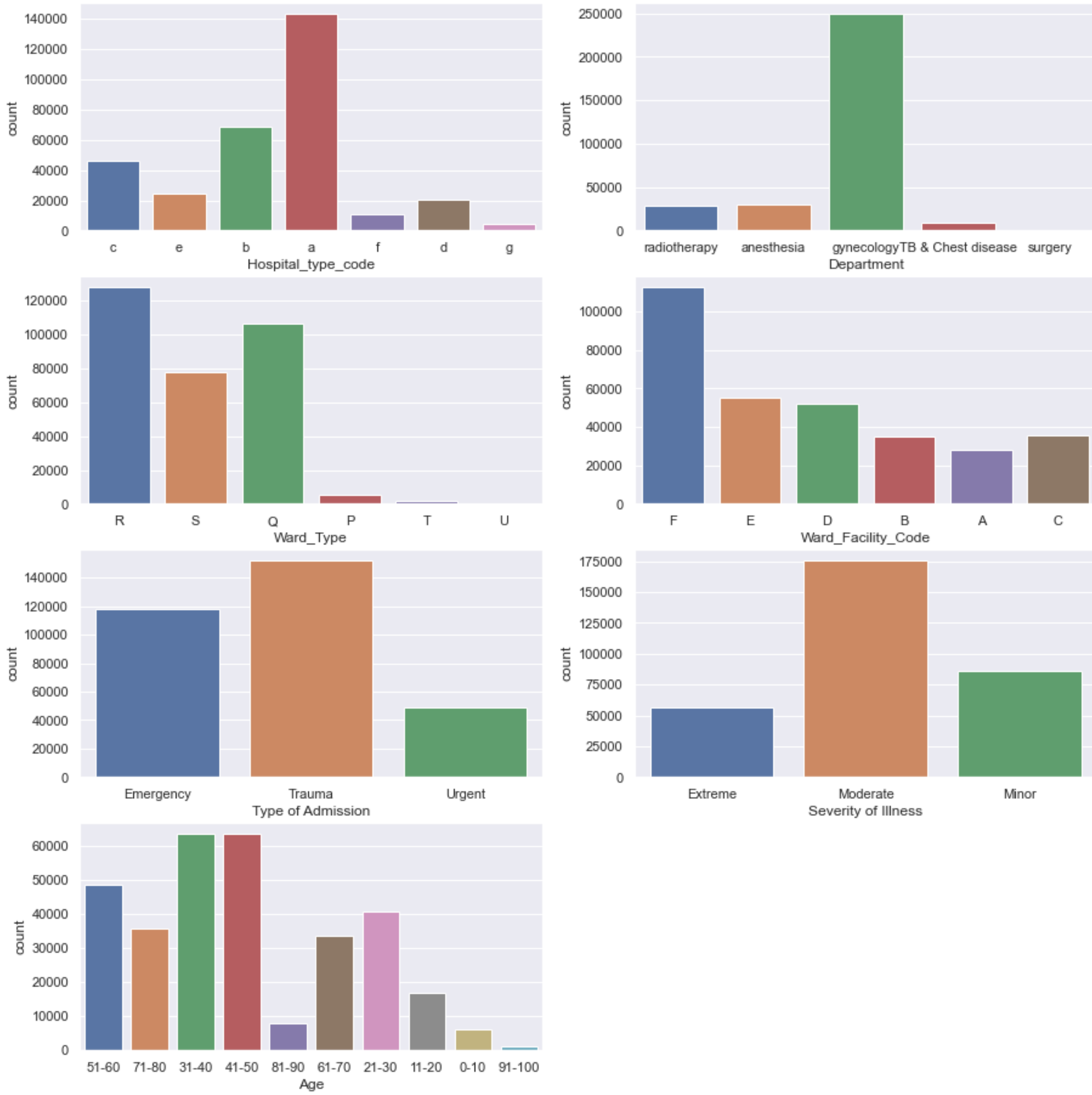
In [14]:
```
# Split data into numerical and categorical categories

num_data = df_train[['Hospital_code', 'City_Code_Hospital', 'Available Extra Roo
cat_data = df_train[['Hospital_type_code', 'Department', 'Ward_Type', 'Ward_Faci
```

Below I am going to conduct a simple EDA of the different variables in order to see if there are any anomalies or differences in the data. I will summarize below.

In [15]:
```
# Categorical data quick analysis to see any trends or key takeaways

i=1
plt.figure(figsize=(15,20))
for col in cat_data:
    plt.subplot(5,2,i)
    sns.countplot(df_train[col])
    i=i+1
plt.show()
```

```
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keywo
rd arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keywo
rd arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
```

```
seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keywo
rd arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keywo
rd arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keywo
rd arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keywo
rd arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keywo
rd arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
```



Categorical EDA Takeaways

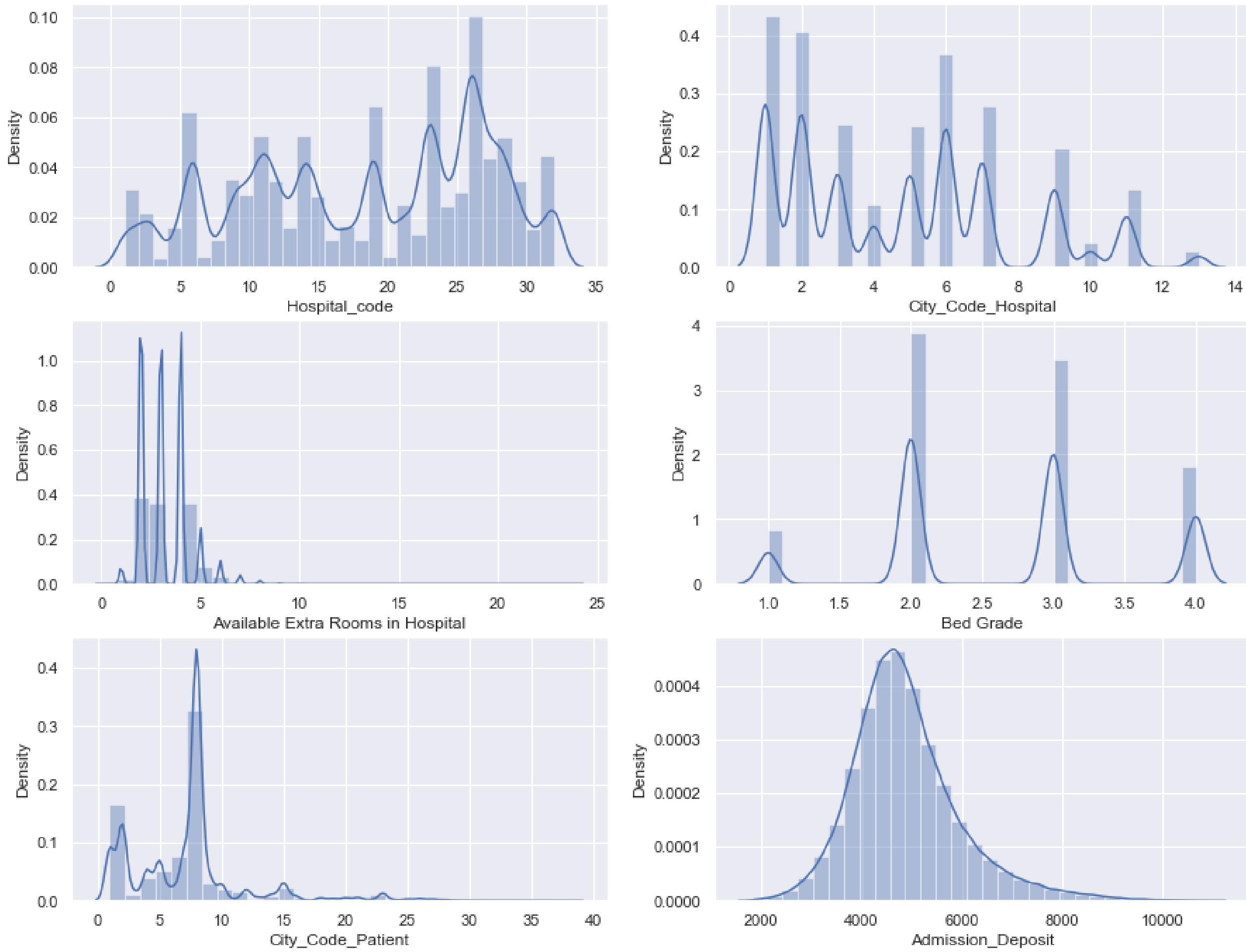
`Hospital type code` : the vast majority of hospitals types fall under code 'a'
`Department` : the majority of departments were gynecology which leads me to believe these were created as carry over departments for patient overflow
`Type of Admission` : most patient admissions fell under 'Trauma' or 'Emergency'
`Severity of Illness` : most patients arrived with 'Moderate' severity followed by 'Minor' then 'Extreme'
`Age` : very few patients were under the age of 20 and most fell between the ages of 31-60

Overall the data shows that there is a good distribution to work with and that these variables do indeed have predictive elements when it comes to modeling. Next I will look at the numerical features.

In [16]:
```
# Numerical data quick analysis

i=1
```

```python
plt.figure(figsize=(15,20))
for col in num_data:
    plt.subplot(5,2,i)
    sns.distplot(df_train[col], bins=30)
    i=i+1
plt.show()

# df.hist(bins=30, figsize=(15, 10))
```

```
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated functio
n and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an a
xes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated functio
n and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an a
xes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated functio
n and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an a
xes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated functio
n and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an a
xes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated functio
n and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an a
xes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated functio
n and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an a
xes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Numerical EDA Takeaways

`Bed Grade` : most bed grades were labeled as 2 or 3

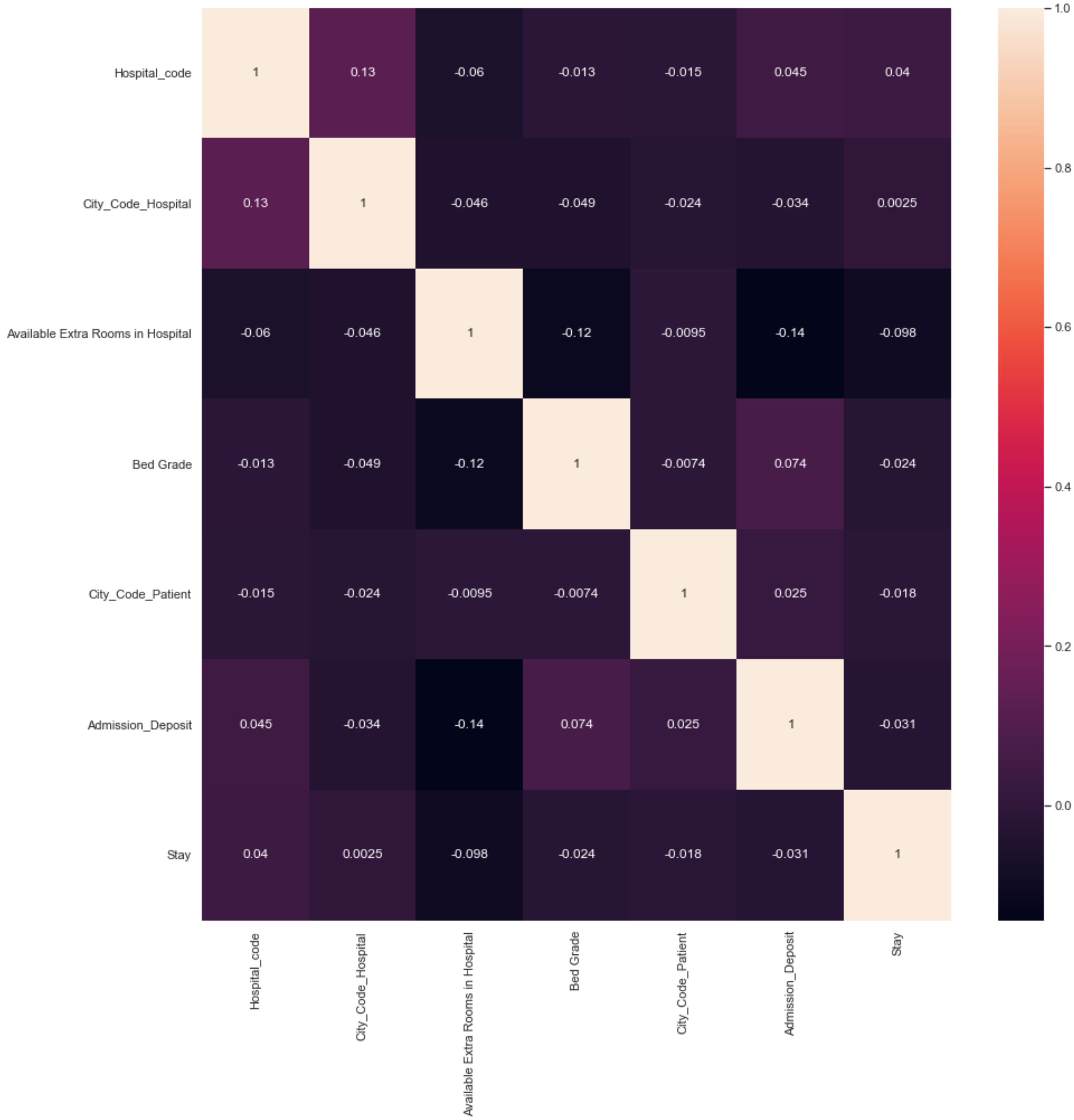
`City_Code_Patient` : a vast majority of patients came from city code 6

`Admission_Demposit` : on average, admission deposit fell around $4,900

No major concerns sticking out from the analysis. We are good to move on to pre-processing steps.

In [17]:
```
# Check to see if there are any autocorrelation issues in dataset

plt.figure(figsize=(15,15))
sns.heatmap(df_train.corr(), annot=True)
```

Out[17]: `<AxesSubplot:>`



V. Modeling

Train-Test Split

In [18]:
```
# Separate target 'Stay' from predictors

y = df_train['Stay']
X = df_train.drop('Stay', axis=1)

#print(y)
#print(X)
```

In [19]:
```
# Train-test split using test_size = 0.2

X_train, X_test, y_train, y_test= train_test_split(X,y,test_size= 0.2, random_st
```

Label Encoding Categorical Data

In order to use the categorical variables in my model, I have implemented the `LabelEncoder()` to give each variable a unique value:

In [20]:

```
# Label encode categorical predictors to given them each a unique value

labels = LabelEncoder()

for col in cat_data:
    X_train[col]= labels.fit_transform(X_train[col])
    X_test[col]= labels.fit_transform(X_test[col])


# Check changes

X_train.head()
```

```
<ipython-input-20-88b08c3be4bd>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train[col]= labels.fit_transform(X_train[col])
<ipython-input-20-88b08c3be4bd>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test[col]= labels.fit_transform(X_test[col])
```

Out[20]:

	Hospital_code	Hospital_type_code	City_Code_Hospital	Available Extra Rooms in Hospital	Department	Ward_Ty
231676	19	0	7	4	2	
166821	19	0	7	2	3	
70566	26	1	2	2	3	
197982	26	1	2	2	2	
280389	18	3	13	4	3	

In [21]:

```
df_train.dtypes
```

Out[21]:

```
Hospital_code                          int64
Hospital_type_code                    object
City_Code_Hospital                     int64
Available Extra Rooms in Hospital      int64
Department                            object
Ward_Type                             object
Ward_Facility_Code                    object
Bed Grade                            float64
City_Code_Patient                    float64
Type of Admission                     object
Severity of Illness                   object
Age                                   object
```

```
Admission_Deposit                    float64
Stay                                   int64
dtype: object
```

Scaling Numerical Data

In order to run my model, I now standardize all predictors using `StandardScaler()` :

In [22]:
```
# Scale numeric predictors using StandardScaler()

scaler = StandardScaler()

for col in num_data:
    X_train[col]= scaler.fit_transform(X_train[col].values.reshape(-1,1))
    X_test[col] = scaler.fit_transform(X_test[col].values.reshape(-1,1))

# Check changes

X_train.head()
```

```
<ipython-input-22-50bf0e07852e>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train[col]= scaler.fit_transform(X_train[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test[col] = scaler.fit_transform(X_test[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train[col]= scaler.fit_transform(X_train[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test[col] = scaler.fit_transform(X_test[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train[col]= scaler.fit_transform(X_train[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test[col] = scaler.fit_transform(X_test[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:6: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train[col]= scaler.fit_transform(X_train[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test[col] = scaler.fit_transform(X_test[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train[col]= scaler.fit_transform(X_train[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test[col] = scaler.fit_transform(X_test[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train[col]= scaler.fit_transform(X_train[col].values.reshape(-1,1))
<ipython-input-22-50bf0e07852e>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test[col] = scaler.fit_transform(X_test[col].values.reshape(-1,1))
```

Out[22]:

	Hospital_code	Hospital_type_code	City_Code_Hospital	Available Extra Rooms in Hospital	Department	Ward_T
231676	0.079564	0	0.717568	0.685853	2	
166821	0.079564	0	0.717568	-1.024005	3	
70566	0.890558	1	-0.893221	-1.024005	3	
197982	0.890558	1	-0.893221	-1.024005	2	
280389	-0.036292	3	2.650514	0.685853	3	

Baseline Model: Decision Tree

In [23]:

```
# First simple model created as Decision Tree

dt_clf = DecisionTreeClassifier()
dt_model = dt_clf.fit(X_train, y_train)
```

```
# Cross validation score

cv_score = np.mean(cross_val_score(dt_clf, X_train, y_train, cv=5))
cv_score
```

Out[23]: 0.438135426889107

In [24]:
```
# Decision Tree predictions

dt_y_preds = dt_clf.predict(X_test)
```

In [25]:
```
# Classification report and accuracy score

print(classification_report(y_test, dt_y_preds))

accuracy = accuracy_score(y_test, dt_y_preds)
print(accuracy)
```

```
              precision    recall  f1-score   support

           0       0.39      0.40      0.39     20250
           1       0.53      0.52      0.53     30941
           2       0.31      0.32      0.32     12497

    accuracy                           0.44     63688
   macro avg       0.41      0.41      0.41     63688
weighted avg       0.44      0.44      0.44     63688

0.44138613239542773
```

Model 2: Random Forest

In [26]:
```
# Second model created as Random Forest

# rf_clf = RandomForestClassifier()
# rf_model = rf_clf.fit(X_train, y_train)

# # Cross validation score

# cv_score = np.mean(cross_val_score(dt_clf, X_train, y_train, cv=5))
# cv_score
```

In [27]:
```
# Random Forest predictions

# rf_y_preds = rf_clf.predict(X_test)
```

In [28]:
```
# Classification report and acuracy score

# print(classification_report(y_test, rf_y_preds))

# accuracy = accuracy_score(y_test, rf_y_preds)
# print(accuracy)
```

Model 3: XGBoost

In [29]:
```
# Third model created as XGBoost

# xgb_clf = XGBClassifier()
```

```
# xgb_model = xgb_clf.fit(X_train, y_train)


# # Cross validation score

# cv_score = np.mean(cross_val_score(xgb_clf, X_train, y_train, cv=5))
# cv_score
```

In [30]:
```
# XGBoost predictions

# xgb_y_preds = xgb_clf.predict(X_test)
```

In [31]:
```
# Classification report and acuracy score

# print(classification_report(y_test, xgb_y_preds))

# accuracy = accuracy_score(y_test, xgb_y_preds)
# print(accuracy)
```

In [32]:
```
# Feature importance function taken from class lecture

def plot_feature_importances(model):
    n_features = X_train.shape[1]
    plt.figure(figsize=(8,8))
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), X_train.columns.values)
    plt.xlabel('Feature importance')
    plt.ylabel('Feature')
```

XGBoost Hyperparameter Tuning

In [33]:
```
# Grid search for optimal parameters for XGBoost model

# param_grid = {
#     'max_depth': [1, 3, 5, 10],
#     'subsample': [0.2, 0.3, 0.4, 0.5],
#     'n_estimators': [100, 200, 500, 600],
# }

# grid_xgb = RandomizedSearchCV(xgb_clf, param_grid, scoring='accuracy', cv=None
# grid_xgb.fit(X_train, y_train)

# best_parameters = grid_xgb.best_params_

# print('Grid Search found the following optimal parameters: ')
# for param_name in sorted(best_parameters.keys()):
#     print('%s: %r' % (param_name, best_parameters[param_name]))

# training_preds = grid_xgb.predict(X_train)
# test_preds = grid_xgb.predict(X_test)
# training_accuracy = accuracy_score(y_train, training_preds)
# test_accuracy = accuracy_score(y_test, test_preds)

# print('')
# print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
# print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

In [34]:
```
# Best estimator based on predictions
```

```
# grid_xgb.best_estimator_
```

Final Model: Tuned XGBoost

In [35]:
```
# XGBoost model with best estimator hyperparameter tuning

xgb_clf_final = XGBClassifier(base_score=0.5, booster='gbtree', colsample_byleve
              colsample_bynode=1, colsample_bytree=1, gamma=1, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=5,
              min_child_weight=8, monotone_constraints='()',
              n_estimators=600, n_jobs=4, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
xgb_model_final = xgb_clf_final.fit(X_train, y_train)

# Cross validation score

cv_score_final = np.mean(cross_val_score(xgb_clf_final, X_train, y_train, cv=5))
cv_score_final
```

Out[35]:
```
0.5414249263984299
```

In [36]:
```
# Final XGBoost predictions

xgb_y_preds = xgb_clf_final.predict(X_test)
```

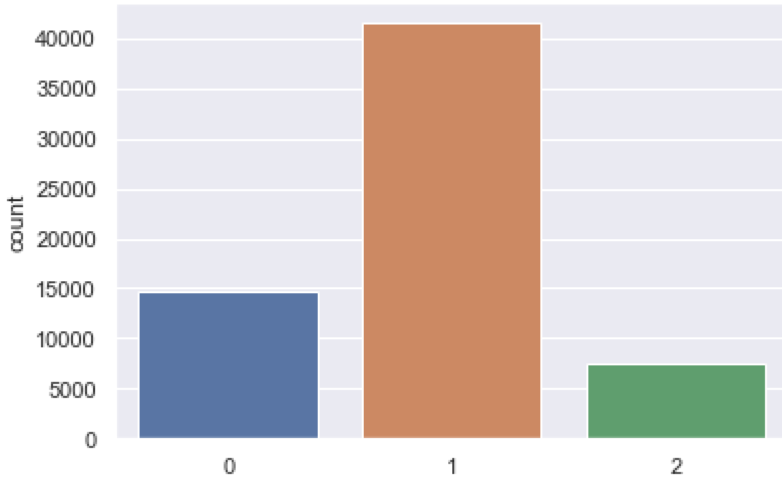
In [37]:
```
# print(confusion_matrix(y_test, rf_y_preds))
print(classification_report(y_test, xgb_y_preds))

accuracy = accuracy_score(y_test, xgb_y_preds)
print(accuracy)
```

```
              precision    recall  f1-score   support

           0       0.53      0.38      0.44     20250
           1       0.56      0.75      0.64     30941
           2       0.51      0.31      0.38     12497

    accuracy                           0.54     63688
   macro avg       0.53      0.48      0.49     63688
weighted avg       0.54      0.54      0.53     63688

0.544388267805552
```

In [38]:
```
sns.countplot(xgb_y_preds)
```

```
/Users/andrewmarinelli/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keywo
rd arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
```

Out[38]:
```
<AxesSubplot:ylabel='count'>
```



In [39]:
```
(unique, counts) = np.unique(xgb_y_preds, return_counts=True)
frequencies = np.asarray((unique, counts)).T

print(frequencies)
```

```
[[    0 14634]
 [    1 41512]
 [    2  7542]]
```

In [40]:
```
y_test.value_counts().sort_values()
```

Out[40]:
```
2    12497
0    20250
1    30941
Name: Stay, dtype: int64
```

VI. Results

Baseline Model: Decision Tree

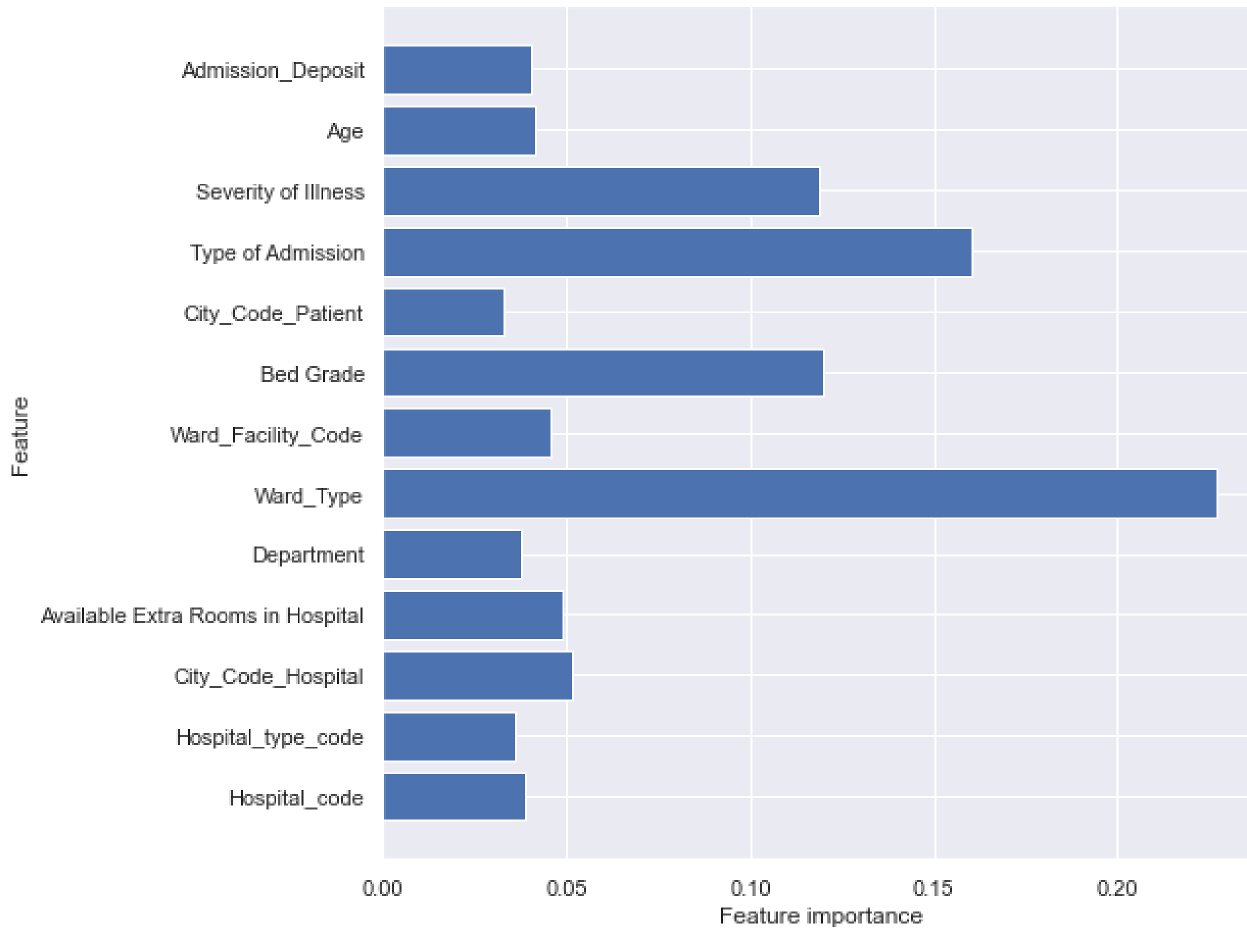
My baseline model used a Decision Tree classifier to classify the three labels ("Short-Term", "Medium-Term", and "Long-Term") with an accuracy score of `44.1%` .

Final Model: Tuned XGBoost

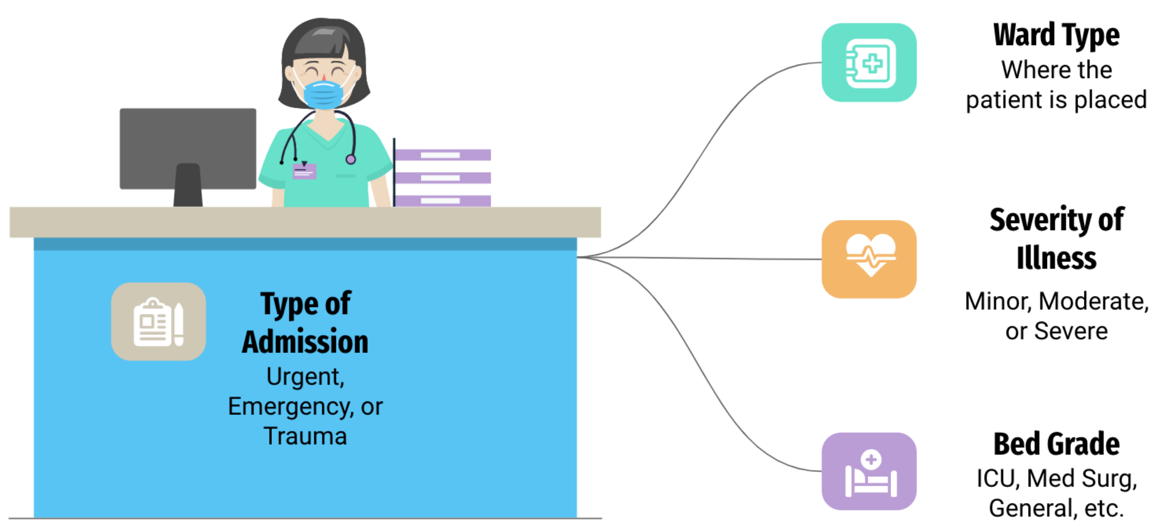
My final model used a tuned XGBoost classifier to classify the three labels (`Short-Term` , `Medium-Term` , and `Long-Term`) with an accuracy score of `54.4%` .

Below I have graphed the features with the most influenece on the model:

In [41]:
```
plot_feature_importances(xgb_clf_final)
```



Based on the modeling and feature importance, the visuals below show how the model can be implemented in a hospital setting.



Upon deployment, the following steps would be followed:

1. Admission

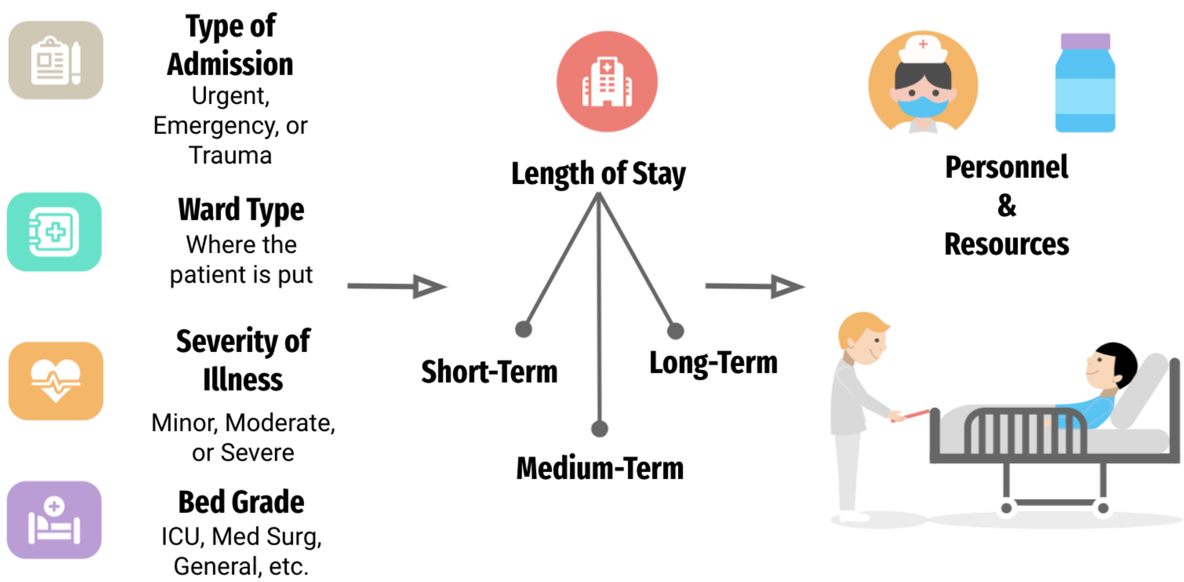
- Upon admission, patient profile would be used to predict "Short-Term", "Medium-Term", or "Long-term" stay based on key predictors.

2. Placement

- Based on classification, the patient could then be placed in a section of the hospital dedicated to patients with a similar profile.

3. Treatment & Discharge

- Resources and personnel can be allocated in an appopriate manner in order to best meet patient needs and expedite the discharge process.

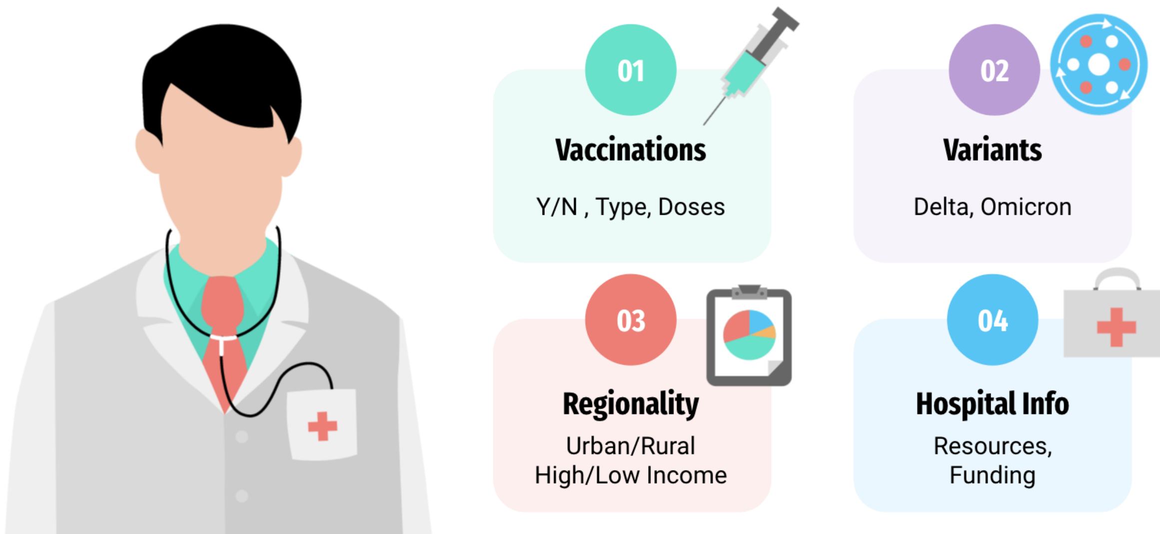


In the context of the business problem, having a 54.4% accuracy score is not bad. The model aims at being a predictor to support the placement of patients in a hospital to improve the use of limited resources and personnel. By being able to place more individuals in the correct assignment, the hospitals will be able to make data-driven decisions on staffing and resource management.

With more patients placed in better hospital settings, they will be able to provide more appropriate healthcare services and expedite the discharge process. This will make room for additional patients whon need treatment without overwhelming the current system.

VII. Future Research

The data lent itself to a nice analysis of Length of Stay in the context of the problem I set out to work on. In addition to the data provided, I would be interested to see how other factors play a role in LOS predictions. In particular, I am interested in:



1. Vaccinations

- Did the individual recieve a vaccination?
- What type?
- How many doses?

2. Variants

- Does being infected with the Delta or Omicron variant affect a LOS prediction?

3. Regionality

- Does being in an urban/rural area affect LOS prediction?
- Does community income level play a role?

4. Hospital Information

- Does hospital funding play a significant role?
- What resources are available to begin with?

Additional EDA

In [52]:
```
# plt.hist(df_train['Admission_Deposit'])
# df_train['Admission_Deposit'].value_counts()
```

In [53]:
```
# plt.hist(df_train['Age'])
# df_train['Age'].value_counts()
```

In [54]:
```
# plt.hist(df_train['Bed Grade'])
# df_train['Bed Grade'].value_counts()
```

In [55]:
```
# plt.hist(df_train['City_Code_Patient'])
# df_train['City_Code_Patient'].value_counts()
```

In [56]:
```
# plt.hist(df_train['Type of Admission'])
# df_train['Type of Admission'].value_counts()
```

In [57]:
```
# plt.hist(df_train['Department'])
# df_train['Department'].value_counts()
```

In [58]:
```
# plt.hist(df_train['Ward_Facility_Code'])
# df_train['Ward_Facility_Code'].value_counts()
```

In [59]:
```
# plt.hist(df_train['Ward_Type'])
# df_train['Ward_Type'].value_counts()
```

In [60]:
```
# plt.hist(df_train['Available Extra Rooms in Hospital'])
# df_train['Available Extra Rooms in Hospital'].value_counts()
```

In [61]:
```
# plt.hist(df_train['Visitors with Patient'])
# df_train['Visitors with Patient'].value_counts()
```

In [62]:
```
# plt.hist(df_train['Hospital_region_code'])
# df_train['Hospital_region_code'].value_counts()
```

In [63]:
```
# plt.hist(df_train['Severity of Illness'])
# df_train['Severity of Illness'].value_counts()
```

In [64]:
```
# plt.hist(df_train['Hospital_code'])
# df_train['Hospital_code'].value_counts()
```

In [65]:
```
# plt.hist(df_train['City_Code_Hospital'])
# df_train['City_Code_Hospital'].value_counts()
```

In [66]:
```
# plt.hist(df_train['Hospital_type_code'])
# df_train['Hospital_type_code'].value_counts()
```