COSC 2673/2793 | Machine Learning

Assignment 1: Introduction to Machine Learning

Student Name: Shekhar Hazari

Student ID: S3831855

Introduction to problem

- How does an individual choose what hospital to go to if they have a condition which requires the individual to be admitted in for care? The natural answer comes down to how efficient a hospital is. One such measure of efficiency is Average Length of stay.The average length of stay in hospitals (ALOS) is often used as an indicator of efficiency. All other things being equal, a shorter stay will reduce the cost per discharge and shift care from inpatient to less expensive post acute settings. The ALOS refers to the average number of days that patients spend in hospital. It is generally measured by dividing the total number of days stayed by all inpatients during a year by the number of admissions or discharges. Day cases are excluded. The indicator is presented both for all acute care cases and for childbirth without complications.
- With this in mind, our task in this assignment will be to distinguish the when a longer stay will be required than an acute care case. If a patient is admitted for less than 4 days, it is termed as an acute care case. A longer stay in hospital will be classified as 1, on the other hand an acute care case will be classified as 0.

In [1]:

```
# Importing required libraries and
!pip install graphviz
import numpy as np
import pandas as pd
pd.set_option("display.precision", 3)
from pandas.api.types import is_string_dtype
from pandas.api.types import is_numeric_dtype
import matplotlib.pyplot as plt
import seaborn as sns
%config InlineBackend.figure_format = 'retina'
```

```
Requirement already satisfied: graphviz in c:\users\shekh\anaconda3\lib\site-packages
(0.16)
```

In [2]:

```
# Reading the data from CSV
# Ignoring column 'HealthServiceArea' (as indicated in the Assignment specification) an
# so that none of these columns are used as training attributes.
data = pd.read_csv('train_data.csv', index_col = 'ID', usecols = ['ID', 'Gender', 'Race
                                                                  'APRSeverityOfIllness
                                                                 'EmergencyDepartmentIn
                                                                  'AverageChargesInCoun
                                                                  'AverageChargesInFaci
```

In [3]:
```
# Looking at the first few rows of the data
data.head()
```

Out[3]:

ID	Gender	Race	TypeOfAdmission	CCSProcedureCode	APRSeverityOfIllnessCode	PaymentTypo
1	F	Other Race	Newborn	228	1	Med
2	M	Black/African American	Newborn	228	1	Med
3	M	Other Race	Newborn	220	1	Private H Insur
4	F	Other Race	Newborn	0	1	Private H Insur
5	F	Other Race	Newborn	228	1	Med

In [4]:
```
# Checking the size of the dataset, if there are null values and the data types of vari
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59966 entries, 1 to 59966
Data columns (total 14 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   Gender                       59966 non-null  object
 1   Race                         59966 non-null  object
 2   TypeOfAdmission              59966 non-null  object
 3   CCSProcedureCode             59966 non-null  int64 
 4   APRSeverityOfIllnessCode     59966 non-null  int64 
 5   PaymentTypology              59966 non-null  object
 6   BirthWeight                  59966 non-null  int64 
 7   EmergencyDepartmentIndicator 59966 non-null  object
 8   AverageCostInCounty          59966 non-null  int64 
 9   AverageChargesInCounty       59966 non-null  int64 
 10  AverageCostInFacility        59966 non-null  int64 
 11  AverageChargesInFacility     59966 non-null  int64 
 12  AverageIncomeInZipCode       59966 non-null  int64 
 13  LengthOfStay                 59966 non-null  int64 
dtypes: int64(9), object(5)
memory usage: 6.9+ MB
```

Reading the data

- While reading the data, the column *HealthServiceArea* was ignored and column 'ID' was used as index (this won't be used in model training).
- Printing the first few rows of the data shows the import was successful and as expected.
- We can see that the data has 59966 rows (number of patients) and 14 columns.
- All these columns are not in the desired type for example *CCSProcedureCode* has been imported as an int data type where it's a category of procedure used. These type conversions

will be done in next section

In [5]:
```
# Converting columns to correct datatypes based on attribute definitions in assignment
cat_col = ['Gender', 'Race', 'TypeOfAdmission', 'CCSProcedureCode',
           'PaymentTypology', 'EmergencyDepartmentIndicator']
num_col = ['AverageCostInCounty','AverageChargesInCounty', 'AverageCostInFacility',
           'AverageChargesInFacility', 'AverageIncomeInZipCode', 'BirthWeight']
data.loc[data.LengthOfStay < 4, "LengthOfStay"] = 0
data.loc[data.LengthOfStay > 3, "LengthOfStay"] = 1
for column_name in cat_col:
    data[column_name] = data[column_name].astype('category')
```

In [6]:
```
# Displaying the data information again
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59966 entries, 1 to 59966
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Gender                        59966 non-null  category
 1   Race                          59966 non-null  category
 2   TypeOfAdmission               59966 non-null  category
 3   CCSProcedureCode              59966 non-null  category
 4   APRSeverityOfIllnessCode      59966 non-null  int64
 5   PaymentTypology               59966 non-null  category
 6   BirthWeight                   59966 non-null  int64
 7   EmergencyDepartmentIndicator  59966 non-null  category
 8   AverageCostInCounty           59966 non-null  int64
 9   AverageChargesInCounty        59966 non-null  int64
 10  AverageCostInFacility         59966 non-null  int64
 11  AverageChargesInFacility      59966 non-null  int64
 12  AverageIncomeInZipCode        59966 non-null  int64
 13  LengthOfStay                  59966 non-null  int64
dtypes: category(6), int64(8)
memory usage: 4.5 MB
```

All the data types have been successfully converted and we can begin our exploratory data analysis

First Glance at the data:

There are approximately 60K observations in the dataset with a total of 13 attributes for model training and 1 target variable.

There are apparently no *NULL* values which is a good sign.

The attributes are a good mix of boolean, categorical, ordinal and numeric type.

In [7]:
```
data.head()
```

Out[7]:

	Gender	Race	TypeOfAdmission	CCSProcedureCode	APRSeverityOfIllnessCode	PaymentTypo
ID						
1	F	Other Race	Newborn	228	1	Med

ID	Gender	Race	TypeOfAdmission	CCSProcedureCode	APRSeverityOfIllnessCode	PaymentTypo
2	M	Black/African American	Newborn	228	1	Med
3	M	Other Race	Newborn	220	1	Private H Insur
4	F	Other Race	Newborn	0	1	Private H Insur
5	F	Other Race	Newborn	228	1	Med

In [8]:

```
# Printing exploratory stats for all numeric columns in the dataset
data.describe()
```

Out[8]:

	APRSeverityOfIllnessCode	BirthWeight	AverageCostInCounty	AverageChargesInCounty	AverageC
count	59966.000	59966.000	59966.000	59966.000	
mean	1.255	3336.299	2372.807	7979.127	
std	0.546	446.244	639.755	3220.291	
min	1.000	2500.000	712.000	1243.000	
25%	1.000	3000.000	2041.000	4620.000	
50%	1.000	3300.000	2533.000	9227.000	
75%	1.000	3600.000	2785.000	10644.000	
max	4.000	7500.000	3242.000	11381.000	

Initial impressions

BirthWeight seems close to normal at first glance as mean and median values are fairly close. But a similar inference can not be made for other variables. Will explore this in more detail in later sections.

In [9]:

```
# Printing exploratory stats for non-numeric columns in the datset
data.describe(include=['category'])
```

Out[9]:

	Gender	Race	TypeOfAdmission	CCSProcedureCode	PaymentTypology	EmergencyDepartment
count	59966	59966	59966	59966	59966	
unique	3	4	4	7	9	
top	M	White	Newborn	228	Medicaid	
freq	30978	32943	58741	19886	28723	

Gender

Has a total of 3 unique values Male being the most frequent. Need to explore the frequency of other two categories.

Race

Has a total of 4 unique values White being the most frequent, which can be expected as the dataset is from the USA. Need to explore the frequency of other three categories.

TypeOfAdmission

Has a total of 4 unique values Newborn being the most frequent, and it seems almost all cases fall into this category (58741 of 59966). Need to explore the frequency of other three categories.

CCSProcedureCode

Has a total of 7 unique values 228 being the most frequent.

APRSeverityOfIllnessCode

Has a total of 4 unique values '1' being the most frequent.

PaymentTypology

Has a total of 9 unique values 'Medicaid' being the highest.

EmergencyDepartmentIndicator

Has a total of 2 unique values '0' being the most frequent, almost all cases fall in this category. That is expected given that there are lesser emergency cases. What would be interesting is to see break-up of these values by *LengthOfStay*

In [10]:

```
plt.figure(figsize=(20,20))
for i, column in enumerate(data.columns):
    plt.subplot(5,3,i+1)
    if is_numeric_dtype(data[column]):
        plt.hist(data[column], alpha=0.3, color='b', density=True)
        plt.title(column)
    else:
        data[column].value_counts().plot.barh()
        plt.title(column)
```

Frequencies

Gender:

There seems to be no or very little examples for *unknown* category.

APRSeverityOfIllness:

There seems to be no or very little examples for *4* category.

PaymentTypology:

There seems to be no or very little examples for *unknown, medicare and miscellenious/other* categories.

Distributions:

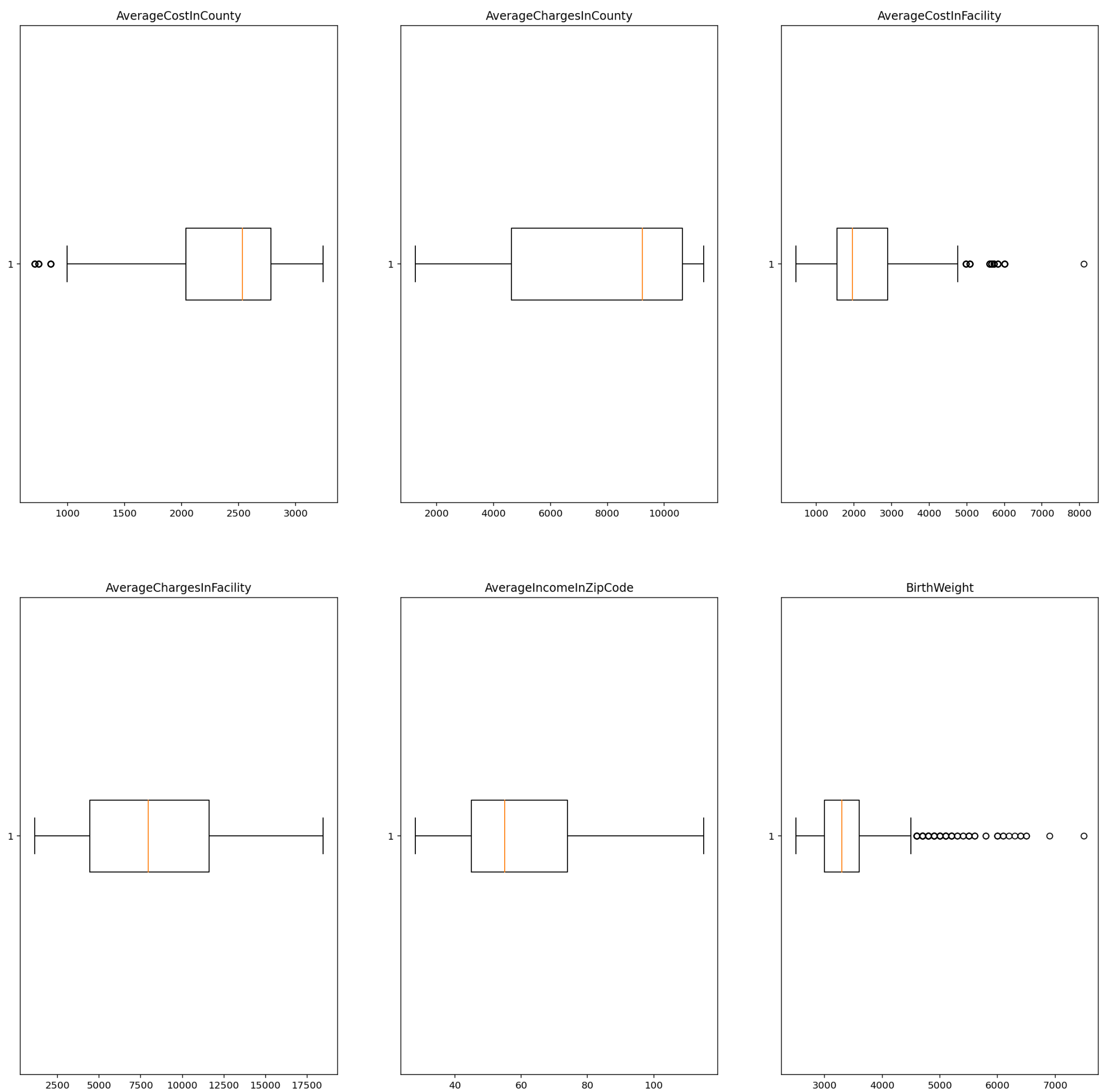
BirthWeight, AverageCostInFacility and AverageChargesInFacility:

Seems fairly normally distributed. Further investigation in distributions of numerical variables is required

LengthOfStay

Has a total of 2 unique values '0' being the most frequent, which means most cases are acute care level. But quite clearly there will be a problem of class imbalance while training the ML model.

In [11]:
```
plt.figure(figsize=(20,20))
i = 1
for column in num_col:
    if is_numeric_dtype(data[column]):
        plt.subplot(2,3,i)
        i = i+1
        plt.boxplot(data[column], vert=False)
        plt.title(column)
```

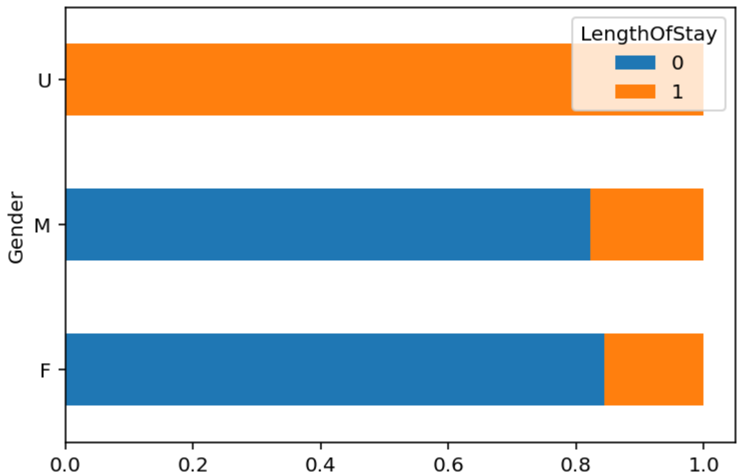


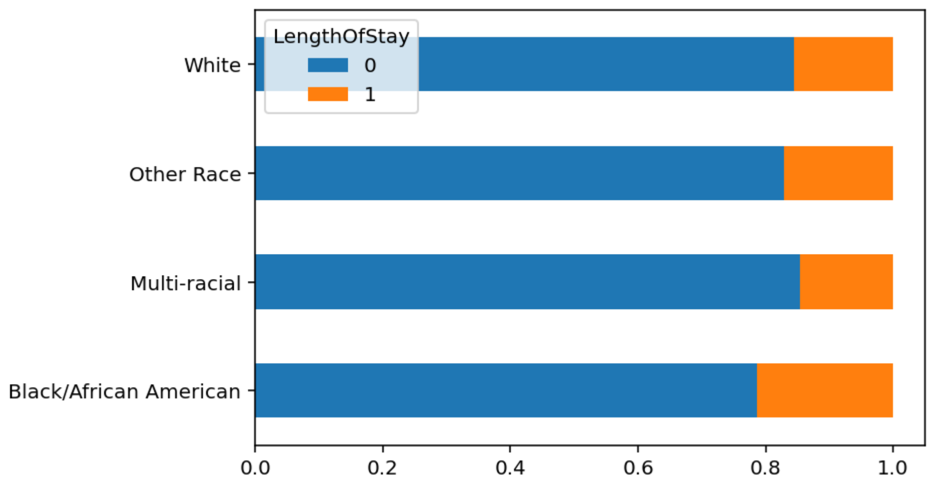
Observation:

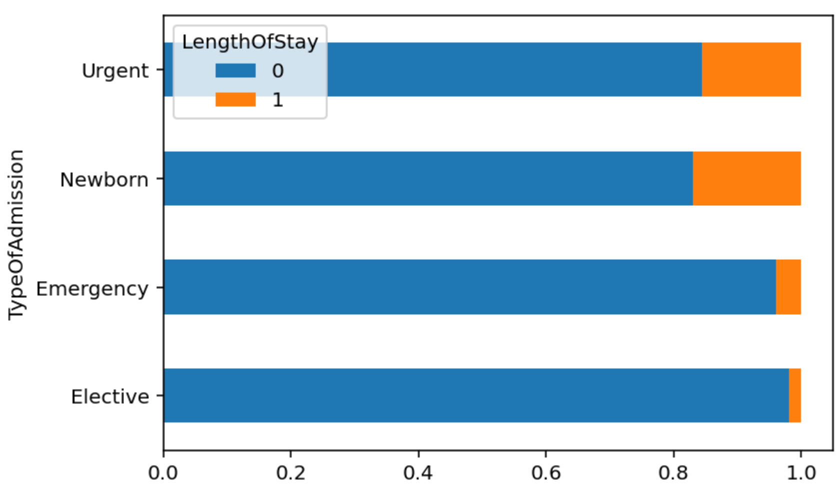
- *Birthweight, AverageCostInCounty* and *AverageCostInFacility* seem to have some outliers. Will need to take this into account while transforming the data.
- Other attributes do not seem to be normally distributed as well, all have some degree of skewness in them and this will also be addressed while transforming these columns.

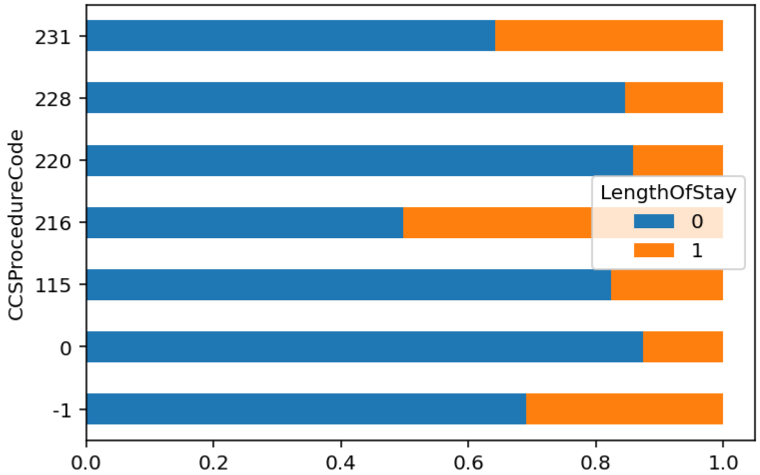
In [12]:

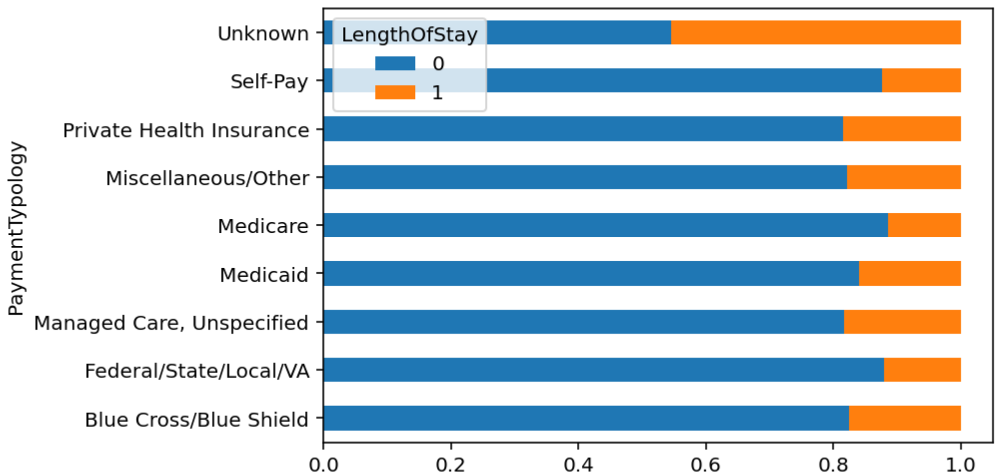
```
for column in data.columns:
    if not(is_numeric_dtype(data[column])):
        pd.crosstab(data[column], data['LengthOfStay'], margins = False, normalize='ind
```

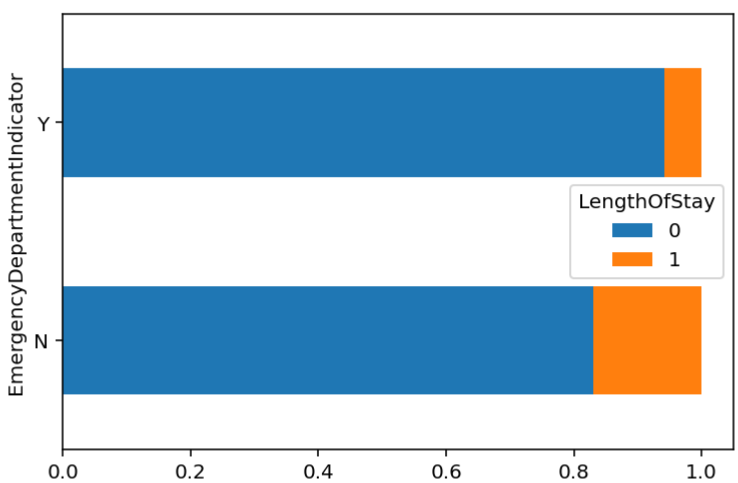












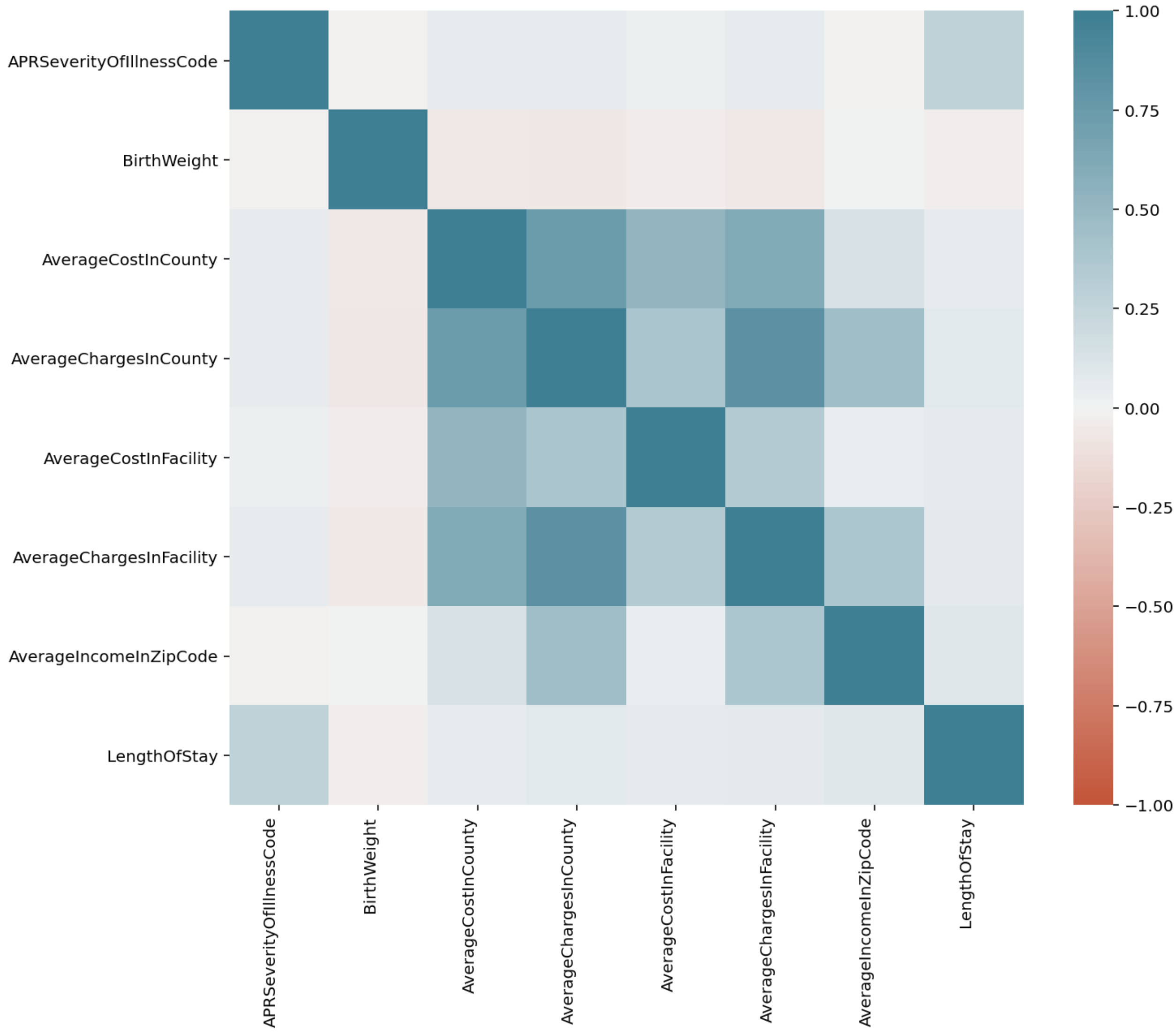
Observations:

- Higher *APRSeverityOfIllnessCode* is more likely to extend the stay of the patient.
- Patients who have *CCSProcedureCode* -1, 216 or 231 are more likely to stay beyond acute care length.

In [13]:

```
f, ax = plt.subplots(figsize=(11, 9))
corr = data.corr()
ax = sns.heatmap(
    corr,
```

```
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=90,
    horizontalalignment='right'
);
```



From Correlation plot:

- There are a few variables which are colrrelated within themselves. For example *AverageCostInCounty and AverageChargesInCounty* seem fairly correlated around the 0.5 mark but not correlated enough (more than 0.7) to remove one of these variables for the purpose of model training.

In [14]:

```
sns.pairplot(data, kind="scatter", hue="LengthOfStay", palette="Set2")
plt.show()
```

Initial impressions from the correlogram

- It seems that the babies with higher birth weights have acute care because they are healthier. It is ususal for underweight babies to remain in care for a few extra days, so this makes sense.
- Rest of the columns do not show any clear patterns in the data.

performance and evaluation metrics

- The goal of the analysis is to predict whether the length of stay will be greater than 3 days or otherwise.
- Since Average length of stay is often an indicator of customer satisfaction and hospital effciency. It is needed that a balance is struck between precision and accuracy of results, as the goal is to improve discharge planning.
- Since importance has to be given to both class 0 (acute care cases) as well as class 1 (longer care cases) but there is a clear case of class imbalance I will use 'macro_f1_score'

- As there is a clear case of class imbalance with 83% cases falling in class 0. I am targeting a model with over 60% f1 score.
- For the purpose of model training and evaluation, I will be using k-fold cross validation as the size of the data set is not huge.
- First I'll split the datset in training and testing sets then apply k-fold cross vaidation for model tuning and on training set then finally test the model on test set.

Using Logistic Regression to set a baseline model performance

In the below model the folowing steps will be taken:

1. Create polynomial features
2. Create one hot encoder for categorical values
3. Create a quantile transformer for bringing the numerical values in a gaussian shape
4. Create a column transformer to embed all preprocessing steps.
5. Create training and test set
6. Create a grid of parameters to fit logistic model and apply regularization
7. Create a logistic classifier
8. Embed transformation steps and classifier in a Pipeline
9. Run randomized search for parameters of Logistic model pipeline
10. Check for over/underfitting on test set.

In [22]:
```
## Importing required libraries for building a logistic regression pipeline

from sklearn.preprocessing import QuantileTransformer
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV

## Create a copy of the original data
data_logit_models = data

## Create a list of c values for regularization
c_values = np.linspace(start = 0.001, stop = 0.3, num = 50)

## Create a column transformer to embed all transformation steps in one step
column_trans_logit_model = make_column_transformer(
    (PolynomialFeatures(3), num_col),
    (OneHotEncoder(handle_unknown='ignore'), cat_col),
    (QuantileTransformer(output_distribution='normal'), num_col),
    remainder = 'passthrough')

## Create training and testing set
with pd.option_context('mode.chained_assignment', None):
    train_data, test_data = train_test_split(data_logit_models, test_size=0.2,
                                             shuffle=True,random_state=0)
```

```
## Printing the shape of training and test set
print("Number of training examples: ", train_data.shape[0])
print("Number of testing examples: ", test_data.shape[0])


## Splitting training set in features and label
train_X = train_data.drop('LengthOfStay', axis='columns')
train_y = train_data[['LengthOfStay']]

## Splitting test set set in features and label
test_X = test_data.drop('LengthOfStay', axis='columns')
test_y = test_data[['LengthOfStay']]

## Creating a parameters list for logistic regression
logit_param_grid = [{'model__penalty': ['l2'],
                     'model__C': c_values
                    }]

## Initializing a logistic classifier
logit_clf = LogisticRegression(class_weight='balanced', random_state=22, max_iter = 100

## Building a pipeline, combining all steps
logit_pipeline = Pipeline( [('column_transformer',column_trans_logit_model),
                            ('model', logit_clf)
                           ])

## Creating a Randomized search CV instance using the pipeline and regularization param
logit_rand_search = RandomizedSearchCV(estimator = logit_pipeline, param_distributions
                                       scoring='f1_macro')

## Fitting the rand search model
logit_rand_search_model = logit_rand_search.fit(train_X, train_y.values.ravel())

## Printing the results
print(logit_rand_search_model.best_params_)
print("The f1_macro score of the best model is: ", logit_rand_search_model.best_score_)
```

```
Number of training examples:  47972
Number of testing examples:  11994
{'model__penalty': 'l2', 'model__C': 0.2633877551020408}
The f1_macro score of the best model is:  0.5440238144804183
```

In [28]:

```
logit_rand_search_model.score(test_X, test_y)
```

Out[28]:

```
0.5450730300708628
```

In [41]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
y_pred_logit = logit_rand_search_model.predict(test_X)
print(classification_report(test_y, y_pred_logit))
```

```
              precision    recall  f1-score   support

           0       0.88      0.65      0.75      9943
           1       0.25      0.56      0.34      2051

    accuracy                           0.64     11994
   macro avg       0.56      0.60      0.55     11994
```

```
weighted avg       0.77      0.64      0.68     11994
```

Observations:

- The logistic models best performance is around the 54% mark for macro F1 score. In further analysis, this will be the baseline model.
- The model still performs very poorly for the class 1

Decision Tree and Random Forest

1. Splitting the data in training, validation and test sets.
2. Convert categorical values to suitable continuous format. Using onehot encoding because test_data has extra categories which onehot encoding can handle gracefully.

In [23]:

```
from sklearn.model_selection import GridSearchCV
# Step 1: Splitting the data in training, validation and test sets.
data_tree_based_models = data
with pd.option_context('mode.chained_assignment', None):
    train_data, test_data = train_test_split(data_tree_based_models, test_size=0.2,
                                              shuffle=True,random_state=0)

print("Number of training examples: ", train_data.shape[0])
print("Number of testing examples: ", test_data.shape[0])

train_X = train_data.drop('LengthOfStay', axis='columns')
train_y = train_data[['LengthOfStay']]

test_X = test_data.drop('LengthOfStay', axis='columns')
test_y = test_data[['LengthOfStay']]

# Step 2: Convert categorical values to suitable continuous format

column_trans_tree_models = make_column_transformer(
    (OneHotEncoder(handle_unknown='ignore'), cat_col),
    remainder = 'passthrough'
)

# Defining Decision tree model
from sklearn.tree import DecisionTreeClassifier

# Tree limiting parameters
param_grid = [{'model__max_depth': np.arange(2,200, 20),
               'model__min_samples_split': np.arange(20,200,20),
               'model__criterion': ['gini', 'entropy']
              }
             ]
dt_clf = DecisionTreeClassifier(class_weight='balanced')


DT_pipeline = Pipeline( [('ohe',column_trans_tree_models),
                         ('model', dt_clf)
                        ]
                      )

grid_search_manual_param = GridSearchCV(estimator=DT_pipeline, param_grid=param_grid, s
```

```
best_model = grid_search_manual_param.fit(train_X, train_y)

print(best_model.best_estimator_)
print("The f1_macro score of the best model is: ", best_model.best_score_)
```

```
Number of training examples:  47972
Number of testing examples:  11994
Fitting 5 folds for each of 180 candidates, totalling 900 fits
Pipeline(steps=[('ohe',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(handle_unknown='ignor
e'),
                                                  ['Gender', 'Race',
                                                   'TypeOfAdmission',
                                                   'CCSProcedureCode',
                                                   'PaymentTypology',
                                                   'EmergencyDepartmentIndicator'])])),
                ('model',
                 DecisionTreeClassifier(class_weight='balanced',
                                        criterion='entropy', max_depth=22,
                                        min_samples_split=180))])
The f1_macro score of the best model is:  0.6114963167480513
```

In [42]:

```
y_pred_DT_RS = best_model.predict(test_X)
print(classification_report(test_y, y_pred_DT_RS))
```

```
              precision    recall  f1-score   support

           0       0.91      0.71      0.80      9943
           1       0.33      0.67      0.44      2051

    accuracy                           0.71     11994
   macro avg       0.62      0.69      0.62     11994
weighted avg       0.81      0.71      0.74     11994
```

Observation:

- The Decision trees show a significant improvement in the performance with an F1_macro score of 0.62%
- Also the predictor is not overfitted so that is a good sign.

In [25]:

```
path = DT_pipeline[-1].cost_complexity_pruning_path(DT_pipeline[:-1].fit_transform(trai

param_grid_post_pruning = [{'model__ccp_alpha': path.ccp_alphas[:-1]
                           }]

rand_search_post_pruning = RandomizedSearchCV(estimator=DT_pipeline, param_distribution
                                              scoring='f1_macro', n_jobs=-1, n_iter = 1
best_model_post_pruning = rand_search_post_pruning.fit(train_X, train_y)

print(best_model_post_pruning.best_estimator_)
print("The f1_macro score of the best model is: ", best_model_post_pruning.best_score_)
```

```
Fitting 3 folds for each of 1000 candidates, totalling 3000 fits
C:\Users\shekh\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:921: UserW
arning: One or more of the test scores are non-finite: [0.58256574 0.57788008 0.58848892
0.57817009 0.57741173 0.58268989
```

```
0.58162366 0.58533426 0.58534022 0.58074747 0.58659758 0.580185
0.57954001 0.57855621 0.58127843 0.58923393 0.58194063 0.57832891
0.58111342 0.58024958 0.57702293 0.57745515 0.58076528 0.58249667
0.58215779 0.57784013 0.57818715 0.58284421 0.57913256 0.58192726
0.57893527 0.58893325 0.5832972  0.58808776 0.57831933 0.58890985
0.5773538  0.57796076 0.57750414 0.58605085 0.58730282 0.57707967
0.5841608  0.57873668 0.58549838 0.57704411 0.58427672 0.57784323
0.57842383 0.58276643 0.57792794 0.57867681 0.58558801 0.58798693
0.58226183 0.58260192 0.57861878 0.57817755 0.58808448 0.57659913
0.57726775 0.58097368 0.58503157 0.58752596 0.57603986 0.58805097
0.60146666 0.57867039 0.58591171 0.58588702 0.58111253 0.58093308
0.57677521 0.58849381 0.58740693 0.58782679 0.58582313 0.58112154
0.58721269 0.58068172 0.58070544 0.5785055  0.58210407 0.58155568
0.57769629 0.57814    0.58480226 0.59362105 0.57826564 0.5798533
0.5845285  0.5763354  0.59150457 0.58273885 0.57630052 0.58184876
0.57747692 0.58243688 0.58410917 0.57645202 0.57838795 0.57908099
0.57809979 0.58025515 0.58212854 0.5891889  0.58277708 0.57722831
0.57719735 0.57870595 0.58475897 0.57964201 0.57914111 0.57867892
0.57683207 0.579169   0.58169653 0.57779397 0.57636728 0.58931292
0.58076853 0.58082728 0.58329128 0.61578691 0.58793615 0.57619983
0.58277071 0.58269423 0.58110376 0.58016385 0.57929242 0.58206485
0.58882014 0.58134569 0.57745879 0.58079344 0.57798605 0.59154041
0.5851075  0.57771221 0.58472702 0.58243473 0.57809046 0.58181029
0.57768404 0.57953607 0.58609361 0.58607358 0.57705188 0.57953963
0.57801364 0.57761112 0.58555164 0.58683615 0.57757207 0.58208767
0.57752563 0.57962986 0.58029188 0.58263709 0.57796882 0.57677085
0.58822741 0.58130988 0.58570316 0.58180537 0.59461224 0.58614017
0.57914188 0.5812159  0.58188934 0.58029014 0.58147879 0.58170692
0.57694539 0.57825546 0.57798481 0.58727158 0.58143873 0.58238259
0.5765704  0.58584265 0.58450074 0.58635623 0.58772174 0.57941599
0.57831181 0.58271628 0.5814669  0.58974866 0.57721357 0.58273964
0.58844566 0.5772353  0.58116694 0.5774429  0.57858577 0.58334892
0.5778594  0.57793022 0.59120673 0.57800937 0.58670044 0.58514939
0.58186893 0.60575428 0.58211579 0.57951434 0.57770352 0.58318801
0.58248218 0.57702675 0.58059436 0.57705524 0.59015763 0.58616908
0.58828406 0.58428993 0.57721185 0.57769856 0.5783627  0.57747282
0.59600547 0.58179814 0.57649663 0.57838763 0.57731134 0.58130996
0.57862921 0.58230785 0.57778608 0.58823155 0.58286851 0.57905082
0.58787145 0.58320577        nan 0.5817454  0.57824586 0.57762699
0.58285376 0.58076555 0.57848344 0.57737232 0.57779028 0.58251755
0.57873711 0.57662317 0.58844465 0.57863171 0.57954462 0.58568754
0.58107263 0.58857323 0.58149069 0.57875233 0.57847675 0.58181859
0.58543943 0.57698382 0.58575633 0.58206207 0.57853877 0.58950855
       nan 0.58301376 0.58128465 0.57948636 0.58051347 0.5815505
0.57870283 0.57843751 0.58195044 0.58334792 0.58083506 0.58660166
0.58272798 0.58165619 0.5852378  0.58033461 0.58124021 0.58043653
0.57690164 0.57904169 0.58679761 0.58086645 0.57875462 0.58261875
0.58268069 0.58879127 0.58575034 0.58181766 0.58286251 0.57664632
0.58795985 0.58198601 0.58820219 0.58570801 0.58667333 0.59147077
0.57847998 0.57808804 0.58018445 0.58727353 0.58188863 0.57715949
0.5801996  0.57772804 0.5867224  0.58259394 0.5783378  0.57833752
0.58174513 0.57734888 0.57672154 0.5858682  0.5819818  0.57867714
0.5817846  0.58526391        nan 0.58559247 0.57697525 0.58232143
0.57882277 0.60094828 0.58108649 0.57768722 0.58213911 0.57734297
0.57836592 0.5773791  0.58184944 0.57813584 0.58265761 0.58102266
0.58757325 0.5865969  0.58288962 0.57869957 0.58051738 0.58053004
0.57859643 0.57799367 0.58994337 0.57658856 0.58102813 0.57791628
0.58192648 0.58217595 0.58153775 0.5783076  0.59937395 0.57918307
0.57815894 0.57802392 0.57927195 0.5820946  0.58401764 0.58196935
0.58788928 0.58543057 0.57988165 0.58013503 0.58731982 0.58108778
0.57867197 0.57708231 0.58223059 0.60122093 0.58139276 0.57747935
0.58217518 0.57782554 0.57783442 0.58265333 0.58248682 0.57850878
0.58032634 0.58233891 0.58480118 0.58368687 0.58037824 0.57938783
0.58437389 0.58534097 0.58657312 0.58670598 0.57885467 0.59125159
0.58060252 0.58090134 0.58555623 0.59688358 0.57729163 0.57839297
```

0.58097047	0.57732722	0.57722874	0.5769323	0.5778239	0.5772252
0.58836722	0.59127159	0.57785235	0.58051195	0.57728774	0.60130244
0.57886549	0.57876238	0.57825875	0.58549146	0.57714877	0.5770994
0.57988331	0.58526448	0.57694057	0.58364051	0.58319853	0.58106392
0.57827258	0.58584315	0.58230819	0.58937155	0.57743412	0.58029823
0.58141205	0.57831281	0.5790943	0.57716238	0.57903913	0.57913464
0.57723303	0.58352284	0.57663828	0.57724206	0.58522167	0.58139417
0.58133062	0.57865569	0.57860393	0.5769862	0.57719549	0.57869084
0.58826313	0.57786362	0.58812916	0.58085225	0.57872489	0.57879656
0.57790874	0.58771436	0.5772382	0.59039061	0.58135695	0.58167154
0.58899206	0.57709882	nan	0.58857921	0.57694765	0.58195102
0.57703078	0.58061673	0.58142979	0.58277916	0.58144934	0.5781253
0.58290793	0.58757556	0.58265464	0.58573643	0.57828946	0.59008245
0.57906899	0.57806266	0.57797111	0.58025111	0.57823467	0.5814302
0.57785544	0.5775027	0.58818466	0.58169181	0.58121868	0.58960366
0.57716586	0.5776101	0.57696868	0.58234258	0.58671062	0.59088803
0.59689789	0.58150514	0.57814551	0.58219119	0.5796051	0.57820913
0.58110251	0.57793818	0.58617373	0.57806172	0.58476418	0.58626968
0.57842102	0.57740869	0.58099513	0.57781083	0.57831167	0.57600363
0.57725801	0.57958884	0.58301258	0.58007261	0.57735983	0.58780467
0.57789844	0.58562161	0.58506513	0.59432859	0.57746154	0.58105412
0.58903152	0.58442564	0.5774977	0.57722761	0.57704336	0.58692678
0.58173287	0.58277887	0.57633928	0.57648311	0.58537667	0.5806982
0.57805762	0.58323153	0.5823955	0.5863814	0.5825018	0.58190312
0.61967247	0.58026316	0.57919753	0.57998695	0.5780033	0.57714477
0.58839387	0.58408638	0.58750133	0.5818039	0.57740934	0.57811605
0.57844949	0.58126382	0.57785283	0.58848252	0.57756727	0.59092809
0.58093117	0.57760467	0.57780692	0.57765972	0.58262914	0.58081083
0.57873827	0.57701966	0.58848443	0.61513984	0.57665342	0.576616
0.5850558	0.57886354	0.58583145	0.58245543	0.58145839	0.58249085
0.57864825	0.58174435	0.57847981	0.5807001	0.58848934	0.5775223
0.57687046	0.58041207	0.5782328	0.5783308	0.57752994	0.58305745
0.57769949	0.58552268	0.58222749	0.57810452	0.58178313	0.58110354
0.61328307	0.58637797	0.58073002	0.58734619	0.58397649	0.60141567
0.57892033	0.57773666	0.57862672	0.5916839	0.57786303	0.58249769
0.57850784	0.58668145	0.57743214	0.57799952	0.58159777	0.58242141
0.57710359	0.58223407	0.5819758	0.58283539	0.57681465	0.58718621
nan	0.57768172	0.58226894	0.57813731	0.57804755	0.58586168
0.57777494	0.57757839	0.58259303	0.58574857	0.57821092	0.57846508
0.58102227	0.57762843	0.58774658	0.57670664	0.5815541	0.58201262
0.58868099	0.57889192	0.57899533	0.5826877	0.5847377	0.58715167
0.58040848	0.58746527	0.58256961	0.57773737	0.58352212	0.57750973
0.58175738	0.57693032	0.57828183	0.57802629	0.5811168	0.5813037
0.58580475	0.57944134	0.58560875	0.58875428	0.58561097	0.57856461
0.58706629	0.57990151	0.58144858	0.57842104	0.58214092	0.58158952
0.57785259	0.58488528	0.57900985	0.58389945	0.593864	0.57712165
0.5851336	0.582251	0.5779293	0.57649823	0.57830446	0.5819029
0.61547386	0.57828834	0.58210763	0.57827274	0.57956718	0.5776197
0.58223691	0.5824861	0.57703345	0.57926077	0.57655886	0.58525949
0.57909354	0.57633386	0.58223322	0.57842843	0.57811255	0.58519585
0.5814902	0.58123296	0.57934152	0.58239325	0.58301227	0.57771294
0.57797688	0.587437	0.58628333	0.58290025	0.58179728	0.57794895
0.58690151	0.58323868	0.58813476	0.59721352	0.58225558	0.58459016
0.57888956	0.5821865	0.58162391	0.58549894	0.58257172	nan
0.57663625	0.57732282	0.58278238	0.58875193	0.58013266	0.57948263
0.57905806	0.58189561	0.58677089	0.57749182	0.57705069	0.57715752
0.5787112	0.57806754	0.5787232	0.58842588	0.5779222	0.58559054
0.57809401	0.59022079	0.58111266	0.57785715	0.58116415	0.57787642
0.58845084	0.57802235	0.58227371	0.58267918	0.58207601	0.58308473
0.61904249	0.57814137	0.5768334	0.57705836	0.57628541	0.5896641
0.57802984	0.5811984	0.57719649	0.58308984	0.57719592	0.58097871
0.57922786	0.57675931	0.57878273	0.58549581	0.58412282	0.57648397
0.59973725	0.58312308	0.57920638	0.5772398	0.57836298	0.58907063
0.57799602	0.58262169	0.57650279	0.58535915	0.57751636	0.5907138
0.57931292	0.58259843	0.58135268	0.59077223	0.59204806	0.59114684

```
 0.57909466 0.58200517 0.57770837 0.57890635 0.57799496 0.5776954
 0.57769968 0.57711204 0.59005999 0.58213644 0.58090955 0.61262769
 0.58249083 0.58130917        nan 0.58221785 0.57773463 0.59031827
 0.57911642 0.58179725 0.57763474 0.57949809 0.57914033 0.61898095
        nan 0.58176415 0.5778062  0.58738339 0.57721212 0.57612508
 0.58303708 0.57840629 0.5777687  0.57737429 0.58644364 0.58373057
 0.6064865  0.5850494  0.58821415 0.58175065 0.57797305 0.59247331
 0.57721841 0.58861805 0.57831046 0.58423548 0.57792749 0.58246528
 0.57748383 0.57877552 0.57600407 0.57641624 0.58787456 0.57848263
 0.58219888 0.58142558 0.57764075 0.57863129 0.57614784 0.58305857
 0.58132152 0.58443072 0.5812782  0.58427072 0.57739265 0.58158867
 0.57879811 0.57784389 0.57853365 0.58015222 0.58056605 0.57873275
 0.5825347  0.57674382 0.57686024 0.57835777 0.58777743 0.58218209
 0.58273177 0.59559262 0.57915215 0.59082448 0.57852733 0.58603982
 0.58809978 0.59270921 0.581131   0.57785959 0.57786202 0.57858522
 0.57784036 0.58130389 0.58295568        nan 0.58691993 0.58817583
 0.57830353 0.58177828 0.57654796 0.58035312 0.57706732 0.58210889
 0.57645972 0.57928835 0.58093988 0.58509687 0.57733576 0.57736597
 0.58070399 0.58192512 0.57599543 0.58240091 0.5773825  0.57834439
 0.57762948 0.58247527 0.57921357 0.58265852 0.58074663 0.57802176
 0.58597633 0.58847363 0.57763141 0.57781423 0.58284787 0.57732361
 0.58175104 0.57873165 0.57918412 0.58519188 0.58013627 0.58387256
 0.57765597 0.58561034 0.58218675 0.57866079 0.57693103 0.57821549
 0.58205782 0.58186301 0.58187968        nan 0.5864614  0.58719207
 0.57673262 0.58155527 0.57791633 0.57863139 0.58391926 0.59249674
 0.57896387 0.57703852 0.5828303  0.5773628  0.58220757 0.57721082
 0.58261578 0.57742838 0.57903055 0.57849897 0.57706884 0.58133123
 0.58472429 0.57791886 0.58635018 0.57816162 0.57875125 0.57774628
 0.60127133 0.58594059 0.58693243 0.58054301 0.5797746  0.58313282
 0.58389383 0.57898405 0.57824384 0.57860493 0.57832742 0.58134819
 0.57805924 0.58439028 0.5833697  0.57807473 0.57707956 0.57971887
 0.58292046 0.57785087 0.57640608 0.57707021 0.59047763 0.57909086
 0.57784731 0.57697109 0.58502739 0.58778289 0.58532804 0.57779709
 0.58725501 0.58623659 0.57758177 0.62080336 0.58098666 0.57878925
 0.58551922        nan 0.57951579 0.57816323 0.58140689 0.58148692
 0.57881819 0.57783278 0.58661349 0.58275474]
  category=UserWarning
Pipeline(steps=[('ohe',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(handle_unknown='ignor
e'),
                                                  ['Gender', 'Race',
                                                   'TypeOfAdmission',
                                                   'CCSProcedureCode',
                                                   'PaymentTypology',
                                                   'EmergencyDepartmentIndicator'])])),
                ('model',
                 DecisionTreeClassifier(ccp_alpha=0.00021375669548743398,
                                        class_weight='balanced'))])
The f1_macro score of the best model is:  0.6208033600811741
```

In [43]:

```python
y_pred_DT_ccp = best_model_post_pruning.predict(test_X)
print(classification_report(test_y, y_pred_DT_ccp))
```

```
              precision    recall  f1-score   support

           0       0.93      0.67      0.78      9943
           1       0.32      0.75      0.45      2051

    accuracy                           0.68     11994
   macro avg       0.62      0.71      0.61     11994
weighted avg       0.83      0.68      0.72     11994
```

Observation

- The model tuned using CCP alpha is also very similar in terms of performance with a macro average of 61%.
- This model is also not overfitted.

In [26]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

data_tree_based_models = data

column_trans_tree_models = make_column_transformer(
    (OneHotEncoder(handle_unknown='ignore'), cat_col),
    remainder = 'passthrough'
)

with pd.option_context('mode.chained_assignment', None):
    train_data, test_data = train_test_split(data_tree_based_models, test_size=0.2,
                                          shuffle=True,random_state=0)

print("Number of training examples: ", train_data.shape[0])
print("Number of testing examples: ", test_data.shape[0])

train_X = train_data.drop('LengthOfStay', axis='columns')
train_y = train_data[['LengthOfStay']].to_numpy()

test_X = test_data.drop('LengthOfStay', axis='columns')
test_y = test_data[['LengthOfStay']].to_numpy()

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 400, stop = 1200, num = 10)]
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 110, num = 11)]
max_depth.append(None)

RF_param_grid = [{'model__max_depth': max_depth,
                  'model__n_estimators': n_estimators
                 }]

RF_clf = RandomForestClassifier(criterion = 'entropy', class_weight='balanced_subsample

RF_pipeline = Pipeline( [('ohe',column_trans_tree_models),
                         ('model', RF_clf)
                        ]
                      )

RF_grid_search = GridSearchCV(estimator=RF_pipeline, param_grid=RF_param_grid,
                              scoring='f1_macro', n_jobs=-1, verbose = 10,cv = 3)

RF_grid_search_model = RF_grid_search.fit(train_X, train_y)

print(RF_grid_search_model.best_estimator_)
print("The f1_macro score of the best model is: ", RF_grid_search_model.best_score_)
```

```
Number of training examples:  47972
Number of testing examples:  11994
Fitting 3 folds for each of 120 candidates, totalling 360 fits
C:\Users\shekh\Anaconda3\lib\site-packages\sklearn\pipeline.py:346: DataConversionWarnin
g: A column-vector y was passed when a 1d array was expected. Please change the shape of
y to (n_samples,), for example using ravel().
  self._final_estimator.fit(Xt, y, **fit_params_last_step)
Pipeline(steps=[('ohe',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(handle_unknown='ignor
e'),
                                                  ['Gender', 'Race',
                                                   'TypeOfAdmission',
                                                   'CCSProcedureCode',
                                                   'PaymentTypology',
                                                   'EmergencyDepartmentIndicator'])])),
                ('model',
                 RandomForestClassifier(class_weight='balanced_subsample',
                                        criterion='entropy', max_depth=15,
                                        n_estimators=1022, n_jobs=-1))])
The f1_macro score of the best model is:  0.6480369931462646
```

In [44]:

```
y_pred_RF = RF_grid_search_model.predict(test_X)
print(classification_report(test_y, y_pred_RF))
```

```
              precision    recall  f1-score   support

           0       0.90      0.80      0.85      9943
           1       0.38      0.58      0.45      2051

    accuracy                           0.76     11994
   macro avg       0.64      0.69      0.65     11994
weighted avg       0.81      0.76      0.78     11994
```

In [67]:

```
## Predicting the test file
data_prediction = pd.read_csv('test_data.csv', index_col = 'ID', usecols = ['ID', 'Gend
                                                                            'APRSeverityOfIllness
                                                                           'EmergencyDepartmentIn
                                                                            'AverageChargesInCoun
                                                                            'AverageChargesInFaci

for column_name in cat_col:
    data_prediction[column_name] = data_prediction[column_name].astype('category')

test_predictions = RF_grid_search_model.predict(data_prediction)
prediction_dataframe = pd.DataFrame(test_predictions)
prediction_dataframe.index = prediction_dataframe.index+1
prediction_dataframe.rename(columns = {0:'LengthOfStay'}, inplace = True)
prediction_dataframe.to_csv('s3831855_predictions.csv')
```

Conclusion

- Using the techniques taught in labs and lectures I have been able to get a best model using Random Forest classifier.
- The performance of best model is around 65% macro f1_score

- The model does a great job at predicting when a patient would be admitted for acute care buut performs poorly for the cases where patients will require longer care than 3 days.
- This could be because of class imbalance, but measures were taken while developing the model to get similar performances for both classes.
- For the class imbalance, maybe oversampling or undersampling would have helped.
- The final model is performing similarly on the test set as well, so I can say that it has fitted well.
- The predictions for the test data have been uploaded in *s3831855_predictions.csv*