Title: **Development and Validation of a Prediction Model for Identifying Critical Patients**

Problem Statement: The characteristics of critically ill patients upon admission remain poorly understood. We aimed to bridge this gap by developing and validating a prediction model for identifying patients at risk of critical conditions during their hospital stay.

Data: The dataset comprises various factors related to patients during hospitalization, encompassing medical history (such as Aids, cirrhosis, hypertension, diabetes mellitus etc.), vital signs, and other pertinent parameters. The primary objective is to leverage this comprehensive dataset to predict whether a patient is likely to become critically ill during their in-hospital stay.

Findings: Through rigorous analysis and modeling, we successfully developed a predictive model for identifying critical patients. The model utilizes the diverse factors present in the dataset to generate accurate predictions. The validation process demonstrated the robustness and generalizability of the model across different patient populations.

Implications: The developed prediction model has significant implications for clinical practice, enabling healthcare providers to proactively identify patients at risk of critical conditions early on. This advancement can lead to timely interventions, potentially improving patient outcomes and optimizing resource allocation within healthcare settings. Future research may further refine the model and explore additional factors contributing to critical conditions in hospitalized patients.

Problem Solving Methodology Overview:

- **Understanding the Problem:**
  - Developing and validating a prediction model for all-cause in-hospital mortality and identifying critical patients.
  - Create a predictive model using a dataset of various patient-related factors such as BMI, elective surgery, ICU admit source, ICU type, Apache scores, heart rate, respiratory rate, glucose level, diastolic bp etc.

- **Exploratory Data Analysis (EDA):**
  - Examine the dataset: Understand the structure, types of variables, and their distributions.

SHAPE OF THE DATA

```
In [4]: # checking shape of the data
        df.shape

Out[4]: (91713, 85)
```

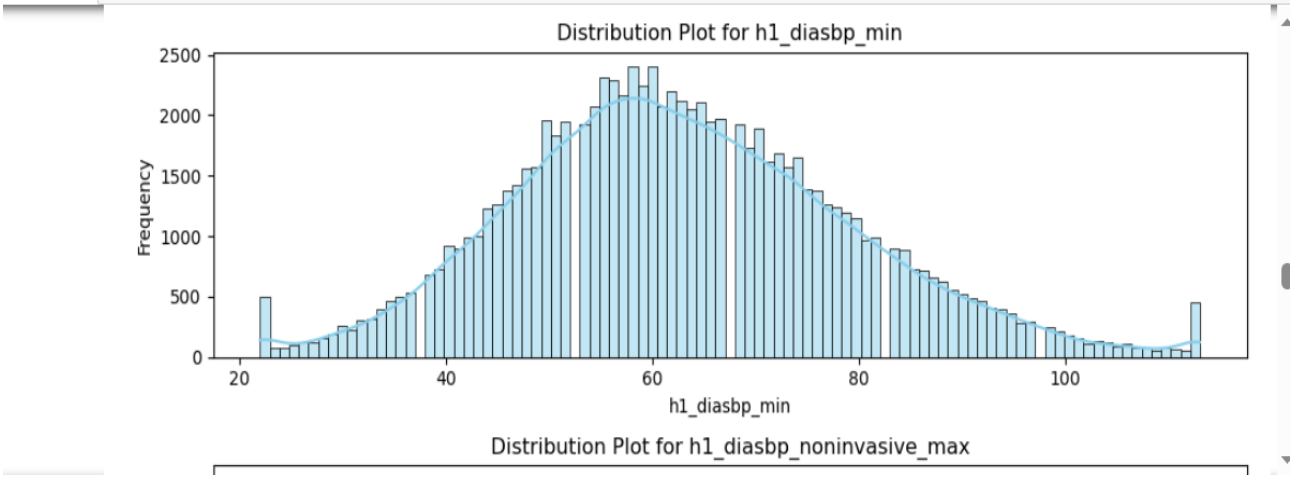
DATA TYPE

```
dtypes: float64(71), int64(7), object(7)
memory usage: 59.5+ MB
```

DISTRIBUTION OF THE DATA

VISUALIZE OF DISTRIBUTION OF THE DATA

In [17]:
```
# Set up a grid of subplots for each column
fig, axes = plt.subplots(nrows=len(df.columns), ncols=1, figsize=(10, 3*len(df.columns)))

# Iterate through each column and create a distribution plot
for i, column in enumerate(df.columns):
    sns.histplot(df[column], kde=True, color='skyblue', ax=axes[i])
    axes[i].set_title(f'Distribution Plot for {column}')
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plots
plt.show()
```



We have dropped *redundant* columns 'encounter_id','patient_id','Unnamed: 83','hospital_id','icu_id'.

a. Unnamed columns have all the nan values
b. After analysis drop all id columns as they have unique values or high variations.

Identify missing values: Handle missing data appropriately.

	total	percentage
hospital_id	0	0.000000
apache_post_operative	0	0.000000
pre_icu_los_days	0	0.000000
icu_type	0	0.000000
icu_stay_type	0	0.000000
icu_id	0	0.000000
hospital_death	0	0.000000
elective_surgery	0	0.000000
gender	25	0.000273
icu_admit_source	112	0.001221
d1_heartrate_min	145	0.001581
d1_heartrate_max	145	0.001581
d1_sysbp_min	159	0.001734
d1_sysbp_max	159	0.001734
d1_diasbp_min	165	0.001799
d1_diasbp_max	165	0.001799
d1_mbp_min	220	0.002399
d1_mbp_max	220	0.002399
d1_spo2_max	333	0.003631
d1_spo2_min	333	0.003631
d1_resprate_max	385	0.004198
d1_resprate_min	385	0.004198
aids	715	0.007796
immunosuppression	715	0.007796
leukemia	715	0.007796
arf_apache	715	0.007796
hepatic_failure	715	0.007796
lymphoma	715	0.007796
ventilated_apache	715	0.007796
diabetes_mellitus	715	0.007796
cirrhosis	715	0.007796
intubated_apache	715	0.007796
solid_tumor_with_metastasis	715	0.007796
heart_rate_apache	878	0.009573
map_apache	994	0.010838
d1_sysbp_noninvasive_max	1027	0.011198
d1_sysbp_noninvasive_min	1027	0.011198
gcs_unable_apache	1037	0.011307

gcs_unable_apache	1037	0.011307
d1_diasbp_noninvasive_min	1040	0.011340
d1_diasbp_noninvasive_max	1040	0.011340
apache_3j_diagnosis	1101	0.012005
resprate_apache	1234	0.013455
height	1334	0.014545
ethnicity	1395	0.015210
d1_mbp_noninvasive_max	1479	0.016126
d1_mbp_noninvasive_min	1479	0.016126
apache_2_bodysystem	1662	0.018122
apache_3j_bodysystem	1662	0.018122
apache_2_diagnosis	1662	0.018122
gcs_verbal_apache	1901	0.020728

NEGITIVE VALUES TREATMENT

checking negitive values

In [11]:
```
df_num = df.select_dtypes(include = np.number)
df_num.head()
```

Out[11]:

	hospital_id	age	bmi	height	icu_id	pre_icu_los_days	weight	apache_2_diagnosis	apache_3j_diagnosis	gcs_eyes_apache	gcs_motor_a
0	118	68.000000	22.730000	180.300000	92	0.541667	73.900000	113.000000	502.010000	3.000000	6.(
1	81	77.000000	27.420000	160.000000	90	0.927778	70.200000	108.000000	203.010000	1.000000	3.(
2	118	25.000000	31.950000	172.700000	93	0.000694	95.300000	122.000000	703.030000	3.000000	6.(
3	118	81.000000	22.640000	165.100000	92	0.000694	61.700000	203.000000	1206.030000	4.000000	6.(
4	33	19.000000	NaN	188.000000	91	0.073611	NaN	119.000000	601.010000	NaN	

In [12]:
```
for col in df_num.columns:
    df_num.loc[df_num[col] < 0, col] = np.nan
```

Treating missing values

Missing values treatment with KNN imputer

```
In [13]: from sklearn.impute import KNNImputer

In [14]: X=np.array(df_num)

In [15]: imputer = KNNImputer(n_neighbors=5)
         imputed_dataset = pd.DataFrame(imputer.fit_transform(X),columns=df_num.columns)

In [16]: imputed_dataset.isnull().sum()
```

```
Out[16]: hospital_id                  0
         age                          0
         bmi                          0
         height                       0
         icu_id                       0
         pre_icu_los_days             0
         weight                       0
         apache_2_diagnosis           0
         apache_3j_diagnosis          0
         gcs_eyes_apache              0
         gcs_motor_apache             0
         gcs_verbal_apache            0
         heart_rate_apache            0
         map_apache                   0
         resprate_apache              0
         temp_apache                  0
         d1_diasbp_max                0
         d1_diasbp_min                0
         d1_diasbp_noninvasive_max    0
         d1_diasbp_noninvasive_min    0
```

Treating missing values of categorical variable

Treating missing values with mode imputation

```
20]: for i in df_cat:
         df_cat[i].fillna(df_cat[i].mode()[0],inplace=True,axis=0)
     df_cat.isnull().sum()
```

```
20]: elective_surgery              0
     ethnicity                     0
     gender                        0
     icu_admit_source              0
     icu_stay_type                 0
     icu_type                      0
     apache_post_operative         0
     arf_apache                    0
     gcs_unable_apache             0
     intubated_apache              0
     ventilated_apache             0
     aids                          0
     cirrhosis                     0
     diabetes_mellitus             0
     hepatic_failure               0
     immunosuppression             0
     leukemia                      0
     lymphoma                      0
     solid_tumor_with_metastasis   0
     apache_3j_bodysystem          0
     apache_2_bodysystem           0
     dtype: int64
```

1. There are high number of missing values in 50% of the numeric and categorical columns – imputed using KNN imputer and mode imputation respectively.
2. Pre_icu_los_days, apache_3j_prob, apache_2j_prob have negative values which is converted into nan and then imputed with KNN imputer

BINNING OF CATEGORICAL COLUMNS

Bining categorical values

In [186]:
```
for i in df_cat.columns:
    print(i)
    print((df_cat[i].value_counts()/len(df_cat[i])*100))
    print('_'*75)
```

```
ethnicity
Caucasian          78.591912
African American   10.409647
Other/Unknown       4.769226
Hispanic            4.138999
Asian               1.231014
Native American     0.859202
Name: ethnicity, dtype: float64
___________________________________________________________________________
gender
M   53.966177
F   46.033823
Name: gender, dtype: float64
___________________________________________________________________________
icu_admit_source
Accident & Emergency        59.066872
Operating Room / Recovery   20.403869
Floor                       17.021578
Other Hospital               2.571064
Other ICU                    0.936617
Name: icu_admit_source, dtype: float64
___________________________________________________________________________
icu_stay_type
admit      93.970320
transfer    5.419079
readmit     0.610600
```

In [187]:
```
df_cat['ethnicity']=df_cat['ethnicity'].replace(['Other/Unknown','Hispanic','Asian','Native American'],'Other')
df_cat['icu_admit_source']=df_cat['icu_admit_source'].replace(['Other Hospital','Other ICU'],'Other')
df_cat['apache_3j_bodysystem']=df_cat['apache_3j_bodysystem'].replace(['Genitourinary','Musculoskeletal/Skin','Hematological',
df_cat['apache_2_bodysystem']=df_cat['apache_2_bodysystem'].replace(['Undefined diagnoses'],'Undefined Diagnoses')
df_cat['apache_2_bodysystem']=df_cat['apache_2_bodysystem'].replace(['Renal/Genitourinary','Haematologic'],'Other')
```

3. We perform binning of categorical columns and dummy encode categorical columns

DUMMY ENCODING CATEGORICAL COLUMNS

Dummies

```
In [514]: df_cat_dum=pd.get_dummies(df_cat,drop_first=True)
          df_cat_dum.shape

Out[514]: (91713, 28)
```

```
In [515]: df_cat_dum.columns

Out[515]: Index(['ethnicity_Caucasian', 'ethnicity_Other', 'gender_M',
                 'icu_admit_source_Floor', 'icu_admit_source_Operating Room / Recovery',
                 'icu_admit_source_Other', 'icu_stay_type_readmit',
                 'icu_stay_type_transfer', 'icu_type_CSICU', 'icu_type_CTICU',
                 'icu_type_Cardiac ICU', 'icu_type_MICU', 'icu_type_Med-Surg ICU',
                 'icu_type_Neuro ICU', 'icu_type_SICU',
                 'apache_3j_bodysystem_Gastrointestinal',
                 'apache_3j_bodysystem_Metabolic',
                 'apache_3j_bodysystem_Musculoskeletal/Skin',
                 'apache_3j_bodysystem_Neurological', 'apache_3j_bodysystem_Other',
                 'apache_3j_bodysystem_Respiratory', 'apache_3j_bodysystem_Sepsis',
                 'apache_2_bodysystem_Gastrointestinal',
                 'apache_2_bodysystem_Haematologic', 'apache_2_bodysystem_Metabolic',
                 'apache_2_bodysystem_Neurologic', 'apache_2_bodysystem_Other',
                 'apache_2_bodysystem_Respiratory'],
                dtype='object')
```

- *Visualize relationships: Explore correlations and patterns between variables to gain insights*

summarize data

```
In [26]: df.describe()
```

Out[26]:

	hospital_id	age	bmi	height	icu_id	pre_icu_los_days	weight	apache_2_diagnosis	apache_3j_diagnosis	gcs_e
count	91713.000000	91713.000000	91713.000000	91713.000000	91713.000000	91713.000000	91713.000000	91713.000000	91713.000000	9
mean	105.669262	62.447459	29.183511	169.652290	508.357692	0.848106	84.054921	185.335117	557.912708	
std	62.854406	16.511961	8.155393	10.734403	228.989661	2.482534	24.727732	85.491466	461.301760	
min	2.000000	16.000000	14.844926	137.200000	82.000000	0.000000	38.600000	101.000000	0.010000	
25%	47.000000	53.000000	23.749765	162.560000	369.000000	0.038194	67.100000	113.000000	203.010000	
50%	109.000000	65.000000	27.739646	170.100000	504.000000	0.142361	80.600000	123.000000	409.020000	
75%	161.000000	75.000000	32.846606	177.800000	679.000000	0.419444	97.000000	301.000000	703.030000	
max	204.000000	89.000000	67.814990	195.590000	927.000000	159.090972	186.000000	308.000000	2201.050000	

Skewness

In [27]:
```
import scipy.stats as stats

df.skew()
```

Out[27]:

hospital_id	-0.045683
age	-0.646352
bmi	1.452578
height	-0.104140
icu_id	-0.163940
pre_icu_los_days	11.065736
weight	1.072545
apache_2_diagnosis	0.510641
apache_3j_diagnosis	1.016895
gcs_eyes_apache	-1.694964
gcs_motor_apache	-2.727640
gcs_verbal_apache	-1.211060
heart_rate_apache	-0.265083
map_apache	0.699270
resprate_apache	0.261030
temp_apache	-0.985721
d1_diasbp_max	0.813837
d1_diasbp_min	0.093335
d1_diasbp_noninvasive_max	0.816897
d1_diasbp_noninvasive_min	0.085641
d1_heartrate_max	0.574217
d1_heartrate_min	-0.113440
d1_mbp_max	0.801198
d1_mbp_min	0.207866
d1_mbp_noninvasive_max	0.757384
d1_mbp_noninvasive_min	0.195747
d1_resprate_max	2.498739
d1_resprate_min	0.164902
d1_spo2_max	-19.292316
d1_spo2_min	-4.774896
d1_sysbp_max	0.506485
d1_sysbp_min	0.221530
d1_sysbp_noninvasive_max	0.509491
d1_sysbp_noninvasive_min	0.220952

Skewness treatment

In [200]:
```
from scipy.stats import yeojohnson
import numpy as np

# Example data
#data = np.array([1, 2, 3, 4, 5])
trans_yj=pd.DataFrame()

# Apply Yeo-Johnson transformation
for i in imputed_dataset.columns:
    transformed_data, lambda_value = yeojohnson(np.array(imputed_dataset[i]))
    trans_yj[i]=pd.Series(transformed_data)
```

In [201]:
```
trans_yj.skew()
```

Out[201]:

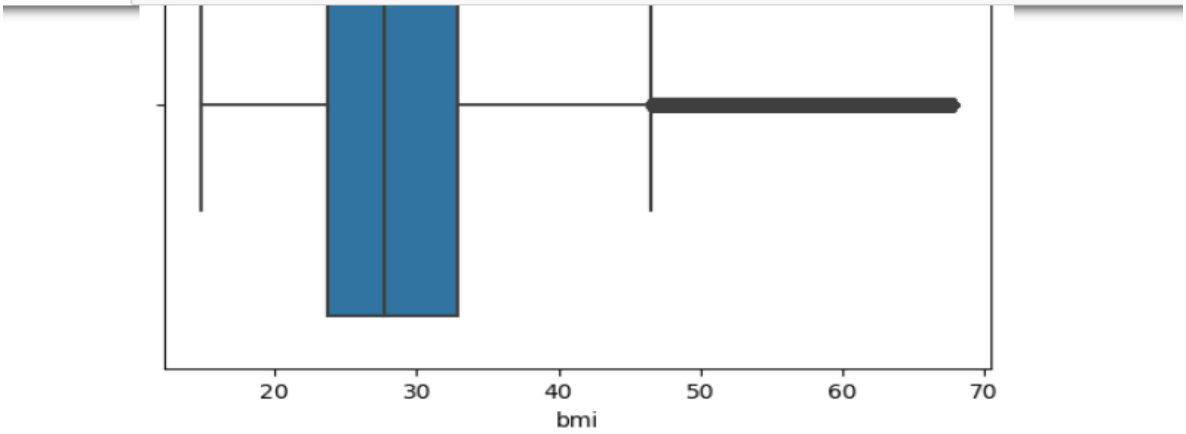
age	-0.159881
bmi	-0.008519
elective_surgery	1.633330
height	-0.013564
pre_icu_los_days	0.730652
weight	-0.000900
apache_2_diagnosis	0.279768
apache_3j_diagnosis	-0.022038
apache_post_operative	1.491413
arf_apache	5.749089
gcs_eyes_apache	-0.998770
gcs_motor_apache	-1.547744
gcs_unable_apache	10.073217
gcs_verbal_apache	-0.897583
heart_rate_apache	-0.101953
intubated_apache	1.959820

The data was highly skewed for some of the columns which is treated using yeo johnson method to get in range between –1 to 1

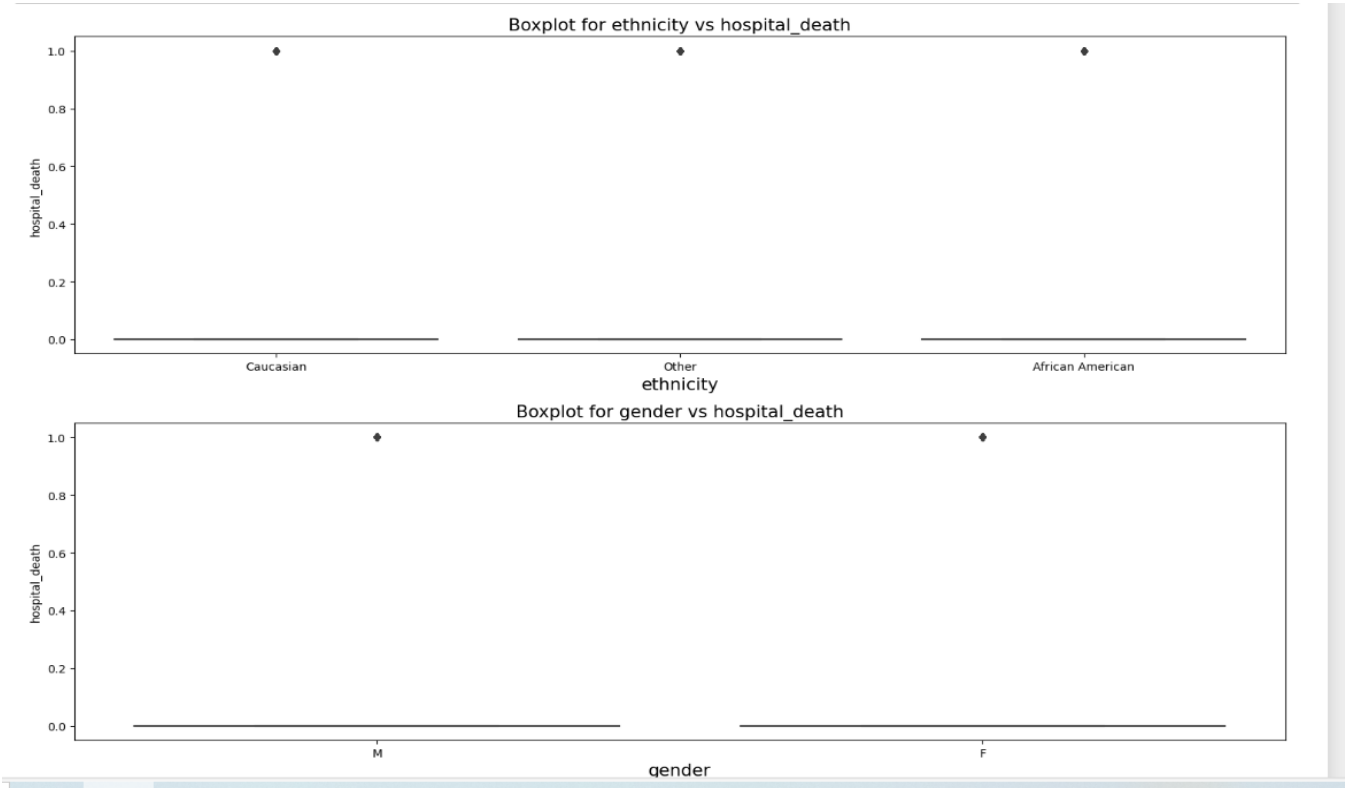
Checking Outliers

In [31]:
```
df_num = df.select_dtypes(include = np.number)
```

In [32]:
```
for i in df_num.columns:
    print(i)
    sns.boxplot(df_num[i])
    plt.show()
    print('_'*75)
```



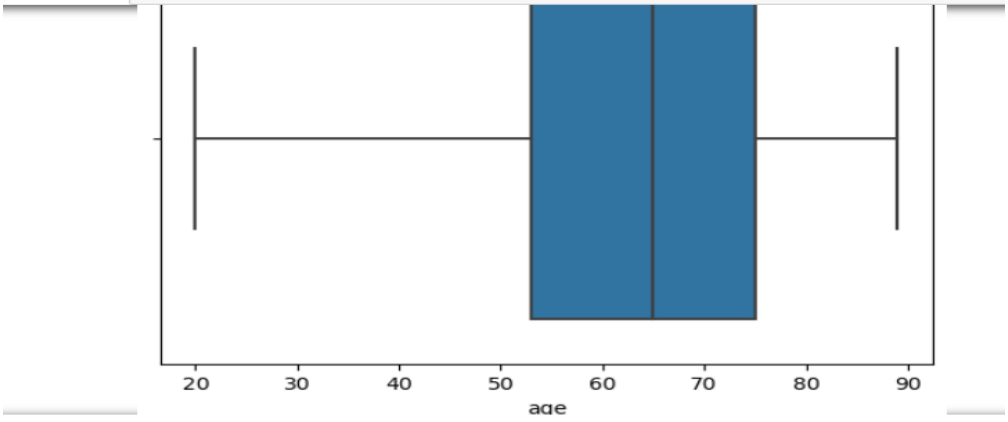
CHECKING OUTLIERS OF CATEGORICAL VALUES



OUTLIERS TREATMENT USING CAPING

In [33]:
```
# Outlier treatment
for col in df_num.columns:
    percentiles = df_num[col].quantile([0.01, 0.99]).values
    df_num[col][df[col] <= percentiles[0]] = percentiles[0]
    df_num[col][df[col] >= percentiles[1]] = percentiles[1]
```

In [34]:
```
for i in df_num.columns:
    print(i)
    sns.boxplot(df_num[i])
    plt.show()
    print('_'*75)
```



4. For outliers we used capping of percentile 0.01 and 0.99.

correlation between numeric columns

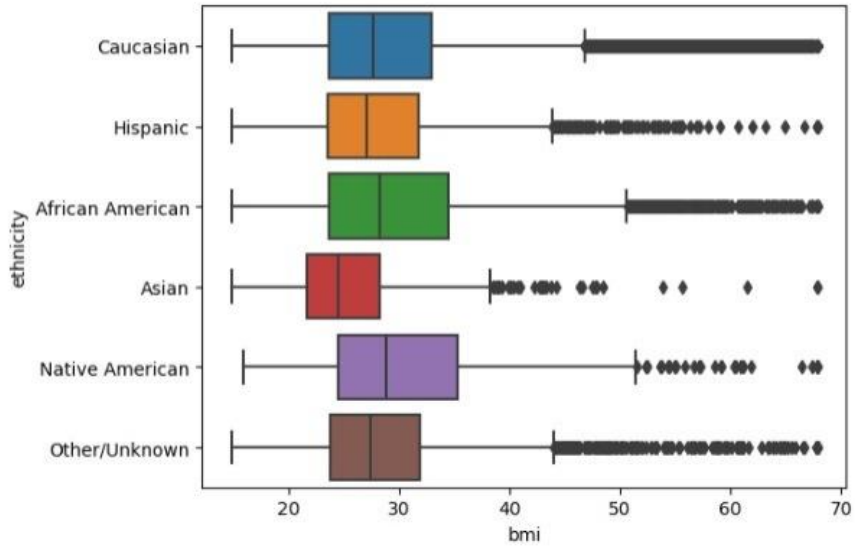
In [37]:
```
# Select numerical columns
df_num = df.select_dtypes(include='number')

# Calculate the correlation matrix
df_cor = df_num.corr()

# Display the correlation matrix
df_cor
```

Out[37]:

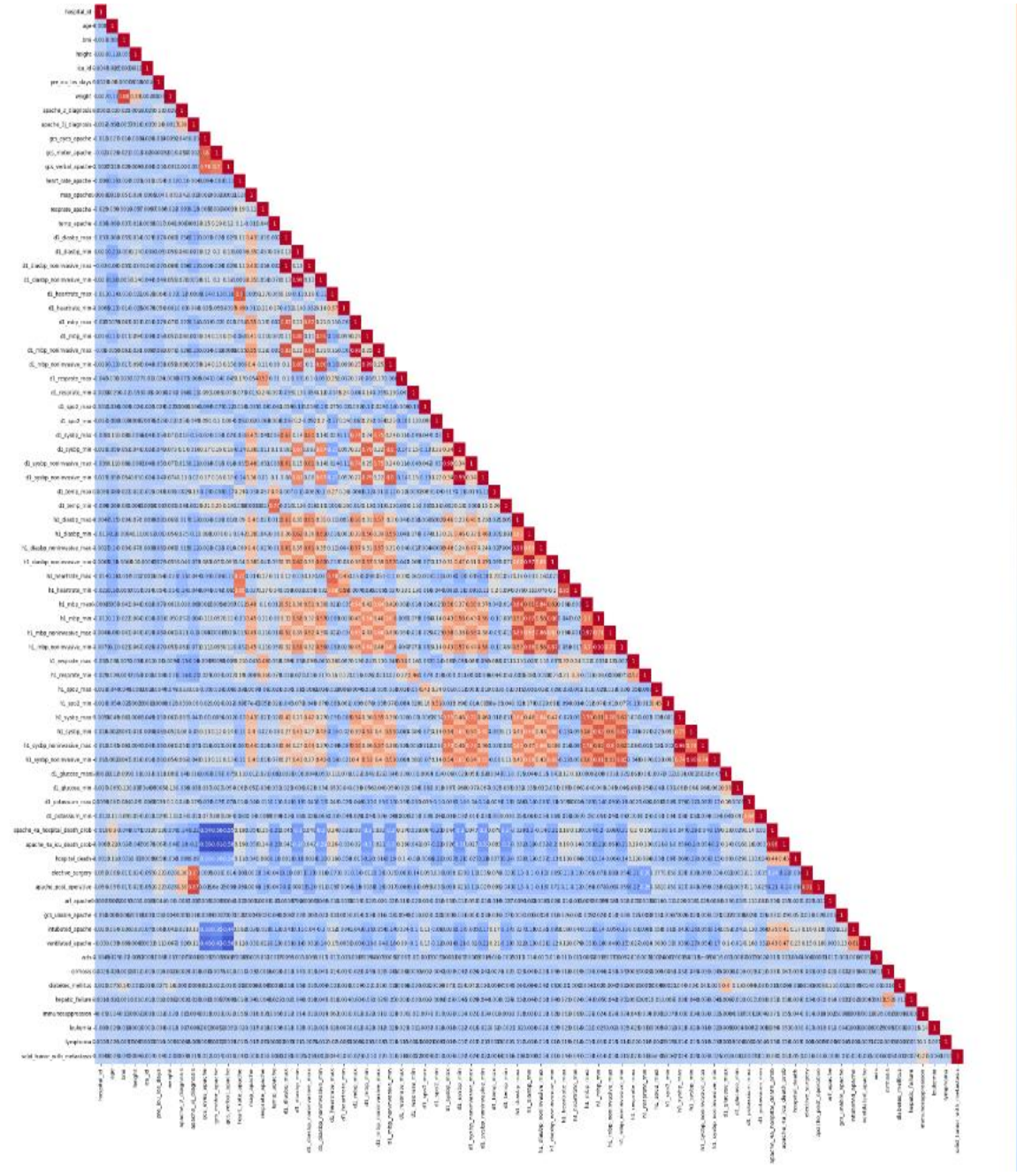
	hospital_id	age	bmi	height	icu_id	pre_icu_los_days	weight	apache_2_diagnosis	apache_3j_diagnosis
hospital_id	1.000000	-0.008667	0.013480	0.028002	0.004526	0.002792	0.027173	0.006149	0.031711
age	-0.008667	1.000000	-0.088312	-0.112863	-0.024845	0.079897	-0.129777	0.019969	-0.057756
bmi	0.013480	-0.088312	1.000000	-0.054852	0.000344	-0.000178	0.875827	0.027462	-0.005711
height	0.028002	-0.112863	-0.054852	1.000000	0.018821	-0.017889	0.387708	0.001304	0.015509
icu_id	0.004526	-0.024845	0.000344	0.018821	1.000000	0.004629	0.007408	-0.029286	-0.034554
pre_icu_los_days	0.002792	0.079897	-0.000178	-0.017889	0.004629	1.000000	-0.007562	0.132764	0.162883
weight	0.027173	-0.129777	0.875827	0.387708	0.007408	-0.007562	1.000000	0.027608	0.001651
apache_2_diagnosis	0.006149	0.019969	0.027462	0.001304	-0.029286	0.132764	0.027608	1.000000	0.393270
apache_3j_diagnosis	0.031711	-0.057756	-0.005711	0.015509	-0.034554	0.162883	0.001651	0.393270	1.000000
gcs_eyes_apache	-0.011533	0.026744	0.013595	-0.008366	-0.025774	-0.013708	0.009230	0.046173	-0.030058
gcs_motor_apache	-0.020175	0.025982	0.021298	-0.013349	-0.023601	-0.000520	0.014042	0.057281	0.000233

ETHINICITY VS BMI

```
]: import seaborn as sns
   sns.boxplot(y=df['ethnicity'],x=df['bmi'])
   plt.show()
```



HEATMAP



Chi sqaure test

In [46]:
```
import scipy.stats
import pandas as pd

contingency_table = pd.crosstab(df['age'], df['hospital_death'].astype(object))
chi2_stat, p_value, dof, expected = scipy.stats.chi2_contingency(contingency_table)

alpha = 0.05

print(f"Chi-Square Statistic: {chi2_stat}")
print(f"P-value: {p_value}")
print(f"Degrees of Freedom: {dof}")

if p_value < alpha:
    print("Reject the null hypothesis. There is a significant association between age and hospital death.")
else:
    print("Fail to reject the null hypothesis. There is no significant association between age and hospital death.")
```

```
Chi-Square Statistic: 1625.5767027099564
P-value: 5.511775869881623e-187
Degrees of Freedom: 283
Reject the null hypothesis. There is a significant association between age and hospital death.
```

In [47]:
```
from scipy.stats import chi2_contingency
alpha = 0.05
contingency_table = pd.crosstab(df['ethnicity'], df['hospital_death'])
chi2, p, _, _ = chi2_contingency(contingency_table)

print(f"Chi-square statistic: {chi2}")
print(f"P-value: {p}")
if p> alpha:
    print(" There is no significant association between 'ethnicity' and 'hospital_death'")
else:
    print( "There is significant association between 'ethnicity' and 'hospital_death'")
```

```
Chi-square statistic: 8.322732982285277
P-value: 0.015586244895672908
There is significant association between 'ethnicity' and 'hospital_death'
```

5. We perform stats test between age, ethnicity with target columns which is hospital death and found that age have significant association while ethnicity did not have significant association with the target variable.

VIF :

```
: from statsmodels.stats.outliers_influence import variance_inflation_factor

def calculate_vif_(X, thresh=5):
    X = X.assign(const=1)
    variables = list(range(X.shape[1]))
    dropped = True
    while dropped:
        dropped = False
        vif = [variance_inflation_factor(X.iloc[:, variables].values, ix)
               for ix in range(X.iloc[:, variables].shape[1])]
        vif = vif[:-1]  # don't let the constant be removed in the loop.
        maxloc = vif.index(max(vif))
        if max(vif) > thresh:
            print('dropping \'' + X.iloc[:, variables].columns[maxloc] + '\' at index: ' + str(maxloc))
            del variables[maxloc]
            dropped = True
    print('Remaining variables:')
    print(variables)
    print(X.columns[variables[:-1]])
    return X.iloc[:, variables[:-1]]
```

```
dropping 'd1_diasbp_noninvasive_min' at index: 17
dropping 'd1_sysbp_noninvasive_min' at index: 30
dropping 'd1_sysbp_noninvasive_max' at index: 29
dropping 'd1_diasbp_noninvasive_max' at index: 16
dropping 'd1_mbp_min' at index: 19
dropping 'h1_mbp_noninvasive_min' at index: 38
dropping 'h1_sysbp_noninvasive_max' at index: 44
dropping 'weight' at index: 4
dropping 'd1_mbp_max' at index: 17
dropping 'h1_diasbp_min' at index: 28
dropping 'h1_sysbp_min' at index: 40
dropping 'h1_mbp_max' at index: 32
dropping 'h1_diasbp_noninvasive_max' at index: 28
dropping 'apache_4a_hospital_death_prob' at index: 43
dropping 'h1_heartrate_max' at index: 29
dropping 'h1_mbp_min' at index: 30
dropping 'h1_mbp_noninvasive_max' at index: 30
dropping 'd1_mbp_noninvasive_min' at index: 18
dropping 'h1_sysbp_noninvasive_min' at index: 34
```

```
52, 53, 54, 55, 57, 58]
Index(['age', 'bmi', 'height', 'pre_icu_los_days', 'apache_2_diagnosis',
       'apache_3j_diagnosis', 'gcs_eyes_apache', 'gcs_motor_apache',
       'gcs_verbal_apache', 'heart_rate_apache', 'map_apache',
       'resprate_apache', 'temp_apache', 'd1_diasbp_max', 'd1_diasbp_min',
       'd1_heartrate_max', 'd1_heartrate_min', 'd1_mbp_noninvasive_max',
       'd1_resprate_max', 'd1_resprate_min', 'd1_spo2_max', 'd1_spo2_min',
       'd1_sysbp_max', 'd1_sysbp_min', 'd1_temp_max', 'd1_temp_min',
       'h1_diasbp_max', 'h1_diasbp_noninvasive_min', 'h1_heartrate_min',
       'h1_resprate_max', 'h1_resprate_min', 'h1_spo2_max', 'h1_spo2_min',
       'h1_sysbp_max', 'd1_glucose_max', 'd1_glucose_min', 'd1_potassium_max',
       'd1_potassium_min', 'apache_4a_icu_death_prob'],
      dtype='object')
```

6. While performing Vif with threshold as 5. Dropped 20 columns which are less significant for model building.

SCALING

Scaling

```
In [182]: from sklearn.preprocessing import StandardScaler
          scaler=StandardScaler()
          data_no=pd.DataFrame(scaler.fit_transform(data_no),columns=data_no.columns)
```

MODEL BUILDING

Balancing

```
: (y_train.value_counts()/len(y_train))*100

: 0    91.224162
  1     8.775838
  Name: hospital_death, dtype: float64

: from imblearn.over_sampling import SMOTE

: #Balancing
  smote=SMOTE(sampling_strategy=0.2)
  X_train,y_train=smote.fit_resample(X_train,y_train)
  (y_train.value_counts()/len(y_train))*100

: 0    83.333333
  1    16.666667
  Name: hospital_death, dtype: float64
```

7. We have unbalanced data in the ratio 91: 8
8. We balanced the data using smote technique. In the ratio of 80 : 20

BASE MODEL

In [277]:
```
print(confusion_matrix(y_test, y_pred))


print(metrics.classification_report(y_test, y_pred)) #logistic regression
```

```
[[24347   886]
 [ 1589   692]]
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     25233
           1       0.44      0.30      0.36      2281

    accuracy                           0.91     27514
   macro avg       0.69      0.63      0.66     27514
weighted avg       0.90      0.91      0.90     27514
```

In [284]:
```
# Example: Adjusting threshold to maximize sensitivity or specificity
optimal_threshold # Adjust this threshold based on your analysis
y_pred_adjusted = (probs >= optimal_threshold).astype(int)
```

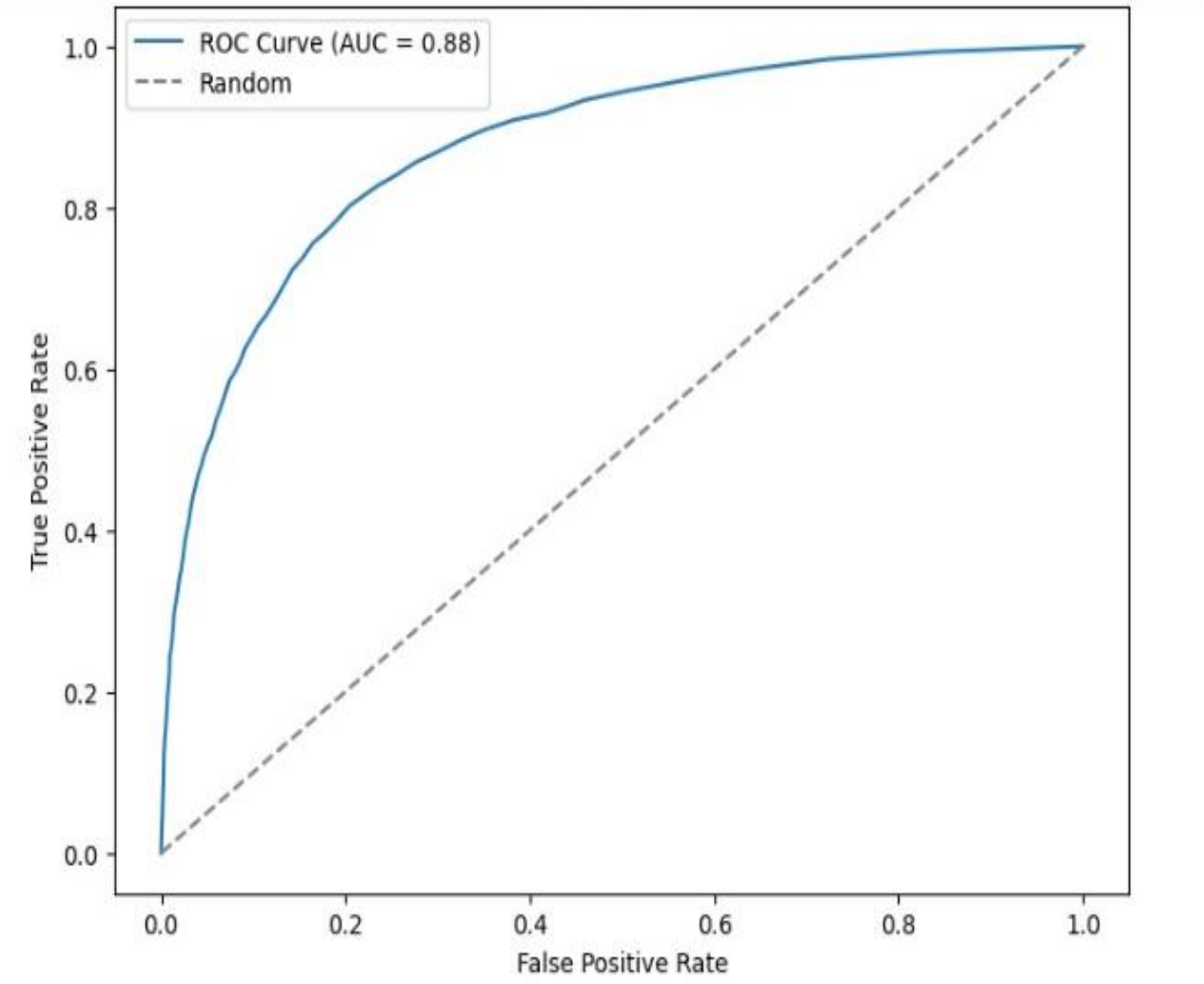
In [285]:
```
# Example: Retrain the model with the adjusted threshold
model_adjusted = LogisticRegression()
model_adjusted.fit(X_train, y_train)
y_pred_adjusted = (model_adjusted.predict_proba(X_test)[:, 1] >= optimal_threshold).astype(int)
```

In [286]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matri
```

```
# Print the optimal threshold and AUC score
print("Optimal Threshold: %f" % optimal_threshold)
print("AUC Score: %f" % auc_score)
```

```
Optimal Threshold: 0.078119
AUC Score: 0.873508
```

ROC- AUC CURVE :



----------Naive Bayes----------

```
91]: from sklearn.naive_bayes import GaussianNB
```

```
92]: nbmodel = GaussianNB()
nbmodel.fit(X_train, y_train)
print(nbmodel)
# make predictions
expected = y_test
predicted = nbmodel.predict(X_test)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

```
GaussianNB()
              precision    recall  f1-score   support

           0       0.97      0.70      0.81     25233
           1       0.18      0.75      0.29      2281

    accuracy                           0.70     27514
   macro avg       0.58      0.72      0.55     27514
weighted avg       0.90      0.70      0.77     27514

[[17687  7546]
 [  581  1700]]
```

```
In [297]: nbmodel = GaussianNB()
nbmodel.fit(X_train_rfe, y_train_rfe)
print(nbmodel)
# make predictions
expected = y_test_rfe
predicted = nbmodel.predict(X_test_rfe)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

```
GaussianNB()
              precision    recall  f1-score   support

           0       0.97      0.71      0.82     25233
           1       0.19      0.72      0.30      2281

    accuracy                           0.71     27514
   macro avg       0.58      0.72      0.56     27514
weighted avg       0.90      0.71      0.78     27514

[[17973  7260]
 [  629  1652]]
```

Bagging Classifier with Decision tree as base model

	precision	recall	f1-score	support
0	0.94	0.99	0.96	25233
1	0.66	0.26	0.37	2281
accuracy			0.93	27514
macro avg	0.80	0.62	0.67	27514
weighted avg	0.91	0.93	0.91	27514

K- Nearest Neighbour

```
6]: # summarize the fit of the model
    print(metrics.classification_report(y_test, y_pred_adjusted)) #k nearest neighbour
```

	precision	recall	f1-score	support
0	0.98	0.80	0.88	25233
1	0.26	0.80	0.39	2281
accuracy			0.80	27514
macro avg	0.62	0.80	0.64	27514
weighted avg	0.92	0.80	0.84	27514

- These are the few models . which we have tried . which did not satisfied the specific bussiness problems
- The recall is not good which is improved in the final model

FINAL MODEL:

Getting Optimal Threshold

In [282]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# Load your dataset
#X, y = load_your_data()

# Create a logistic regression object
model = LogisticRegression()

# Fit the model to the data
model.fit(X, y)

# Predict the probabilities
probs = model.predict_proba(X)

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y, probs[:, 1])

# Calculate the AUC score
auc_score = roc_auc_score(y, probs[:, 1])

# Find the optimal threshold
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]

# Print the optimal threshold and AUC score
print("Optimal Threshold: %f" % optimal_threshold)
print("AUC Score: %f" % auc_score)
```

```
# Print the optimal threshold and AUC score
print("Optimal Threshold: %f" % optimal_threshold)
print("AUC Score: %f" % auc_score)
```

```
Optimal Threshold: 0.078119
AUC Score: 0.873508
```

```
# Example: Adjusting threshold to maximize sensitivity or specificity
optimal_threshold # Adjust this threshold based on your analysis
y_pred_adjusted = (probs >= optimal_threshold).astype(int)
```

```
# Example: Retrain the model with the adjusted threshold
model_adjusted = LogisticRegression()
model_adjusted.fit(X_train, y_train)
y_pred_adjusted = (model_adjusted.predict_proba(X_test)[:, 1] >= optimal_threshold).astype(int)
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Example: Evaluate the model with the adjusted threshold
accuracy = accuracy_score(y_test, y_pred_adjusted)
precision = precision_score(y_test, y_pred_adjusted)
recall = recall_score(y_test, y_pred_adjusted)
f1 = f1_score(y_test, y_pred_adjusted)
confusion = confusion_matrix(y_test, y_pred_adjusted)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"Confusion Matrix:\n{confusion}")
```

```
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"Confusion Matrix:\n{confusion}")
```

```
Accuracy: 0.7072762957040052
Precision: 0.19771703843334382
Recall: 0.8277071459886015
F1 Score: 0.31918850380388847
Confusion Matrix:
[[17572  7661]
 [  393  1888]]
```

[287]:
```
print(metrics.classification_report(y_test, y_pred_adjusted))
```

```
              precision    recall  f1-score   support

           0       0.98      0.70      0.81     25233
           1       0.20      0.83      0.32      2281

    accuracy                           0.71     27514
   macro avg       0.59      0.76      0.57     27514
weighted avg       0.91      0.71      0.77     27514
```

We tried many models such as logistic regression, naive base model, Ada boosting, bagging, random forest classifier. And selected logistic regression with adjusted threshold.

Create a logistic regression object :

- Fit the model to the data
- Predict the probabilities
- Calculate the ROC curve
- Calculate the AUC score
- Find the optimal threshold
- Adjusting threshold to maximize sensitivity or specificity
- Retrain the model with the adjusted threshold
- Evaluate the model with the adjusted threshold

**********Model evaluation**********

- Precision (Positive Predictive Value): 0.20
- Of all instances predicted as critical, only 20% are true critical cases.
- Recall (Sensitivity or True Positive Rate): 0.83
- The model is able to capture 83% of actual critical cases.
- F1-Score: 0.32
- The harmonic mean of precision and recall is relatively low at 0.32.
- Accuracy: 0.71
- The overall accuracy of the model is 71%.

**********Interpretation**********

Precision (0.20):

A low precision suggests that among the instances predicted as critical, a large proportion are false positives. In other words, there are many instances predicted as critical that are not truly critical.

Recall (0.83):

A high recall indicates that the model is effective at capturing true critical cases. However, this comes at the cost of a lower precision.

F1-Score (0.32):

The F1-score is a balance between precision and recall. A lower F1-score suggests that achieving a balance between precision and recall is challenging with the current threshold.

Accuracy (0.71):

Accuracy provides an overall measure of correct predictions but may not be the most informative metric, especially in imbalanced datasets

**********WHY WE SELECTED THIS MODEL*********

1. Priority on Identifying Critical Cases:

- The primary goal in identifying critical patients is to ensure that the model captures as many true critical cases as possible. Missing critical cases (false negatives) can have severe consequences, potentially leading to delayed or lack of timely intervention.

2. Recall Emphasizes Minimizing False Negatives:

- Recall (sensitivity or true positive rate) specifically measures the ability of the model to identify all relevant instances of the positive class. In the context of critical patients, high recall means minimizing the chances of missing a patient who requires urgent attention.

3. Consequences of False Negatives:

- In healthcare scenarios, the consequences of missing a critical patient can be significant, potentially affecting patient outcomes and safety. Therefore, the

emphasis is often on reducing false negatives, even if it results in a higher number of false positives.

4. Trade-off with Precision:

- While precision (positive predictive value) is still important, it may be more acceptable to have a lower precision if it means achieving a higher recall. The cost of false positives (non-critical cases predicted as critical) might be less critical than the cost of missing a true critical case.

5. Timely Intervention and Patient Safety:

- Timely intervention for critical patients is a key consideration. High recall ensures that the model is effective at identifying most, if not all, cases that require urgent medical attention, contributing to patient safety.

TO DO LIST

1. *Regularization Techniques can be done to improve precision values*
2. **Continuous Model Improvement:**

- Implement a continuous improvement strategy for the model by regularly updating it based on feedback, new data, and evolving medical knowledge. This ensures that the model stays relevant and effective in identifying critical patients across different scenarios.

3. **Feedback Loop with Healthcare Professionals:**

- Establish a robust feedback loop with healthcare professionals to gather insights into the model's performance. Encourage open communication to understand cases where the model may have made incorrect predictions and use this feedback to refine the model further.

Limitation :

- In distribution plot the depression is seen which shows normal range (for example heart rate) and we got ICU patient data whose which cannot be in normal range
- There were three columns which had negative values
- Categorical columns show high number of variability and less repeated values so that we perform Binning of the columns
- There was highly skewed for some of the columns.
- Low precision suggests there are many instances predicted as critical that are not truly critical.

**********Business insights**********

1. Patient-Centric Approach:

- Prioritize a patient-centric approach that focuses on individualized care. Acknowledge the uniqueness of each patient's medical data and the variability in parameters. Emphasize the importance of personalized risk assessment to cater to diverse patient profiles.

2. Dynamic Model Thresholds:

- Recognize that medical data can vary significantly, and different patient populations may exhibit diverse patterns. Allow for dynamic adjustment of the model's decision threshold based on specific patient groups or clinical contexts. This flexibility helps in balancing sensitivity and specificity as needed.

3. Multidimensional Risk Assessment:

- Develop a multidimensional risk assessment approach that considers a broad range of parameters and their interactions. Beyond traditional vital signs, incorporate relevant patient history, comorbidities, and laboratory results to provide a comprehensive evaluation of a patient's condition.

4. Resource Optimization and Alerts:

- Leverage the model's predictions to optimize resource allocation by providing alerts or recommendations for additional tests and resources when a patient is flagged as potentially critical. This ensures timely interventions and minimizes the risk of overlooking critical cases.

5. Integration with Electronic Health Records (EHR):

- Integrate the model seamlessly with electronic health records to facilitate a holistic view of a patient's medical history. This integration enhances the model's ability to make informed predictions by considering longitudinal data and historical trends.

6. Ethical Considerations and Patient Privacy:

- Prioritize ethical considerations and patient privacy in the deployment of the model. Ensure that the model adheres to relevant healthcare regulations and standards. Transparency in how the model operates and handles patient data is essential for building trust with both healthcare professionals and patients.

7. Clinical Validation and Real-World Testing:

- Conduct rigorous clinical validation and real-world testing to assess the model's performance in diverse healthcare settings. Collaborate with medical professionals to validate the model's effectiveness in different hospitals, clinics, or healthcare systems.

8. Education and Training for Healthcare Providers:

- Provide comprehensive education and training to healthcare providers on the model's capabilities, limitations, and interpretability. Empower them to use the model as a supportive tool in their decision-making processes.

Closing statement

To improve the model – we can have more balanced data with better sampling techniques.so to improve the precision of the model.

REFRENCE USED

https://www.kaggle.com/datasets/mitishaagarwal/patient
