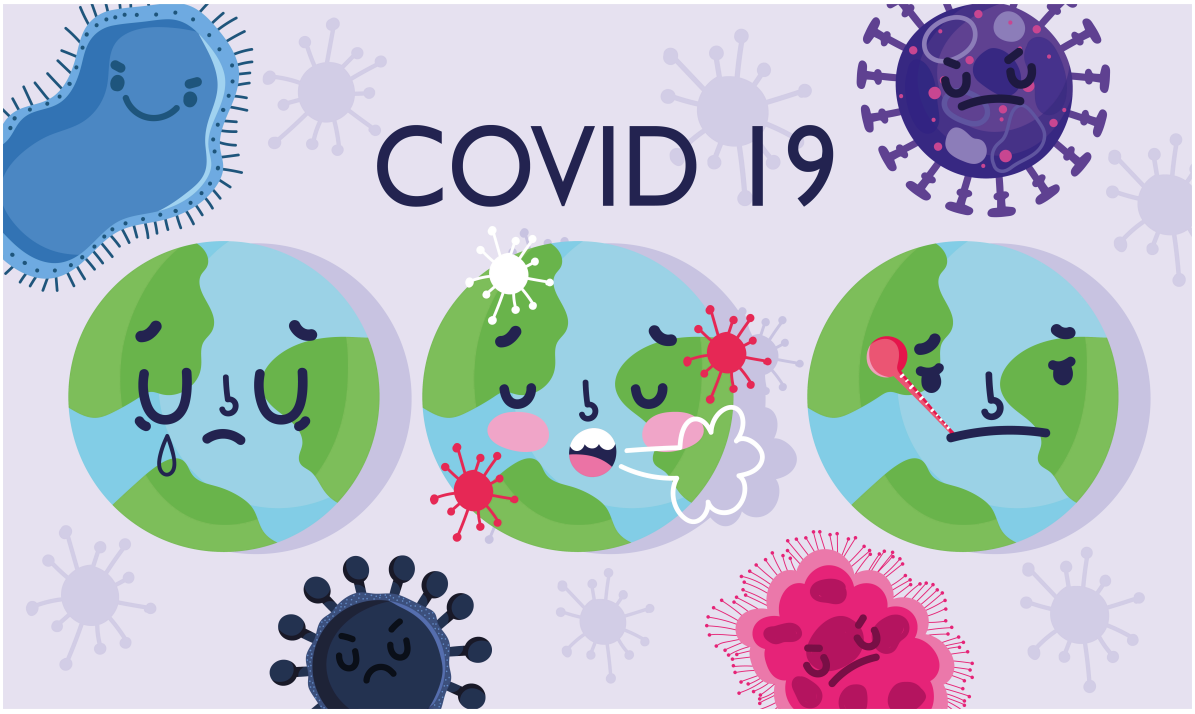
Swarnalatha Sethuraman

Capstone Project Description and Objective

Coronavirus disease (COVID-19) is an infectious disease caused by the SARS-CoV-2 virus.

Most people infected with the virus will experience mild to moderate respiratory illness and recover without requiring special treatment. However, some will become seriously ill and require medical attention. Older people and those with underlying medical conditions like cardiovascular disease, diabetes, chronic respiratory disease, or cancer are more likely to develop serious illness. Anyone can get sick with COVID-19 and become seriously ill or die at any age.

Machine learning algorithms and models can be used to predict important future Covid case data such as infection rates (outbreaks) and mortality rates. The predicted case data can be used to help control Covid, aid in better treatments and vaccines, or even lead to a cure.



Dataset Description

The datasets consists of several medical predictor variables and one target variable (Classification Final). Predictor variables includes age, sex, patient type, different pregnancy-related situations, patients with various conditions, and more.

Variables - Description

SEX: Female or male.

AGE: Age of the patient.

CLASSIFICATION_FINAL[Output column] : Different level of covid disease degrees.

PATIENT TYPE: Type of care the patient received in the unit.

PNEUMONIA: Showing the patient has air sacs inflammation level in medical terms measure.

PREGNANCY: Whether the patient is pregnant or not.

DIABETES: Whether the patient has diabetes or not.

COPD: Indicates whether the patient has Chronic obstructive pulmonary disease or not.

ASTHMA: Whether the patient has asthma or not.

INMSUPR: Whether the patient is immunosuppressed or not.

HYPERTENSION: Whether the patient has hypertension or not.

CARDIOVASCULAR: Whether the patient has heart or blood vessels related disease or not.

RENAL CHRONIC: Whether the patient has chronic renal disease or not.

OTHER DISEASE: Whether the patient has another disease or not.

OBESITY: Whether the patient is obese or not.

TOBACCO: Whether the patient is a tobacco user or not.

USMER: Indicates different levels of medical units the patient has been treated.

MEDICAL UNIT: Type of institution of the National Health System that provided the care.

INTUBED: Showing the different level of ventilator a patient is connected.

ICU: Indicates whether the patient had been admitted to an Intensive Care Unit.

DATE DIED: Indicates whether patient died or not.

Read Data and Perform descriptive analysis:

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

```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
data = pd.read_csv('Covid_Dataset.csv')
data.head()
```

Out[2]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	PNEUMONIA	AGE	PREGNAN
0	2	1	1	1	03-05-2020	3	1	65	
1	2	1	2	1	03-06-2020	3	1	72	
2	2	1	2	2	09-06-2020	1	2	55	
3	2	1	1	1	12-06-2020	3	2	53	
4	2	1	2	1	21-06-2020	3	2	68	

5 rows × 21 columns

In [3]:
```
# Total number of rows and columns
data.shape
```

Out[3]:
```
(199999, 21)
```

We have 199999 rows and 21 columns.

In [4]:
```
# Getting information of the dataframe
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 199999 entries, 0 to 199998
Data columns (total 21 columns):
 #   Column                Non-Null Count   Dtype 
---  ------                --------------   ----- 
 0   USMER                 199999 non-null  int64 
 1   MEDICAL_UNIT          199999 non-null  int64 
 2   SEX                   199999 non-null  int64 
 3   PATIENT_TYPE          199999 non-null  int64 
 4   DATE_DIED             199999 non-null  object
 5   INTUBED               199999 non-null  int64 
 6   PNEUMONIA             199999 non-null  int64 
 7   AGE                   199999 non-null  int64 
 8   PREGNANT              199999 non-null  int64 
 9   DIABETES              199999 non-null  int64 
 10  COPD                  199999 non-null  int64 
 11  ASTHMA                199999 non-null  int64 
 12  INMSUPR               199999 non-null  int64 
 13  HIPERTENSION          199999 non-null  int64 
 14  OTHER_DISEASE         199999 non-null  int64 
 15  CARDIOVASCULAR        199999 non-null  int64 
 16  OBESITY               199999 non-null  int64 
 17  RENAL_CHRONIC         199999 non-null  int64 
 18  TOBACCO               199999 non-null  int64 
 19  CLASIFFICATION_FINAL  199999 non-null  int64 
 20  ICU                   199999 non-null  int64 
dtypes: int64(20), object(1)
memory usage: 32.0+ MB
```

In [5]:
```
# Getting column names in the dateframe
data.columns
```

Out[5]:
```
Index(['USMER', 'MEDICAL_UNIT', 'SEX', 'PATIENT_TYPE', 'DATE_DIED', 'INTUBED',
       'PNEUMONIA', 'AGE', 'PREGNANT', 'DIABETES', 'COPD', 'ASTHMA', 'INMSUPR',
       'HIPERTENSION', 'OTHER_DISEASE', 'CARDIOVASCULAR', 'OBESITY',
       'RENAL_CHRONIC', 'TOBACCO', 'CLASIFFICATION_FINAL', 'ICU'],
      dtype='object')
```

In [6]:
```
# Checking unique values in each column
data.nunique()
```

Out[6]:
```
USMER                  2
MEDICAL_UNIT           4
SEX                    2
PATIENT_TYPE           2
DATE_DIED            337
INTUBED                4
PNEUMONIA              3
AGE                  105
PREGNANT               4
DIABETES               2
COPD                   2
ASTHMA                 2
INMSUPR                2
HIPERTENSION           2
OTHER_DISEASE          2
CARDIOVASCULAR         2
OBESITY                2
RENAL_CHRONIC          2
TOBACCO                2
CLASIFFICATION_FINAL   7
ICU                    2
dtype: int64
```

In [7]:
```
# Describes the data-Descriptive Statistics
data.describe()
```

Out[7]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	INTUBED	PNEUMONIA	
count	199999.000000	199999.000000	199999.000000	199999.000000	199999.000000	199999.000000	1
mean	1.560453	3.900170	1.523433	1.346587	2.578108	4.095105	
std	0.496333	0.309976	0.499452	0.475884	0.656066	14.817906	
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	
25%	1.000000	4.000000	1.000000	1.000000	2.000000	2.000000	
50%	2.000000	4.000000	2.000000	1.000000	3.000000	2.000000	
75%	2.000000	4.000000	2.000000	2.000000	3.000000	2.000000	
max	2.000000	4.000000	2.000000	2.000000	4.000000	99.000000	

In [8]:
```
# Check the total missing values in each column
data.isnull().sum()
```

Out[8]:
```
USMER                 0
MEDICAL_UNIT          0
SEX                   0
PATIENT_TYPE          0
DATE_DIED             0
INTUBED               0
PNEUMONIA             0
AGE                   0
PREGNANT              0
DIABETES              0
COPD                  0
ASTHMA                0
INMSUPR               0
HIPERTENSION          0
OTHER_DISEASE         0
CARDIOVASCULAR        0
OBESITY               0
RENAL_CHRONIC         0
TOBACCO               0
CLASIFFICATION_FINAL  0
ICU                   0
dtype: int64
```

This looks to be a clean dataset without any missing values.

In [9]:
```
# Checking data types of columns
data.dtypes
```

Out[9]:
```
USMER                  int64
MEDICAL_UNIT           int64
SEX                    int64
PATIENT_TYPE           int64
DATE_DIED             object
INTUBED                int64
PNEUMONIA              int64
AGE                    int64
PREGNANT               int64
DIABETES               int64
COPD                   int64
ASTHMA                 int64
INMSUPR                int64
HIPERTENSION           int64
OTHER_DISEASE          int64
CARDIOVASCULAR         int64
OBESITY                int64
RENAL_CHRONIC          int64
TOBACCO                int64
CLASIFFICATION_FINAL   int64
ICU                    int64
dtype: object
```

Only Date Died column is an object, rest columns are integers.

In [10]:
```
# Check the duplicate value
data.duplicated(keep='last').sum()
```

Out[10]:
```
125590
```

In [11]:
```
# Dropping the duplicate values
data.drop_duplicates(keep='last',inplace=True)
```

In [12]:
```
# Checking for dropped duplicate values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 74409 entries, 0 to 199998
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   USMER                 74409 non-null  int64
 1   MEDICAL_UNIT          74409 non-null  int64
 2   SEX                   74409 non-null  int64
 3   PATIENT_TYPE          74409 non-null  int64
 4   DATE_DIED             74409 non-null  object
 5   INTUBED               74409 non-null  int64
 6   PNEUMONIA             74409 non-null  int64
 7   AGE                   74409 non-null  int64
 8   PREGNANT              74409 non-null  int64
 9   DIABETES              74409 non-null  int64
 10  COPD                  74409 non-null  int64
 11  ASTHMA                74409 non-null  int64
 12  INMSUPR               74409 non-null  int64
 13  HIPERTENSION          74409 non-null  int64
 14  OTHER_DISEASE         74409 non-null  int64
 15  CARDIOVASCULAR        74409 non-null  int64
 16  OBESITY               74409 non-null  int64
 17  RENAL_CHRONIC         74409 non-null  int64
 18  TOBACCO               74409 non-null  int64
 19  CLASIFFICATION_FINAL  74409 non-null  int64
 20  ICU                   74409 non-null  int64
dtypes: int64(20), object(1)
memory usage: 12.5+ MB
```

In [13]:
```
# Renaming columns
data.rename({'HIPERTENSION':'HYPERTENSION','CLASIFFICATION_FINAL':'CLASSIFICATION_FINA
                   inplace=True)
```

In [14]:
```
data.columns
```

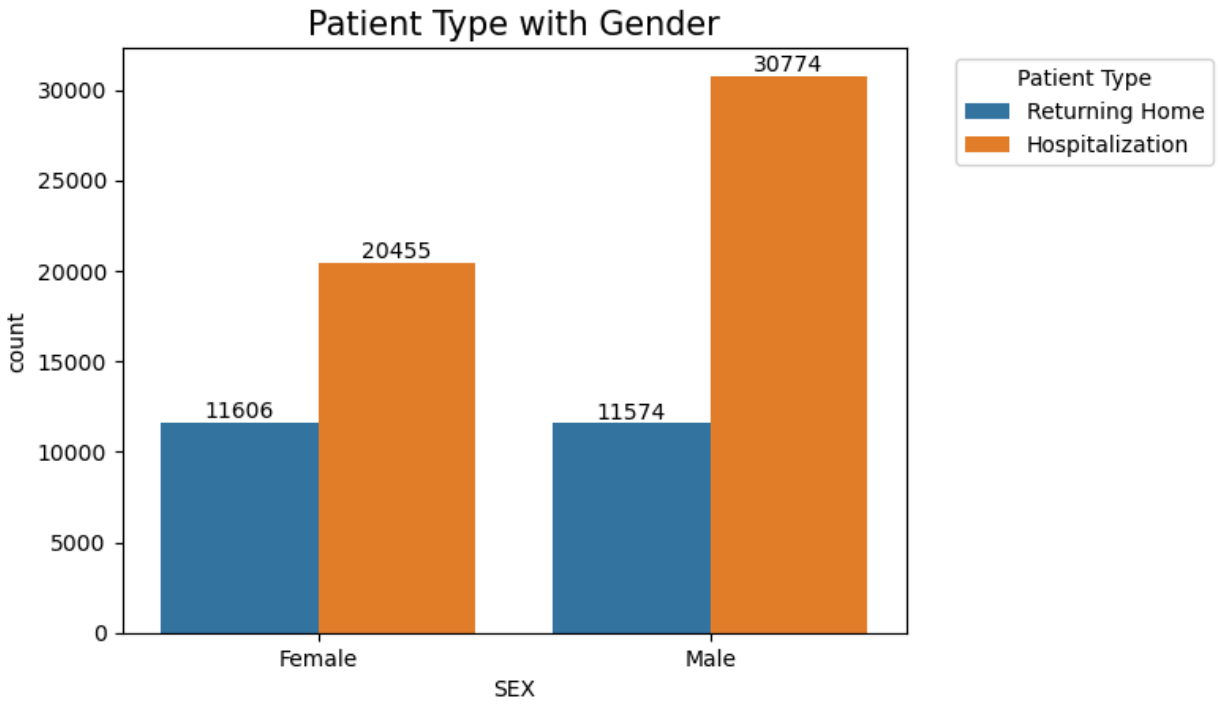
Out[14]:
```
Index(['USMER', 'MEDICAL_UNIT', 'SEX', 'PATIENT_TYPE', 'DATE_DIED', 'INTUBED',
       'PNEUMONIA', 'AGE', 'PREGNANT', 'DIABETES', 'COPD', 'ASTHMA',
       'IMMUNOSUPRS', 'HYPERTENSION', 'OTHER_DISEASE', 'CARDIOVASCULAR',
       'OBESITY', 'RENAL_CHRONIC', 'TOBACCO', 'CLASSIFICATION_FINAL', 'ICU'],
      dtype='object')
```

Exploratory Data Analysis

Plotting against the columns to get insights

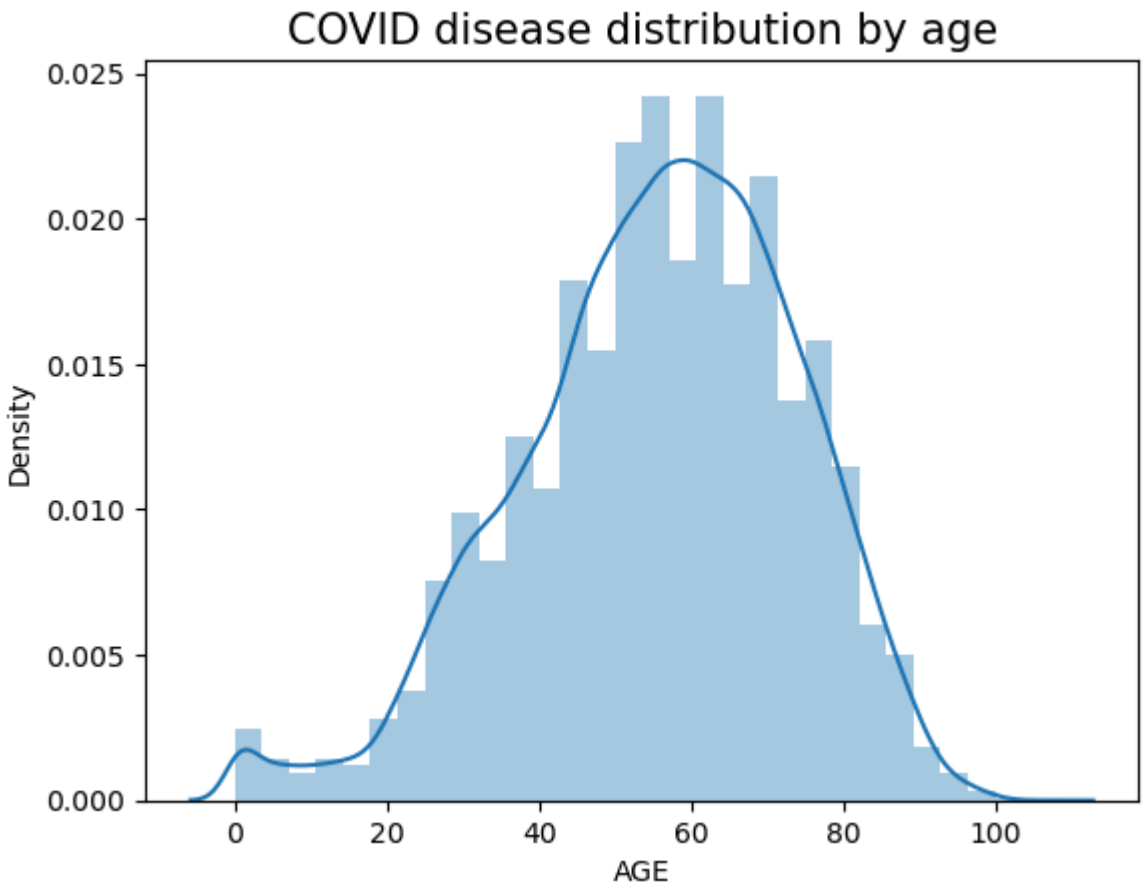
In [15]:
```
ax = sns.countplot(data=data,x='SEX',hue='PATIENT_TYPE');
label = ['Female','Male']
ax.set_xticklabels(label)

plt.legend(title='Patient Type',labels=['Returning Home', 'Hospitalization'],bbox_to_a
```

```
plt.title('Patient Type with Gender',fontsize=15)

for bars in ax.containers:
    ax.bar_label(bars)
```



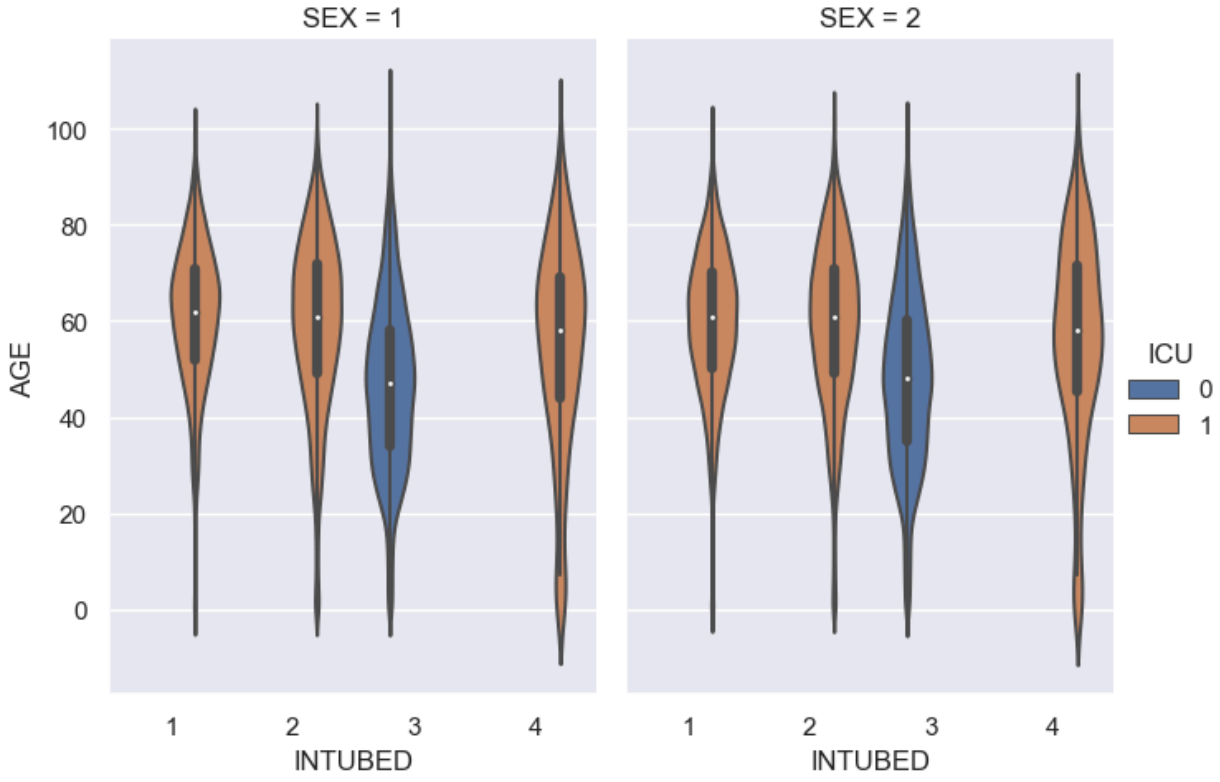
From the plot, Male are affected more compared to female . They were hospitalized at high rate compared to female.

In [16]:
```
sns.distplot(data['AGE'],bins=30);
plt.title('COVID disease distribution by age', fontsize=15);
```



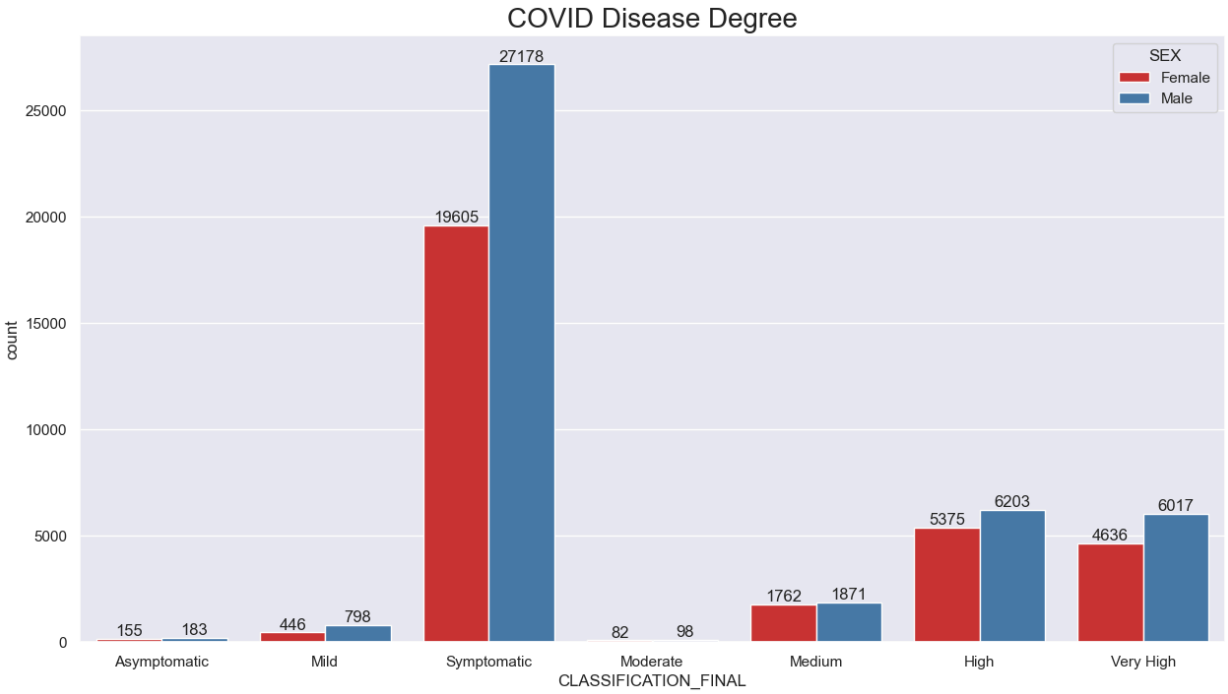
From the plot, age group 40-75 got affected more than other age groups.

In [17]:
```
sns.set(rc={'figure.figsize':(15,8)})
sns.catplot(data=data,x='INTUBED',y='AGE',hue='ICU',col='SEX',kind='violin',aspect=.7)
```



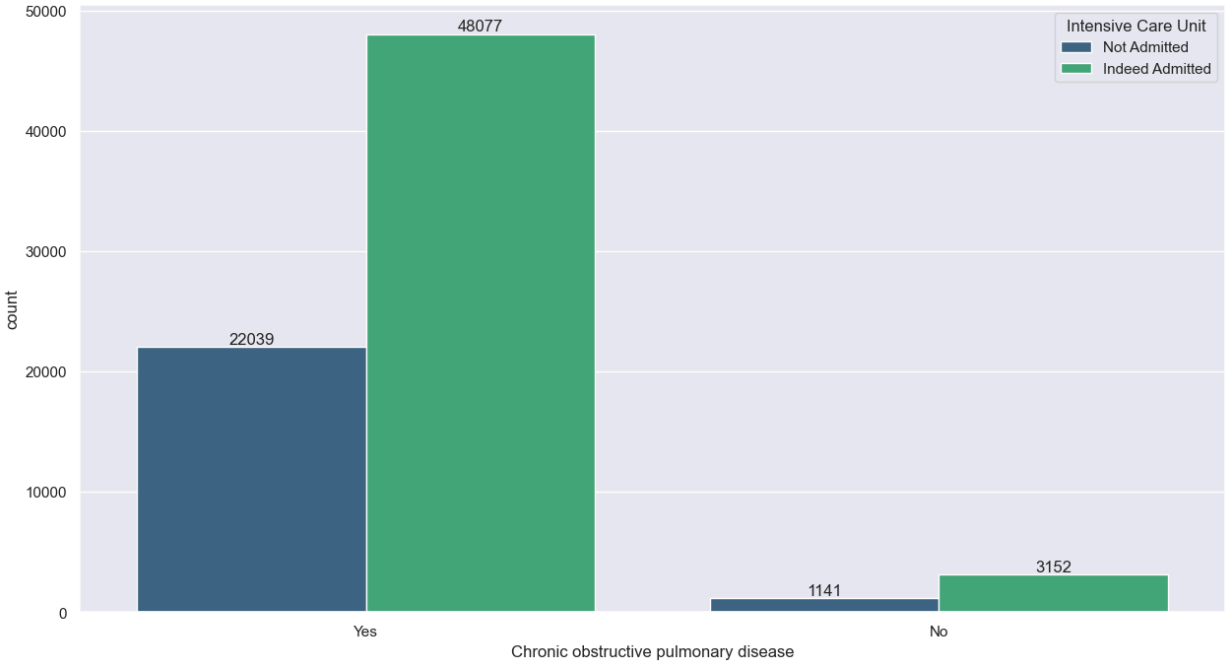
From the above plot, both sex used all the types of ventilators, people using specialized ventilators was not admitted in ICU.

In [18]:
```
sns.set(rc={'figure.figsize':(15,8)})
ax = sns.countplot(data=data,x='CLASSIFICATION_FINAL',palette='Set1',hue='SEX');
label = ['Asymptomatic','Mild','Symptomatic','Moderate','Medium','High','Very High']
ax.set_xticklabels(label);

plt.title('COVID Disease Degree',fontsize=20)
plt.legend(title='SEX',labels=['Female', 'Male'])

for bars in ax.containers:
    ax.bar_label(bars)
```



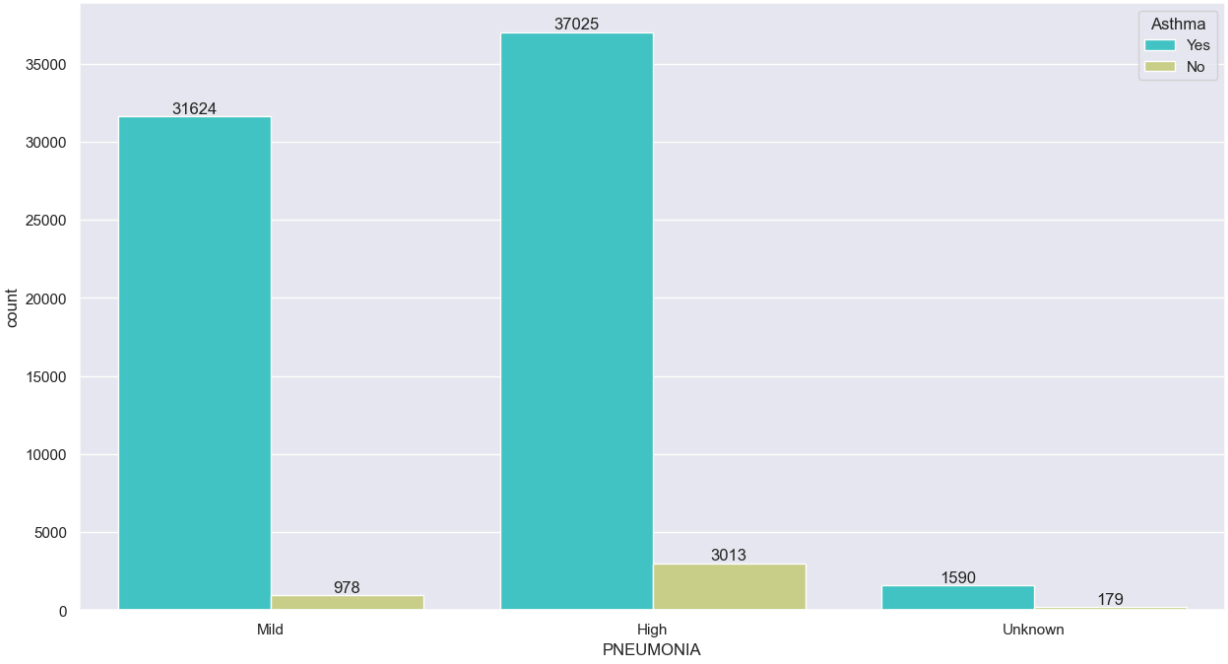
Around 90% of the population showed variety of COVID diseased symptoms. Male are more affected in every symtom type compared to female.

In [19]:
```
ax = sns.countplot(data=data,x='COPD',hue='ICU',palette='viridis');
label = ['Yes', 'No']
ax.set_xticklabels(label);
plt.xlabel('Chronic obstructive pulmonary disease')

plt.legend(title='Intensive Care Unit',labels=['Not Admitted','Indeed Admitted'],bbox_

for bars in ax.containers:
    ax.bar_label(bars)
```



People with COPD was admitted in hospital with COVID disease at higher rate.

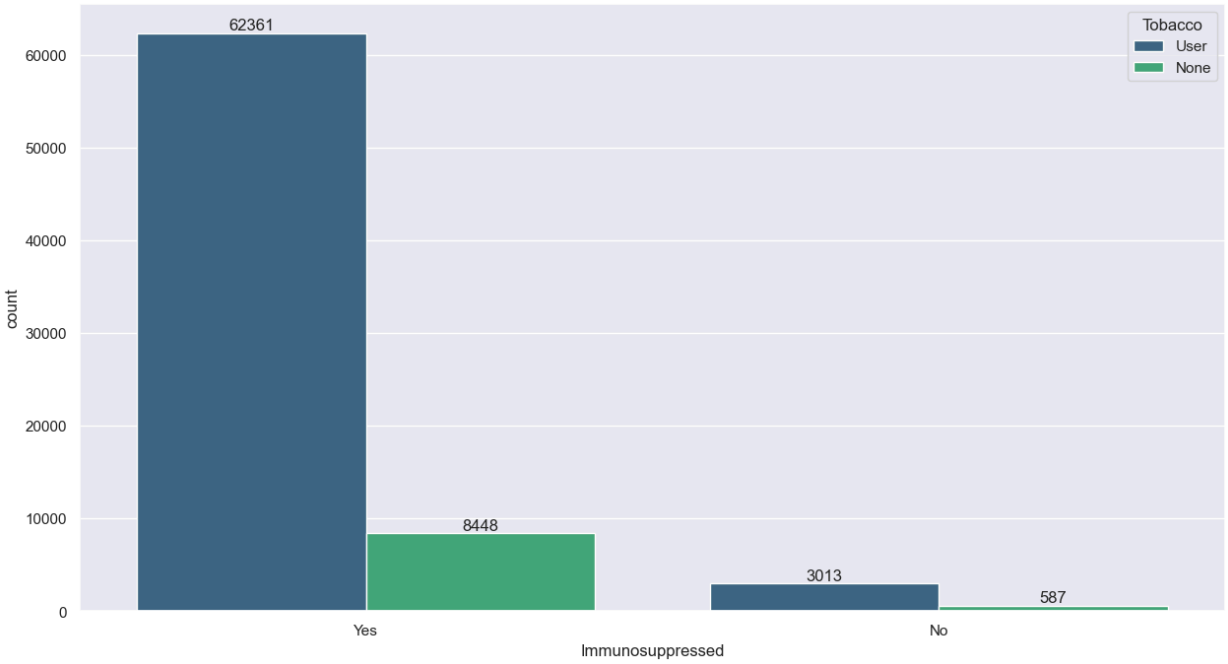
In [20]:
```
ax = sns.countplot(data=data,x='PNEUMONIA',hue='ASTHMA',palette='rainbow');
label = ['Mild','High','Unknown']
ax.set_xticklabels(label);

plt.legend(title='Asthma',labels=['Yes', 'No'],bbox_to_anchor=(1,1))

for bars in ax.containers:
    ax.bar_label(bars)
```



People with Asthma has high rate of Pneumonia and hence easily affected with COVID.

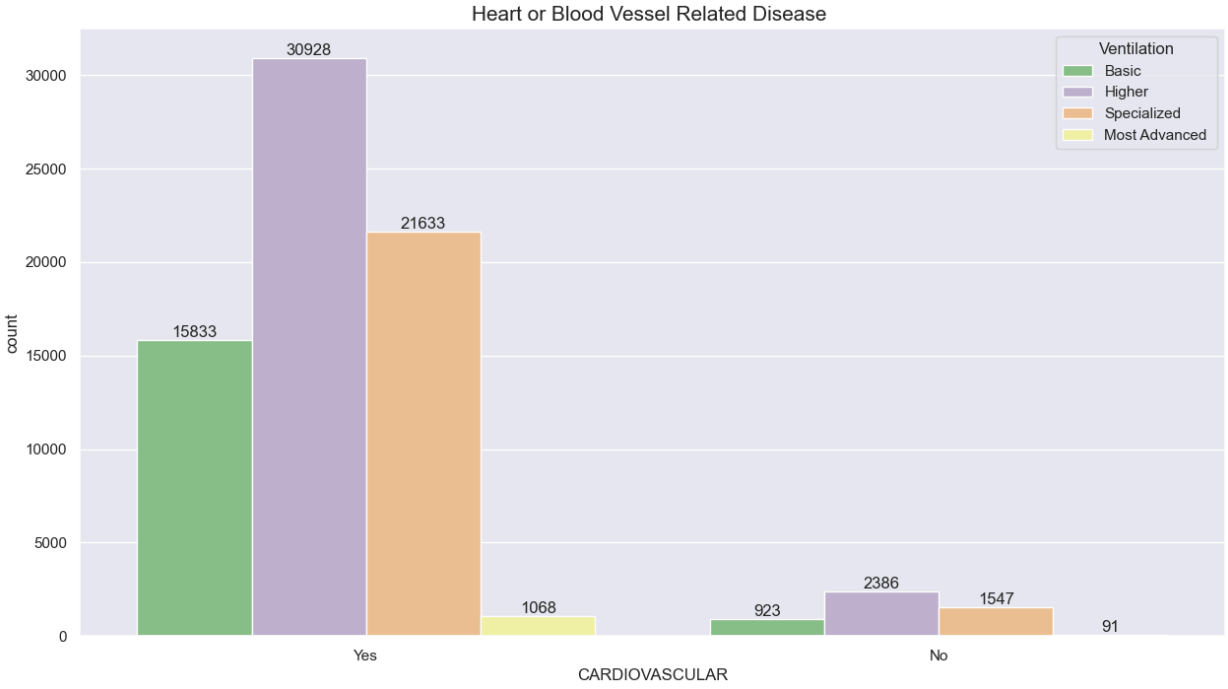
In [21]:
```
ax = sns.countplot(data=data,x='IMMUNOSUPRS',hue='TOBACCO',palette='viridis');
label = ['Yes','No']
ax.set_xticklabels(label);
```

```
plt.xlabel('Immunosuppressed')

plt.legend(title='Tobacco',labels=['User', 'None'],bbox_to_anchor=(1,1))

for bars in ax.containers:
    ax.bar_label(bars)
```



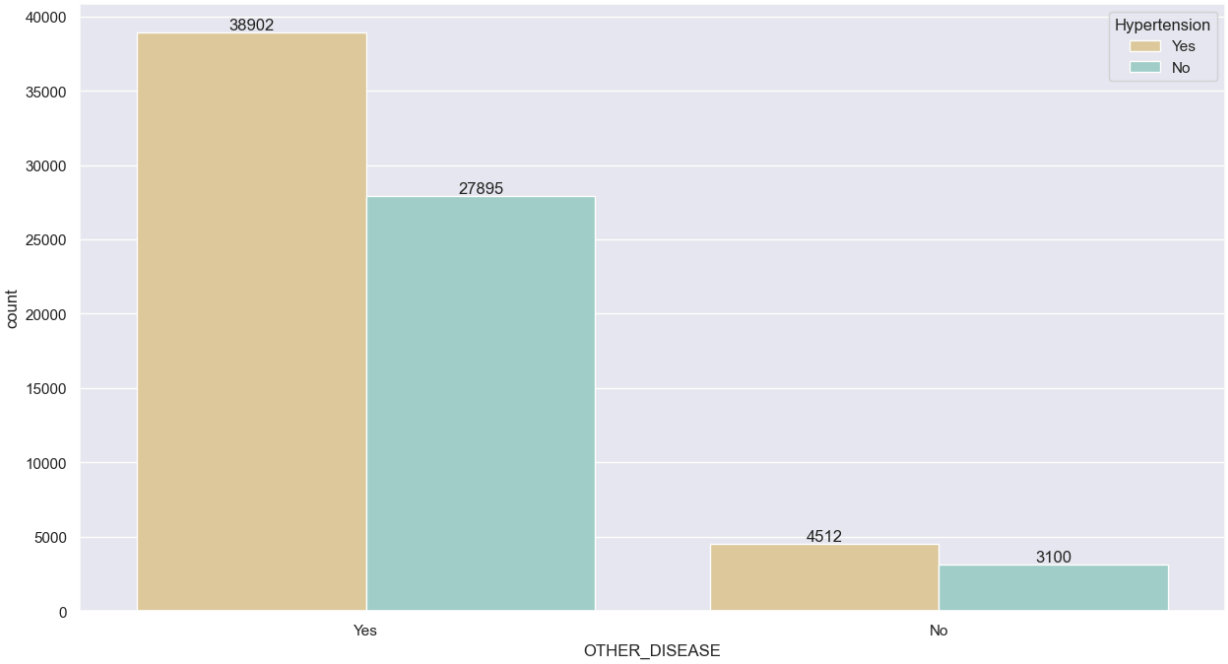
People using Tobacco were more Immunosuppressed which was also the cause for infection.

In [22]:
```
ax = sns.countplot(data=data,x='CARDIOVASCULAR',hue='INTUBED',palette='Accent');
label = ['Yes','No']
ax.set_xticklabels(label);
plt.title('Heart or Blood Vessel Related Disease ',fontsize=15)

plt.legend(title='Ventilation',labels=['Basic', 'Higher','Specialized','Most Advanced

for bars in ax.containers:
    ax.bar_label(bars)
```



People with Cardiovasular disease used ventilator highly.

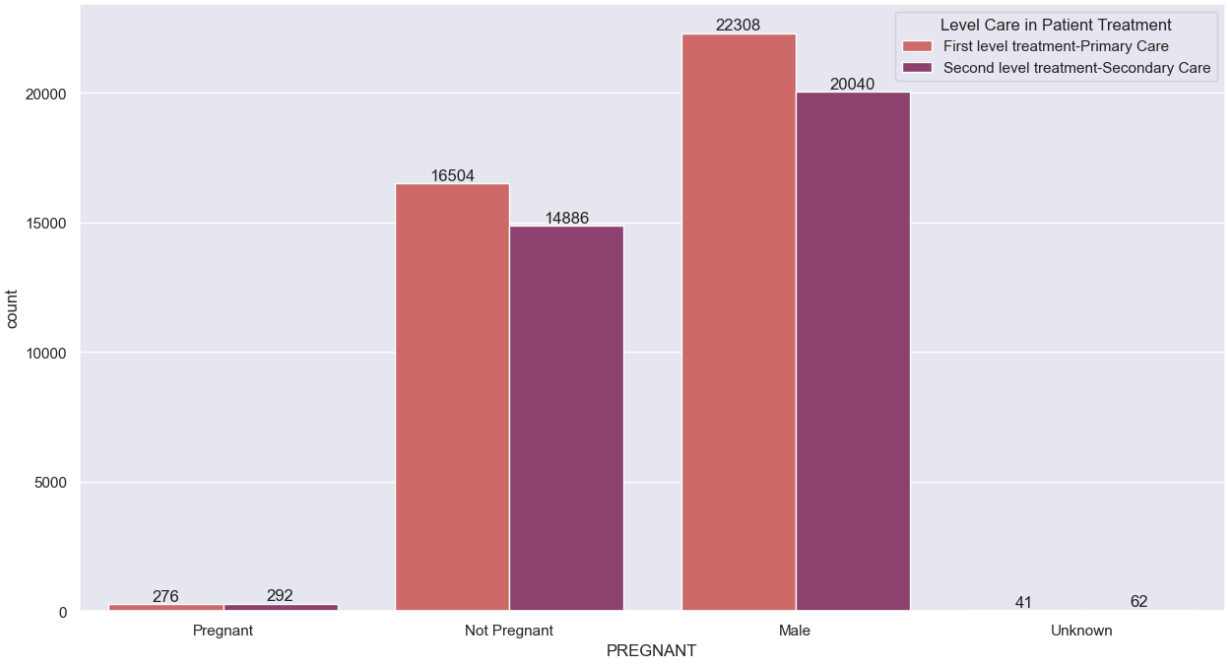
In [23]:
```
ax = sns.countplot(data=data,x='OTHER_DISEASE',hue='HYPERTENSION',palette='BrBG');
label = ['Yes','No']
ax.set_xticklabels(label);

plt.legend(title='Hypertension',labels=['Yes','No'],bbox_to_anchor=(1,1))

for bars in ax.containers:
    ax.bar_label(bars)
```



People with other diseases were more prone with high blood pressure.

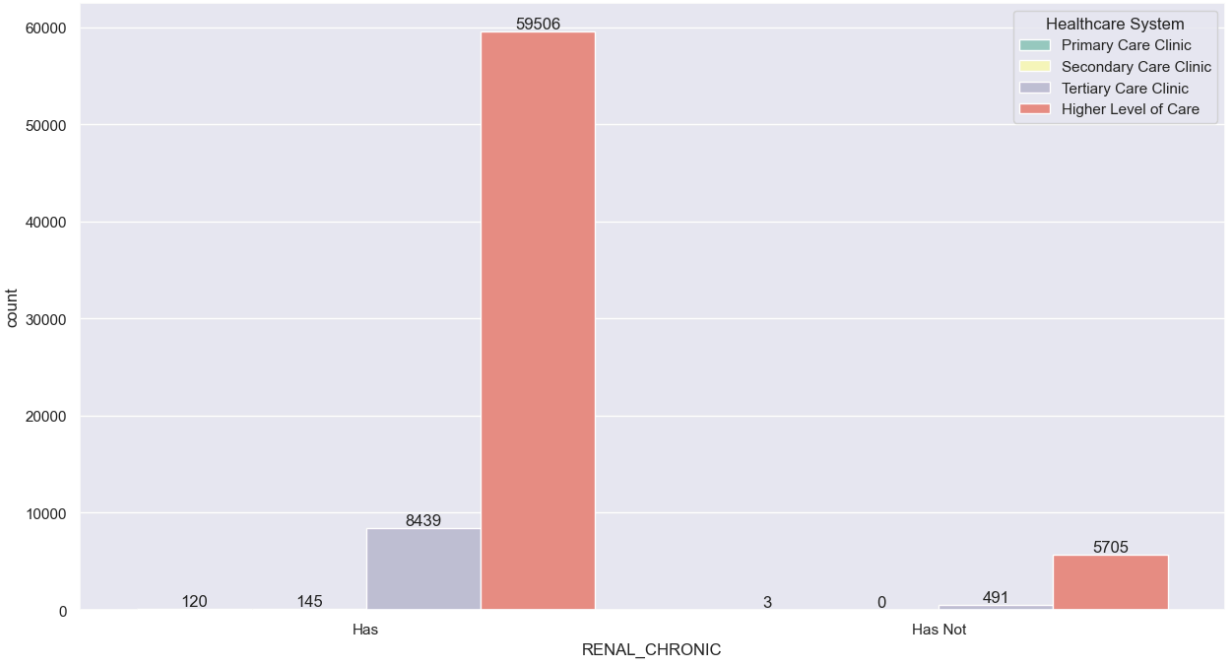
In [24]:
```
ax = sns.countplot(data=data,x='PREGNANT',hue='USMER',palette='flare');
label = ['Pregnant','Not Pregnant','Male','Unknown']
ax.set_xticklabels(label);
```

```
plt.legend(title='Level Care in Patient Treatment',
           labels=['First level treatment-Primary Care','Second level treatment-Seconc

for bars in ax.containers:
    ax.bar_label(bars)
```



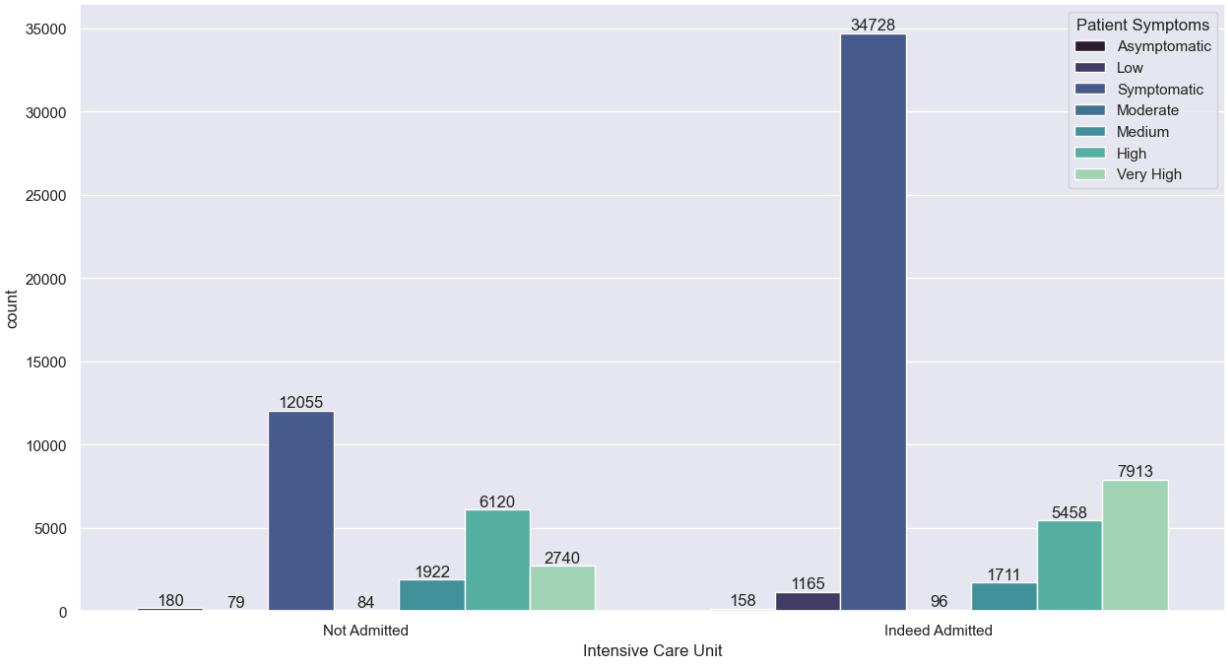
Women who are not pregnant, and Men received both level care treatments highly, compared to pregnant and others.

In [25]:
```
ax = sns.countplot(data=data,x='RENAL_CHRONIC',hue='MEDICAL_UNIT',palette='Set3');
label = ['Has','Has Not']
ax.set_xticklabels(label);

plt.legend(title='Healthcare System',
           labels=['Primary Care Clinic','Secondary Care Clinic','Tertiary Care Clinic
           bbox_to_anchor=(1,1))

for bars in ax.containers:
    ax.bar_label(bars)
```



People who has kidney related disease were admitted in higher level of care or specialized services in national or specialized medical centers.

In [26]:
```
ax = sns.countplot(data=data,x='ICU',hue='CLASSIFICATION_FINAL',palette='mako');
label = ['Not Admitted','Indeed Admitted']
ax.set_xticklabels(label);
plt.xlabel('Intensive Care Unit')

plt.legend(title='Patient Symptoms',labels=['Asymptomatic','Low','Symptomatic','Modera
           bbox_to_anchor=(1,1))

for bars in ax.containers:
    ax.bar_label(bars)
```

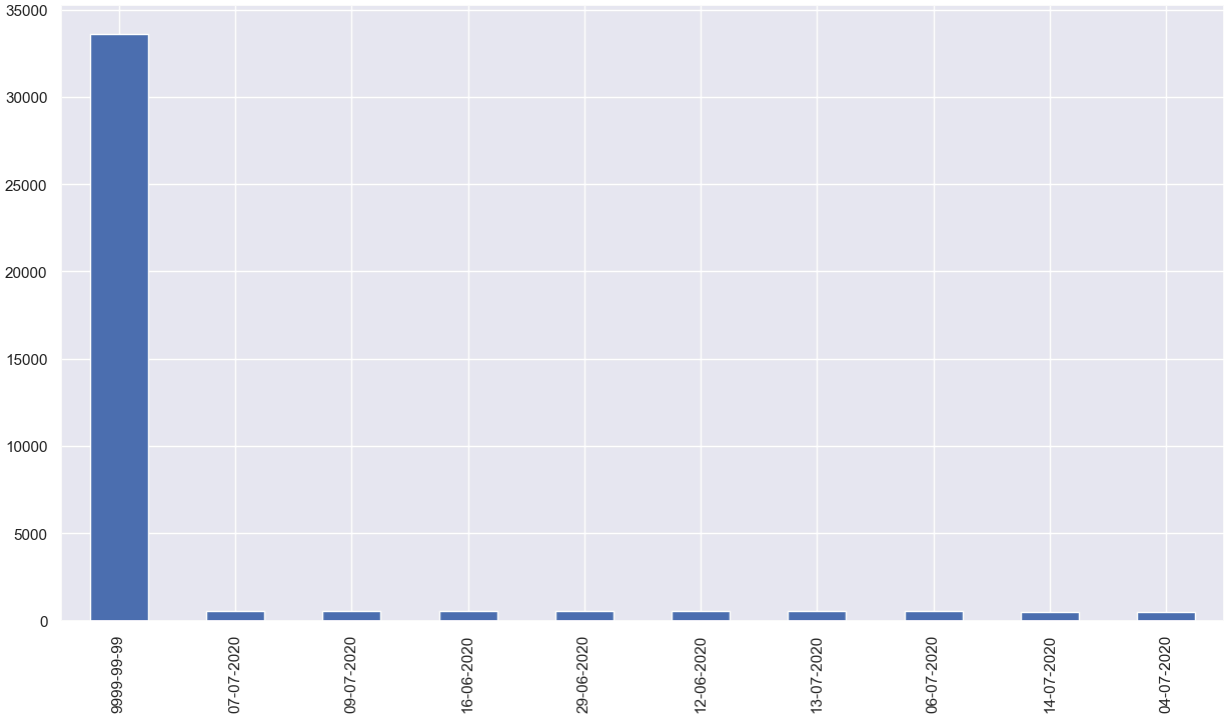


People showing variety of symptoms of COVID disease which became worse were hospitalized in Intensive Care Unit.

In [27]:
```
# Checking unique values with counts in date died column
data['DATE_DIED'].value_counts()
```

Out[27]:
```
9999-99-99    33607
07-07-2020      543
09-07-2020      531
16-06-2020      527
29-06-2020      526
               ...
20-10-2020        1
18-10-2020        1
16-10-2020        1
14-10-2020        1
12-10-2020        1
Name: DATE_DIED, Length: 337, dtype: int64
```

In [28]:
```
data['DATE_DIED'].value_counts().head(10).plot.bar();
```



Around 45% of the population survived and rest died.

In [29]:
```
# Adding a new column in the dataframe and creating that column with patient survived
data['PATIENT_SURVIVED'] = (data['DATE_DIED'] == '9999-99-99').astype(int)
```

In [30]:
```
# New dataframe without Survival date '9999-99-99'
df = data.loc[data['DATE_DIED'] != '9999-99-99']
df
```

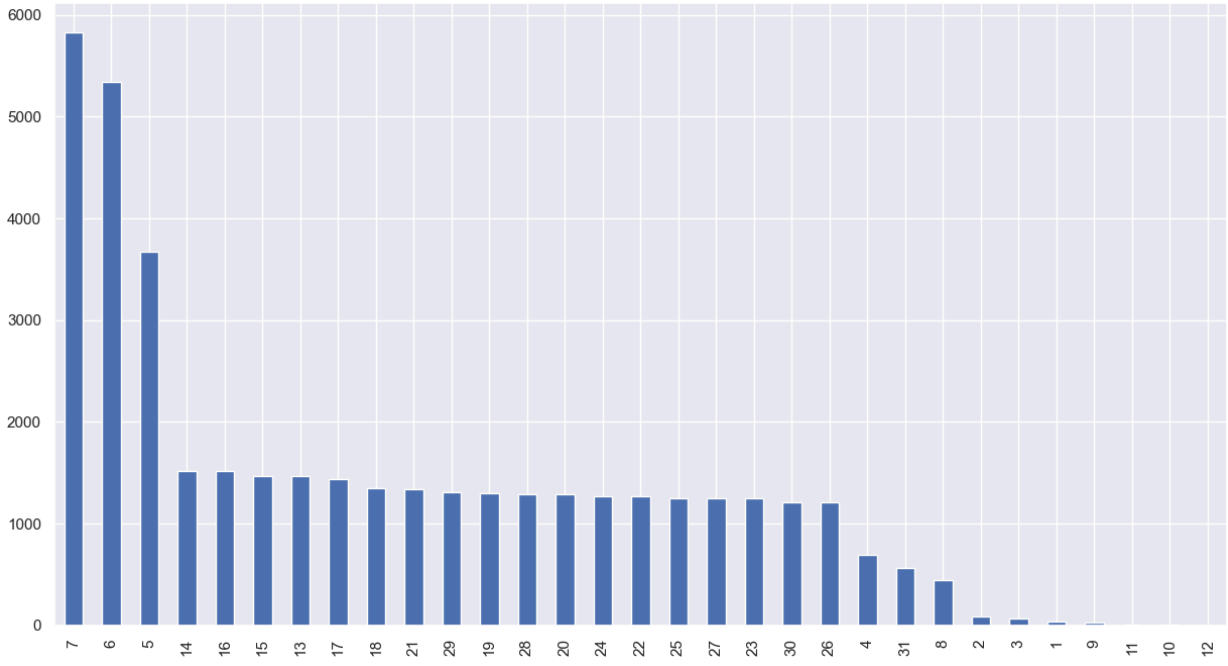
Out[30]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	PNEUMONIA	AGE	PREG
0	2	1	1	1	03-05-2020	3	1	65	
1	2	1	2	1	03-06-2020	3	1	72	
2	2	1	2	2	09-06-2020	1	2	55	
3	2	1	1	1	12-06-2020	3	2	53	
4	2	1	2	1	21-06-2020	3	2	68	
...	...	...	...	...	...	...	...	...	
59395	1	4	2	2	02-05-2021	2	2	75	
59396	1	4	1	2	02-05-2021	2	1	84	
59397	1	4	1	2	02-05-2021	2	2	79	
59398	2	4	2	2	02-05-2021	2	2	66	
59399	2	4	2	2	02-05-2021	2	2	54	

40802 rows × 22 columns

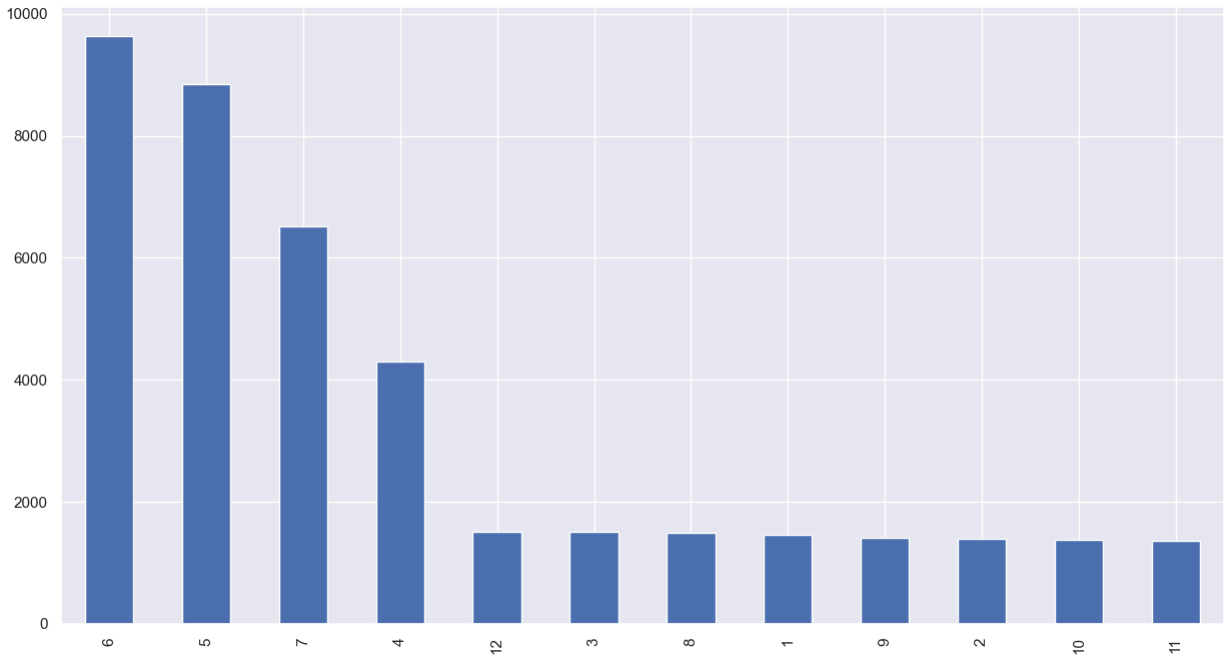
In [31]:
```
# Change Date Died dtype to datetime
df['DATE_DIED'] = pd.to_datetime(df['DATE_DIED'])
```

In [32]:
```
# Getting days of death in the column
DATE = df['DATE_DIED'].dt.day
DATE.value_counts().plot.bar();
```



More people died in certain days, while other days has less or no death.

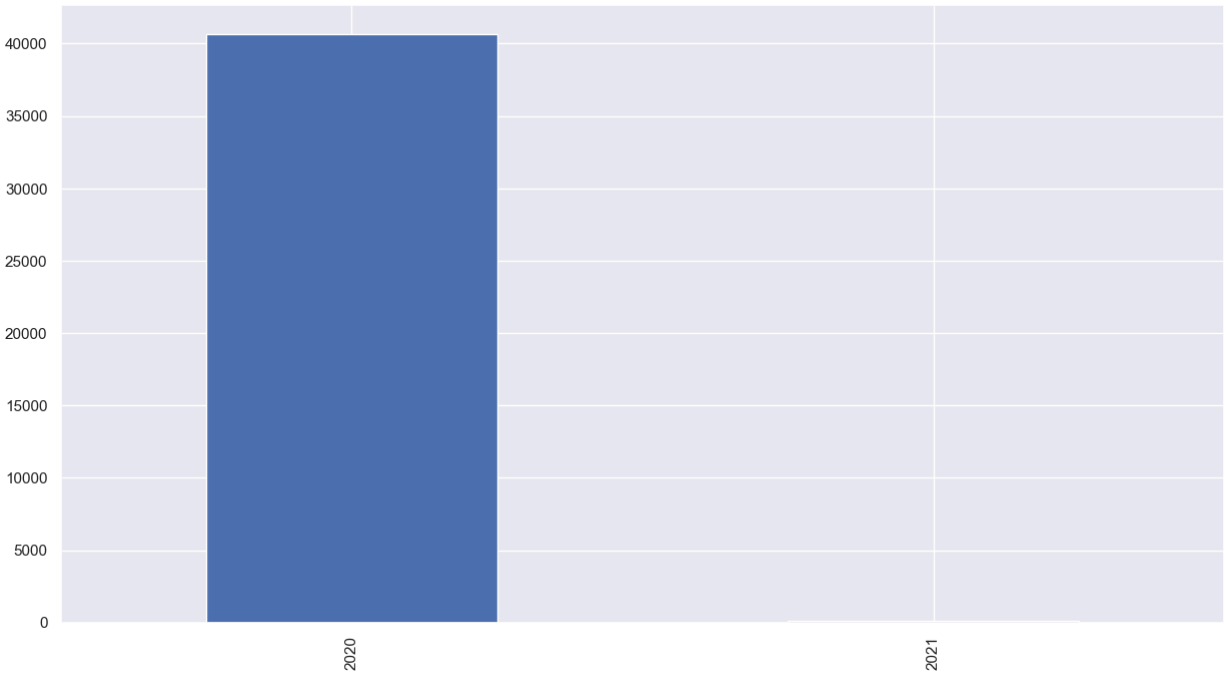
In [33]:
```
# Getting months of death in the column
MONTH = df['DATE_DIED'].dt.month
MONTH.value_counts().plot.bar();
```



April-July were in peak of deaths compared to other months.

In [34]:
```
# year of death
YEAR = df['DATE_DIED'].dt.year
YEAR.value_counts().plot.bar()
```

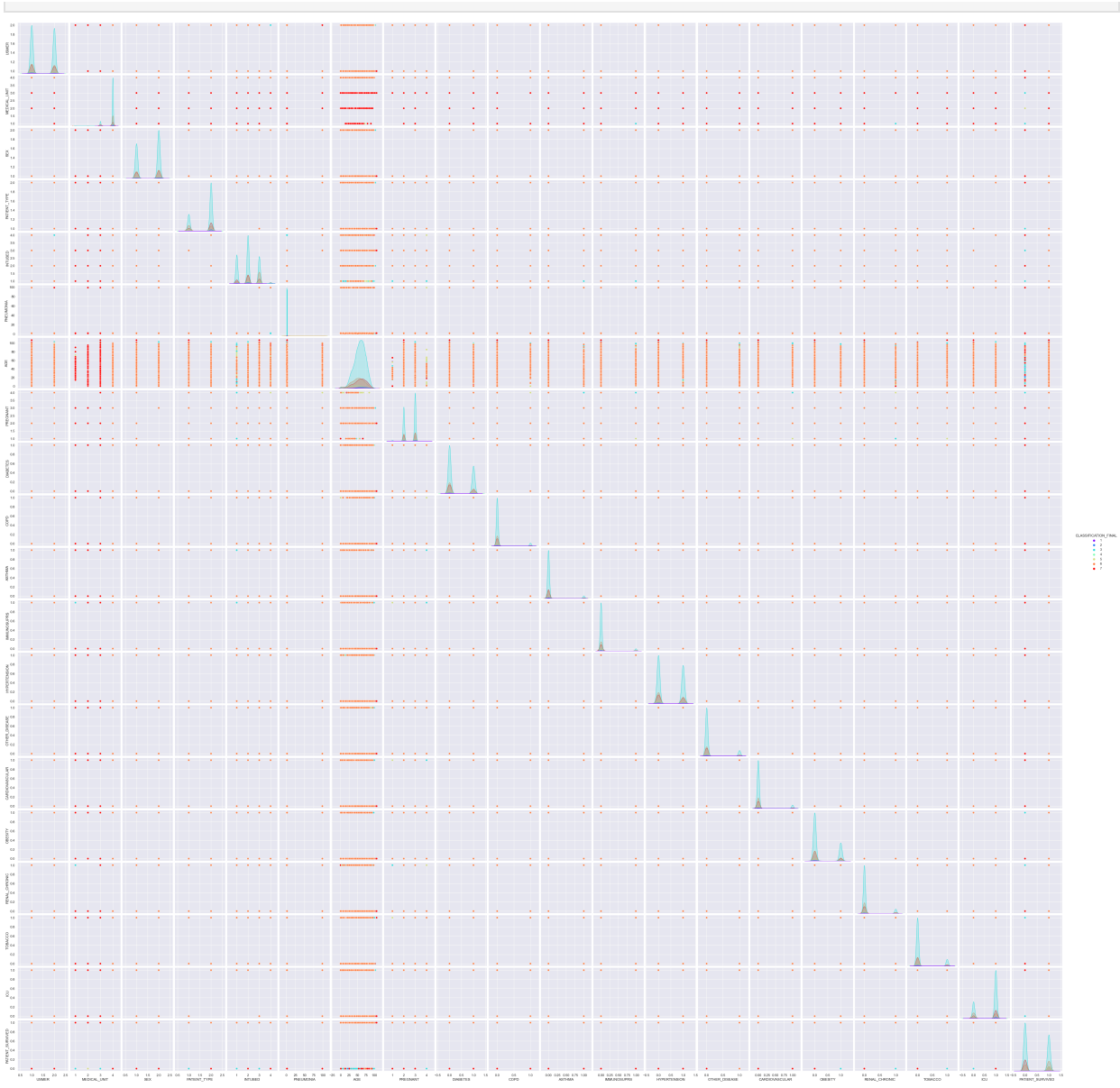
Out[34]:
```
<Axes: >
```



People got affected more in 2020 year with the COVID disease and hence had great loss in that year.

Visualization of multiclass data with Pairplot

In [35]:
```
ax = sns.pairplot(data,hue='CLASSIFICATION_FINAL',palette='rainbow');
sns.set_context("paper", rc={"font.size":12,"axes.titlesize":12,"axes.labelsize":10})
plt.show()
```



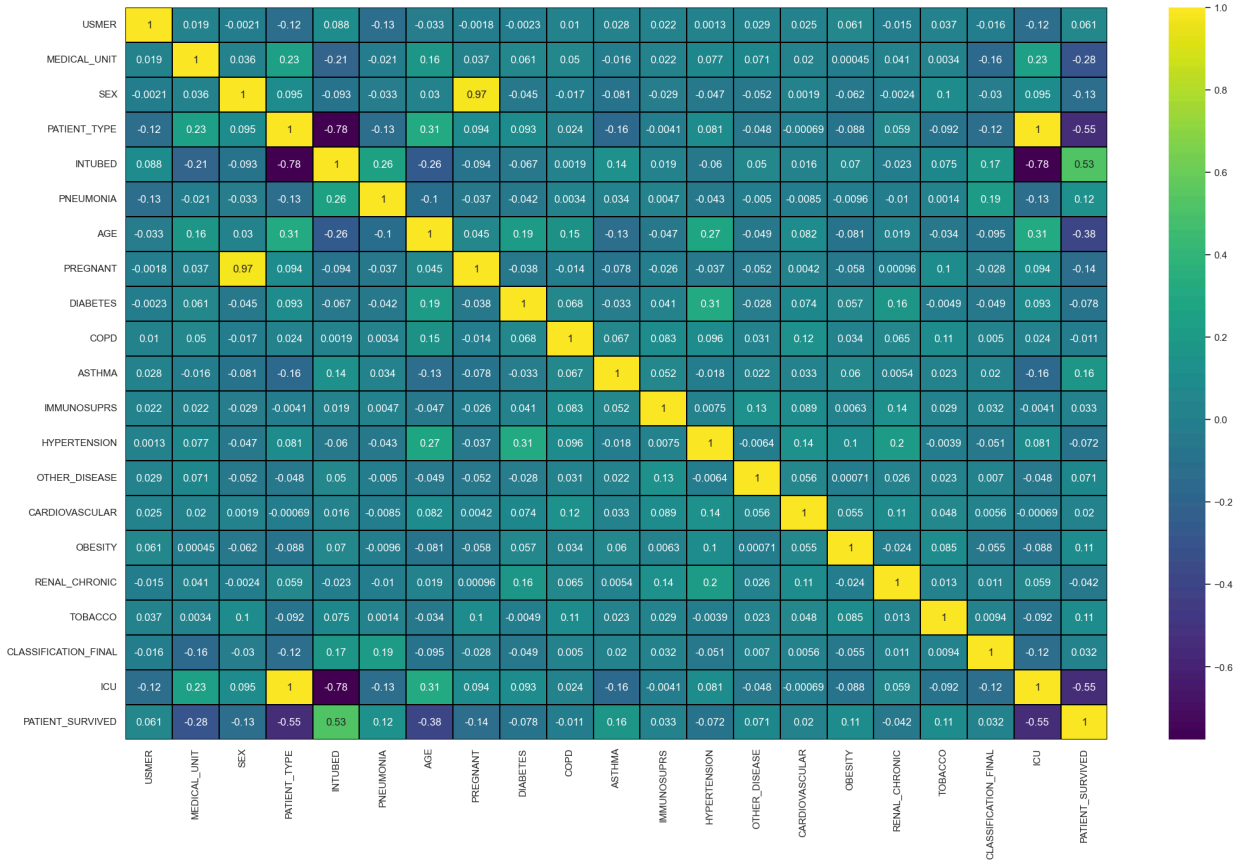
By visualizing relationships between features in the multiclass classification dataset, we can gain insights into how different features contribute to the classification task and identify any patterns or correlations that exist within the data.

Here, Seaborn's pairplot is used with a hue parameter to distinguish between classes. The hue parameter is set to the column containing the class labels, which will color the data points based on their class.

- A clear separation of outcome column classes-3, 6 and 7 indicates that these features are informative and can help distinguish between different classes.
- The clear separation between the KDE curves for class-3, suggests that the feature value for the class provides good discrimination.
- Conversely, overlapping KDE curves indicate that the feature values may not be sufficient to distinguish between certain classes.

Correlation analysis and visualization through Heatmap

In [36]:
```
sns.set(rc={'figure.figsize':(25,15)})
sns.heatmap(data.corr(),annot=True,cmap='viridis',linewidths=0.3, linecolor='black');
```



From the above plot,

- Patient type and ICU -Strongly positively correlated,
- Sex and Age - Strongly positively correlated,
- Intubed and Patient Survived, Age and ICU, Diabetes and Hypertension, Patient type and Medical Unit - Positively correlated,
- Intubed and ICU - Negatively correlated,
- Patient type and Intubed - Negatively correlated,
- ICU and Patient Survived - Negatively correlated,
- Patient type and Patient Survived - Negatively correlated
- Variables are independent - value 0 - No correlation

In [37]:
```
data.columns
```

Out[37]:
```
Index(['USMER', 'MEDICAL_UNIT', 'SEX', 'PATIENT_TYPE', 'DATE_DIED', 'INTUBED',
       'PNEUMONIA', 'AGE', 'PREGNANT', 'DIABETES', 'COPD', 'ASTHMA',
       'IMMUNOSUPRS', 'HYPERTENSION', 'OTHER_DISEASE', 'CARDIOVASCULAR',
       'OBESITY', 'RENAL_CHRONIC', 'TOBACCO', 'CLASSIFICATION_FINAL', 'ICU',
       'PATIENT_SURVIVED'],
      dtype='object')
```

In [38]:
```
data.drop(['DATE_DIED'],axis=1,inplace=True)
```

One Hot Encoding

In [39]:
```
n_data = pd.get_dummies(data, columns = ['USMER','MEDICAL_UNIT','SEX','PATIENT_TYPE','
                                         'PREGNANT','DIABETES','COPD','ASTHMA','IMMUNO
```

```
                                        'HYPERTENSION','OTHER_DISEASE','CARDIOVASCULA
                                        'RENAL_CHRONIC','TOBACCO','CLASSIFICATION_FIN
                                        drop_first=True)
n_data.head()
```

Out[39]:

	AGE	USMER_2	MEDICAL_UNIT_2	MEDICAL_UNIT_3	MEDICAL_UNIT_4	SEX_2	PATIENT_TYPE_2	IN
0	65	1	0	0	0	0	0	
1	72	1	0	0	0	1	0	
2	55	1	0	0	0	1	1	
3	53	1	0	0	0	0	0	
4	68	1	0	0	0	1	0	

5 rows × 33 columns

Model building

In [40]:
```
x = data.drop('CLASSIFICATION_FINAL',axis=1)
y = data['CLASSIFICATION_FINAL']
```

Train Test Split

In [41]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=40)
```

Data Normalization

In [42]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
x_train = ss.fit_transform(x_train)
x_test = ss.transform(x_test)
```

Training the model using different classification algorithms

In [43]:
```
from sklearn.metrics import multilabel_confusion_matrix, classification_report, accura
```

Decision Tree

In [44]:
```
from sklearn.tree import DecisionTreeClassifier
model1 = DecisionTreeClassifier()
```

In [45]:
```
model1.fit(x_train,y_train)
y_pred1 = model1.predict(x_test)
```

In [46]:
```
model1.score(x_train,y_train)
```

Out[46]:
```
0.8252313481549745
```

In [47]:
```
model1.score(x_test,y_test)
```

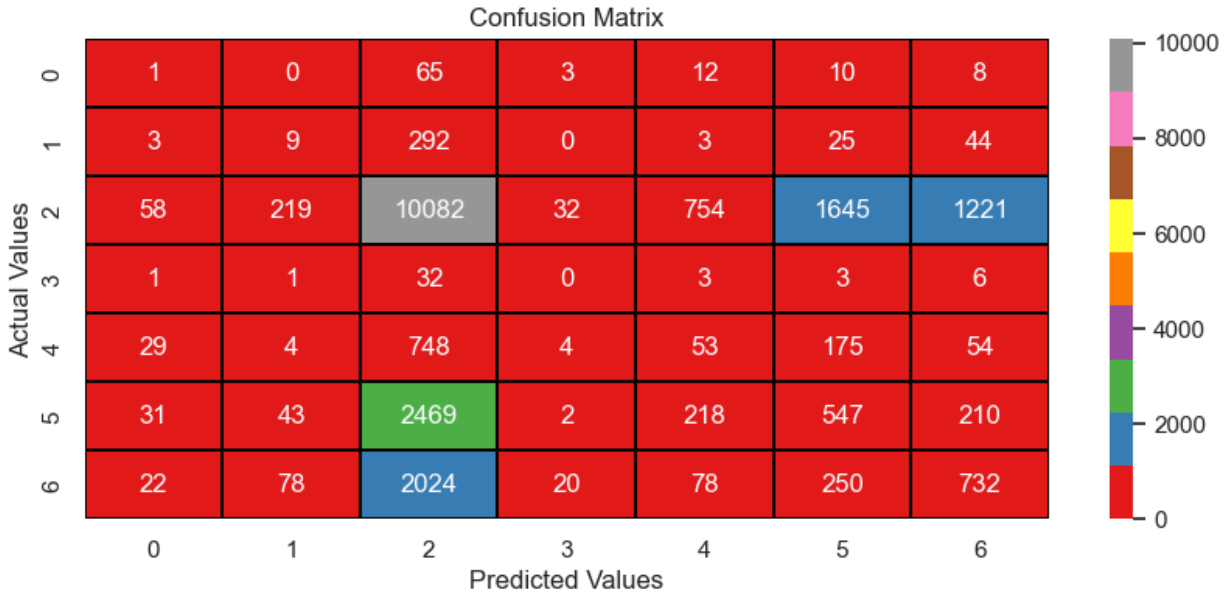
Out[47]:
```
0.5117591721542804
```

In [48]:
```
print('Decision Tree results')
print('---------------------------')
print('Accuracy is: ',accuracy_score(y_test,y_pred1))
print('\n')
print('Confusion Matrix is: ')
print(multilabel_confusion_matrix(y_test,y_pred1))
print('\n')
print('Classification Report is: ')
print(classification_report(y_test,y_pred1))
```

```
Decision Tree results
--------------------------
Accuracy is:  0.5117591721542804


Confusion Matrix is:
[[[22080   144]
  [   98     1]]

 [[21602   345]
  [  367     9]]

 [[ 2682  5630]
  [ 3929 10082]]

 [[22216    61]
  [   46     0]]

 [[20188  1068]
  [ 1014    53]]

 [[16695  2108]
  [ 2973   547]]

 [[17576  1543]
  [ 2472   732]]]


Classification Report is:
              precision    recall  f1-score   support

           1       0.01      0.01      0.01        99
           2       0.03      0.02      0.02       376
           3       0.64      0.72      0.68     14011
           4       0.00      0.00      0.00        46
           5       0.05      0.05      0.05      1067
           6       0.21      0.16      0.18      3520
           7       0.32      0.23      0.27      3204

    accuracy                           0.51     22323
   macro avg       0.18      0.17      0.17     22323
weighted avg       0.48      0.51      0.49     22323
```

In [49]:

```
plt.figure(figsize=(10,4))
sns.heatmap(confusion_matrix(y_test,y_pred1), annot=True,cmap='Set1',linewidths=0.3, l
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```



Performance evaluation and optimizing parameters using GridSearchCV:

In [50]:
```
from sklearn.model_selection import GridSearchCV
```

In [51]:
```
params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'criterion': ["gini", "entropy"]
}
```

In [52]:
```
gs_dt = GridSearchCV(model1, param_grid = params, cv=5, verbose=0)
gs_dt.fit(x_train, y_train)
```

Out[52]:

▸ **GridSearchCV**

▸ **estimator: DecisionTreeClassifier**

▸ DecisionTreeClassifier

In [53]:
```
gs_dt.best_params_
```

Out[53]:
```
{'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 100}
```

In [54]:
```
gs_dt.best_score_
```

Out[54]:
```
0.6562991741223391
```

In [55]:
```
gs_model = gs_dt.best_estimator_
```

In [56]:
```
gsy_pred = gs_model.predict(x_test)
```

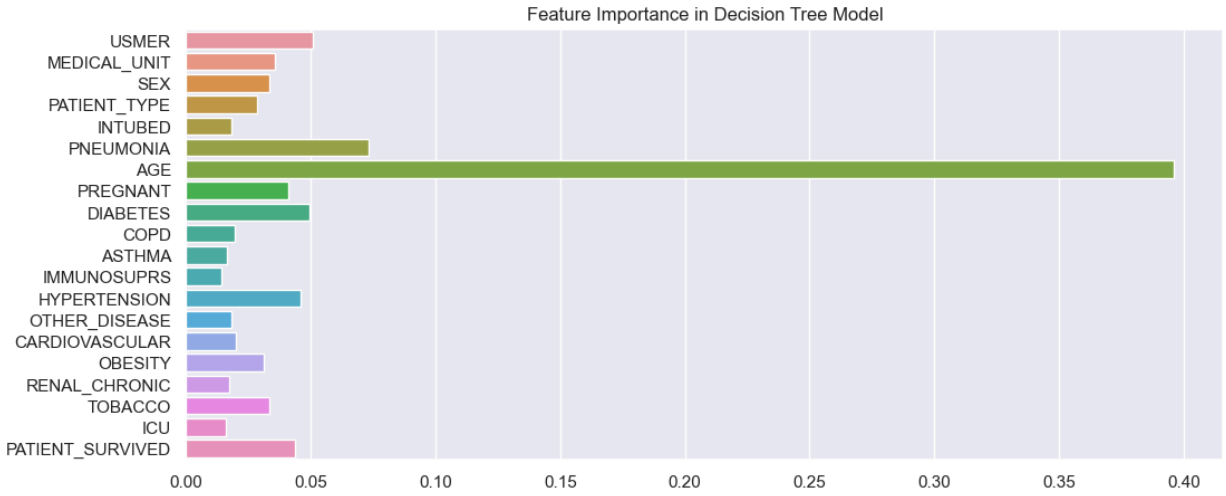
In [57]:
```
print('Tuned Decision Tree Classifier results')
print('--------------------------')
print('Accuracy is: ',accuracy_score(y_test,gsy_pred))
print('\n')
```

```
print('Confusion Matrix is: ')
print(multilabel_confusion_matrix(y_test,gsy_pred))
print('\n')
print('Classification Report is: ')
print(classification_report(y_test,gsy_pred))
```

```
Tuned Decision Tree Classifier results
--------------------------
Accuracy is:  0.6536307843927788


Confusion Matrix is:
[[[22224     0]
  [   99     0]]

 [[21947     0]
  [  376     0]]

 [[ 1609  6703]
  [  606 13405]]

 [[22277     0]
  [   46     0]]

 [[21221    35]
  [ 1021    46]]

 [[18697   106]
  [ 3156   364]]

 [[18231   888]
  [ 2428   776]]]


Classification Report is:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        99
           2       0.00      0.00      0.00       376
           3       0.67      0.96      0.79     14011
           4       0.00      0.00      0.00        46
           5       0.57      0.04      0.08      1067
           6       0.77      0.10      0.18      3520
           7       0.47      0.24      0.32      3204

    accuracy                           0.65     22323
   macro avg       0.35      0.19      0.20     22323
weighted avg       0.63      0.65      0.57     22323
```

In [58]:

```
plt.figure(figsize=(12,5))
sns.barplot(y=x.columns, x=model1.feature_importances_)
plt.title("Feature Importance in Decision Tree Model");
```



K-Nearest Neighbor

In [59]:
```
from sklearn.neighbors import KNeighborsClassifier
model2 = KNeighborsClassifier(n_neighbors=11)
```

In [60]:
```
model2.fit(x_train,y_train)
y_pred2 = model2.predict(x_test)
```

In [61]:
```
model2.score(x_train,y_train)
```

Out[61]:
```
0.6717352071573935
```

In [62]:
```
model2.score(x_test,y_test)
```

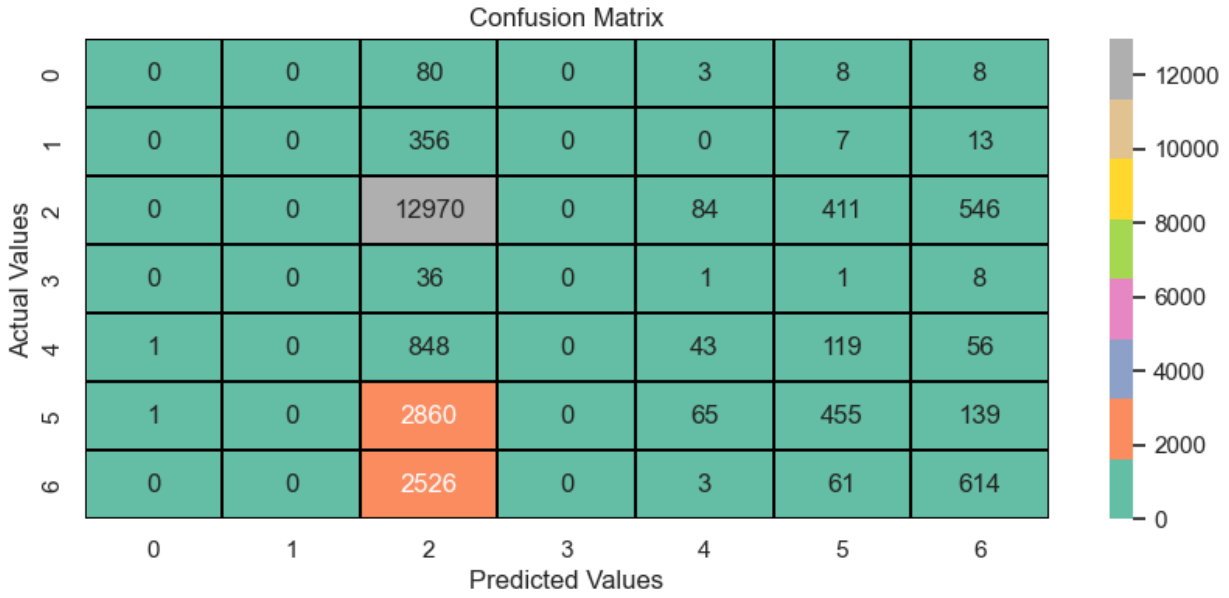
Out[62]:
```
0.6308291896250504
```

In [63]:
```
print('K Nearest Neighbors results')
print('---------------------------')
print('Accuracy is: ',accuracy_score(y_test,y_pred2))
print('\n')
print('Confusion Matrix is: ')
print(multilabel_confusion_matrix(y_test,y_pred2))
print('\n')
print('Classification Report is: ')
print(classification_report(y_test,y_pred2))
```

```
K Nearest Neighbors results
---------------------------
Accuracy is:  0.6308291896250504


Confusion Matrix is:
[[[22222     2]
  [   99     0]]

 [[21947     0]
  [  376     0]]

 [[ 1606  6706]
  [ 1041 12970]]

 [[22277     0]
  [   46     0]]

 [[21100   156]
  [ 1024    43]]

 [[18196   607]
  [ 3065   455]]

 [[18349   770]
  [ 2590   614]]]


Classification Report is:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        99
           2       0.00      0.00      0.00       376
           3       0.66      0.93      0.77     14011
           4       0.00      0.00      0.00        46
           5       0.22      0.04      0.07      1067
           6       0.43      0.13      0.20      3520
           7       0.44      0.19      0.27      3204

    accuracy                           0.63     22323
   macro avg       0.25      0.18      0.19     22323
weighted avg       0.56      0.63      0.56     22323
```

In [64]:

```
plt.figure(figsize=(10,4))
sns.heatmap(confusion_matrix(y_test,y_pred2), annot=True,cmap='Set2',linewidths=0.3, l
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```



Performance evaluation and optimizing parameters using GridSearchCV:

In [65]:
```
grid_params = { 'n_neighbors' : [5,7,9,11,13,15],
                'weights' : ['uniform','distance'],
                'metric' : ['minkowski','euclidean','manhattan']}
```

In [66]:
```
gs = GridSearchCV(KNeighborsClassifier(), grid_params, verbose = 1, cv=3, n_jobs = -1)
```

In [67]:
```
g_res = gs.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
```

In [68]:
```
g_res.best_score_
```

Out[68]:
```
0.6400376300733402
```

In [69]:
```
g_res.best_params_
```

Out[69]:
```
{'metric': 'manhattan', 'n_neighbors': 15, 'weights': 'uniform'}
```

In [70]:
```
gs_knnmodel = g_res.best_estimator_
```

In [71]:
```
gsknny_pred = gs_knnmodel.predict(x_test)
```

In [72]:
```
print('Tuned KNN results')
print('--------------------------')
print('Accuracy is: ',accuracy_score(y_test,gsknny_pred))
print('\n')
print('Confusion Matrix is: ')
print(multilabel_confusion_matrix(y_test,gsknny_pred))
print('\n')
print('Classification Report is: ')
print(classification_report(y_test,gsknny_pred))
```

```
Tuned KNN results
--------------------------
Accuracy is:  0.6369215607221251


Confusion Matrix is:
[[[22222     2]
  [   99     0]]

 [[21945     2]
  [  376     0]]

 [[ 1571  6741]
  [  901 13110]]

 [[22275     2]
  [   46     0]]

 [[21152   104]
  [ 1017    50]]

 [[18260   543]
  [ 3064   456]]

 [[18408   711]
  [ 2602   602]]]


Classification Report is:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        99
           2       0.00      0.00      0.00       376
           3       0.66      0.94      0.77     14011
           4       0.00      0.00      0.00        46
           5       0.32      0.05      0.08      1067
           6       0.46      0.13      0.20      3520
           7       0.46      0.19      0.27      3204

    accuracy                           0.64     22323
   macro avg       0.27      0.19      0.19     22323
weighted avg       0.57      0.64      0.56     22323
```

Random Forest Classifier

In [73]:
```
from sklearn.ensemble import RandomForestClassifier
model3 = RandomForestClassifier(n_estimators=150)
```

In [74]:
```
model3.fit(x_train,y_train)
y_pred3 = model3.predict(x_test)
```

In [75]:
```
model3.score(x_train,y_train)
```

Out[75]:
```
0.8252313481549745
```

In [76]:
```
model3.score(x_test,y_test)
```

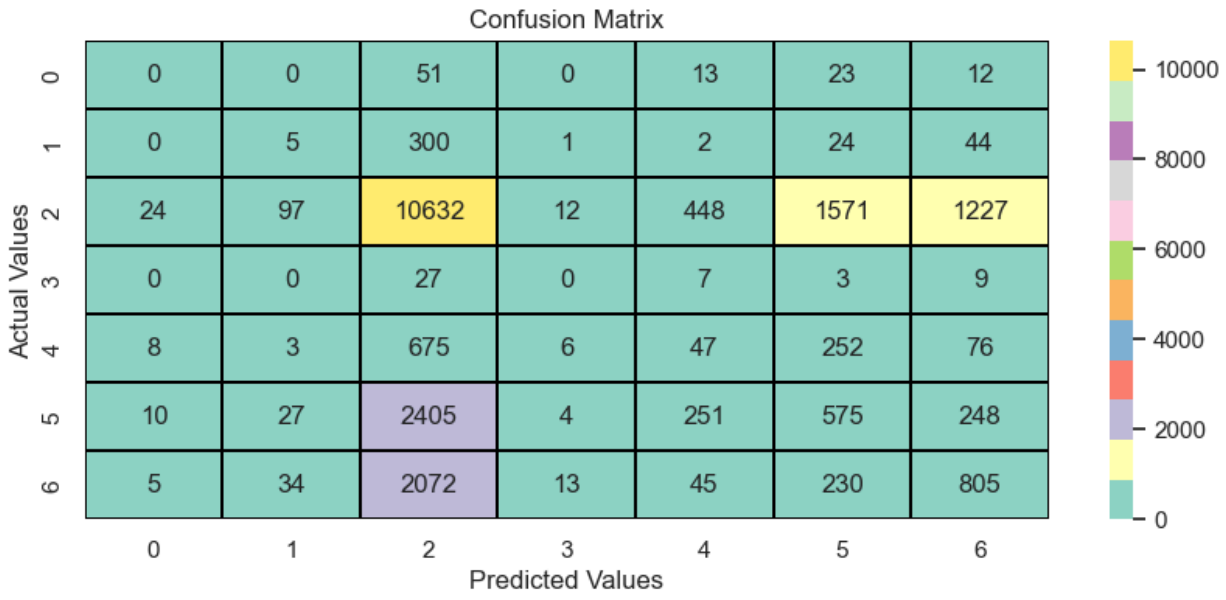
Out[76]:
```
0.5404291537875734
```

In [77]:
```
print('Random Forest Classifier results')
print('--------------------------')
print('Accuracy is: ',accuracy_score(y_test,y_pred3))
print('\n')
print('Confusion Matrix is: ')
print(multilabel_confusion_matrix(y_test,y_pred3))
print('\n')
print('Classification Report is: ')
print(classification_report(y_test,y_pred3))
```

```
Random Forest Classifier results
--------------------------
Accuracy is:  0.5404291537875734


Confusion Matrix is:
[[[22177    47]
  [   99     0]]

 [[21786   161]
  [  371     5]]

 [[ 2782  5530]
  [ 3379 10632]]

 [[22241    36]
  [   46     0]]

 [[20490   766]
  [ 1020    47]]

 [[16700  2103]
  [ 2945   575]]

 [[17503  1616]
  [ 2399   805]]]


Classification Report is:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        99
           2       0.03      0.01      0.02       376
           3       0.66      0.76      0.70     14011
           4       0.00      0.00      0.00        46
           5       0.06      0.04      0.05      1067
           6       0.21      0.16      0.19      3520
           7       0.33      0.25      0.29      3204

    accuracy                           0.54     22323
   macro avg       0.18      0.18      0.18     22323
weighted avg       0.50      0.54      0.52     22323
```

In [78]:
```
plt.figure(figsize=(10,4))
sns.heatmap(confusion_matrix(y_test,y_pred3), annot=True,cmap='Set3',linewidths=0.3, l
plt.title('Confusion Matrix')
```

```
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```



Performance evaluation and optimizing parameters using GridSearchCV:

In [79]:
```
param_grid = {
    'bootstrap': [True],
    'max_depth': [5,10,15,20],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [10, 20, 30, 100]
}
```

In [80]:
```
grid_search = GridSearchCV(estimator = model3, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)
```

In [81]:
```
grid_search.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 288 candidates, totalling 864 fits
```

Out[81]:

▸ **GridSearchCV**

▸ **estimator: RandomForestClassifier**

▸ RandomForestClassifier

In [82]:
```
grid_search.best_score_
```

Out[82]:
```
0.6568943670084092
```

In [83]:
```
grid_search.best_params_
```

Out[83]:
```
{'bootstrap': True,
 'max_depth': 15,
 'max_features': 3,
 'min_samples_leaf': 4,
 'min_samples_split': 12,
 'n_estimators': 100}
```

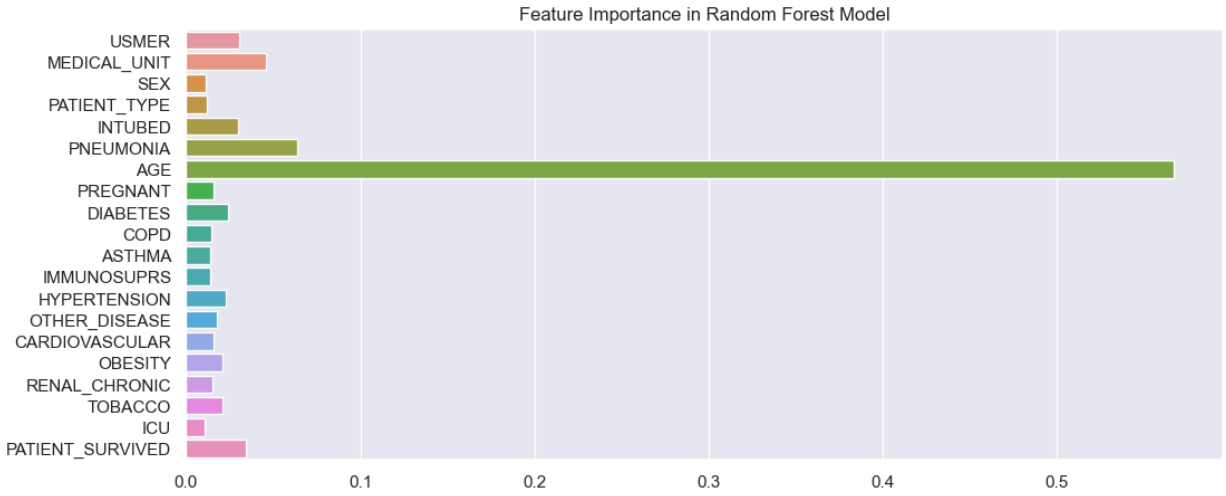
In [84]:
```
grid_rfmodel = grid_search.best_estimator_
```

In [85]:
```
gridrfy_pred = grid_rfmodel.predict(x_test)
```

In [86]:
```
print('Tuned Random Forest Classifier results')
print('---------------------------')
print('Accuracy is: ',accuracy_score(y_test,gridrfy_pred))
print('\n')
print('Confusion Matrix is: ')
print(multilabel_confusion_matrix(y_test,gridrfy_pred))
print('\n')
print('Classification Report is: ')
print(classification_report(y_test,gridrfy_pred))
```

```
Tuned Random Forest Classifier results
---------------------------
Accuracy is:  0.6544371276262151


Confusion Matrix is:
[[[22224     0]
  [   99     0]]

 [[21947     0]
  [  376     0]]

 [[ 1315  6997]
  [  404 13607]]

 [[22277     0]
  [   46     0]]

 [[21235    21]
  [ 1022    45]]

 [[18689   114]
  [ 3154   366]]

 [[18537   582]
  [ 2613   591]]]


Classification Report is:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        99
           2       0.00      0.00      0.00       376
           3       0.66      0.97      0.79     14011
           4       0.00      0.00      0.00        46
           5       0.68      0.04      0.08      1067
           6       0.76      0.10      0.18      3520
           7       0.50      0.18      0.27      3204

    accuracy                           0.65     22323
   macro avg       0.37      0.19      0.19     22323
weighted avg       0.64      0.65      0.56     22323
```

In [87]:
```
plt.figure(figsize=(12,5))
sns.barplot(y=x.columns, x=model3.feature_importances_)
plt.title("Feature Importance in Random Forest Model");
```



Ensemble Learning --> Boosting --> Adaptive Boosting:

In [88]:
```
from sklearn.ensemble import AdaBoostClassifier
model4 = AdaBoostClassifier(n_estimators=2, learning_rate=1)
```

In [89]:
```
model4.fit(x_train,y_train)
y_pred4 = model4.predict(x_test)
```

In [90]:
```
model4.score(x_train,y_train)
```

Out[90]:
```
0.6429750796759206
```

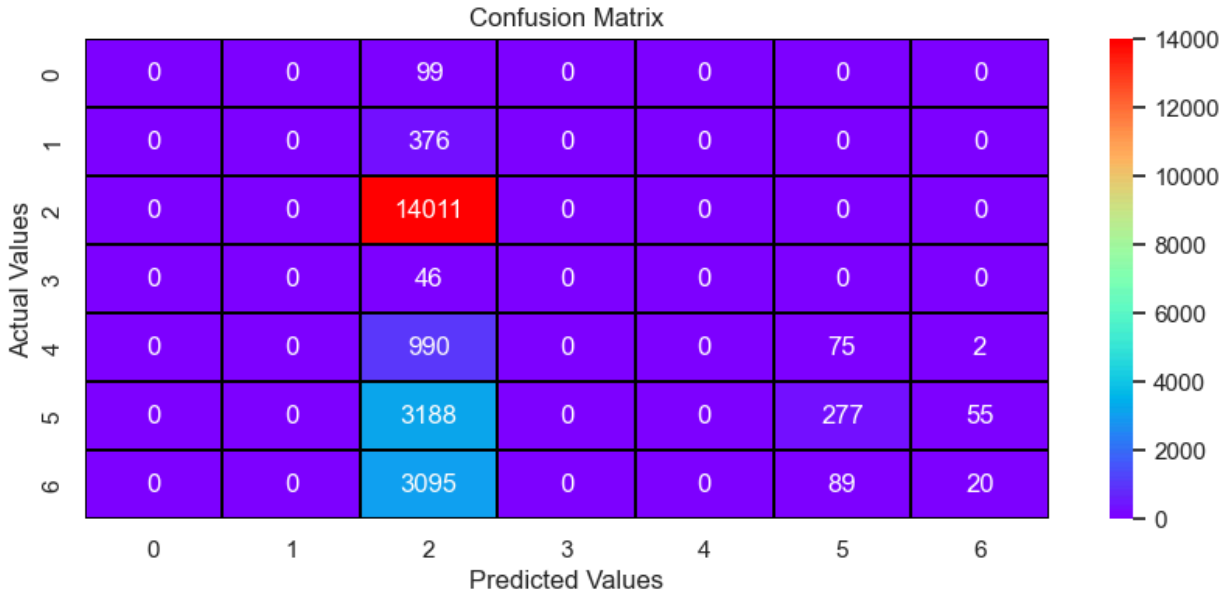
In [91]:
```
model4.score(x_test,y_test)
```

Out[91]:
```
0.640953276889307
```

In [92]:
```
print('AdaBoost Classifier results')
print('---------------------------')
print('Accuracy is: ',accuracy_score(y_test,y_pred4))
print('\n')
print('Confusion Matrix is: ')
print(multilabel_confusion_matrix(y_test,y_pred4))
print('\n')
print('Classification Report is: ')
print(classification_report(y_test,y_pred4))
```

```
AdaBoost Classifier results
---------------------------
Accuracy is:  0.640953276889307


Confusion Matrix is:
[[[22224     0]
  [   99     0]]

 [[21947     0]
  [  376     0]]

 [[  518  7794]
  [    0 14011]]

 [[22277     0]
  [   46     0]]

 [[21256     0]
  [ 1067     0]]

 [[18639   164]
  [ 3243   277]]

 [[19062    57]
  [ 3184    20]]]


Classification Report is:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        99
           2       0.00      0.00      0.00       376
           3       0.64      1.00      0.78     14011
           4       0.00      0.00      0.00        46
           5       0.00      0.00      0.00      1067
           6       0.63      0.08      0.14      3520
           7       0.26      0.01      0.01      3204

    accuracy                           0.64     22323
   macro avg       0.22      0.15      0.13     22323
weighted avg       0.54      0.64      0.51     22323
```

In [93]:
```
plt.figure(figsize=(10,4))
sns.heatmap(confusion_matrix(y_test,y_pred4), annot=True,cmap='rainbow',linewidths=0.3
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```



Performance evaluation and optimizing parameters using GridSearch and cross_val_score:

In [94]:
```
parameters = {
    'n_estimators': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 20, 30]
}
```

In [95]:
```
grid = GridSearchCV(model4, parameters, cv=5, verbose=0)
```

In [96]:
```
grid.fit(x_train,y_train)
```

Out[96]:

GridSearchCV

estimator: AdaBoostClassifier

AdaBoostClassifier

In [97]:
```
grid.best_score_
```

Out[97]:
```
0.6442038228744813
```

In [98]:
```
grid.best_params_
```

Out[98]:
```
{'n_estimators': 1}
```

In [99]:
```
grid_admodel = grid.best_estimator_
```

In [100…
```
gridady_pred = grid_admodel.predict(x_test)
```

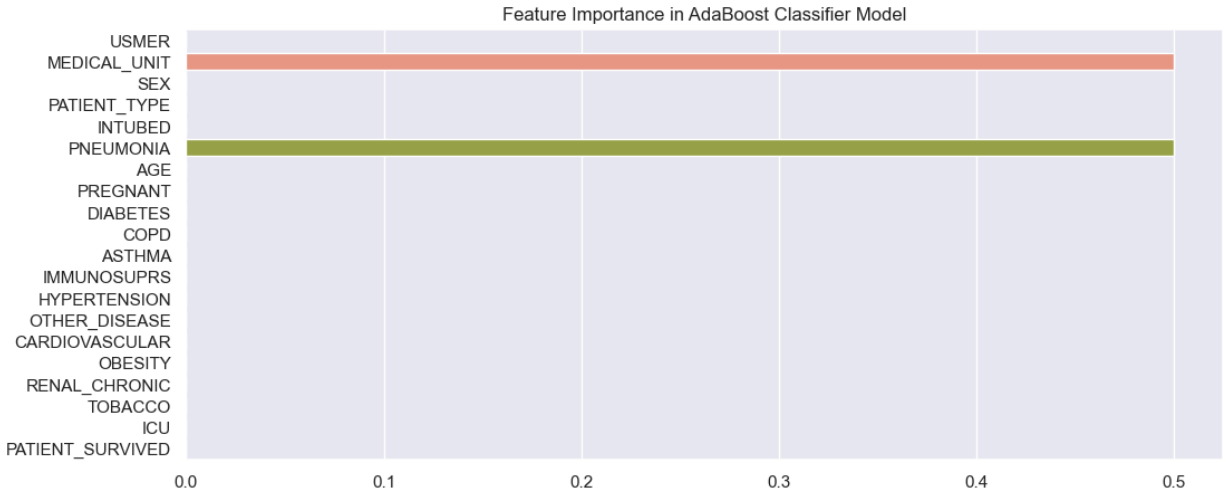
In [101…
```
print('Tuned Adaboost Classifier results')
print('---------------------------')
print('Accuracy is: ',accuracy_score(y_test,gridady_pred))
print('\n')
print('Confusion Matrix is: ')
print(multilabel_confusion_matrix(y_test,gridady_pred))
```

```
print('\n')
print('Classification Report is: ')
print(classification_report(y_test,gridady_pred))
```

```
Tuned Adaboost Classifier results
--------------------------
Accuracy is:  0.6425211665098777


Confusion Matrix is:
[[[22224     0]
  [   99     0]]

 [[21947     0]
  [  376     0]]

 [[  518  7794]
  [    0 14011]]

 [[22277     0]
  [   46     0]]

 [[21256     0]
  [ 1067     0]]

 [[18617   186]
  [ 3188   332]]

 [[19119     0]
  [ 3204     0]]]


Classification Report is:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        99
           2       0.00      0.00      0.00       376
           3       0.64      1.00      0.78     14011
           4       0.00      0.00      0.00        46
           5       0.00      0.00      0.00      1067
           6       0.64      0.09      0.16      3520
           7       0.00      0.00      0.00      3204

    accuracy                           0.64     22323
   macro avg       0.18      0.16      0.14     22323
weighted avg       0.50      0.64      0.52     22323
```

In [102…

```
plt.figure(figsize=(12,5))
sns.barplot(y=x.columns, x=model4.feature_importances_)
plt.title("Feature Importance in AdaBoost Classifier Model");
```



Accuracy scores from all models

In [103…

```
print("Accuracy Scores:\n")

print("Decision Tree Classifier: ", accuracy_score(y_test,y_pred1) * 100)
print("Tuned Decision Tree Classifier: ", accuracy_score(y_test,gsy_pred) * 100)
print("KNN: ", accuracy_score(y_test,y_pred2) * 100)
print("Tuned KNN:", accuracy_score(y_test,gsknny_pred) * 100)
print("Random Forest: ", accuracy_score(y_test,y_pred3) * 100)
print("Tuned Random Forest: ", accuracy_score(y_test,gridrfy_pred) * 100)
print("Adaboost Classifier: ", accuracy_score(y_test, y_pred4) * 100)
print("Tuned Adaboost Classifier: ", accuracy_score(y_test,gridady_pred) * 100)
```

```
Accuracy Scores:

Decision Tree Classifier:  51.175917215428036
Tuned Decision Tree Classifier:  65.36307843927787
KNN:  63.08291896250504
Tuned KNN: 63.69215607221251
Random Forest:  54.04291537875734
Tuned Random Forest:  65.44371276262152
Adaboost Classifier:  64.09532768893071
Tuned Adaboost Classifier:  64.25211665098777
```

From above scores,

- Adaboost Classifier accuracy score is around 64% before tuning and increased 0.15% after tuning.
- KNN classifier accuracy score is around 63% before tuning and increased .50 after tuning.
- Decision Tree Classifier, Random Forest give similar Accuracy Scores. After tuning, accuracy scores of each models were increased to 65% and above.
- This means that out of the total data points 64% are correct predicted values. If we consider this value alone then we can choose either Adaboost Classifier or KNN Classifier; or other models like Decision Tree and Random Forest models after tuning.

Conclusion

- Age, Pneumonia and Medical Unit seems to be the features that are most closely related to output column.
- People with respiratory problems like Pneumonia and COPD are easily affected by the COVID disease.
- Age plays an important role in the disease degree and plays major role in the recovery process.
- Patient Survived plays a part in determining the death from the output column(disease degree).
- Patient with Asthma has high risk of Pneumonia and hence related to output column.
- Patient using tobacco are immunosuppressed and hence easily affected with the disease.
- Patient with COVID-19 and other pre-existing medical conditions can significantly impact their prognosis, treatment options, and overall management. Hypertension, Diabetes, Cardiovasular and Other diseases are closely related and hence patient needed ventilation and specialized care treatments.
- Patient type and ICU, Sex and Age are Strongly positively correlated; Intubed and Patient Survived, Age and ICU, Patient type and Medical Unit are Positively correlated.
- Patient type and Intubed, ICU and Patient Survived, Patient type and Patient Survived are Negatively correlated.
- Classes - 3, 6, 7 indicates that these features are informative and can help to distinguish between different classes.
- Machine Learning models can prove to be a time saving factor in predicting the event of disease degree and can help the doctors take additional precautionary measures with critical care.
- While accuracy is a commonly used metric, it might not always provide a complete picture, especially in situations when different classes have different levels of importance.
- The models with high precision and high recall can accurately identify positive cases while minimizing false positives and capturing most of the true positive cases. Class 3 in outcome column have high precision and high recall which is desirable as it indicates that the model can accurately identify positive instances of that class while minimizing both false positives and false negatives.
- An F1 score of 0.75 and above, in Class 3 of all the above models of multiclass classification problem using a healthcare dataset indicates a relatively good performance of the model in classifying instances across multiple classes.

Data Reporting

Tableau Dashboard

The Dashboard is published on Tableau Public under the link :

Creating a dashboard in tableau by choosing appropriate chart types and metrics:

https://public.tableau.com/app/profile/swarnalatha.sethuraman/viz/COVID19_17071672255650/CO\
publish=yes