Machine Learning Model for Predicting COVID-19 Patient Survival

The aim of this machine learning model is to predict whether a COVID-19 patient will survive or not based on their current symptoms, status, and medical history. The goal is to achieve an accuracy of 90% or higher.

Dataset Information:

```
The dataset contains 21 features.
It includes information for 1,048,575 unique patients.
In the Boolean features, the value 1 indicates "yes," and 2 indicates "no."
```

Features:

```
usmer: Indicates whether the patient received medical treatment at the first,
second, or third level.
medical unit: Type of institution within the National Health System that
provided the care.
sex: Gender of the patient.
patient type: Indicates whether the patient returned home or was hospitalized.
date died: If the patient died, it indicates the date of death; otherwise, it
is set to "9999-99-99."
intubed: Indicates whether the patient was connected to a ventilator.
pneumonia: Indicates whether the patient already had air sac inflammation.
age: Age of the patient.
pregnant: Indicates whether the patient is pregnant.
diabetes: Indicates whether the patient has diabetes.
copd: Indicates whether the patient has Chronic Obstructive Pulmonary Disease
(COPD).
asthma: Indicates whether the patient has asthma.
inmsupr: Indicates whether the patient is immunosuppressed.
hypertension: Indicates whether the patient has hypertension.
other disease: Indicates whether the patient has other diseases.
cardiovascular: Indicates whether the patient has heart or blood vessel-related
diseases.
obesity: Indicates whether the patient is obese.
renal chronic: Indicates whether the patient has chronic renal disease.
tobacco: Indicates whether the patient is a tobacco user.
classification: COVID-19 test findings. Values 1-3 indicate different degrees
of COVID-19 diagnosis, while 4 or higher means the patient is not a carrier or
the test is inconclusive.
icu: Indicates whether the patient had been admitted to an Intensive Care Unit
(ICU).
```

In [264…

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import GridSearchCV
from imblearn.under_sampling import RandomUnderSampler
```

In [148…
```
df = pd.read_csv("../datasets/covid19.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	PNEUMONIA	AGE	PREGNANT	DIABE
0	2	1	female	returned home	03/05/2020	NaN	1.0	65	2.0	
1	2	1	male	returned home	03/06/2020	NaN	1.0	72	NaN	
2	2	1	male	hospitalization	09/06/2020	1.0	2.0	55	NaN	
3	2	1	female	returned home	12/06/2020	NaN	2.0	53	2.0	
4	2	1	male	returned home	21/06/2020	NaN	2.0	68	NaN	

5 rows × 21 columns

EDA - Exploratory Data Analysis

1. Technical EDA

In [4]:
```
df.dtypes
```

Out[4]:
```
USMER                     int64
MEDICAL_UNIT              int64
SEX                      object
PATIENT_TYPE             object
DATE_DIED                object
INTUBED                 float64
PNEUMONIA               float64
AGE                       int64
PREGNANT                float64
DIABETES                float64
COPD                    float64
ASTHMA                  float64
INMSUPR                 float64
HIPERTENSION            float64
OTHER_DISEASE           float64
CARDIOVASCULAR          float64
OBESITY                 float64
RENAL_CHRONIC           float64
TOBACCO                 float64
CLASIFFICATION_FINAL      int64
ICU                     float64
dtype: object
```

==> The majority of the values are in numeric format. Only sex, patient_type and date_died might require encoding

In [5]:
```
df.nunique()
```

Out[5]:
```
USMER                     2
MEDICAL_UNIT             13
SEX                       2
PATIENT_TYPE              2
DATE_DIED               401
INTUBED                   2
```

```
PNEUMONIA                 2
AGE                     121
PREGNANT                  2
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
```

==> sex and patient_type have only two unique values that will be easily encoded. Date_died will require special treatment

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
USMER                        0
MEDICAL_UNIT                 0
SEX                          0
PATIENT_TYPE                 0
DATE_DIED                    0
INTUBED                 855869
PNEUMONIA                16003
AGE                          0
PREGNANT                527265
DIABETES                  3338
COPD                      3003
ASTHMA                    2979
INMSUPR                   3404
HIPERTENSION              3104
OTHER_DISEASE             5045
CARDIOVASCULAR            3076
OBESITY                   3032
RENAL_CHRONIC             3006
TOBACCO                   3220
CLASIFFICATION_FINAL         0
ICU                     856032
dtype: int64
```

==> a lot of null values that will need to be resolved

In [7]:
```
alive_count = df[df['DATE_DIED']=='9999-99-99'].shape[0]
```

In [8]:
```
dead_count = df[df['DATE_DIED']!='9999-99-99'].shape[0]
```

In [9]:
```
total_count = df.shape[0]
```

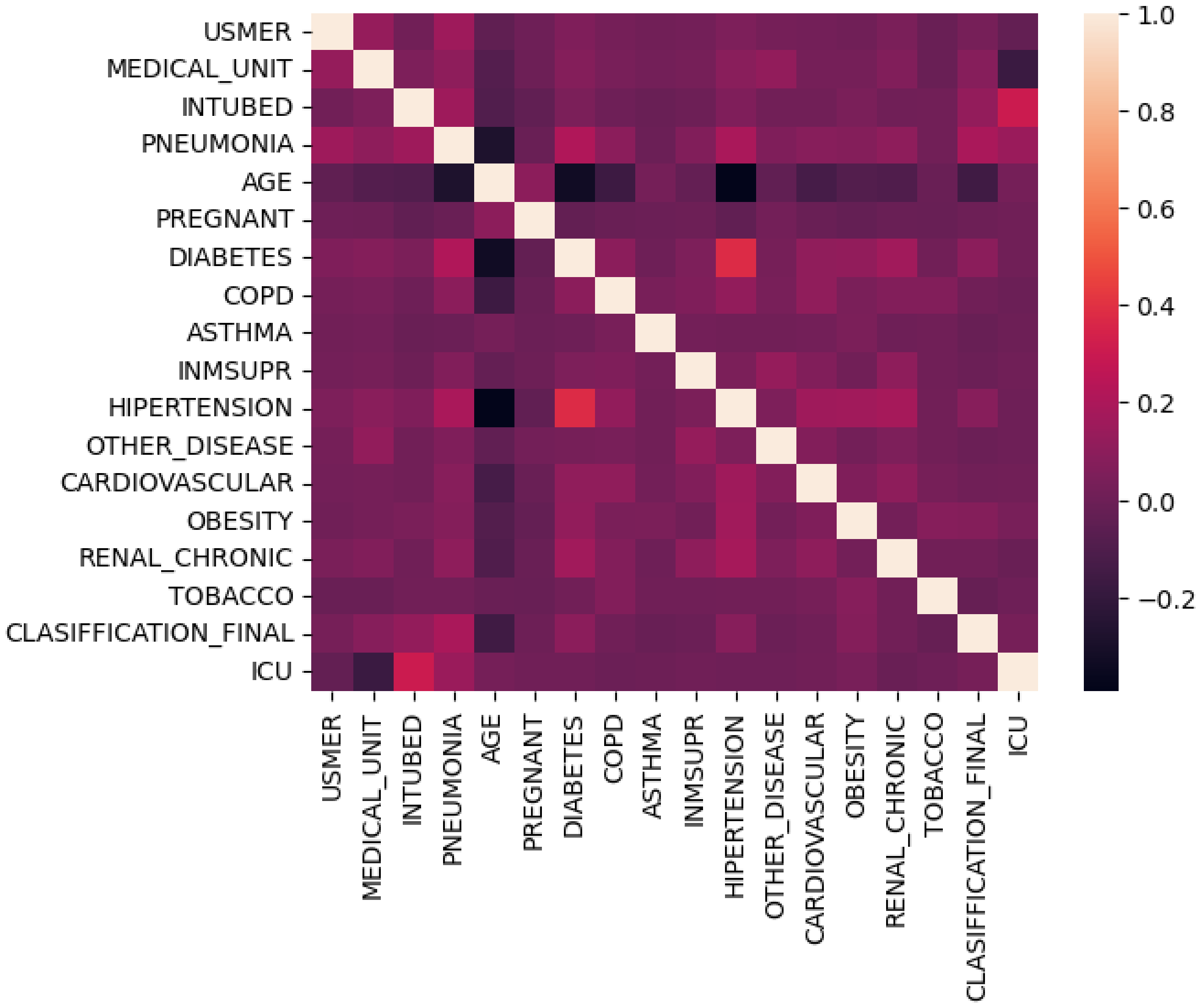
In [10]:
```
dead_count/total_count
```

Out[10]:
```
0.07337767923133777
```

==> Dataset seems to be unbalanced

In [11]:
```
sns.heatmap(df.corr())
```

Out[11]:
```
<AxesSubplot:>
```

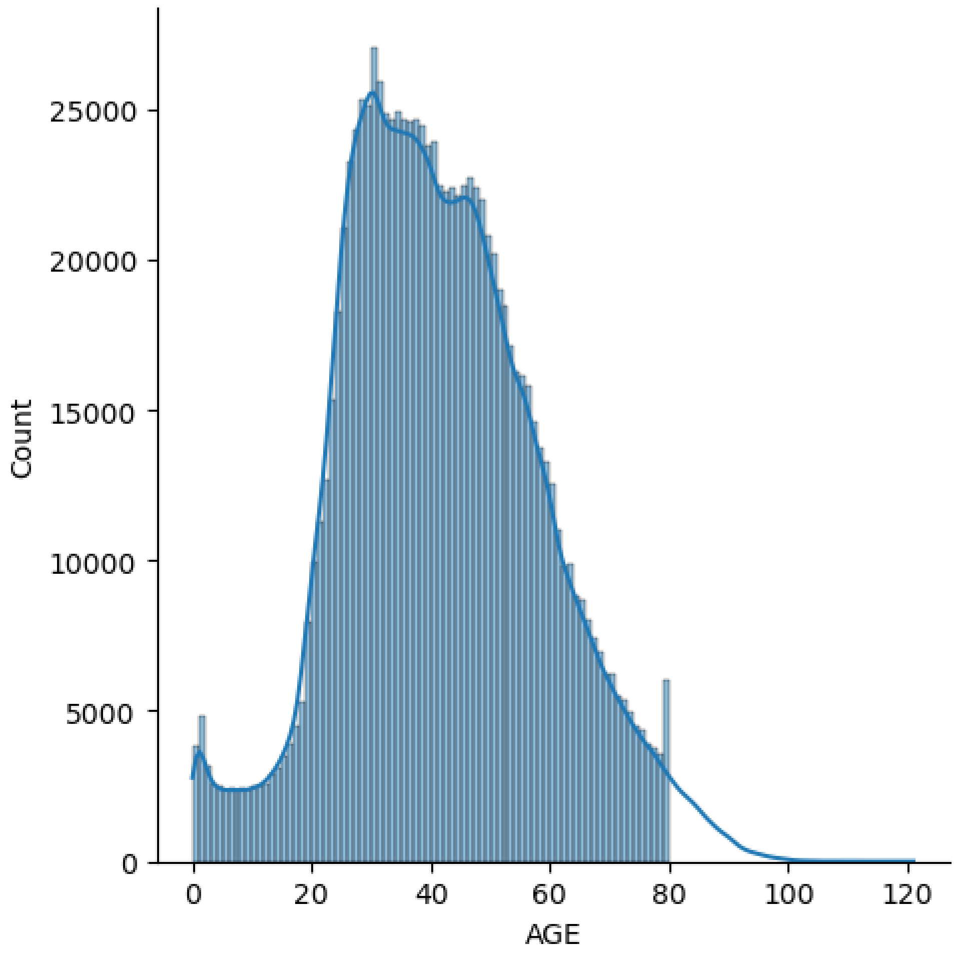


==> diabetes and hipertension shows average correlation

2. Interpretive EDA

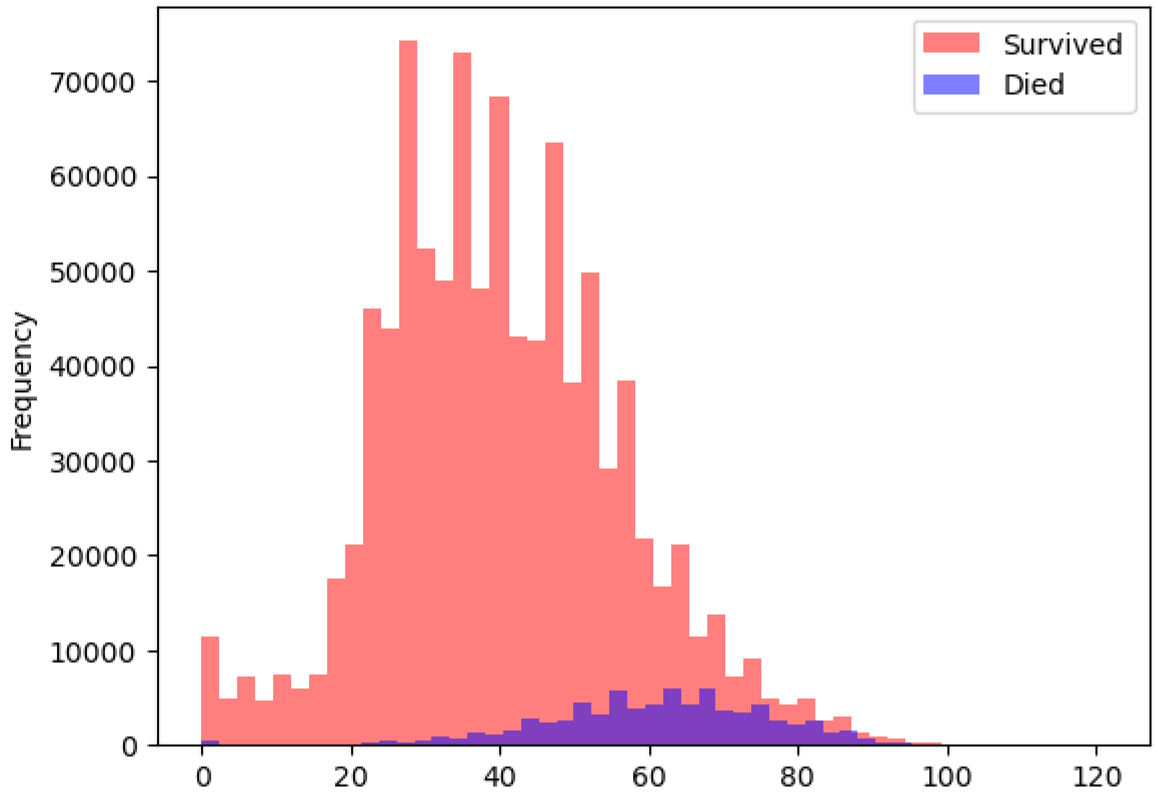
Checking age distribution

In [12]:
```
a = sns.displot(df['AGE'].dropna(), bins=range(0,81,1), kde=True)
plt.show()
```



Survival distribution across age

In [13]:
```
survived = df[df['DATE_DIED']=='9999-99-99']
died = df[df['DATE_DIED']!='9999-99-99']

survived['AGE'].plot.hist(alpha=0.5, color='red', bins=50) # alpha - transperancy, bins
died["AGE"].plot.hist(alpha=0.5, color='blue', bins=50)

plt.legend(['Survived', 'Died'])
plt.show()
```



```
In [14]: died["AGE"].describe()
```

```
Out[14]: count    76942.000000
         mean        61.068545
         std         15.366451
         min          0.000000
         25%         52.000000
         50%         62.000000
         75%         72.000000
         max        119.000000
         Name: AGE, dtype: float64
```

```
In [15]: survived["AGE"].describe()
```

```
Out[15]: count    971633.000000
         mean         40.267791
         std          16.063928
         min           0.000000
         25%          29.000000
         50%          39.000000
         75%          50.000000
         max         121.000000
         Name: AGE, dtype: float64
```

Insights

- Average age of of people who died is 61 years.
- Average age of people who survived is 40 years.
- 75% of survived people had less than 50 years.
- 75% of people who died had less than 72 years.

Some data processing to plot distribution of the target value

```
In [149…
for index, value in df['DATE_DIED'].iteritems():
    if value == '9999-99-99':
        df.at[index, 'DIED'] = 0
    else: df.at[index, 'DIED'] = 1
    # 0 - no, 1 - yes
```

In [150…

```
df.head()
```

Out[150]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	PNEUMONIA	AGE	PREGNANT	DIABE
0	2	1	female	returned home	03/05/2020	NaN	1.0	65	2.0	
1	2	1	male	returned home	03/06/2020	NaN	1.0	72	NaN	
2	2	1	male	hospitalization	09/06/2020	1.0	2.0	55	NaN	
3	2	1	female	returned home	12/06/2020	NaN	2.0	53	2.0	
4	2	1	male	returned home	21/06/2020	NaN	2.0	68	NaN	

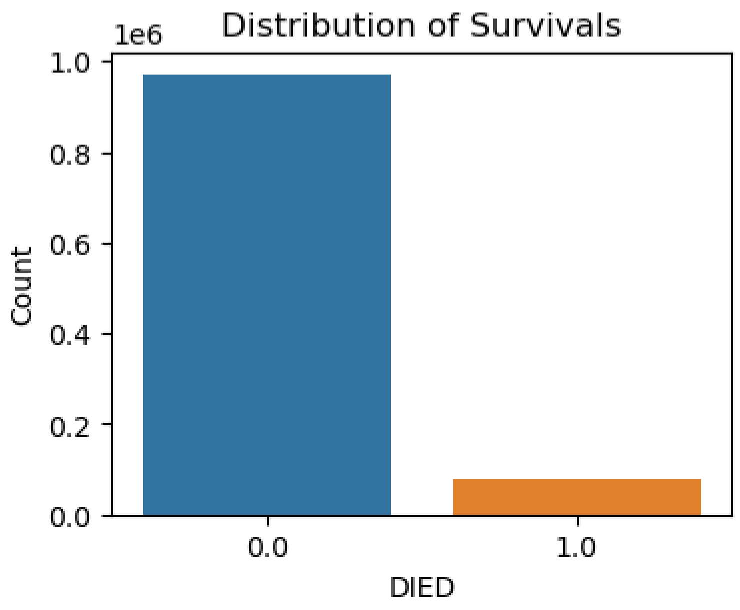
5 rows × 22 columns

==> standart scaler is not required because values are in similar scales

Visualization of unbalanced dataset

In [260…

```
Class = df['DIED'].value_counts()

plt.figure(figsize=(4, 3))   # Adjust the figsize as desired

sns.barplot(data=df, x=Class.index, y=Class.values)

plt.xlabel('DIED')
plt.ylabel('Count')
plt.title('Distribution of Survivals')

plt.show()
```



In [19]:

```
Class
```

Out[19]:

```
0.0    971633
1.0     76942
Name: DIED, dtype: int64
```

Preprocessing

Needed preprocessing

- 1) resolve NaN values
- 2) encode sex, patient_type columns
- 3) drop date_died because we have a derived DIED column from it

1) replace NaN values with zeros because we have too many NaN values and removing them would result in significantly smaller dataset

```
In [151…  df = df.fillna(0)
```

```
In [152…  df
```

Out[152]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	PNEUMONIA	AGE	PREGNANT
0	2	1	female	returned home	03/05/2020	0.0	1.0	65	2.0
1	2	1	male	returned home	03/06/2020	0.0	1.0	72	0.0
2	2	1	male	hospitalization	09/06/2020	1.0	2.0	55	0.0
3	2	1	female	returned home	12/06/2020	0.0	2.0	53	2.0
4	2	1	male	returned home	21/06/2020	0.0	2.0	68	0.0
...	...	...	...	...	...	...	...	...	...
1048570	2	13	male	returned home	9999-99-99	0.0	2.0	40	0.0
1048571	1	13	male	hospitalization	9999-99-99	2.0	2.0	51	0.0
1048572	2	13	male	returned home	9999-99-99	0.0	2.0	55	0.0
1048573	2	13	male	returned home	9999-99-99	0.0	2.0	28	0.0
1048574	2	13	male	returned home	9999-99-99	0.0	2.0	52	0.0

1048575 rows × 22 columns

2) encode sex, patient_type columns

```
In [153…  columns_to_encode = ['SEX', 'PATIENT_TYPE']
          enc = OrdinalEncoder()
          df[columns_to_encode] = enc.fit_transform(df[columns_to_encode])
```

3) drop DATE_DIED because we have a derived DIED column from it

```
In [154…  df = df.drop(['DATE_DIED'], axis=1)
```

```
In [155…  df.head()
```

Out[155]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	INTUBED	PNEUMONIA	AGE	PREGNANT	DIABETES	COPD	...
0	2	1	0.0	1.0	0.0	1.0	65	2.0	2.0	2.0	...
1	2	1	1.0	1.0	0.0	1.0	72	0.0	2.0	2.0	...
2	2	1	1.0	0.0	1.0	2.0	55	0.0	1.0	2.0	...
3	2	1	0.0	1.0	0.0	2.0	53	2.0	2.0	2.0	...
4	2	1	1.0	1.0	0.0	2.0	68	0.0	1.0	2.0	...

5 rows × 21 columns

Idea

We can employ oversampling now to make data more balanced but it is important to check the base case scenario, so it is going to be done later after running basic training and prediction

Training

Classification exercise. Algorihtms to be considered

- XGBoost
- Random Forest Classifier
- SVC => after some reaserch SVC might take too long to train on dataset that has about 1 million of rows as it has quadratic time compexity, so the alternative will be Stochastic Gradient Descent (SGD) Classifier

Split

```
In [24]: y = df['DIED'].astype(int)
         X = df.drop(['DIED'], axis=1)
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21
```

XGBoost

```
In [27]: xgb_c = xgb.XGBRFClassifier(random_state=22)
         xgb_model = xgb_c.fit(X_train, y_train, verbose=False)
         xgb_score = xgb_model.score(X_test, y_test)
         print('test accuracy: ', xgb_score)
```

```
test accuracy:  0.9495982643110888
```

```
In [28]: test_predictions = xgb_c.predict(X_test)
         confusion_matrix(y_test, test_predictions)
```

```
Out[28]: array([[191581,   2746],
                [  7824,   7564]], dtype=int64)
```

Random Forest Classifier

```
In [29]: rf = RandomForestClassifier(random_state=42)
         rf_model = rf.fit(X_train, y_train)
```

```
In [30]: rf_pred = rf_model.predict(X_test)
         rf_pred_score = accuracy_score(y_test, rf_pred)
         print("Random Forest accuracy:", accuracy_score(y_test, rf_pred))
```

```
Random Forest accuracy: 0.944410271082183
```

```
In [31]: confusion_matrix(y_test, rf_pred)
```

```
Out[31]: array([[189749,   4578],
                [  7080,   8308]], dtype=int64)
```

SGDClassifier

In [33]:
```
sgd = SGDClassifier(random_state=42)
```

In [34]:
```
sgd_model = sgd.fit(X_train, y_train)
```

In [35]:
```
sgd_pred = sgd_model.predict(X_test)
```

In [36]:
```
sgd_pred_score = accuracy_score(y_test, sgd_pred)
```

Displaying table of accuracy score results

In [37]:
```
data = {
    'Model': ['XGBoost', 'RFC', 'SGD'],
    'Accuracy score': [
        xgb_c.score(X_test, y_test),
        accuracy_score(y_test, rf_pred),
        accuracy_score(y_test, sgd_pred)
    ]
}

pd.DataFrame(data)
```

Out[37]:

	Model	Accuracy score
0	XGBoost	0.949598
1	RFC	0.944410
2	SGD	0.935021

Let's experiment with some oversampling

In [39]:
```
sm = SMOTE(random_state=22, sampling_strategy=0.12) # previously ratio was 0.07
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
```

run training again with XGBoost

In [40]:
```
xgb_res = xgb_c.fit(X_train_res, y_train_res)
```

In [41]:
```
xgb_res_score = xgb_res.score(X_test, y_test)
print('XGBoost test accuracy after oversampling: ', xgb_res_score)
```

```
XGBoost test accuracy after oversampling:  0.9465798822211096
```

run training again with RFC

In [42]:
```
rf_res = rf.fit(X_train_res, y_train_res)
rf_pred_res = rf_res.predict(X_test)
rf_pred_res_score = accuracy_score(y_test, rf_pred_res)
print("Random Forest accuracy after oversampling:", rf_pred_res_score)
```

```
Random Forest accuracy after oversampling: 0.9427318026845958
```

run training again with SGD

In [43]:
```
sgd_res = sgd.fit(X_train_res, y_train_res)
```

In [44]:
```
sgd_pred_res = sgd.predict(X_test)
```

In [45]:
```
sgd_pred_res_score = accuracy_score(y_test, sgd_pred_res)
```

Table with end results of oversampling

In [46]:
```
data = {
    'Model': ['XGBoost', 'RFC', 'SGD'],
    'Accuracy score': [
        xgb_score,
        rf_pred_score,
        sgd_pred_score
    ],
    'After oversampling': [
        xgb_res_score,
        rf_pred_res_score,
        sgd_pred_res_score
    ],
    'Change': [
        xgb_res_score - xgb_score,
        rf_pred_res_score - rf_pred_score,
        sgd_pred_res_score - sgd_pred_score,
    ]
}

pd.DataFrame(data)
```

Out[46]:

	Model	Accuracy score	After oversampling	Change
0	XGBoost	0.949598	0.946580	-0.003018
1	RFC	0.944410	0.942732	-0.001678
2	SGD	0.935021	0.936199	0.001178

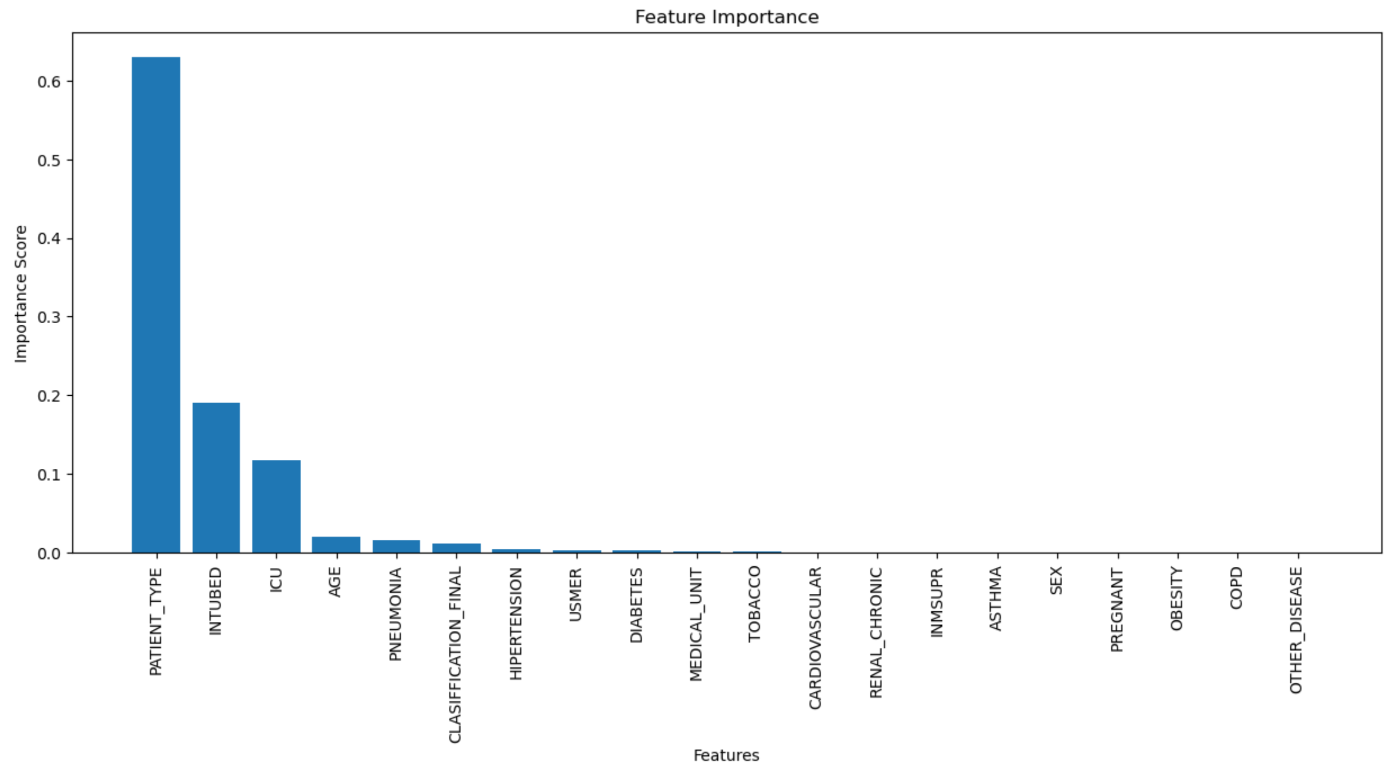
==> after trying few sampling strategies we were unlucky to increase the prediction accuracy score

Get the importance of each feature

In [132...
```
importance_scores = xgb_model.feature_importances_
feature_names = X_train.columns
```

In [133...
```
sorted_indices = importance_scores.argsort()[::-1]
sorted_scores = importance_scores[sorted_indices]
sorted_features = feature_names[sorted_indices]
```

In [134...
```
plt.figure(figsize=(15, 6))
plt.bar(range(len(sorted_scores)), sorted_scores, tick_label=sorted_features)
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.title('Feature Importance')
plt.show()
```



```
In [54]: sorted_scores
```

```
Out[54]: array([6.2987763e-01, 1.9045393e-01, 1.1805581e-01, 1.9742483e-02,
                1.5109948e-02, 1.1119973e-02, 4.7146557e-03, 2.6800705e-03,
                2.4255824e-03, 1.8720088e-03, 7.0700998e-04, 5.9298112e-04,
                5.4513011e-04, 4.9566140e-04, 4.2314420e-04, 3.5191496e-04,
                3.2413739e-04, 2.6683530e-04, 1.2151147e-04, 1.1954844e-04],
               dtype=float32)
```

```
In [55]: sorted_features
```

```
Out[55]: Index(['PATIENT_TYPE', 'INTUBED', 'ICU', 'AGE', 'PNEUMONIA',
                'CLASIFFICATION_FINAL', 'HIPERTENSION', 'USMER', 'DIABETES',
                'MEDICAL_UNIT', 'TOBACCO', 'CARDIOVASCULAR', 'RENAL_CHRONIC', 'INMSUPR',
                'ASTHMA', 'SEX', 'PREGNANT', 'OBESITY', 'COPD', 'OTHER_DISEASE'],
               dtype='object')
```

```
In [135…: pd.DataFrame({
              'Feature': sorted_features,
              'Importance score': sorted_scores
          })
```

Out[135]:

	Feature	Importance score
0	PATIENT_TYPE	0.629878
1	INTUBED	0.190454
2	ICU	0.118056
3	AGE	0.019742
4	PNEUMONIA	0.015110
5	CLASIFFICATION_FINAL	0.011120
6	HIPERTENSION	0.004715
7	USMER	0.002680
8	DIABETES	0.002426
9	MEDICAL_UNIT	0.001872

10	TOBACCO	0.000707
11	CARDIOVASCULAR	0.000593
12	RENAL_CHRONIC	0.000545
13	INMSUPR	0.000496
14	ASTHMA	0.000423
15	SEX	0.000352
16	PREGNANT	0.000324
17	OBESITY	0.000267
18	COPD	0.000122
19	OTHER_DISEASE	0.000120

Insight

After examining the importance of each feature, ['TOBACCO', 'CARDIOVASCULAR', 'RENAL_CHRONIC', 'INMSUPR', 'ASTHMA', 'SEX', 'PREGNANT', 'OBESITY', 'COPD', 'OTHER_DISEASE'] showed really low siginificance on the prediction model ==> remove them to save training and optimization time or even achieve better results due to less distractions for the training model

In [156…

```
df_min = df.drop(['TOBACCO', 'CARDIOVASCULAR', 'RENAL_CHRONIC', 'INMSUPR',
        'ASTHMA', 'SEX', 'PREGNANT', 'OBESITY', 'COPD', 'OTHER_DISEASE'], axis=1)
```

In [98]:

```
y_min = df_min['DIED'].astype(int)
X_min = df_min.drop(['DIED'], axis=1)
X_train_min, X_test_min, y_train_min, y_test_min = train_test_split(X_min, y_min, test_s
```

In [99]:

```
xgb_c_min = xgb.XGBRFClassifier(random_state=22)
xgb_model_min = xgb_c_min.fit(X_train_min, y_train_min, verbose=False)
xgb_score_min = xgb_model_min.score(X_test_min, y_test_min)
print('test accuracy: ', xgb_score_min)
```

```
test accuracy:  0.9496793267052905
```

==> result: test accuracy increased score from 0.949598% to 0.949679% (0.0085 percentage points increase)

In [157…

```
df_min.head()
```

Out[157]:

	USMER	MEDICAL_UNIT	PATIENT_TYPE	INTUBED	PNEUMONIA	AGE	DIABETES	HIPERTENSION	CLASIFFICA
0	2	1	1.0	0.0	1.0	65	2.0	1.0	
1	2	1	1.0	0.0	1.0	72	2.0	1.0	
2	2	1	0.0	1.0	2.0	55	1.0	2.0	
3	2	1	1.0	0.0	2.0	53	2.0	2.0	
4	2	1	1.0	0.0	2.0	68	1.0	1.0	

In [78]:

```
importance_scores_min = xgb_model_min.feature_importances_
feature_names_min = X_train_min.columns
```

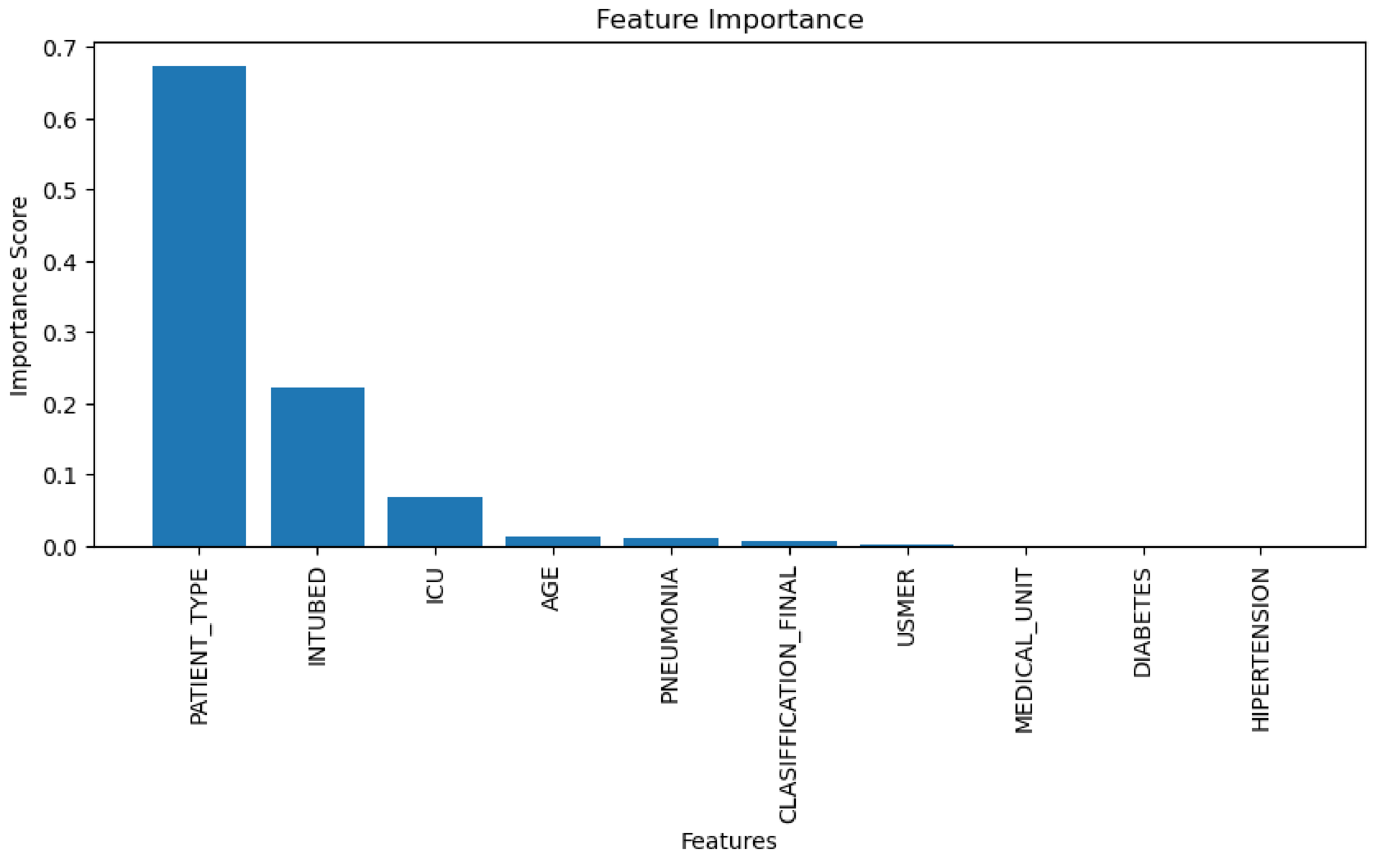
In [79]:

```
sorted_indices_min = importance_scores_min.argsort()[::-1]
```

```
sorted_scores_min = importance_scores_min[sorted_indices_min]
sorted_features_min = feature_names_min[sorted_indices_min]
```

In [251…

```
plt.figure(figsize=(10, 4))
plt.bar(range(len(sorted_scores_min)), sorted_scores_min, tick_label=sorted_features_min
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.title('Feature Importance')
plt.show()
```



In [81]:

```
sorted_scores_min
```

Out[81]:

```
array([6.7389810e-01, 2.2359285e-01, 6.7947805e-02, 1.2587040e-02,
       1.1389460e-02, 6.5300497e-03, 1.7971499e-03, 1.0736458e-03,
       7.0464256e-04, 4.7920737e-04], dtype=float32)
```

In [82]:

```
sorted_features_min
```

Out[82]:

```
Index(['PATIENT_TYPE', 'INTUBED', 'ICU', 'AGE', 'PNEUMONIA',
       'CLASIFFICATION_FINAL', 'USMER', 'MEDICAL_UNIT', 'DIABETES',
       'HIPERTENSION'],
      dtype='object')
```

Maybe hypertension is also redundant?

In [139…

```
df_min_2 = df.drop(['HIPERTENSION'], axis=1)
```

In [140…

```
y_min_2 = df_min_2['DIED'].astype(int)
X_min_2 = df_min_2.drop(['DIED'], axis=1)
X_train_min_2, X_test_min_2, y_train_min_2, y_test_min_2 = train_test_split(X_min_2, y_m
```

In [85]:

```
# xgb_c_min = xgb.XGBRFClassifier(random_state=22)
# xgb_model_min = xgb_c_min.fit(X_train_min, y_train_min, verbose=False)
# xgb_score_min = xgb_model_min.score(X_test_min, y_test_min)
# print('test accuracy: ', xgb_score_min)
```

```
test accuracy:  0.94959349593495 93
```

Hypertension is important feature for the model. Removal of it resulted in worse accuracy score

== EXTRA EDA ==

Ploting data to see how certain conditions influence the rate of death

In [167…

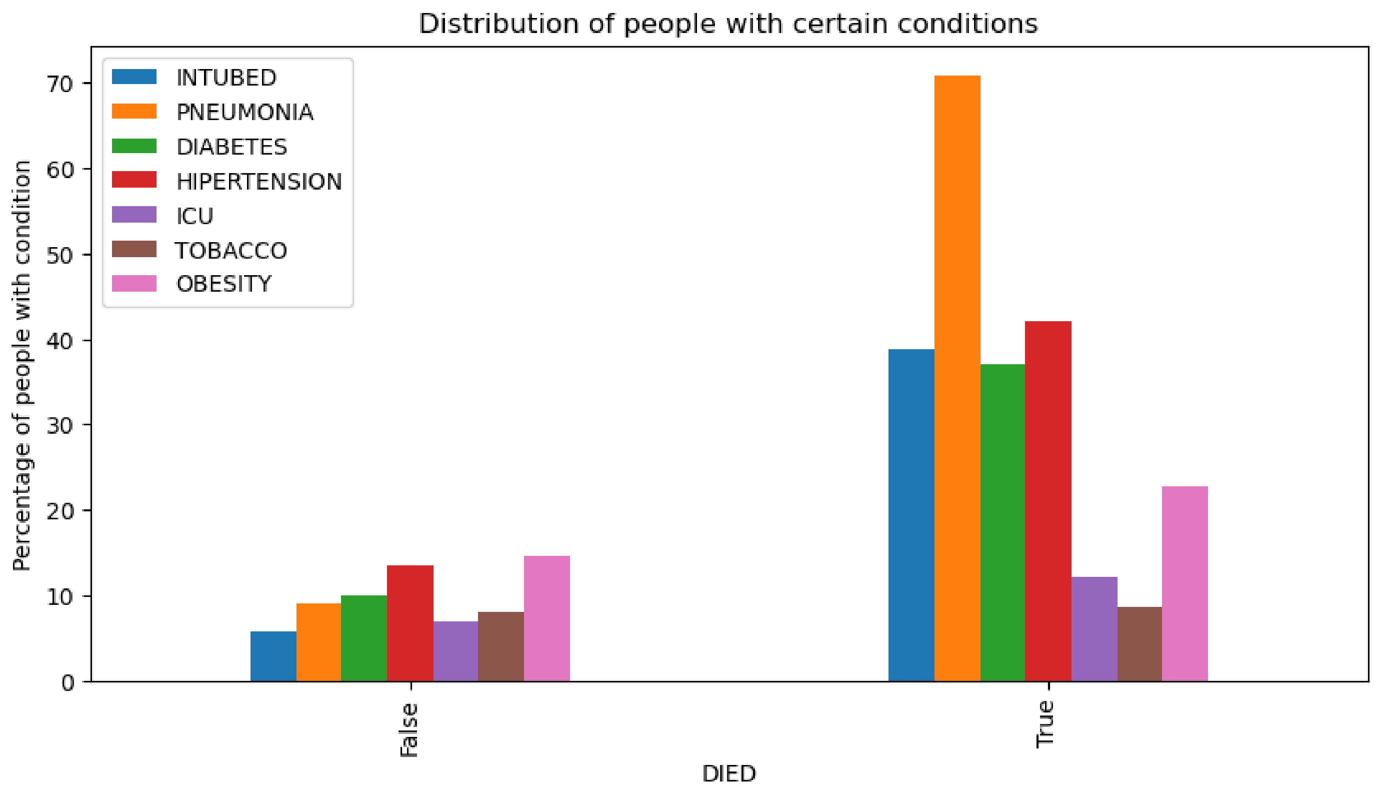
```python
# Make data copy
df_plot = df.copy()
```

In [168…

```python
# Map values to boolean (True/False)
binary_columns = ['INTUBED', 'PNEUMONIA', 'DIABETES', 'HIPERTENSION', 'ICU', 'TOBACCO',
for column in binary_columns:
    df_plot[column] = df_plot[column].map({0: np.nan, 1: True, 2: False})
```

In [179…

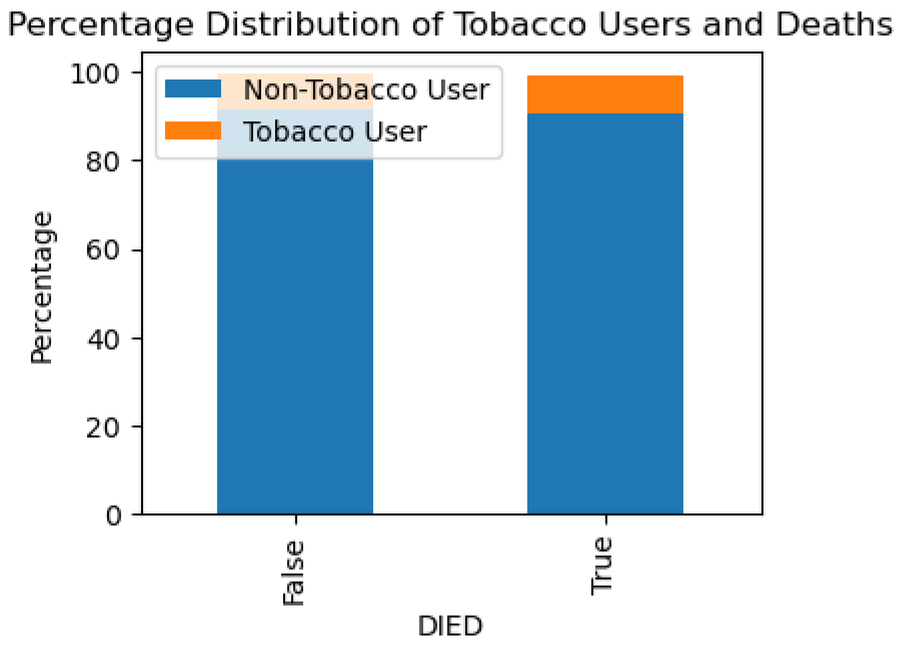
```python
# Group by "DIED" column and calculate the percentage of true values for each column
grouped = df_plot.groupby('DIED')[binary_columns].mean() * 100

ax = grouped.plot(kind='bar', figsize=(10, 5))

custom_labels = ['False', 'True']
ax.set_xticklabels(custom_labels)

# Create a bar plot
plt.xlabel('DIED')
plt.ylabel('Percentage of people with condition')
plt.title('Distribution of people with certain conditions')
plt.legend(loc='upper left')
plt.show()
```



Checking whether the tobacco consumers had higher rates of death

In [249…

```
# Group by "DIED" and "TOBACCO" columns and calculate the percentage of true values for
grouped = (df_plot.groupby(['DIED', 'TOBACCO']).size() / df_plot.groupby('DIED').size())

# Create the stacked bar plot
ax = grouped.plot(kind='bar', stacked=True, figsize=(4, 3))

# Set the x-axis tick labels
ax.set_xticklabels(['False', 'True'])

# Set the plot labels and title
plt.xlabel('DIED')
plt.ylabel('Percentage')
plt.title('Percentage Distribution of Tobacco Users and Deaths')

# Display the legend and show the plot
plt.legend(['Non-Tobacco User', 'Tobacco User'], loc='upper left')
plt.show()
```

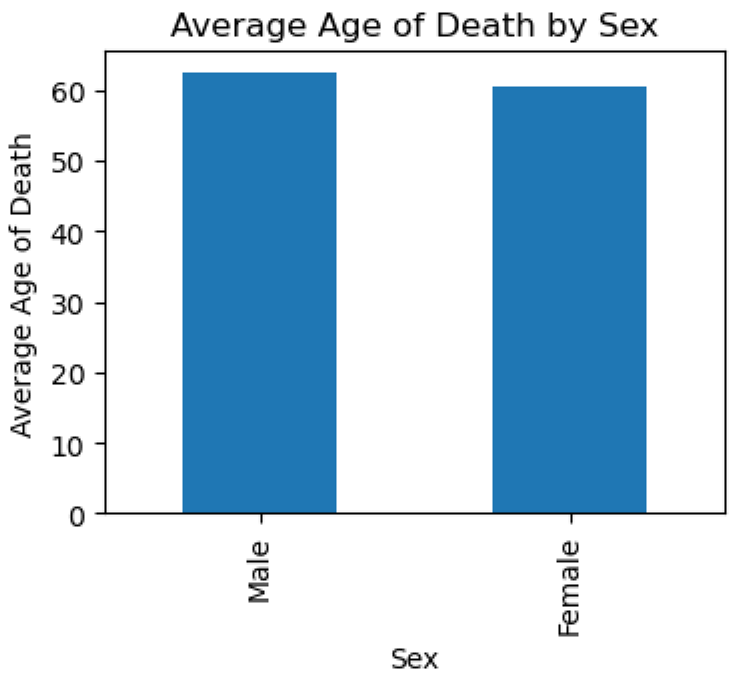


==> Seems that tobacco usage didn't have a big effect on the chance of death

Now let's see what is the average age of death for both sexes

In [250…

```
# Filter the DataFrame to include only rows where individuals died
df_filtered = df_plot[df_plot['DIED'] == True]

# Group by 'DIED' and 'SEX' columns and calculate the average age of death for each comb
grouped = df_filtered.groupby('SEX')['AGE'].mean()

# Create the bar plot
ax = grouped.plot(kind='bar', figsize=(4, 3))

# Set the x-axis tick labels
ax.set_xticklabels(['Male', 'Female'])

# Set the plot labels and title
plt.xlabel('Sex')
plt.ylabel('Average Age of Death')
plt.title('Average Age of Death by Sex')

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



==> Average age of death by sex is approximately the same

Checking the conditions across different age groups

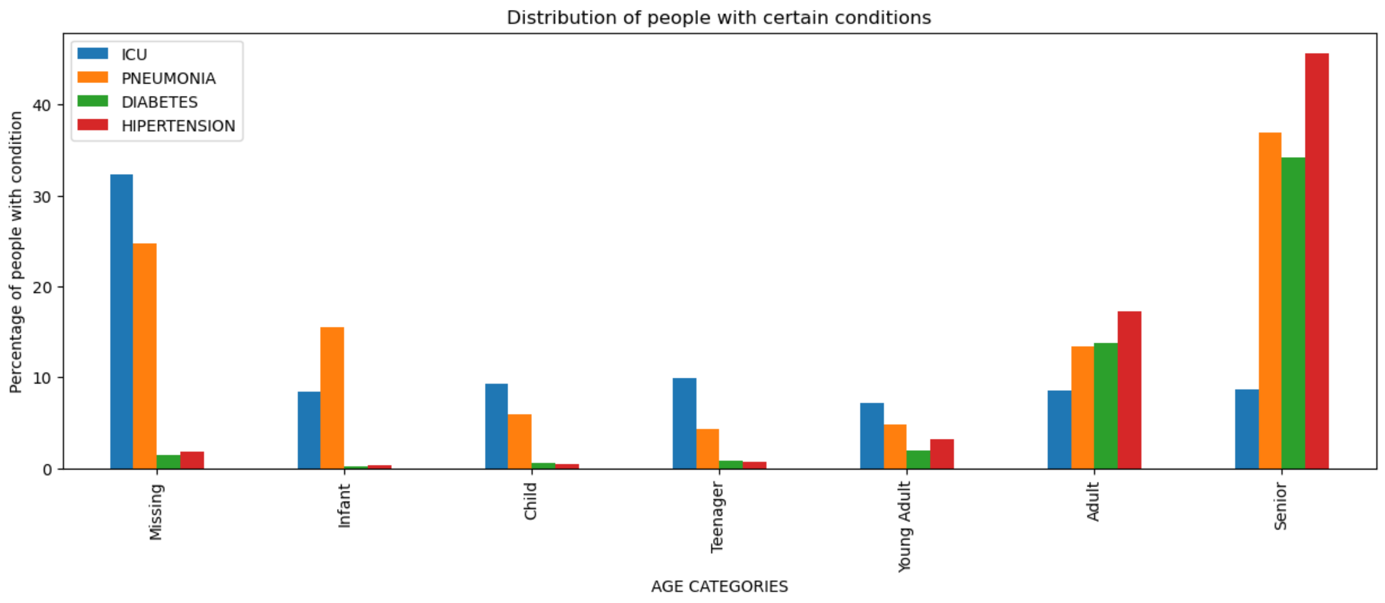
In [238…
```
def process_age(df, cut_points, label_names):
    df_age = df.copy()
    df_age['AGE'] = df_age['AGE'].fillna(-0.5)
    df_age['AGE_CATEGORIES'] = pd.cut(df_age['AGE'], cut_points, labels=label_names)
    return df_age
```

In [239…
```
cut_points = [-1, 0, 5, 12, 18, 35, 60, 121]
label_names = ['Missing', 'Infant', 'Child', 'Teenager', 'Young Adult', 'Adult', 'Senior

df_age = process_age(df, cut_points, label_names)
```

In [240…
```
# Make data copy
df_age_plot = df_age.copy()
```

In [241…
```
# Map values to boolean (True/False)
binary_columns = [ 'ICU', 'PNEUMONIA', 'DIABETES', 'HIPERTENSION']
for column in binary_columns:
    df_age_plot[column] = df_age_plot[column].map({0: np.nan, 1: True, 2: False})
```

In [242…
```
# Group by "AGE_CATEGORIES" column and calculate the percentage of true values for each
grouped = df_age_plot.groupby('AGE_CATEGORIES')[binary_columns].mean() * 100

ax = grouped.plot(kind='bar', figsize=(15, 5))
# Create a bar plot
plt.xlabel('AGE_CATEGORIES')
plt.ylabel('Percentage of people with condition')
plt.title('Distribution of people with certain conditions')
plt.legend(loc='upper left')
plt.show()
```



==> As expected higher rate of death in adults and seniors probably were caused by illnesses and other conditions

Optimization

Things to consider

- Knowing that XGBoost gave the best base result, we will try to optimize it
- During optimization it is important to include base parameters

In [243…

```
xgb_model_min.get_params()
```

Out[243]:

```
{'colsample_bynode': 0.8,
 'learning_rate': 1.0,
 'reg_lambda': 1e-05,
 'subsample': 0.8,
 'objective': 'binary:logistic',
 'use_label_encoder': None,
 'base_score': None,
 'booster': None,
 'callbacks': None,
 'colsample_bylevel': None,
 'colsample_bytree': None,
 'early_stopping_rounds': None,
 'enable_categorical': False,
 'eval_metric': None,
 'feature_types': None,
 'gamma': None,
 'gpu_id': None,
 'grow_policy': None,
 'importance_type': None,
 'interaction_constraints': None,
 'max_bin': None,
 'max_cat_threshold': None,
 'max_cat_to_onehot': None,
 'max_delta_step': None,
 'max_depth': None,
 'max_leaves': None,
 'min_child_weight': None,
 'missing': nan,
 'monotone_constraints': None,
 'n_estimators': 100,
 'n_jobs': None,
 'num_parallel_tree': None,
 'predictor': None,
 'random_state': 22,
 'reg_alpha': None,
 'sampling_method': None,
 'scale_pos_weight': None,
 'tree_method': None,
 'validate_parameters': None,
 'verbosity': None}
```

In [246…

```
#Define the parameter grid for hyperparameter search. Important to include current xgb_m
param_grid = {
    'learning_rate': [1, 0.9, 0.8],
    'max_depth': [None, 2, 3],
    'n_estimators': [100, 200, 500]
}

grid_search = GridSearchCV(xgb_model_min, param_grid, cv=3, scoring='accuracy', n_jobs=-
grid_search.fit(X_train_min, y_train_min)

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Evaluate the best model on the test set
best_score = best_model.score(X_test_min, y_test_min)

# Print the best hyperparameters and model performance
print("Best Hyperparameters:", best_params)
print("Best Model Score:", best_score)
```

```
Best Hyperparameters: {'learning_rate': 1, 'max_depth': None, 'n_estimators': 100}
Best Model Score: 0.9496793267052905
```

Try different parameters' values

In [263…

```
param_grid_2 = {
    'learning_rate': [1, 0.2, 0.1],
    'max_depth': [None, 2, 3, 5],   # acording to the creators going above 5 is rarely ad
    'n_estimators': [100, 200, 300],
}

grid_search_2 = GridSearchCV(xgb_model_min, param_grid, cv=3, scoring='accuracy', n_jobs
grid_search_2.fit(X_train_min, y_train_min)

best_params_2 = grid_search_2.best_params_
best_model_2 = grid_search_2.best_estimator_

best_score_2 = best_model_2.score(X_test_min, y_test_min)

print("Best Hyperparameters:", best_params_2)
print("Best Model Score:", best_score_2)
```

```
Best Hyperparameters: {'learning_rate': 1, 'max_depth': None, 'n_estimators': 100}
Best Model Score: 0.9496793267052905
```

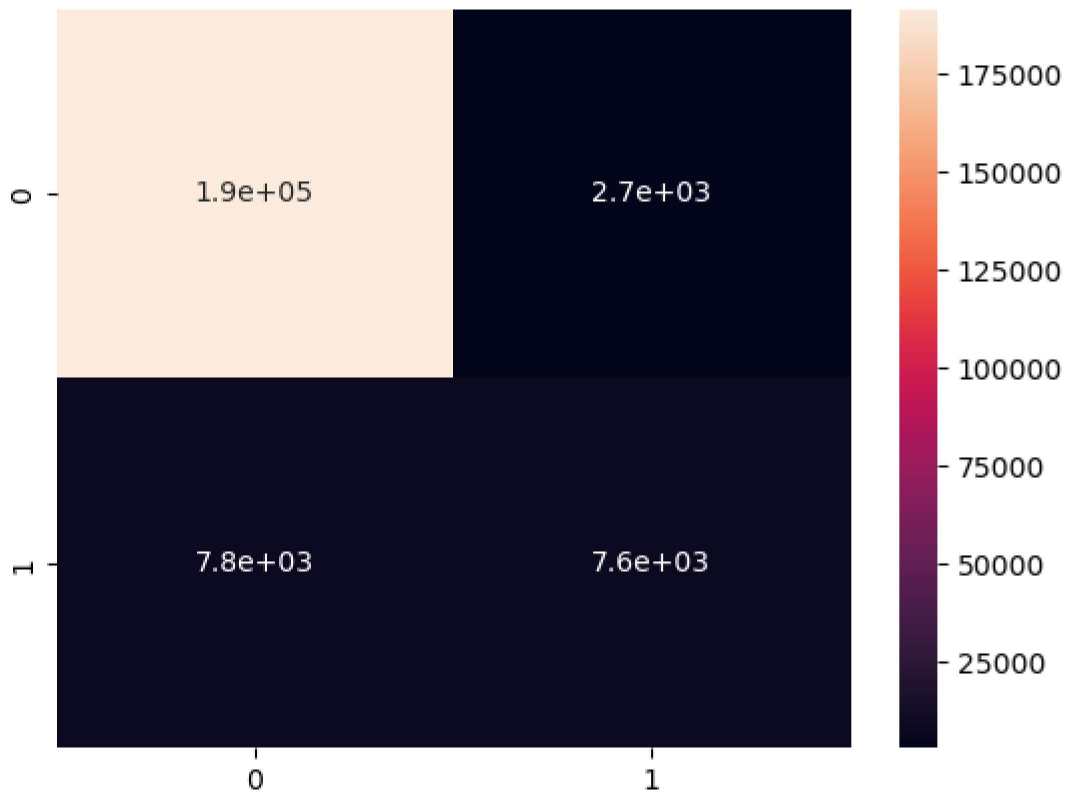
==> no serious ahievements were reached after optimization

In [247…

```
test_predictions = best_model.predict(X_test_min)
cm = confusion_matrix(y_test, test_predictions)
sns.heatmap(cm, annot=True)
```

Out[247]:

```
<AxesSubplot:>
```



```
In [248…  cm
```

```
Out[248]: array([[191583,   2744],
                 [  7809,   7579]], dtype=int64)
```

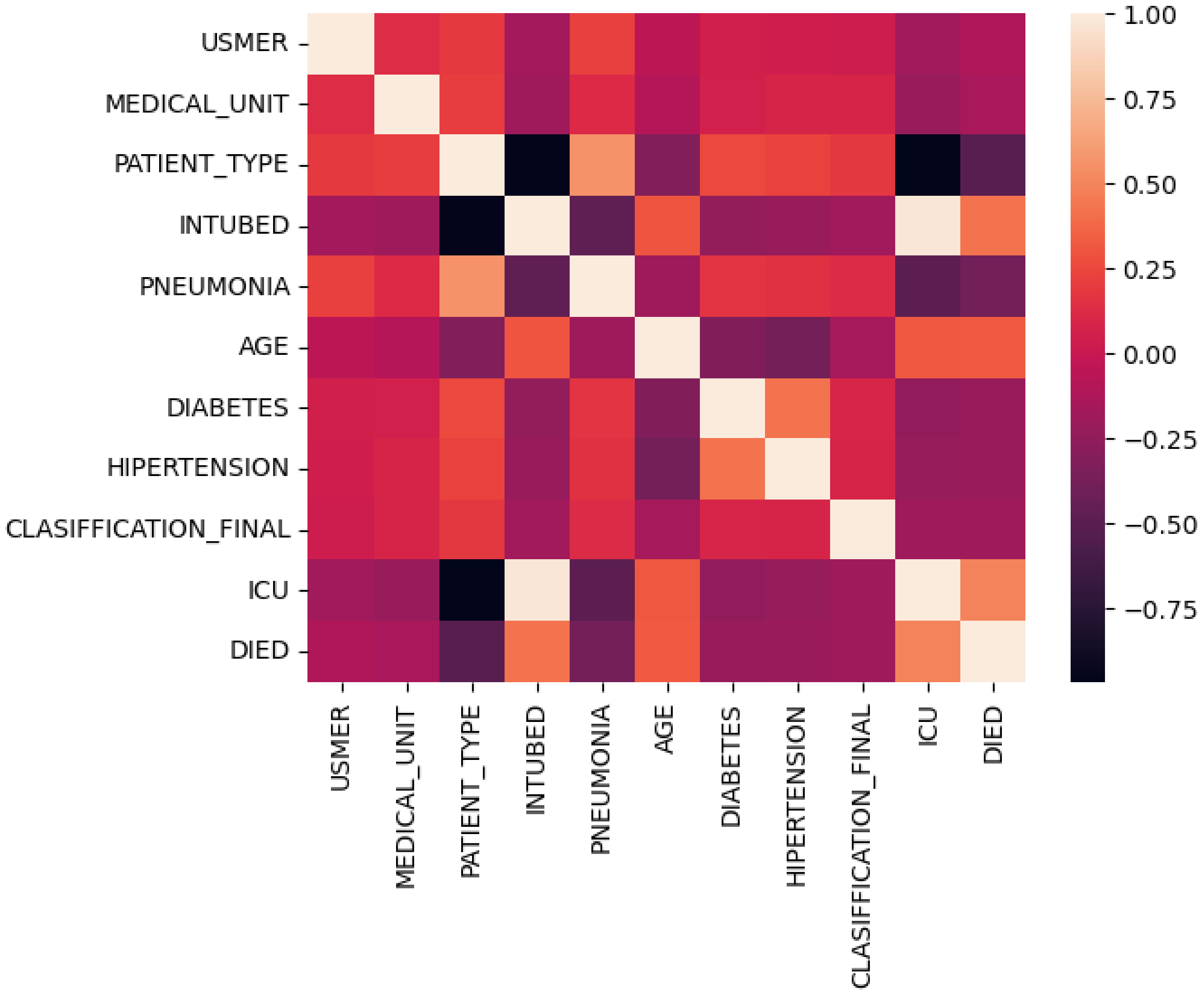
Based on the provided confusion matrix:

- True Positives (TP): The model predicted 191,583 patients would survive, and they actually did survive.
- False Positives (FP): The model predicted 2,744 patients would survive, but they actually died.
- False Negatives (FN): The model predicted 7,809 patients would die, but they actually survived.
- True Negatives (TN): The model predicted 7,579 patients would die, and they indeed died.

What about correlation now?

```
In [261…  sns.heatmap(df_min.corr())
```

```
Out[261]: <AxesSubplot:>
```



In [262…
```
df_min.corr()
```

Out[262]:

	USMER	MEDICAL_UNIT	PATIENT_TYPE	INTUBED	PNEUMONIA	AGE	DIABETES
USMER	1.000000	0.127927	0.190570	-0.169939	0.213391	-0.045088	0.049919
MEDICAL_UNIT	0.127927	1.000000	0.205413	-0.195279	0.116055	-0.082223	0.063395
PATIENT_TYPE	0.190570	0.205413	1.000000	-0.952396	0.559036	-0.315193	0.250458
INTUBED	-0.169939	-0.195279	-0.952396	1.000000	-0.476610	0.299167	-0.235915
PNEUMONIA	0.213391	0.116055	0.559036	-0.476610	1.000000	-0.192778	0.170428
AGE	-0.045088	-0.082223	-0.315193	0.299167	-0.192778	1.000000	-0.321798
DIABETES	0.049919	0.063395	0.250458	-0.235915	0.170428	-0.321798	1.000000
HIPERTENSION	0.047990	0.084270	0.229903	-0.214888	0.150247	-0.382671	0.418234
CLASIFFICATION_FINAL	0.028840	0.079981	0.183370	-0.175733	0.121063	-0.152637	0.094151
ICU	-0.175673	-0.214996	-0.963993	0.972450	-0.488532	0.315663	-0.243898
DIED	-0.112671	-0.149030	-0.515582	0.422331	-0.381977	0.320801	-0.215319

==> Difficult to asses. High correlation among PATIENT_TYPE, INTUBED and ICU but these features are the most important for prediction model...

Some tests with undersampling

In [265…
```
rus = RandomUnderSampler(sampling_strategy=0.9, random_state=33)

X_train_under, y_train_under = rus.fit_resample(X_train, y_train)
```

In [266…
```
xgb_c_under = xgb.XGBRFClassifier(random_state=22)
xgb_model_under = xgb_c_under.fit(X_train_under, y_train_under, verbose=False)
xgb_score_under = xgb_model_under.score(X_test, y_test)
print('test accuracy: ', xgb_score_under)
```

```
test accuracy:  0.8838280523567699
```

==> Worse performance from undersampling

Summary:

During this project, comprehensive steps were undertaken to thoroughly explore the dataset related to Covid-19 death cases. The process included essential tasks such as exploratory data analysis (both technical and interpretative), pre-processing, training, feature selection, and optimization. The primary objective of achieving an accuracy of over 90% was not only met but surpassed successfully.

- final accuracy score: 0.9496793267052905 > 0.9

In [ ]:
```
```