Project Report On
A COVID-19 Outcome Classification

College of Computing

ITMD 522-01 Data Mining and Machine Learning

Professor Zheng

Submitted by:

Gyuhwan Choi (gchoi6@hawk.iit.edu)

Table of Contents

1. Introduction

The COVID-19 pandemic, a global health crisis of unprecedented scale, has catalyzed the need for advanced data-driven solutions to better understand the dynamics of the virus's impact on public health. As a response to this pressing challenge, our project embarks on the exploration of COVID-19 test result data to glean valuable insights and contribute to the border understanding of the pandemic.

The main goal of this project is to build a machine learning model that, given a Covid-19 patient's current symptom, status, and medical history, will predict whether the patient is at high risk or not.

2. Dataset

I've found this COVID-19 Dataset from Kaggle:

https://www.kaggle.com/datasets/meirnizri/covid19-dataset/data

The size of the data is **21 columns** x **1000k rows**.

The rows are 1,048,575 which indicates the information for each unique patient.

Variables:

sex: 1 for female and 2 for male.

age: of the patient.

classification: covid test findings. Values 1-3 mean that the patient was diagnosed with covid in different degrees. 4 or higher means that the patient is not a carrier of covid or that the test is inconclusive.

patient type: type of care the patient received in the unit. 1 for returning home and 2 for hospitalization.

pneumonia: whether the patient already has air sacs inflammation or not.

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.

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.

usmr: Indicates whether the patient treated medical units of the first, second or third level.

medical unit: type of institution of the National Health System that provided the care.

intubed: whether the patient was connected to the ventilator.

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

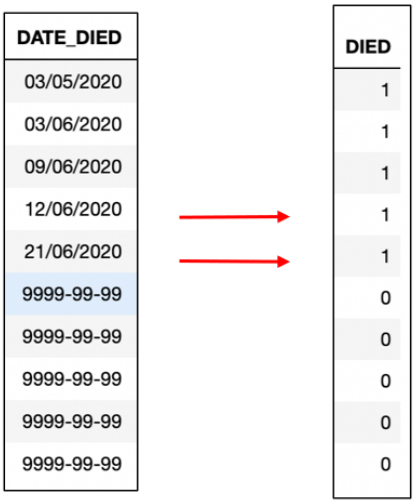
date died: If the patient died indicate the date of death, and 9999-99-99 otherwise.

Target Variables:

The target variable in our machine learning project is to determine whether an individual is at high risk or not. To facilitate our analysis, I transformed the **'Date_Died'** column, originally an object variable, into a **'Died'** column with numerical features. In this context, 0 denotes that the person is alive, while 1 indicates that the person has died. I leverage this column for our predictive modeling efforts.

```
PATIENT_TYPE                int64
DATE_DIED                  object
INTUBED                     int64
PNEUMONIA                   int64
AGE                         int64
```

```
CLASIFFICATION_FINAL        int64
ICU                         int64
DIED                        int64
dtype: object
```



3. Problems and Solutions

3.1 Research Problems

Problem:

Develop a classification model to predict the high-risk status of COVID-19 patients based on their habits and health indicators.

Challenges:

1. The dataset includes nominal values that require appropriate handling, such as through data transformation (discretization).

2. The presence of numerous missing values in our dataset poses a potential impact on the accuracy and reliability of our analysis and predictive outcomes.
3. Address data imbalance concerning the target variable (y label) to mitigate potential overfitting issues.

3.2 Research Solutions

In this section, a brief overview of the solutions will be presented. Subsequent sections will delve into a more detailed discussion and implementation of these solutions to address the identified challenges.

Solutions:

1. To address nominal values in our dataset, I performed data transformation by converting the "Date_died" column to "Died," transitioning it from a nominal to a numerical representation.
2. First, I identified the number of missing values in our dataset, followed by an analysis of the missing data. Finally, I imputed the appropriate values for those missing values.
3. Initially, I identified the imbalance issue within the dataset. Then, I applied oversampling and reassessed for any remaining imbalance in the data.

4. Data Preprocessing

4.1 Discretization

1. First of all, checking the data types:

```
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
```

I see there is one object variable in our dataset.

2. The value '9999-99-99' in 'DATE_DIED' represents the person who has not died and if there is a date specified, it means the person has died. Based on that I converted DATE_DIED into a binary feature where 0 means alive and 1 means died. Now I have the data type from DATE_DIE (object) to DIED (integer).

```
df['DIED'] = [0 if i=='9999-99-99' else 1 for i in df.DATE_DIED]
df.drop('DATE_DIED', axis=1, inplace=True)

# Check for data types
print(df.dtypes)

display(HTML(df.head(10).to_html()))
```

```
PATIENT_TYPE            int64
DATE_DIED              object
INTUBED                 int64
PNEUMONIA               int64
AGE                     int64
```

```
CLASIFFICATION_FINAL    int64
ICU                     int64
DIED                    int64
dtype: object
```

DATE_DIED
03/05/2020
03/06/2020
09/06/2020
12/06/2020
21/06/2020
9999-99-99
9999-99-99
9999-99-99
9999-99-99
9999-99-99

DIED
1
1
1
1
1
0
0
0
0
0

4.2 Missing values

Checking the missing values (In the image below it says there are 0 missing values, because 97, 98, and 99 represent missing values in our dataset).

```
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
```

```python
# Check for missing values
print(df.isnull().sum())
```

Subsequently, I searched how many missing values are in each feature. Notably, the 3 variables '**INTUBED', 'PREGNANT',** and **'ICU'** exhibit a considerable number of missing values.

```python
# Check how many missing values each feature have
df_null = df.copy()
for i in [97, 98, 99]:
    df_null.replace(i , np.nan, inplace = True)

print(df_null.isnull().sum())

df_null.head()
```

```
USMER                        0
MEDICAL_UNIT                 0
SEX                          0
PATIENT_TYPE                 0
INTUBED                 855869
PNEUMONIA                16003
AGE                        345
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
DIED                         0
dtype: int64
```

INTUBED:

For the intubated column, it is observed that all missing values denoted by (97) correspond exclusively to instances where the patient status is (2), as individuals who have never been hospitalized logically cannot be connected to a ventilator.

```
At PATIENT_TYPE = 1 and at INTUBED = 1 the shape will be:
(0, 21)
---
At PATIENT_TYPE = 2 and at INTUBED = 1 the shape will be:
(33656, 21)
---
At PATIENT_TYPE = 1 and at INTUBED = 2 the shape will be:
(0, 21)
---
At PATIENT_TYPE = 2 and at INTUBED = 2 the shape will be:
(159050, 21)
---
At PATIENT_TYPE = 1 and at INTUBED = 97 the shape will be:
(848544, 21)
---
At PATIENT_TYPE = 2 and at INTUBED = 97 the shape will be:
(0, 21)
---
At PATIENT_TYPE = 1 and at INTUBED = 99 the shape will be:
(0, 21)
---
At PATIENT_TYPE = 2 and at INTUBED = 99 the shape will be:
(7325, 21)
---
```

```
df.INTUBED.value_counts()
```

```
97     848544
2      159050
1       33656
99       7325
Name: INTUBED, dtype: int64
```



```
df['INTUBED'].replace (97, 2, inplace = True)
df.INTUBED.value_counts()
```

```
2     1007594
1       33656
99       7325
Name: INTUBED, dtype: int64
```

PREGNANT:

In the pregnant column, the identification of missing values (98) reveals a correlation with female gender entries. Consequently, to address this, the corresponding values in the male section (97) are replaced with (2), as it is biologically impossible for men to be pregnant.

```
df.SEX.shape
```

```
(1048575,)
```

```
print(df[(df['SEX'] == 1)].shape)
print(df[(df['SEX'] == 2)].shape)
```

```
(525064, 21)
(523511, 21)
```

```
df[(df['SEX'] == 1)]['PREGNANT']
```

```
0          2
3          2
5          2
6          2
7          2
          ..
1048563    2
1048564    2
1048565    2
1048567    2
1048569    2
Name: PREGNANT, Length: 525064, dtype: int64
```

```
df[(df['SEX'] == 1)]['PREGNANT'].value_counts()
```

```
2     513179
1       8131
98      3754
Name: PREGNANT, dtype: int64
```

```
513179+8131+3754
```

```
525064
```

```
df[(df['SEX'] == 2)]['PREGNANT']
```

```
1          97
2          97
4          97
11         97
12         97
           ..
1048570    97
1048571    97
1048572    97
1048573    97
1048574    97
Name: PREGNANT, Length: 523511, dtype: int64
```

```
df[(df['SEX'] == 2)]['PREGNANT'].value_counts()
```

```
97    523511
Name: PREGNANT, dtype: int64
```

```
df[(df['SEX'] == 2) & (df['PREGNANT'])]['PREGNANT'].value_counts()
```

```
97    523511
Name: PREGNANT, dtype: int64
```

```
df['PREGNANT'].value_counts()
```

```
97    523511
2     513179
1       8131
98      3754
Name: PREGNANT, dtype: int64
```

```
df['PREGNANT'].value_counts()
```

```
97    523511
2     513179
1       8131
98      3754
Name: PREGNANT, dtype: int64
```

```
df['PREGNANT'].replace (97, 2, inplace = True)
df['PREGNANT'].value_counts()
```

```
2     1036690
1        8131
98       3754
Name: PREGNANT, dtype: int64
```

ICU:

Regarding the ICU column, missing values represented by (97) are consistently associated with PATIENT_TYPE = 1, signifying non-hospitalized patients. Simultaneously, missing values denoted by (99) are linked to hospitalized patients, rendering them indiscernible or unpredictable. Hence, I can replace all (97) values with (2), as patients who have never been hospitalized cannot feasibly be admitted to the ICU.

```
At PATIENT_TYPE = 1 and at ICU = 1 the shape will be:
(0, 21)
--------------------------------------------------------------------------------

At PATIENT_TYPE = 2 and at ICU = 1 the shape will be:
(16858, 21)
--------------------------------------------------------------------------------

At PATIENT_TYPE = 1 and at ICU = 2 the shape will be:
(0, 21)
--------------------------------------------------------------------------------

At PATIENT_TYPE = 2 and at ICU = 2 the shape will be:
(175685, 21)
--------------------------------------------------------------------------------

At PATIENT_TYPE = 1 and at ICU = 97 the shape will be:
(848544, 21)
--------------------------------------------------------------------------------

At PATIENT_TYPE = 2 and at ICU = 97 the shape will be:
(0, 21)
--------------------------------------------------------------------------------

At PATIENT_TYPE = 1 and at ICU = 99 the shape will be:
(0, 21)
--------------------------------------------------------------------------------

At PATIENT_TYPE = 2 and at ICU = 99 the shape will be:
(7488, 21)
--------------------------------------------------------------------------------
```

```
df.ICU.value_counts()
```

```
97    848544
2     175685
1      16858
99      7488
Name: ICU, dtype: int64
```

```
df['ICU'].replace (97, 2, inplace = True)
df.ICU.value_counts()
```

```
2     1024229
1       16858
99       7488
Name: ICU, dtype: int64
```

All the rest of these missing values (values of 98 & 99) can't be predicted as they will affect our analysis, which is based on true and actual real life values.

I checked the remaining missing values to constitute only 2.7% of our dataset. Hence, I did remove them since it is only the minor portion of our dataset.

```
for i in [98, 99]:
    df.replace(i , np.nan, inplace = True)
```

```
# Missing values

print(df.shape)
print(df.dropna().shape)

print(1048575 - 1019473)

print(29102 / 1048575)
```

```
(1048575, 21)
(1019473, 21)
29102
0.027753856424194742
```

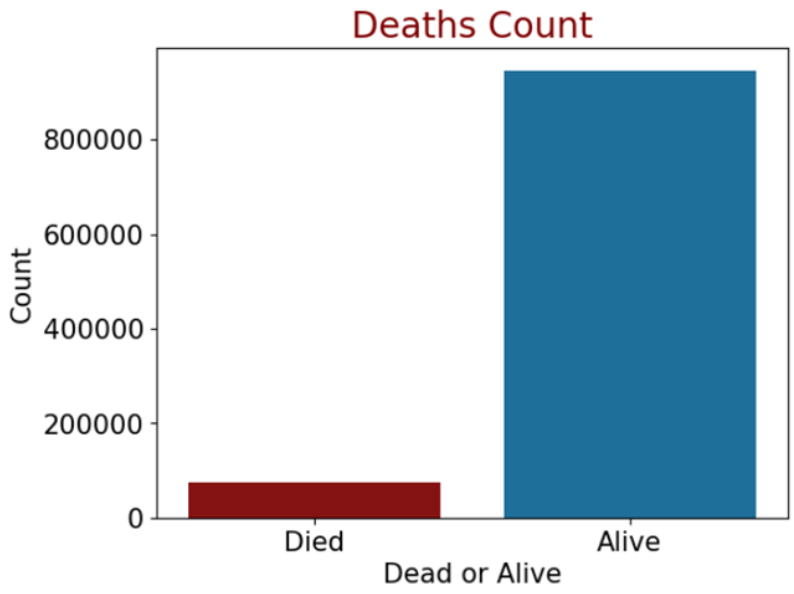
```
df.dropna(inplace = True)
```

5. Data Analysis

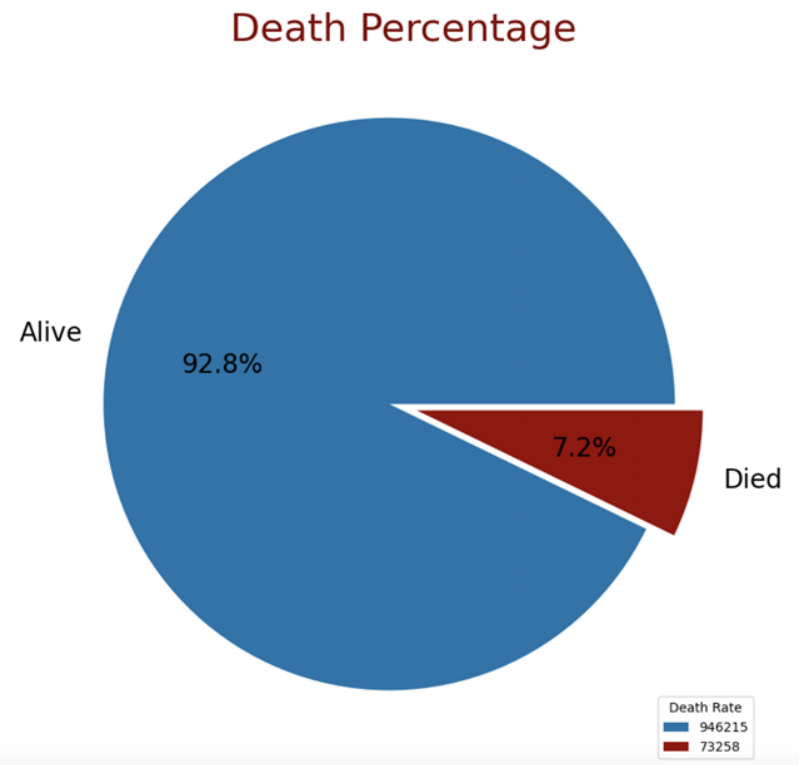
To gain insights into the dataset, I conducted a comprehensive data analysis.

Initially, I conducted an analysis by printing and visualizing the frequency counts of individuals classified as "alive" and "died," providing a comprehensive overview of the distribution within our dataset. This visual representation served as a clear snapshot, allowing me to grasp the proportion and balance between these categories, thereby aiding in our understanding of the dataset's composition and potential implications for subsequent analyses.

Bar Chart:



Pie Chart:



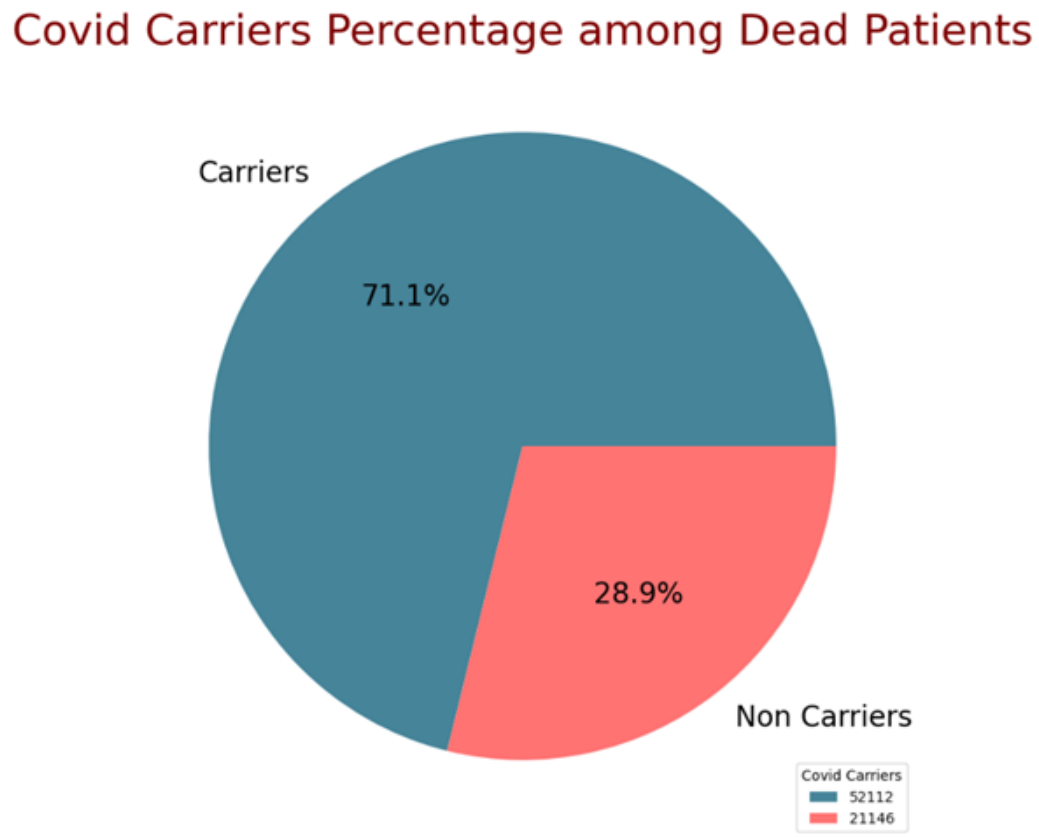
Following the initial analysis, I proceeded by augmenting the dataset's functionality. A new column titled "Covid_or_Not" was introduced within the data frame, specifically for the subset

of dead individuals (df_dead). This column was crafted to facilitate the computation of the proportion of Covid carriers within this specific group. By isolating the deceased cohort and calculating the prevalence of Covid cases among them, I aimed to glean insights into the virus's impact within this particular segment of the dataset.

Define people in covid carrier:

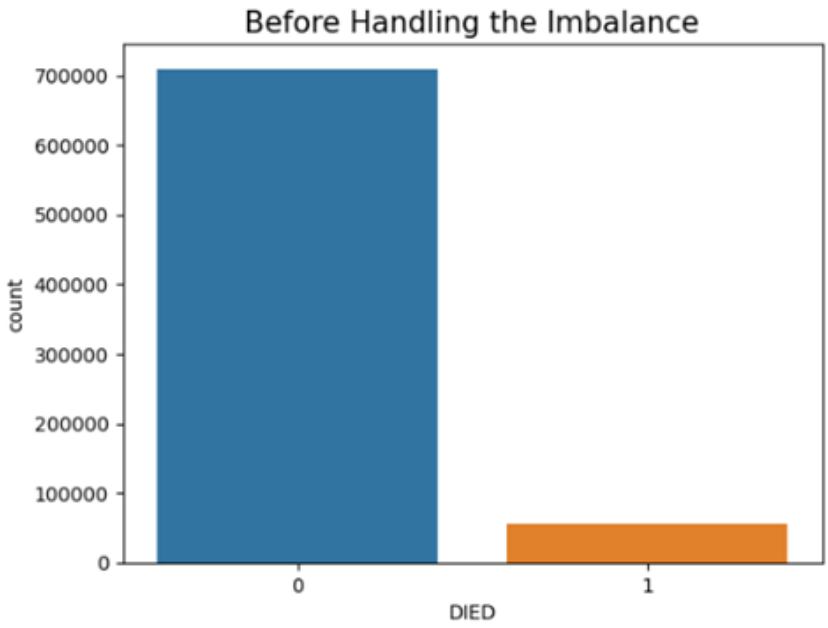
```
def Covid_or_Not(val):
    if val >= 4:
        return "Not a Covid 19 Carrier"
    else:
        return "A Covid 19 Carrier"

df_dead = df[df["DIED"] == 1]

df_dead['Covid_or_Not'] = df_dead["CLASIFFICATION_FINAL"].apply(Covid_or_Not)

print(df_dead["Covid_or_Not"].value_counts())

df_dead
```

Pie Chart:

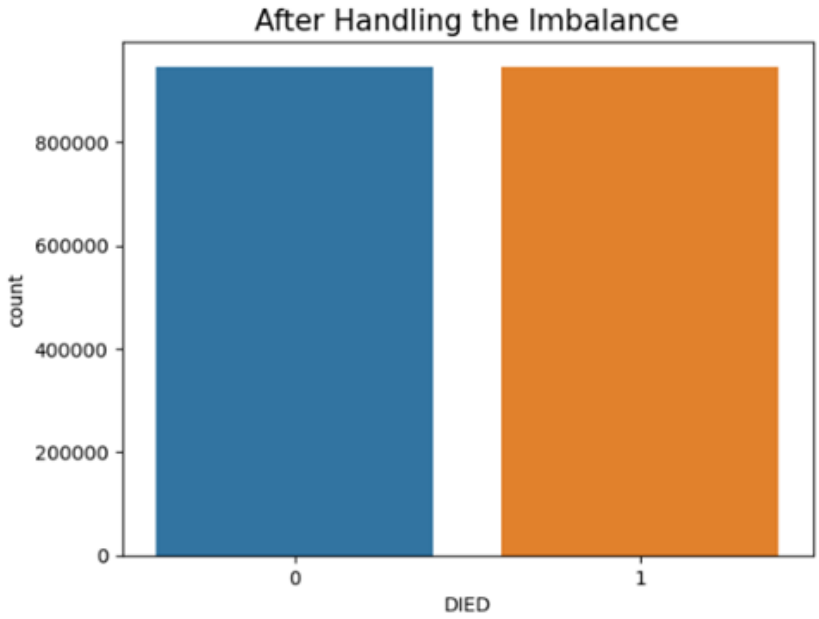


Consequently, our analysis revealed that among the deceased patients, 71.1% were identified as COVID carriers, while the remaining 28.9% did not exhibit signs of being infected with COVID.

Prior to delving into feature selection, I conducted a thorough assessment to ascertain the presence of any imbalance issues within our dataset. This preliminary examination aimed to scrutinize the distribution and representation of various classes or categories present in the data. By evaluating the balance among different groups or labels, I sought to ensure that our subsequent analyses and modeling processes are based on a dataset that maintains an equitable representation of the various factors under consideration.



As I interpreted from the above, there are significantly more individuals who are alive than those who have died, indicating an imbalance issue. To rectify this, I applied an oversampling method, achieving a balanced distribution as follows.



6. Data Mining Methods and Processes

6.1 Feature Selection

In feature selection, I will test the accuracy, F1, and AUC from the full model, filter model, and wrapper model. First of all, I set column 'DIED' as y label. Moreover, the partitioning of the dataset into training and testing subsets adheres to a distribution ratio of 75% for training and 25% for testing. Notably, the random state parameter is set to a fixed value of 42 to ensure replicability and consistency in the randomized data splitting process.

```
# Oversampling

df_num=df.copy(deep=True)

x = df_num.drop('DIED' , axis= 1)
y = df_num['DIED']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)

ros = RandomOverSampler(random_state=42)
ros.fit(x, y)
x_train, y_train = ros.fit_resample(x, y)
```

Full Model:

I build a full model using all features.

The result of accuracy, F1, and AUC from full model:

Full Model

```
# All features result:

df_all = df_num.copy(deep=True)

# Create and fit a logistic regression model with the best parameters
clf = LogisticRegression(penalty='l2', solver='newton-cg', max_iter=100, C=1.0, fit_intercept=True)
clf = clf.fit(x_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(x_test)
y_pred_proba = clf.predict_proba(x_test)[:, 1]

# Calculate accuracy, F1 score, and AUC using hold-out evaluation
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='micro')
auc = roc_auc_score(y_test, y_pred_proba, average='micro')

print('By hold-out evaluation: ')
print("Accuracy =", acc)
print("F1 =", f1)
print("AUC =", auc)
```

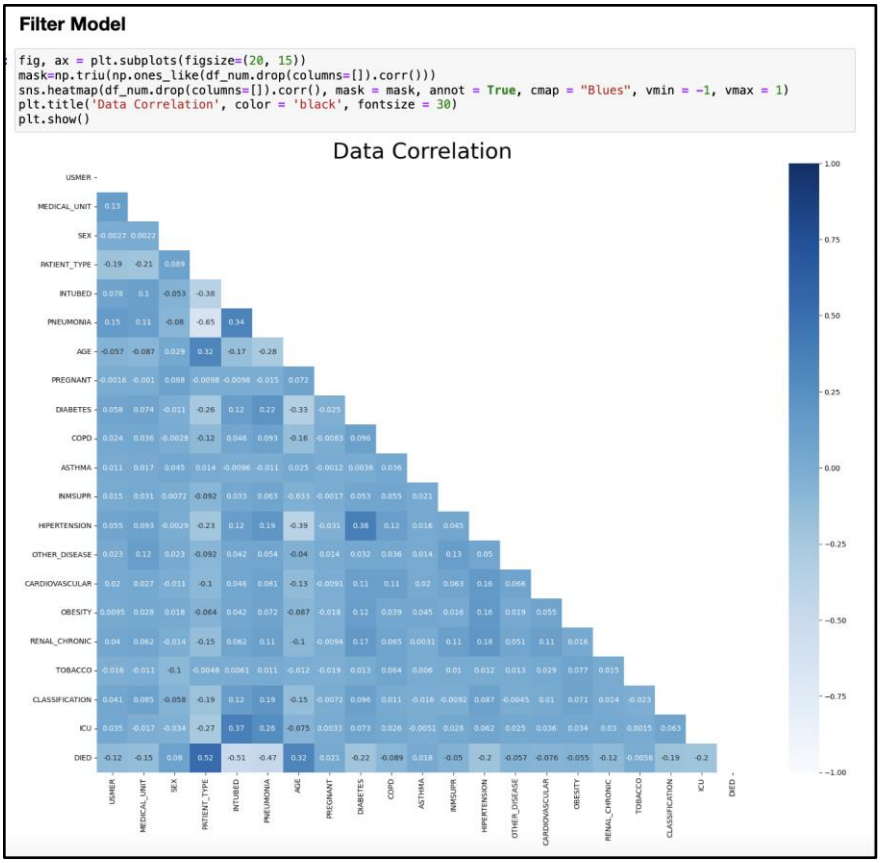
```
By hold-out evaluation:
Accuracy = 0.8995601662030298
F1 = 0.8995601662030297
AUC = 0.9615766939671342
```

Filter Model:

I built a filter model by calculating the correlation between the variables and selecting the feature over our threshold.

Filter Model

```
fig, ax = plt.subplots(figsize=(20, 15))
mask=np.triu(np.ones_like(df_num.drop(columns=[]).corr()))
sns.heatmap(df_num.drop(columns=[]).corr(), mask = mask, annot = True, cmap = "Blues", vmin = -1, vmax = 1)
plt.title('Data Correlation', color = 'black', fontsize = 30)
plt.show()
```



In the filter model, I set the threshold as 0.3, and find the correlation in the dataset. As a result, there are 4 columns which are higher than 0.3, PATIENT_TYPE, INTUBED, PNEUMONIA, and AGE respectively.

```
cor = df_num.corr()

cor_target = abs(cor["DIED"])
threshold = 0.3  # Adjust the threshold as needed
relevant_features = cor_target[cor_target > threshold]

print('\nSelected features by Pearson correlation:\n', relevant_features)
```

```
Selected features by Pearson correlation:
 PATIENT_TYPE    0.517780
INTUBED          0.505019
PNEUMONIA        0.471518
AGE              0.320668
DIED             1.000000
Name: DIED, dtype: float64
```

AS a result, I have ‘PAITNET_TYPE’, ‘INTUBED’, ‘ PNEUMONIA’, and ‘AGE’ as the selected features so I redefine the data frame called df_num_cor, and build the model.

```
# cor features:

selected_columns = ['DIED', 'PATIENT_TYPE', 'INTUBED', 'PNEUMONIA', 'AGE']

# Create a new DataFrame with the selected features
df_num_cor = df_num.copy(deep=True)
df_num_cor = df_num[selected_columns]

x = df_num_cor.drop('DIED' , axis= 1)
y = df_num_cor['DIED']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)

ros = RandomOverSampler(random_state=42)
ros.fit(x, y)
x_train, y_train = ros.fit_resample(x, y)

# Print out and display the updated dataframe as tables in HTML
display('df_num_feature:', HTML(df_num_cor.head(10).to_html()))
```

'df_num_feature:'

	DIED	PATIENT_TYPE	INTUBED	PNEUMONIA	AGE
0	1	1	2.0	1.0	65.0
1	1	1	2.0	1.0	72.0
2	1	2	1.0	2.0	55.0
3	1	1	2.0	2.0	53.0
4	1	1	2.0	2.0	68.0
5	0	2	2.0	1.0	40.0
6	0	1	2.0	2.0	64.0
7	0	1	2.0	1.0	64.0
8	0	2	2.0	2.0	37.0
9	0	2	2.0	2.0	25.0

The result of accuracy, F1, and AUC from filter model:

```
# Filter Model result

# Create and fit a logistic regression model with the best parameters
clf = LogisticRegression(penalty='l2', solver='newton-cg', max_iter=100, C=1.0, fit_intercept=True)
clf = clf.fit(x_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(x_test)
y_pred_proba = clf.predict_proba(x_test)[:, 1]

# Calculate accuracy, F1 score, and AUC using hold-out evaluation
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='micro')
auc = roc_auc_score(y_test, y_pred_proba, average='micro')

print('By hold-out evaluation: ')
print("Accuracy =", acc)
print("F1 =", f1)
print("AUC =", auc)
```

```
By hold-out evaluation:
Accuracy = 0.8971157732011347
F1 = 0.8971157732011347
AUC = 0.9570666265316232
```

Wrapper Model:

In the wrapper model, I used the forward method for feature selection. Once I identify the selected 10 features, I employ them to create a new data frame.

Wrapper Model

```
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import SequentialFeatureSelector

# search method (forward)
feature_selector = SequentialFeatureSelector(RandomForestClassifier(n_jobs=-1),
          n_features_to_select='auto',
          direction='forward',
          scoring='accuracy',
          cv=5)

x = df_num.drop('DIED', axis=1)
y = df_num['DIED']

feature_selector.fit(x, y)
selected_features= feature_selector.get_support()
print(x.columns)
print(selected_features)

# Print the selected feautres after forward method
selected_columns = x.columns[selected_features]
print("\nThe selected features after forward method:\n", selected_columns)
```

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

The selected features after forward method:
 Index(['SEX', 'INTUBED', 'PREGNANT', 'COPD', 'ASTHMA', 'INMSUPR',
       'HIPERTENSION', 'CARDIOVASCULAR', 'TOBACCO', 'CLASSIFICATION'],
      dtype='object')
```

Define new data frame called df_num_feature from the selected features:

```
# Define the columns to keep after the feature selection
selected_columns = ['DIED', 'SEX', 'INTUBED', 'PREGNANT', 'COPD', 'ASTHMA', 'INMSUPR',
       'HIPERTENSION', 'CARDIOVASCULAR', 'TOBACCO', 'CLASSIFICATION']

# Create a new DataFrame with the selected features
df_num_feature = df_num.copy(deep=True)
df_num_feature = df_num_feature[selected_columns]

x = df_num_feature.drop('DIED' , axis= 1)
y = df_num_feature['DIED']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)

ros = RandomOverSampler(random_state=42)
ros.fit(x, y)
x_train, y_train = ros.fit_resample(x, y)

# Print out and display the updated dataframe as tables in HTML
display('df_num_feature:', HTML(df_num_feature.head(10).to_html()))
```

'df_num_feature:'

	DIED	SEX	INTUBED	PREGNANT	COPD	ASTHMA	INMSUPR	HIPERTENSION	CARDIOVASCULAR	TOBACCO	CLASSIFICATION
0	1	1	2.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	3
1	1	2	2.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	5
2	1	2	1.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	3
3	1	1	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	7
4	1	2	2.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	3
5	0	1	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	3
6	0	1	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	3
7	0	1	2.0	2.0	2.0	2.0	1.0	1.0	2.0	2.0	3
8	0	1	2.0	2.0	2.0	2.0	2.0	1.0	2.0	2.0	3
9	0	1	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	3

The result of accuracy, F1, and AUC from wrapper model:

```
# Wrapper Model result

# Create and fit a logistic regression model with the best parameters
clf = LogisticRegression(penalty='l2', solver='newton-cg', max_iter=100, C=1.0, fit_intercept=True)
clf = clf.fit(x_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(x_test)
y_pred_proba = clf.predict_proba(x_test)[:, 1]

# Calculate accuracy, F1 score, and AUC using hold-out evaluation
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='micro')
auc = roc_auc_score(y_test, y_pred_proba, average='micro')

print('By hold-out evaluation: ')
print("Accuracy =", acc)
print("F1 =", f1)
print("AUC =", auc)
```

```
By hold-out evaluation:
Accuracy = 0.7750412957244702
F1 = 0.7750412957244702
AUC = 0.8546000401666558
```

6.2 Data Mining Methods and process

Resampling:

Initially, our approach involved resampling the entire dataset to address any imbalances. However, I've since revised our strategy, recognizing that oversampling is intended to be applied only to the training set, not the entire dataset. This adjustment allows me to concentrate our efforts on improving the balance within the training data specifically.

```
# get features and labels
x = df_num.drop('DIED', axis=1)
y = df_num['DIED']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)

print('₩nOriginal dataset shape {}'.format(Counter(y_train)))
ros = RandomOverSampler(random_state=42)
ros.fit(x, y)
x_train, y_train = ros.fit_resample(x, y)
print('After undersampling dataset shape {}'.format(Counter(y_train)))
```

```
Original dataset shape Counter({0: 709606, 1: 54998})
After undersampling dataset shape Counter({1: 946215, 0: 946215})
```

Consequently, the values are now balanced, enabling me to construct our classification models without encountering overfitting issues.

Best Parameters:

Systematically applying various parameter optimization methods across all models, I aimed to identify the most effective parameter configurations. The example below illustrates our approach with the logistic regression model. The loop tests out different pairs of parameters, presenting in the output the best ones according to the evaluation metrics.

```python
# 01. Logistic Regression

# Define the parameter combinations to test
penalties = ['l1', 'l2']
solvers = ['newton-cg', 'lbfgs', 'liblinear', 'saga']
max_iters = [100, 150, 200]
Cs = [0.1, 1.0, 10.0]
fit_intercepts = [True, False]

# Initialize variables to track the best F1 score and AUC
best_f1 = 0
best_auc = 0
best_params = {}

# Iterate through parameter combinations
for penalty in penalties:
    for solver in solvers:
        # Check if the combination is valid
        if (penalty == 'l1' and solver not in ['liblinear', 'saga']) or \
           (penalty == 'elasticnet' and solver != 'saga') or \
           (penalty == 'l2' and solver == 'lbfgs'):
            continue

        for max_iter in max_iters:
            # liblinear solver doesn't support elasticnet penalty
            if solver == 'liblinear' and penalty == 'l1':
                continue

            for C in Cs:
                for fit_intercept in fit_intercepts:
                    # Create and fit a logistic regression model
                    clf = LogisticRegression(
                        penalty=penalty,
                        solver=solver,
                        max_iter=max_iter,
                        C=C,
                        fit_intercept=fit_intercept,
                        random_state=42
                    )
                    clf = clf.fit(x_train, y_train)
                    y_pred = clf.predict(x_test)
                    f1 = f1_score(y_test, y_pred, average='micro')

                    # Use LabelBinarizer to handle multiclass classification for AUC
                    lb = LabelBinarizer()
                    y_test_bin = lb.fit_transform(y_test)
                    y_pred_proba = clf.predict_proba(x_test)[:, 1]  # Extracting the relevant column
                    auc = roc_auc_score(y_test_bin, y_pred_proba, average='macro')

                    # Update best F1 score, AUC, and parameters if a better model is found
                    if f1 > best_f1 and auc > best_auc:
                        best_f1 = f1
                        best_auc = auc
                        best_params = {
                            'penalty': penalty,
                            'solver': solver,
                            'max_iter': max_iter,
                            'C': C,
                            'fit_intercept': fit_intercept
                            }

# Print the best parameters and corresponding F1 score and AUC
print("Best Parameters:")
print(best_params)
```

```
Best Parameters:
{'penalty': 'l2', 'solver': 'liblinear', 'max_iter': 100, 'C': 0.1, 'fit_intercept': True}
```

Then I constructed a model with the classifier using the best parameters obtained from the above process:

```
# Create and fit a logistic regression model with the best parameters
clf = LogisticRegression(penalty='l2', solver='liblinear', max_iter=100, C=0.1, fit_intercept=True, random_state=42)
clf = clf.fit(x_train, y_train)
```

I applied the same steps when I was building each model.

Models:

Previously, for certain models, I would redefine our training and testing sets each time I built them. I refined this process by defining our training and testing sets only once and utilizing the same sets consistently when constructing models. To ensure reproducibility and consistency in our analyses, I incorporated the parameter 'random_state=42' into our code. This parameter assignment guarantees the consistent use of the same pairs of the dataset across executions, ensuring identical results each time the code is run. This adjustment maintains stability in our experimentation process, facilitates easier comparison of outcomes, and ensures the reliability of our findings.

Following are the 6 classification models I performed:

1. Logistic Regression

```
clf = LogisticRegression(penalty='l2', solver='liblinear', max_iter=100, C=0.1, fit_intercept=True, random_state=42)
clf = clf.fit(x_train, y_train)
```

2. Decision Tree

```
clf = DecisionTreeClassifier(criterion='gini', max_depth=None, min_samples_leaf=1, random_state=42)
clf.fit(x_train, y_train)
```

3. SVM

```
clf = SVC(C=0.1, kernel='linear', gamma=0.1, random_state=42)
clf = clf.fit(x_train, y_train)
```

4. Random Forest

```
clf = RandomForestClassifier(n_estimators=100, max_depth=None, min_samples_split=2, random_state=42)
clf = clf.fit(x_train, y_train)
```

5. Bagging

```
clf = BaggingClassifier(n_estimators=50, max_samples=1.0, max_features=0.6, random_state=42)
clf = clf.fit(x_train, y_train)
```

6. Gradient boosting

```
clf = GradientBoostingClassifier(learning_rate=0.1, n_estimators=100, max_depth=5, random_state=42)
clf = clf.fit(x_train, y_train)
```

7. Evaluations and Results

7.1 Evaluation

I used **hold-out evaluation** since I have **1000k** rows in our dataset.

I calculated **accuracy**, **f1 score**, and **AUC** as our evaluation metrics.

```
# Make predictions on the test set
y_pred = clf.predict(x_test)
y_pred_proba = clf.predict_proba(x_test)[:, 1]

# Calculate accuracy, F1 score, and AUC using hold-out evaluation
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='micro')
auc = roc_auc_score(y_test, y_pred_proba, average='micro')
```

Following are the output from each model:

1. Logistic Regression

```
By hold-out evaluation:
Accuracy = 0.8994149935849397
F1 = 0.8994149935849397
AUC = 0.9617132112972513
```

2. Decision Tree

```
By hold-out evaluation:
Accuracy = 0.9481066744092063
F1 = 0.9481066744092063
AUC = 0.9929233205629101
```

3. SVM

```
By hold-out evaluation:
Accuracy = 0.5005558984417934
F1 = 0.5005558984417934
AUC = 0.16319166135420335
```

4. Random Forest

```
By hold-out evaluation:
Accuracy = 0.9546847654235396
F1 = 0.9546847654235396
AUC = 0.9859147938002153
```

5. Bagging

```
By hold-out evaluation:
Accuracy = 0.9315927864250869
F1 = 0.9315927864250869
AUC = 0.9767590522449515
```

6. Gradient Boosting

```
By hold-out evaluation:
Accuracy = 0.9183928405353534
F1 = 0.9183928405353534
AUC = 0.968408217187078
```

7.2 Result

As a result, the best models are the decision tree, and random forest.

Decision Tree is the best model according to **AUC**.

Random Forest is the best model according to **accuracy & F1 score**.

Category	**Logistic Regression**	**Decision Tree**	**SVM**	**Random Forest**	**Bagging**	**Gradient Boosting**
Accuracy	0.8994	0.9481	0.5006	0.9547	0.9315	0.9183
F1	0.8994	0.9481	0.5006	0.9547	0.9315	0.9183
AUC	0.9617	0.9929	0.1632	0.9859	0.9767	0.9684

Top Features:

```
# Assuming x_train is a DataFrame with column names
feature_names = x_train.columns

# Fit the Random Forest model
clf = RandomForestClassifier(n_estimators=100, max_depth=None, min_samples_split=2, random_state=42)
clf = clf.fit(x_train, y_train)

# Get feature importances based on correlation with the target variable "DIED"
feature_importances = pd.Series(clf.feature_importances_, index=feature_names)

# Sort features based on importance
top5_features = feature_importances.abs().nlargest(5).index

# Print the top 5 features and their importance scores
print("Top 5 Features:")
i = 1
for feature in top5_features:
    print(f"{i}. {feature}")
    i += 1
```

```
Top 5 Features:
1. PATIENT_TYPE
2. AGE
3. PNEUMONIA
4. INTUBED
5. MEDICAL_UNIT
```

The top features from the model are the patient_type, age, pneumonia, intubed, and medical_unit.

8. Conclusion and Future Work

8.1. Conclusions

Here are the best models I got:

Random Forest: Accuracy of 95.47%

Decision Tree: AUC of 99.29%

The Random Forest model exhibited an impressive accuracy rate of 95.47%, showcasing its robustness in predicting outcomes within our dataset. Meanwhile, the Decision Tree model achieved an exceptional Area Under the Curve (AUC) value of 99.29%, affirming its proficiency in distinguishing between classes or categories within the data.

The top 4 features for prediction are:

Patient Type = Hospitalized

INTUBED = Connected to the ventilator

PNEUMONIA = Have air sacs inflammation

AGE = Old

In conclusion, among COVID patients, the presence of certain factors increases the likelihood of being at high risk. Specifically:

Hospitalized: If an individual is classified as hospitalized, it strongly suggests an escalated risk profile within the dataset, indicating a higher probability of severe medical conditions or adverse outcomes.

Connected to the ventilator: Being connected to a ventilator signifies an increased level of risk, implying a critical condition or a higher likelihood of adverse health outcomes for individuals requiring this medical intervention.

Have air sacs inflammation: The presence of air sac inflammation implies an elevated risk level, indicating a higher probability of severe medical conditions or adverse outcomes associated with this inflammatory condition.

Old: Advanced age correlates with an escalated risk profile, suggesting a higher likelihood of adverse health outcomes or a more severe prognosis for individuals in older age groups within the dataset.

8.2. Challenges

I encountered a significant challenge during our project related to the substantial computational resources required, as certain codes necessitated several hours to run each time. This was particularly evident in the context of using Support Vector Machines (SVM).

8.3. Potential Improvements or Future Work

In the future, with improved computational power to execute n-fold cross-validation, I anticipate achieving more reliable outcomes.