Project 3 : Supervised Machine Learning - Classification

Objective

The main objective of my analysis is to focus on prediction that the person would be dead or alive based on the covid and health conditions given in the dataset.

Dataset

The dataset was provided by the Mexican government (link). This dataset contains an enormous number of anonymized patient-related information including pre-conditions. The raw dataset consists of 21 unique features and 1,048,576 unique patients. In the Boolean features, 1 means "yes" and 2 means "no". values as 97 and 99 are missing data.

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 returned home and 2 for hospitalization. pneumonia: whether the patient already have 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 other disease or not. obesity: whether the patient is obese or not. tobacco: whether the patient is a tobacco user. usmer: 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.

Dependant Variable :

alive : it will be created during EDA through Date_died, Whether Patient died or not.

Import Packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

from sklearn.preprocessing import OrdinalEncoder
```

```python
from sklearn.feature_selection import chi2, SelectFromModel,
SelectKBest
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, roc_auc_score,
ConfusionMatrixDisplay, confusion_matrix,
precision_recall_fscore_support
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE, ADASYN

#Load the dataset
data= pd.read_csv("Covid Data.csv")
data
```

```
         USMER  MEDICAL_UNIT  SEX  PATIENT_TYPE   DATE_DIED
INTUBED  \
0            2             1    1             1  03/05/2020         97

1            2             1    2             1  03/06/2020         97

2            2             1    2             2  09/06/2020          1

3            2             1    1             1  12/06/2020         97

4            2             1    2             1  21/06/2020         97

...        ...           ...  ...           ...         ...        ...

1048570      2            13    2             1  9999-99-99         97

1048571      1            13    2             2  9999-99-99          2

1048572      2            13    2             1  9999-99-99         97

1048573      2            13    2             1  9999-99-99         97

1048574      2            13    2             1  9999-99-99         97


         PNEUMONIA  AGE  PREGNANT  DIABETES  ...  ASTHMA  INMSUPR  \
0                1   65         2         2  ...       2        2
1                1   72        97         2  ...       2        2
2                2   55        97         1  ...       2        2
3                2   53         2         2  ...       2        2
4                2   68        97         1  ...       2        2
...            ...  ...       ...       ...  ...     ...      ...
1048570          2   40        97         2  ...       2        2
1048571          2   51        97         2  ...       2        2
```

```
1048572          2   55         97          2  ...       2        2
1048573          2   28         97          2  ...       2        2
1048574          2   52         97          2  ...       2        2

         HIPERTENSION  OTHER_DISEASE  CARDIOVASCULAR  OBESITY
RENAL_CHRONIC  \
0                   1              2               2        2
2
1                   1              2               2        1
1
2                   2              2               2        2
2
3                   2              2               2        2
2
4                   1              2               2        2
2
...               ...            ...             ...      ...
...
1048570             2              2               2        2
2
1048571             1              2               2        2
2
1048572             2              2               2        2
2
1048573             2              2               2        2
2
1048574             2              2               2        2
2

         TOBACCO  CLASIFFICATION_FINAL  ICU
0              2                     3   97
1              2                     5   97
2              2                     3    2
3              2                     7   97
4              2                     3   97
...          ...                   ...  ...
1048570        2                     7   97
1048571        2                     7    2
1048572        2                     7   97
1048573        2                     7   97
1048574        2                     7   97

[1048575 rows x 21 columns]
```

EDA

EDA consists of:

- Look for missing values, we already know 97-99 are the missing value

- Fill or remove missing values, here we have 1st replaced the missing values of pregant for man as '2', ICU and INTUBED as '2' where PATIENT_TYPE was 2 as patient were sent home, rest were dropped.
- See the value count of each column to get details of distribution.
- Add a new dependable classification feature 'Alive', such that where DEATH_DATE is '9999-99-99' the person is alive i.e '0' and others mark as not alive i.e '1'

- Make a heatmap to get a better understanding of relationship between dependant and independent features. As it shows the correlation between each feature.
- Create ordinal encoding for age as it would create better relation with dependant variable.
- Using Count plot of different feature with hue as 'Alive' to get a clear understanding of distribution throguh visualization.
- Drop the unnecessary column such as 'AGE', 'DEATH_DATE' as it will create unnecessary noise.
- Get feature importance through chi as data is unbalanced.

Observation

- There are no null values in any column
- The dataset is very unbalanced as almost every column has a majority class with larger imbalance ratio.

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 21 columns):
 #   Column                Non-Null Count    Dtype 
---  ------                --------------    ----- 
 0   USMER                 1048575 non-null  int64 
 1   MEDICAL_UNIT          1048575 non-null  int64 
 2   SEX                   1048575 non-null  int64 
 3   PATIENT_TYPE          1048575 non-null  int64 
 4   DATE_DIED             1048575 non-null  object
 5   INTUBED               1048575 non-null  int64 
 6   PNEUMONIA             1048575 non-null  int64 
 7   AGE                   1048575 non-null  int64 
 8   PREGNANT              1048575 non-null  int64 
 9   DIABETES              1048575 non-null  int64 
 10  COPD                  1048575 non-null  int64 
 11  ASTHMA                1048575 non-null  int64 
 12  INMSUPR               1048575 non-null  int64 
 13  HIPERTENSION          1048575 non-null  int64 
 14  OTHER_DISEASE         1048575 non-null  int64 
 15  CARDIOVASCULAR        1048575 non-null  int64 
 16  OBESITY               1048575 non-null  int64 
 17  RENAL_CHRONIC         1048575 non-null  int64 
 18  TOBACCO               1048575 non-null  int64 
 19  CLASIFFICATION_FINAL  1048575 non-null  int64 
```

```
 20  ICU                      1048575 non-null  int64
dtypes: int64(20), object(1)
memory usage: 168.0+ MB
```

```python
for column in data.columns:
    print(data[column].value_counts(), end="\n\n")
```

```
USMER
2    662903
1    385672
Name: count, dtype: int64

MEDICAL_UNIT
12    602995
4     314405
6      40584
9      38116
3      19175
8      10399
10      7873
5       7244
11      5577
13       996
7        891
2        169
1        151
Name: count, dtype: int64

SEX
1    525064
2    523511
Name: count, dtype: int64

PATIENT_TYPE
1    848544
2    200031
Name: count, dtype: int64

DATE_DIED
9999-99-99    971633
06/07/2020      1000
07/07/2020       996
13/07/2020       990
16/06/2020       979
               ...
24/11/2020         1
17/12/2020         1
08/12/2020         1
16/03/2021         1
22/04/2021         1
```

```
Name: count, Length: 401, dtype: int64

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

PNEUMONIA
2     892534
1     140038
99     16003
Name: count, dtype: int64

AGE
30     27010
31     25927
28     25313
29     25134
34     24872
       ...
114        2
116        2
111        1
121        1
113        1
Name: count, Length: 121, dtype: int64

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

DIABETES
2     920248
1     124989
98      3338
Name: count, dtype: int64

COPD
2     1030510
1       15062
98       3003
Name: count, dtype: int64

ASTHMA
2     1014024
```

```
1       31572
98       2979
Name: count, dtype: int64

INMSUPR
2     1031001
1       14170
98       3404
Name: count, dtype: int64

HIPERTENSION
2     882742
1     162729
98      3104
Name: count, dtype: int64

OTHER_DISEASE
2     1015490
1       28040
98       5045
Name: count, dtype: int64

CARDIOVASCULAR
2     1024730
1       20769
98       3076
Name: count, dtype: int64

OBESITY
2     885727
1     159816
98      3032
Name: count, dtype: int64

RENAL_CHRONIC
2     1026665
1       18904
98       3006
Name: count, dtype: int64

TOBACCO
2     960979
1      84376
98      3220
Name: count, dtype: int64

CLASIFFICATION_FINAL
7    499250
3    381527
6    128133
```

```
5      26091
1       8601
4       3122
2       1851
Name: count, dtype: int64

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

```
data['Alive'] = data['DATE_DIED'].apply(lambda x: 1 if x =='9999-99-
99' else 2)

data['Alive'].value_counts()
```

```
Alive
1    971633
2     76942
Name: count, dtype: int64
```

```
mask = data['PREGNANT'].isin([97, 98, 99])
data['PREGNANT'] = data['PREGNANT'].mask(mask & (data['SEX'] == 1),
None)
data['PREGNANT'] = data['PREGNANT'].mask(mask & (data['SEX'] == 2), 2)
data.loc[data['PATIENT_TYPE'] == 1, ['INTUBED', 'ICU']] = 2
refined_data = data.replace(to_replace=[97,98,99], value=None)
refined_data['AGE']= data['AGE']

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

```
        USMER MEDICAL_UNIT SEX PATIENT_TYPE   DATE_DIED INTUBED
PNEUMONIA  \
0           2            1   1            1  03/05/2020       2
1
1           2            1   2            1  03/06/2020       2
1
2           2            1   2            2  09/06/2020       1
2
3           2            1   1            1  12/06/2020       2
2
4           2            1   2            1  21/06/2020       2
2
...       ...          ...  ..          ...         ...     ...
...
1048570     2           13   2            1  9999-99-99       2
2
```

```
1048571      1           13   2             2  9999-99-99       2
2
1048572      2           13   2             1  9999-99-99       2
2
1048573      2           13   2             1  9999-99-99       2
2
1048574      2           13   2             1  9999-99-99       2
2

         AGE  PREGNANT DIABETES  ... INMSUPR HIPERTENSION
OTHER_DISEASE  \
0         65       2.0        2  ...       2            1
2
1         72       2.0        2  ...       2            1
2
2         55       2.0        1  ...       2            2
2
3         53       2.0        2  ...       2            2
2
4         68       2.0        1  ...       2            1
2
...      ...       ...      ...  ...     ...          ...            ..
.
1048570   40       2.0        2  ...       2            2
2
1048571   51       2.0        2  ...       2            1
2
1048572   55       2.0        2  ...       2            2
2
1048573   28       2.0        2  ...       2            2
2
1048574   52       2.0        2  ...       2            2
2

        CARDIOVASCULAR OBESITY RENAL_CHRONIC TOBACCO
CLASIFFICATION_FINAL ICU  \
0                    2       2             2       2
3   2
1                    2       1             1       2
5   2
2                    2       2             2       2
3   2
3                    2       2             2       2
7   2
4                    2       2             2       2
3   2
...                ...     ...           ...     ...
...  ..
1048570              2       2             2       2
```

```
7   2
1048571              2       2             2       2
7   2
1048572              2       2             2       2
7   2
1048573              2       2             2       2
7   2
1048574              2       2             2       2
7   2

        Alive
0           2
1           2
2           2
3           2
4           2
...       ...
1048570     1
1048571     1
1048572     1
1048573     1
1048574     1

[1019666 rows x 22 columns]
```

```
for column in refined_data.columns:
    print(refined_data[column].value_counts(), end="\n\n")
```

```
USMER
2    655076
1    364590
Name: count, dtype: int64

MEDICAL_UNIT
12    587800
4     306502
6      37675
9      36983
3      18566
8      10070
10      7511
5       7045
11      5541
7        856
13       808
2        158
1        151
Name: count, dtype: int64

SEX
```

```
2    510596
1    509070
Name: count, dtype: int64

PATIENT_TYPE
1    830385
2    189281
Name: count, dtype: int64

DATE_DIED
9999-99-99    946356
06/07/2020       966
07/07/2020       963
13/07/2020       947
16/06/2020       937
               ...
10/09/2020         1
19/04/2021         1
30/10/2020         1
24/03/2021         1
22/04/2021         1
Name: count, Length: 393, dtype: int64

INTUBED
2    986730
1     32936
Name: count, dtype: int64

PNEUMONIA
2    883254
1    136412
Name: count, dtype: int64

AGE
30     26435
31     25386
28     24708
29     24574
32     24375
       ...
115        2
119        2
111        1
121        1
113        1
Name: count, Length: 121, dtype: int64

PREGNANT
2.0    1011838
1.0       7828
```

```
Name: count, dtype: int64

DIABETES
2    898137
1    121529
Name: count, dtype: int64

COPD
2    1005398
1      14268
Name: count, dtype: int64

ASTHMA
2    989337
1     30329
Name: count, dtype: int64

INMSUPR
2    1006143
1      13523
Name: count, dtype: int64

HIPERTENSION
2    861207
1    158459
Name: count, dtype: int64

OTHER_DISEASE
2    992670
1     26996
Name: count, dtype: int64

CARDIOVASCULAR
2    999677
1     19989
Name: count, dtype: int64

OBESITY
2    863614
1    156052
Name: count, dtype: int64

RENAL_CHRONIC
2    1001446
1      18220
Name: count, dtype: int64

TOBACCO
2    937307
1     82359
```

```
Name: count, dtype: int64

CLASIFFICATION_FINAL
7    488131
3    375920
6    117178
5     25175
1      8397
4      3082
2      1783
Name: count, dtype: int64

ICU
2    1003258
1      16408
Name: count, dtype: int64

Alive
1    946356
2     73310
Name: count, dtype: int64
```

```
# Define age groups
bins = [0, 12, 18, 35, 50, 65, 80,125]
labels = ['Child', 'Teen', 'Young Adult', 'Adult', 'Middle-aged',
'Senior', 'Elderly']

# Apply binning
refined_data['Age Group'] = pd.cut(refined_data['AGE'], bins=bins,
labels=labels, right=False)
# Ordinal Encoding
encoder = OrdinalEncoder(categories=[labels])  # Preserve order
refined_data['Age Group'] = encoder.fit_transform(refined_data[['Age
Group']])

refined_data
```

```
        USMER MEDICAL_UNIT SEX PATIENT_TYPE   DATE_DIED INTUBED
PNEUMONIA  \
0           2            1   1            1  03/05/2020       2
1
1           2            1   2            1  03/06/2020       2
1
2           2            1   2            2  09/06/2020       1
2
3           2            1   1            1  12/06/2020       2
2
4           2            1   2            1  21/06/2020       2
2
```

```
...        ...          ...  ..           ...          ...     ...
...
1048570     2           13   2             1  9999-99-99       2
2
1048571     1           13   2             2  9999-99-99       2
2
1048572     2           13   2             1  9999-99-99       2
2
1048573     2           13   2             1  9999-99-99       2
2
1048574     2           13   2             1  9999-99-99       2
2

         AGE  PREGNANT DIABETES  ... HIPERTENSION OTHER_DISEASE  \
0         65       2.0        2  ...            1             2
1         72       2.0        2  ...            1             2
2         55       2.0        1  ...            2             2
3         53       2.0        2  ...            2             2
4         68       2.0        1  ...            1             2
...      ...       ...      ...  ...          ...           ...
1048570   40       2.0        2  ...            2             2
1048571   51       2.0        2  ...            1             2
1048572   55       2.0        2  ...            2             2
1048573   28       2.0        2  ...            2             2
1048574   52       2.0        2  ...            2             2

        CARDIOVASCULAR OBESITY RENAL_CHRONIC TOBACCO
CLASIFFICATION_FINAL ICU  \
0                    2       2             2       2
3   2
1                    2       1             1       2
5   2
2                    2       2             2       2
3   2
3                    2       2             2       2
7   2
4                    2       2             2       2
3   2
...                ...     ...           ...     ...
...  ..
1048570              2       2             2       2
7   2
1048571              2       2             2       2
7   2
1048572              2       2             2       2
7   2
1048573              2       2             2       2
7   2
1048574              2       2             2       2
```

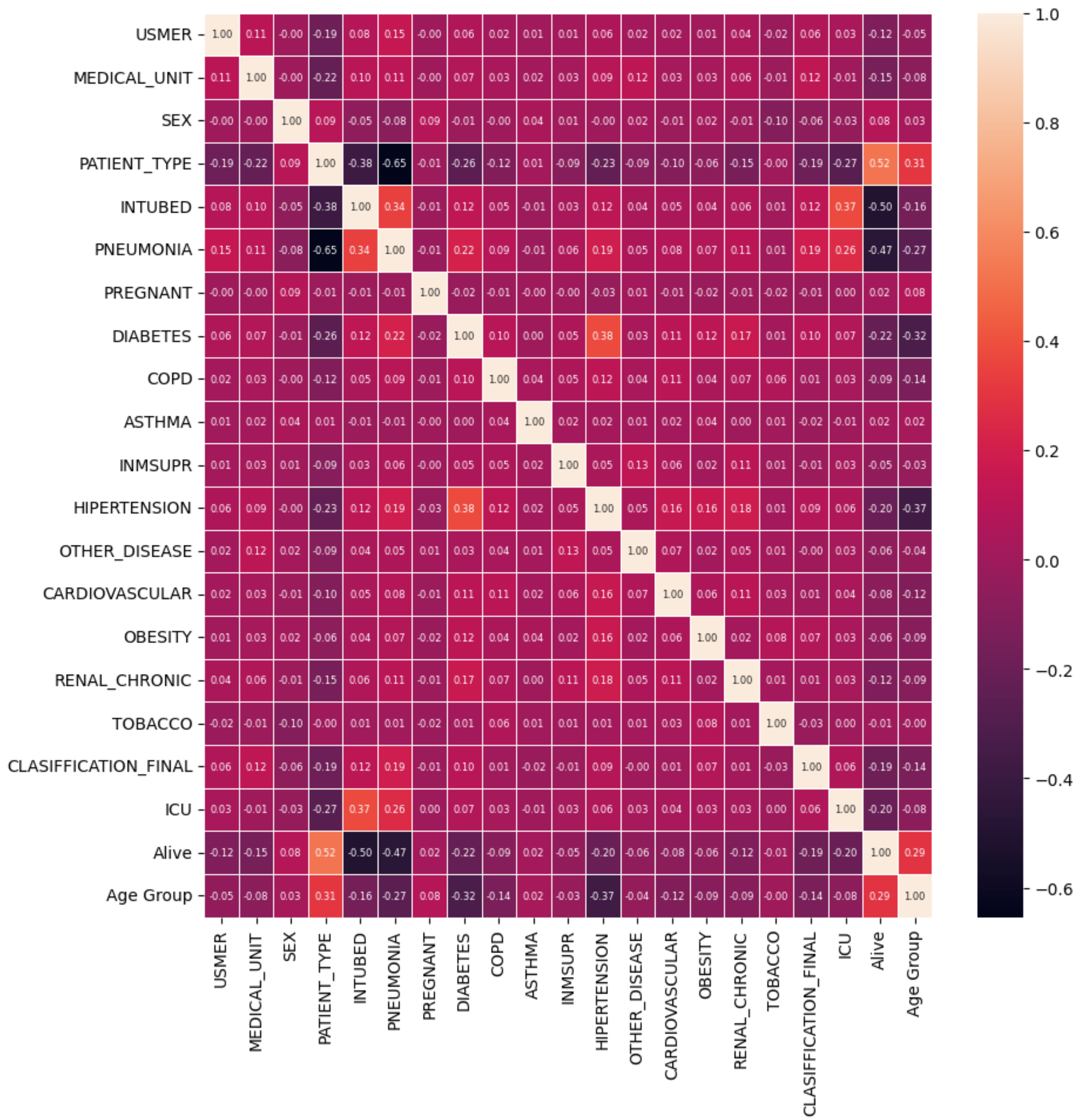
```
7    2

        Alive  Age Group
0           2        5.0
1           2        5.0
2           2        4.0
3           2        4.0
4           2        5.0
...       ...        ...
1048570     1        3.0
1048571     1        4.0
1048572     1        4.0
1048573     1        2.0
1048574     1        4.0

[1019666 rows x 23 columns]
```

```
refined_data.drop(columns=['DATE_DIED','AGE'], axis=1,inplace=True)
```

Observations

- With respect to 'Alive' we have 'PNEUMONIA', 'INTUBED','PATIENT_TYPE','Age Group' are most prominent features.
- But as the dataset is unbalanced corr might not help much in feature selection.

```
plt.figure(figsize=(10, 10))
sns.heatmap(refined_data.corr(method='spearman'), annot=True,
fmt=".2f",linewidths=0.5, annot_kws={"size": 6})
plt.show()
```



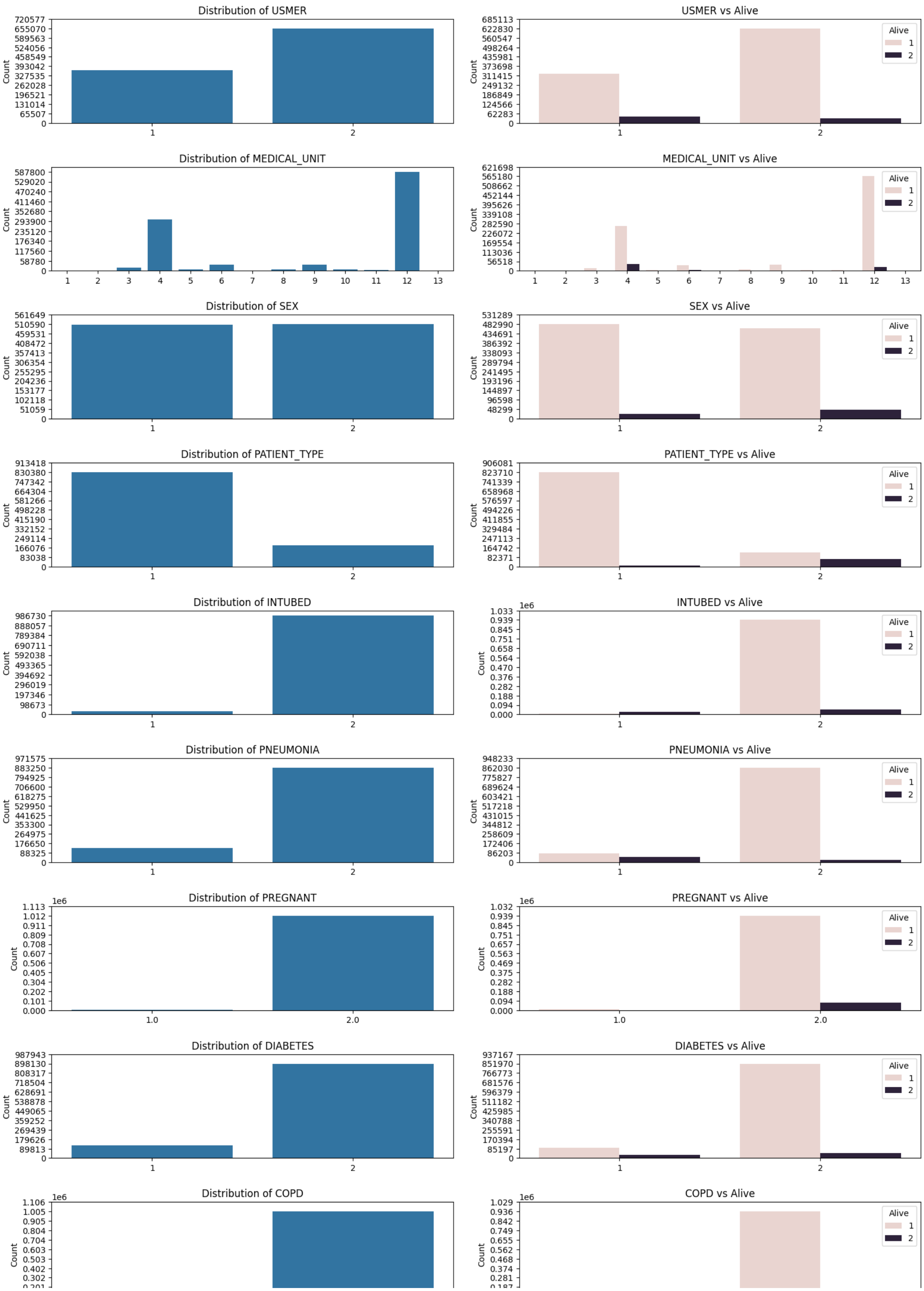

```
discrete_features = refined_data.columns
```

Observation

- We could see it is diificult to make prediction for 'Alive' as value count is almost similar in ratio with all classes.
- 'PNEUMONIA', 'INTUBED', 'Age Group' shows a major differentiation.

```
# Define the number of features and set a dynamic height
num_features = len(discrete_features)
fig, ax = plt.subplots(num_features, 2, figsize=(16, num_features *
```

```
2.5))  # Increase height dynamically

for i, feature in enumerate(discrete_features):
    # Plot countplot without hue
    sns.countplot(ax=ax[i, 0], x=feature, data=refined_data)
    ax[i, 0].set_title(f"Distribution of {feature}", fontsize=12)
    ax[i, 0].set_xlabel("")
    ax[i, 0].set_ylabel("Count", fontsize=10)

    # Plot countplot with 'Alive' hue
    sns.countplot(ax=ax[i, 1], x=feature, hue='Alive',
data=refined_data)
    ax[i, 1].set_title(f"{feature} vs Alive", fontsize=12)
    ax[i, 1].set_xlabel("")
    ax[i, 1].set_ylabel("Count", fontsize=10)

    # Set y-axis ticks dynamically
    max_count = refined_data[feature].value_counts().max()
    tick_interval = max(1, max_count // 10)  # Set dynamic interval
for better spacing
    ax[i, 0].set_yticks(range(0, max_count + tick_interval,
tick_interval))

    # Set y-axis tick spacing dynamically for the second plot (based
on hue)
    max_count_2 = refined_data.groupby(feature)
['Alive'].value_counts().max()
    tick_interval_2 = max(1, max_count_2 // 10)
    ax[i, 1].set_yticks(range(0, max_count_2 + tick_interval_2,
tick_interval_2))

plt.subplots_adjust(hspace=0.5, wspace=0.3)  # Increase spacing
between subplots
fig.tight_layout(pad=2)
plt.show()
```


Distribution of USMER
USMER vs Alive
Distribution of MEDICAL_UNIT
MEDICAL_UNIT vs Alive
Distribution of SEX
SEX vs Alive
Distribution of PATIENT_TYPE
PATIENT_TYPE vs Alive
Distribution of INTUBED
INTUBED vs Alive
Distribution of PNEUMONIA
PNEUMONIA vs Alive
Distribution of PREGNANT
PREGNANT vs Alive
Distribution of DIABETES
DIABETES vs Alive
Distribution of COPD
COPD vs Alive
Count
Alive
1
2
1e6

Observation

- We could see the top features

  Age Group : 43822.594537976875
  MEDICAL_UNIT : 37796.72489121804
  PATIENT_TYPE : 34859.798318027286
  CLASIFFICATION_FINAL : 26083.104291404423
  PNEUMONIA : 14077.887325773387
  INTUBED : 4130.213961265305
  HIPERTENSION : 2994.5650815717036
  DIABETES : 2649.2413529553733
  USMER : 1973.9843896470545
  SEX : 1085.1027848188737

```
x = refined_data.drop(['Alive'], axis=1).astype(int)
y = refined_data['Alive'].astype(int)
y = y.apply(lambda v: v-1)
best_features = SelectKBest(chi2, k=10)
features_ranking = best_features.fit(x, y)
ranking_dictionary = {}
for i in range(len(features_ranking.scores_)):
    ranking_dictionary[x.columns[i]] = features_ranking.scores_[i]

asc_sort = sorted(ranking_dictionary.items(), key = lambda
x:x[1],reverse=True)

for feature, score in asc_sort:
    print(feature, ':', score)
```

```
Age Group : 43822.594537976875
MEDICAL_UNIT : 37796.72489121804
PATIENT_TYPE : 34859.798318027286
CLASIFFICATION_FINAL : 26083.104291404423
PNEUMONIA : 14077.887325773387
INTUBED : 4130.213961265305
HIPERTENSION : 2994.5650815717036
DIABETES : 2649.2413529553733
USMER : 1973.9843896470545
SEX : 1085.1027848188737
ICU : 338.67464097467257
OBESITY : 219.7056593954017
RENAL_CHRONIC : 129.00281808135935
CARDIOVASCULAR : 57.432984168373984
COPD : 56.831732330119344
OTHER_DISEASE : 43.61760073301738
INMSUPR : 16.639369764568322
ASTHMA : 4.673685536567792
PREGNANT : 1.7195691419068941
TOBACCO : 1.2481071577055332
```

Observation and Insights

- Missing data was easily interpretable
- With increasing 'Age Group' death chances increases.
- Patient with 'PNUEMONIA', 'INTUBED' are having higher chances of death.
- Patients with 'Covid_classification as '3' are highly death prone than any other classification. That means Covid with level 3 is the most Severe and damaging.
- Overall people who had pre-diagnosed diseases have higher chances of death.
- Patients who have been hospital had severe covid condition leading to more deaths.

Model Building and Hyper Tuning

- Using Logistic Regression, Descision Classifier and Random Forrest modelsfor training.
- Using XGBClassifier to apply boosting for unbalanced dataset.
- as it has too many features and values KNN, and SVC are too slow and complex to fit but could use in future with selected features.
- Using SMOTE technique for resampling the class distributionn.
- Using F1 score, Recall as primary metrics as the dataset is unbalanced

```
# Stratified train-test split (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, stratify=y, random_state=123)

# Check class distribution in train & test sets
print("Train: ", y_train.value_counts(normalize=True))
print("Test: ", y_test.value_counts(normalize=True))

Train:  Alive
0    0.928104
1    0.071896
Name: proportion, dtype: float64
Test:  Alive
0    0.928104
1    0.071896
Name: proportion, dtype: float64

lr_model = LogisticRegression(max_iter=1000,
random_state=123,verbose=1)
dt_model = DecisionTreeClassifier(max_depth=25, max_features=10,
random_state=123)
rf_model = RandomForestClassifier(n_estimators=150, max_depth=25,
random_state=123,verbose=1)
xgb_model = XGBClassifier(n_estimators=150, random_state=123)

X_train= X_train.astype(int)
X_test= X_test.astype(int)
y_test = y_test.astype(int)
y_train= y_train.astype(int)

lr_model.fit(X_train,y_train)
lr_y_pred= lr_model.predict(X_test)
```

```
print('Logistic Regression Complete')

xgb_model.fit(X_train,y_train,verbose=1)
xgb_y_pred= xgb_model.predict(X_test)
print('XGB Complete')
dt_model.fit(X_train,y_train)
dt_y_pred= dt_model.predict(X_test)
print('Decision Tree Regression Complete')
rf_model.fit(X_train,y_train)
rf_y_pred= rf_model.predict(X_test)
print('Random Forrest Complete')
```

```
Logistic Regression Complete
XGB Complete
Decision Tree Regression Complete

[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:   50.3s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.9s

Random Forrest Complete
```

```
print('lr_model: ',roc_auc_score(y_test,
lr_model.predict_proba(X_test)[:, 1]),end='\n\n')
print('dt_model: ',roc_auc_score(y_test,
dt_model.predict_proba(X_test)[:, 1]),end='\n\n')
print('rf_model: ',roc_auc_score(y_test,
rf_model.predict_proba(X_test)[:, 1]),end='\n\n')
print('xgb_model: ',roc_auc_score(y_test,
xgb_model.predict_proba(X_test)[:, 1]),end='\n\n')


print('lr_model: ',classification_report(y_test,lr_y_pred),end='\n\n')
print('dt_model: ',classification_report(y_test,dt_y_pred),end='\n\n')
print('rf_model: ',classification_report(y_test,rf_y_pred),end='\n\n')
print('xgb_model: ',classification_report(y_test,xgb_y_pred),end='\n\
n')
```

```
lr_model:  0.9611906536124379

dt_model:  0.8842313179781945


[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    2.0s

rf_model:  0.9539316447546056

xgb_model:  0.9676899147520295

lr_model:                 precision    recall  f1-score   support

           0       0.96      0.98      0.97    189272
```

```
           1       0.70      0.50      0.58     14662

    accuracy                           0.95    203934
   macro avg       0.83      0.74      0.78    203934
weighted avg       0.94      0.95      0.94    203934


dt_model:                precision    recall  f1-score   support

           0       0.96      0.98      0.97    189272
           1       0.68      0.49      0.57     14662

    accuracy                           0.95    203934
   macro avg       0.82      0.74      0.77    203934
weighted avg       0.94      0.95      0.94    203934


rf_model:                precision    recall  f1-score   support

           0       0.96      0.98      0.97    189272
           1       0.69      0.52      0.59     14662

    accuracy                           0.95    203934
   macro avg       0.83      0.75      0.78    203934
weighted avg       0.94      0.95      0.95    203934


xgb_model:                precision    recall  f1-score   support

           0       0.96      0.99      0.97    189272
           1       0.73      0.52      0.61     14662

    accuracy                           0.95    203934
   macro avg       0.85      0.75      0.79    203934
weighted avg       0.95      0.95      0.95    203934
```

```
smote = SMOTE(sampling_strategy=0.5, random_state=123)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

lr_model.fit(X_resampled,y_resampled)
lr_y_pred= lr_model.predict(X_test)
print('Logistic Regression Complete')

xgb_model.fit(X_resampled,y_resampled,verbose=1)
xgb_y_pred= xgb_model.predict(X_test)
print('XGB Complete')
dt_model.fit(X_resampled,y_resampled)
dt_y_pred= dt_model.predict(X_test)
print('Decision Tree Regression Complete')
```

```
rf_model.fit(X_resampled,y_resampled)
rf_y_pred= rf_model.predict(X_test)
print('Random Forrest Complete')

print('lr_model: ',roc_auc_score(y_test,
lr_model.predict_proba(X_test)[:, 1]),end='\n\n')
print('dt_model: ',roc_auc_score(y_test,
dt_model.predict_proba(X_test)[:, 1]),end='\n\n')
print('rf_model: ',roc_auc_score(y_test,
rf_model.predict_proba(X_test)[:, 1]),end='\n\n')
print('xgb_model: ',roc_auc_score(y_test,
xgb_model.predict_proba(X_test)[:, 1]),end='\n\n')


print('lr_model: ',classification_report(y_test,lr_y_pred),end='\n\n')
print('dt_model: ',classification_report(y_test,dt_y_pred),end='\n\n')
print('rf_model: ',classification_report(y_test,rf_y_pred),end='\n\n')
print('xgb_model: ',classification_report(y_test,xgb_y_pred),end='\n\
n')
```

```
Logistic Regression Complete
XGB Complete
Decision Tree Regression Complete

[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:  1.3min
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    1.9s

Random Forrest Complete
lr_model:  0.9611741113618207

dt_model:  0.8876410984628946


[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    2.2s

rf_model:  0.950391577537917

xgb_model:  0.9672017062047686

lr_model:                precision    recall  f1-score   support

           0       0.99      0.92      0.95    189272
           1       0.45      0.86      0.59     14662

    accuracy                           0.92    203934
   macro avg       0.72      0.89      0.77    203934
weighted avg       0.95      0.92      0.93    203934


dt_model:                precision    recall  f1-score   support
```

```
           0       0.98      0.93      0.95    189272
           1       0.46      0.82      0.59     14662

    accuracy                           0.92    203934
   macro avg       0.72      0.87      0.77    203934
weighted avg       0.95      0.92      0.93    203934


rf_model:                 precision    recall  f1-score   support

           0       0.99      0.93      0.95    189272
           1       0.46      0.83      0.60     14662

    accuracy                           0.92    203934
   macro avg       0.73      0.88      0.78    203934
weighted avg       0.95      0.92      0.93    203934


xgb_model:                 precision    recall  f1-score   support

           0       0.99      0.92      0.95    189272
           1       0.45      0.90      0.60     14662

    accuracy                           0.91    203934
   macro avg       0.72      0.91      0.78    203934
weighted avg       0.95      0.91      0.93    203934
```

```
class_weight= {0:9.2,1:0.8}
lr_model = LogisticRegression(max_iter=1000,
random_state=123,verbose=1,class_weight=class_weight)
dt_model = DecisionTreeClassifier(max_depth=25, max_features=10,
random_state=123,class_weight=class_weight)
rf_model = RandomForestClassifier(n_estimators=150, max_depth=25,
random_state=123,verbose=1,class_weight=class_weight)
xgb_model = XGBClassifier(n_estimators=150, random_state=123)

lr_model.fit(X_train,y_train)
lr_y_pred= lr_model.predict(X_test)
print('Logistic Regression Complete')

dt_model.fit(X_train,y_train)
dt_y_pred= dt_model.predict(X_test)
print('Decision Tree Regression Complete')
rf_model.fit(X_train,y_train)
rf_y_pred= rf_model.predict(X_test)
print('Random Forrest Complete')

print('lr_model: ',roc_auc_score(y_test,
lr_model.predict_proba(X_test)[:, 1]),end='\n\n')
```

```python
print('dt_model: ',roc_auc_score(y_test,
dt_model.predict_proba(X_test)[:, 1]),end='\n\n')
print('rf_model: ',roc_auc_score(y_test,
rf_model.predict_proba(X_test)[:, 1]),end='\n\n')
# print('xgb_model: ',roc_auc_score(y_test,
xgb_model.predict_proba(X_test)[:, 1]),end='\n\n')
```

```
Logistic Regression Complete
Decision Tree Regression Complete

[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:   51.9s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    2.2s

Random Forrest Complete
lr_model:  0.9607487445568135

dt_model:  0.886563029397784


[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    3.0s

rf_model:  0.9522905035171297
```

```python
print('lr_model: ',classification_report(y_test,lr_y_pred),end='\n\n')
print('dt_model: ',classification_report(y_test,dt_y_pred),end='\n\n')
print('rf_model: ',classification_report(y_test,rf_y_pred),end='\n\n')
print('xgb_model: ',classification_report(y_test,xgb_y_pred),end='\n\
n')
```

```
lr_model:                 precision    recall  f1-score   support

           0       0.93      1.00      0.97    189272
           1       0.93      0.08      0.15     14662

    accuracy                           0.93    203934
   macro avg       0.93      0.54      0.56    203934
weighted avg       0.93      0.93      0.91    203934


dt_model:                 precision    recall  f1-score   support

           0       0.94      0.99      0.97    189272
           1       0.70      0.24      0.36     14662

    accuracy                           0.94    203934
   macro avg       0.82      0.62      0.66    203934
weighted avg       0.93      0.94      0.92    203934
```

```
rf_model:                precision    recall  f1-score   support

           0       0.95      0.99      0.97    189272
           1       0.76      0.27      0.40     14662

    accuracy                           0.94    203934
   macro avg       0.85      0.63      0.68    203934
weighted avg       0.93      0.94      0.93    203934


xgb_model:                precision    recall  f1-score   support

           0       0.99      0.92      0.95    189272
           1       0.45      0.90      0.60     14662

    accuracy                           0.91    203934
   macro avg       0.72      0.91      0.78    203934
weighted avg       0.95      0.91      0.93    203934
```

Best Model To Choose

- XGBClassifier with Smote is the best model to choose due to high Accuracy 91 and recall 91, as we could go with False positive in this case rather than taking risk.

Future Suggestion/Scope

- Should work on a better trade off balance for minority class in this case.
- Should Focus on Hyperparameter tuning for better results.
- More classification could be done such as predicting the level of covid through other features.