Covid-19 Detection via ML

By Zohaib Akhtar under the supervision of Sir Jawed Naseem

Motivation

As we navigate through the tumultuous waters of this global pandemic, it is imperative that we harness the power of data to shed light on the darkest corners of uncertainty.

Drawing inspiration from the groundbreaking research articles that have paved the way, I made a decision to work on the HealthCare Dataset provided by Dr Rashid Nadeem under the supervision of Sir Jawed Naseem.

Our objective was to develop a model which, based on minimal decision variables, could correctly identify if the patient is at risk of death due to Covid-19.

Decision and Target Variables.

While the dataset consisted of approximately 50 variables, I focused my attention on 8 key decision variables, with 1 serving as the target variable.

The decision variables employed in my analysis include ***'age,' 'gender,' 'BMI,' 'prior lung,' 'smokers,' 'day number of PM/PTX,' 'LOSICU,' and 'LOSH.'***

The primary objective of the study was to utilize these variables to accurately determine **the risk of mortality for each patient (death0**."

```
df = df[['age', 'gender', 'BMI', 'prior lung', 'smokers', 'day no of PM/PTX', 'LOSICU', 'LOSH', 'died']]
```

```
df
```

	age	gender	BMI	prior lung	smokers	day no of PM/PTX	LOSICU	LOSH	died
0	57	1.0	24.30	0.0	0	12.0	19.0	30.0	1.0
1	38	1.0	25.16	0.0	0	34.0	57.0	57.0	0.0
2	47	1.0	30.80	0.0	0	4.0	21.0	26.0	0.0
3	24	1.0	23.40	0.0	0	28.0	166.0	296.0	0.0
4	55	1.0	31.20	0.0	0	27.0	23.0	29.0	1.0

Data Preprocessing

The column ‘smokers’ had a few string values. This was fixed by replacing the string ‘0’ and ‘1’ with their respective integers. Also, the string value ‘U’ was replaced by ‘0’ considering majority of the data values were ‘0’

```
df.dtypes
```

```
age                    int64
gender               float64
BMI                  float64
prior_lung           float64
smokers               object
day_no_of_PM/PIX     float64
LOSICU               float64
LOSH                 float64
died                 float64
dtype: object
```

```
df['smokers'].unique()
df['smokers'].value_counts()
```

```
0    219
0     88
1     19
1      2
U      1
```

```
df['smokers'] = df['smokers'].replace('0', 0)
df['smokers'] = df['smokers'].replace('1', 0)
df['smokers'] = df['smokers'].replace('U', 0)
```

Dealing With Null Values:

There were columns in dataset which had null values.

```
df.isna().sum()
```

```
age                 0
gender              1
BMI                23
prior lung          2
smokers            47
day no of PM/PTX    3
LOSICU              2
LOSH                5
died                4
dtype: int64
```

To make things easier, I had created a column `Age_Category` based on age groups. This column was selected since it didn't have any null values in it. The age column was used to find the count. As per the details most of the people were in age group 47-65 and were 1.0 (males).

```
df.loc[:, ('Age_Category')]=pd.cut(df.loc[:, ('age')],bins=[0, 18, 25, 35, 46, 65, 100], labels=['0-18', '19-25', '26-35', '36-46','47-65', '66+'])
```

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

```
47-65    169
36-46     97
66+       70
26-35     35
19-25      5
0-18       0
Name: Age_Category, dtype: int64
```

```
df[['Age_Category','gender', 'age']].groupby(by=['Age_Category', 'gender']).count()
```

Age_Category	gender	age
0-18	0.0	0
	1.0	0
19-25	0.0	1
	1.0	4
26-35	0.0	10
	1.0	25
36-46	0.0	21
	1.0	76
47-65	0.0	34
	1.0	134
66+	0.0	20
	1.0	50

```
df['gender'].fillna(1.0, inplace=True)
```

For dealing with Null Values in BMI column. I calculated the average BMI based of each age-category based on their genders and than substituted the null values with those averages.

```
df[['Age_Category','gender', 'BMI']].groupby(by=['Age_Category', 'gender']).mean()
```

		BMI
Age_Category	**gender**	
0-18	**0.0**	NaN
	1.0	NaN
19-25	**0.0**	39.060000
	1.0	26.470000
26-35	**0.0**	28.951000
	1.0	29.812727
36-46	**0.0**	32.346190
	1.0	28.918571
47-65	**0.0**	34.109375
	1.0	30.311890
66+	**0.0**	30.617368
	1.0	30.008542

```
tmp=df[(df['Age_Category'] == '19-25') & (df['gender'] == 0.0)].BMI.fillna(39.06)
df.update(tmp)
tmp=df[(df['Age_Category'] == '19-25') & (df['gender'] == 1.0)].BMI.fillna(26.47)
df.update(tmp)
tmp=df[(df['Age_Category'] == '26-35') & (df['gender'] == 0.0)].BMI.fillna(28.95)
df.update(tmp)
tmp=df[(df['Age_Category'] == '26-35') & (df['gender'] == 1.0)].BMI.fillna(29.81)
df.update(tmp)
tmp=df[(df['Age_Category'] == '36-46') & (df['gender'] == 0.0)].BMI.fillna(32.34)
df.update(tmp)
tmp=df[(df['Age_Category'] == '36-46') & (df['gender'] == 1.0)].BMI.fillna(28.91)
df.update(tmp)
tmp=df[(df['Age_Category'] == '47-65') & (df['gender'] == 0.0)].BMI.fillna(34.11)
df.update(tmp)
tmp=df[(df['Age_Category'] == '47-65') & (df['gender'] == 1.0)].BMI.fillna(30.31)
df.update(tmp)
tmp=df[(df['Age_Category'] == '66+') & (df['gender'] == 0.0)].BMI.fillna(30.61)
df.update(tmp)
tmp=df[(df['Age_Category'] == '66+') & (df['gender'] == 1.0)].BMI.fillna(30.00)
df.update(tmp)
```

Next was prior-lung which had two null values,

```
df[df['prior lung'].isna()]
```

	age	gender	BMI	prior lung	smokers	day no of PM/PTX	LOSICU	LOSH	died	Age_Category
53	34	1.0	25.3	NaN	0.0	20.0	41.0	43.0	0.0	26-35
196	66	0.0	47.7	NaN	NaN	NaN	9.0	36.0	1.0	66+

First was to compare results of smokers and prior_lung. This way we found that most of the people who had lung disease earlier were non-smokers.

```
print(df[['smokers', 'prior lung']].groupby(by=['smokers']).count())
```

```
         prior lung
smokers
0.0             309
1.0              19
```

```
df['prior lung'][53]=1
df=df.dropna(subset=['prior lung'])
```

The second row was removed in which smoker column was also Null since I wasn't able to find a suitable correlation for it.

For smokers column, I found the count of smokers based on age-category, gender and whether they had lung disease earlier. Based on it we found that

- People above age 36 were mostly smokers
- Mostly males were smokers
- Most of the people which lung disease earlier were not smokers

```
print(df[['Age_Category', 'smokers']].groupby(by=['Age_Category']).count())
print(df[['gender', 'smokers']].groupby(by=['gender']).count())
print(df[['prior lung', 'smokers']].groupby(by=['prior lung']).count())
```

```
              smokers
Age_Category
0-18                0
19-25               5
26-35              32
36-46              86
47-65             140
66+                66
        smokers
gender
0.0          74
1.0         255
            smokers
prior lung
0.0            316
1.0             13
```

```
tmp=df[df['Age_Category'] == '36-46'].smokers.fillna(1)
df.update(tmp)
tmp=df[df['Age_Category'] == '47-65'].smokers.fillna(1)
df.update(tmp)
tmp=df[df['Age_Category'] == '66+'].smokers.fillna(1)
df.update(tmp)
tmp=df[df['gender'] == 1].smokers.fillna(1)
df.update(tmp)
tmp=df[df['prior lung'] == 0.0].smokers.fillna(1)
df.update(tmp)
```

On further checking I found that the target also had a few null values on which none of the other variables were null

```
df[df['died'].isna()]
```

	age	gender	BMI	prior lung	smokers	day no of PM/PTX	LOSICU	LOSH	died	Age_Category
120	61	0.0	29.37	0.0	0.0	24.0	5.0	24.0	NaN	47-65
121	46	1.0	42.56	0.0	0.0	6.0	11.0	15.0	NaN	36-46
312	36	0.0	30.85	0.0	1.0	0.0	3.0	6.0	NaN	36-46
366	42	1.0	24.80	0.0	0.0	0.0	34.0	52.0	NaN	36-46

I stored these in a variable excluding the columns ‘died’ and ‘Age_Category’ called `predict_y` so I can predict them later on once the model is ready.

```
predict_y=df[df['died'].isna()].drop(columns=['died', 'Age_Category'])
```

Rest of the records (09) were dropped which had null values

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

```
(375, 10)
(362, 10)
```

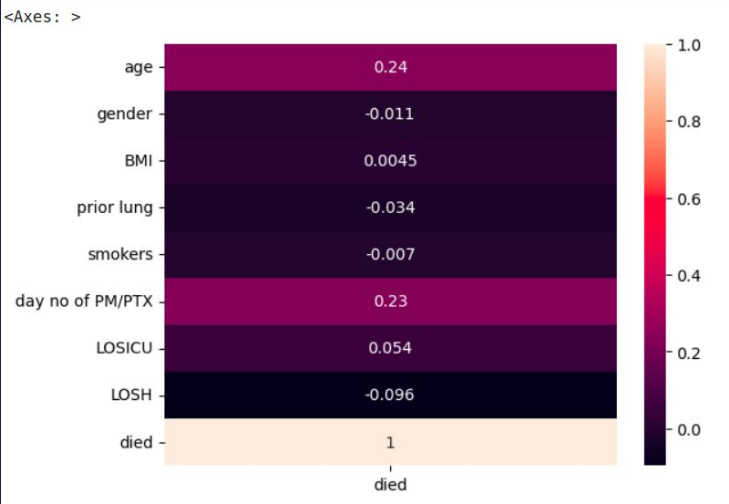
```
df = df.dropna()
print(df.shape)
```

```
(362, 10)
```

Correlations

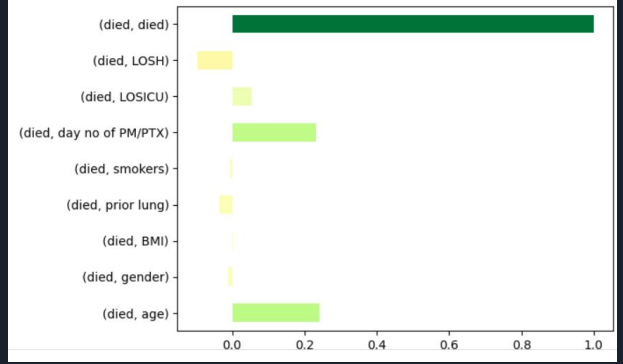
Now that since our data was ready, to get some insights, correlation heatmap and plots was used, this shows that our target column is more correlated to ‘age’ and ‘day no of PM/PTX’

```
sns.heatmap(df.drop(columns=['Age_Category']).corr()[['died']], annot=True)
```




```
import matplotlib.colors as mcolors
from matplotlib.colors import TwoSlopeNorm

norm = TwoSlopeNorm(vmin=-1, vcenter =0, vmax=1)
corrplot = df.corr().unstack()[['died']]
colors = [plt.cm.RdYlGn(norm(c)) for c in corrplot.values]
corrplot.plot.barh(color=colors)
```

```
<ipython-input-28-2ee2caf95654>:5: FutureWarning: The default value of numeric_onl
  corrplot = df.corr().unstack()[['died']]
<Axes: >
```



Applying ML

In order to train and test the Machine Learning model, the dataset of 362 records were split into the ratio of 85% and 15% for training and testing purposes respectively.

This way we had about 308 records for training and about 54 records for testing purpose.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.drop(columns=['died', 'Age_Category']), df['died'], test_size=0.15, random_state=42)
```

The model which were trained were

- Linear Regression
- Logistic Regression
- RandomForestClassifier
- DecisionTreeClassifier
- XGBClassifier
- SupportVectorClassifier

The performance metrics which are used were Mean_absolute_error and R_squared_value for Linear Regression, and ConfusionMatrix for rest of the classification models.

LinearRegression

Linear regression models the relationship between a dependent variable and one or more independent variables by fitting a straight line to the data points, aiming to predict the dependent variable's value based on the independent variables' values.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error

lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
```

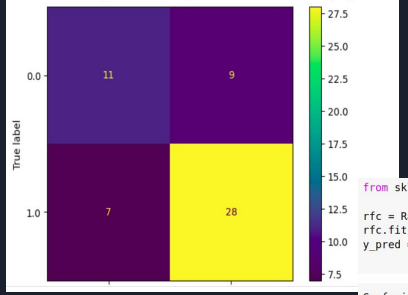
```
mse = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean absolute error: {mse}")
print(f"R-squared score: {r2}")
```

```
Mean absolute error: 0.4219909844914668
R-squared score: 0.047950561549189086
```

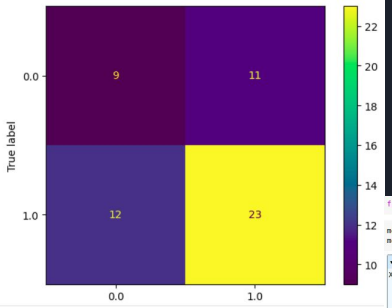
Since it was a classification problem, it was almost certain to have this result. The mean absolute error is quite high and the R value is also almost zero, which means the model wouldn’t perform well.

If we train the model on remaining classification algorithms. The results start getting better. We used ConfusionMatrix to check the models performance.

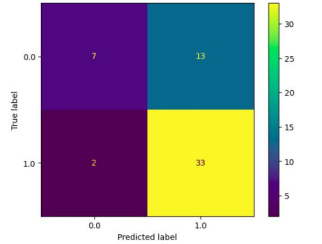
```
from sklearn.linear_model import LogisticRegression

lg = LogisticRegression(max_iter=200)
lg.fit(X_train, y_train)
y_pred = lg.predict(X_test)

from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7a
```



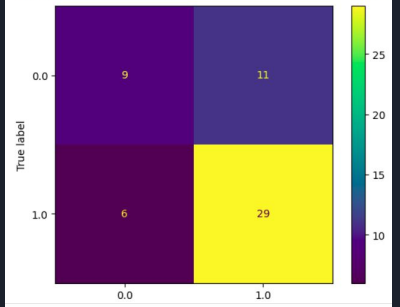

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDispla
```



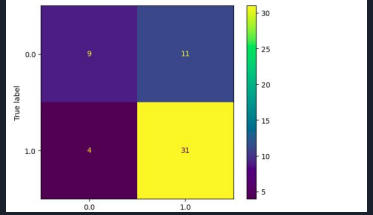

```
from sklearn.svm import SVC
svc=SVC()
svc.fit(X_train, y_train)

▾ SVC
SVC()

y_pred=svc.predict(X_test)
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
```




```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)

ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDispla
```




```
from xgboost import XGBClassifier

model=XGBClassifier()
model.fit(X_train, y_train)

▾ XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=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, learning_rate=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=None, ...)

y_pred=model.predict(X_test)

ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7aa66cc5e208>
```



Hyper Parameter Tuning

Based on the screenshots attached in previous slide. It was clear that the support vector classifier had provided better results than others. It is because, it was able to correctly identify true labels.

However, there were a true-negative and false-positive results.

The performance of this model can be further improved by tuning the hyper parameters. For which `GridSearchCV` was used. I checked different values of C (penalty parameter), gamma (how far the influence of a single training example reaches_and Kernel (how the model show train).

The results now seemed a lot better. The model was able to correctly identify all the true labels.

There were still a few false-positives, which can later be rectified manually. While providing treatment to patients

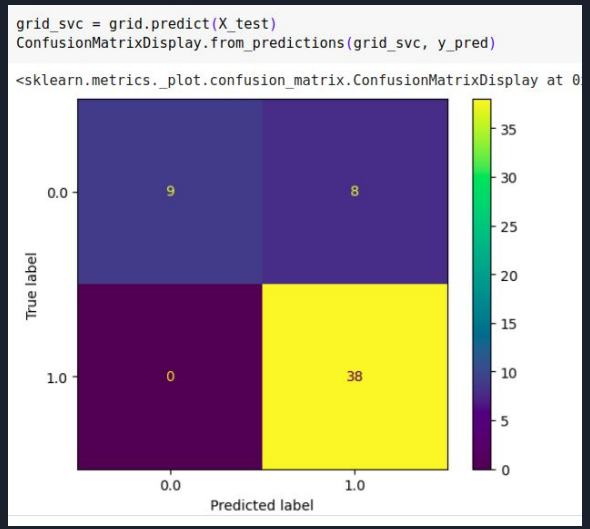
Continue…

Continue...

```
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.01, 0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf','linear']}
grid = GridSearchCV(svc,param_grid)
grid.fit(X_train, y_train)
```

▸ GridSearchCV
▸ estimator: SVC
▸ SVC

```
grid_svc = grid.predict(X_test)
ConfusionMatrixDisplay.from_predictions(grid_svc, y_pred)
```



The following parametric values seemed to be more useful.

```
print(grid.best_params_)
print(grid.best_estimator_.get_params())
```

```
{'C': 0.1, 'gamma': 1, 'kernel': 'linear'}
```

```
{'C': 0.1, 'break ties': False, 'cache size': 200, 'class weight': None, 'coef0': 0.0,
'decision function shape': 'ovr', 'degree': 3, 'gamma': 1, 'kernel': 'linear', 'max iter':
-1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose':
False}
```

Predicting results for Null values in ‘died’ column

Moving back to the data we stored in predict_y which contained null values of our target variable. We can now predict those with the help of our model

```
predicted_y = grid.predict(predict_y)
df_result = predict_y
df_result['died'] = predicted_y
print(df_result)
```

```
     age  gender    BMI  prior lung  smokers  day no of PM/PTX  LOSICU  LOSH
120   61     0.0  29.37         0.0      0.0               24.0     5.0  24.0
121   46     1.0  42.56         0.0      0.0                6.0    11.0  15.0
312   36     0.0  30.85         0.0      1.0                0.0     3.0   6.0
366   42     1.0  24.80         0.0      0.0                0.0    34.0  52.0

     died
120   1.0
121   1.0
312   1.0
366   0.0
```

Also, a copy of trained model has been saved on GoogleDrive using joblib library so it later used if needed and can be accessed using https://drive.google.com/file/d/1-3I-JI6iFXZ_cc6lQl0d779dCFUCnayC/view?usp=drive_link

```
import joblib
joblib.dump(grid, '/content/drive/MyDrive/svc-model.sav')
```

```
['/content/drive/MyDrive/svc-model.sav']
```