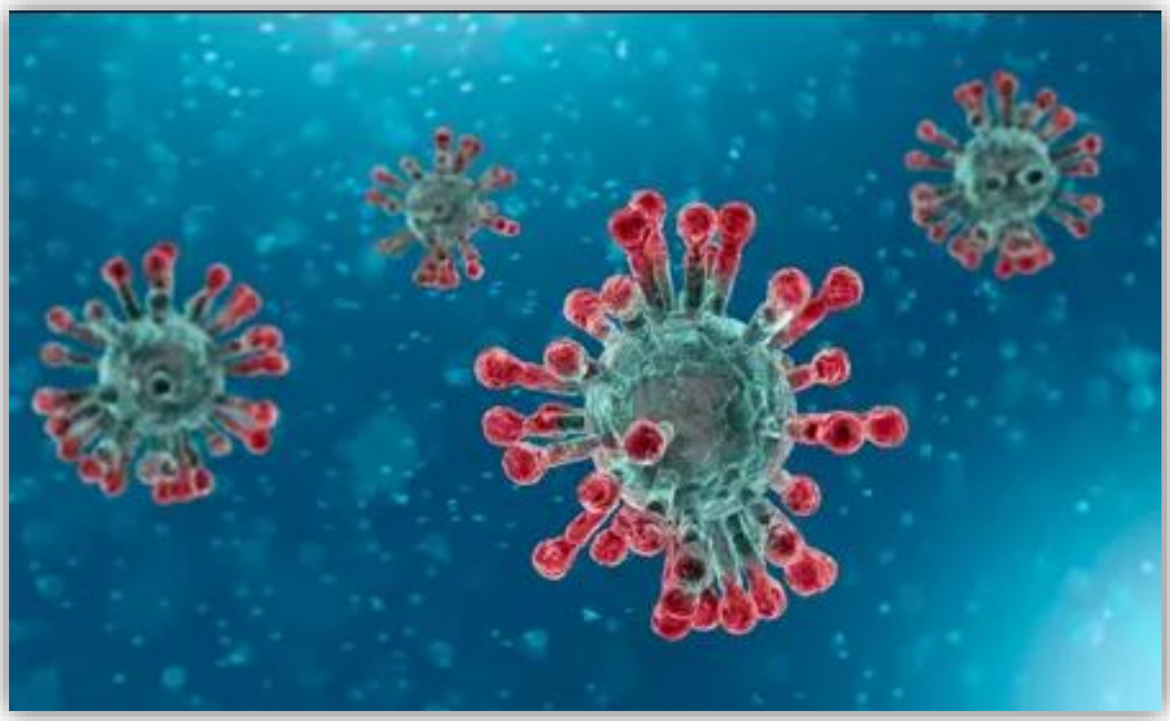
Predictive Modelling Project on Corona Virus Detection



Project by:

Maaz Ansari J002

Riddhi Mehta J030

Table of Contents

Introduction:

Like most people in the world right now, we all are genuinely concerned about COVID-19.

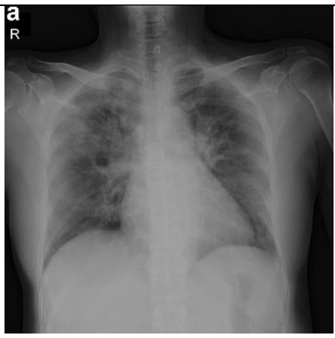
We all find ourselves constantly analysing our personal health and wonder if have it. The more we worry about it, the more it turns into a painful mind game of legitimate symptoms. Cough and low-grade fever? That could be COVID-19…or it could simply be cough and fever.

It’s impossible to know without a test whether you actually have the virus or not.

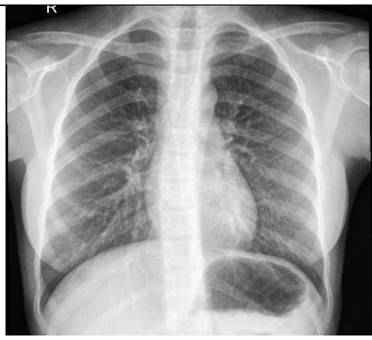
Instead of sitting idly at home in this lockdown, we decided to create awareness by making this project which detects whether the person has Corona virus or not using X-Ray images of the person.

Dataset:

1. X-ray images of patients who have tested positive for COVID-19.
2. X-ray images of normal (not infected) healthy patients.
3. COVID-19 (csv file) country wise. (as per 10^{th} March 2020)



Example of an X-Ray image of a person who has tested positive for COVID-19



Example of an X-Ray image of a person who has tested negative for COVID-19

Code:

1. Data Cleaning and Data Visualization (CSV Dataset)
2. Prediction on image dataset (X-Ray Images)

1) Data Cleaning and Data Visualization (CSV Dataset)

Importing necessary libraries and reading the data

```
[2] import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from matplotlib import cm
    from mpl_toolkits.mplot3d import Axes3D
```

```
[3] data= pd.read_csv("C:/Users/Maaz/Downloads/CORONA.csv")
    data.head()
```

Data Cleaning

```
[4] data = data.drop(['id','location','symptom_onset','link'],1)
    data = data.dropna()
    data.isna().sum()
```

```
country           0
gender            0
age               0
visiting Wuhan    0
from Wuhan        0
death             0
recovered         0
dtype: int64
```

```
[5] from sklearn.preprocessing import LabelEncoder
    data_cat = data.loc[:,('gender')]
    data_one_hot = pd.get_dummies(data_cat)
    data = pd.concat([data,data_one_hot],1)
    data= data.drop(['gender'],1)
    data.head()
```

	country	age	visiting Wuhan	from Wuhan	death	recovered	female	male
0	China	66.0	1	0.0	0	0	0	1
1	China	56.0	0	1.0	0	0	1	0
2	China	46.0	0	1.0	0	0	0	1
3	China	60.0	1	0.0	0	0	1	0
4	China	58.0	0	0.0	0	0	0	1

Cleaning the data by removing redundant columns, NA values and applying one hot encoding.

Descriptive Analysis

```
[6] data.describe()
```

	age	visiting Wuhan	from Wuhan	death	recovered	female	male
count	821.000000	821.000000	821.000000	821.000000	821.000000	821.000000	821.000000
mean	49.820341	0.177832	0.182704	0.070646	0.174178	0.422655	0.577345
std	17.940000	0.382604	0.386659	0.256388	0.379494	0.494283	0.494283
min	0.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	35.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	51.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
75%	64.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000
max	96.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

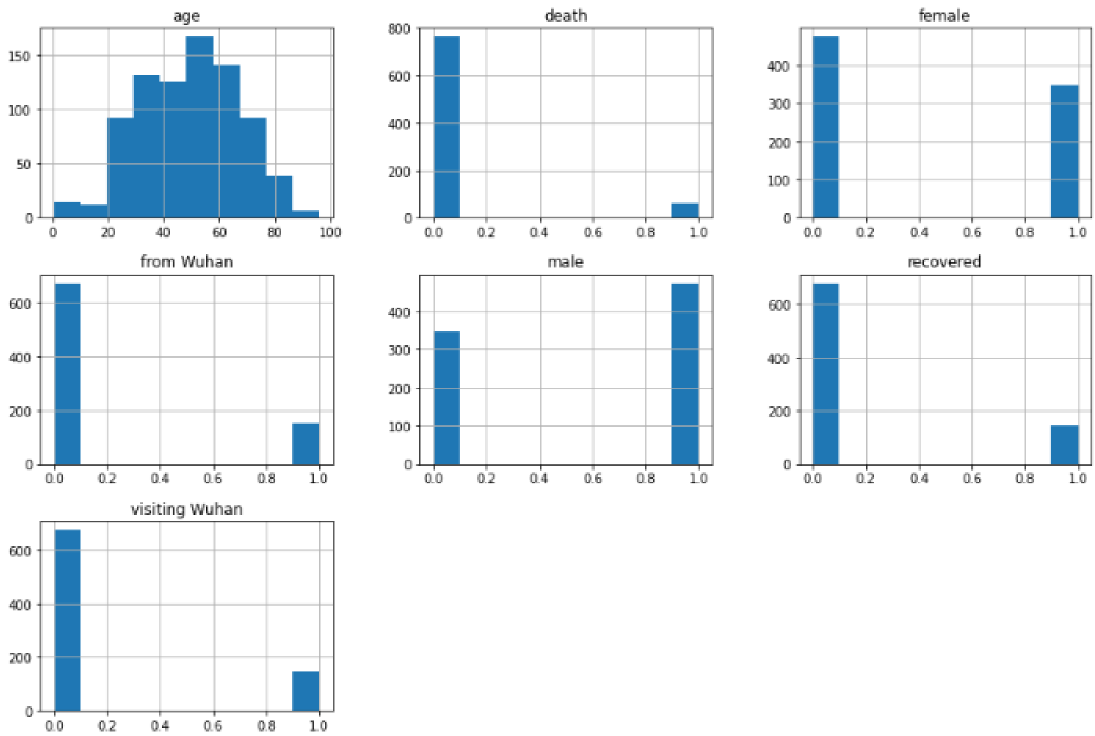
```
[7] print(data.death.value_counts())
    print(data.country.value_counts())
```

```
0    763
1     58
Name: death, dtype: int64
China           186
Japan           185
Hong Kong        93
South Korea      92
Singapore        90
Taiwan           31
Malaysia         23
Thailand         16
France           16
Australia        15
Spain            15
Germany          14
Canada           12
Vietnam           8
UAE               7
USA               6
Phillipines       3
Finland           1
Cambodia          1
Lebanon           1
UK                1
Sri Lanka         1
Sweden            1
Italy             1
Nepal             1
Switzerland       1
Name: country, dtype: int64
```

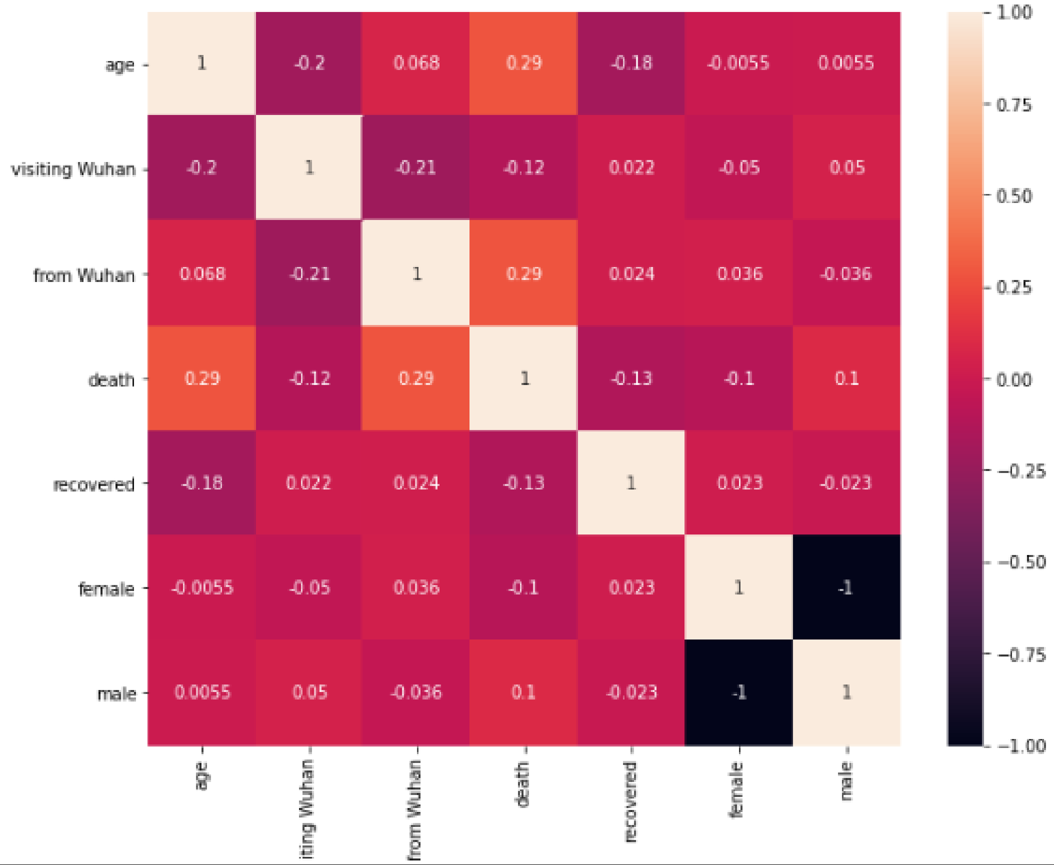
Number of COVID-19 cases in each country. (as per 10th March 2020)

Exploratory Data Analysis

```
plt.rcParams['figure.figsize']=[15,10]
data.hist()
plt.show()
```

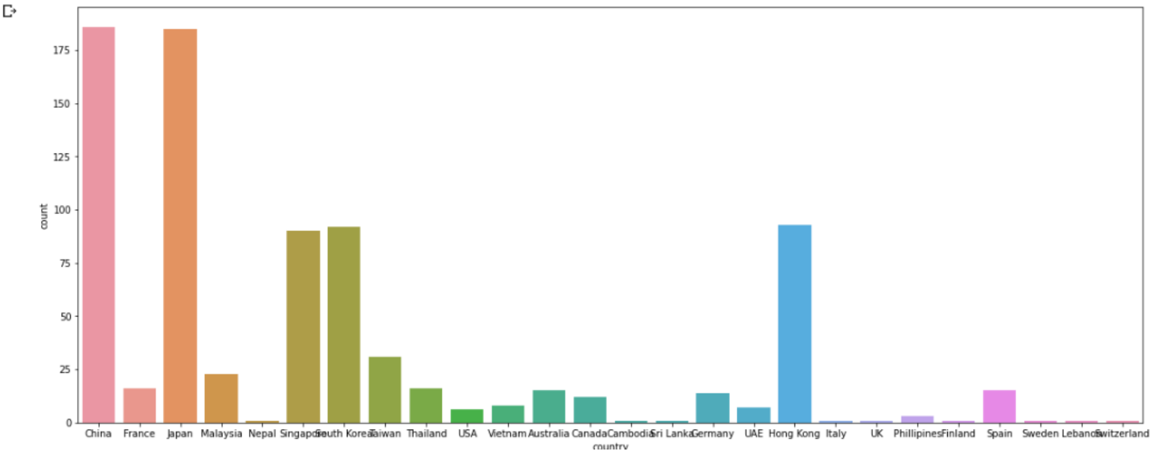


```python
#Correlation Heatmap
plt.figure(figsize=(10,8))
sns.heatmap(data.corr(), annot=True)
plt.show()
```

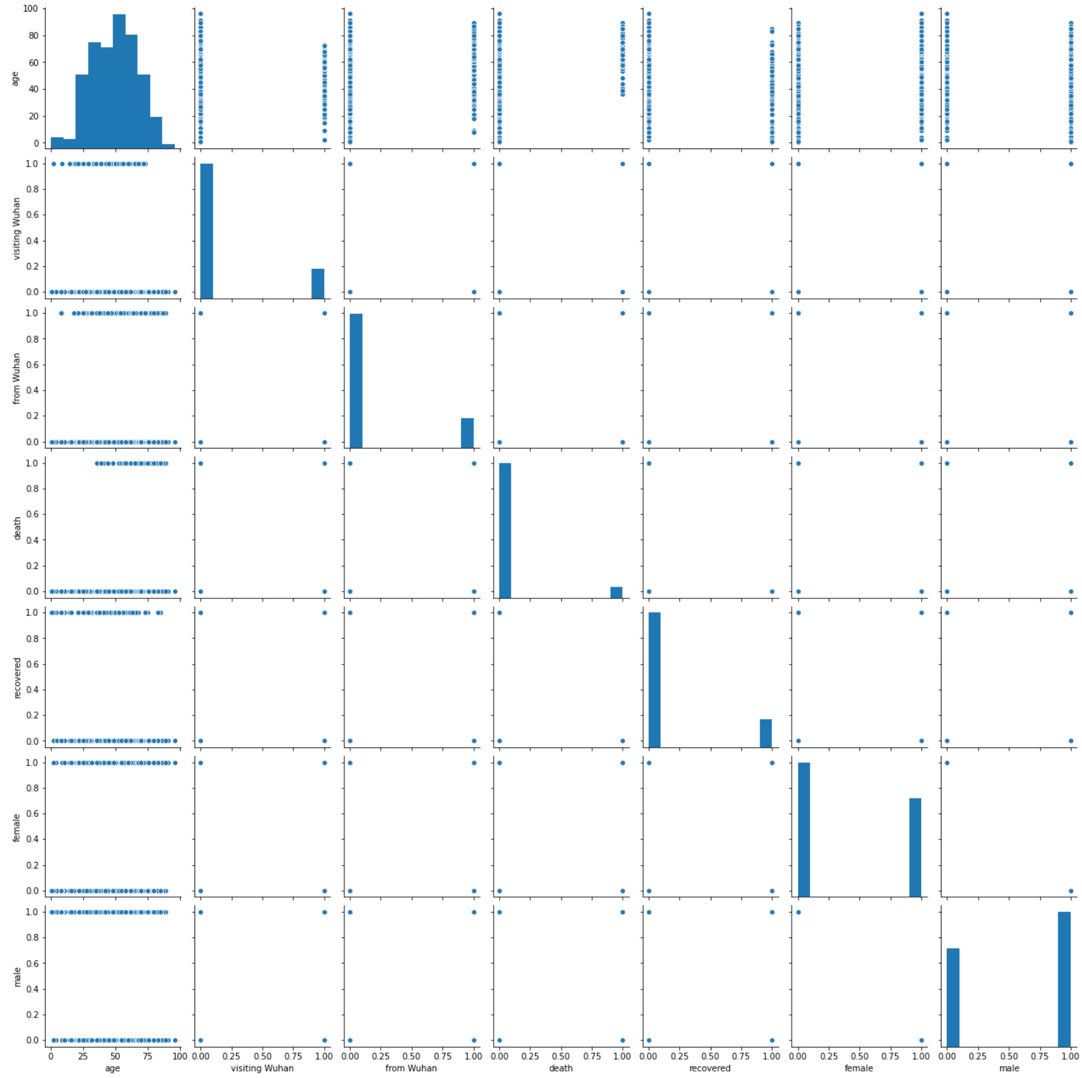


Each square shows the correlation between the variables on each axis. Correlation ranges from -1 to +1. Values closer to zero means there is no linear trend between the two variables. A positive correlation, when the correlation coefficient is greater than 0, signifies that both variables move in the same direction or are correlated. A negative (inverse) correlation occurs when the correlation coefficient is less than 0 and indicates that both variables move in the opposite direction.

```
[17] plt.figure(figsize=(20,8))
     sns.countplot(x='country', data=data)
     plt.show()
```



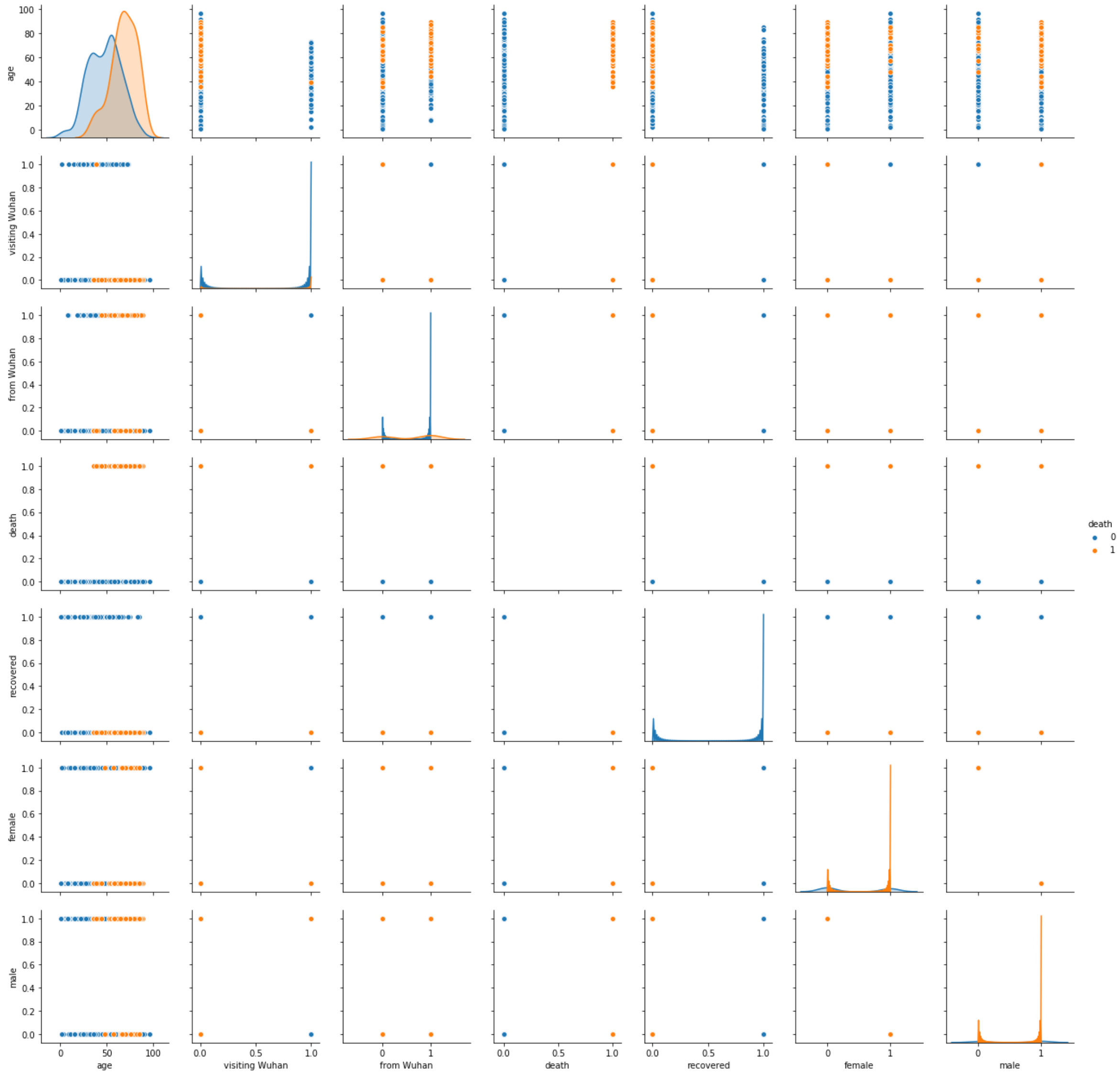
Number of cases country wise (Please refer to the .ipynb file for a clear Output)



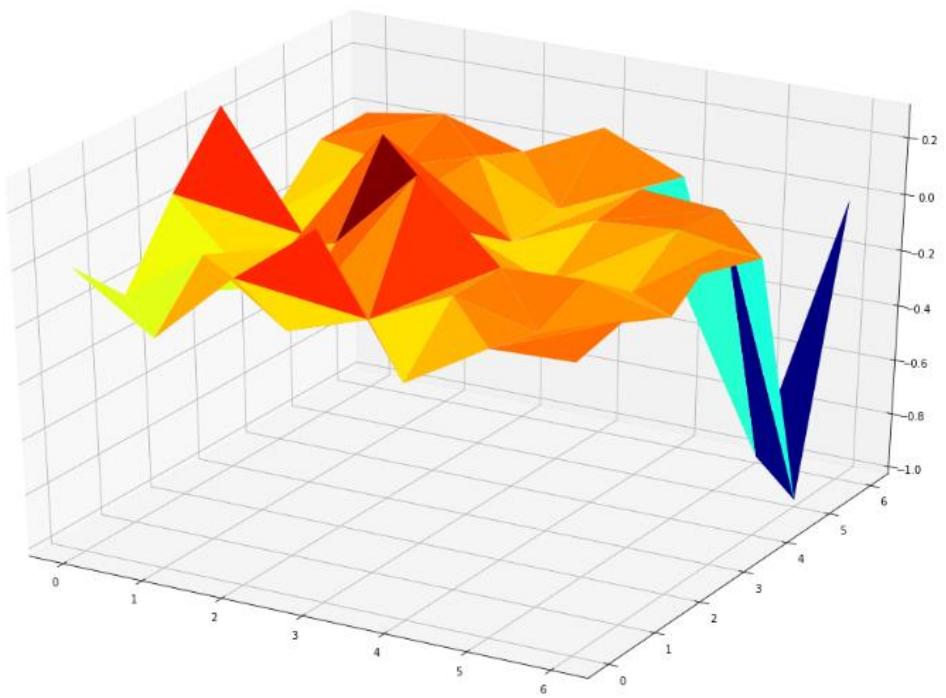
A pairs plot allows us to see both distribution of single variables and relationships between two variables. (Please refer to the .ipynb file for a clear Output)

```
sns.pairplot(data, hue='death')
```

This Pair plot is with respect to the variable death. (Blue colour signifies Recovered or still a patient, Orange colour signifies Death.)



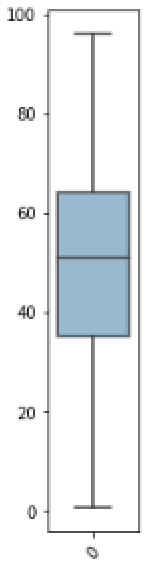
```
[ ]  # generating correlation data
    df = data.corr()
    df.index = range(0, len(df))
    df.rename(columns = dict(zip(df.columns, df.index)), inplace = True)
    df = df.astype(object)

    for i in range(0, len(df)):
        for j in range(0, len(df)):
            if i != j:
                df.iloc[i, j] = (i, j, df.iloc[i, j])
            else :
                df.iloc[i, j] = (i, j, 0)

    df_list = []

    for sub_list in df.values:
        df_list.extend(sub_list)

    # converting list of tuples into trivariate dataframe
    plot_df = pd.DataFrame(df_list)

    fig = plt.figure()
    ax = Axes3D(fig)

    # plotting 3D trisurface plot
    ax.plot_trisurf(plot_df[0], plot_df[1], plot_df[2],
                    cmap = cm.jet, linewidth = 0.2)

    plt.show()
```



3D Correlation Plot

```
[12] plt.figure(figsize = (1, 6))
     ax = sns.boxplot(data=data['age'])
     plt.setp(ax.artists, alpha=.5, linewidth=2, edgecolor="k")
     plt.xticks(rotation=45)
```

```
(array([0]), <a list of 1 Text major ticklabel objects>)
```



Boxplot of Age of the COVID-19 Patients. (A box plot is a method for graphically depicting groups of numerical data through their quartiles.)

2) **Prediction on image dataset (X-Ray Images):**

Import necessary packages

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer, LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from imutils import paths
from scipy import misc
from skimage import io
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import cv2
import os
import warnings
warnings.filterwarnings("ignore")
INIT_LR = 1e-3
EPOCHS = 20
BS = 8
```

Importing necessary packages and initializing the initial learning rate, number of epochs and batch size for CNN.

Reading Image Paths

```
[ ]  imagePaths = list(paths.list_images("D:/RIDDHI MEHTA/keras-covid-19"))
```

```python
def machine_learning(paths):

    data_ml = pd.DataFrame()
    labels = []

    imagePaths = paths

    for imagePath in imagePaths:

        label = imagePath.split('\\')[-2]
        image = pd.DataFrame(misc.imresize(cv2.cvtColor(cv2.imread(imagePath),
                                                        cv2.COLOR_BGR2GRAY),
                                                        (224,224)).reshape(1,-1))

        data_ml = pd.concat([data_ml, image])
        labels.append(label)

    data_ml = StandardScaler().fit_transform(data_ml)
    pca = PCA(n_components=50)
    data_ml = pca.fit_transform(data_ml)
    data_ml = pd.DataFrame(data_ml)

    labels = list(labels)
    data_ml["label"] = labels
    data_ml["label"] = LabelEncoder().fit_transform(data_ml["label"])


    return data_ml

data_ml = machine_learning(imagePaths)
```

Pre-Processing the images by:

1. Resizing
2. Reshaping
3. Converting RGB to Grey scale
4. Scaling the data using StandardScaler
5. Principal Component Analysis

Train Test Split

```python
[ ]  X_train, X_test, Y_train, Y_test = train_test_split(data_ml.iloc[:,:-1], data_ml["label"],
                                                         test_size=0.20, stratify=data_ml["label"],
                                                         random_state=42)
```

Split the data into training set (80%) and testing set (20%)

Support Vector Machine

Support Vector Classification

```
[ ] from sklearn.svm import SVC
    svc = SVC()
    svc.fit(X_train, Y_train)
    Y_pred = svc.predict(X_test)
```

```
[ ] print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.20      0.33         5
           1       0.56      1.00      0.71         5

    accuracy                           0.60        10
   macro avg       0.78      0.60      0.52        10
weighted avg       0.78      0.60      0.52        10
```

Support Vector Machine Accuracy: 60%.

Decision Tree

Decision Trees Classification

```
[ ] from sklearn.tree import DecisionTreeClassifier
    dtc = DecisionTreeClassifier()
    dtc.fit(X_train, Y_train)
    Y_pred = dtc.predict(X_test)
```

```
[ ] print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.60      0.75         5
           1       0.71      1.00      0.83         5

    accuracy                           0.80        10
   macro avg       0.86      0.80      0.79        10
weighted avg       0.86      0.80      0.79        10
```

Decision Tree Accuracy: 80%

Random Forest

Random Forests

```
[ ] from sklearn.ensemble import RandomForestClassifier
    rfc = RandomForestClassifier()
    rfc.fit(X_train, Y_train)
    Y_pred = rfc.predict(X_test)
```

```
[ ] print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       1.00      1.00      1.00         5

    accuracy                           1.00        10
   macro avg       1.00      1.00      1.00        10
weighted avg       1.00      1.00      1.00        10
```

Random Forest Accuracy: 100%

Logistic Regression

Logistic Regression

```
[ ] from sklearn.linear_model import LogisticRegression

    logr = LogisticRegression()
    logr.fit(X_train, Y_train)
    Y_pred = logr.predict(X_test)
```

```
[ ] print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       1.00      1.00      1.00         5

    accuracy                           1.00        10
   macro avg       1.00      1.00      1.00        10
weighted avg       1.00      1.00      1.00        10
```

Logistic Regression Accuracy: 100%

Summary of all Machine Learning alogrithms

```
[ ]  name = ["Logistic Regression","Support Vector Classifier",
             "Decision Tree Classifier","Random Forests Classifier"]
     train_acc = [logr.score(X_train, Y_train),svc.score(X_train, Y_train),
                  dtc.score(X_train, Y_train),rfc.score(X_train, Y_train)]
     test_acc = [logr.score(X_test, Y_test),svc.score(X_test, Y_test),
                 dtc.score(X_test, Y_test),rfc.score(X_test, Y_test)]
     summary = pd.DataFrame()
     summary["name"] = name
     summary["train_acc"] = train_acc
     summary["test_acc"] = test_acc
     summary
```

	name	train_acc	test_acc
0	Logistic Regression	1.0	1.0
1	Support Vector Classifier	1.0	0.6
2	Decision Tree Classifier	1.0	0.8
3	Random Forests Classifier	1.0	1.0

We can observe that Logistic Regression and Random forest gives us the best results with 100% accuracy, followed by Decision Tree (80%) and then Support Vector Machine(60%)

Now that we have tested few of the Machine Learning algorithms, let's try and apply CNN to check how it performs on the image dataset.

CNN

Reading Images and Pre-Processing

```
[ ]  def deep_learning(paths):

         data_dl = []
         labels = []

         imagePaths = paths

         for imagePath in imagePaths:

             label = imagePath.split('\\')[-2]
             image = cv2.imread(imagePath)
             image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
             image = cv2.resize(image, (224, 224))

             data_dl.append(image)
             labels.append(label)

         data_dl = np.array(data_dl) / 255.0
         labels = np.array(labels)

         return data_dl, labels

     data_dl, labels = deep_learning(imagePaths)
```

Pre-Processing the images by

1. Converting RGB to Grey scale
2. Resizing

Train Test Split

```
[ ]  lb = LabelBinarizer()
     labels = lb.fit_transform(labels)
     labels = to_categorical(labels)
     X_train, X_test, Y_train, Y_test = train_test_split(data_dl, labels,
                                                         test_size=0.20, stratify=labels,
                                                         random_state=42)
```

Split the data into training set (80%) and testing set (20%)

CNN

```
[ ] baseModel = VGG16(weights="imagenet", include_top=False,
                      input_tensor=Input(shape=(224, 224, 3)))
```

```
[ ] headModel = baseModel.output
    headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
    headModel = Flatten(name="flatten")(headModel)
    headModel = Dense(64, activation="relu")(headModel)
    headModel = Dropout(0.5)(headModel)
    headModel = Dense(2, activation="softmax")(headModel)
```

```
[ ] model = Model(inputs=baseModel.input, outputs=headModel)
```

```
[ ] for layer in baseModel.layers:
        layer.trainable = False
```

```
[ ] opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
    model.compile(loss="binary_crossentropy", optimizer=opt,
                  metrics=["accuracy"])
```

```
[ ] trainAug = ImageDataGenerator(rotation_range=15, fill_mode="nearest")
```

```
[ ] H = model.fit_generator(trainAug.flow(X_train, Y_train, batch_size=BS),
        steps_per_epoch=len(X_train) // BS,
        validation_data=(X_test, Y_test),
        validation_steps=len(X_test) // BS,
        epochs=EPOCHS)
```

Parameters:
Initial learning rate: 1e-3
Number of epochs: 20
Batch size: 8
Optimizer: Adam
Loss: Binary Crossentropy

```
[ ]  predIdxs = model.predict(X_test, batch_size=BS)
```

```
[ ]  print(classification_report(Y_test.argmax(axis = 1),
                                 predIdxs.argmax(axis = 1),
                                 target_names=lb.classes_))
```

```
              precision    recall  f1-score   support

       covid       0.83      1.00      0.91         5
      normal       1.00      0.80      0.89         5

    accuracy                           0.90        10
   macro avg       0.92      0.90      0.90        10
weighted avg       0.92      0.90      0.90        10
```

```
cm = confusion_matrix(Y_test.argmax(axis=1),
                      predIdxs.argmax(axis = 1))
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
print(cm)
print("accuracy: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))
```

```
[[5 0]
 [1 4]]
accuracy: 0.9000
sensitivity: 1.0000
specificity: 0.8000
```

CNN Accuracy: 90%

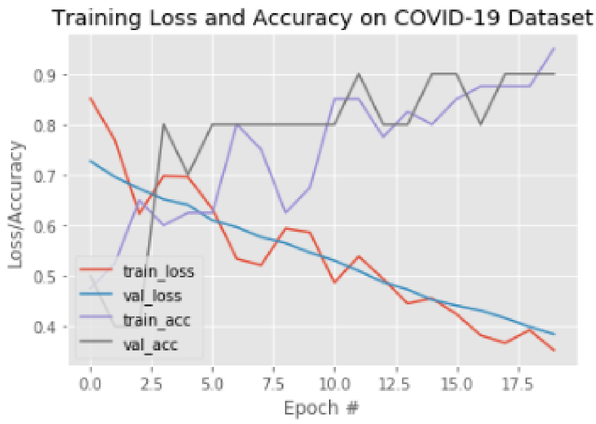
Sensitivity (also called the true positive rate or recall) measures the proportion of actual positives that are correctly identified as such. Here, 5 people had Corona Virus and they all were tested positive (100%).

Specificity (also called the true negative rate) measures the proportion of actual negatives that are correctly identified as such. Here, Specificity is 80%.

Accuracy and Loss Plots

```
[ ] # plot the training loss and accuracy
    N = EPOCHS
    plt.style.use("ggplot")
    plt.figure()
    plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
    plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
    plt.plot(np.arange(0, N), H.history["acc"], label="train_acc")
    plt.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
    plt.title("Training Loss and Accuracy on COVID-19 Dataset")
    plt.xlabel("Epoch #")
    plt.ylabel("Loss/Accuracy")
    plt.legend(loc="lower left")
```

```
<matplotlib.legend.Legend at 0x2838002c3c8>
```



Conclusion:

Summary of all the Algorithms we have used:

Sr. No	Algorithm	Training Accuracy	Testing Accuracy
1	Support Vector Machine	100%	60%
2	Decision Tree	100%	80%
3	Random Forest	100%	100%
4	Logistic Regression	100%	100%
5	CNN (Deep Learning)	95%	90%

Often Machine Learning algorithms outperform Neural Networks when the dataset is very small. Since our dataset had only 50 images, we can see the accuracy of Logistic Regression and Random Forest is more than CNN.

Hence, we can identify whether a person has Corona Virus or not with an accuracy of 100%.