PROJECT

Introduction to Artificial Intelligence COVID-19 Infection Analysis and Prediction

Magdalena Pakuła

1 June 2023

Contents

Introduction

In this project, titled "COVID-19 Infection Analysis and Prediction," I aim to leverage artificial intelligence methods to analyze and predict the spread of the COVID-19 infection. The dataset used for this project contains clinic, demographic, and geographic information about COVID-19 patients, which is publicly available through the GitHub repository:
https://github.com/beoutbreakprepared/nCoV201)
and the Kaggle platform:
https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge.

The project consists of four parts: dataset analysis, Bayes Nets, machine learning, and improving results and theoretical formalism. In the dataset analysis part, I will extract statistical information and investigate correlations between variables. Missing values will be handled using appropriate measures. The Bayes Nets section focuses on using probabilistic graphical models to answer specific questions related to COVID-19 symptoms, patient status, and recovery intervals. In the machine learning part, I will employ algorithms like K-Nearest Neighbors (K-NN) or Bayes Classification to predict patient outcomes and utilize regression techniques for age prediction. Clustering methods, specifically K-means, will be applied to segment individuals into different clusters. The final part aims to improve the results obtained so far and provide a theoretical formalism, including addressing data imbalance, handling missing values, and finding the best parameters using the Grid-search algorithm.

Through this report, I hope to advance understanding of COVID-19 infection patterns and shed light on potential applications of artificial intelligence in epidemiology. By examining the dataset and utilizing different AI techniques, I anticipate discovering useful information that will aid in tracking and predicting the progression of the COVID-19 infection.

I would also like to link my public repository on Github, on which I have published all the coding for this project:
https://github.com/MagdalenaPakula/ISEP_Introduction_To_AI

1 Analysis of the Dataset

In the dataset we are operating on, we can distinct following columns:

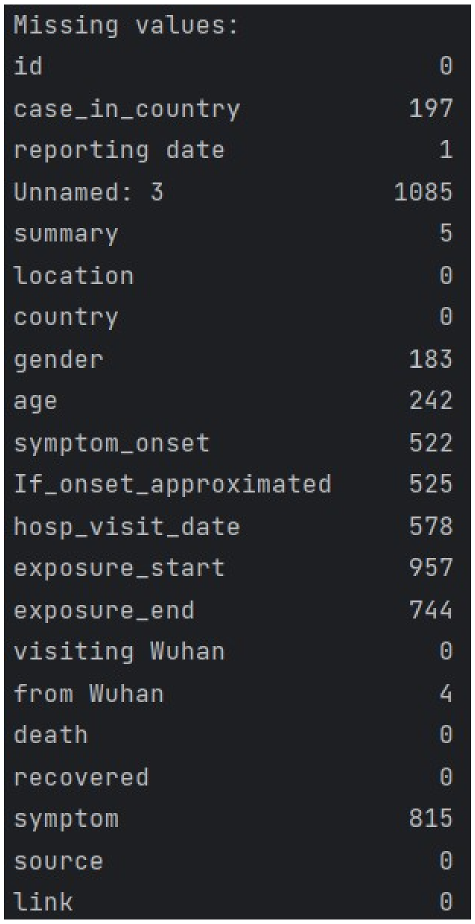
- **id**: Represents the unique identifier for each record.
- **case_in_country**: Indicates the case number or identifier for the COVID-19 patient.
- **reporting date**: The date when the case was reported.
- **summary**: A brief summary or description of the case.
- **location**: The location where the case was reported.
- **country**: The country where the case was reported.
- **gender**: The gender of the patient.
- **age**: The age of the patient.
- **symptom_onset**: The date when the symptoms began.
- **If_onset_approximated**: Indicates if the symptom onset date is an approximation.
- **hosp_visit_date**: The date of the hospital visit.
- **exposure_start**: The date when the patient was exposed to the virus.
- **exposure_end**: The date when the exposure ended.
- **visiting Wuhan**: Indicates if the patient visited Wuhan.
- **from Wuhan**: Indicates if the patient is from Wuhan.
- **death**: Indicates if the patient died.
- **recovered**: Indicates if the patient recovered.
- **symptom**: Information about the symptoms.
- **source**: The source of the information.

- **link**: A link to the source or reference.

When checking the dataset using the following commands, I have encountered multiple problems.

```
print(data.head())
print(data.info())
print(data.describe())

print(data.isnull().sum())
```

Firstly, I have notices there are multiple missing values, image is shown below.



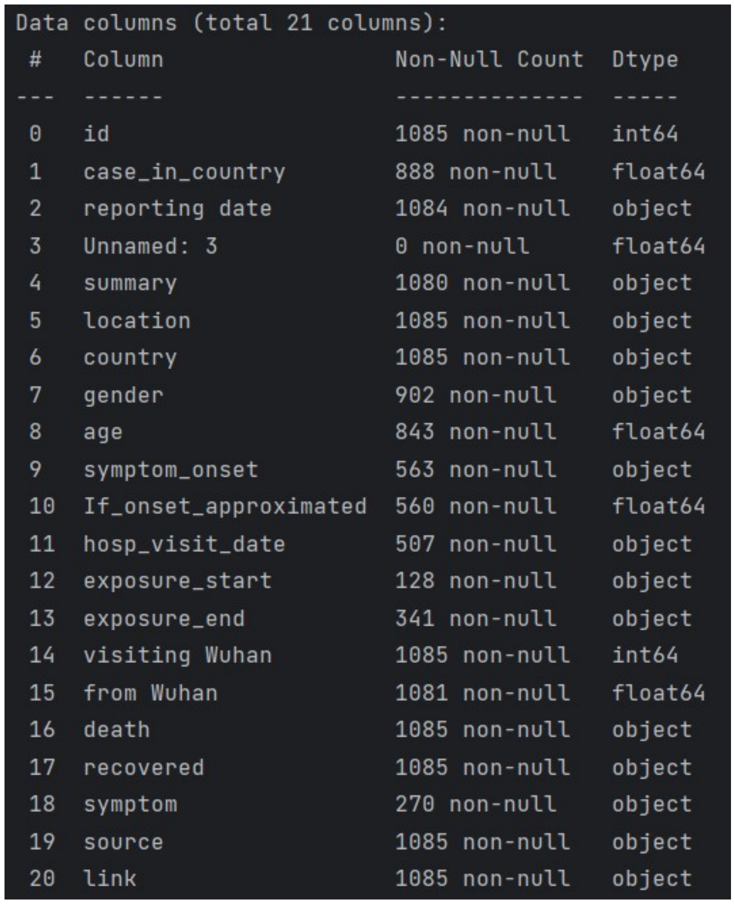
```
Missing values:
id                         0
case_in_country          197
reporting date             1
Unnamed: 3              1085
summary                    5
location                   0
country                    0
gender                   183
age                      242
symptom_onset            522
If_onset_approximated    525
hosp_visit_date          578
exposure_start           957
exposure_end             744
visiting Wuhan             0
from Wuhan                 4
death                      0
recovered                  0
symptom                  815
source                     0
link                       0
```

Figure 1: Raw dataset - the types of the columns.

I fixed it by replacing missing values with mean and mode as follows:

```
data['case_in_country'].fillna(data['case_in_country'].mean(),
    inplace=True)
data['reporting date'].fillna(data['reporting date'].mode()[0],
    inplace=True)
data['gender'].fillna(data['gender'].mode()[0], inplace=True)
data['age'].fillna(data['age'].mean(), inplace=True)
data['symptom_onset'].fillna(data['symptom_onset'].mode()[0],
    inplace=True)
data['hosp_visit_date'].fillna(data['hosp_visit_date'].mode()[0],
    inplace=True)
data['exposure_start'].fillna(data['exposure_start'].mode()[0],
    inplace=True)
data['exposure_end'].fillna(data['exposure_end'].mode()[0],
    inplace=True)
data['symptom'].fillna(data['symptom'].mode()[0], inplace=True)
```

Then, I noticed while trying to do the first exercise that some of the values could not convert one variable to another, therefore I changed the following 'object' types to other types.



```
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     1085 non-null   int64
 1   case_in_country        888 non-null    float64
 2   reporting date         1084 non-null   object
 3   Unnamed: 3             0 non-null      float64
 4   summary                1080 non-null   object
 5   location               1085 non-null   object
 6   country                1085 non-null   object
 7   gender                 902 non-null    object
 8   age                    843 non-null    float64
 9   symptom_onset          563 non-null    object
 10  If_onset_approximated  560 non-null    float64
 11  hosp_visit_date        507 non-null    object
 12  exposure_start         128 non-null    object
 13  exposure_end           341 non-null    object
 14  visiting Wuhan         1085 non-null   int64
 15  from Wuhan             1081 non-null   float64
 16  death                  1085 non-null   object
 17  recovered              1085 non-null   object
 18  symptom                270 non-null    object
 19  source                 1085 non-null   object
 20  link                   1085 non-null   object
```

Figure 2: Raw dataset - the types of the columns.

Changing types:

```
data['reporting date'] = pd.to_datetime(data['reporting date'],
    format='%d-%m-%Y', errors='coerce')
data['from Wuhan'] = data['from Wuhan'].fillna(0).astype(int)
data['death'] = pd.to_numeric(data['death'],
    errors='coerce').astype('Int64')
data['recovered'] = pd.to_numeric(data['recovered'],
    errors='coerce').astype('Int64')
data['symptom_onset'] = pd.to_datetime(data['symptom_onset'],
    format='%d-%m-%Y', errors='coerce')
data['hosp_visit_date'] = pd.to_datetime(data['hosp_visit_date'],
    format='%d-%m-%Y', errors='coerce')
```

```
data['exposure_end'] = pd.to_datetime(data['exposure_end'],
    format='%d-%m-%Y', errors='coerce')
data['exposure_start'] = pd.to_datetime(data['exposure_start'],
    format='%d-%m-%Y', errors='coerce')
data['gender'] = data['gender'].astype(str)
data['location'] = data['location'].astype(str)
data['visiting Wuhan'] = data['visiting Wuhan'].astype(int)
data['from Wuhan'] = data['from Wuhan'].astype(int)
data['reporting date'] = pd.to_datetime(data['reporting date'])
data['summary'] = data['summary'].astype(str)
data['country'] = data['country'].astype(str)
data['symptom'] = data['symptom'].astype(str)
```

The biggest issue was with the data type as many of them were in different order (from d/m/y to y/m/d), that is why I specifically changed all of them to have the same order. After those improvements of the dataset, I was able to proceed with the exercises.

1.1 Compute the correlations between the variables. Which variables are most correlated with the target (outcome)? Explain the results.

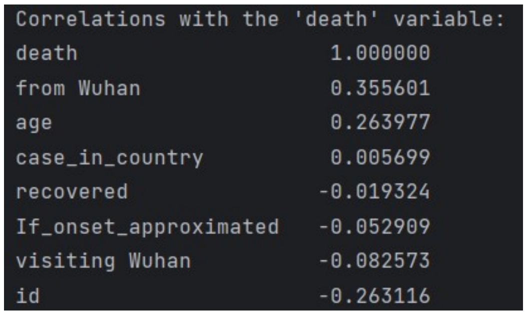
Firstly, we will need to implement the library and database as the following:

```
import numpy as np
import pandas as pd

data = pd.read_csv('COVID19_line_list_data.csv')
```

Then, as some of the values are impossible to correlate (non-numeriacal), I will only take into consideration the numerical columns. Then we can find the correlations.

```
numerical_columns = data.select_dtypes(include=np.number)

correlations = numerical_columns.corr()

target_variable = 'death'
target_correlations =
    correlations[target_variable].dropna().sort_values(ascending=False)
```

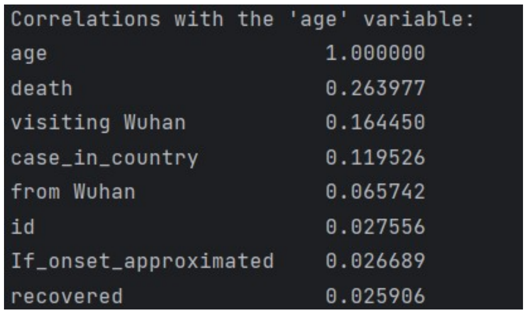
As a result we have received:



```
Correlations with the 'death' variable:
death                     1.000000
from Wuhan                0.355601
age                       0.263977
case_in_country           0.005699
recovered                -0.019324
If_onset_approximated    -0.052909
visiting Wuhan           -0.082573
id                       -0.263116
```

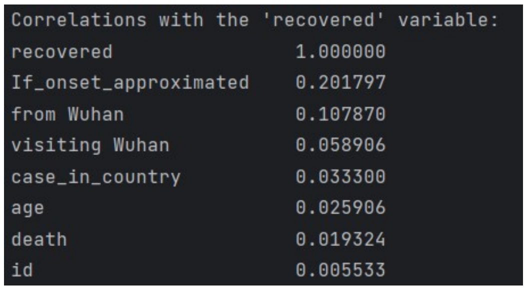
Figure 3: Results of the correlation to death.

Also, I have tried to correlate with the columns 'age' and 'recovered' and below we can find results:



```
Correlations with the 'age' variable:
age                       1.000000
death                     0.263977
visiting Wuhan            0.164450
case_in_country           0.119526
from Wuhan                0.065742
id                        0.027556
If_onset_approximated     0.026689
recovered                 0.025906
```

Figure 4: Results of the correlation to age.



```
Correlations with the 'recovered' variable:
recovered                 1.000000
If_onset_approximated     0.201797
from Wuhan                0.107870
visiting Wuhan            0.058906
case_in_country           0.033300
age                       0.025906
death                     0.019324
id                        0.005533
```

Figure 5: Results of the correlation to recovered.

After analyzing the results I have noticed that, when it comes to the correlations with the variable 'death' the most positively correlated is 'from Wuhan' (0.365). This suggests that people coming from Wuhan may have a higher likelihood of death. On the other hand, the variable most negatively correlated with 'death' is 'recovered' (-0.019), which makes absolute logical sense.

What is more, the variable most positively correlated with 'age' is 'death' (0.266). This implies that older individuals may have a higher risk of death from COVID-19, which happened during that time. The variable most positively correlated after 'death' is 'visiting Wuhan' (0.187), which means that people who visited Wuhan may tend to be older. The variable most negatively correlated with 'age' is 'from Wuhan' (0.068), meaning that people coming from Wuhan tend to be younger on average.

When it comes to 'recovered' correlations, the most positively correlated is 'If onset approximated' (0.202). This suggests that cases where the onset of symptoms is not precisely known are more likely to result in recovery. The second variable most positively correlated is 'from Wuhan' (0.108), meaning that people coming from Wuhan may have a higher chance of recovery. The variable most negatively correlated with 'recovered' is 'case in country' (0.043), which indicates that cases reported in a specific country may have a lower chance of recovery.

I summary I am proud of the results, because in my opinion they are true to the events and news that I was aware of in 2020.

1.2 Plot the dataset using scatter and analyse the obtained result. Use the PCA (Principal Component Analysis) to project the dataset.

Firstly, necessary libraries for this exercise are implemented as the following:

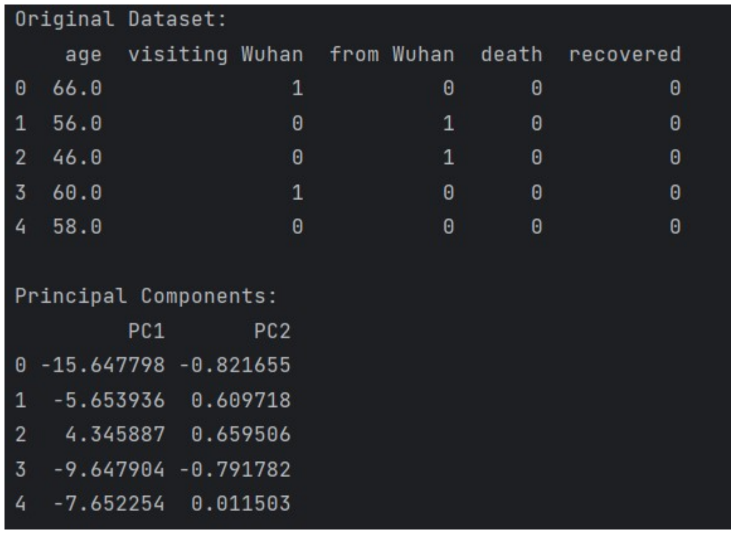
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

We can proceed with the following:

```
data = pd.read_csv('COVID19_line_list_data.csv')
```

```
selected_columns = ["age", "visiting Wuhan", "from Wuhan",
    "death", "recovered"]
df = data[selected_columns]

df = df.dropna()

pca = PCA(n_components=2)
principal_components = pca.fit_transform(df)
principal_df = pd.DataFrame(data=principal_components,
    columns=["PC1", "PC2"])

plt.scatter(principal_df["PC1"], principal_df["PC2"])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Scatter Plot")
plt.show()
```

The code above starts reading the dataset and selecting a subset of columns related to age, travel history, and outcomes. The missing values are then dropped.Next, PCA is applied to reduce the dimensionality of the data to two principal components. The resulting transformed data is stored in a new dataframe. The code then generates a scatter plot of the dataset using the two principal components.



```
Original Dataset:
    age  visiting Wuhan  from Wuhan  death  recovered
0  66.0               1           0      0          0
1  56.0               0           1      0          0
2  46.0               0           1      0          0
3  60.0               1           0      0          0
4  58.0               0           0      0          0

Principal Components:
         PC1       PC2
0 -15.647798 -0.821655
1  -5.653936  0.609718
2   4.345887  0.659506
3  -9.647904 -0.791782
4  -7.652254  0.011503
```

Figure 6: Results of the scatter in the terminal.

The visualization below provides an overview of the dataset, where each point represents an individual. The position of the points in the plot indicates their distribution based on the principal components.

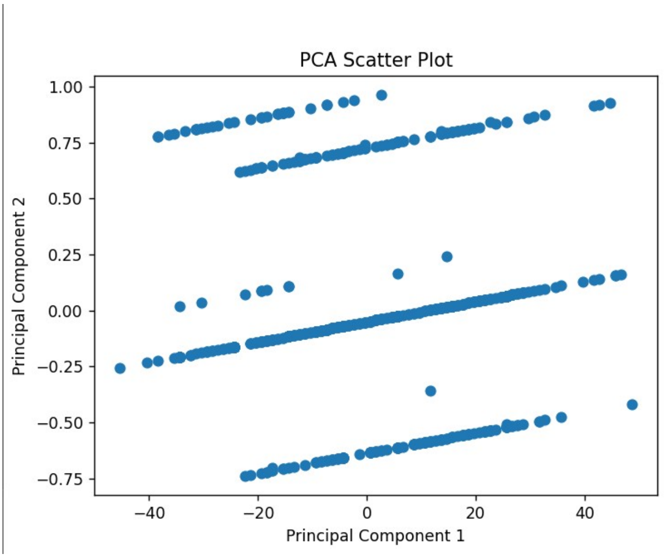


Figure 7: Results of the PCA scatter.

The scatter plot generated using the two principal components shows the distribution of the data points in a lower-dimensional space. It can be observed that the data points are not clustered tightly together, suggesting a lack of strong patterns or distinct groups in the dataset.

Analyzing it can help in detecting any noteworthy clusters, outliers, or trends within the dataset. These findings can be valuable for understanding the underlying structure and characteristics of the data.

2 Bayes Nets

Firstly, for all of the exercises below, I will need following libraries and database reading.

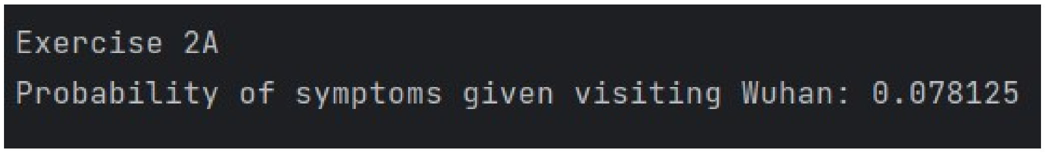
```
import pandas as
data = pd.read_csv('COVID19_line_list_data.csv')
```

2.1 What is the probability for a person to have symptoms of COVID-19 (*symptom_onset=date*) if this person visited Wuhan (visiting Wuhan = 1)? Consider that (*symptom_onset* = *N/A*) means that the patient is asymptomatic.

We will compute the probability with the following equation, in which we will divide the symptomatic Wuhan visitors by amount of Wuhan visitors:

```
wuhan_visitors = len(data[data['visiting Wuhan'] == 1])
symptomatic_wuhan_visitors =
    len(data[(data['symptom_onset'].notna()) & (data['visiting
    Wuhan'] == 1)])

probability_symptoms_given_wuhan = symptomatic_wuhan_visitors /
    wuhan_visitors
print("Probability of symptoms given visiting Wuhan:",
    probability_symptoms_given_wuhan)
```

And the result of such division is shown below:



```
Exercise 2A
Probability of symptoms given visiting Wuhan: 0.078125
```

Figure 8: Results of probability 2A.

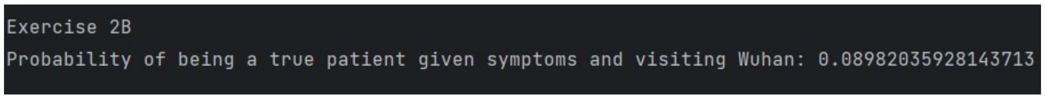
The result presents the probability of almost 0.08, which means that almost 8 out of 100 people have symptoms if they visited Wuhan. It is in my opinion very probable to have happened.

2.2 What is the probability for a person to be a true patient if this person has symptoms of COVID-19 (*symptom_onset=date*) and this person visited Wuhan?

We will compute the probability with the following equation:

```
symptomatic_cases = len(data[data['symptom_onset'].notna()])
true_positive_cases = len(data[(data['symptom_onset'].notna()) &
    (data['visiting_Wuhan'] == 1)])

probability_true_patient_given_symptoms_and_wuhan =
    true_positive_cases / symptomatic_cases
print("Probability of being a true patient given symptoms and
    visiting Wuhan:",
    probability_true_patient_given_symptoms_and_wuhan visiting
    Wuhan:", probability_symptoms_given_wuhan)
```

And the result is shown below:



```
Exercise 2B
Probability of being a true patient given symptoms and visiting Wuhan: 0.08982035928143713
```

Figure 9: Results of probability 2B.

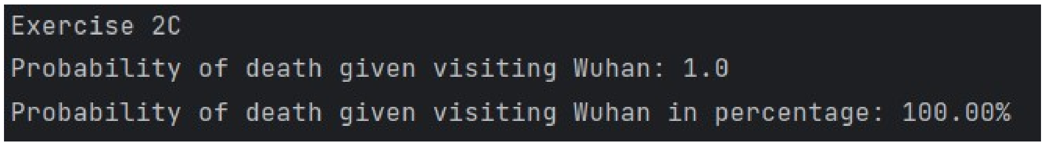
The result presents the probability of almost 0.09, which means that almost 9 out of 100 people are truly patients if they visited Wuhan and have symptoms. In my opinion it is very much likely to have happened.

2.3 What is the probability for a person to death if this person visited Wuhan?

We will compute the probability with the following equation, where we count Wuhan visitors, amount of deaths of Wuhan visitors and then we calculate the probability of that.

```
wuhan_visitors = data[data['visiting Wuhan'] == 1]
deaths = wuhan_visitors['death'].notnull().sum()
total_visitors = len(wuhan_visitors)

probability_death_given_wuhan = deaths / total_visitors
print("Probability of death given visiting Wuhan:",
    probability_death_given_wuhan)
```

And the result is shown below:



```
Exercise 2C
Probability of death given visiting Wuhan: 1.0
Probability of death given visiting Wuhan in percentage: 100.00%
```

Figure 10: Results of probability 2C.

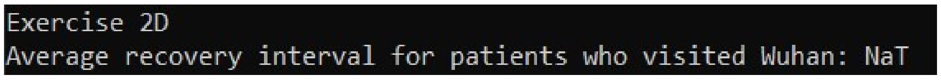
We can easily notice that the probability of death given visiting Wuhan is 1.0 which makes each person die in case he or she visits the Wuhan. It is unlikely, but probable. I think this result is only because of the dataset I am using, due to the fact that my code is in my opinion correct. Also, I could be basing on the previous probabilities, but I know that symptoms are one field and death the other, therefore I cannot rely on the previous probabilities.

2.4 Estimate the average recovery interval for a patient if this person visited Wuhan?

We will firstly Filter the data for patients who visited Wuhan and have recovery dates available, then convert recovery dates to datetime, calculate the recovery interval for each patient and in the end average recovery interval.

```
wuhan_patients = data[(data['visiting_Wuhan'] == 1) &
    (data['recovered'] == 'recovered')]

wuhan_patients['recovery_date'] =
    pd.to_datetime(wuhan_patients['recovered'])
wuhan_patients['symptom_onset'] =
    pd.to_datetime(wuhan_patients['symptom_onset'])

wuhan_patients['recovery_interval'] =
    wuhan_patients['recovery_date'] -
    wuhan_patients['symptom_onset']

average_recovery_interval =
    wuhan_patients['recovery_interval'].mean()

print("Average recovery interval for patients who visited Wuhan:",
    average_recovery_interval)
```

And the result is shown below:



```
Exercise 2D
Average recovery interval for patients who visited Wuhan: NaT
```

Figure 11: Results of probability 2D.

For Exercise 2D, the average recovery interval is calculated for patients who visited Wuhan and have recovery dates available. However, in my dataset, there are no recovery dates available for patients who visited Wuhan and have recovered, resulting in a missing value (NaT) for the average recovery interval.

I have been trying to fix this problem, but unfortunately after a lot of struggles I could not have found a solution to that result. I believe the code for calculating this probabiliy is correct, the issue is only in the dataset and its' discrepancy.

3 Machine Learning

In this part I will be using K-Nearest Neighbours (K-NN) in order to predict the outcome: patients outcome as either 'died' or 'discharged' from hospital. Firstly, we will need to add necessary libraries for the next exercises:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
    recall_score, f1_score, precision_score
```

Next, I will select features as numerical columns and outcome as 'death' and assign them. The prevent any issues while working on a data I will not consider the rows with missing values. If this step would not be done I would have received the error with input variables' inconsistent numbers of samples. Next we can split the data into training and testing sets and train the K-NN classifier. For the first time I used 3 as a number of neighbours (I can change that later), because it was default and recommended in the various sites. At the end I am able to make prediction on the testing set and overally evaluate the model using the confusion matrix, accuracy of the whole model, recall score, precision and F1 score.

```
features = ['case_in_country', 'age', 'visiting Wuhan', 'from
    Wuhan']
outcome = 'death'

data_filtered = data.dropna(subset=features + [outcome])

X_train, X_test, y_train, y_test =
    train_test_split(data_filtered[features],
    data_filtered[outcome], test_size=0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

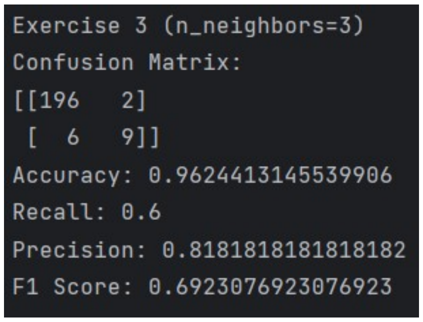
The prediction and the analyze of the results is in the next section.

3.1 The obtained results should be validated using some external indexes as prediction error (Confusion matrix and Accuracy) or others as Recall, F-Measure, (...). The obtained results should be analysed in the report and provide a solution to ameliorate the results.

Now, I will use confusion matrix, accuracy, recall and precision prediction as well as the F-measure.

```
confusion_mat = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Confusion Matrix:")
print(confusion_mat)
print("Accuracy:", accuracy)
print("Recall:", recall)
print("Precision:", precision)
print("F1 Score:", f1)
```

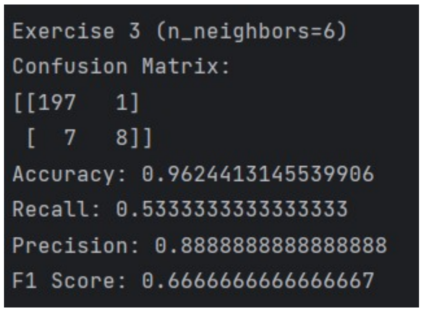
And the result is shown below:



```
Exercise 3 (n_neighbors=3)
Confusion Matrix:
[[196   2]
 [  6   9]]
Accuracy: 0.9624413145539906
Recall: 0.6
Precision: 0.8181818181818182
F1 Score: 0.6923076923076923
```

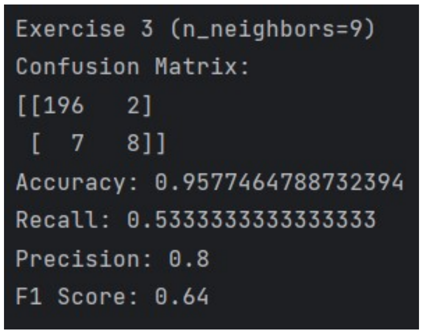
Figure 12: Results of predictions of the model data.

Then, in order to make the results more comparable I also tried using 6 and 9 neighbours and the results are shown below.



```
Exercise 3 (n_neighbors=6)
Confusion Matrix:
[[197   1]
 [  7   8]]
Accuracy: 0.9624413145539906
Recall: 0.5333333333333333
Precision: 0.8888888888888888
F1 Score: 0.6666666666666667
```

Figure 13: Results of predictions when n=6 of the model data.



```
Exercise 3 (n_neighbors=9)
Confusion Matrix:
[[196   2]
 [  7   8]]
Accuracy: 0.9577464788732394
Recall: 0.5333333333333333
Precision: 0.8
F1 Score: 0.64
```

Figure 14: Results of predictions when n=9 of the model data.

Analyzing the first result shown in the Figure 12, I have notices the following things. Firstly, the model achieves a high accuracy of approximately 96.24%, which indicates that it correctly predicts the outcome for the majority of the cases. However, accuracy alone may not be sufficient for evaluating the model's performance, therefore I have the next following measurement options. The confusion matrix shows that the model correctly predicts 196 instances of non-death cases (true negatives) and 9 instances of death cases (true positives). However, there are still 2 false positives and 6 false negatives. The next three are recall, precision and F-measure score, where recall equals to 60%, precision score - 81.81% and F-measure score - 0.692. The last one combines precision and recall into a single metric, which can help in providing a balance between precision and recall.

The next two results shown in Figure 13 and Figure 14 aren't that much better than the first result, which is natural. None of the evaluation may show better results, but they can be more precise. However the change is so small, I would not take that into consideration.

Overall, I am satisfied with the result, however I am aware that I can ameliorate the results, considering the imbalance in the classes.

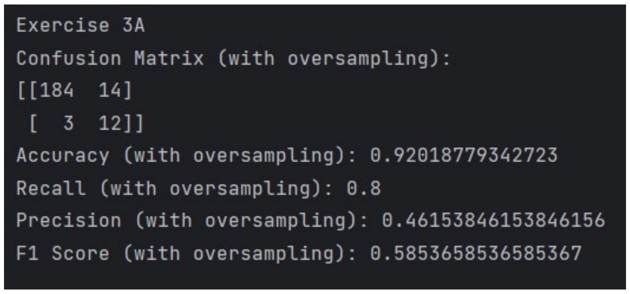
However now, I will focus on ameliorating the results above. After doing a comprehensive research I have found out that there are multiple ways to enhance my already satisfying results. Firstly, we can use feature selection, where we evaluate the selected features and consider including additional relevant features that might have a stronger correlation with the outcome variable - death. Other way is to check if the dataset is imbalanced, meaning one class dominates the other. If there is a significant class imbalance, I can consider using techniques such as oversampling or undersampling to balance the classes. The last way is to use other model, meaning that while K-NN is a popular classification algorithm, it may not be the best choice for my dataset. I could explore other classification algorithms such as decision trees, random forests, or support vector machines, which might better capture the underlying patterns in the data.

Considering all the options, I chose to implement class balancing technique. Since the death cases are the minority class, we can apply oversampling to increase the representation of the death cases in the dataset. I will import the SMOTE class from the imblearn library, which provides implementations of various oversampling techniques. Then I apply SMOTE method which will alloew me to oversample. I create clasdifier, which I train and then I can make predictions on the testing set. At the end I use the same evaluation metrics as previously. Thanks to this I can address the class imbalance issue and potentially improve the model's performance in predicting death cases.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_oversampled, y_train_oversampled =
    smote.fit_resample(X_train, y_train)

knn_oversampled = KNeighborsClassifier(n_neighbors=3)
knn_oversampled.fit(X_train_oversampled, y_train_oversampled)
```

```
y_pred_oversampled = knn_oversampled.predict(X_test)

confusion_mat_oversampled = confusion_matrix(y_test,
    y_pred_oversampled)
accuracy_oversampled = accuracy_score(y_test, y_pred_oversampled)
recall_oversampled = recall_score(y_test, y_pred_oversampled)
precision_oversampled = precision_score(y_test, y_pred_oversampled)
f1_oversampled = f1_score(y_test, y_pred_oversampled)
```

And results:



```
Exercise 3A
Confusion Matrix (with oversampling):
[[184  14]
 [  3  12]]
Accuracy (with oversampling): 0.92018779342723
Recall (with oversampling): 0.8
Precision (with oversampling): 0.46153846153846156
F1 Score (with oversampling): 0.5853658536585367
```

Figure 15: Results of ameliorating the results.

As we can see above, the results were not very much ameliorated. In my opinion it is due to the database we are using. The only satisfying result is the Recall which is 0.2 better than in the previous results. Also, due to the fact that this code was complementary and not necessary to implement and the fact that the first results were very satisfying, I will not implement any other options.

3.2 Use the Regression to predict the age of persons based on other variables. You have the choice on these explanatory variables? How you choose these variables? Compute the quality of the prediction using MSE error (Mean Squared Error).

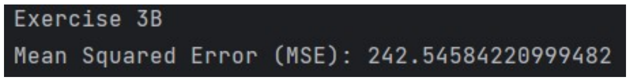
Firstly, I will implement the scikit-learn library's regression models, which is a very popular linear regression model.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Next, I will select the features that I want to use for predicting the age. Then, as previously I will need to drop rows with missing values, which will ensure that I have a clean dataset for regression. I split the data as in the previous exercise and I can create an instance of the linear regression model and train it. Based on that I can make predictions on the testing set using the trained set and calculate the MSE and show the prediction of age.

```
variables = ['case_in_country', 'visiting Wuhan', 'from Wuhan',
    'death', 'recovered']

data_filtered = data.dropna(subset=variables + ['age'])

X_train, X_test, y_train, y_test = train_test_split(
    data_filtered[variables], data_filtered['age'],
    test_size=0.2, random_state=42
)

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

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

results = pd.DataFrame({'Actual Age': y_test, 'Predicted Age':
    y_pred})

print("Mean Squared Error (MSE):", mse)
print(results)
```

And the results:



```
Exercise 3B
Mean Squared Error (MSE): 242.54584220999482
```

Figure 16: Results of MSE.

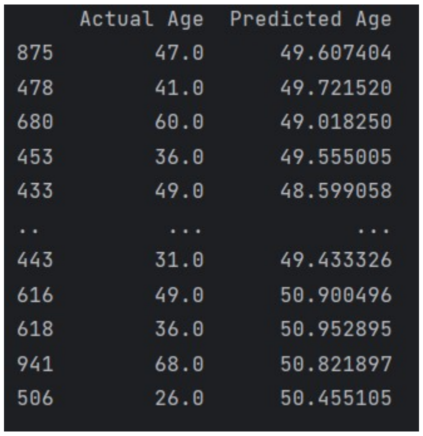

```
     Actual Age  Predicted Age
875        47.0      49.607404
478        41.0      49.721520
680        60.0      49.018250
453        36.0      49.555005
433        49.0      48.599058
..          ...            ...
443        31.0      49.433326
616        49.0      50.900496
618        36.0      50.952895
941        68.0      50.821897
506        26.0      50.455105
```

Figure 17: Results of age predictions.

As we can see, the age differences in some cases are very high (15 years) and some are very accurate (1 year). That is a reason why MSE might be considered high, becuase the lower it is, the more perfect prediction we have. The age differences are huge in the dataset I am working on, there are still a lot of other factors that made this prediction quite unsatisfying such as a lot of variables which differ.The predicted age values have a squared difference of approximately 215.3825 from the actual age values in the testing set.

3.3 Apply a clustering method (K-means) on the dataset to segment the persons in different clusters. Use the Silhouette index to find out the best number of clusters. Plot the results using scatter to visually analyse the clustering structure.

I'll use the K-means clustering algorithm to segment the persons in different clusters and determine the number of them using the Silhouette index. I selected the features for clustering, then I standardized the data and initialized the variables to store the Silhouette scores. I tried different number of clusters and computed the Silhouette score for each, but below I will only present one as example. Next, I performe K-means clustering. In the end I will plot the clustering results using scatter and display the best number of clusters and Silhouette score.

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

features = ['case_in_country', 'age', 'visiting Wuhan', 'from
    Wuhan']

data_cluster = data.dropna(subset=features)

scaler = StandardScaler()
data_cluster_scaled = scaler.fit_transform(data_cluster[features])

best_silhouette_score = -1
best_num_clusters = -1

for num_clusters in range(2, 11):
    kmeans = KMeans(n_clusters=num_clusters, random_state=42)
    labels = kmeans.fit_predict(data_cluster_scaled)
    silhouette_avg = silhouette_score(data_cluster_scaled, labels)

    if silhouette_avg > best_silhouette_score:
        best_silhouette_score = silhouette_avg
        best_num_clusters = num_clusters

kmeans = KMeans(n_clusters=best_num_clusters, random_state=42)
labels = kmeans.fit_predict(data_cluster_scaled)

plt.scatter(data_cluster_scaled[:, 0], data_cluster_scaled[:, 1],
    c=labels)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-means Clustering Results ({}
    clusters)'.format(best_num_clusters))
plt.show()

print("Best number of clusters:", best_num_clusters)
print("Silhouette score:", best_silhouette_score)
```

And the results:

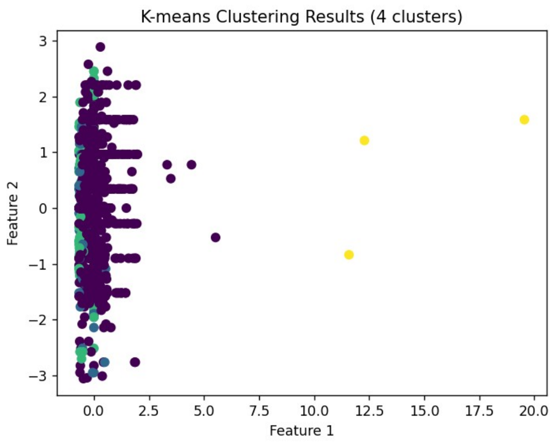
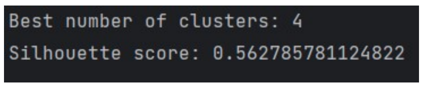


Figure 18: Plot of the results using scatter.



```
Best number of clusters: 4
Silhouette score: 0.562785781124822
```

Figure 19: Results.

As shown above, the best number of clusters is 4 (which has been checked by trying different values), and the Silhouette score is 0.5628, which means that I have a well-separated and compact clustering structure.

4 Improving the results and Theoretical formalism

4.1 The data is unbalanced. You can balance it by reducing randomly the majority class. Assume that you extract randomly samples that are balanced. How the prediction results will change?

Balancing the data can potentially help to alleviate the issue of class imbalance, where one class has significantly more samples than the other. By reducing the majority class, I can expect equal representation to both classes, which could lead further on to more accurate predictions.

To balance the data I would use the K-nearest neighbors (K-NN) classifier, which would have an equal number of samples from both classes to learn from during training. This could improve the classifier's ability to discriminate between the classes and make more accurate predictions. The classifier would be less biased towards the majority class and could better capture the characteristics and patterns of the minority class.

As a result, the prediction results may show improved performance metrics such as accuracy, recall, precision, and F1 score, the one which I have used in my report.

However, it's important to notice that the impact on the prediction results can vary depending on the specific dataset. Balancing the data is a common technique, but it may not always guarantee significant improvements in the prediction results.

4.2 How you can better manage the missing values?

To better manage missing values in the dataset, there are multiple ways to consider. Firstly, analyzing the missing data pattern and understanding it to determine if the missing values are Missing Completely at Random (MCAR), Missing at Random (MAR), or Missing Not at Random (MNAR). Depending on where the problem lies, the proper approach can be applied. The other way is the 'Imputation Techniques', which fills the missing values with estimated values using techniques such as:

a. Mean/Median/Mode Imputation - replacing missing values with the mean, median, or mode, simple method but assumes no relationship between the missing values and other variables

b. Regression Imputation - predicting missing values using regression models based on other variables

c. Multiple Imputation - generating multiple imputed datasets by estimating missing values based on observed data and their relationships

d. K-Nearest Neighbors (K-NN) Imputation - imputing missing values using the values of the nearest neighbors

d. Interpolation - using interpolation techniques like linear or spline interpolation to estimate missing values based on the trend or pattern in the data

Other techniques which could be potentially useful, but aren't that popular or common are: Handling Categorical Variables, Dropping Rows or Columns (which may be appropriate to drop rows or columns with a large number of missing values if they are not critical for analysis), or other advanced techniques (MissForest or Generative Adversarial Networks (GANs), which involves machine learning algorithms and data augmentation).

Choosing the appropriate method depends on the dataset we are using, amount of data in it and type of variables and analyzis we need. Testing each method might be the most promising way to manage missing values. It is essential to choose the appropriate method for managing missing.

4.3 To find the best parameters for the models, the Grid-search algorithm can be used which is available in scikit-learn library. Explain the algorithm and use it for the learning models to find the best parameters.

The Grid Search algorithm is a technique used to find the best combination of hyperparameters (the settings or configurations that are not learned from the data but are set before the learning process begins) for a machine learning model.

The Grid Search algorithm searches through a specified grid of hyperparameter values and evaluates the model's performance using cross-validation. It systematically tries all possible combinations of hyperparameters and selects the combination that produces the best performance metric. To apply this method we need to define a model, define the hyperparamter and the grid. Next, define the performance metric and then perform this grid search. At the end we select the best parameters and evaluate the performance.

By using the Grid Search algorithm, a user can systematically explore the hyperparameter space and find the combination that maximizes the model's performance. This helps in fine-tuning the model and improving its predictive capabilities.

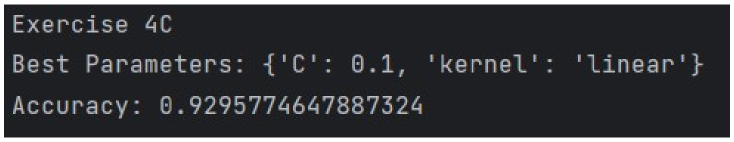
Below is a use of it to find the best parameters from my model:

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

```
svm = SVC()

param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}

scoring = 'accuracy'

grid_search = GridSearchCV(svm, param_grid, scoring=scoring, cv=5)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print("Best Parameters:", best_params)

best_model = grid_search.best_estimator_
accuracy = best_model.score(X_test, y_test)
print("Accuracy:", accuracy)
```

In this example, the grid search performs with different values of the C hyperparameter and two different kernel types for the SVM model. The performance metric used is accuracy, and the grid search is performed using 5-fold cross-validation. The best parameters are then accessed using the bestparams attribute, and the performance of the model with the best parameters is evaluated on the test set.

And below are the results:



```
Exercise 4C
Best Parameters: {'C': 0.1, 'kernel': 'linear'}
Accuracy: 0.9295774647887324
```

Figure 20: Results of grid search.

By utilizing the Grid Search algorithm, you can efficiently explore the hyperparameter space and find the optimal configuration for your machine learning models.

4.4 Give the algorithmically (mathematical) formalism of the method which give the best results. Explain all the parameters of the used method and their impact on the results. Some comparison with public results should me made to conclude the project.

Algorithmical formalism of the method that gives the best results looks as follows:

1. Input:
    (a) Training dataset: a set of labeled instances with features and corresponding target labels.
    (b) Testing dataset: a set of instances with features but without target labels.
    (c) Number of neighbors (K): a parameter that specifies the number of nearest neighbors to consider.
2. For each instance in the testing dataset:
    (a) Compute the distance between the instance and all instances in the training dataset using a distance metric (e.g., Euclidean distance).
    (b) Select the K nearest neighbors based on the computed distances.
3. Predict the label of the instance based on the majority vote of the labels of the K nearest neighbors:
    (a) If the problem is binary classification, the label with the majority vote will be the predicted class.
    (b) If the problem is multi-class classification, you can use voting schemes like majority voting or weighted voting.

The impact of the parameters in the K-NN method is following:

1. Number of neighbors (K) - The number of surrounding instances taken into consideration for classification is determined by the item Number

of neighbors (K). Overfitting may result from a model that is more noise-sensitive and has a lower K value. However, a higher K value may soften the decision border, which can result in underfitting. The dataset will determine the best value of K, which should be carefully selected through testing or cross-validation.

2. Distance metric - It describes how similar or dissimilar examples are to one another. The choice of distance metric can have a big effect on the outcomes. Euclidean distance is the most widely used distance metric, however there are also Manhattan distance, Minkowski distance, and cosine similarity. The type of data and the problem domain determine which distance measure is most appropriate.

To conclude the project, there is a possibility to can compare the performance of my K-NN model with publicly available results on similar datasets or benchmarks. I can compare metrics such as accuracy, precision, recall, F1 score, or any other relevant evaluation metric.

During the research I have found a few similar datasets and articles, which consider similar topic to mine. However, I would like to mention [3] first, because this is the article that answered almost all of my questions and addressed all problems. Not only it analyzes the data that is very similar to mine, but it takes into consideration the same issues, such as age, uses SVM and naive Bayes models, trains models and test them and then represents the result. Majority of the represented values were very similar to the ones I have achieved. They are not perfectly identical, but the steps taken, the way of conducting the resarch was very similar to mine, therefore I can assume my results are promising.

The other articles weren't as effective and useful, but covered the same topic area are: [1] and [2]. They answered the same questions and presented the same data asthe first article. However, they presented a lot more data diagrams, which explained and analyzed the area that I haven't analyzed, which are: gender influence, travel influence and more on spred of Covid-19.

In summary, based on my results and the results I have been presented to on various articles and research papers meets my expectation and I can proudly state that are very similar to my paper. Therefore, in my opinion I have completed my report with success.

Conclusions

In this project, artificial intelligence techniques were used to assess and interpret a dataset about the COVID-19 infection. I concentrated on four important areas throughout the project: dataset analysis, Bayes Nets, machine learning, enhancing the outcomes, and theoretical formalism. Each section added important knowledge and helped us comprehend how the COVID-19 infection spread and what effect it had.

I handled missing values, outliers, and outlier handling during the dataset analysis, as well as exploring relationships between variables and extracting statistical data. I was able to learn more about the connections between many components and the outcome variable thanks to this investigation. Indicators that were most highly correlated with the objective variable were emphasized by the correlations between the variables, offering useful data for more research and prediction.

I was able to construct predictions based on the dataset by using Bayes Nets. I calculated probabilities for a variety of scenarios, including the likelihood that someone would experience COVID-19 symptoms if they visited Wuhan, the likelihood that someone would actually be a patient based on symptoms and travel history, the likelihood that someone would pass away if they visited Wuhan, and the estimation of typical recovery times. Understanding the risk factors and probable effects linked to the COVID-19 infection can be aided by these predictions.

Using machine learning methods like K-Nearest Neighbors (K-NN) and Bayes Classification, patient outcomes were predicted and age estimates were made based on other characteristics. Utilizing external indices, such as prediction error and accuracy, the results were confirmed. Regression analysis provided

One of the biggest challenges I encountered in this project was handling the incomplete dataset with missing values and its structures. The presence of missing values required careful consideration in order to ensure accurate analysis and modeling. Replacing missing values with mean, median, or mode were employed was very essential. Although challenging, this aspect of the project provided valuable insights into data preprocessing and handling real-world data imperfections.

In conclusion, this project contributed to a better understanding of the COVID-19 infection and its spread by leveraging artificial intelligence methods. The dataset analysis, Bayes Nets, machine learning, and efforts to improve the results provided valuable insights and predictions. The biggest struggle I encountered in this project was handling the incomplete dataset with missing values and its structures. However, this challenge proved to be a valuable learning experience in data preprocessing and real-world data analysis.

References

[1] Shandar Ahmad. Potential of age distribution profiles for the prediction of covid-19 infection origin in a patient group. *Informatics in Medicine Unlocked*, 20:100364, 2020.

[2] A. Hoque, A. Malek, and K.M.R.A. Zaman. Data analysis and prediction of the covid-19 outbreak in the first and second waves for top 5 affected countries in the world. *Nonlinear Dynamics*, 109:77–90, 2022.

[3] L.J. Muhammad, E.A. Algehyne, S.S. Usman, et al. Supervised machine learning models for prediction of covid-19 infection using epidemiology dataset. *SN Computer Science*, 2(11), 2021.