Prediction of COVID-19 diagnosis

(Odin School – Capstone Project2)

Prepared By:

Name: Neha Koti

Batch: DS227B

Student ID: S3554

Contents

Prediction of COVID-19 diagnosis

Business Problem: Prediction of COVID-19 diagnosis based on symptoms.

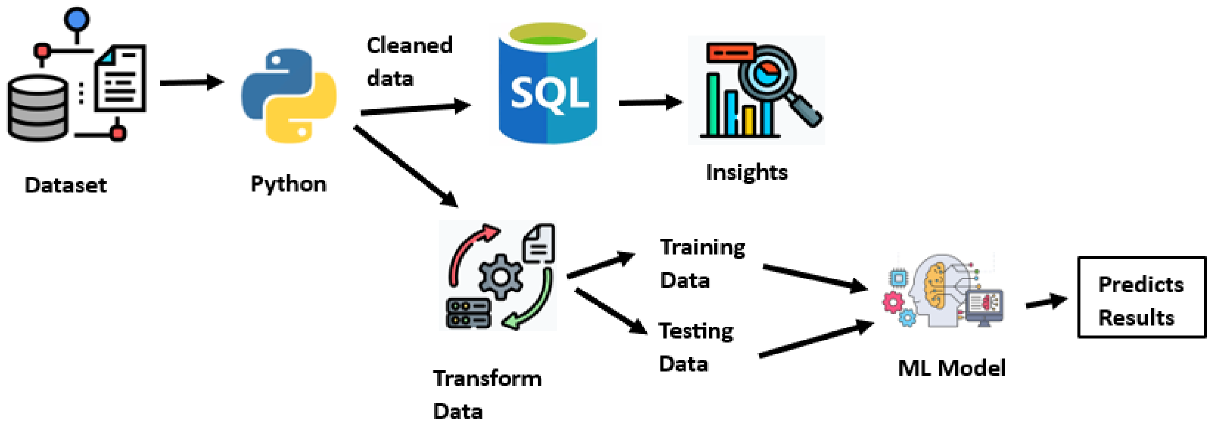
Given dataset named "Covid19" contains 11 columns. This dataset contains information about different symptoms and covid test result details. We need to predict accurate diagnosis of Covid-19 using Machine learning (ML) models. The current dataset has been downloaded from Kaggle and contains around 2,78,848 individuals who have gone through the RT-PCR test.

Proposal:

"Covid-19 Positive" is the most negative word heard in the year 2019. Detection of covid-19 was very important to stop the spread of pandemic. There was severe shortage of doctors as more people got infected. In these kinds of situations, if we could develop a Machine Learning model which could predict and diagnose Covid-19 accurately would help doctors and also lessen the burden on healthcare systems.

Tools Required: MySQL, Python, PowerBI

Using Python packages like NumPy, Pandas we do data cleaning for the given dataset first. We deal with all wrong data, missing and null values in data cleaning. Then this cleaned data is loaded into MySQL and we try to get few useful insights from various queries. Later, we transform the data such that it is suitable for ML models. This transformed data is splitted into two parts: **training dataset, testing dataset**. We train Machine Learning model using training dataset. After training the model we give testing dataset to the model. ML model predicts the output as covid-19 positive or negative based on its learning from training dataset (which contains various symptoms). We calculate few metrics like Accuracy, precision etc to know the ML model performance. We develop 4 ML models namely: **Decision tree, Random Forest, Logistic Regression, K-Nearest Neighbour models/Algorithms.** Lastly, we compare these four models to know which ML model works best and gives better prediction results.



Proposal for predicting Covid-19 Data

Questions:

1. **Why is your proposal important in today's world? How predicting a disease accurately can improve medical treatment?**

There is an exponential increase in world's population but resources are limited. When any epidemic or pandemic occurs, the effect is even more severe. If we could use technology and tools to solve these problems it will be of great help to people. We can create Machine Learning models train them in such a way that they can predict results very accurately. By doing this it will help doctors for cross checking diagnosis and for faster detection and prevention of any diseases.

2. **How is it going to impact the medical field when it comes to effective screening and reducing health care burden.**

Machine Learning plays a vital role and has great impact on medical field. Using machine learning we can achieve few things like early disease detection, personalized medicine, drug discovery, cost reduction etc.

Machine learning models can analyze huge amounts of medical data to identify patterns associated with diseases. For example, textual data like patient records and even images like CT-scans, X-rays etc can be analyzed for early detection of cancer at an earlier stage only. This is crucial for better patient outcomes and reducing the burden on healthcare systems.

3. **If any, what is the gap in the knowledge or how your proposed method can be helpful if required in future for any other disease.**

My proposed method can be helpful in future also. Instead of covid-19 dataset if we give other different dataset say for example cancer dataset or some other disease related patient records, we can still train Machine learning model and predict the results. The process is same for other datasets also.

Approach/ Implementation:

We need to first understand the data then clean the data. Cleaned data is loaded into SQL and retrieve some useful insights through various queries. Later we transform the data and design machine learning models. We train the model using training dataset and then pass the testing dataset for prediction. Using few performance metrics, we can tell which model works best for given dataset.

Step 1: Data Understanding and Exploration

In this step we try to understand raw data and which are important & unimportant columns/features of the dataset and their datatypes, descriptive statistics measures like mean, median, mode etc for numerical datatypes and unique values of categorical datatype columns.

Our Problem statement was "Prediction of COVID-19 diagnosis based on symptoms" (a dataset is given to us we need to predict whether a person is affected with covid-19 or not based on symptoms. **output**: Yes/No)

The following are the columns/features used by the ML models:

A. **Basic information:**

1. ID (Individual ID) --- int
2. Sex (male/female) --- categorical
3. Age ≥60 above years (true/false) --- categorical
4. Test date (date when tested for COVID) --- date

B. **Symptoms**:

5. Cough (true/false) --- categorical
6. Fever (true/false) --- categorical
7. Sore throat (true/false) --- categorical
8. Shortness of breath (true/false) --- categorical
9. Headache (true/false) --- categorical

C. **Other information:**

10. Known contact with an individual confirmed to have COVID-19 (true/false) --- categorical

D. **Covid report**

11. Corona positive or negative --- categorical

The following Python code is executed in Jupyter notebook. The below code explores various columns and their datatypes of the given dataset. Renaming of columns where ever necessary is done. And also descriptive Statistics measures are calculated for numerical columns and all possible unique values are found out for categorical columns.

```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns
import plotly.express as px
# the below line makes the images to display always without loading dataset
# whenever you open always
%matplotlib inline
```

```
import warnings
warnings.filterwarnings("ignore") #ignoring warnings
```

```
df_original = pd.read_csv('covid_detection.csv') # reading csv file
df_original.head() # displays first five rows of the dataset
```

	Ind_ID	Test_date	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	Corona	Age_60_above	Sex	Known_contact
0	1	11-03-2020	TRUE	FALSE	TRUE	FALSE	FALSE	negative	None	None	Abroad
1	2	11-03-2020	FALSE	TRUE	FALSE	FALSE	FALSE	positive	None	None	Abroad
2	3	11-03-2020	FALSE	TRUE	FALSE	FALSE	FALSE	positive	None	None	Abroad
3	4	11-03-2020	TRUE	FALSE	FALSE	FALSE	FALSE	negative	None	None	Abroad
4	5	11-03-2020	TRUE	FALSE	FALSE	FALSE	FALSE	negative	None	None	Contact with confirmed

```
df = df_original.copy() # copy of original dataset
df.head()# displays first five rows of the dataset
```

	Ind_ID	Test_date	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	Corona	Age_60_above	Sex	Known_contact
0	1	11-03-2020	TRUE	FALSE	TRUE	FALSE	FALSE	negative	None	None	Abroad
1	2	11-03-2020	FALSE	TRUE	FALSE	FALSE	FALSE	positive	None	None	Abroad
2	3	11-03-2020	FALSE	TRUE	FALSE	FALSE	FALSE	positive	None	None	Abroad
3	4	11-03-2020	TRUE	FALSE	FALSE	FALSE	FALSE	negative	None	None	Abroad
4	5	11-03-2020	TRUE	FALSE	FALSE	FALSE	FALSE	negative	None	None	Contact with confirmed

```
df.tail()# displays last five rows of the dataset
```

	Ind_ID	Test_date	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	Corona	Age_60_above	Sex	Known_contact
278843	278844	30-04-2020	False	False	False	False	False	positive	None	male	Other
278844	278845	30-04-2020	False	False	False	False	False	negative	None	female	Other
278845	278846	30-04-2020	False	False	False	False	False	negative	None	male	Other
278846	278847	30-04-2020	False	False	False	False	False	negative	None	male	Other
278847	278848	30-04-2020	False	False	False	False	False	negative	None	female	Other

```
df.shape #displays no.ofrows and columns of the dataset
```

OP

```
(278848, 11)
```

Observation: we have 278848 rows and 11 columns

```
df.columns # displays all column names
```

```
Index(['Ind_ID', 'Test_date', 'Cough_symptoms', 'Fever', 'Sore_throat',
       'Shortness_of_breath', 'Headache', 'Corona', 'Age_60_above', 'Sex',
       'Known_contact'],
      dtype='object')
```

```
# renaming necessary columns which do not have clear names
df = df.rename(columns = {'Ind_ID':'ID','Corona':'Test_result'})
```

```
df.columns # checking renamed column names
```

```
Index(['ID', 'Test_date', 'Cough_symptoms', 'Fever', 'Sore_throat',
       'Shortness_of_breath', 'Headache', 'Test_result', 'Age_60_above', 'Sex',
       'Known_contact'],
      dtype='object')
```

Observation: here 'Corona' column is renamed as 'Test_result' similarly 'Ind_ID' column is renamed as 'ID'

```
df.info() # displays information about columns and its datatypes and non-null values count
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278848 entries, 0 to 278847
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ID                   278848 non-null  int64
 1   Test_date            278848 non-null  object
 2   Cough_symptoms       278848 non-null  object
 3   Fever                278848 non-null  object
 4   Sore_throat          278848 non-null  object
 5   Shortness_of_breath  278848 non-null  object
 6   Headache             278848 non-null  object
 7   Test_result          278848 non-null  object
 8   Age_60_above         278848 non-null  object
 9   Sex                  278848 non-null  object
 10  Known_contact        278848 non-null  object
dtypes: int64(1), object(10)
memory usage: 23.4+ MB
```

Observation: except ID (integer type) rest all other columns are of string datatype

```
df.dtypes #displays just datatypes of columns
```

```
ID                      int64
Test_date              object
Cough_symptoms         object
Fever                  object
Sore_throat            object
Shortness_of_breath    object
Headache               object
Test_result            object
Age_60_above           object
Sex                    object
Known_contact          object
dtype: object
```

```
#display descriptive stats info for all numeric datatype columns
df.describe()
```

	ID
count	278848.000000
mean	139424.500000
std	80496.628269
min	1.000000
25%	69712.750000
50%	139424.500000
75%	209136.250000
max	278848.000000

```
# it displays desccriptive stats information.
#include='all' will consider all datatypes even categorical
df.describe(include='all')

#count: no.of rows, unique: displays no.of unique values in that column,
#top: displays most repeated value, freq: displays count of values
```

	ID	Test_date	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	Test_result	Age_60_above	Sex	Known_contact
count	278848.000000	278848	278848	278848	278848	278848	278848	278848	278848	278848	278848
unique	NaN	51	5	5	5	5	5	3	3	3	3
top	NaN	20-04-2020	False	False	False	False	False	negative	None	female	Other
freq	NaN	10921	127531	137774	212584	212842	212326	260227	127320	130158	242741
mean	139424.500000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	80496.628269	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	69712.750000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	139424.500000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	209136.250000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	278848.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Obersation: We have more no.of female patients records in this dataset

```
df.nunique() #displays count of unique values for each column
```

```
ID                     278848
Test_date                  51
Cough_symptoms              5
Fever                       5
Sore_throat                 5
Shortness_of_breath         5
Headache                    5
Test_result                 3
Age_60_above                3
Sex                         3
Known_contact               3
dtype: int64
```

```
# value_counts() displays unique values along with frequency
#for the given column
df.Cough_symptoms.value_counts()
# or we can execute as below as
# df[Cough.symptoms].value_counts()
```

```
False     127531
FALSE     108837
TRUE       21983
True       20245
None         252
Name: Cough_symptoms, dtype: int64
```

```
# columns is a list containing all column names
columns = ['Cough_symptoms','Fever','Sore_throat','Shortness_of_breath','Headache','Test_result','Age_60_above','Sex','Known_cont
```

```
#displays unique values and its count for each and every column
for i in columns:
    print(i)
    print(df[i].value_counts())
    print()
```

```
Cough_symptoms
False    127531
FALSE    108837
TRUE      21983
True      20245
None        252
Name: Cough_symptoms, dtype: int64

Fever
False    137774
FALSE    119070
TRUE      11750
True      10002
None        252
Name: Fever, dtype: int64

Sore_throat
False    212584
FALSE     64337
TRUE       1198
True        728
None          1
Name: Sore_throat, dtype: int64

Shortness_of_breath
False    212842
FALSE     64428
TRUE       1107
True        470
None          1
Name: Shortness_of_breath, dtype: int64


Test_result
negative    260227
positive     14729
other         3892
Name: Test_result, dtype: int64

Age_60_above
None    127320
No      125703
Yes      25825
Name: Age_60_above, dtype: int64

Sex
female    130158
male      129127
None       19563
Name: Sex, dtype: int64

Known_contact
Other                     242741
Abroad                     25468
Contact with confirmed     10639
Name: Known_contact, dtype: int64
```

Observation: In columns True and False values are written in two different spelling types. we need update these typing errors. We also have null (None) values in few columns. we need to remove them.

After this Data understanding step, we got to know that all columns except id and date columns are categorical datatype columns. And each categorical column mostly has True/False unique values. Only "Known_contact" column has three unique values i.e., Abroad, other, contact with confirmed values. And "ID" column not so important column and all other columns are important for further analysis and model creation.

Step 2: Data Preparation / Feature engineering

In this step we do data cleaning, data wrangling/data transformation, Feature Selection.

i) **Data cleaning:** In this sub-step we find wrong data, wrong data type, missing values and outliers in all columns and eliminate these issues.

The following is the python code for data cleaning:

Missing Values/wrong values Detection and Removal

```
df.isnull().sum()
```

```
ID                     0
Test_date              0
Cough_symptoms         0
Fever                  0
Sore_throat            0
Shortness_of_breath    0
Headache               0
Test_result            0
Age_60_above           0
Sex                    0
Known_contact          0
dtype: int64
```

observation: Here it is showing 0 as all columns are actegorical but have "None" values.we need to replace them with mode of that column. then these missing values will be gone.

```
# columns is a list containing all column names
columns = ['Cough_symptoms','Fever','Sore_throat','Shortness_of_breath','Headache']
```

```
# for loop for replacing "TRUE" with "True" and "FALSE" with "False" values
for i in columns:
    print(i,": ")
    print(df[i].value_counts())
    df[i].replace('FALSE',False,inplace=True) # replacing FALSE  values
    df[i].replace('TRUE',True,inplace=True) # replacing TRUE values
    a = list(df[i].mode())
    df[i].replace('None',a[0],inplace=True)
    print(df[i].value_counts())
    print()
```

```
Cough_symptoms :
False     127531
FALSE     108837
TRUE       21983
True       20245
None         252
Name: Cough_symptoms, dtype: int64
False     236620
True       42228
Name: Cough_symptoms, dtype: int64

Fever :
False     137774
FALSE     119070
TRUE       11750
True       10002
None         252
Name: Fever, dtype: int64
False     257096
True       21752
Name: Fever, dtype: int64
```

```
Sore_throat :
False     212584
FALSE      64337
TRUE        1198
True         728
None           1
Name: Sore_throat, dtype: int64
False     276922
True        1926
Name: Sore_throat, dtype: int64

Shortness_of_breath :
False     212842
FALSE      64428
TRUE        1107
True         470
None           1
Name: Shortness_of_breath, dtype: int64
False     277271
True        1577
Name: Shortness_of_breath, dtype: int64
```

```
Headache :
False     212326
FALSE      64107
TRUE        1428
True         986
None           1
Name: Headache, dtype: int64
False     276434
True        2414
Name: Headache, dtype: int64
```

Observation: Test_result column replacing missing values with mode

```
!pip install prettytable
```

```
from prettytable import PrettyTable # printing all values in tabular format
x = PrettyTable()
columns = ['Cough_symptoms','Fever','Sore_throat','Shortness_of_breath','Headache'] #only categorical list of columns
for i in columns:
    m = df[i].mode() # mode
    x.field_names = ["column-Name","Mode"]
    x.add_row([i,m])
print(x)
```

```
+---------------------+----------------------------------------+
|     column-Name     |                  Mode                  |
+---------------------+----------------------------------------+
|    Cough_symptoms   |              0     False               |
|                     |   Name: Cough_symptoms, dtype: bool    |
|        Fever        |              0     False               |
|                     |       Name: Fever, dtype: bool         |
|     Sore_throat     |              0     False               |
|                     |    Name: Sore_throat, dtype: bool      |
| Shortness_of_breath |              0     False               |
|                     | Name: Shortness_of_breath, dtype: bool |
|       Headache      |              0     False               |
|                     |      Name: Headache, dtype: bool       |
+---------------------+----------------------------------------+
```

```
# unique values in specified column
df['Test_result'].unique()
```

```
array(['negative', 'positive', 'other'], dtype=object)
```

```
#count of unique values in specified column
df['Test_result'].value_counts()
```

```
negative    260227
positive     14729
other         3892
Name: Test_result, dtype: int64
```

```
# mode as columns are categorical
df['Test_result'].mode()
```

```
0    negative
Name: Test_result, dtype: object
```

```
df['Test_result'].replace('other','negative',inplace=True)
```

```
#count of unique values in specified column
df['Test_result'].value_counts()
```

```
negative    264119
positive     14729
Name: Test_result, dtype: int64
```

Age_60_above column replacing missing values with mode

```
# unique values in specified column
df['Age_60_above'].unique()
```

```
array(['None', 'No', 'Yes'], dtype=object)
```

```
#count of unique values in specified column
df['Age_60_above'].value_counts()
```

```
None    127320
No      125703
Yes      25825
Name: Age_60_above, dtype: int64
```

```
# mode as columns are categorical
df['Age_60_above'].mode()
```

```
0    None
Name: Age_60_above, dtype: object
```

```
df['Age_60_above'].replace('None','No',inplace=True)
```

```
#count of unique values in specified column
df['Age_60_above'].value_counts()
```

```
No     253023
Yes     25825
Name: Age_60_above, dtype: int64
```

sex column replacing missing values with mode

```
# unique values in specified column
df['Sex'].unique()
```

```
array(['None', 'male', 'female'], dtype=object)
```

```
#count of unique values in specified column
df['Sex'].value_counts()
```

```
female    130158
male      129127
None       19563
Name: Sex, dtype: int64
```

```
# mode as columns are categorical
df['Sex'].mode()
```

```
0    female
Name: Sex, dtype: object
```

```
df['Sex'].replace('None','female',inplace=True)
```

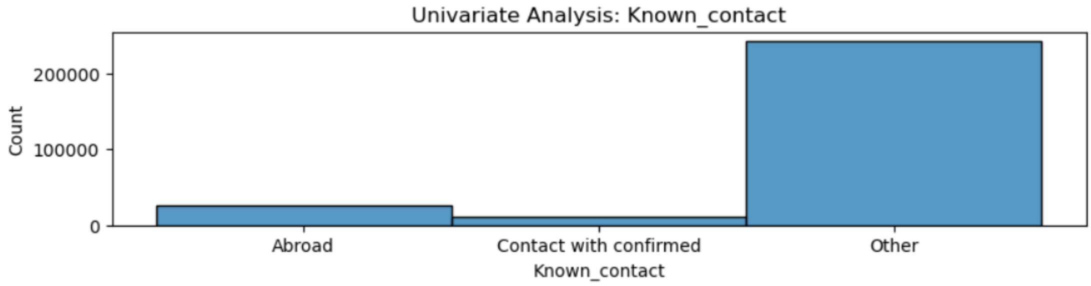
```
#count of unique values in specified column
df['Sex'].value_counts()
```

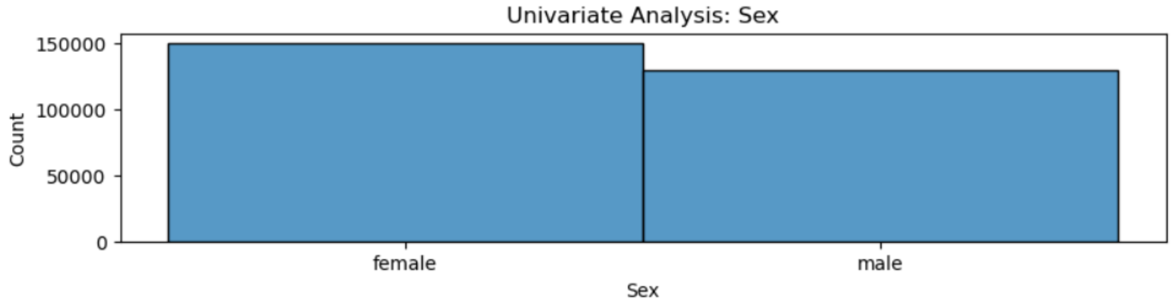
```
female    149721
male      129127
Name: Sex, dtype: int64
```

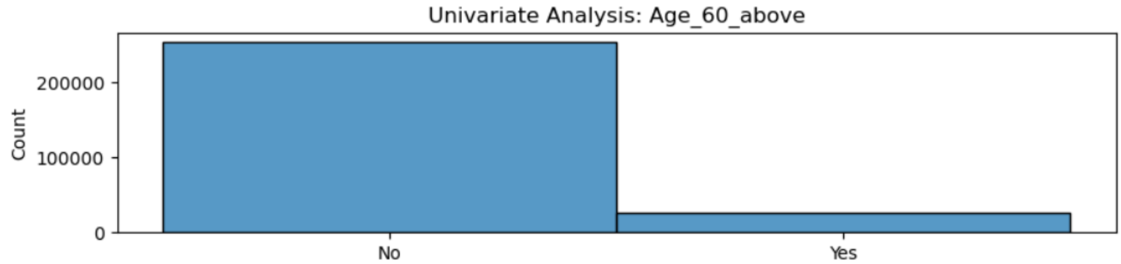
In the given dataset we do not have any blank values but null values are represented as "None" so all null values are replaced statistically with mode of the column as all columns are of categorical datatype. We do not have any numerical datatype columns so no outliers are present in this dataset. To confirm for outliers existence, we can go for data visualization.

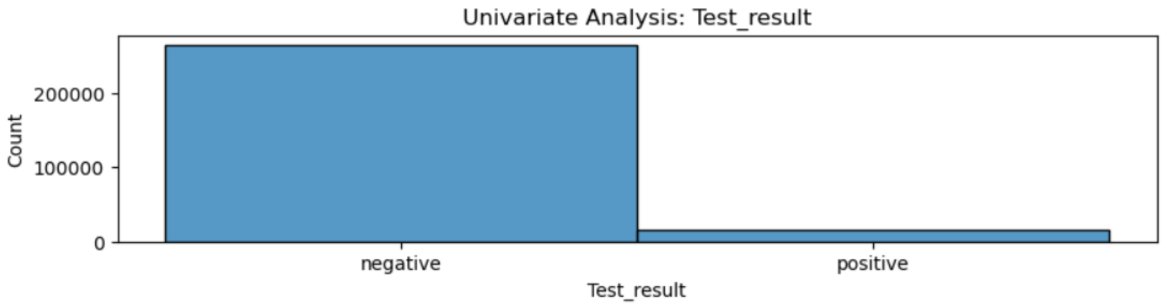
Data Visualization

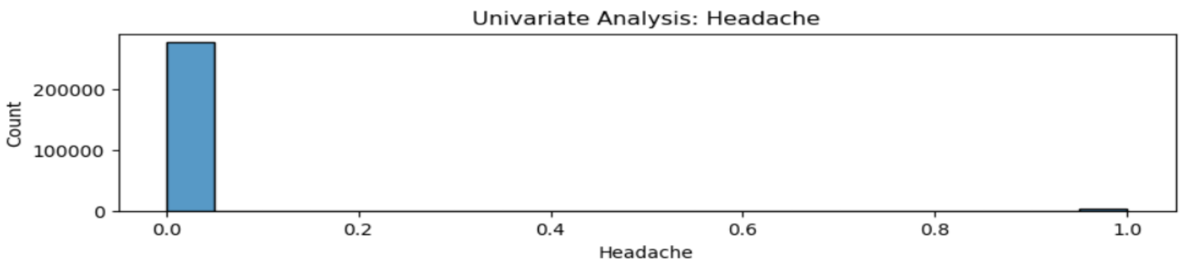
```
# the below code displays graphs for count Vs each column unique values data
for columns in df.columns:
    plt.figure(figsize=(10,2))
    sns.histplot(df[columns])
    plt.title(f'Univariate Analysis: {columns}')
    plt.xlabel(columns)
    plt.ylabel('Count')
    plt.show()
```

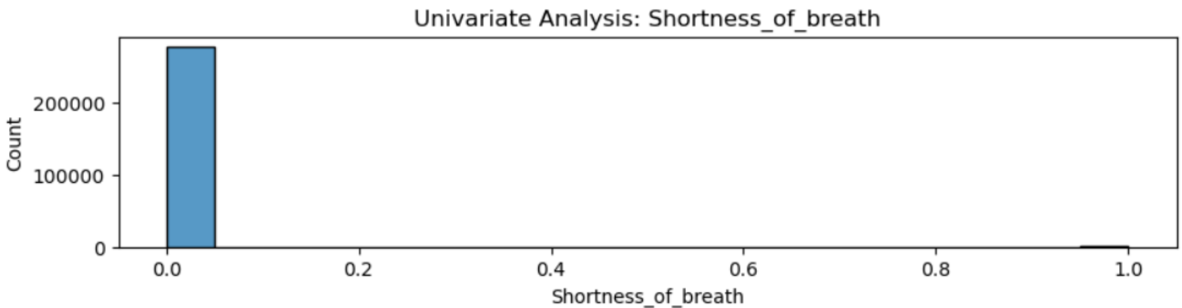

Univariate Analysis: Known_contact
Count
0
100000
200000
Abroad
Contact with confirmed
Other
Known_contact

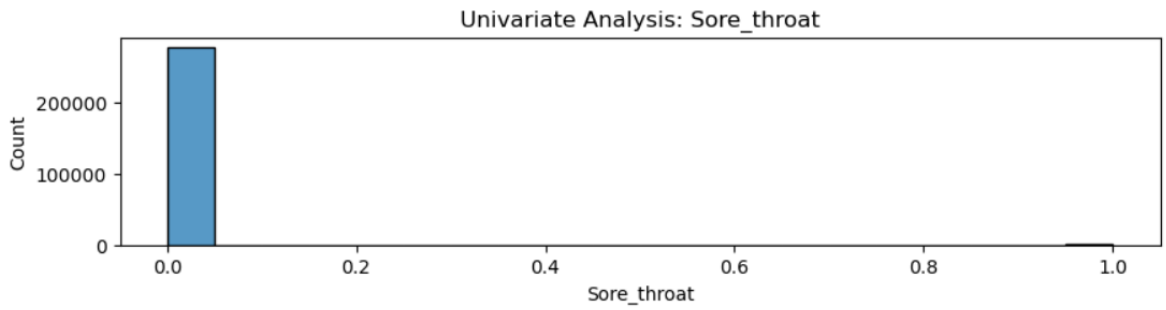

Univariate Analysis: Sex
Count
0
50000
100000
150000
female
male
Sex

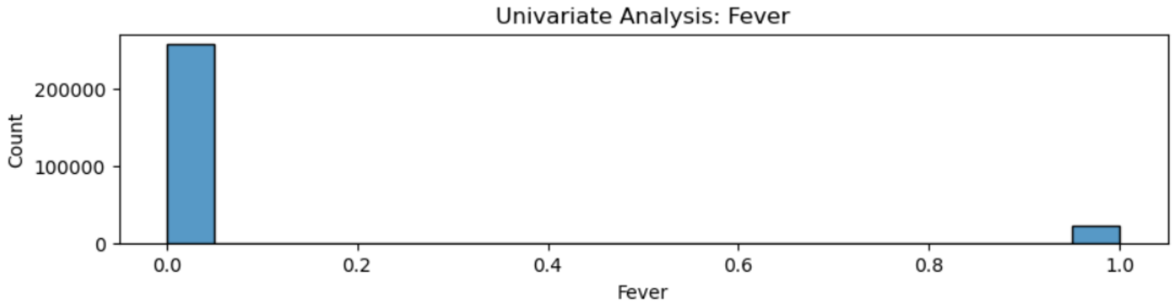

Univariate Analysis: Age_60_above
Count
0
100000
200000
No
Yes

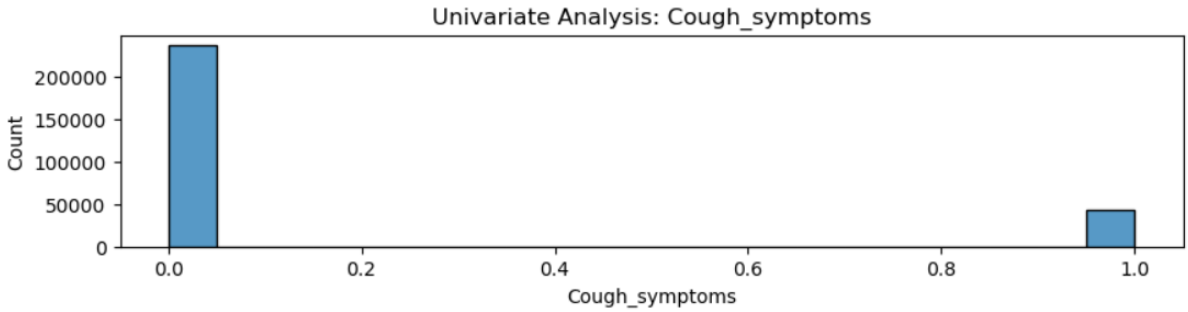

Univariate Analysis: Test_result
Count
0
100000
200000
negative
positive
Test_result

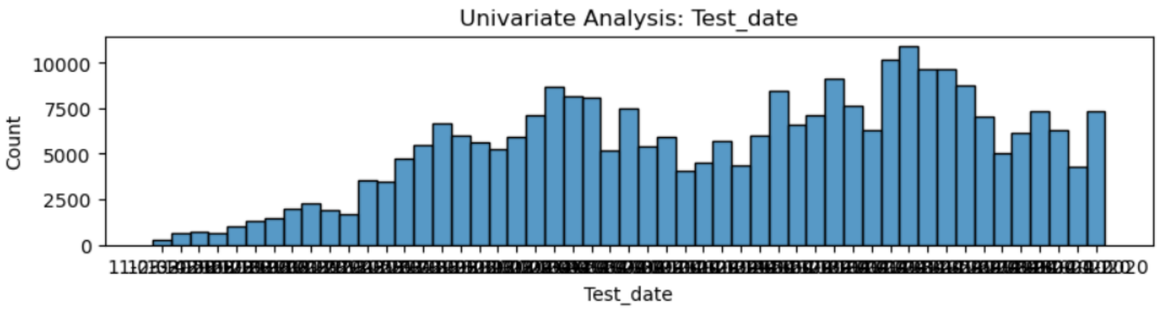

Univariate Analysis: Headache
Count
0
100000
200000
0.0
0.2
0.4
0.6
0.8
1.0
Headache

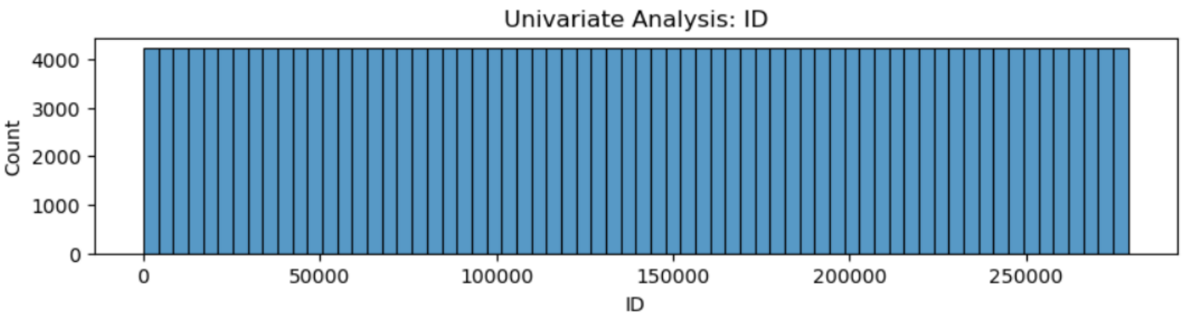

Univariate Analysis: Shortness_of_breath
Count
0
100000
200000
0.0
0.2
0.4
0.6
0.8
1.0
Shortness_of_breath


Univariate Analysis: Sore_throat
Count
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0
Sore_throat


Univariate Analysis: Fever
Count
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0
Fever


Univariate Analysis: Cough_symptoms
Count
200000
150000
100000
50000
0
0.0
0.2
0.4
0.6
0.8
1.0
Cough_symptoms


Univariate Analysis: Test_date
Count
10000
7500
5000
2500
0
Test_date


Univariate Analysis: ID
Count
4000
3000
2000
1000
0
0
50000
100000
150000
200000
250000
ID

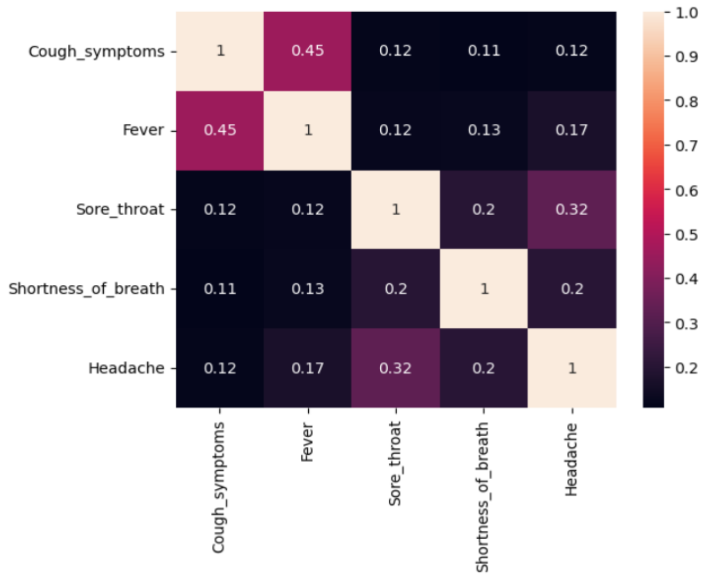
Heat Map

```
# Heat map
c = df[['Cough_symptoms','Fever','Sore_throat','Shortness_of_breath','Headache','Test_result','Age_60_above','Sex','Known_contact
c
```

	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache
Cough_symptoms	1.000000	0.454386	0.115637	0.106749	0.116350
Fever	0.454386	1.000000	0.122832	0.126070	0.168841
Sore_throat	0.115637	0.122832	1.000000	0.197540	0.323132
Shortness_of_breath	0.106749	0.126070	0.197540	1.000000	0.202538
Headache	0.116350	0.168841	0.323132	0.202538	1.000000

```
sns.heatmap(c,annot=True) #plotting heatmap
```

```
<Axes: >
```



Observations:

Coungh_symptoms Vs Fever have high correlation.

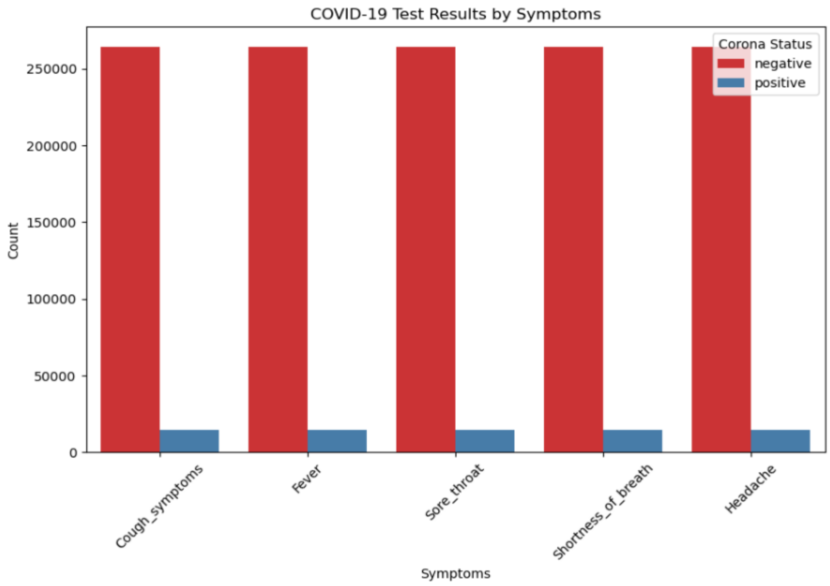
sore_throat Vs Headache have next higher correlation in the above heatmap.

values which are near to 1 value have high correlation, values which are near to zero have leass correlation.

(correlation shows the strength of relationship between two variables whereas covarience shows maginitude)

Bar chart for all symptoms Vs No.of covid-positive/negative patients

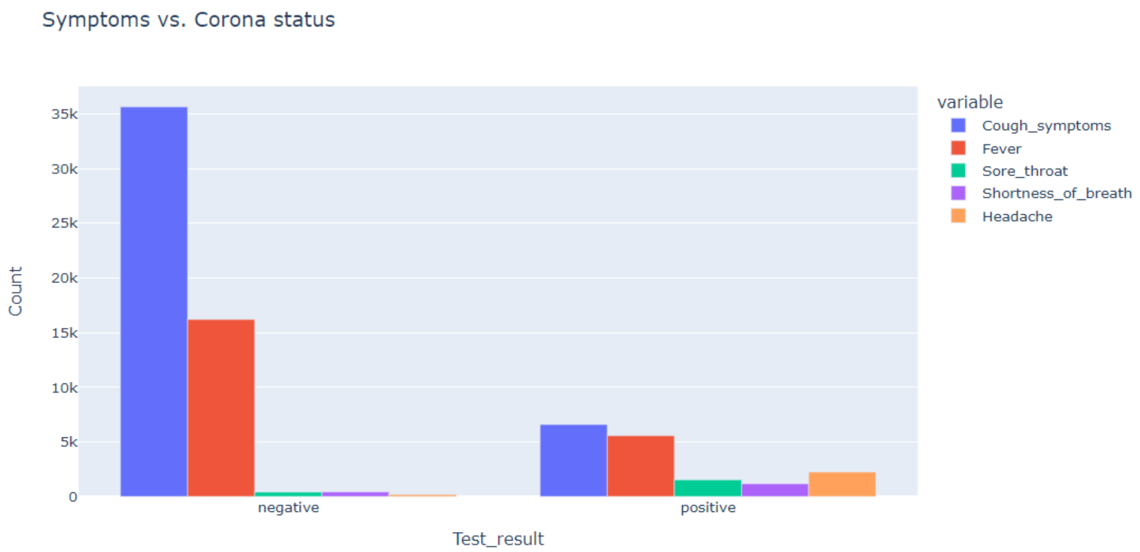
```
# Melt the DataFrame to create a long-form dataset for plotting
df_melted = pd.melt(df, id_vars=['Test_result'], value_vars=['Cough_symptoms', 'Fever', 'Sore_throat', 'Shortness_of_breath', 'He

# Create a grouped bar chart using Seaborn
plt.figure(figsize=(10, 6))
sns.countplot(data=df_melted, x='variable', hue='Test_result', palette='Set1')

# Customize the chart
plt.xlabel('Symptoms')
plt.ylabel('Count')
plt.title('COVID-19 Test Results by Symptoms')
plt.legend(title='Corona Status')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Show the plot
plt.show()
```



Observation:

we have more covid-negative patients compared to covid positive patients for all symptoms

```
# Grouping data by 'Corona positive or negative' and aggregating symptoms counts
grouped_data = df.groupby('Test_result')[['Cough_symptoms', 'Fever', 'Sore_throat', 'Shortness_of_breath', 'Headache']].sum().res

# Create the grouped bar chart
fig = px.bar(grouped_data, x='Test_result', y=['Cough_symptoms', 'Fever', 'Sore_throat', 'Shortness_of_breath', 'Headache'],
             title='Symptoms vs. Corona status',
             labels={'value': 'Count'},
             barmode='group')

# Show the chart
#sometime inline charts are not displayed then use below argument for show
fig.show(renderer='notebook')
```



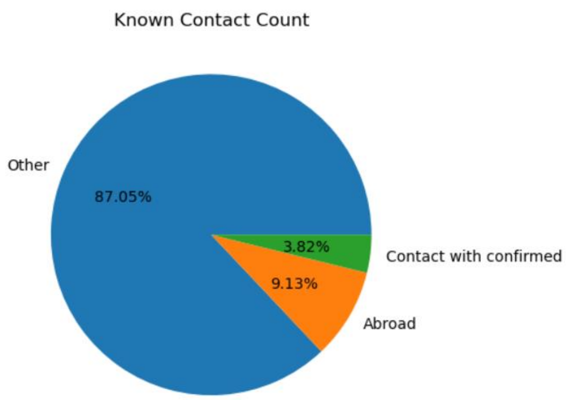
Observation:

for covid negative patients cough and fever symptoms are most common.

for covid positive patients also cough and fever symptoms are most common but less count when compared to covid negative symptoms count

Pie chart for Known contact column

```python
d = df['Known_contact'].value_counts()
v = ['Other','Abroad','Contact with confirmed']
plt.pie(d,labels=v, autopct='%1.2f%%')
plt.title("Known Contact Count")
#plt.legend()
plt.show()
```



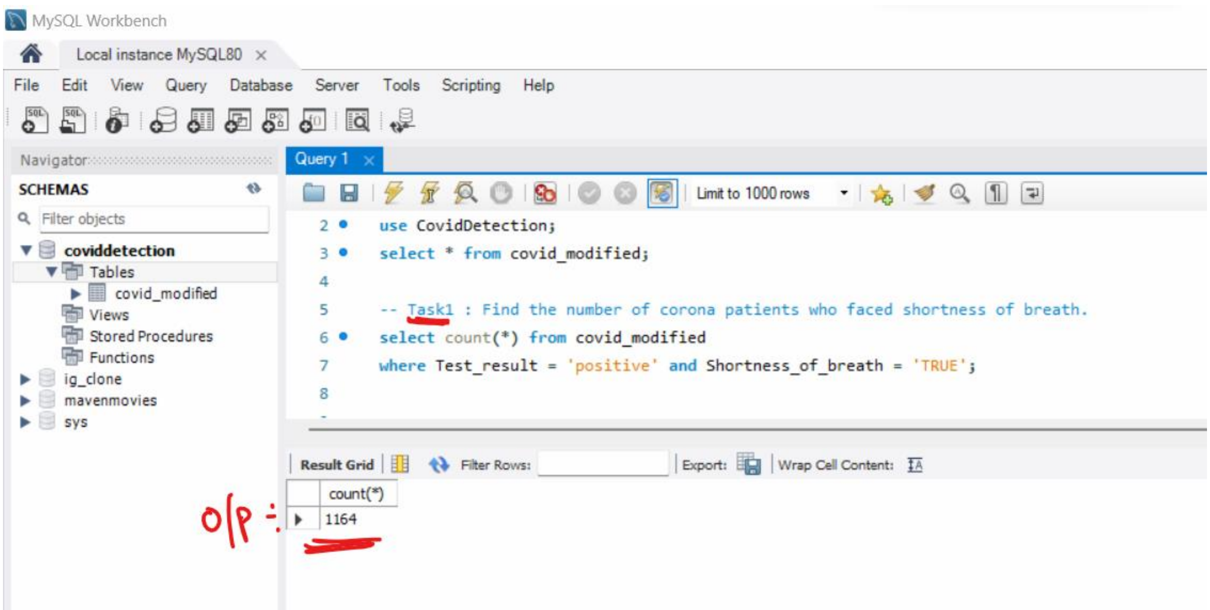
Observation:

In known contact column Other values have higher percentage than Abroad and contact with confirmed values

After Data cleaning, this cleaned data is loaded into MySQL and we try to retrieve out useful hidden insights from few queries.

SQL QUERIES for Covid19-Detection:

1) **Find the number of corona patients who faced shortness of breath.**



Observation: There are total 1164 corona positive patients who had shortness of breath symptom.

2) Find the number of negative corona patients who have fever and sore_throat.

```sql
-- Task2: Find the number of negative corona patients who have fever and sore_throat.
select count(*) from covid_modified
where Test_result = 'negative' and Fever = 'TRUE' and Sore_throat = 'TRUE';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

O/p:

count(*)
142

Observation: There are total 142 corona negative patients who had fever and sore_throat symptoms.

3) Group the data by month and rank the number of positive cases.

Observation: The dataset contains information of patients from March and April months. Among these two months, **April month have more no.of covid positive cases i.e 8881** than March month which has 5848 covid positive cases.

```sql
-- Task 3: Group the data by month and rank the number of positive cases.
select mid(Test_date, 4, 2) as Test_Month, count(ID) as Total_Positive_Cases
from covid_modified
where Test_Result = 'positive'
group by Test_Month
order by count(ID) desc;
```

output

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Test_Month	Total_Positive_Cases
04	8881
03	5848

April Month have more Positive cases than March month

4) Find the female negative corona patients who faced cough and headache.

```sql
-- Task 4: Find the female negative corona patients who faced cough and headache.
select count(*) as No_of_Female_patients from covid_modified
where Sex = 'Female' and
Test_result = 'negative' and
Cough_symptoms = 'TRUE' and
Headache = 'TRUE';
```

O/p:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

No_of_Female_patients
69

Observation: There are 69 corona negative patients who had cough and headache symptoms.

5) How many elderly corona patients have faced breathing problems?

```sql
-- Task 5: How many elderly corona patients have faced breathing problems?
select count(*) as No_of_Elderly_patients from covid_modified
where Age_60_above = 'Yes' and
Test_result = 'positive' and
Shortness_of_breath = 'TRUE';
```

O/P:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

No_of_Elderly_patients
263

Observation: There are total 263 elderly corona positive patients who had shortness_of_breath symptom.

6) Which three symptoms were more common among COVID positive patients?

```sql
-- Task 6: Which three symptoms were more common among COVID positive patients?
SELECT 'Headache' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'positive' AND Headache = 'True'
UNION ALL
SELECT 'Shortness_of_breath' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'positive' AND Shortness_of_breath = 'True'
UNION ALL
SELECT 'Sore_throat' AS Symptom, COUNT(*) AS Count
FROM covid_modified

WHERE Test_result = 'positive' AND Sore_throat = 'True'
UNION ALL
SELECT 'Fever' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'positive' AND Fever = 'True'
UNION ALL
SELECT 'Cough_symptoms' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'positive' AND Cough_symptoms = 'True'
Order BY count desc
limit 3;
```

O/P:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Symptom	Count
Cough_symptoms	6584
Fever	5559
Headache	2235

Observation: There are total 5 different symptoms in the given dataset namely: fever, Headache, shortness_of_breath, cough and Sore_throat. Covid positive patients have these three Cough, fever and Headache as most common symptoms.

7) Which symptom was less common among COVID negative people?

```sql
-- Task 7: Which symptom was less common among COVID negative people?
SELECT 'Headache' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'negative' AND Headache = 'True'
UNION ALL
SELECT 'Shortness_of_breath' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'negative' AND Shortness_of_breath = 'True'
UNION ALL
SELECT 'Sore_throat' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'negative' AND Sore_throat = 'True'
UNION ALL
SELECT 'Fever' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'negative' AND Fever = 'True'
UNION ALL
SELECT 'Cough_symptoms' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'negative' AND Cough_symptoms = 'True'
Order BY count;
```

o/p :-

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Symptom	Count
Headache	179
Sore_throat	400
Shortness_of_breath	413
Fever	16193
Cough_symptoms	35644

Observation: There are total 5 different symptoms in the given dataset namely: fever, Headache, shortness_of_breath, cough and Sore_throat. Covid negative patients have these three Headache, Sore_throat, Shortness_of_breath as least common symptoms.

8) What are the most common symptoms among COVID positive males whose known contact was abroad?

```sql
-- Task 8: What are the most common symptoms among COVID positive males whose
--  known contact was abroad?
SELECT 'Headache' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'positive' AND Headache = 'True' AND sex = 'MALE' AND known_contact = 'Abroad'
UNION ALL
SELECT 'Shortness_of_breath' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'positive' AND Shortness_of_breath = 'True' AND sex = 'MALE' AND known_contact = 'Abroad'
UNION ALL
SELECT 'Sore_throat' AS Symptom, COUNT(*) AS Count
```

```sql
FROM covid_modified
WHERE Test_result = 'positive' AND Sore_throat = 'True' AND sex = 'MALE' AND known_contact = 'Abroad'
UNION ALL
SELECT 'Fever' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'positive' AND Fever = 'True' AND sex = 'MALE' AND known_contact = 'Abroad'
UNION ALL
SELECT 'Cough_symptoms' AS Symptom, COUNT(*) AS Count
FROM covid_modified
WHERE Test_result = 'positive' AND Cough_symptoms = 'True' AND sex = 'MALE' AND known_contact = 'Abroad'
Order BY count desc;
```

O/P :-

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Symptom	Count
Cough_symptoms	532
Fever	407
Headache	129
Sore_throat	87
Shortness_of_breath	84

Observation: Cough, Fever and Headache were the most common symptoms among covid positive male patients whose known contact was ‘abroad’.

Step 2: Data Preparation / Feature engineering

ii) Data Wrangling or Data Transformation

We do this step only when we pass data to machine like ML models or else not needed.
(Converting the data from one format to another format)

- ✓ **Encoding**: converting text/categorical data into numeric data
- ✓ **Discretization**: converting continuous data into discrete/categorical data
- ✓ **Feature Transformations**: converting the skewed data (right/left skewed) into normal distributed data. **APPLIED ONLY FOR CONTINOUS VARIABLE.**
- ✓ **Split the Training and Test datasets**
- ✓ **Feature Scaling:** converting high magnitude data to low magnitude data.

In the given dataset we have all categorical datatype columns. So, we just need to do Encoding and then split the dataset into Training and Testing datasets.

Data Transformation: Feature Encoding

(converting categorical values to continous values)

```python
# Assuming 'Cough_symptoms' column contains boolean values
df['Cough_symptoms'] = df['Cough_symptoms'].astype(int)
```

```python
# for unique values we reassign True = 1, False = 0
df['Fever'] = df['Fever'].astype(int)
```

```python
# for unique values we reassign True = 1, False = 0
df['Sore_throat'] = df['Sore_throat'].astype(int)
```

```python
# for unique values we reassign True = 1, False = 0
df['Shortness_of_breath'] = df['Shortness_of_breath'].astype(int)
```

```python
# for unique values we reassign True = 1, False = 0
df['Headache'] = df['Headache'].astype(int)
```

```python
# for unique values we reassign positive = 1, False = 0
df['Test_result'] = df['Test_result'].map({'positive':1, 'negative':0})
```

```python
# for unique values we reassign yes = 1, no = 0
df['Age_60_above'] = df['Age_60_above'].map({'Yes':1, 'No':0})
```

```python
# for unique values we reassign female = 1, male = 0
df['Sex'] = df['Sex'].map({'female':1, 'male':0})
```

```python
# for unique values we reassign other = 1, Abroad = 2, contact with confrmed = 3
df['Known_contact'] = df['Known_contact'].map({'Other':1, 'Abroad':2, 'Contact with confirmed':3})
```

```python
df['Test_date'] = pd.to_datetime(df['Test_date'], format="%d-%m-%Y")
```

```python
columns = ['Cough_symptoms','Fever','Sore_throat','Shortness_of_breath','Headache','Test_result','Age_60_above','Sex','Known_cont
```

```python
#displays unique values and its count for each and every column
for i in columns:
    print(i)
    print(df[i].unique())
    print()
```

```
Cough_symptoms
[1 0]

Fever
[0 1]

Sore_throat
[1 0]

Shortness_of_breath
[0 1]

Headache
[0 1]

Test_result
[0 1]

Age_60_above
[0 1]


Sex
[1 0]

Known_contact
[2 3 1]
```

```python
df
```

	ID	Test_date	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	Test_result	Age_60_above	Sex	Known_contact
0	1	2020-03-11	1	0	1	0	0	0	0	1	2
1	2	2020-03-11	0	1	0	0	0	1	0	1	2
2	3	2020-03-11	0	1	0	0	0	1	0	1	2
3	4	2020-03-11	1	0	0	0	0	0	0	1	2
4	5	2020-03-11	1	0	0	0	0	0	0	1	3
...	...	...	...	...	...	...	...	...	...	...	...
278843	278844	2020-04-30	0	0	0	0	0	1	0	0	1
278844	278845	2020-04-30	0	0	0	0	0	0	0	1	1
278845	278846	2020-04-30	0	0	0	0	0	0	0	0	1
278846	278847	2020-04-30	0	0	0	0	0	0	0	0	1
278847	278848	2020-04-30	0	0	0	0	0	0	0	1	1

278848 rows × 11 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278848 entries, 0 to 278847
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ID                   278848 non-null  int64
 1   Test_date            278848 non-null  datetime64[ns]
 2   Cough_symptoms       278848 non-null  int32
 3   Fever                278848 non-null  int32
 4   Sore_throat          278848 non-null  int32
 5   Shortness_of_breath  278848 non-null  int32
 6   Headache             278848 non-null  int32
 7   Test_result          278848 non-null  int64
 8   Age_60_above         278848 non-null  int64
 9   Sex                  278848 non-null  int64
 10  Known_contact        278848 non-null  int64
dtypes: datetime64[ns](1), int32(5), int64(5)
memory usage: 18.1 MB
```

observation:

all columns are converted from categorical to integer datatype after feature encoding. Since there are no Continuous datatype columns Feature Tranformation and Feature Scaling is not applicable for this dataset.

splitting Training and Testing Data

```
df.head()
```

	ID	Test_date	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	Test_result	Age_60_above	Sex	Known_contact
0	1	2020-03-11	1	0	1	0	0	0	0	1	2
1	2	2020-03-11	0	1	0	0	0	1	0	1	2
2	3	2020-03-11	0	1	0	0	0	1	0	1	2
3	4	2020-03-11	1	0	0	0	0	0	0	1	2
4	5	2020-03-11	1	0	0	0	0	0	0	1	3

```
df.shape
```

```
(278848, 11)
```

```
df.loc[196491] # till 196491 row we have 19th april dates.
# so till these rows our training data must be there
```

```
ID                                  196492
Test_date              2020-04-19 00:00:00
Cough_symptoms                           1
Fever                                    1
Sore_throat                              0
Shortness_of_breath                      1
Headache                                 0
Test_result                              0
Age_60_above                             0
Sex                                      1
Known_contact                            1
Name: 196491, dtype: object
```

```
df.loc[196492]
# from 196492 rows to last rows is our test dataset.
```

```
ID                                  196493
Test_date              2020-04-20 00:00:00
Cough_symptoms                           0
Fever                                    0
Sore_throat                              0
Shortness_of_breath                      0
Headache                                 0
Test_result                              0
Age_60_above                             0
Sex                                      0
Known_contact                            1
Name: 196492, dtype: object
```

As per our business requirement we have to take data from 11th March 2020 to 19th April 2020 as Training Set & Validation.And , data from 20th April to 30th april as Test Set.

```
break_date = pd.Timestamp("2020-04-19")
df_ninteenthApr = df[df["Test_date"] <= break_date]
df_twentyApr = df[df["Test_date"] > break_date]
```

```
df_ninteenthApr
```

	ID	Test_date	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	Test_result	Age_60_above	Sex	Known_contact
0	1	2020-03-11	1	0	1	0	0	0	0	1	2
1	2	2020-03-11	0	1	0	0	0	1	0	1	2
2	3	2020-03-11	0	1	0	0	0	1	0	1	2
3	4	2020-03-11	1	0	0	0	0	0	0	1	2
4	5	2020-03-11	1	0	0	0	0	0	0	1	3
...	...	...	...	...	...	...	...	...	...	...	...
196487	196488	2020-04-19	1	1	0	0	0	0	0	0	1
196488	196489	2020-04-19	1	1	0	0	0	0	0	0	2
196489	196490	2020-04-19	0	0	0	0	0	0	0	0	1
196490	196491	2020-04-19	0	0	0	0	0	0	0	0	1
196491	196492	2020-04-19	1	1	0	1	0	0	0	1	1

196492 rows × 11 columns

```
df_twentyApr
```

	ID	Test_date	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	Test_result	Age_60_above	Sex	Known_contact
196492	196493	2020-04-20	0	0	0	0	0	0	0	0	1
196493	196494	2020-04-20	1	0	0	0	0	0	0	1	1
196494	196495	2020-04-20	0	0	0	0	0	0	0	1	1
196495	196496	2020-04-20	1	1	0	0	0	0	0	0	2
196496	196497	2020-04-20	0	0	0	0	0	0	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...
278843	278844	2020-04-30	0	0	0	0	0	1	0	0	1
278844	278845	2020-04-30	0	0	0	0	0	0	0	1	1
278845	278846	2020-04-30	0	0	0	0	0	0	0	0	1
278846	278847	2020-04-30	0	0	0	0	0	0	0	0	1
278847	278848	2020-04-30	0	0	0	0	0	0	0	1	1

82356 rows × 11 columns

```
#dropping column Test_date and Outcome variable(Corona) column , and storing it to X_train
X_train = df_ninteenthApr.drop(columns = ['ID','Test_date','Test_result'],axis = 1)
X_train
```

	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	Age_60_above	Sex	Known_contact
0	1	0	1	0	0	0	1	2
1	0	1	0	0	0	0	1	2
2	0	1	0	0	0	0	1	2
3	1	0	0	0	0	0	1	2
4	1	0	0	0	0	0	1	3
...	...	...	...	...	...	...	...	...
196487	1	1	0	0	0	0	0	1
196488	1	1	0	0	0	0	0	2
196489	0	0	0	0	0	0	0	1
196490	0	0	0	0	0	0	0	1
196491	1	1	0	1	0	0	1	1

```
#storing outcome variable in y_train.
y_train = df_ninteenthApr['Test_result']
y_train
```

```
0         0
1         1
2         1
3         0
4         0
         ..
196487    0
196488    0
196489    0
196490    0
196491    0
Name: Test_result, Length: 196492, dtype: int64
```

```
#dropping Test_date column and Outcome variable(Corona) column from df_test and storing it to X_test.
X_test = df_twentyApr.drop(columns = ['ID','Test_date','Test_result'],axis=1)
X_test
```

	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	Age_60_above	Sex	Known_contact
196492	0	0	0	0	0	0	0	1
196493	1	0	0	0	0	0	1	1
196494	0	0	0	0	0	0	1	1
196495	1	1	0	0	0	0	0	2
196496	0	0	0	0	0	0	1	1
...	...	...	...	...	...	...	...	...
278843	0	0	0	0	0	0	0	1
278844	0	0	0	0	0	0	1	1
278845	0	0	0	0	0	0	0	1
278846	0	0	0	0	0	0	0	1
278847	0	0	0	0	0	0	1	1

82356 rows × 8 columns

```
#storing Outcome variable test Set data into y_test.
y_test = df_twentyApr['Test_result']
y_test
```

```
196492    0
196493    0
196494    0
196495    0
196496    0
         ..
278843    1
278844    0
278845    0
278846    0
278847    0
Name: Test_result, Length: 82356, dtype: int64
```

In the problem statement if we split the training data from march to 15th april and Test-data from 16th april to 30th april it is not dividing the test and train data properly. Training data is 58.53% Test data is 41.47% we are getting.

so we took Training data from 11th March to 19th Apr = 70.4% , Test set data from 20th apr to 30 apr = 29.53%

Step 3: Feature Selection:

```
from scipy.stats import chi2_contingency
```

```
columns = ['ID','Test_date','Cough_symptoms','Fever','Sore_throat','Shortness_of_breath','Headache','Age_60_above','Sex','Known_c
for i in columns:
    #creating a cotingency table
    contingency_table = pd.crosstab(df[i],df["Test_result"])
    # perform chi-square test, calculating p-value
    chi2, p_value, dof, expected = chi2_contingency(contingency_table)
    #prints the results
    print("-----{}-----".format(i))
    print("chi-square statistic: ",chi2)
    print("p_value = ",p_value)
```

```
-----ID-----
chi-square statistic:  278847.99999999994
p_value =  0.4991096512866681
-----Test_date-----
chi-square statistic:  4498.032907406993
p_value =  0.0
-----Cough_symptoms-----
chi-square statistic:  10569.415074648161
p_value =  0.0
-----Fever-----
chi-square statistic:  19378.570935486066
p_value =  0.0
-----Sore_throat-----
chi-square statistic:  21183.30774235602
p_value =  0.0
-----Shortness_of_breath-----
chi-square statistic:  14873.153774171122
p_value =  0.0
-----Headache-----
chi-square statistic:  37078.834270861014
p_value =  0.0
```

```
-----Age_60_above-----
chi-square statistic:  600.9907438227524
p_value =  1.0193061909600926e-132
-----Sex-----
chi-square statistic:  140.4145884069575
p_value =  2.1604974877258956e-32
-----Known_contact-----
chi-square statistic:  90331.28046978849
p value =  0.0
```

observation:

we did chi-square test as we have all categorical columns. here we got p_vlues for columns less than 0.05 except 'ID' column(p_value = 0.49). And all independent columns except 'ID' column have relationship with dependent column i.e 'test_result' . so we need to drop 'ID' column and all other remaining independent columns remain in dataset.

Step 4: Machine learning Models

Problem Statement: We need to predict covid positive or negative result based on different independent columns.

The output result column i.e 'Test_result' is a binary classification task. So we can use Supervised ML algorithms for this dataset. We will use the following 4 ML algorithms namely:

1. Logistic Regression
2. Decision Tree
3. Random Forest
4. K-Nearest Neighbour(KNN) models

1. Decision Tree

```
# importing libraries
from sklearn.tree import DecisionTreeClassifier
```

```
# creating a Decision Tree classifier
dtree = DecisionTreeClassifier()
```

```
# Fit the model to the training data: x_train and y_train
dtree.fit(X_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Predicting the model

```
#Making predictions on test dataset
predictions = dtree.predict(X_test)
predictions
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Evaluation of Decision Tree

```
#importing libraries
from sklearn.metrics import classification_report, confusion_matrix
```

```
#calculating precision, recall, F1-score and support
print(classification_report(y_test, predictions))
# precision = (total TP /(TP+FP))
# recall = (total TP /(TP+FN))
# F1_score =
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     80614
           1       0.79      0.43      0.56      1742

    accuracy                           0.99     82356
   macro avg       0.89      0.72      0.78     82356
weighted avg       0.98      0.99      0.98     82356
```

```
# calculating confusion matrix
print("-----Confusion Matrix-------")
print(confusion_matrix(y_test, predictions))
""" we get confusion matrix as below
Truenegative, FalseNegative, FalsePositive, truePositive:
[TN FN
FP TP] """
```

```
-----Confusion Matrix-------
[[80410   204]
 [  986   756]]
```

```
#calculating accuracy
accuracy = dtree.score(X_test, y_test)
accuracy # accuracy = (TN+TP)/total_samples
```

```
0.9855505366943513
```

```
#cross checking accuracy manually
80410+204+986+756 # total no.of samples
```

```
82356
```

```
#Calculating accuracy : (TN+TP)/total_samples
(80410+756)/82356
```

```
0.9855505366943513
```

Observation:

Decision-Tree model got 98.55% accuracy which means this model prediction is good.

For all covid_negative cases precision is very good it is 0.99 for covid_positive cases precision is 0.79. In this model precision is better for covid_negative cases than covid_positive cases.

2. Random Forest

```
from sklearn.ensemble import RandomForestClassifier #importing libraries
# creating a rondom forest classifier
rfc  = RandomForestClassifier(n_estimators = 100)
# n_estimators = 100 means our forest consists 100 trees.
# fit the model to Train dataset: X_train and y_train
rfc.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
# making predictions on test data
rfc_pred = rfc.predict(X_test)
```

```
#calculating precision, recall, F1-score and support
print(classification_report(y_test, rfc_pred))
# precision = (total  TP /(TP+FP))
# recall = (total  TP /(TP+FN))
# F1_score =
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     80614
           1       0.79      0.43      0.56      1742

    accuracy                           0.99     82356
   macro avg       0.89      0.72      0.78     82356
weighted avg       0.98      0.99      0.98     82356
```

```
#calculating confusion matrix
print("-----Confusion Matrix-------")
print(confusion_matrix(y_test, rfc_pred))
""" we get confusion matrix as below
Truenegative, FalseNegative, FalsePositive, truePositive:
[TN FN
FP TP] """
```

```
-----Confusion Matrix-------
[[80408   206]
 [  986   756]]
```

```
#calculating accuracy
accuracy = rfc.score(X_test, y_test)
accuracy # accuracy = (TN+TP)/total_samples
```

```
0.985526251882073
```

```
#cross checking accuracy manually
80409+205+986+756 #total no.of samples
```

```
82356
```

```
#Calculating accuracy : (TN+TP)/total_samples
(80409+756)/82356
```

```
0.9855383942882121
```

```
(80409+756)
```

```
81165
```

Observation:

Random Forest model got 98.55% accuracy which means this model prediction is good.

For all covid_negative cases precision is very good it is 0.99 for covid_positive cases precision is 0.79. In this model precision is better for covid_negative cases than covid_positive cases.

Logistic Regression

creating Logistic Regression model

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```

```
# Fit the model to the training data: X_train and y_train
lr.fit(X_train, y_train)

# Here we pass both X_train, y_train because model has to learn from
#existing values. Thats why we pass label as well (y_train).
```

```
▾ LogisticRegression
LogisticRegression()
```

Make predictions on the test data

```
y_pred = lr.predict(X_test)
y_pred
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Evaluation of Logistic Regression Model

```
#calculating confusion matrix
print("-----Confusion Matrix-------")
print(confusion_matrix(y_test, y_pred))
""" we get confusion matrix as below
Truenegative, FalseNegative, FalsePositive, truePositive:
[TN FN
FP TP] """
```

```
-----Confusion Matrix-------
[[80521    93]
 [ 1570   172]]
```

```
from sklearn.metrics import classification_report  #importing libraries
#calculating precision, recall, F1-score and support
print(classification_report(y_test, y_pred))
# precision = (total  TP /(TP+FP))
# recall  = (total  TP /(TP+FN))
# F1_score =
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99     80614
           1       0.65      0.10      0.17      1742

    accuracy                           0.98     82356
   macro avg       0.81      0.55      0.58     82356
weighted avg       0.97      0.98      0.97     82356
```

```
#calculating accuracy
accuracy = lr.score(X_test, y_test)
accuracy # accuracy = (TN+TP)/total_samples
```

```
0.9798071785905095
```

Observation:

Logistic Regression model got 97.98% accuracy which means this model prediction is also good but less than previous models.

For all covid_negative cases precision is very good it is 0.98 for covid_positive cases precision is 0.65. In this model precision and other metrics are better for covid_negative cases than covid_positive cases.

K Nearest Neighbors Algorithm (KNN)

```
# importing libraries
from sklearn.neighbors import KNeighborsClassifier
```

Creating KNN classifier

```
k = 5  # You can give any number of neighbors (k) as needed
classifier = KNeighborsClassifier(n_neighbors=k)
```

```
# Fit the model to the training data: X_train and y_train
classifier.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Make predictions on the test data

```
y_pred = classifier.predict(X_test)
```

```
y_pred
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Evalation of KNN model

```
#calculating confusion matrix
print("-----Confusion Matrix-------")
print(confusion_matrix(y_test, y_pred))
""" we get confusion matrix as below
Truenegative, FalseNegative, FalsePositive, truePositive:
[TN FN
FP TP] """
```

```
-----Confusion Matrix-------
[[80393   221]
 [  977   765]]
```

```
from sklearn.metrics import classification_report  #importing libraries
#calculating precision, recall, F1-score and support
print(classification_report(y_test, y_pred))
# precision = (total TP /(TP+FP))
# recall = (total TP /(TP+FN))
# F1_score =
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     80614
           1       0.78      0.44      0.56      1742

    accuracy                           0.99     82356
   macro avg       0.88      0.72      0.78     82356
weighted avg       0.98      0.99      0.98     82356
```

```
#calculating accuracy
accuracy = classifier.score(X_test, y_test)
accuracy # accuracy = (TN+TP)/total_samples
```

```
0.9854533974452377
```

Observation:

KNN model got 98.54% accuracy which means this model prediction is good.

For all covid_negative cases precision is very good it is 0.97 for covid_positive cases precision is 0.69. In this model precision and other metrics are better for covid_negative cases than covid_positive cases.

Step 5: Evaluation and Comparision of 4 ML Models

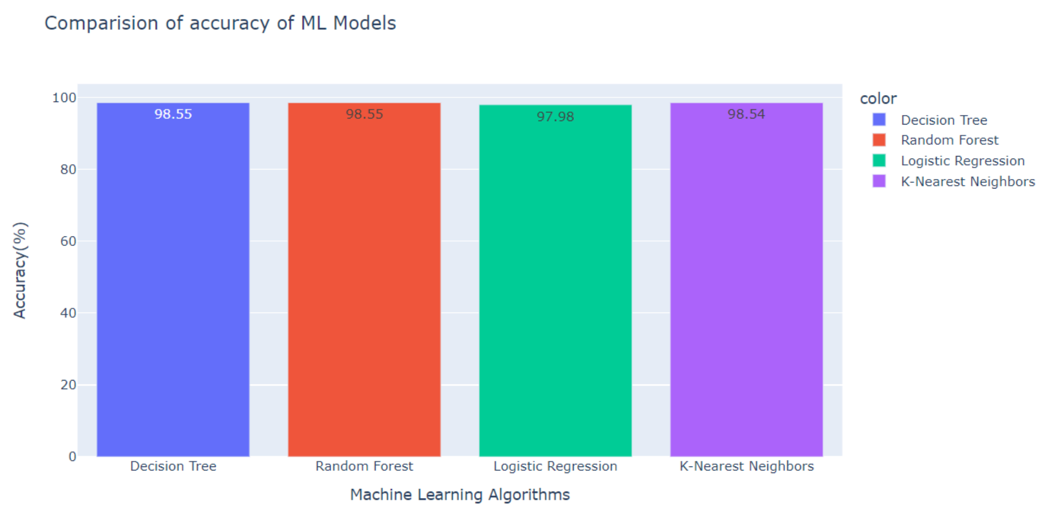
```
#accuracy,precision,recall,f1-score,support
dtree_negativecase = ["dtree_negativecase",0.9855505366943513,0.99,1.00,0.99,80614]
dtree_positivecase = ["dtree_Positivecase",0.9855505366943513,0.79,0.43,0.56,1742]

rfc_negativecase = ["RandForest_negative",0.9855383942882121,0.99,1.00,0.99,80614]
rfc_positivecase = ["RandForest_positive",0.9855383942882121,0.79,0.43,0.56,1742]

lr_negativecase = ["LogisticReg_negative",0.9798071785905095,0.98,1.00,0.99,80614]
lr_positivecase = ["LogisticReg_Positive",0.9798071785905095,0.65,0.10,0.17,1742]

knn_negativecase = ["knn_negativecase",0.9854533974452377,0.99,1.00,0.99,80614]
knn_positivecase = ["knn_positivecase",0.9854533974452377,0.78,0.44,0.56,1742]

l = [dtree_negativecase,dtree_positivecase,rfc_negativecase,
     rfc_positivecase,lr_negativecase,lr_positivecase,knn_negativecase,knn_positivecase]
names = ["dtree_negativecase","dtree_positivecase","rfc_negativecase",
     "rfc_positivecase","lr_negativecase","lr_positivecase","knn_negativecase","knn_positivecase"]
colnames = ["Model Name","Accuracy","Precision","Recall","F1-Score","Support"]
for n in colnames:
    print(n,end="              ")
print(" ")

for i in range(8):
    #x.field_names = ["column-Name","Accuracy","Precision","Recall","F1-Score","Support"]
    for j in range(6):
        print(l[i][j],end="           ")
    print("")
```

```
Model Name              Accuracy              Precision              Recall              F1-Score              Support
dtree_negativecase           0.9855505366943513           0.99           1.0           0.99           80614
dtree_Positivecase           0.9855505366943513           0.79           0.43           0.56           1742
RandForest_negative           0.9855383942882121           0.99           1.0           0.99           80614
RandForest_positive           0.9855383942882121           0.79           0.43           0.56           1742
LogisticReg_negative           0.9798071785905095           0.98           1.0           0.99           80614
LogisticReg_Positive           0.9798071785905095           0.65           0.1           0.17           1742
knn_negativecase           0.9854533974452377           0.99           1.0           0.99           80614
knn_positivecase           0.9854533974452377           0.78           0.44           0.56           1742
```

Accuracy Comparision of 4 ML Models

```
import plotly
import pandas as pd
import plotly.express as px
```

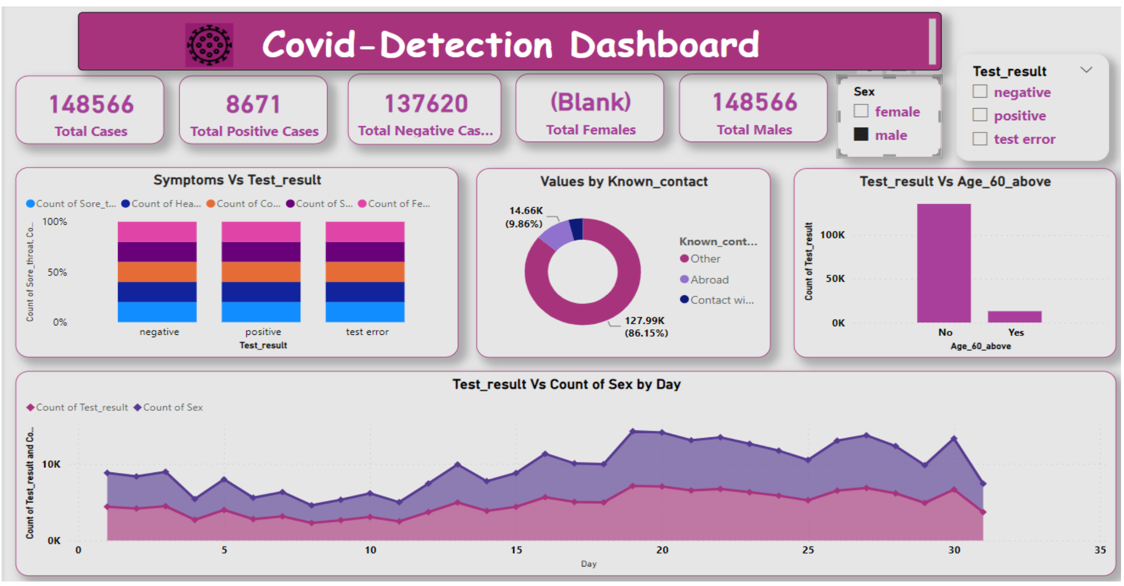
```python
# Visualisation of our ML algorithms

fig = px.bar(x = ['Decision Tree', 'Random Forest', 'Logistic Regression', 'K-Nearest Neighbors'],
             y = [98.55, 98.55, 97.98, 98.54],
             text = [98.55, 98.55, 97.98, 98.54],
             labels = {'x': 'Machine Learning Algorithms', 'y':'Accuracy(%)'},
             title = "Comparision of accuracy of ML Models",
             color = ['Decision Tree', 'Random Forest', 'Logistic Regression', 'K-Nearest Neighbors'],
            #color_discrete_sequence=['red', 'blue', 'green', 'yellow']
            )
fig.show()
```

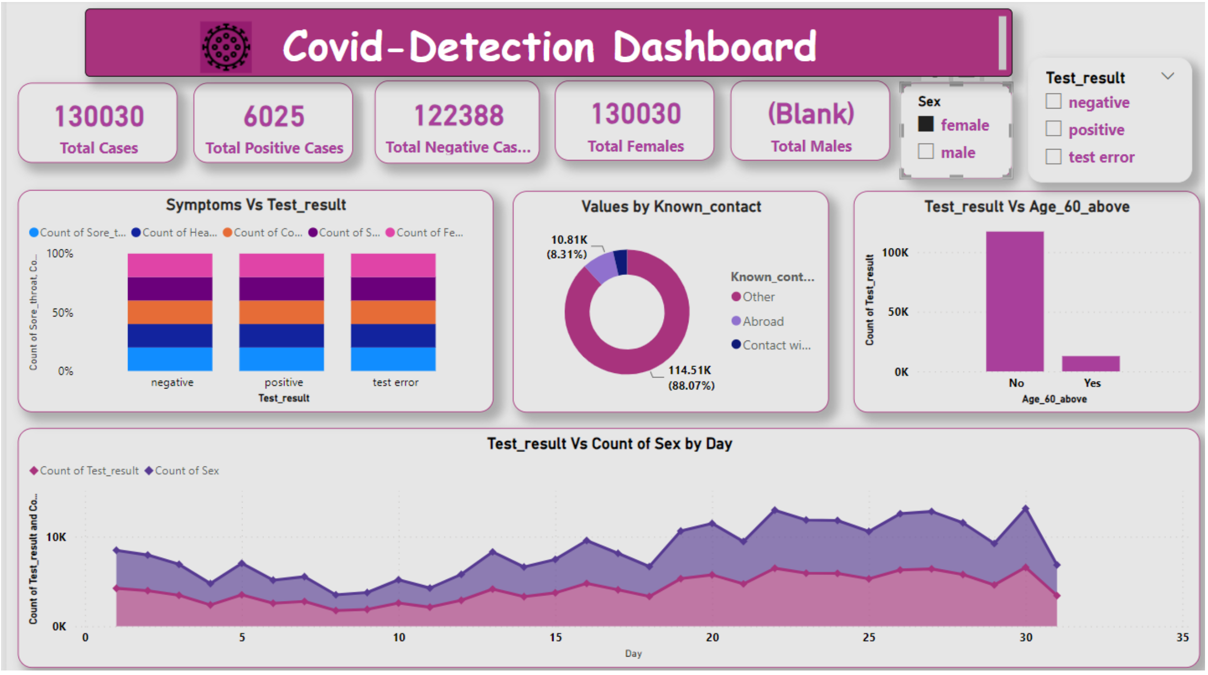


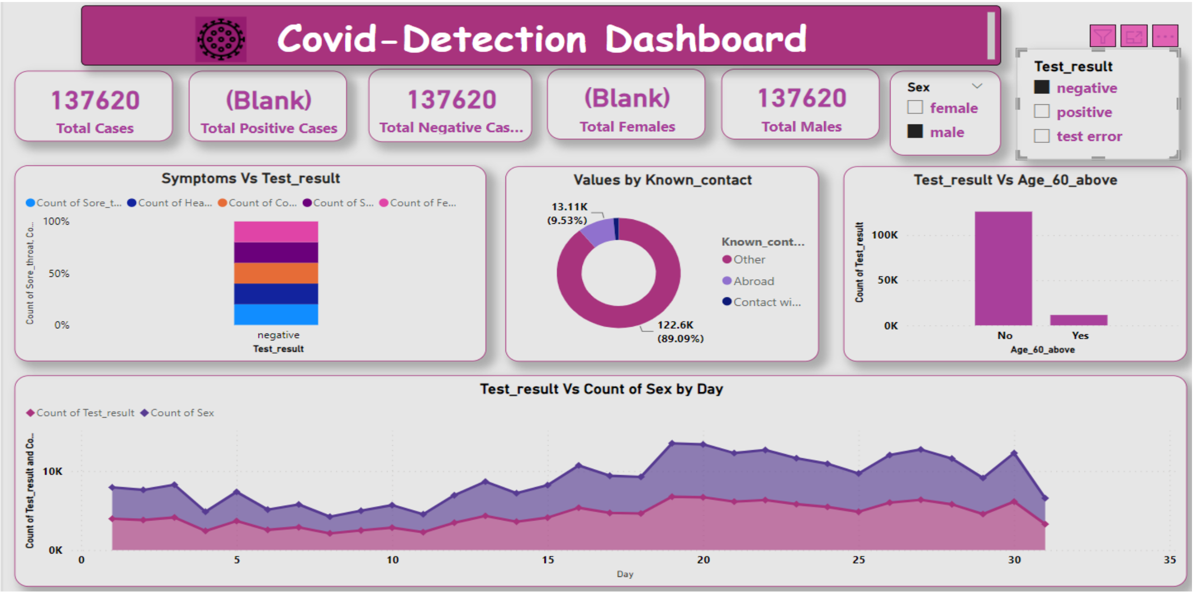
Observation: From above results we can observe that accuracy of these four Algorithms are very close to each other. If we compare all models then we can conclude that 'Decision Tree' or 'Random Forest' Algorithm are best for our Covid-19 dataset. Next best algorithm is KNN and least accurate model is Logistic Regression model for this covid dataset.

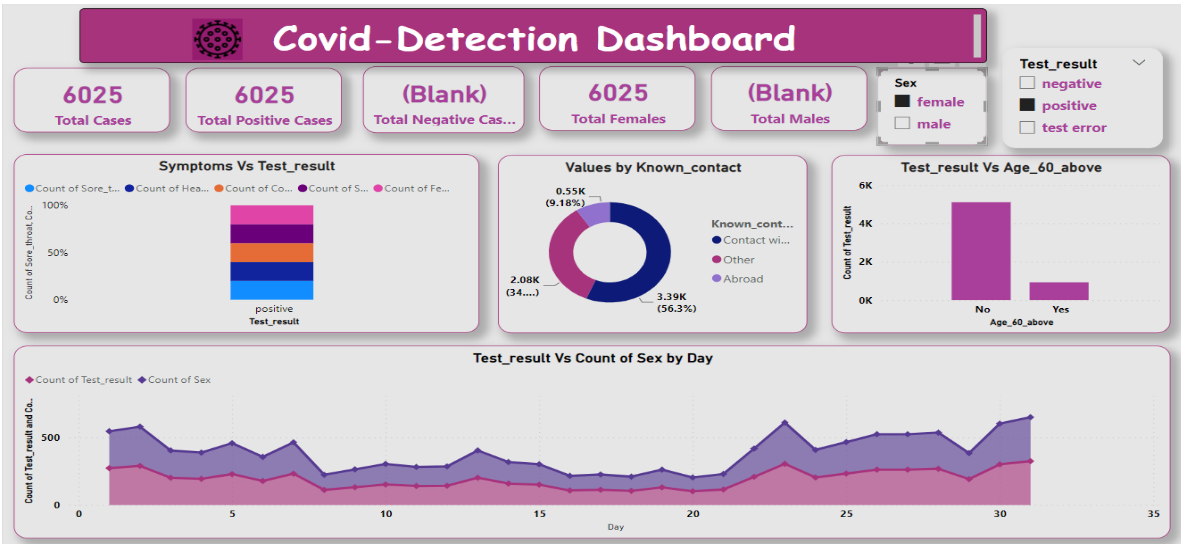
Step 6: Presentation

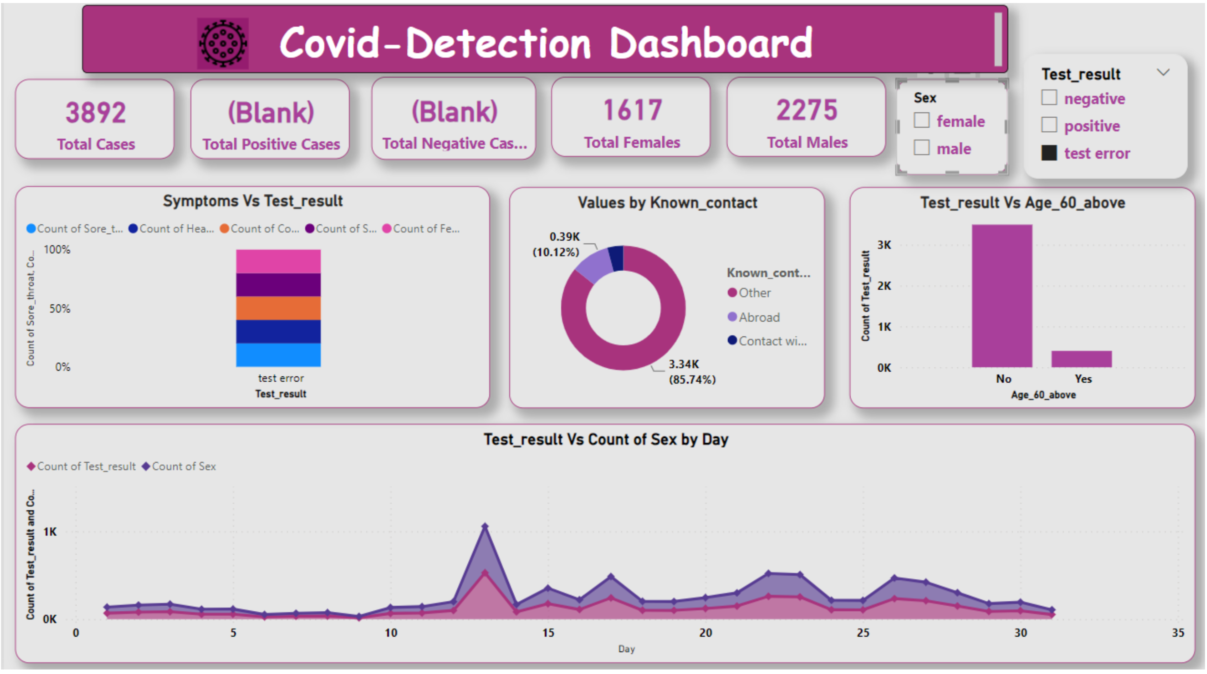
To create interactive dashboards and presenting these insights to Business customer we use PowerBI tool.




Covid-Detection Dashboard
130030
Total Cases
6025
Total Positive Cases
122388
Total Negative Cas...
130030
Total Females
(Blank)
Total Males
Sex
female
male
Test_result
negative
positive
test error
Symptoms Vs Test_result
Count of Sore_t... Count of Hea... Count of Co... Count of S... Count of Fe...
100%
50%
0%
negative
positive
test error
Test_result
Values by Known_contact
10.81K (8.31%)
114.51K (88.07%)
Known_cont...
Other
Abroad
Contact wi...
Test_result Vs Age_60_above
100K
50K
0K
No
Yes
Age_60_above
Count of Test_result
Test_result Vs Count of Sex by Day
Count of Test_result Count of Sex
10K
0K
0
5
10
15
20
25
30
35
Day


Covid-Detection Dashboard
137620
Total Cases
(Blank)
Total Positive Cases
137620
Total Negative Cas...
(Blank)
Total Females
137620
Total Males
Sex
female
male
Test_result
negative
positive
test error
Symptoms Vs Test_result
Count of Sore_t... Count of Hea... Count of Co... Count of S... Count of Fe...
100%
50%
0%
negative
Test_result
Values by Known_contact
13.11K (9.53%)
122.6K (89.09%)
Known_cont...
Other
Abroad
Contact wi...
Test_result Vs Age_60_above
100K
50K
0K
No
Yes
Age_60_above
Count of Test_result
Test_result Vs Count of Sex by Day
Count of Test_result Count of Sex
10K
0K
0
5
10
15
20
25
30
35
Day


Covid-Detection Dashboard
6025
Total Cases
6025
Total Positive Cases
(Blank)
Total Negative Cas...
6025
Total Females
(Blank)
Total Males
Sex
female
male
Test_result
negative
positive
test error
Symptoms Vs Test_result
Count of Sore_t... Count of Hea... Count of Co... Count of S... Count of Fe...
100%
50%
0%
positive
Test_result
Values by Known_contact
0.55K (9.18%)
2.08K (34....)
3.39K (56.3%)
Known_cont...
Contact wi...
Other
Abroad
Test_result Vs Age_60_above
6K
4K
2K
0K
No
Yes
Age_60_above
Count of Test_result
Test_result Vs Count of Sex by Day
Count of Test_result Count of Sex
500
0
0
5
10
15
20
25
30
35
Day



Conclusion:

We can see graphs using slicers like female, male and Positive and negative test cases and test error cases. The following are some of the insights from the above dashboards:

- There are more no. of covid negative cases than covid positive cases in all slicers.
- In Covid positive cases majority people (56.3%) got infected with covid-19, when they are in contact with another covid-19 positive patient.
- There are more male covid positive and negative patients than female patients.
- More no. of people less than 60 years of age are been tested for covid-19 detection.
- April month have more no. of covid positive cases i.e., 8881 than March month which has 5848 covid positive cases.
- Covid positive patients have these three Cough, fever and Headache as most common symptoms.
- Covid negative patients have these three Headache, Sore_throat, Shortness_of_breath as least common symptoms.