covid-project

October 6, 2023

1 COVID19 prediction

1.0.1 Understandig Business Problem:-

A speedy and accurate diagnosis of COVID-19 is made possible by effective SARS-CoV-2 screening, which can also lessen the
burden on healthcare systems. There have been built prediction models that assess the likelihood of infection by combining a number of parameters. These are meant to help medical professionals all over the world treat patients, especially in light of the scarcity of healthcare resources. The current dataset has been downloaded from 'ABC' government website and contains around 2,78,848 individuals who have gone through the RT-PCR test.Dataset contains 11 columns,including 8 features suspected to play an important role in the prediction of COVID19 outcome. Outcome variable is covid result test positive or negative. We have data from 11th March 2020 till 30th April 2020. Please consider 11th March till 15th April as a training and validation set.From 16th April till 30th April as a test set. Please further divide training and validation set at a ratio of 4:1.

Q1.Why is your proposal important in today's world? How predicting a disease accurately can improve medical treatment? Predicting disesase accurately using Machine learning. Particularly for COVID-19 hold significant importance in today's world as early detection of such disease on the basis of symptoms can help the healthcare facility to isolate and treat the patient timely this is crucial to prevent the spread of disease

Q2.How is it going to impact the medical field when it comes to effective screening and reducing health care burden. Our medical facilities are using test such as RAT (Rapid Antigen Test) to detect whether a person is COVID affected,while our ML model use patient data to detect whether a particular patient is COVID affected or not with almost accuracy this help to reduce the burden of testing and screening on the healthcare facilities.

Q3.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. Bad quality of data or missing data can prove to be a problem for a such model but if it works well for COVID-19,We might use a similar approach for other diseases in the future.It is like having a valuable tool for our medical facilities.

1.1 Initial Hypothesis

- Prediction hypothesis: Prediction hypothesis with think we can predict COVID-19 by looking at symptoms and some personal info.

- Benefit of accurate prediction: Accurate prediction can speed up treatment, resource use and disease management, reducing the strain on healthcare.
- Common symptoms of hypothesis: COVID-19 often causes coughing, fever and breathing problem.
- Uncommon symptoms Hypothesis: People without COVID-19 are less likely to have a store sore throat or headache.
- symptoms Specific group: Men with COVID-19, who have been in contact with people from other countries might have more coughing and fever.
- Age and severity Hypothesis: Those 60 and older age, might have more severe symptoms, especially trouble getting breathing.

1.2 Description of the Dataset

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

2 Importing Libraries

[111]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

[112]:
```
# To ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

[114]:
```
data = pd.read_csv("corona_tested_006.csv") # reading csv file
```

3 Exploratory Data Analysis (EDA)

[115]:
```
df = data.copy() # Copying data to new variable
```

[116]:
```
df.head()# To show first 5 rows.
```

[116]:
```
    Ind_ID   Test_date Cough_symptoms  Fever Sore_throat Shortness_of_breath  \
0        1  11-03-2020           TRUE  FALSE        TRUE                FALSE
1        2  11-03-2020          FALSE   TRUE       FALSE                FALSE
2        3  11-03-2020          FALSE   TRUE       FALSE                FALSE
3        4  11-03-2020           TRUE  FALSE       FALSE                FALSE
4        5  11-03-2020           TRUE  FALSE       FALSE                FALSE

  Headache    Corona Age_60_above   Sex           Known_contact
0    FALSE  negative         None  None                  Abroad
1    FALSE  positive         None  None                  Abroad
2    FALSE  positive         None  None                  Abroad
3    FALSE  negative         None  None                  Abroad
4    FALSE  negative         None  None  Contact with confirmed
```

[117]:
```
df.shape # To check row and columns
```

[117]:
```
(278848, 11)
```

[118]:
```
df.tail() # To check last 5 rows
```

[118]:
```
        Ind_ID   Test_date Cough_symptoms  Fever Sore_throat  \
278843  278844  30-04-2020          False  False       False
278844  278845  30-04-2020          False  False       False
278845  278846  30-04-2020          False  False       False
278846  278847  30-04-2020          False  False       False
278847  278848  30-04-2020          False  False       False

       Shortness_of_breath Headache    Corona Age_60_above     Sex  \
278843               False    False  positive         None    male
278844               False    False  negative         None  female
278845               False    False  negative         None    male
278846               False    False  negative         None    male
278847               False    False  negative         None  female
```

```
       Known_contact
278843         Other
278844         Other
278845         Other
278846         Other
278847         Other
```

[119]:
```
df.columns # To check all columns names.
```

[119]:
```
Index(['Ind_ID', 'Test_date', 'Cough_symptoms', 'Fever', 'Sore_throat',
       'Shortness_of_breath', 'Headache', 'Corona', 'Age_60_above', 'Sex',
       'Known_contact'],
      dtype='object')
```

[120]:
```
df.info() # To check the dataframe along with datatype
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278848 entries, 0 to 278847
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   Ind_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   Corona               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
```

3.1 Data Cleaning

3.1.1 Observation :-

Till here we have renamed the 2 columns "Ind_ID as ID" and "Corona as Covid result"

We don't have any null value in any column.

3.1.2 Renaming the columns

[121]:
```
df= df.rename(columns ={"Ind_ID":"ID","Corona":"Covid_result"}) # Renaming 2
↪columns,which are inappropriate
```

[122]:
```
df.columns # To checking , whether the named has changed
```

[122]:
```
Index(['ID', 'Test_date', 'Cough_symptoms', 'Fever', 'Sore_throat',
       'Shortness_of_breath', 'Headache', 'Covid_result', 'Age_60_above',
       'Sex', 'Known_contact'],
      dtype='object')
```

[123]:
```
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  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   Covid_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
```

[124]:
```
df.isnull().sum() #To check any null value present or not
```

[124]:
```
ID                    0
Test_date             0
Cough_symptoms        0
Fever                 0
Sore_throat           0
Shortness_of_breath   0
Headache              0
Covid_result          0
Age_60_above          0
Sex                   0
Known_contact         0
dtype: int64
```

3.1.3 Observation:

except (ID) rest all the columns are of string datatype.

[125]:
```
df.dtypes
```

[125]:
```
ID                     int64
Test_date             object
Cough_symptoms        object
Fever                 object
Sore_throat           object
Shortness_of_breath   object
Headache              object
Covid_result          object
Age_60_above          object
Sex                   object
Known_contact         object
dtype: object
```

[126]:
```
df.describe(include="all") # To check all the descriptive information from the dataset.
                           # "all", to see all dattype even it's categorical.
```

[126]:

	ID	Test_date	Cough_symptoms	Fever	Sore_throat
count	278848.000000	278848	278848	278848	278848
unique	NaN	51	5	5	5
top	NaN	20-04-2020	False	False	False
freq	NaN	10921	127531	137774	212584
mean	139424.500000	NaN	NaN	NaN	NaN
std	80496.628269	NaN	NaN	NaN	NaN
min	1.000000	NaN	NaN	NaN	NaN
25%	69712.750000	NaN	NaN	NaN	NaN
50%	139424.500000	NaN	NaN	NaN	NaN
75%	209136.250000	NaN	NaN	NaN	NaN
max	278848.000000	NaN	NaN	NaN	NaN

	Shortness_of_breath	Headache	Covid_result	Age_60_above	Sex
count	278848	278848	278848	278848	278848
unique	5	5	3	3	3
top	False	False	negative	None	female
freq	212842	212326	260227	127320	130158
mean	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN	NaN

```
       Known_contact
count         278848
unique             3
top            Other
freq          242741
mean             NaN
std              NaN
min              NaN
25%              NaN
50%              NaN
75%              NaN
max              NaN
```

[127]:
```
df.nunique() # To check count of unique value in each column.
```

[127]:
```
ID                    278848
Test_date                 51
Cough_symptoms             5
Fever                      5
Sore_throat                5
Shortness_of_breath        5
Headache                   5
Covid_result               3
Age_60_above               3
Sex                        3
Known_contact              3
dtype: int64
```

[128]:
```
#To check each column unique count at once.
for x in df.columns:
    print(x)
    print(df[x].value_counts())
    print()
```

```
ID
1         1
185898    1
185904    1
185903    1
185902    1
         ..
92955     1
92956     1
92957     1
92958     1
278848    1
```

```
Name: ID, Length: 278848, dtype: int64

Test_date
20-04-2020    10921
19-04-2020    10199
22-04-2020     9646
21-04-2020     9624
16-04-2020     9138
23-04-2020     8744
01-04-2020     8654
13-04-2020     8425
02-04-2020     8188
03-04-2020     8079
17-04-2020     7645
05-04-2020     7509
30-04-2020     7313
27-04-2020     7304
15-04-2020     7149
31-03-2020     7134
24-04-2020     7028
26-03-2020     6663
14-04-2020     6571
28-04-2020     6334
18-04-2020     6321
26-04-2020     6131
12-04-2020     5984
27-03-2020     5963
07-04-2020     5931
30-03-2020     5915
10-04-2020     5678
28-03-2020     5602
25-03-2020     5495
06-04-2020     5368
29-03-2020     5277
04-04-2020     5145
25-04-2020     5052
24-03-2020     4735
09-04-2020     4539
11-04-2020     4341
29-04-2020     4259
08-04-2020     4058
22-03-2020     3565
23-03-2020     3494
19-03-2020     2243
18-03-2020     1991
20-03-2020     1870
21-03-2020     1648
17-03-2020     1463
```

```
16-03-2020     1304
15-03-2020      985
13-03-2020      686
12-03-2020      634
14-03-2020      609
11-03-2020      294
Name: Test_date, dtype: int64

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

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

```
Covid_result
negative    260227
positive     14729
other         3892
Name: Covid_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
```

[129]:
```
df.isnull().sum()
```

[129]:
```
ID                    0
Test_date             0
Cough_symptoms        0
Fever                 0
Sore_throat           0
Shortness_of_breath   0
Headache              0
Covid_result          0
Age_60_above          0
Sex                   0
Known_contact         0
dtype: int64
```

3.2 observation :-

we can see that mostly females are highly affected.

[130]:
```
# Creating a list (column_to_replace ), which contain all column name.
# Replacing boolean values and None with appropraite Name and Nan.
column_to_replace =␣
 ↪["Cough_symptoms","Fever","Sore_throat","Shortness_of_breath","Headache","Covid_result","Ag
```

[20]:
```
for  i in column_to_replace:
    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

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

Covid_result
negative    260227
```

```
positive     14729
other         3892
Name: Covid_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
```

3.3 Observation:-

3.3.1 - In some of the columns (True and False) are written twice, we need to replace it.

3.3.2 - We have to remove null(None) value which are present in the dataset.

[131]:
```
df.isnull().sum()
```

[131]:
```
ID                    0
Test_date             0
Cough_symptoms        0
Fever                 0
Sore_throat           0
Shortness_of_breath   0
Headache              0
Covid_result          0
Age_60_above          0
Sex                   0
Known_contact         0
dtype: int64
```

[132]:
```
columns=["Cough_symptoms","Fever","Sore_throat","Shortness_of_breath","Headache"]
```

[133]:
```
# 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)
    df[i].replace("TRUE",True,inplace=True)
    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
```

3.3.3 Covid_result column replacing with mode.

[134]:
```
# to check unique values in "Covid_result" column
df["Covid_result"].unique()
```

[134]:
```
array(['negative', 'positive', 'other'], dtype=object)
```

[135]:
```
#Count of unique values in covid_result
df["Covid_result"].value_counts()
```

[135]:
```
negative    260227
positive     14729
other         3892
Name: Covid_result, dtype: int64
```

[136]:
```
df["Covid_result"].mode()   # To check mode(categorical data)
```

[136]:
```
0    negative
Name: Covid_result, dtype: object
```

[137]:
```
df["Covid_result"].replace("other","negative",inplace=True)
```

[138]:
```
df["Covid_result"].value_counts() #To cross verfify the column.
```

[138]:
```
negative    264119
positive     14729
Name: Covid_result, dtype: int64
```

3.3.4 Age_60_above column replacing with mode.

[139]:
```
df["Age_60_above"].unique()
```

[139]:
```
array(['None', 'No', 'Yes'], dtype=object)
```

[140]:
```
df["Age_60_above"].value_counts()
```

[140]:
```
None    127320
No      125703
Yes      25825
Name: Age_60_above, dtype: int64
```

[141]:
```
df["Age_60_above"].mode()
```

[141]:
```
0    None
Name: Age_60_above, dtype: object
```

[142]:
```
df["Age_60_above"].replace("None","No",inplace=True)
```

[143]:
```
df["Age_60_above"].value_counts() #To cross verfify the column.
```

[143]:
```
No     253023
Yes     25825
Name: Age_60_above, dtype: int64
```

3.3.5 sex column replacing missing values with mode.

[144]:
```
df["Sex"].unique()
```

[144]:
```
array(['None', 'male', 'female'], dtype=object)
```

[145]:
```
#Count of unique values in covid_result
df["Sex"].value_counts()
```

[145]:
```
female    130158
male      129127
None       19563
Name: Sex, dtype: int64
```

[146]:
```
df["Sex"].mode()   # To check mode(categorical data)
```

[146]:
```
0    female
Name: Sex, dtype: object
```

[147]:
```
df["Sex"].replace("None","female",inplace=True)
```

[148]:
```
df["Sex"].value_counts() #To cross verfify the column.
```

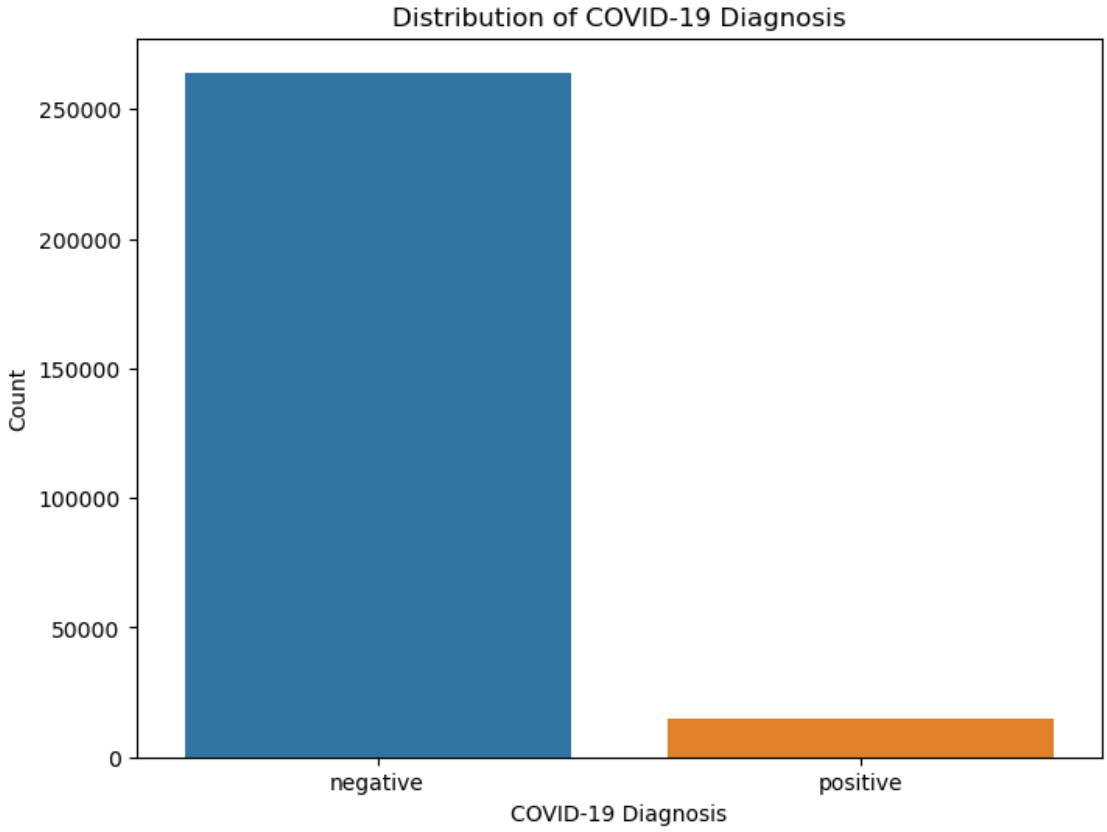
[148]:
```
female    149721
male      129127
Name: Sex, dtype: int64
```

3.4 Converting the data into CSV Format

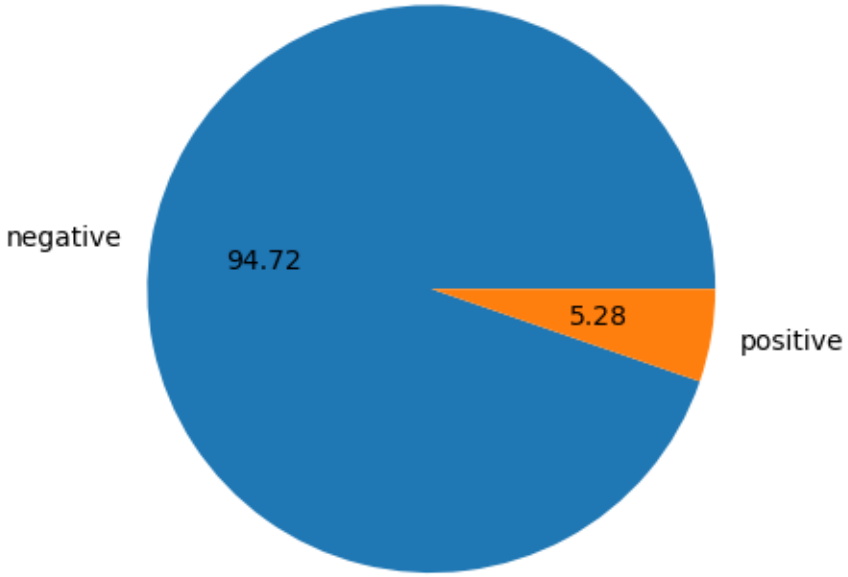
[149]:
```
df.to_csv("Covid_sql.csv")
```

3.5 Data Visualization

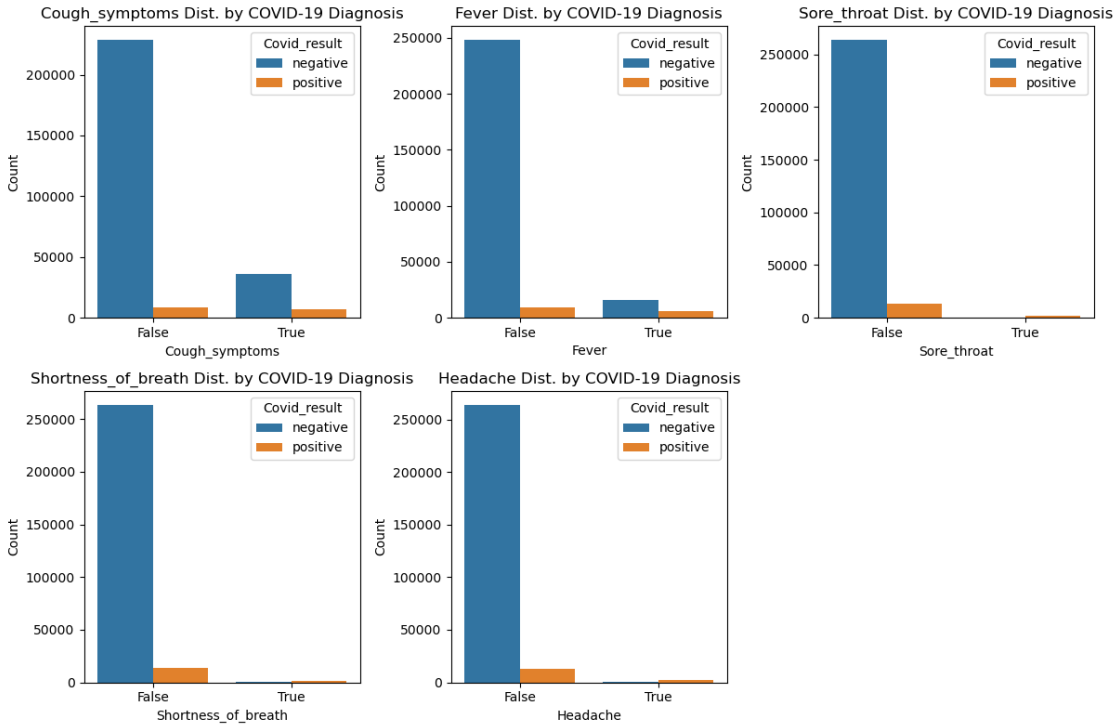
[39]:
```
# Visualize the data to identify covid_result.

plt.figure(figsize=(8, 6))
sns.countplot(x='Covid_result', data=df)
plt.title('Distribution of COVID-19 Diagnosis')
plt.xlabel('COVID-19 Diagnosis')
plt.ylabel('Count')
plt.show()
```



[40]:
```
#Representing into Pie-chart for better understanding.
plt.pie(df['Covid_result'].value_counts(),labels =␣
 ↪['negative','positive'],autopct = "%0.2f")
plt.show()
```



[41]:
```
#Symptoms Distribution Visualization
symptoms = ['Cough_symptoms', 'Fever', 'Sore_throat', 'Shortness_of_breath',␣
 ↪'Headache']

plt.figure(figsize=(12, 8))
for i, symptom in enumerate(symptoms):
    plt.subplot(2, 3, i + 1)
    sns.countplot(x=symptom, data=df, hue='Covid_result')
    plt.title(f'{symptom} Dist. by COVID-19 Diagnosis')
    plt.xlabel(symptom)
    plt.ylabel('Count')

plt.tight_layout()
plt.show()
```



4 Heat Map.

Correlation shows the strength of relationship netween two variable

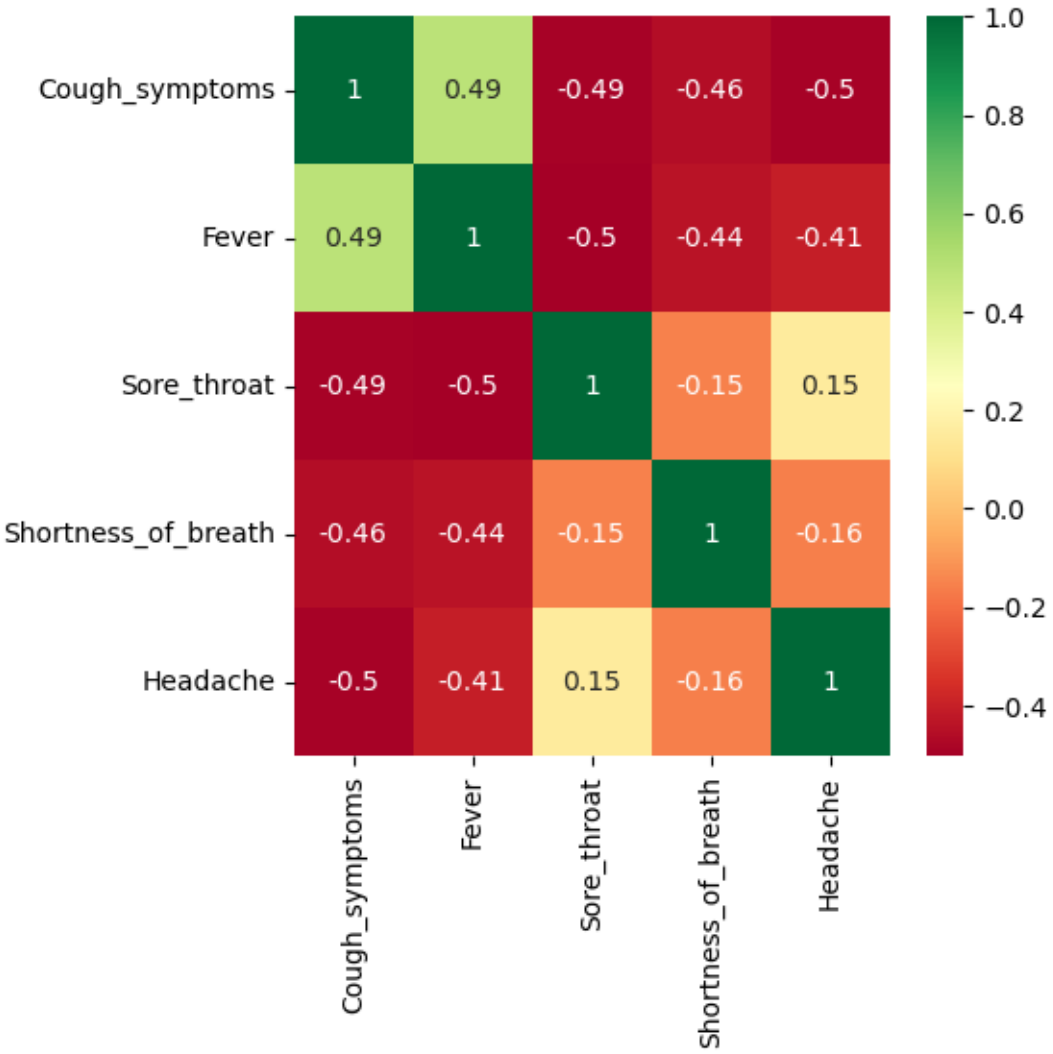
[42]:
```
x=df[["Cough_symptoms","Fever","Sore_throat","Shortness_of_breath","Headache","Covid_result","
  ↪corr()
x
```

[42]:

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

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

[43]:
```
corr_x=x.corr()

plt.figure(figsize = (5,5))
sns.heatmap(data = corr_x,cmap="RdYlGn" ,annot = True)
plt.show()
```



4.1 Observation:-

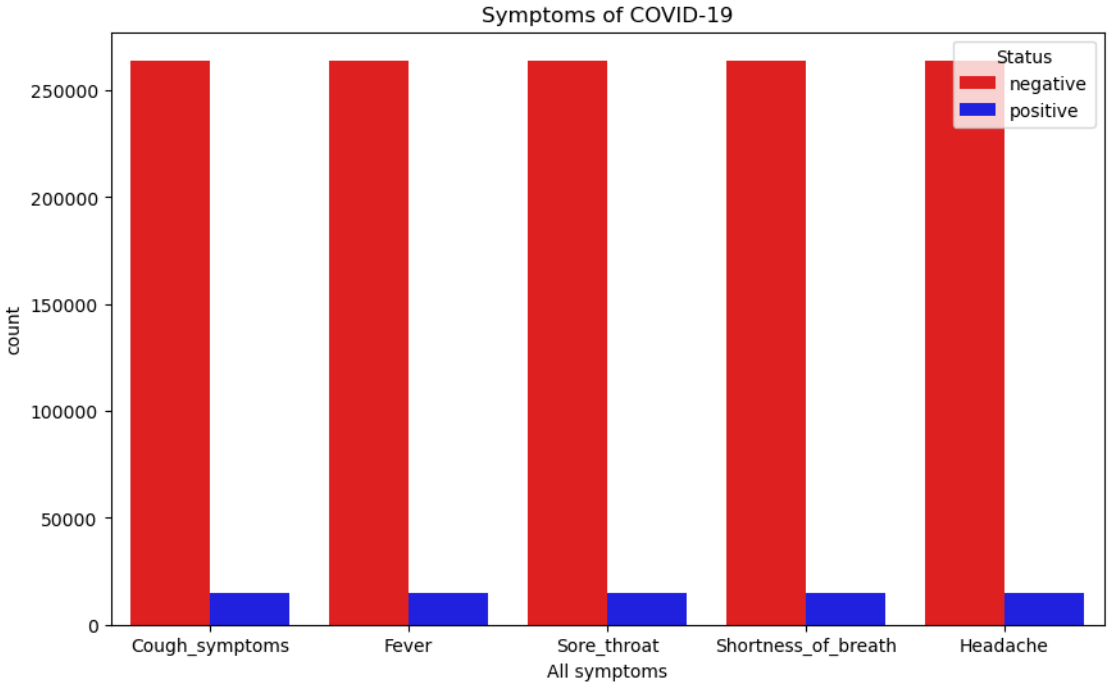
we can see that Cough_symptoms and Fever is having high correlation.

Sore_throat and Headache has next higher correlation in the above Heatmap.

We can easily see from the above heatmap, the value are likely highly Correlated ,which are near to 1.

5 Plotting Bar chart to show all symptoms of Negative and Positive patients.

[44]:
```
# Assuming df is your DataFrame
df_melted = pd.melt(df, id_vars=["Covid_result"], value_vars=["Cough_symptoms",␣
 ↪"Fever", "Sore_throat", "Shortness_of_breath", "Headache"])

# Define a custom color palette
custom_palette = {"positive": "blue", "negative": "red"}  # Replace "Positive"␣
 ↪and "Negative" with your actual class labels

# Creating Grouped Bar chart using Seaborn Library with custom palette
plt.figure(figsize=(10, 6))
sns.countplot(data=df_melted, x="variable", hue="Covid_result",␣
 ↪palette=custom_palette)
plt.xlabel("All symptoms")
plt.ylabel("count")
plt.title("Symptoms of COVID-19")
plt.legend(title="Status")
plt.show()
```



5.1 Observation:-

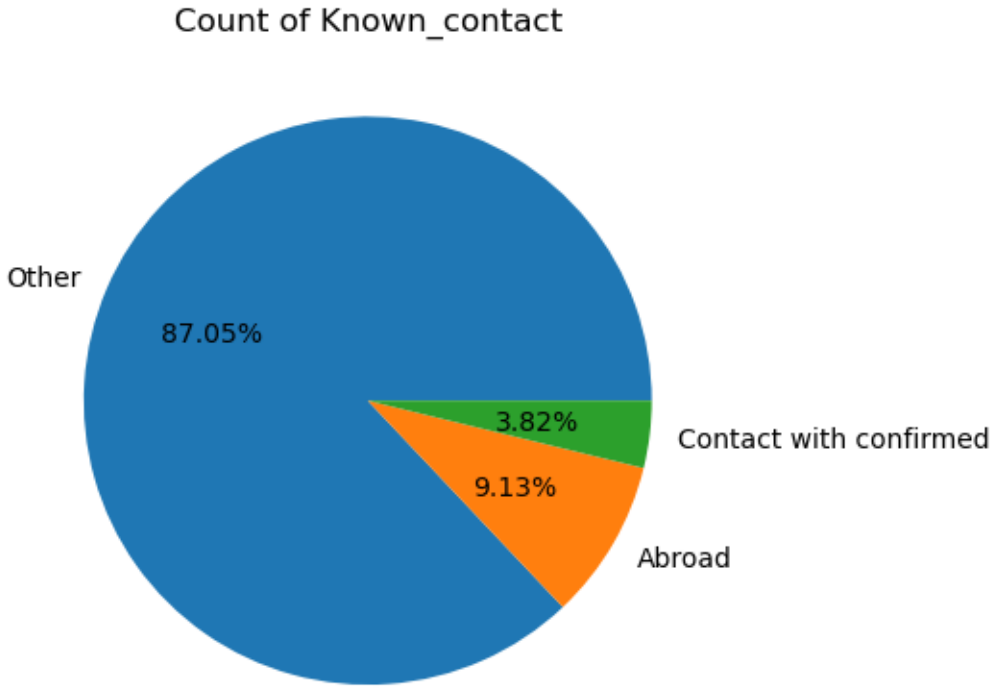
5.1.1 From the above Graph we undestand that Negative patients are more compared to Postive.

5.2 Count of Patient using Pie chart.

[45]:
```
df["Known_contact"].unique()
```

[45]:
```
array(['Abroad', 'Contact with confirmed', 'Other'], dtype=object)
```

[46]:
```
a=df["Known_contact"].value_counts()
b=['Other','Abroad', 'Contact with confirmed']
plt.pie(a,labels=b,autopct="%1.2f%%")
plt.title("Count of Known_contact")
plt.show()
```



5.3 Observation:-

By above chart , we get that (Other) has 87.05% , which is highest compared to (Abroad),(Contact with confirmed).

5.4 Feature Engineering

It is the process of transforming the features that better represent the underlying problem to predictive models, resulting in improved model accuracy on unseen data.

It includes feature creation, Feature scaling,feature extraction and feature selection.

Here , we will convert categorical data into numerical(continous) data first.

[47]:
```
#Assuming "Cough_symptons" contains boolean Values
df["Cough_symptoms"] =df["Cough_symptoms"].astype(int)
```

[48]:
```
#In Fever we assign True= 1 , False =0
df["Fever"]=df["Fever"].astype(int)
```

[49]:
```
#In Sore_throat we assign True= 1 , False =0
df["Sore_throat"]=df["Sore_throat"].astype(int)
```

[50]:
```
#In Shortness_of_breath we assign True= 1 , False =0
df["Shortness_of_breath"]=df["Shortness_of_breath"].astype(int)
```

[51]:
```
#In Headache we assign True= 1 , False =0
df["Headache"]=df["Headache"].astype(int)
```

[52]:
```
#In Covid_result we assign positive= 1 , negative =0
df["Covid_result"]=df["Covid_result"].map({"positive":1,"negative":0})
```

[53]:
```
#In Age_60_above we assign Yes= 1 , No =0
df["Age_60_above"]=df["Age_60_above"].map({"Yes":1,"No":0})
```

[54]:
```
#In Sex we assign female= 1 , male =0
df["Sex"]=df["Sex"].map({"female":1,"male":0})
```

[55]:
```
#In Known_contact we assign other= 1 , Abroad =2, Contact with confirmed=3
df["Known_contact"]=df["Known_contact"].map({"Other":1, "Abroad":2, "Contact␣
 ↪with confirmed":3})
```

5.5 Converting our Test_date columns into Date Datatype

[56]:
```
df["Test_date"]=pd.to_datetime(df["Test_date"],format="%d-%m-%Y")
```

[57]:
```
columns=["Cough_symptoms","Fever","Sore_throat","Shortness_of_breath","Headache","Covid_result
columns
```

[57]:
```
['Cough_symptoms',
 'Fever',
 'Sore_throat',
 'Shortness_of_breath',
```

```
 'Headache',
 'Covid_result',
 'Age_60_above',
 'Sex',
 'Known_contact']
```

[58]:
```
#To check count of each and every columns.
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]

Covid_result
[0 1]

Age_60_above
[0 1]

Sex
[1 0]

Known_contact
[2 3 1]
```

[59]:
```
df
```

[59]:

	ID	Test_date	Cough_symptoms	Fever	Sore_throat	\
0	1	2020-03-11	1	0	1	
1	2	2020-03-11	0	1	0	
2	3	2020-03-11	0	1	0	
3	4	2020-03-11	1	0	0	

```
4            5 2020-03-11               1      0            0
…          …       …              …     …          …
278843  278844 2020-04-30               0      0            0
278844  278845 2020-04-30               0      0            0
278845  278846 2020-04-30               0      0            0
278846  278847 2020-04-30               0      0            0
278847  278848 2020-04-30               0      0            0

        Shortness_of_breath  Headache  Covid_result  Age_60_above  Sex  \
0                         0         0             0             0    1
1                         0         0             1             0    1
2                         0         0             1             0    1
3                         0         0             0             0    1
4                         0         0             0             0    1
…                       …       …           …           …  …
278843                    0         0             1             0    0
278844                    0         0             0             0    1
278845                    0         0             0             0    0
278846                    0         0             0             0    0
278847                    0         0             0             0    1

        Known_contact
0                   2
1                   2
2                   2
3                   2
4                   3
…                 …
278843              1
278844              1
278845              1
278846              1
278847              1

[278848 rows x 11 columns]
```

[60]:
```
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   Covid_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
```

5.6 Observation:-

5.6.1 As we converted all datatype(categorical) into integer. Since there is no categorical columns are present, therefore (Feature Transformation) and (Feature scaling) is not applicable for this dataset.

5.7 Spliting Training and Testing Data.

[61]:
```
df.head()
```

[61]:

	ID	Test_date	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	\
0	1	2020-03-11	1	0	1	0	
1	2	2020-03-11	0	1	0	0	
2	3	2020-03-11	0	1	0	0	
3	4	2020-03-11	1	0	0	0	
4	5	2020-03-11	1	0	0	0	

	Headache	Covid_result	Age_60_above	Sex	Known_contact
0	0	0	0	1	2
1	0	1	0	1	2
2	0	1	0	1	2
3	0	0	0	1	2
4	0	0	0	1	3

[62]:
```
df.shape
```

[62]:
```
(278848, 11)
```

5.8 Train/Test Split based on date

5.8.1 As per our business requirement we have to take data from 11th March 2020 to 15th April 2020 as Training Set & Validation Set.

5.8.2 And , data from 20th April to 30th april as Test Set.

[63]:
```
#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("2020-04-15")
```

```
df_fifteenthapril = df[df["Test_date"] <= break_date]
df_twentyapril = df[df["Test_date"] > break_date]
```

[64]:
```
#data from 11th march 2020 to 15th april 2020
df_fifteenthapril
```

[64]:

	ID	Test_date	Cough_symptoms	Fever	Sore_throat	\
0	1	2020-03-11	1	0	1	
1	2	2020-03-11	0	1	0	
2	3	2020-03-11	0	1	0	
3	4	2020-03-11	1	0	0	
4	5	2020-03-11	1	0	0	
…	…	…	…	…	…	
163184	163185	2020-04-15	0	0	0	
163185	163186	2020-04-15	0	0	0	
163186	163187	2020-04-15	0	0	0	
163187	163188	2020-04-15	0	0	0	
163188	163189	2020-04-15	0	0	0	

	Shortness_of_breath	Headache	Covid_result	Age_60_above	Sex	\
0	0	0	0	0	1	
1	0	0	1	0	1	
2	0	0	1	0	1	
3	0	0	0	0	1	
4	0	0	0	0	1	
…	…	…	…	…	…	
163184	0	0	0	0	0	
163185	0	0	0	0	0	
163186	0	0	0	0	1	
163187	0	0	0	0	0	
163188	0	0	0	0	0	

	Known_contact
0	2
1	2
2	2
3	2
4	3
…	…
163184	1
163185	1
163186	1
163187	1
163188	1

[163189 rows x 11 columns]

[65]:
```
df_twentyapril
```

[65]:
```
            ID  Test_date  Cough_symptoms  Fever  Sore_throat  \
163189  163190 2020-04-16               1      0            0
163190  163191 2020-04-16               0      0            0
163191  163192 2020-04-16               1      0            0
163192  163193 2020-04-16               0      0            0
163193  163194 2020-04-16               0      0            0
…          …        …               …     …           …
278843  278844 2020-04-30               0      0            0
278844  278845 2020-04-30               0      0            0
278845  278846 2020-04-30               0      0            0
278846  278847 2020-04-30               0      0            0
278847  278848 2020-04-30               0      0            0

        Shortness_of_breath  Headache  Covid_result  Age_60_above  Sex  \
163189                    0         0             0             0    1
163190                    0         0             0             0    1
163191                    0         0             0             0    1
163192                    0         0             0             0    0
163193                    0         0             0             0    0
…                       …        …             …             …   …
278843                    0         0             1             0    0
278844                    0         0             0             0    1
278845                    0         0             0             0    0
278846                    0         0             0             0    0
278847                    0         0             0             0    1

        Known_contact
163189              2
163190              1
163191              2
163192              1
163193              1
…                 …
278843              1
278844              1
278845              1
278846              1
278847              1

[115659 rows x 11 columns]
```

[66]:
```
# Dropping column ID,Test_date and Outcome variable(covid_result), and starting␣
 ↪is to X_train .
X_train=df_fifteenthapril.drop(columns=["ID","Test_date","Covid_result"],axis=1)
X_train
```

[66]:
```
        Cough_symptoms  Fever  Sore_throat  Shortness_of_breath  Headache  \
0                    1      0            1                    0         0
1                    0      1            0                    0         0
2                    0      1            0                    0         0
3                    1      0            0                    0         0
4                    1      0            0                    0         0
…                  …    …          …                  …        …
163184               0      0            0                    0         0
163185               0      0            0                    0         0
163186               0      0            0                    0         0
163187               0      0            0                    0         0
163188               0      0            0                    0         0

        Age_60_above  Sex  Known_contact
0                  0    1              2
1                  0    1              2
2                  0    1              2
3                  0    1              2
4                  0    1              3
…                …  …            …
163184             0    0              1
163185             0    0              1
163186             0    1              1
163187             0    0              1
163188             0    0              1

[163189 rows x 8 columns]
```

[67]:
```
# starting outcome variable in y_train.
y_train=df_fifteenthapril["Covid_result"]
y_train
```

[67]:
```
0         0
1         1
2         1
3         0
4         0
         ..
163184    0
163185    0
163186    0
163187    0
163188    0
Name: Covid_result, Length: 163189, dtype: int64
```

[68]:
```
#dropping Test_date and outcome variable(covid_result) column from df_test and␣
 ↪starting it to X_test.
```

```
X_test =df_twentyapril.drop(columns=["ID","Test_date","Covid_result"],axis=1)
X_test
```

[68]:

	Cough_symptoms	Fever	Sore_throat	Shortness_of_breath	Headache	\
163189	1	0	0	0	0	
163190	0	0	0	0	0	
163191	1	0	0	0	0	
163192	0	0	0	0	0	
163193	0	0	0	0	0	
…	…	…	…	…	…	
278843	0	0	0	0	0	
278844	0	0	0	0	0	
278845	0	0	0	0	0	
278846	0	0	0	0	0	
278847	0	0	0	0	0	

	Age_60_above	Sex	Known_contact
163189	0	1	2
163190	0	1	1
163191	0	1	2
163192	0	0	1
163193	0	0	1
…	…	…	…
278843	0	0	1
278844	0	1	1
278845	0	0	1
278846	0	0	1
278847	0	1	1

[115659 rows x 8 columns]

[69]:
```
# starting outcome variable in y_train.
y_test=df_twentyapril["Covid_result"]
y_test
```

[69]:
```
163189    0
163190    0
163191    0
163192    0
163193    0
         ..
278843    1
278844    0
278845    0
278846    0
278847    0
Name: Covid_result, Length: 115659, dtype: int64
```

5.9 Train Test Split

[70]:
```
#As we have to divide into 4:1 means20%
from sklearn.model_selection import train_test_split
X_train,X_val,y_train,y_val =train_test_split(X_train,y_train,test_size=0.
 ↪20,random_state=0)
```

[71]:
```
# Training set
print("X_train shape:{}".format(X_train.shape))
print("y_train shape:{}".format(y_train.shape))

#Test set
print("X_test shape:{}".format(X_test.shape))
print("y_test shape:{}".format(y_test.shape))

#Validation set
print("X_val shape:{}".format(X_val.shape))
print("y_val shape:{}".format(y_val.shape))
```

```
X_train shape:(130551, 8)
y_train shape:(130551,)
X_test shape:(115659, 8)
y_test shape:(115659,)
X_val shape:(32638, 8)
y_val shape:(32638,)
```

5.10 Feature Selection

[72]:
```
from scipy.stats import chi2_contingency
columns␣
 ↪=['ID','Test_date','Cough_symptoms','Fever','Sore_throat','Shortness_of_breath','Headache',
for i in columns:
# creating a comtingency table
    contingency_table = pd.crosstab(df[i],df["Covid_result"])
    #pd.crosstab , It is a useful tool for analyzing the relationship between␣
 ↪differnt categorical variable in your dataset.

    #perform chi-square-test,calculating p=value
    chi2,p_value,dof,expected = chi2_contingency(contingency_table)


    #To print the result.
    print('------{}------'.format(i))
    print('chi-square statistics: ',chi2)
    print('p_value = ',p_value)
```

```
------ID------
chi-square statistics:  278847.99999999994
```

```
p_value =  0.4991096512866681
------Test_date------
chi-square statistics:  4498.032907406993
p_value =  0.0
------Cough_symptoms------
chi-square statistics:  10569.415074648161
p_value =  0.0
------Fever------
chi-square statistics:  19378.570935486066
p_value =  0.0
------Sore_throat------
chi-square statistics:  21183.30774235602
p_value =  0.0
------Shortness_of_breath------
chi-square statistics:  14873.153774171122
p_value =  0.0
------Headache------
chi-square statistics:  37078.834270861014
p_value =  0.0
------Age_60_above------
chi-square statistics:  600.9907438227524
p_value =  1.0193061909600926e-132
------Sex------
chi-square statistics:  140.4145884069575
p_value =  2.1604974877258956e-32
------Known_contact------
chi-square statistics:  90331.28046978849
p_value =  0.0
```

5.11 Observation:-

5.11.1 - We have taken Chi-square test Because, All columns are categorical .

5.11.2 - Here we got P_value for columns less than 0.5 so all indepedent columns have relationship with dependent column i.e-"Covid_result".

5.11.3 - We have getting p_value for "ID" column=0.499. If P_value Is greater than 0.05. Then we can draw particular column.

6 Machine Learning Algorithms

6.0.1 As from the Problem statement :: We need to predict the Covid Positive or Negative Based on different Independent columns.

The output result column i.e,(Covid_result) is a binary classification task. So we can use supervised machine learning algorithm for this data set. We will use the following 4 ML algorithm, namely

1.. Decision tree

2.. Logistic Regression.

3.. Random forest.

4.. K-Nearest Neighbour(KNN) model.

6.1 1. Decision Tree

[73]:
```
#Importing Libraries.
from sklearn.tree import DecisionTreeClassifier
```

[74]:
```
#Creating a decision tree classifier.
dtree = DecisionTreeClassifier()
```

[75]:
```
# Fit the model to the training data: X_train and Y_train?
dtree.fit(X_train,y_train)
```

[75]:
```
DecisionTreeClassifier()
```

6.1.1 Predicting the model

[76]:
```
# Making predictionson test data set.
predictions=dtree.predict(X_test)
predictions
```

[76]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int64)
```

6.1.2 Evaluation of Decision Tree

[77]:
```
#Importing Libraries.
from sklearn.metrics import classification_report, confusion_matrix
```

[78]:
```
#calculating Precison,recall,F1-score and support.
print(classification_report(y_test,predictions))
#precison =(totalTP /(TP+FP))
#recall = (totalTP /(TP+FN))
#F1_score =2×Precision×Recall/Precision+Recall
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    112961
           1       0.77      0.44      0.56      2698

    accuracy                           0.98    115659
   macro avg       0.88      0.72      0.77    115659
weighted avg       0.98      0.98      0.98    115659
```

[79]:
```
# calculating Confusion matrix.
print("-----Confusion Matrix-----")
print(confusion_matrix(y_test,predictions))
```

```
-----Confusion Matrix-----
[[112611    350]
 [  1523   1175]]
```

[80]:
```
# Calculating accuracy.
accuracy= dtree.score(X_test,y_test)
accuracy
```

[80]: 0.9838058430385875

6.2 Observation By Decison Tree.

6.2.1 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 case and Precision is 0.79. In this model prediction is better for COVID-19 negative cases rather than COVID positive cases.

6.3 2. Random Forest.

[81]:
```
# Importing Libraries.
from sklearn.ensemble import RandomForestClassifier
```

[82]:
```
rfc=RandomForestClassifier(n_estimators=100)
rfc.fit(X_train,y_train)
```

[82]: RandomForestClassifier()

[83]:
```
#Making preditions on testdata.
rfc_pred=rfc.predict(X_test)
rfc_pred
```

[83]: array([0, 0, 0, …, 0, 0, 0], dtype=int64)

[84]:
```
#calculating Precison,recall,F1-score and support.
print(classification_report(y_test,rfc_pred))
#precison =(totalTP /(TP+FP))
#recall = (totalTP /(TP+FN))
#F1_score =2×Precision×Recall/Precision+Recall
```

	precision	recall	f1-score	support
0	0.99	1.00	0.99	112961

```
           1       0.77      0.44      0.56      2698

    accuracy                           0.98    115659
   macro avg       0.88      0.72      0.77    115659
weighted avg       0.98      0.98      0.98    115659
```

6.4 Evaluation of Random forest.

[85]:
```
# calculating Confusion matrix.
print("-----Confusion Matrix-----")
print(confusion_matrix(y_test,rfc_pred))
```

```
-----Confusion Matrix-----
[[112609    352]
 [  1523   1175]]
```

[86]:
```
# Calculating accuracy.
accuracy= rfc.score(X_test,y_test)
accuracy
```

[86]: 0.9837885508261355

6.5 Observation By Random Forest.

6.5.1 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 case and Precision is 0.79. In this model prediction is better for COVID-19 negative cases rather than COVID positive cases.

6.6 3. Logistic Regression.

[87]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Creating Logistic Regression Model.

[88]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
```

[89]:
```
# To fit the model to the training data: X_train and y_train
lr.fit(X_train,y_train)
```

[89]:
```
LogisticRegression()
```

6.6.1 Make predictions on the test data.

[90]:
```
y_pred = lr.predict(X_test)
y_pred
```

[90]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int64)
```

[91]:
```
# calculating Confusion matrix.
print("-----Confusion Matrix-----")
print(confusion_matrix(y_test,y_pred))
```

```
-----Confusion Matrix-----
[[112802    159]
 [  2370    328]]
```

6.6.2 Evalution of Logistic Regression.

[ ]:
```
#calculating Precison,recall,F1-score and support.
print(classification_report(y_test,rfc_pred))
#precison =(totalTP /(TP+FP))
#recall = (totalTP /(TP+FN))
#F1_score =2×Precision×Recall/Precision+Recall
```

[93]:
```
#Calculating accuracy.
accuracy = lr.score(X_test,y_test)
accuracy
```

[93]:
```
0.9781339973542915
```

6.7 Observation of Logistic Regression.

- Logistic regression model code 97.98% accuracy, which means this model prediction is also good compared to the previous model.

- For all COVID negative cases, precision is very good. It is. 0.99 for COVID positive cases precision is 0.77.

6.8 4.K-Nearest Neighbour(KNN) model.

[94]:
```
# Importing Libraries
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5)
```

[95]:
```
# To fit the model to the training data: X_train and y_train
classifier.fit(X_train,y_train)
```

[95]:
```
KNeighborsClassifier()
```

6.8.1 Make predictions on the test data.

[96]:
```
y_pred = classifier.predict(X_test)
```

[97]:
```
y_pred
```

[97]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int64)
```

[98]:
```
# calculating Confusion matrix.
print("-----Confusion Matrix-----")
print(confusion_matrix(y_test,y_pred))
```

```
-----Confusion Matrix-----
[[111823   1138]
 [  1194   1504]]
```

6.8.2 Evalution of KNN Model.

[99]:
```
#calculating Precison,recall,F1-score and support.
print(classification_report(y_test,y_pred))
#precison =(totalTP /(TP+FP))
#recall = (totalTP /(TP+FN))
#F1_score =2×Precision×Recall/Precision+Recall
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    112961
           1       0.57      0.56      0.56      2698

    accuracy                           0.98    115659
   macro avg       0.78      0.77      0.78    115659
weighted avg       0.98      0.98      0.98    115659
```

[ ]:
```
#Calculating accuracy.
accuracy = classifier.score(X_test,y_test)
```

[ ]:
```
accuracy
```

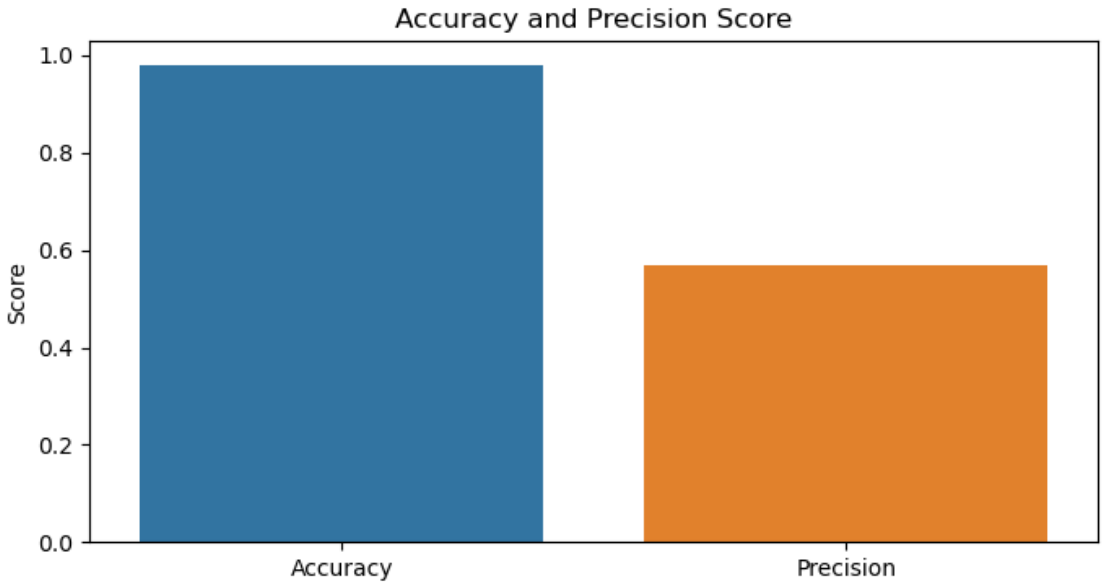
6.8.3 Observation of KNN Model.

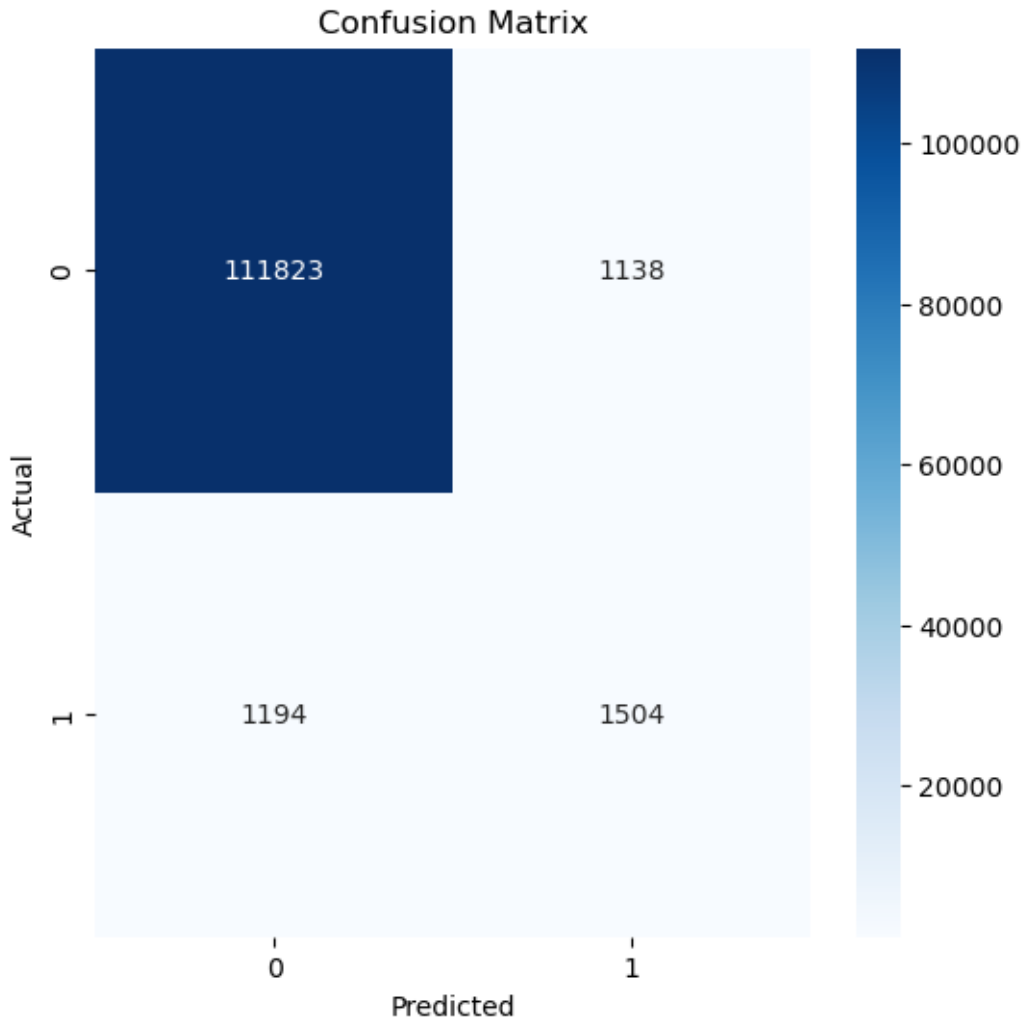
6.8.4 - This model predicted 97% accuracy with KNN algorithm.

6.9 Accuracy comparison of 4 ML Models.

[102]:
```
from sklearn.metrics import accuracy_score,precision_score, confusion_matrix,␣
 ↪classification_report

accuracy = accuracy_score(y_test,y_pred)
print("Accuracy:", accuracy)

knn_precision = precision_score(y_test, y_pred,zero_division=0)
print("Precision Score",knn_precision)

knn_confusion_matrix = confusion_matrix(y_test, y_pred)
print("Score Of Confusion Matrix",knn_confusion_matrix)

knn_classification_report = classification_report(y_test,␣
 ↪y_pred,zero_division=0)

plt.figure(figsize=(8, 4))
scores = {'Accuracy': accuracy, 'Precision': knn_precision}
sns.barplot(x=list(scores.keys()), y=list(scores.values()))
plt.title('Accuracy and Precision Score')
plt.ylabel('Score')
plt.show()

plt.figure(figsize=(6, 6))
sns.heatmap(knn_confusion_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("Classification Report:")
print(knn_classification_report)
```

```
Accuracy: 0.9798372802808255
Precision Score 0.5692657077971234
Score Of Confusion Matrix [[111823   1138]
 [  1194   1504]]
```


Accuracy and Precision Score
1.0
0.8
0.6
0.4
0.2
0.0
Score
Accuracy
Precision




```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    112961
           1       0.57      0.56      0.56      2698

    accuracy                           0.98    115659
   macro avg       0.78      0.77      0.78    115659
weighted avg       0.98      0.98      0.98    115659
```

[103]:
```
fig =px.bar(x=["Decision Tree","Random Forest","Logistic Regression","K-Nearest␣
 ↪Neighbors"],
            y=[98.38,98.37,97.81,97.98],
            text=[98.38,98.37,97.81,97.98],
```

```
            labels={"x":"Machine Learning ALgorithms","y":"Accuracy(%)"},
            title="Comparison of accuracy of ML models",
            color=["Decision Tree","Random Forest","Logistic␣
 ↪Regression","K-Nearest Neighbors"])
fig.show()
```

7 Conclusion.

7.0.1 - We are getting 98% accuracy using Decision Tree , RandomForest and KNN

7.0.2 - with Logistics regression we are getting 97% accuracy.

7.0.3 - So, for deployment we can use any model between best three.

7.0.4 - We got best accuracy using Decision Tree, RandomForest.