**Author:**

Govind G
https://github.com/govindgopidas

## Section 1: Questions to Answer

1. **Why is your proposal important in today's world?**

   - In the midst of the ongoing COVID-19 pandemic, the importance of this proposal cannot be overstated. Accurate disease prediction is crucial to address the urgent need for effective screening and diagnosis. By harnessing the power of machine learning and data analytics, this project can play a pivotal role in saving lives and mitigating the social and economic impact of the virus. Furthermore, the knowledge gained from this project has broader implications for managing future disease outbreaks, making it highly relevant and timely.

2. **How predicting a disease accurately can improve medical treatment?**

   - Accurate disease prediction is not just a matter of convenience but a matter of life and death. By predicting COVID-19 outcomes with precision, healthcare professionals can intervene early, administer appropriate treatments, and prevent severe complications. This can lead to reduced mortality rates, shorter hospital stays, and more efficient resource utilization, ultimately resulting in better healthcare outcomes.

3. **How is it going to impact the medical field when it comes to effective screening and reducing healthcare burden?**

   - The impact of this project on the medical field is profound. Accurate prediction models can revolutionize the screening process, allowing for targeted testing of high-risk individuals. By identifying and isolating COVID-19-positive cases more efficiently, healthcare systems can reduce the burden of excessive testing and focus resources on those who need it most. This optimization not only conserves resources but also minimizes the risk of disease transmission within healthcare settings.

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

   - Beyond its immediate relevance, this project addresses a broader knowledge gap in disease prediction and management. The methodologies and insights gained can be adapted for future infectious diseases, providing a valuable framework for early detection and intervention. This project, therefore, serves as a blueprint for proactive healthcare strategies that can be deployed swiftly in the face of emerging diseases, strengthening global healthcare preparedness.

## Section 2: Initial Hypothesis (or hypotheses)

Initial hypotheses may include:

- Patients with certain symptoms (e.g., shortness of breath, fever) are more likely to test positive for COVID-19.
- Elderly individuals (age ≥ 60) are at higher risk of testing positive.
- Known contact with COVID-19-positive individuals increases the likelihood of testing positive.
- Different machine learning models may perform differently in predicting COVID-19 outcomes, and we need to justify the choice of the best model.

## Section 3: Data Analysis Approach

Data analysis approach:

- Perform exploratory data analysis (EDA) to identify data patterns, distributions, and outliers.
- Conduct feature engineering to create relevant features or transform existing ones.
- Visualize data to understand relationships between variables.
- Use statistical tests to validate hypotheses about the relationship between symptoms, age, contact history, and COVID-19 outcomes.

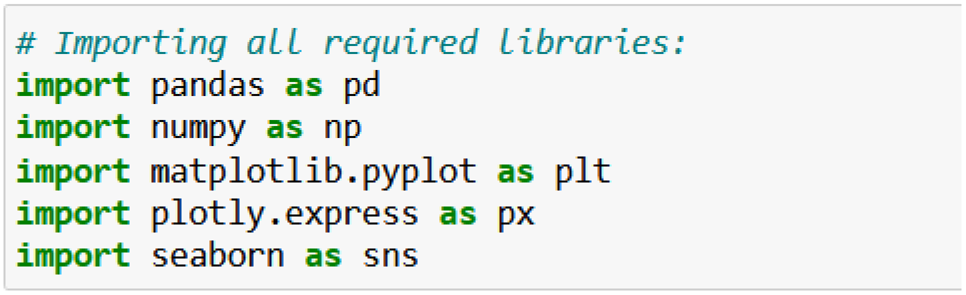
### 3.1 Business objective

### 1. Business objective

The project aims to develop an AI-driven predictive model, 'COVID-19 Prognosis AI: Enhancing Diagnosis, Reducing Burden,' to significantly improve the accuracy of COVID-19 diagnosis by leveraging patient symptoms, age, and contact history. This initiative seeks to enhance medical diagnosis, streamline resource allocation in healthcare, reduce the burden on healthcare systems, provide a scalable framework for future infectious diseases, and contribute valuable insights to ongoing research efforts in the field, ultimately improving public health outcomes and preparedness.

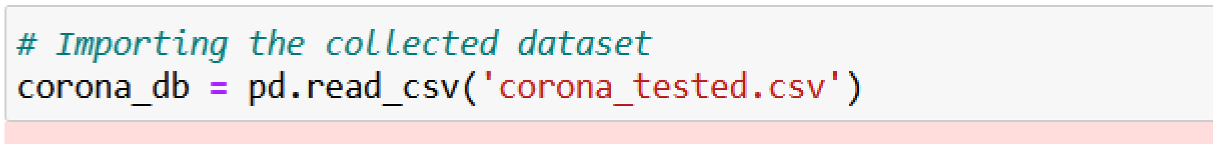
## Data Preparation/Data preprocessing: Feature  Engineering + Feature selection

### 3.2 Importing the required libraries



```python
# Importing all required libraries:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
```

### 3.3 Importing the dataset



```python
# Importing the collected dataset
corona_db = pd.read_csv('corona_tested.csv')
```

## 3.4 A quick glance

```
# Quick glance
corona_db.head()
```

| | 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 |

## 3.5 Table info

```
corona_db.info()
```

```
<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
```

### 3.6 Table columns

```
corona_db.columns
```

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

### 3.7 Table data type

```
corona_db.dtypes
```

```
Ind_ID                  int64
Test_date              object
Cough_symptoms         object
Fever                  object
Sore_throat            object
Shortness_of_breath    object
Headache               object
Corona                 object
Age_60_above           object
Sex                    object
Known_contact          object
dtype: object
```

## 3.8 Table Description

```
corona_db.describe(include='all')
```

| | Ind_ID | Test_date | Cough_symptoms | Fever | Sore_throat | Shortness_of_breath | Headache | Corona | 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 |

## 3.9 Checking whether there is any wrong entries

```
corona_db["Cough_symptoms"].unique()
```

```
array(['TRUE', 'FALSE', 'None', False, True], dtype=object)
```

```
corona_db["Fever"].unique()
```

```
array(['FALSE', 'TRUE', 'None', False, True], dtype=object)
```

```
corona_db["Sore_throat"].unique()
```

```
array(['TRUE', 'FALSE', 'None', False, True], dtype=object)
```

```
corona_db["Headache"].unique()
```

```
array(['FALSE', 'TRUE', 'None', False, True], dtype=object)
```

```
corona_db["Corona"].unique()
```

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

```
corona_db["Age_60_above"].unique()
```

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

```
corona_db["Sex"].unique()
```

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

```
corona_db["Known_contact"].unique()
```

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

## 3.10 Standardizing the entries (Data cleaning)

- As we can see there are some columns with various representations of boolean values, including strings ('TRUE' and 'FALSE') as well as Python's boolean values (True and False).
- The presence of multiple representations of the same information (e.g., 'TRUE', 'FALSE', True, False) can affect the nominal encoding of categorical data in a machine learning context. To effectively encode such data, you should ensure that the values are consistent and properly converted to a single data type.

```
# Standardizing the entries in the column Cough_symptoms
corona_db['Cough_symptoms'] = corona_db['Cough_symptoms'].replace({'TRUE': True, 'FALSE': False})
```

```
corona_db["Cough_symptoms"].unique()
```

```
array([True, False, 'None'], dtype=object)
```

```
# Standardizing the entries in the column Fever
corona_db['Fever'] = corona_db['Fever'].replace({'TRUE': True, 'FALSE': False})
```

```
corona_db['Fever'].unique()
```

```
array([False, True, 'None'], dtype=object)
```

```
# Standardizing the entries in the column Sore_throat
corona_db['Sore_throat'] = corona_db['Sore_throat'].replace({'TRUE': True, 'FALSE': False})
```

```
corona_db['Sore_throat'].unique()
```

```
array([True, False, 'None'], dtype=object)
```

```
# Standardizing the entries in the column Sore_throat
corona_db['Headache'] = corona_db['Headache'].replace({'TRUE': True, 'FALSE': False})
```

```
corona_db['Headache'].unique()
```

```
array([False, True, 'None'], dtype=object)
```

### 3.11 Checking for null values

```
corona_db.isnull().sum()
```

```
Ind_ID                 0
Test_date              0
Cough_symptoms         0
Fever                  0
Sore_throat            0
Shortness_of_breath    0
Headache               0
Corona                 0
Age_60_above           0
Sex                    0
Known_contact          0
dtype: int64
```

### 3.12 Checking for duplicates

```
corona_db.duplicated()
```

```
0         False
1         False
2         False
3         False
4         False
          ...
278843    False
278844    False
278845    False
278846    False
278847    False
Length: 278848, dtype: bool
```

```
corona_db.duplicated().sum()
```

```
0
```

## 3,13 Renaming the columns properly (Data cleaning)

```
# Renaming columns properly:

corona_db.rename(columns = {
    'Ind_ID':'Id',
    'Cough_symptoms':'Cough',
    'Corona':'Test_results'
}, inplace = True)
```

## 3.14 Rearranging the columns (Data cleaning)

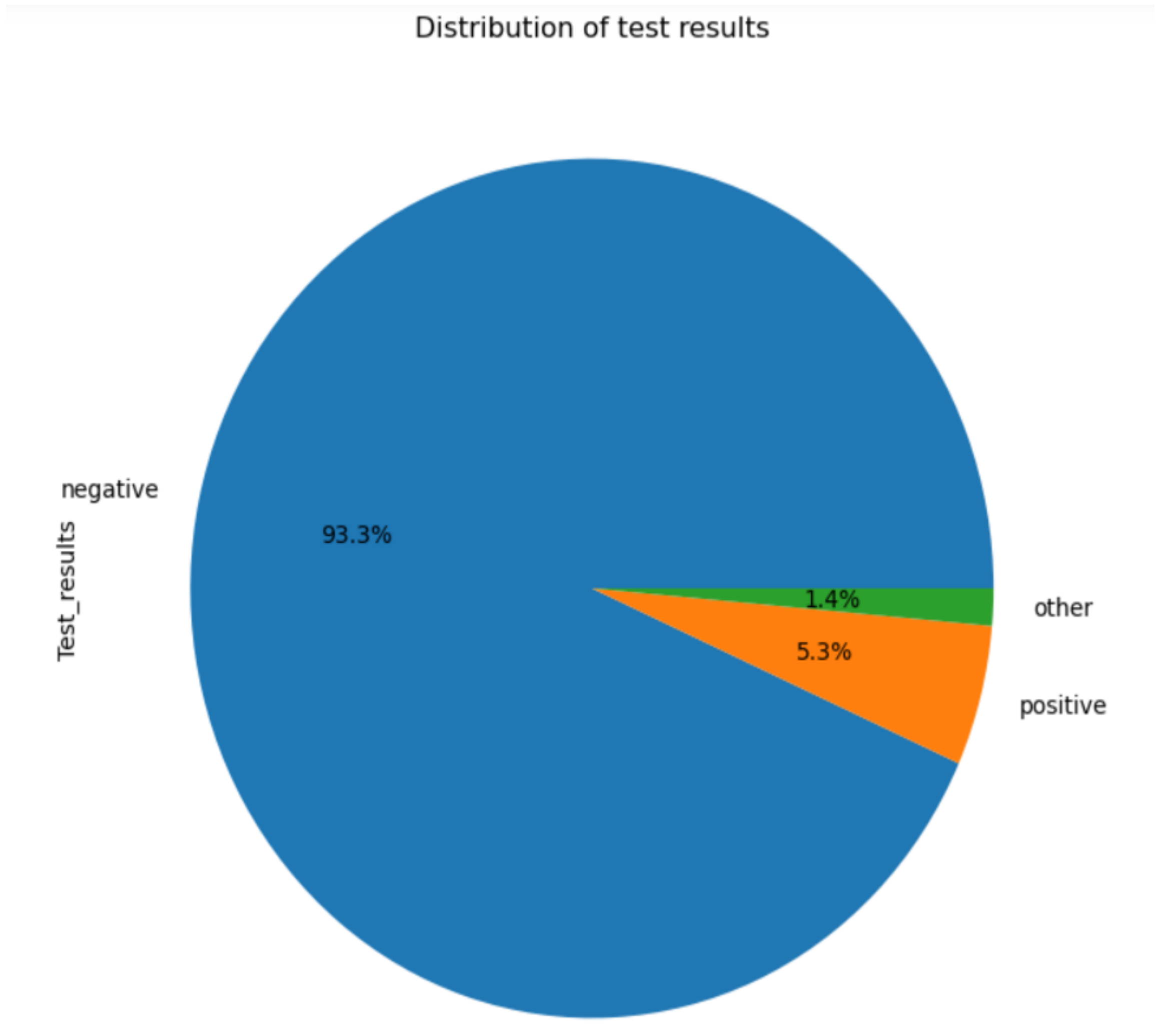
```
columns=["Id", "Test_date", "Cough", "Fever", "Sore_throat", "Shortness_of_breath", "Headache",
         "Age_60_above", "Sex", "Known_contact", "Test_results"]
corona_df = corona_db[columns]
```

## 3.15 Saving the cleaned data as a csv file to be used in sql queries

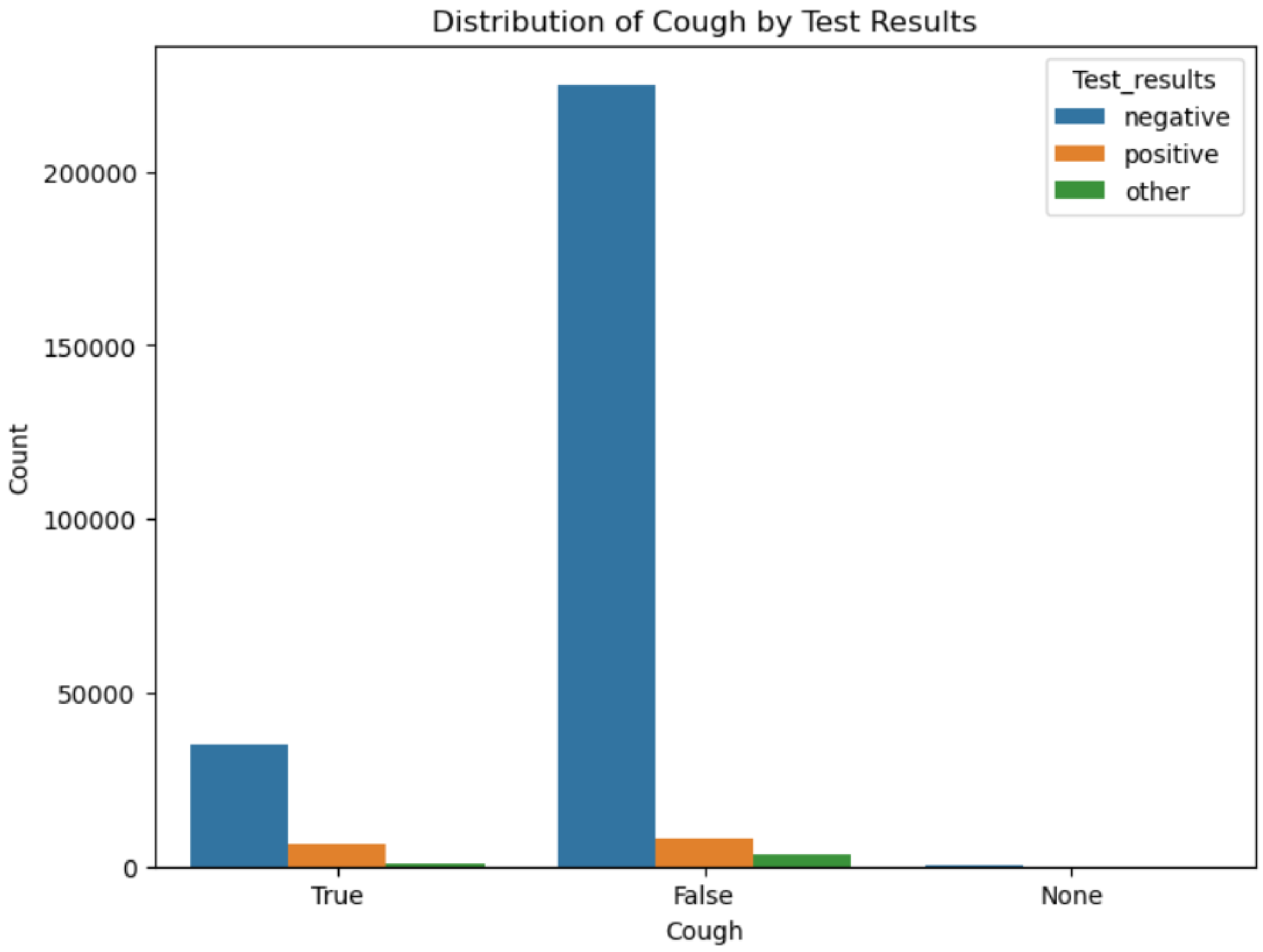
```
# Saving the cleaned data as a csv file to be used in sql queries
corona_df.to_csv('corona.csv', index=False)
```

## 3.16 Data exploration using plots

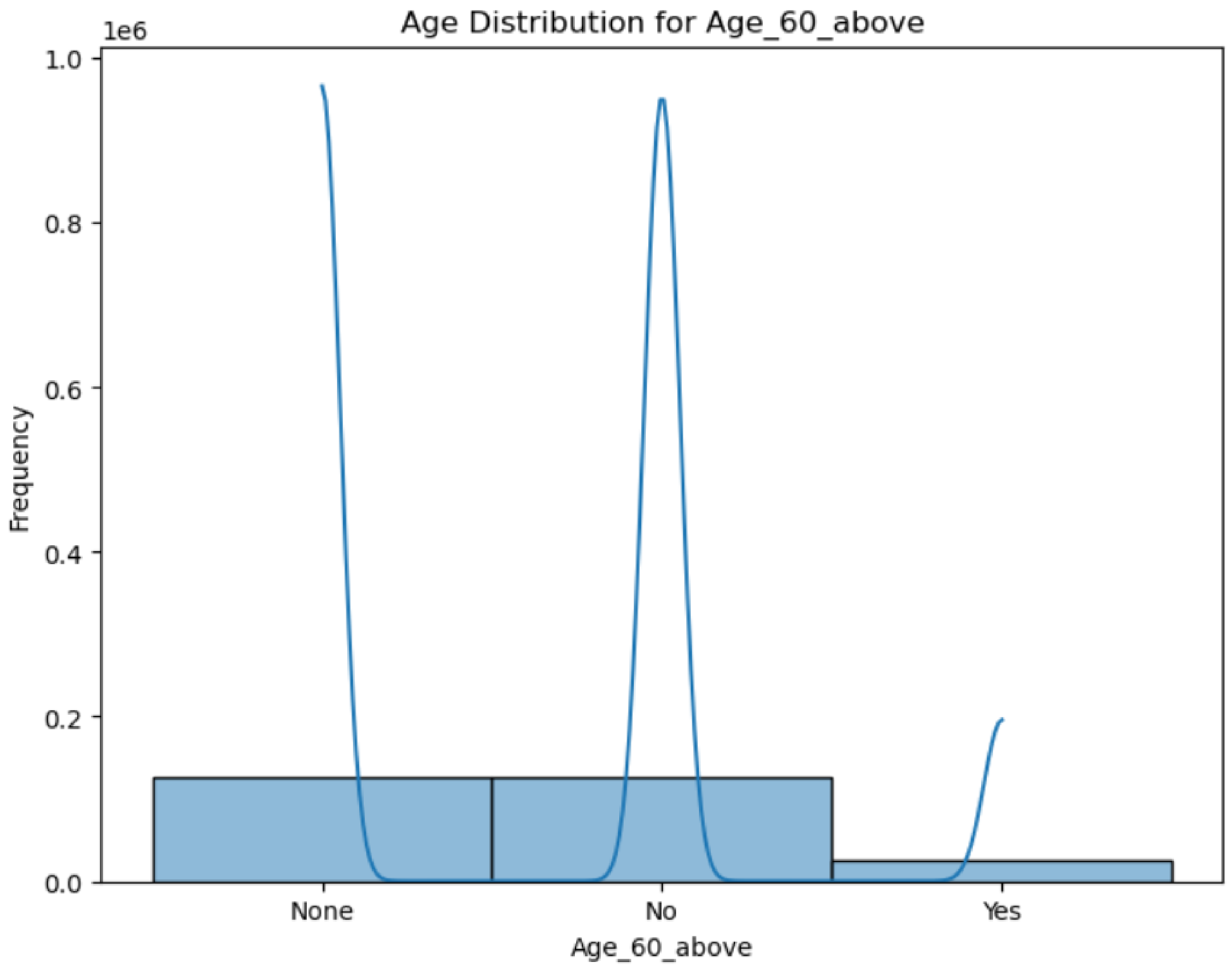
```
# Pie Chart-Sex
plt.figure(figsize=(8, 8))
corona_df['Test_results'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Distribution of test results')
plt.show()
```



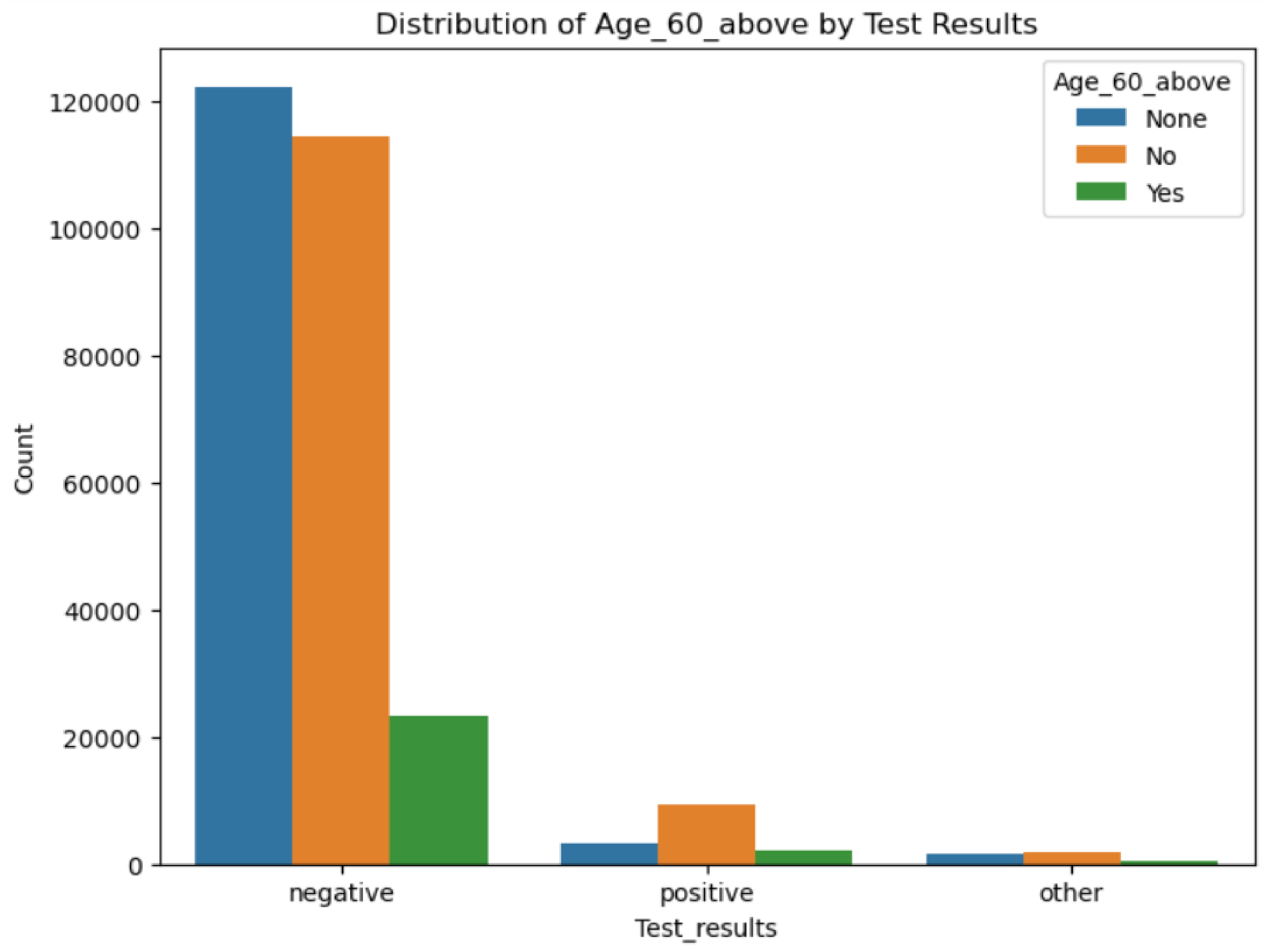
```python
# Count plot-Distribution of Cough by Test Results
plt.figure(figsize=(8, 6))
sns.countplot(data=corona_df, x='Cough', hue='Test_results')
plt.title('Distribution of Cough by Test Results')
plt.xlabel('Cough')
plt.ylabel('Count')
plt.show()
```



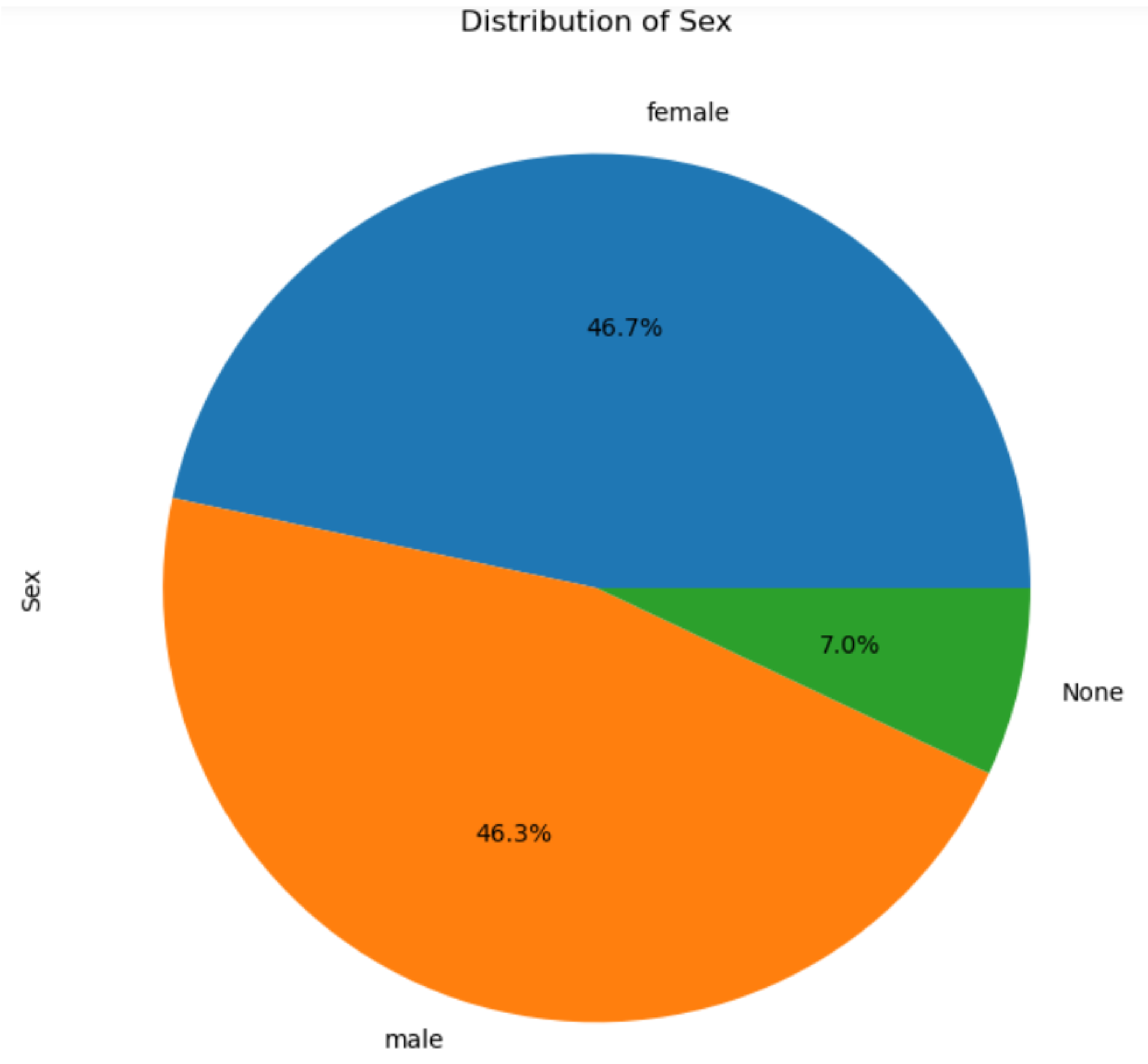
```python
# Hist plot-Age Distribution for Age_60_above
plt.figure(figsize=(8, 6))
sns.histplot(data=corona_df, x='Age_60_above', bins=10, kde=True)
plt.title('Age Distribution for Age_60_above')
plt.xlabel('Age_60_above')
plt.ylabel('Frequency')
plt.show()
```



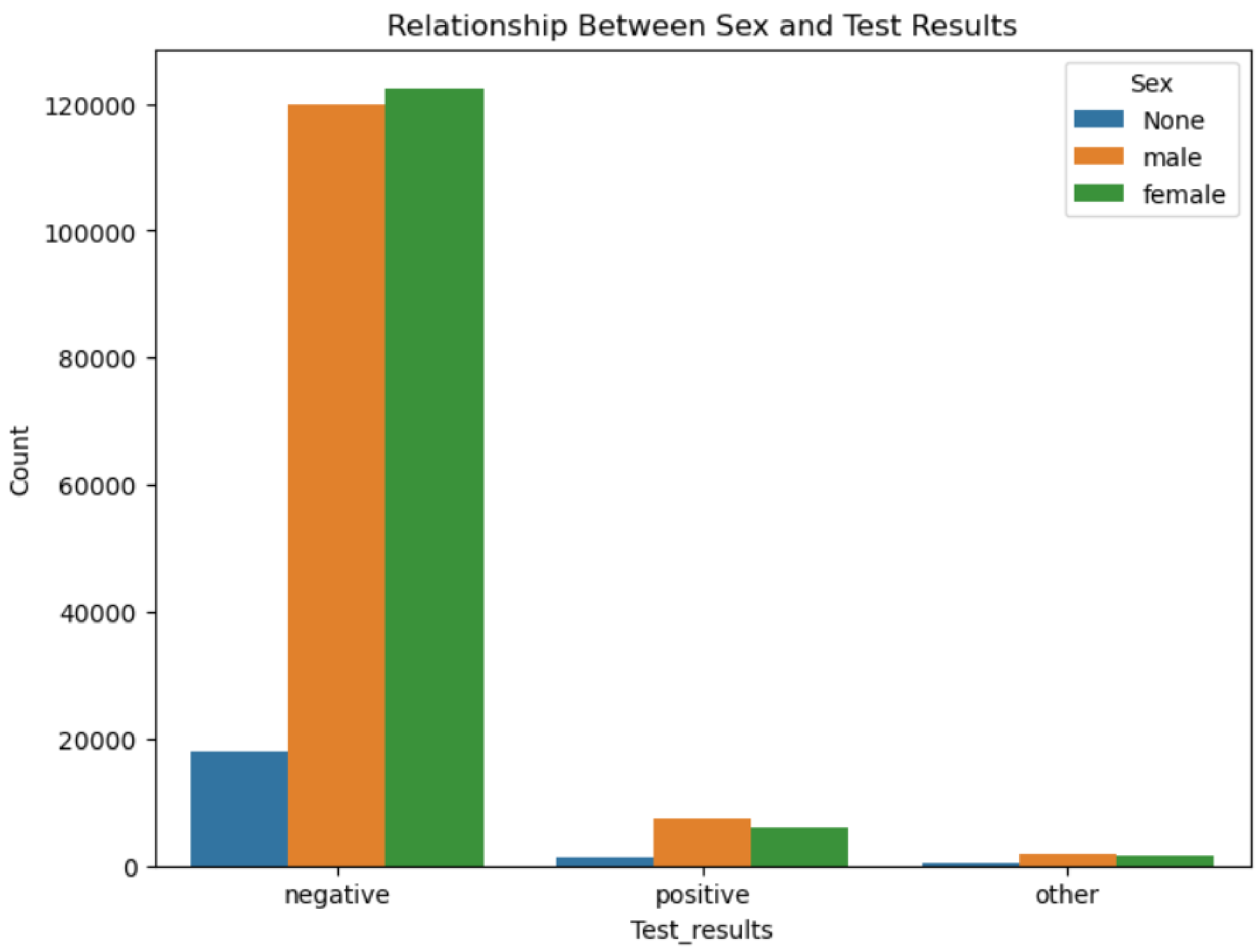
```python
# Count Plot-Age_60_above by Test_results
plt.figure(figsize=(8, 6))
sns.countplot(data=corona_df, x='Test_results', hue='Age_60_above')
plt.title('Distribution of Age_60_above by Test Results')
plt.xlabel('Test_results')
plt.ylabel('Count')
plt.show()
```



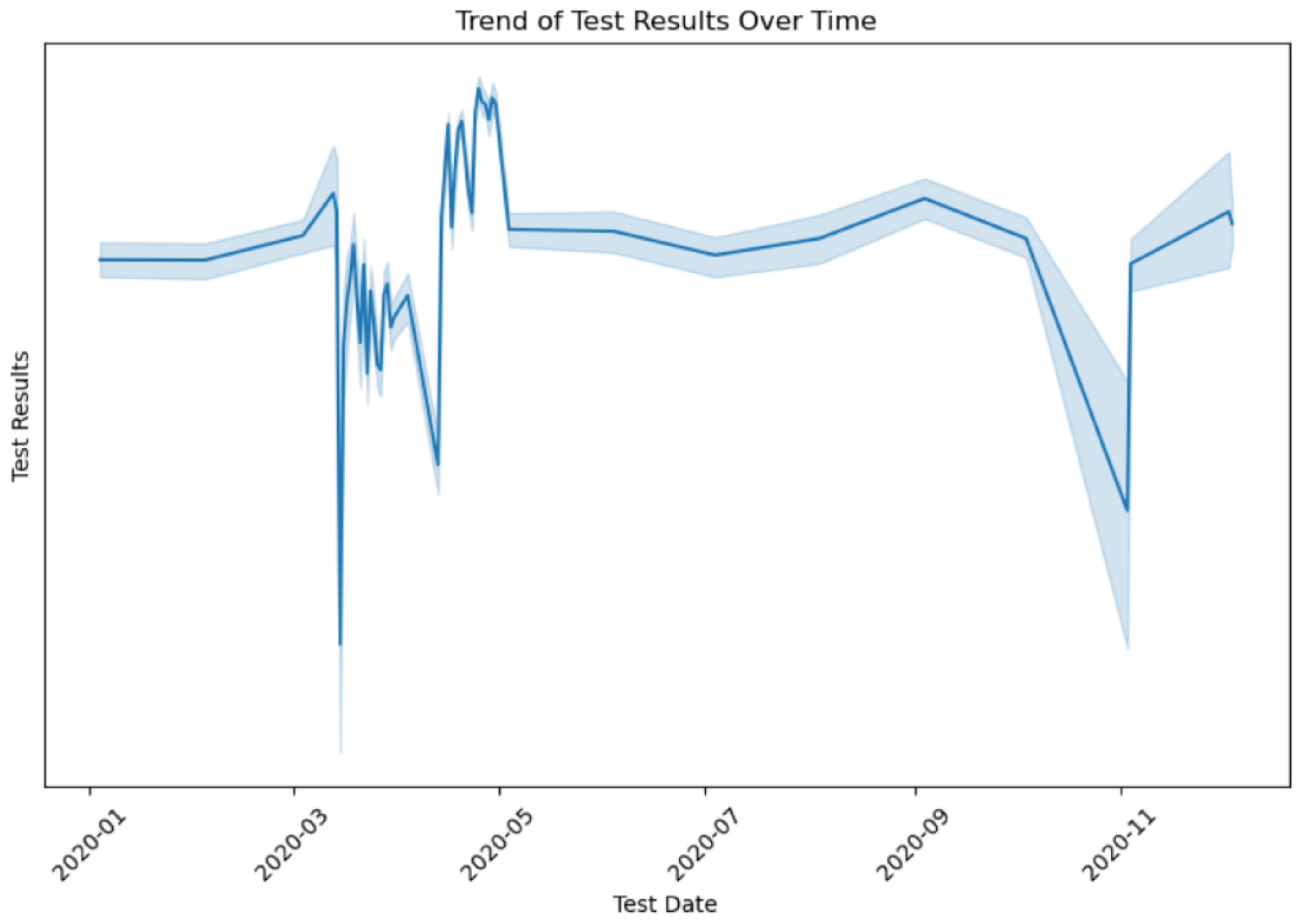
```python
# Pie Chart-Sex
plt.figure(figsize=(8, 8))
corona_df['Sex'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Distribution of Sex')
plt.show()
```



```
# Stacked Bar Chart-Sex and Test_results
plt.figure(figsize=(8, 6))
sns.countplot(data=corona_df, x='Test_results', hue='Sex')
plt.title('Relationship Between Sex and Test Results')
plt.xlabel('Test_results')
plt.ylabel('Count')
plt.legend(title='Sex', loc='upper right')
plt.show()
```



```python
# Line Chart-Test_date against Test_results
plt.figure(figsize=(10, 6))
sns.lineplot(data=corona_df, x='Test_date', y='Test_results')
plt.title('Trend of Test Results Over Time')
plt.xlabel('Test Date')
plt.ylabel('Test Results')
plt.xticks(rotation=45)
plt.show()
```



### 3.17 Dropping unwanted rows (Data cleaning)

```
x = corona_df['Test_results'] != 'other'

corona_df = corona_df[x]
```

```
corona_df['Test_results'].value_counts()
```

```
negative    260227
positive     14729
Name: Test_results, dtype: int64
```

```
corona_df['Test_results'] = corona_df['Test_results'].replace({'negative': 0, 'positive': 1})
```

## Data transformation or Data Wrangling

### 3.18 Encoding (Dummy encoding)

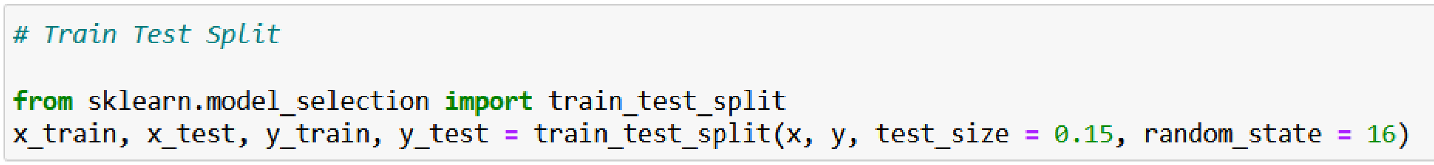
```
# Dummy encoding

covid_df = pd.get_dummies(corona_df, columns=['Cough', 'Fever', 'Sore_throat', 'Shortness_of_breath', 'Headache',
                                              'Age_60_above', 'Sex', 'Known_contact'], drop_first=True)
```

### 3.19 Train Test Split

```
# Seperating Independent and dependent variables
x = covid_df.iloc[:,:-1]
y = covid_df.iloc[:,-1]
```

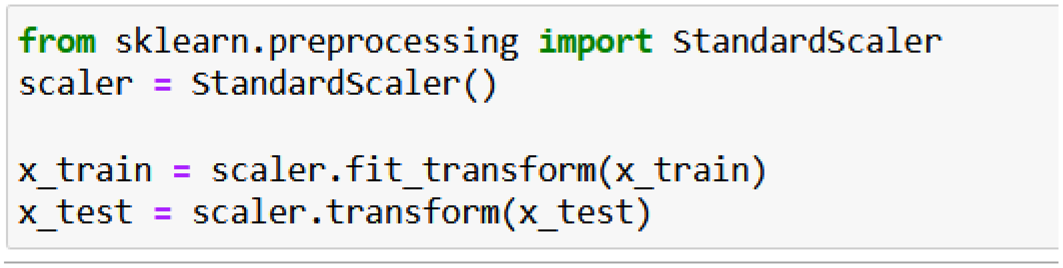
```
x.head()
```

| | Test_results | Cough_True | Cough_None | Fever_True | Fever_None | Sore_throat_True | Sore_throat_None | Shortness_of_breath_True |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| **1** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **2** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **3** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |



```python
# Train Test Split

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.15, random_state = 16)
```

### 3.20 Standardization / Feature scaling



```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

## Section 4: Machine Learning Approach

1. **Machine Learning Method**: Use various machine learning models such as logistic regression, decision trees, random forests, and support vector machines to predict COVID-19 outcomes.
2. **Justification for Model Choice**: Compare models based on metrics like accuracy, precision, recall, and F1-score. Choose the model that provides the best overall performance.
3. **Model Improvement**: Implement techniques like hyperparameter tuning, cross-validation, and feature selection to improve model accuracy.
4. **Model Comparison**: Compare at least four machine learning models and justify their selection based on cost functions and performance metrics.

## 4.1 Logistic Regression

```
# Modeling
from sklearn.linear_model import LogisticRegression
logistic_reg = LogisticRegression(random_state = 0, max_iter=1000)
logistic_reg.fit(x_train, y_train)

# Predictions
ypred_train = logistic_reg.predict(x_train)
ypred_test  = logistic_reg.predict(x_test)


# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

# Calculation of cross validation score:
from sklearn.model_selection import cross_val_score
print("Cross validation score: ", cross_val_score(logistic_reg, x,y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9160804751146711
Test Accuracy:  0.9158907962370284
Cross validation score:  0.9082434280369215
```

## 4.2 KNN

```
# K-Nearest Neighbors with default parameters

# Modeling
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(x_train, y_train)

# Prediction
ypred_train = model.predict(x_train)
ypred_test  = model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("Cross validation score: ", cross_val_score(model,x,y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9149423221742999
Test Accuracy:  0.9134419551934827
Cross validation score:  0.9078979076280935
```

## 4.2.2 Using Hyper parameter tuning to improve the model performance

```
# Hyper Parameter Tuning to improve the model's performance with best combination of hyperparameters

from sklearn.model_selection import GridSearchCV

estimator = KNeighborsClassifier()

param_grid = {'n_neighbors': list(range(1,11))}

cv_classifier = GridSearchCV(estimator, param_grid, cv=5, scoring='accuracy')

cv_classifier.fit(x_train, y_train)

cv_classifier.best_params_
```

```
{'n_neighbors': 9}
```

## 4.2.3 Applying the k value obtained from Hyper parameter tuning

```
# KNN with the best combinations
# Applying the k value obtained from Hyper parameter tuning to the KNN to get the best accuracy & CVS

# Modeling
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors = 9)     # Changed the default K value to the value obtained from HPT
model.fit(x_train, y_train)

# Prediction
ypred = model.predict(x_test)
ypred_train = model.predict(x_train)
ypred_test  = model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("Cross validation score: ", cross_val_score(model,x,y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9178176559183953
Test Accuracy:  0.9159150421879546
Cross validation score:  0.9073632755479277
```

## 4.3 SVM

```
# Modeling
from sklearn.svm import SVC
svm_model = SVC()
svm_model.fit(x_train, y_train)

# Prediction
ypred_train = svm_model.predict(x_train)
ypred_test  = svm_model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("cross validation score: ", cross_val_score(model, x, y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9220279660436776
Test Accuracy:  0.9206430026185627
cross validation score:  0.9073632755479277
```

## 4.4 Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()

dt_model.fit(x_train, y_train)
```

```
DecisionTreeClassifier()
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
# Modeling
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(random_state=16)
dt_model.fit(x_train, y_train)

# Prediction
ypred_train = dt_model.predict(x_train)
ypred_test  = dt_model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("cross validation score: ", cross_val_score(model, x, y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9223274799753542
Test Accuracy:  0.9202793133546697
cross validation score:  0.9073632755479277
```

## 4.5 Random Forest

```python
# Random forest with default parameters

# Modeling
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=16)
model.fit(x_train, y_train)

# Prediction
ypred_train = model.predict(x_train)
ypred_test  = model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("cross validation score: ", cross_val_score(model, x, y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9223274799753542
Test Accuracy:  0.9204005431093008
cross validation score:  0.9089199019722681
```

### 4.5.2 Measuring the importance of each feature in this model

```python
# Measuring the importance of each feature in this model
model.feature_importances_
```

```
array([7.75637861e-02, 4.48771550e-01, 6.37197422e-05, 1.04966066e-01,
       5.67602277e-05, 7.66291430e-03, 3.41449641e-07, 1.20679377e-03,
       1.09050986e-02, 9.39656730e-07, 3.92016700e-03, 9.96012422e-03,
       1.46618080e-06, 6.30776205e-03, 4.03184438e-03, 2.64083960e-03,
       3.05158359e-03, 3.18888243e-01])
```

### 4.6 XG Boost

```python
# Modeling
from xgboost import XGBClassifier
xgb_model = XGBClassifier()
xgb_model.fit(x_train, y_train)

# Prediction
ypred_train = xgb_model.predict(x_train)
ypred_test  = xgb_model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("cross validation score: ", cross_val_score(model, x, y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9221477716163483
Test Accuracy:  0.9204975269130056
cross validation score:  0.9089199019722681
```

## Model Selection

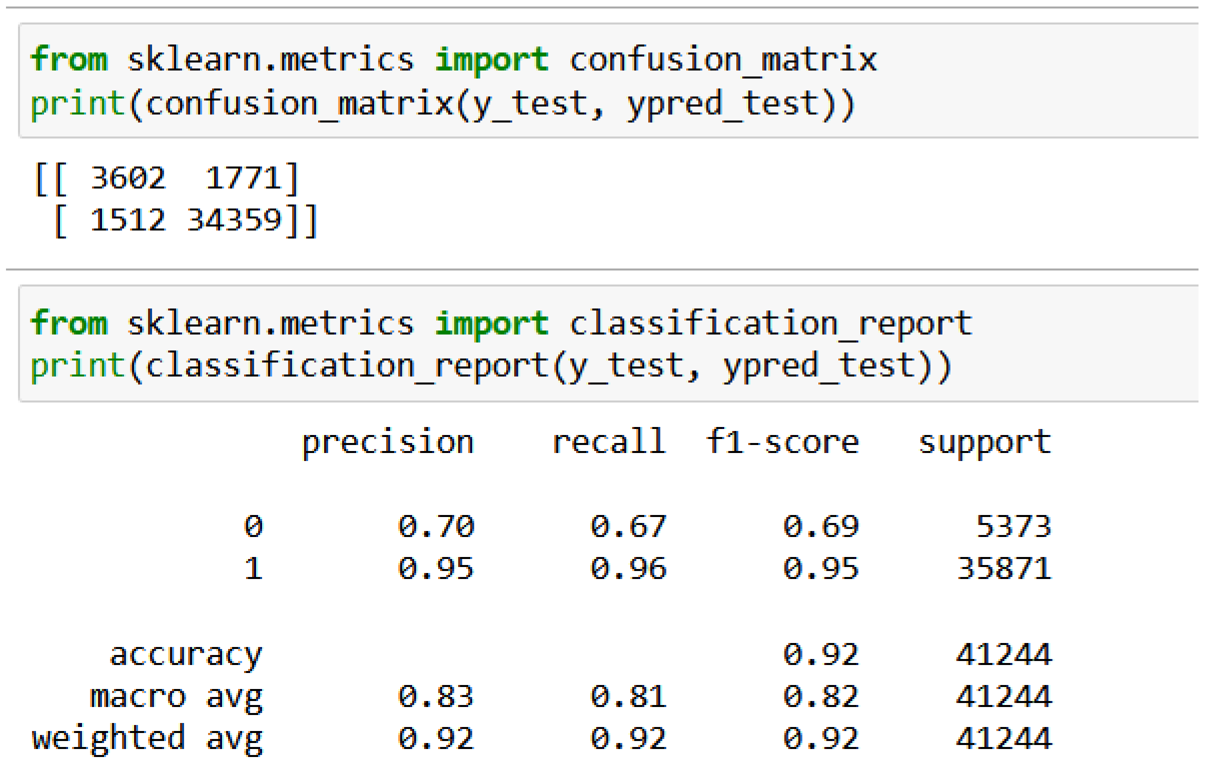
Selecting the best model on the basic of:

(a) Condition1: Test accuracy = Train accuracy = Cross validation score (+/- 5 is accaptable)

(b) Condition2: If the more than one model is satisfing the 1st condition go for the model having the highest CV

- While examing all the models Random forest and XGBoost appears to be almost equal giving high train accuracy, test accuracy and cross validation score.
- But Random forest gives little more train accuracy.
- So i opt for Random forest as the best model.

**Checking for confusion metrix and Classification report for the final model**



```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, ypred_test))
```

```
[[ 3602  1771]
 [ 1512 34359]]
```

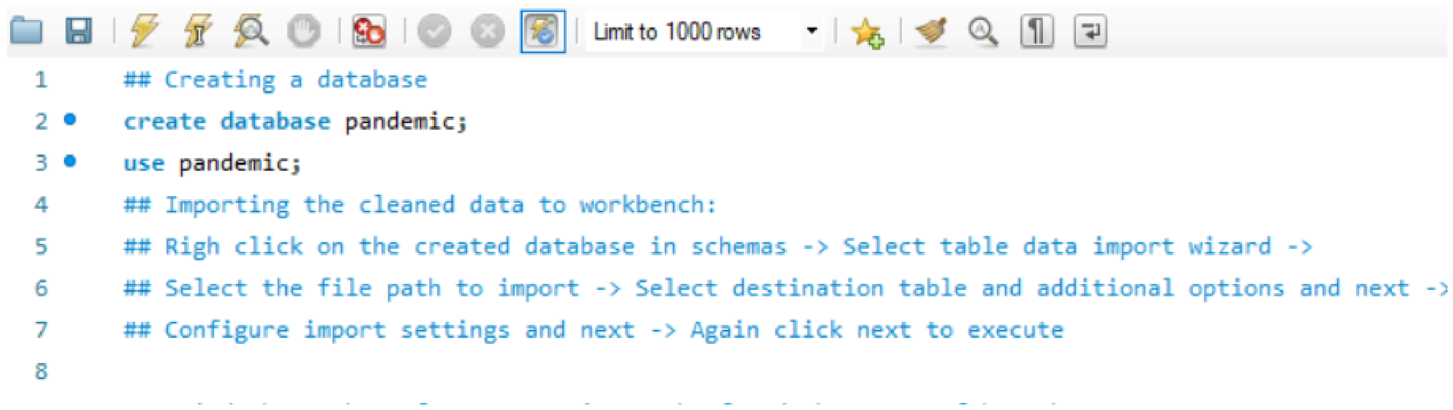
```
from sklearn.metrics import classification_report
print(classification_report(y_test, ypred_test))
```

```
              precision    recall  f1-score   support

           0       0.70      0.67      0.69      5373
           1       0.95      0.96      0.95     35871

    accuracy                           0.92     41244
   macro avg       0.83      0.81      0.82     41244
weighted avg       0.92      0.92      0.92     41244
```

## SQL Part of the Project

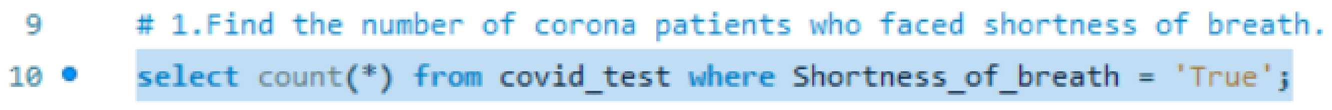
Used MySQL to perform the following queries on the cleaned data:

**Creating the database and importing the dataset:**

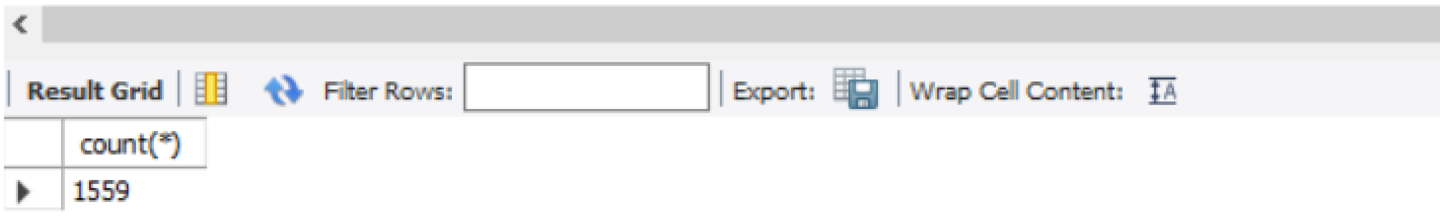


```
## Creating a database
create database pandemic;
use pandemic;
## Importing the cleaned data to workbench:
## Righ click on the created database in schemas -> Select table data import wizard ->
## Select the file path to import -> Select destination table and additional options and next ->
## Configure import settings and next -> Again click next to execute
```

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

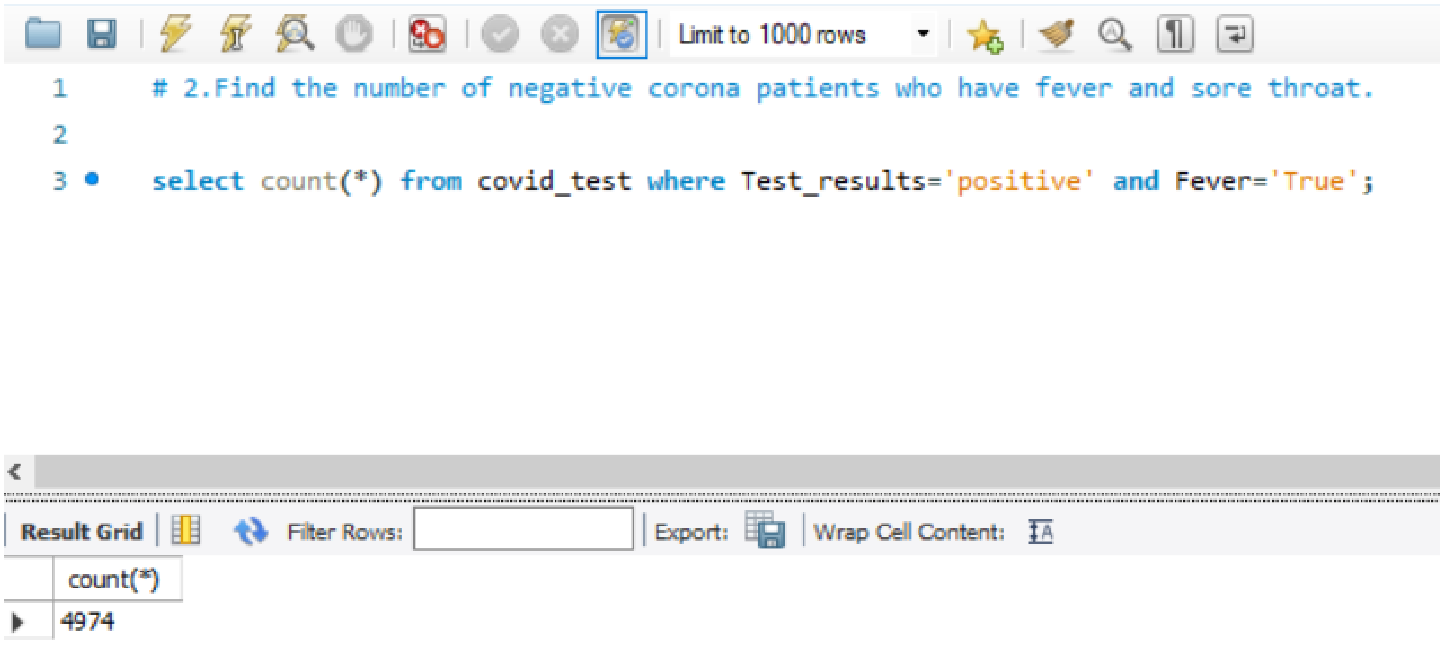


```
# 1.Find the number of corona patients who faced shortness of breath.
select count(*) from covid_test where Shortness_of_breath = 'True';
```



```
count(*)
1559
```

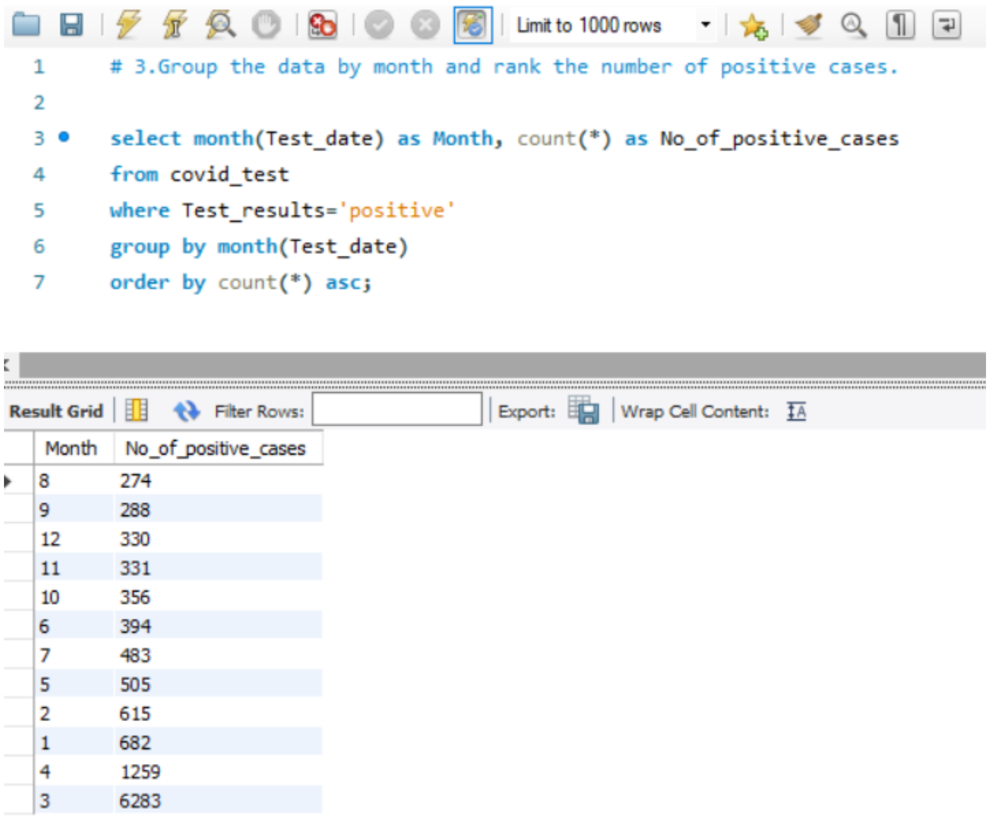
2. **Find the number of negative corona patients who have fever and sore throat.**



```
# 2.Find the number of negative corona patients who have fever and sore throat.

select count(*) from covid_test where Test_results='positive' and Fever='True';
```

```
count(*)
4974
```

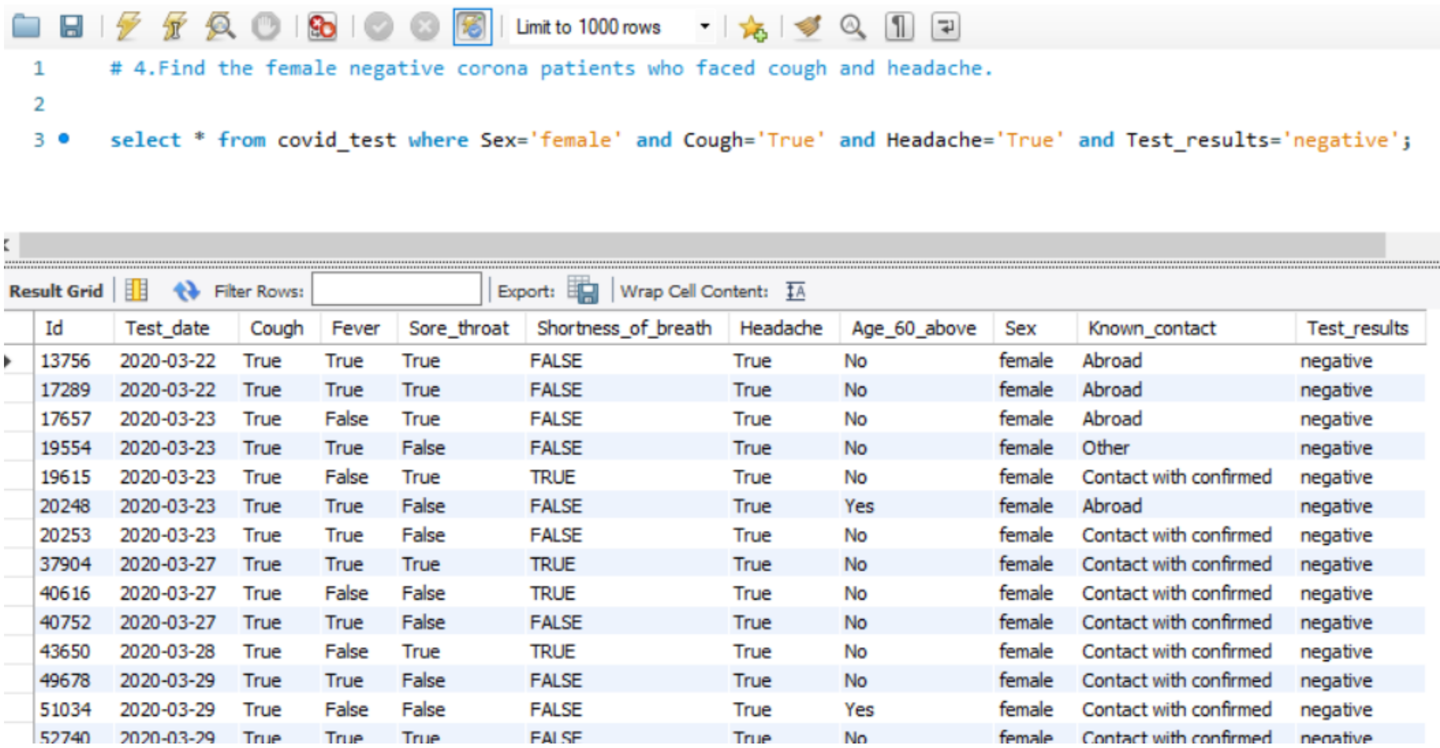
3. **Group the data by month and rank the number of positive cases.**



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

select month(Test_date) as Month, count(*) as No_of_positive_cases
from covid_test
where Test_results='positive'
group by month(Test_date)
order by count(*) asc;
```

| Month | No_of_positive_cases |
|---|---|
| 8 | 274 |
| 9 | 288 |
| 12 | 330 |
| 11 | 331 |
| 10 | 356 |
| 6 | 394 |
| 7 | 483 |
| 5 | 505 |
| 2 | 615 |
| 1 | 682 |
| 4 | 1259 |
| 3 | 6283 |

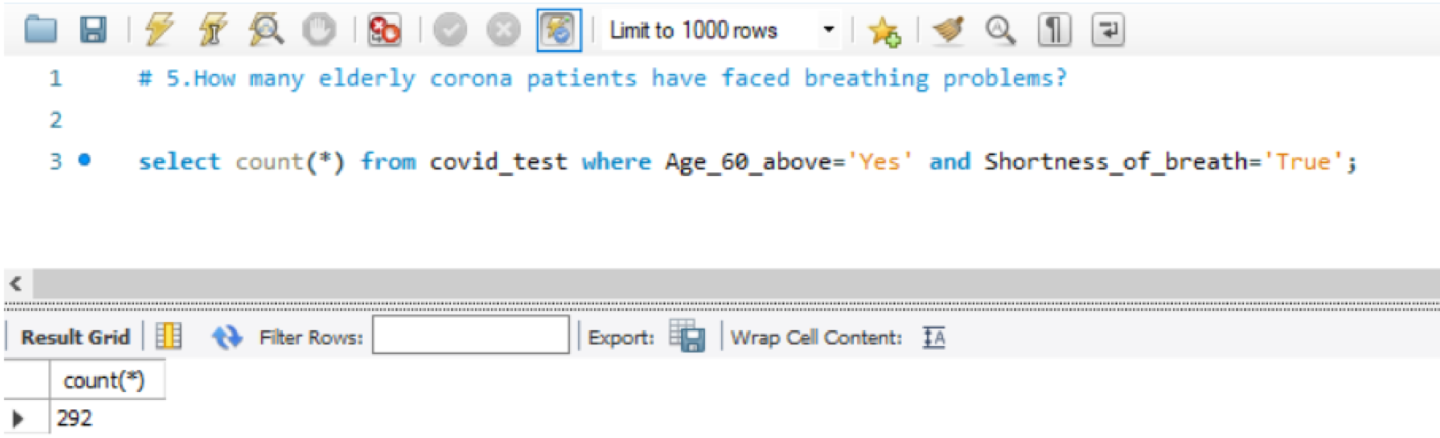
4. **Find the female negative corona patients who faced cough and headache.**



```sql
# 4.Find the female negative corona patients who faced cough and headache.

select * from covid_test where Sex='female' and Cough='True' and Headache='True' and Test_results='negative';
```

| Id | Test_date | Cough | Fever | Sore_throat | Shortness_of_breath | Headache | Age_60_above | Sex | Known_contact | Test_results |
|---|---|---|---|---|---|---|---|---|---|---|
| 13756 | 2020-03-22 | True | True | True | FALSE | True | No | female | Abroad | negative |
| 17289 | 2020-03-22 | True | True | True | FALSE | True | No | female | Abroad | negative |
| 17657 | 2020-03-23 | True | False | True | FALSE | True | No | female | Abroad | negative |
| 19554 | 2020-03-23 | True | True | False | FALSE | True | No | female | Other | negative |
| 19615 | 2020-03-23 | True | False | True | TRUE | True | No | female | Contact with confirmed | negative |
| 20248 | 2020-03-23 | True | True | False | FALSE | True | Yes | female | Abroad | negative |
| 20253 | 2020-03-23 | True | True | False | FALSE | True | No | female | Contact with confirmed | negative |
| 37904 | 2020-03-27 | True | True | True | TRUE | True | No | female | Contact with confirmed | negative |
| 40616 | 2020-03-27 | True | False | False | TRUE | True | No | female | Contact with confirmed | negative |
| 40752 | 2020-03-27 | True | True | False | FALSE | True | No | female | Contact with confirmed | negative |
| 43650 | 2020-03-28 | True | False | True | TRUE | True | No | female | Contact with confirmed | negative |
| 49678 | 2020-03-29 | True | True | False | FALSE | True | No | female | Contact with confirmed | negative |
| 51034 | 2020-03-29 | True | False | False | FALSE | True | Yes | female | Contact with confirmed | negative |
| 52740 | 2020-03-29 | True | True | True | FALSE | True | No | female | Contact with confirmed | negative |

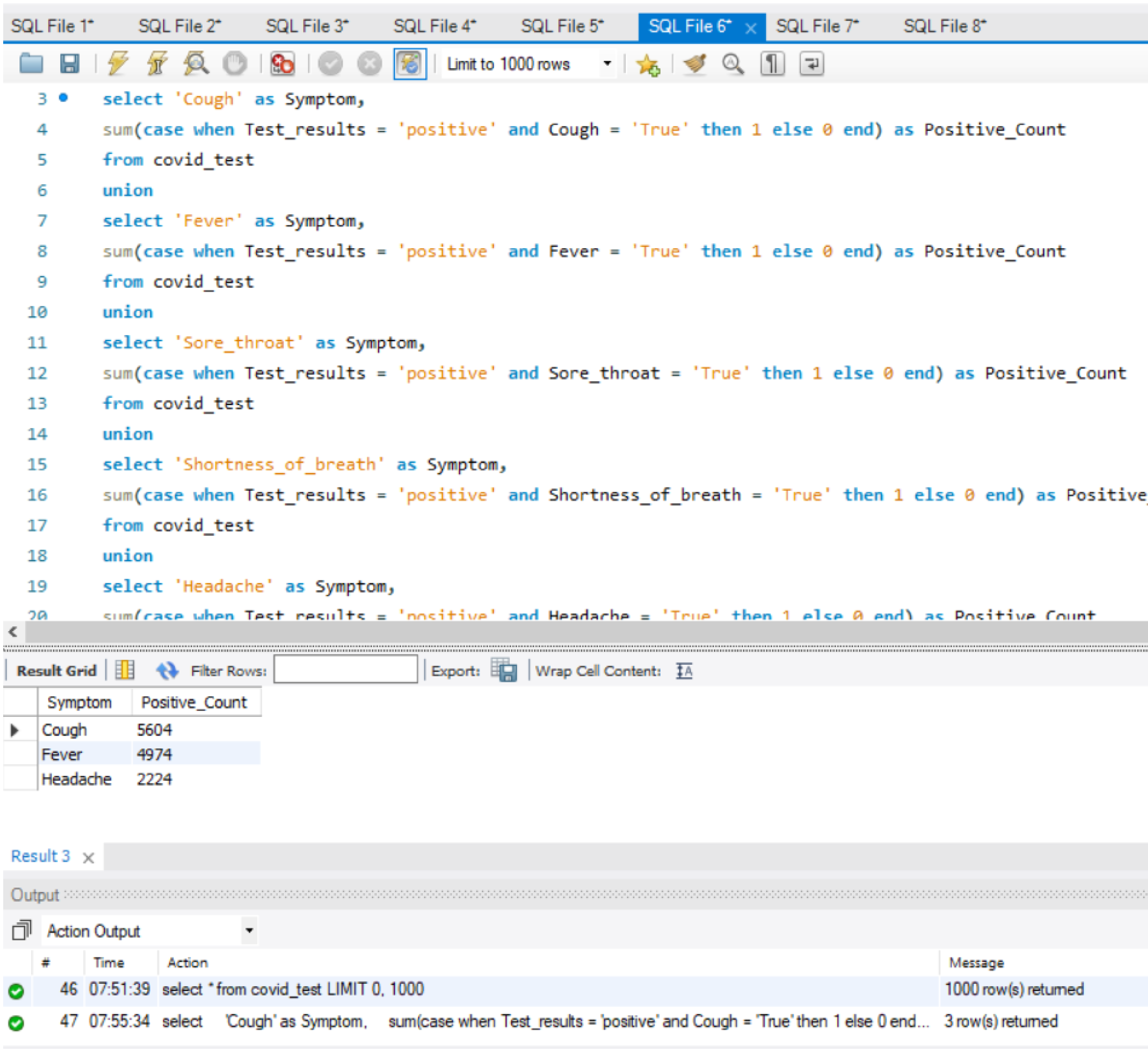
5. **How many elderly corona patients have faced breathing problems?**



```sql
# 5.How many elderly corona patients have faced breathing problems?

select count(*) from covid_test where Age_60_above='Yes' and Shortness_of_breath='True';
```

| count(*) |
|---|
| 292 |

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


SQL File 1* | SQL File 2* | SQL File 3* | SQL File 4* | SQL File 5* | SQL File 6* | SQL File 7* | SQL File 8*

Limit to 1000 rows

```sql
select 'Cough' as Symptom,
sum(case when Test_results = 'positive' and Cough = 'True' then 1 else 0 end) as Positive_Count
from covid_test
union
select 'Fever' as Symptom,
sum(case when Test_results = 'positive' and Fever = 'True' then 1 else 0 end) as Positive_Count
from covid_test
union
select 'Sore_throat' as Symptom,
sum(case when Test_results = 'positive' and Sore_throat = 'True' then 1 else 0 end) as Positive_Count
from covid_test
union
select 'Shortness_of_breath' as Symptom,
sum(case when Test_results = 'positive' and Shortness_of_breath = 'True' then 1 else 0 end) as Positive
from covid_test
union
select 'Headache' as Symptom,
sum(case when Test_results = 'positive' and Headache = 'True' then 1 else 0 end) as Positive_Count
```

| Symptom | Positive_Count |
|---|---|
| Cough | 5604 |
| Fever | 4974 |
| Headache | 2224 |

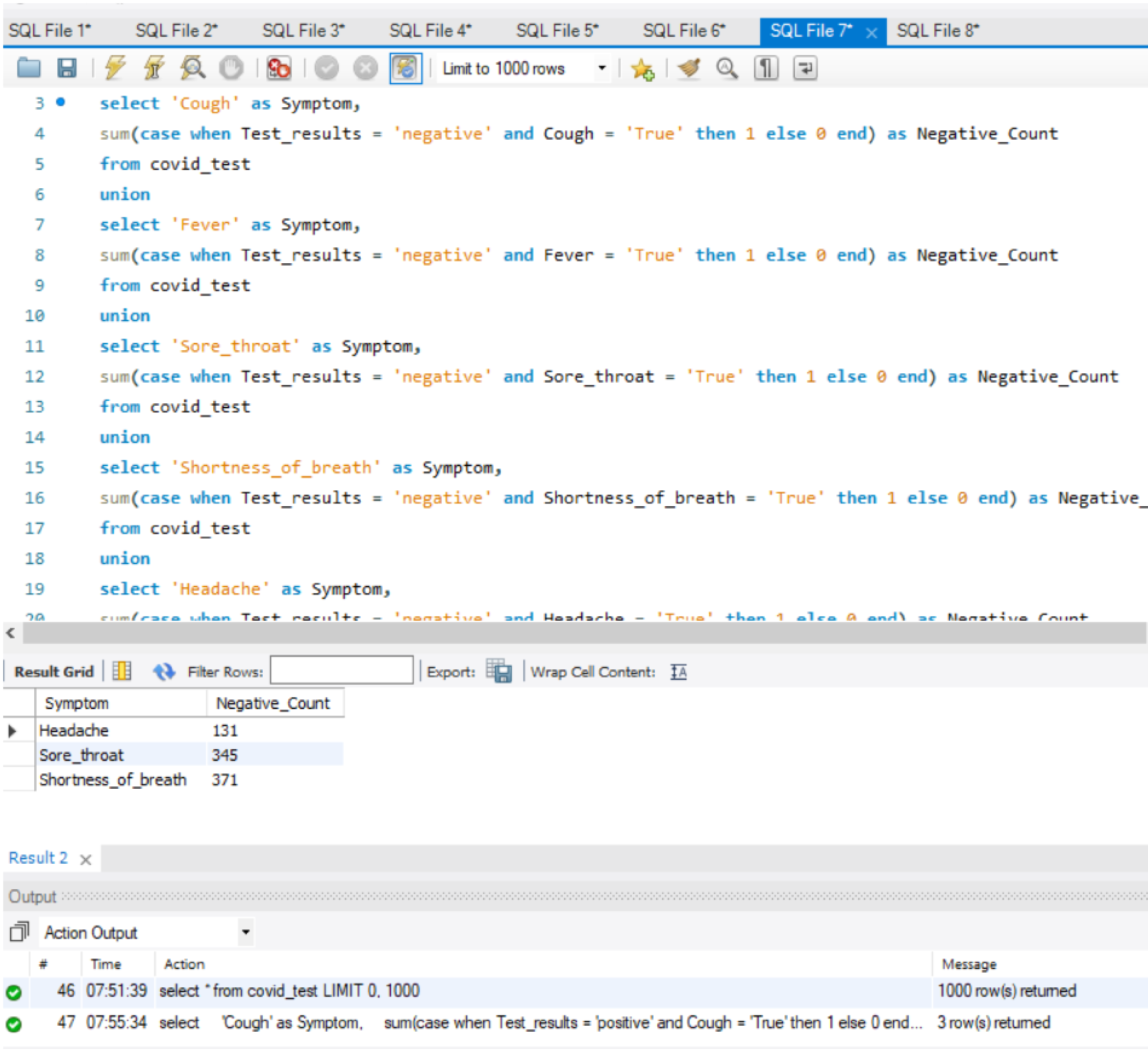
Result 3

Output

Action Output

```
#   Time      Action                                                                                    Message
46  07:51:39  select * from covid_test LIMIT 0, 1000                                                    1000 row(s) returned
47  07:55:34  select  'Cough' as Symptom,  sum(case when Test_results = 'positive' and Cough = 'True' then 1 else 0 end...  3 row(s) returned
```

7. **Which symptom was less common among COVID negative people?**

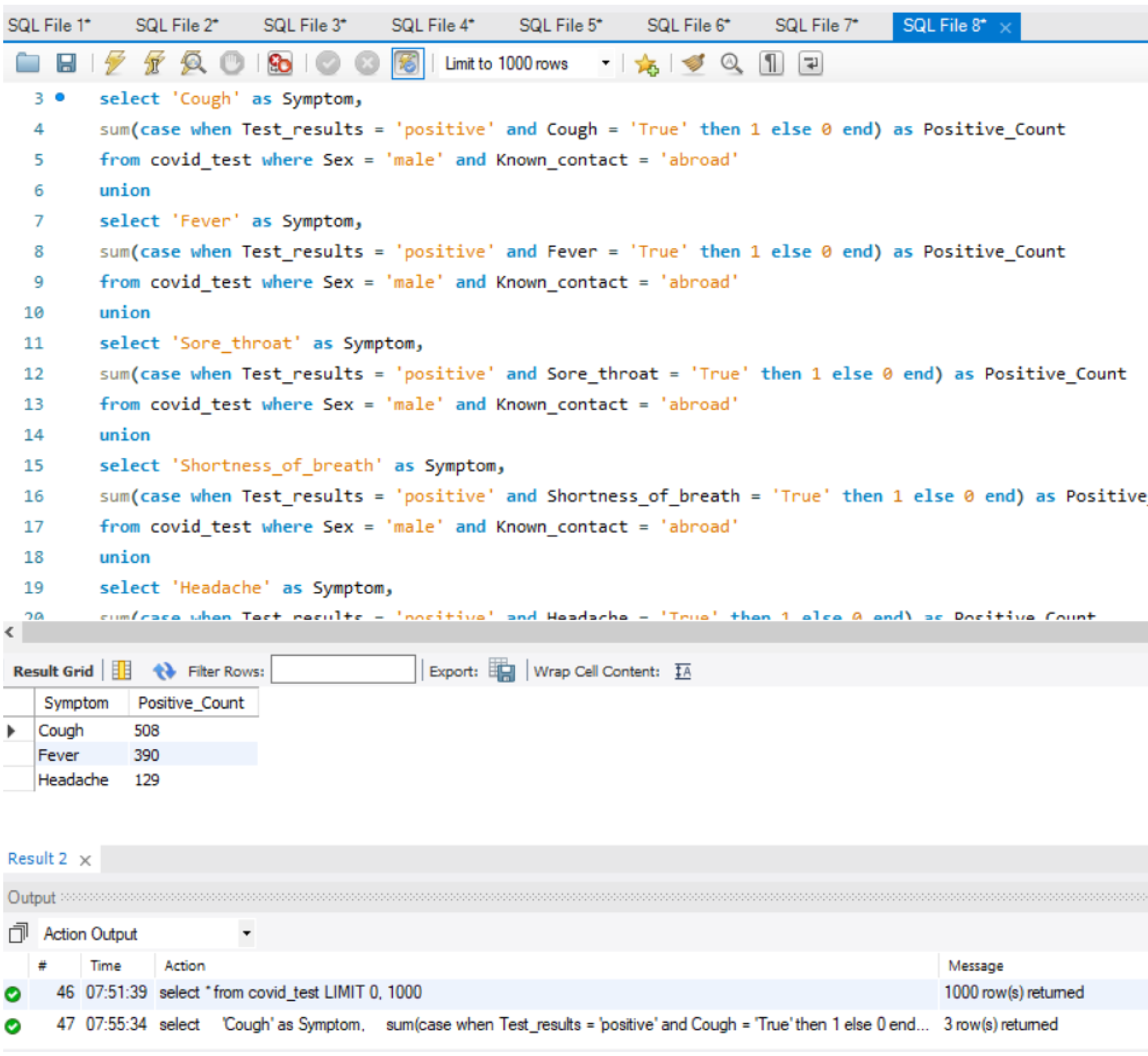


```sql
select 'Cough' as Symptom,
sum(case when Test_results = 'negative' and Cough = 'True' then 1 else 0 end) as Negative_Count
from covid_test
union
select 'Fever' as Symptom,
sum(case when Test_results = 'negative' and Fever = 'True' then 1 else 0 end) as Negative_Count
from covid_test
union
select 'Sore_throat' as Symptom,
sum(case when Test_results = 'negative' and Sore_throat = 'True' then 1 else 0 end) as Negative_Count
from covid_test
union
select 'Shortness_of_breath' as Symptom,
sum(case when Test_results = 'negative' and Shortness_of_breath = 'True' then 1 else 0 end) as Negative_
from covid_test
union
select 'Headache' as Symptom,
sum(case when Test_results = 'negative' and Headache = 'True' then 1 else 0 end) as Negative_Count
```

| Symptom | Negative_Count |
|---|---|
| Headache | 131 |
| Sore_throat | 345 |
| Shortness_of_breath | 371 |

```
#   Time      Action                                                                                Message
46  07:51:39  select * from covid_test LIMIT 0, 1000                                                1000 row(s) returned
47  07:55:34  select  'Cough' as Symptom,  sum(case when Test_results = 'positive' and Cough = 'True' then 1 else 0 end...  3 row(s) returned
```

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



SQL File 1* SQL File 2* SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8*

Limit to 1000 rows

```
select 'Cough' as Symptom,
sum(case when Test_results = 'positive' and Cough = 'True' then 1 else 0 end) as Positive_Count
from covid_test where Sex = 'male' and Known_contact = 'abroad'
union
select 'Fever' as Symptom,
sum(case when Test_results = 'positive' and Fever = 'True' then 1 else 0 end) as Positive_Count
from covid_test where Sex = 'male' and Known_contact = 'abroad'
union
select 'Sore_throat' as Symptom,
sum(case when Test_results = 'positive' and Sore_throat = 'True' then 1 else 0 end) as Positive_Count
from covid_test where Sex = 'male' and Known_contact = 'abroad'
union
select 'Shortness_of_breath' as Symptom,
sum(case when Test_results = 'positive' and Shortness_of_breath = 'True' then 1 else 0 end) as Positive
from covid_test where Sex = 'male' and Known_contact = 'abroad'
union
select 'Headache' as Symptom,
sum(case when Test_results = 'positive' and Headache = 'True' then 1 else 0 end) as Positive_Count
```

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

| Symptom | Positive_Count |
|---|---|
| Cough | 508 |
| Fever | 390 |
| Headache | 129 |

Result 2

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 46 | 07:51:39 | select * from covid_test LIMIT 0, 1000 | 1000 row(s) returned |
| 47 | 07:55:34 | select 'Cough' as Symptom, sum(case when Test_results = 'positive' and Cough = 'True' then 1 else 0 end... | 3 row(s) returned |