Title: Covid-19-detection-using-Machine-Learning

Group Member Names : Bhavin Patel (200584974), Kunj Kansara (200584153)

INTRODUCTION:

- In December 2019, the novel coronavirus appeared in the city of Wuhan in China [1] and was reported to the World Health Organization (WHO) on 31 December 2019. The virus posed a global threat and was named COVID-19 by the WHO on the 11th. February 2020. W.H.O declared the outbreak a public health emergency [2] and stated the following; “the virus is spread through the respiratory tract when a healthy person comes in contact with an infected person”. An infected person shows symptoms within 2-14 days. According to W.H.O the symptoms and signs of moderate to severe conditions are dry cough, fatigue and fever while in severe cases dyspnea, fever and fatigue may occur. People with other illnesses such as asthma, diabetes, and heart disease are at greater risk of contracting the virus and may become seriously ill. A system which can be used to detect the virus has become necessary due to the rapid spread of the virus, killing hundreds of thousands of people. Machine learning classification algorithms, data sets and machine learning software are essential tools for designing the COVID-19 predictive model.

AIM :

- This project aims to compare different machine learning algorithms like K-nearest neighbors, Random forest and Naive Bayes with respect to their accuracies and then use the best one among them to develop a system which predicts whether a person has COVID or not using the data provided to the model.

Github Repo: https://github.com/Bhuvaneswar005/Covid-19-detection-using-Machine-Learning

---

DESCRIPTION OF PAPER:

- The research looks at how machine learning methods can be used to detect COVID-19 early. The report, authored by Bhuvaneswar, Harshitha K, and Sahana of the National Institute of Technology Karnataka in India, underlines the vital need for efficient detection systems in the middle of the global pandemic. Using datasets from Kaggle, the researchers examine the effectiveness of algorithms including Logistic Regression, K Nearest Neighbors, and Random Forest. The study's goal is to create a prediction model that can accurately detect COVID-19 patients based on symptoms and other clinical markers using extensive data preparation, exploratory analysis, and hyperparameter tuning. The results show encouraging accuracy rates, with Random Forest emerging as the most successful algorithm. The study's findings have important implications for both medical practitioners and patients, potentially allowing for more rapid interventions and risk assessments. This study adds to the ongoing efforts to fight the spread of COVID-19 by utilizing machine learning approaches for early detection and diagnosis.

PROBLEM STATEMENT :

- The rapid spread of COVID-19 has presented a huge challenge to global healthcare systems, mandating the development of efficient detection tools to prevent transmission and ensure timely intervention. Traditional diagnostic approaches may have limited scalability and speed, emphasizing the need for novel solutions that take advantage of technological improvements.

CONTEXT OF THE PROBLEM:

- In response to the COVID-19 epidemic, researchers from India's National Institute of Technology Karnataka conducted a study to investigate the potential of machine learning algorithms for early virus identification. Using accessible datasets and advanced analytics approaches, the researchers want to create a predictive model capable of properly identifying COVID-19 patients based on important clinical signs and symptoms.

SOLUTION:

- The proposed solution takes a comprehensive approach, including data collection, preprocessing, exploratory analysis, and algorithm selection. The researchers use machine learning methods such as Logistic Regression, K Nearest Neighbors, and Random Forest to analyze the performance of each strategy and determine the best effective approach for COVID-19 identification. The study's goal is to help develop dependable and scalable techniques for combatting the pandemic through rigorous experimentation and model improvement.

Background

Reference	Explanation	Dataset/Input	Weakness
Wu et al. (2020)	Provides background on the emergence of COVID-19, its global impact, and transmission dynamics.	Kaggle dataset containing COVID-19 related data, including symptoms, demographic information, and diagnostic outcomes.	Dataset may have limitations in terms of completeness and representativeness, potentially affecting the generalizability of the findings.
Medscape Medical News	Contextualizes the public health emergency declared by the World Health Organization (WHO) in response to the COVID-19 outbreak.	COVID-19 dataset procured from open-access sources, including research institutions and healthcare facilities.	Lack of standardization in data collection methods across different sources may introduce inconsistencies and biases in the dataset.
Batista et al. (2020)	Discusses the application of machine learning algorithms, specifically support vector machines (SVM), in COVID-19 diagnosis prediction based on clinical data.	Dataset containing clinical features, laboratory results, and COVID-19 diagnostic outcomes from Hospital Israelita Albert Einstein (HIAE) in Sao Paulo, Brazil.	Dependency on a single dataset from a specific geographical location may limit the model's generalizability to diverse populations and healthcare settings.
Mondal et al. (2020)	Explores data analytics approaches for COVID-19 diagnosis and prediction using logistic regression, multilayer perceptron (MLP), and XGBoost algorithms.	Dataset derived from a Brazilian hospital, including clinical variables, patient demographics, and COVID-19 test results.	Limited discussion on potential biases or confounding factors in the dataset, raising questions about the robustness of the findings.
Goodman-Meza et al. (2020)	Investigates the development of machine learning algorithms to increase COVID-19 inpatient diagnostic capacity, focusing on demographic and clinical features.	UCLA Health System dataset comprising emergency room visits, laboratory tests, and COVID-19 testing outcomes.	Potential challenges in data preprocessing and feature selection due to the complexity and heterogeneity of healthcare data.
Schwab et al. (2020)	Studies clinical predictive models for COVID-19, aiming to improve risk assessment and management strategies.	Dataset containing clinical parameters, comorbidities, and COVID-19 outcomes from multiple healthcare institutions.	Limited discussion on the interpretability of the machine learning models and their implications for clinical decision-making.
Sun et al. (2020)	Investigates epidemiological and clinical predictors of COVID-19, including machine learning-based prediction of ICU/semi-ICU needs.	Clinical dataset from diverse patient populations, encompassing demographics, medical history, symptoms, and COVID-19 severity indicators.	Potential biases introduced by retrospective data collection and reliance on electronic health records (EHR) for variable extraction.
Meng et al. (2020)	Explores the development and utilization of an intelligent application for aiding COVID-19 diagnosis, integrating machine learning algorithms with clinical decision support systems.	Dataset comprising patient-reported symptoms, laboratory tests, imaging findings, and COVID-19 diagnostic outcomes.	Limited discussion on the scalability and deployment challenges of integrating machine learning models into clinical workflows.

Implement paper code :

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import time
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report,confusion_matrix, roc_auc_score, mean_squared_error,r2_score, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import tree
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
```

In [2]:
```
covid_data = pd.read_csv("Covid Dataset.csv")
```

In [3]:
```
covid_data
```

Out[3]:

	Breathing Problem	Fever	Dry Cough	Sore throat	Running Nose	Asthma	Chronic Lung Disease	Headache	Heart Disease	Diabetes	...	Fatigue	Gastrointestinal	Abroad travel	Contact with COVID Patient	Attended Large Gathering	Visited Public Exposed Places	Family working in Public Exposed Places	Wearing Masks	Sanitization from Market	COVID-19
0	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	...	Yes	Yes	No	Yes	No	Yes	Yes	No	No	Yes
1	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	...	Yes	No	No	No	Yes	Yes	No	No	No	Yes
2	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	...	Yes	Yes	Yes	No	No	No	No	No	No	Yes
3	Yes	Yes	Yes	No	No	Yes	No	No	Yes	Yes	...	No	No	Yes	No	Yes	Yes	No	No	No	Yes
4	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	...	No	Yes	No	Yes	No	Yes	No	No	No	Yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5429	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No	No	...	Yes	Yes	No	No	No	No	No	No	No	Yes
5430	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	...	Yes	No	No	No	No	No	No	No	No	Yes
5431	Yes	Yes	Yes	No	No	No	No	No	Yes	No	...	No	No	No	No	No	No	No	No	No	No
5432	Yes	Yes	Yes	No	Yes	No	No	Yes	Yes	No	...	No	No	No	No	No	No	No	No	No	No
5433	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	...	Yes	No	No	No	No	No	No	No	No	No

5434 rows × 21 columns

In [4]:
```
covid_data.shape
```

Out[4]:
```
(5434, 21)
```

In [5]:
```
covid_data.columns
```

Out[5]:
```
Index(['Breathing Problem', 'Fever', 'Dry Cough', 'Sore throat',
       'Running Nose', 'Asthma', 'Chronic Lung Disease', 'Headache',
       'Heart Disease', 'Diabetes', 'Hyper Tension', 'Fatigue ',
       'Gastrointestinal ', 'Abroad travel', 'Contact with COVID Patient',
       'Attended Large Gathering', 'Visited Public Exposed Places',
       'Family working in Public Exposed Places', 'Wearing Masks',
       'Sanitization from Market', 'COVID-19'],
      dtype='object')
```

In [6]:
```
covid_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5434 entries, 0 to 5433
Data columns (total 21 columns):
 #   Column                                Non-Null Count  Dtype 
---  ------                                --------------  ----- 
 0   Breathing Problem                     5434 non-null   object
 1   Fever                                 5434 non-null   object
 2   Dry Cough                             5434 non-null   object
 3   Sore throat                           5434 non-null   object
 4   Running Nose                          5434 non-null   object
 5   Asthma                                5434 non-null   object
 6   Chronic Lung Disease                  5434 non-null   object
 7   Headache                              5434 non-null   object
 8   Heart Disease                         5434 non-null   object
 9   Diabetes                              5434 non-null   object
 10  Hyper Tension                         5434 non-null   object
 11  Fatigue                               5434 non-null   object
 12  Gastrointestinal                      5434 non-null   object
 13  Abroad travel                         5434 non-null   object
 14  Contact with COVID Patient            5434 non-null   object
 15  Attended Large Gathering              5434 non-null   object
 16  Visited Public Exposed Places         5434 non-null   object
 17  Family working in Public Exposed Places  5434 non-null   object
 18  Wearing Masks                         5434 non-null   object
 19  Sanitization from Market              5434 non-null   object
 20  COVID-19                              5434 non-null   object
dtypes: object(21)
memory usage: 891.6+ KB
```

In [7]:
```
covid_data.describe().T
```

Out[7]:

	count	unique	top	freq
Breathing Problem	5434	2	Yes	3620
Fever	5434	2	Yes	4273
Dry Cough	5434	2	Yes	4307
Sore throat	5434	2	Yes	3953
Running Nose	5434	2	Yes	2952
Asthma	5434	2	No	2920
Chronic Lung Disease	5434	2	No	2869
Headache	5434	2	Yes	2736
Heart Disease	5434	2	No	2911
Diabetes	5434	2	No	2846
Hyper Tension	5434	2	No	2771
Fatigue	5434	2	Yes	2821
Gastrointestinal	5434	2	No	2883
Abroad travel	5434	2	No	2983
Contact with COVID Patient	5434	2	Yes	2726
Attended Large Gathering	5434	2	No	2924
Visited Public Exposed Places	5434	2	Yes	2820
Family working in Public Exposed Places	5434	2	No	3172
Wearing Masks	5434	1	No	5434
Sanitization from Market	5434	1	No	5434
COVID-19	5434	2	Yes	4383

In [8]:
```
covid_data.head()
```

Out[8]:

	Breathing Problem	Fever	Dry Cough	Sore throat	Running Nose	Asthma	Chronic Lung Disease	Headache	Heart Disease	Diabetes	...	Fatigue	Gastrointestinal	Abroad travel	Contact with COVID Patient	Attended Large Gathering	Visited Public Exposed Places	Family working in Public Exposed Places	Wearing Masks	Sanitization from Market	COVID-19
0	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	...	Yes	Yes	No	Yes	No	Yes	Yes	No	No	Yes
1	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	...	Yes	No	No	No	Yes	Yes	No	No	No	Yes
2	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	...	Yes	Yes	Yes	No	No	No	No	No	No	Yes
3	Yes	Yes	Yes	No	No	Yes	No	No	Yes	Yes	...	No	No	Yes	No	Yes	Yes	No	No	No	Yes
4	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	...	No	Yes	No	Yes	No	Yes	No	No	No	Yes

5 rows × 21 columns

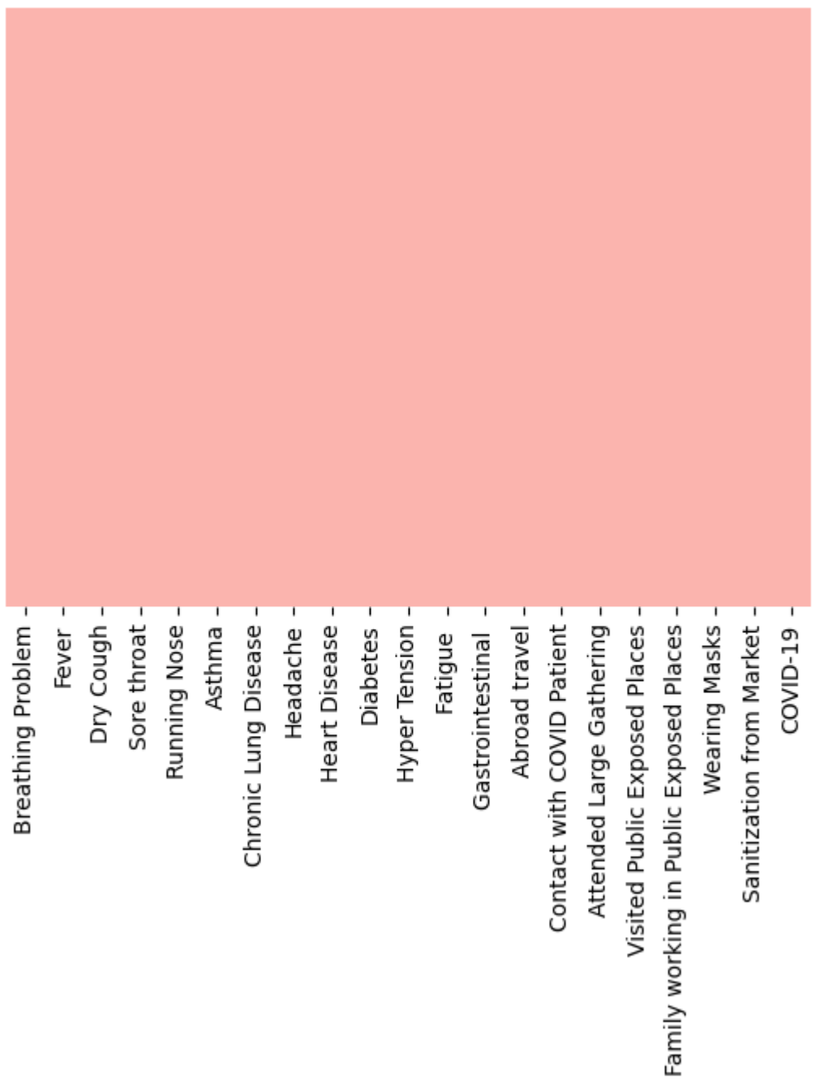
In [9]:
```
# create a table with data missing
missing_values=covid_data.isnull().sum() # missing values

percent_missing = covid_data.isnull().sum()/covid_data.shape[0]*100 # missing value %

value = {
    'missing_values ':missing_values,
    'percent_missing %':percent_missing
}
frame=pd.DataFrame(value)
frame
```

Out[9]:

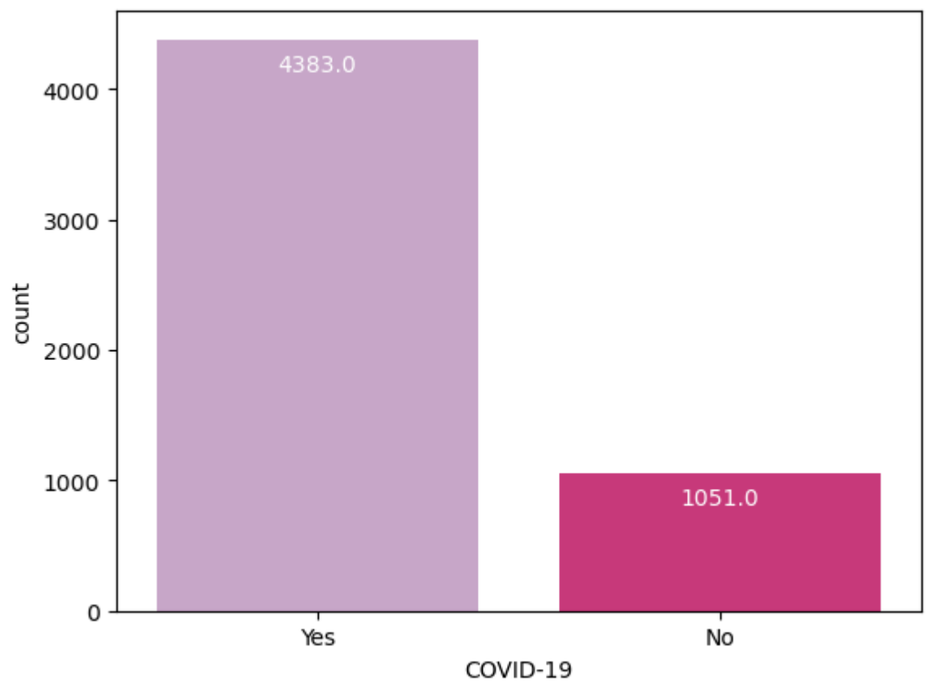
	missing_values	percent_missing %
Breathing Problem	0	0.0
Fever	0	0.0
Dry Cough	0	0.0
Sore throat	0	0.0
Running Nose	0	0.0
Asthma	0	0.0
Chronic Lung Disease	0	0.0
Headache	0	0.0
Heart Disease	0	0.0
Diabetes	0	0.0
Hyper Tension	0	0.0
Fatigue	0	0.0
Gastrointestinal	0	0.0
Abroad travel	0	0.0
Contact with COVID Patient	0	0.0
Attended Large Gathering	0	0.0
Visited Public Exposed Places	0	0.0
Family working in Public Exposed Places	0	0.0
Wearing Masks	0	0.0
Sanitization from Market	0	0.0
COVID-19	0	0.0

In [10]:
```
sns.heatmap(covid_data.isnull(),yticklabels=False,cbar=False,cmap='Pastel1')
```

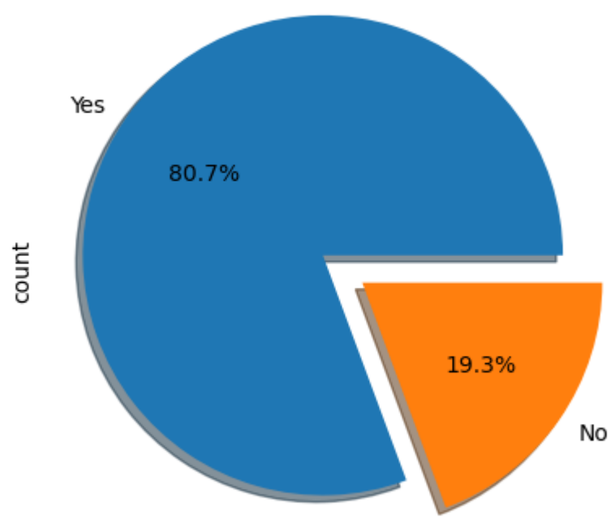
Out[10]:
```
<Axes: >
```



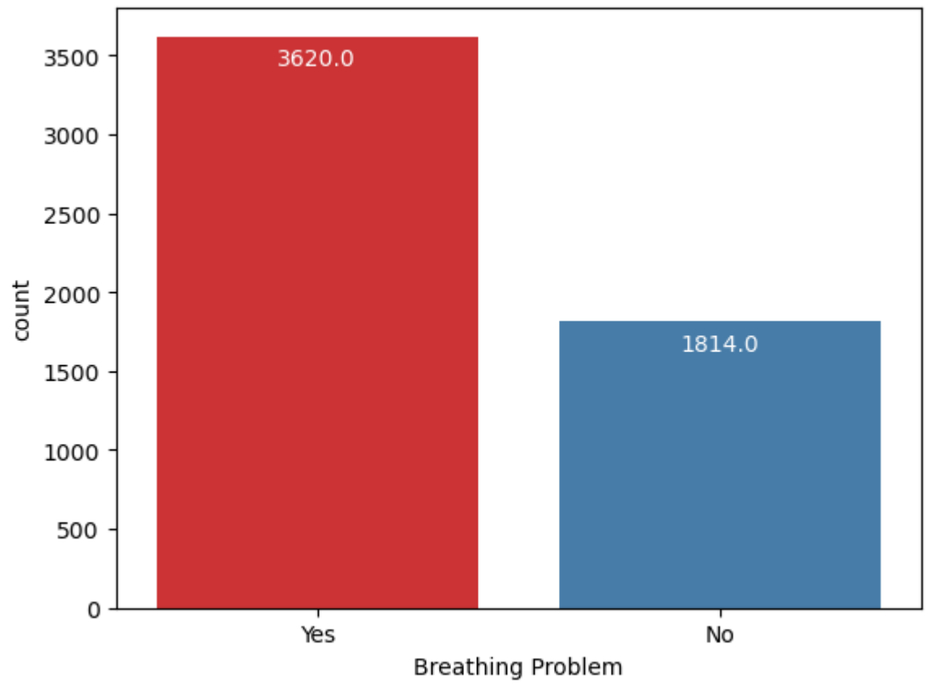
In [11]:
```
ax = sns.countplot(x='COVID-19',data=covid_data, palette="PuRd")
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.4, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```



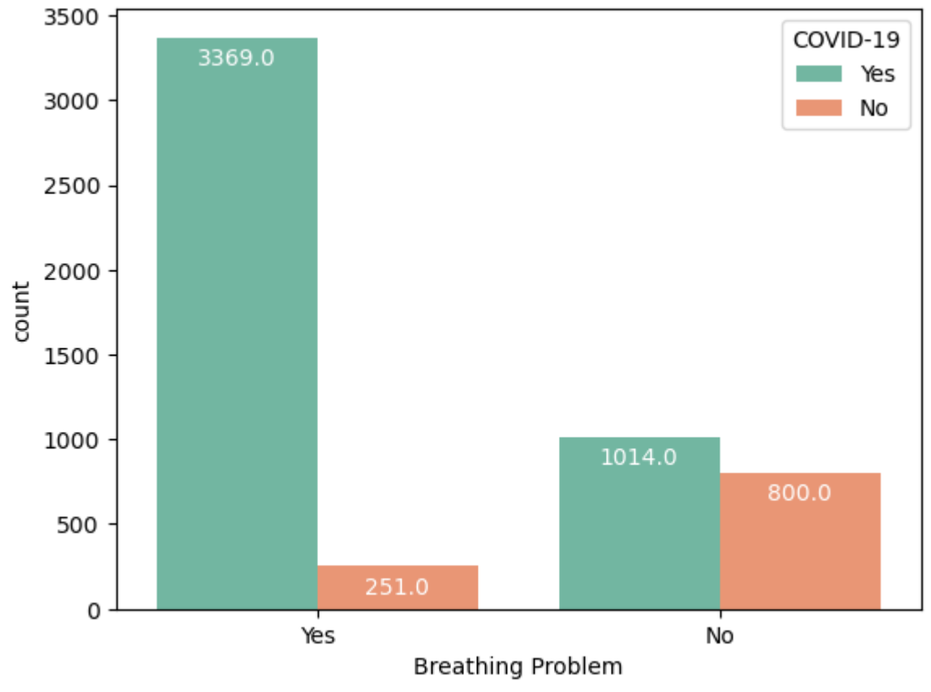
In [12]:
```
covid_data["COVID-19"].value_counts().plot.pie(explode=[0.1,0.1],autopct='%1.1f%%',shadow=True)
plt.title('Covid Positive');
```



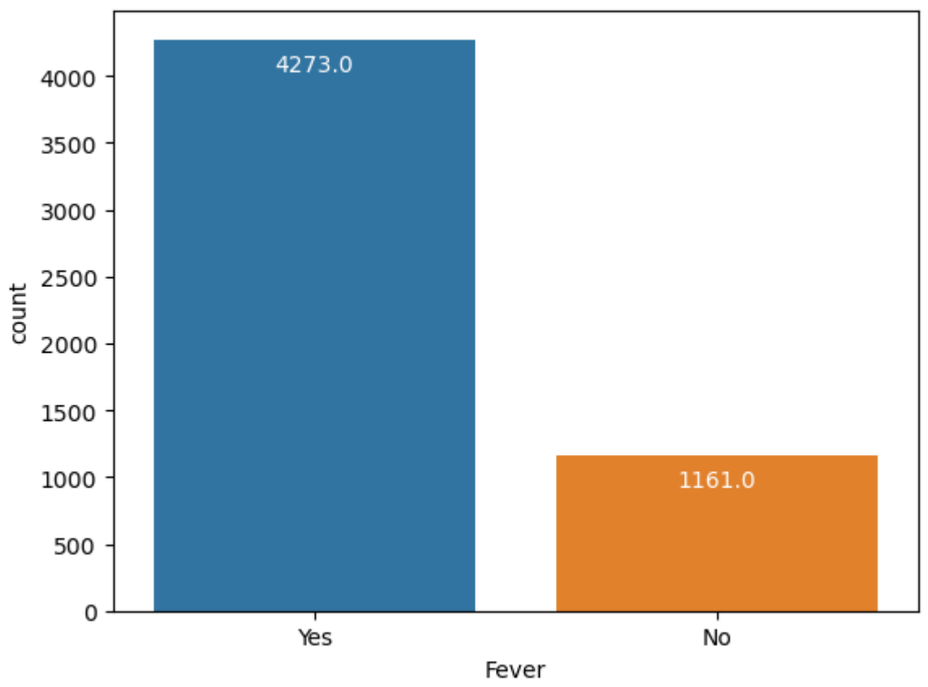
In [13]:
```
ax = sns.countplot(x='Breathing Problem',data=covid_data, palette="Set1")
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.4, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```



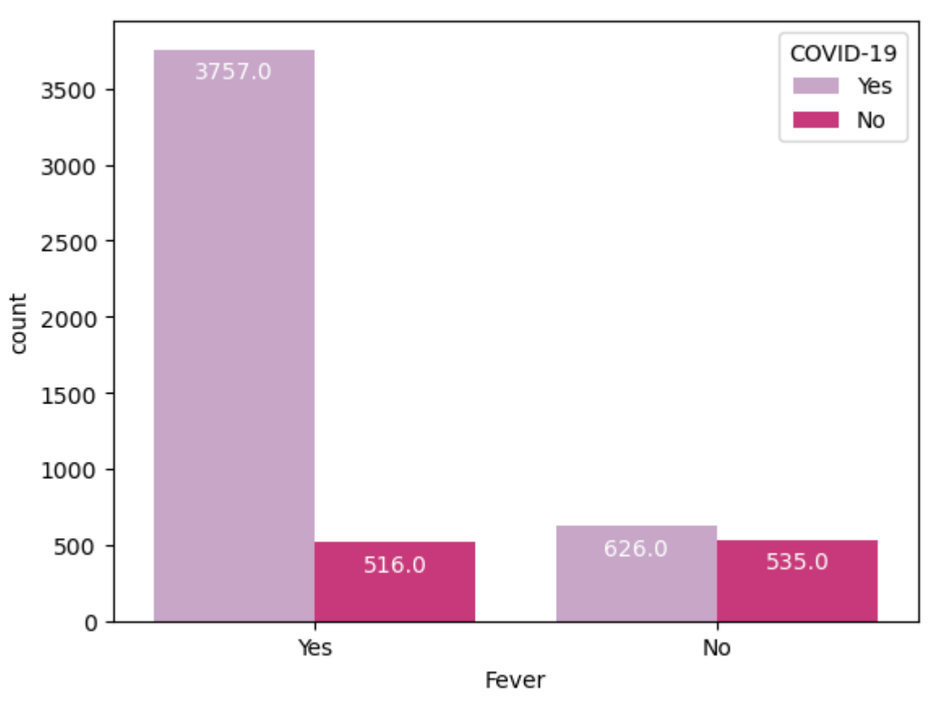
In [14]:
```
ax = sns.countplot(x='Breathing Problem',hue='COVID-19',data=covid_data, palette="Set2")
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.2, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```



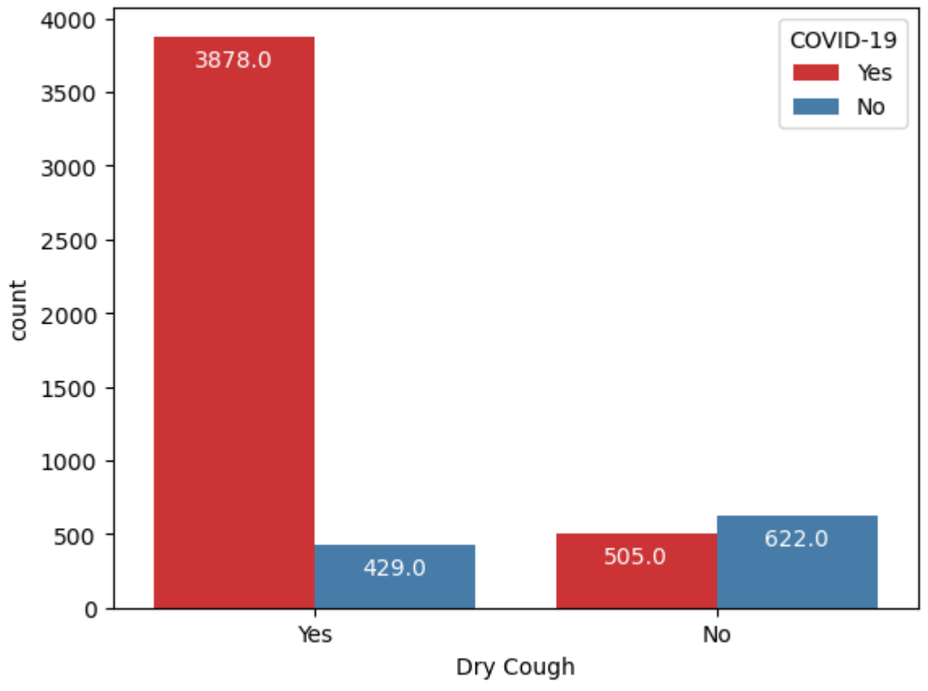
In [15]:
```
ax = sns.countplot(x='Fever',data=covid_data)
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.4, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```



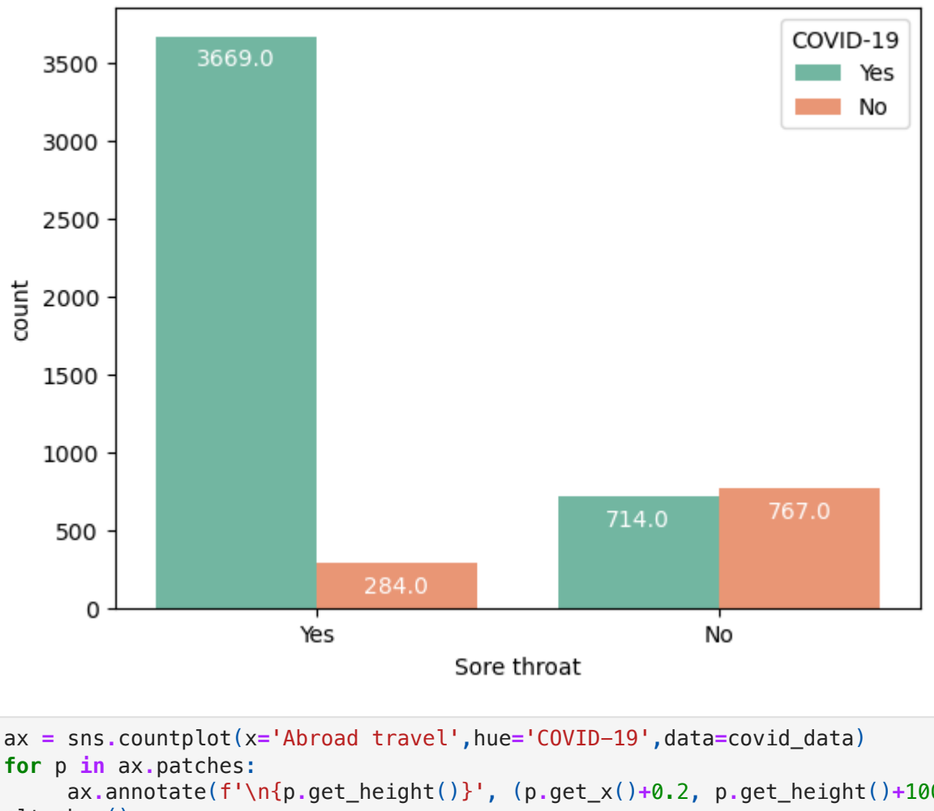
In [16]:
```
ax = sns.countplot(x='Fever',hue='COVID-19',data=covid_data, palette="PuRd")
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.2, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```



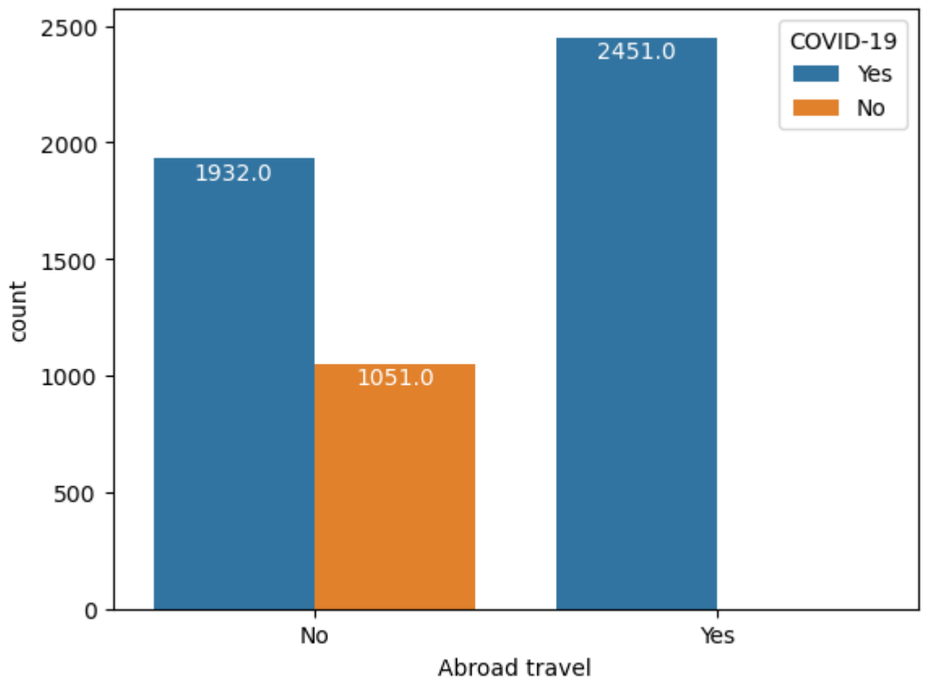
In [17]:
```
ax = sns.countplot(x='Dry Cough',hue='COVID-19',data=covid_data, palette="Set1")
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.2, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```



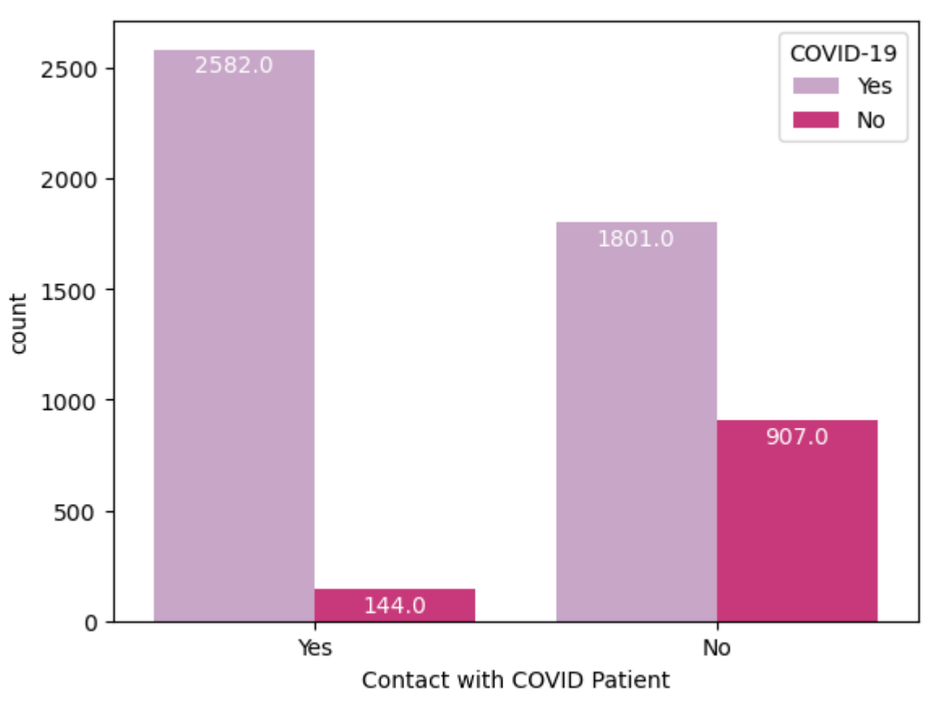
In [18]:
```
ax = sns.countplot(x='Sore throat',hue='COVID-19',data=covid_data, palette="Set2")
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.2, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```



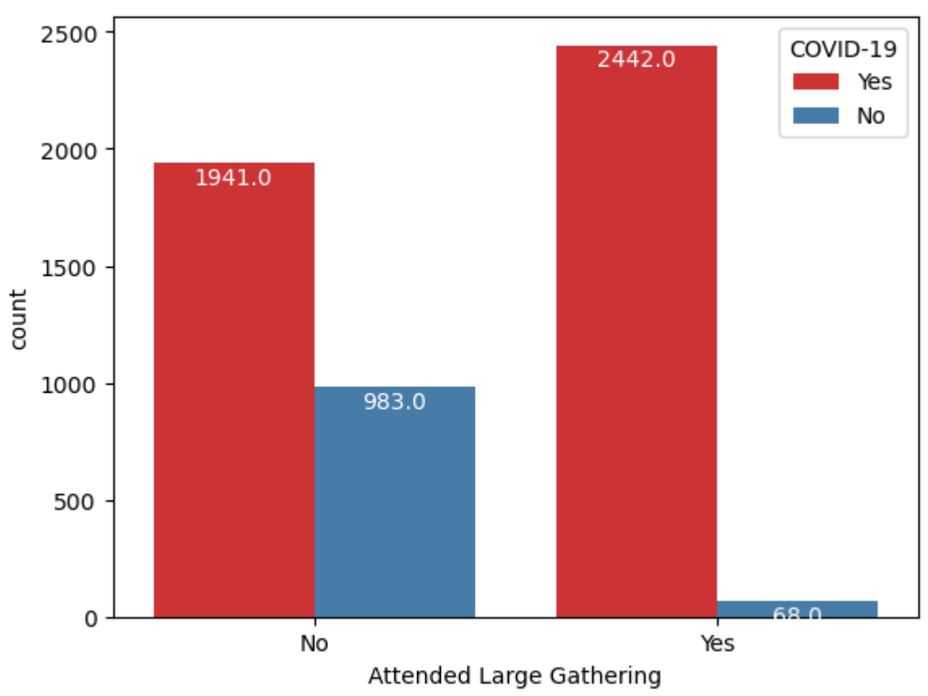
In [19]:
```
ax = sns.countplot(x='Abroad travel',hue='COVID-19',data=covid_data)
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.2, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```



In [20]:
```
ax = sns.countplot(x='Contact with COVID Patient',hue='COVID-19',data=covid_data, palette="PuRd")
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.2, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```



In [21]:
```
ax = sns.countplot(x='Attended Large Gathering',hue='COVID-19',data=covid_data, palette="Set1")
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.2, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```



In [22]:
```
e=LabelEncoder()
```

In [23]:
```
covid_data['Breathing Problem']=e.fit_transform(covid_data['Breathing Problem'])
covid_data['Fever']=e.fit_transform(covid_data['Fever'])
covid_data['Dry Cough']=e.fit_transform(covid_data['Dry Cough'])
covid_data['Sore throat']=e.fit_transform(covid_data['Sore throat'])
covid_data['Running Nose']=e.fit_transform(covid_data['Running Nose'])
covid_data['Asthma']=e.fit_transform(covid_data['Asthma'])
covid_data['Chronic Lung Disease']=e.fit_transform(covid_data['Chronic Lung Disease'])
covid_data['Headache']=e.fit_transform(covid_data['Headache'])
covid_data['Heart Disease']=e.fit_transform(covid_data['Heart Disease'])
covid_data['Diabetes']=e.fit_transform(covid_data['Diabetes'])
covid_data['Hyper Tension']=e.fit_transform(covid_data['Hyper Tension'])
covid_data['Abroad travel']=e.fit_transform(covid_data['Abroad travel'])
covid_data['Contact with COVID Patient']=e.fit_transform(covid_data['Contact with COVID Patient'])
covid_data['Attended Large Gathering']=e.fit_transform(covid_data['Attended Large Gathering'])
covid_data['Visited Public Exposed Places']=e.fit_transform(covid_data['Visited Public Exposed Places'])
covid_data['Family working in Public Exposed Places']=e.fit_transform(covid_data['Family working in Public Exposed Places'])
covid_data['Wearing Masks']=e.fit_transform(covid_data['Wearing Masks'])
covid_data['Sanitization from Market']=e.fit_transform(covid_data['Sanitization from Market'])
covid_data['COVID-19']=e.fit_transform(covid_data['COVID-19'])
covid_data['Dry Cough']=e.fit_transform(covid_data['Dry Cough'])
covid_data['Sore throat']=e.fit_transform(covid_data['Sore throat'])
covid_data['Gastrointestinal ']=e.fit_transform(covid_data['Gastrointestinal '])
covid_data['Fatigue ']=e.fit_transform(covid_data['Fatigue '])
```

In [24]:
```
covid_data.head()
```

Out[24]:

	Breathing Problem	Fever	Dry Cough	Sore throat	Running Nose	Asthma	Chronic Lung Disease	Headache	Heart Disease	Diabetes	...	Fatigue	Gastrointestinal	Abroad travel	Contact with COVID Patient	Attended Large Gathering	Visited Public Exposed Places	Family working in Public Exposed Places	Wearing Masks	Sanitization from Market	COVID-19
0	1	1	1	1	1	0	0	0	0	1	...	1	1	0	1	0	1	1	0	0	1
1	1	1	1	1	0	1	1	1	0	0	...	1	0	0	0	1	1	0	0	0	1
2	1	1	1	1	1	1	1	1	0	1	...	1	1	1	0	0	0	0	0	0	1
3	1	1	1	0	0	1	0	0	1	1	...	0	0	1	0	1	1	0	0	0	1
4	1	1	1	1	1	0	1	1	1	1	...	0	1	0	1	0	1	0	0	0	1

5 rows × 21 columns

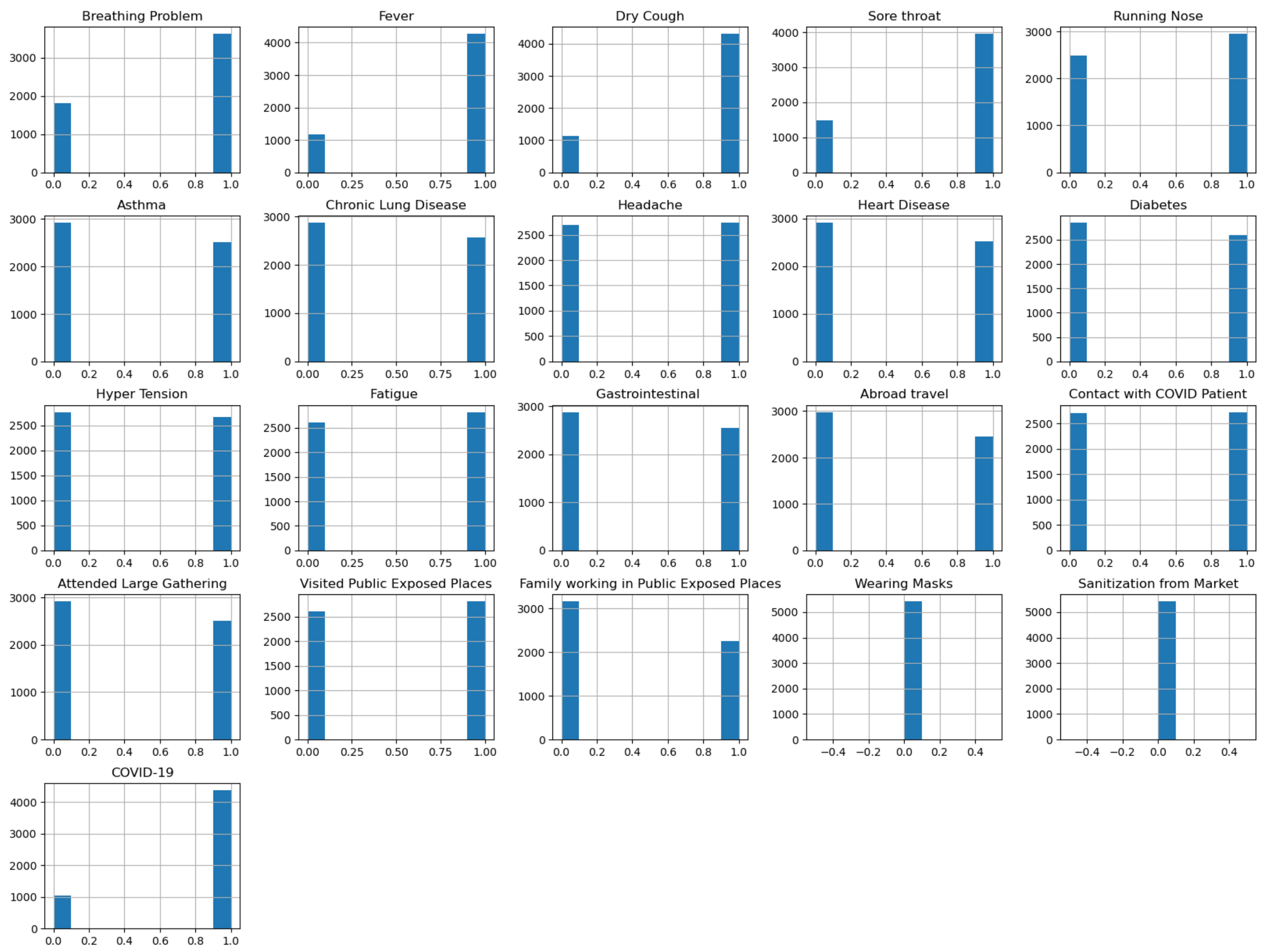
In [25]:
```
covid_data
```

Out[25]:

	Breathing Problem	Fever	Dry Cough	Sore throat	Running Nose	Asthma	Chronic Lung Disease	Headache	Heart Disease	Diabetes	...	Fatigue	Gastrointestinal	Abroad travel	Contact with COVID Patient	Attended Large Gathering	Visited Public Exposed Places	Family working in Public Exposed Places	Wearing Masks	Sanitization from Market	COVID-19
0	1	1	1	1	1	0	0	0	0	1	...	1	1	0	1	0	1	1	0	0	1
1	1	1	1	1	0	1	1	1	0	0	...	1	0	0	0	1	1	0	0	0	1
2	1	1	1	1	1	1	1	1	0	1	...	1	1	1	0	0	0	0	0	0	1
3	1	1	1	0	0	1	0	0	1	1	...	0	0	1	0	1	1	0	0	0	1
4	1	1	1	1	1	0	1	1	1	1	...	0	1	0	1	0	1	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5429	1	1	0	1	1	1	1	0	0	0	...	1	1	0	0	0	0	0	0	0	1
5430	1	1	1	0	1	1	0	1	0	1	...	1	0	0	0	0	0	0	0	0	1
5431	1	1	1	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0
5432	1	1	1	0	1	0	0	1	1	0	...	0	0	0	0	0	0	0	0	0	0
5433	1	1	1	0	1	1	0	1	0	1	...	1	0	0	0	0	0	0	0	0	0

5434 rows × 21 columns

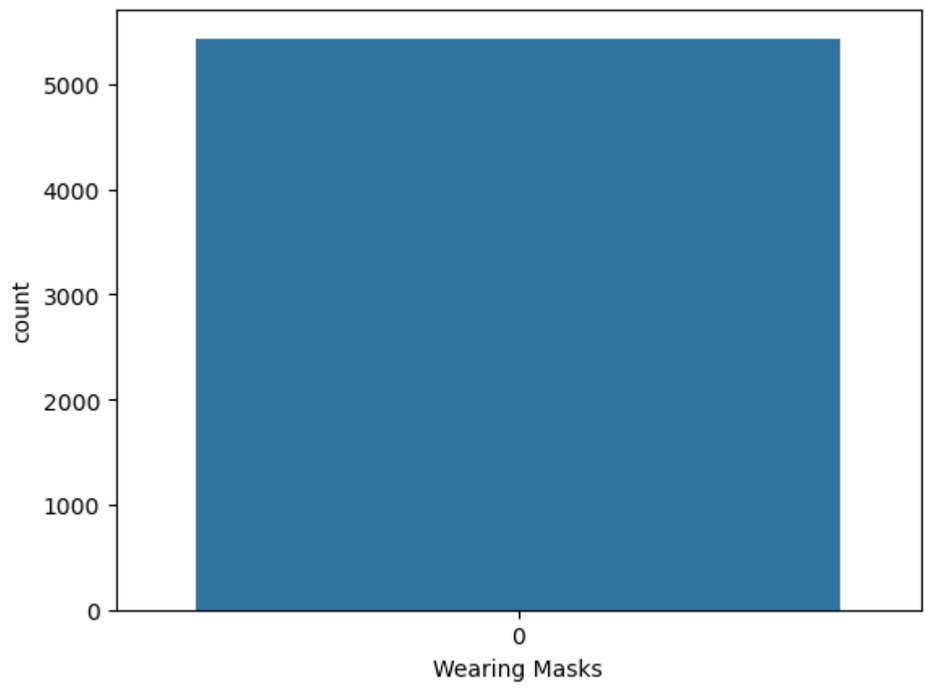
In [26]:
```
covid_data.hist(figsize=(20,15));
```



In [27]:
```
print(covid_data['Wearing Masks'].value_counts())
sns.countplot(x='Wearing Masks',data=covid_data)
```

```
Wearing Masks
0    5434
Name: count, dtype: int64
```

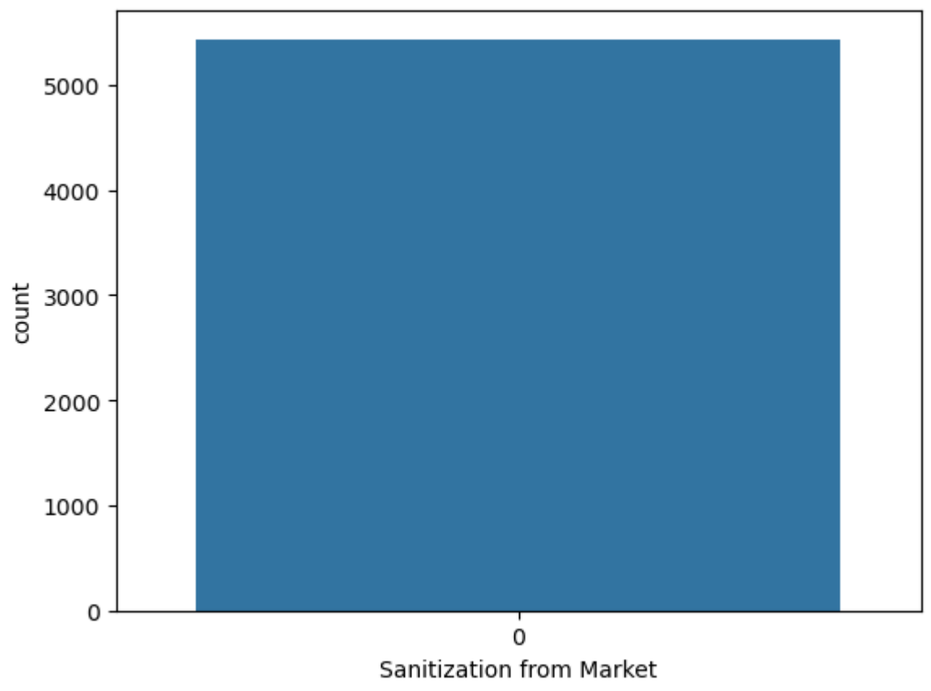
Out[27]:
```
<Axes: xlabel='Wearing Masks', ylabel='count'>
```



In [28]:
```
print(covid_data['Sanitization from Market'].value_counts())
sns.countplot(x='Sanitization from Market',data=covid_data)
```

```
Sanitization from Market
0    5434
Name: count, dtype: int64
<Axes: xlabel='Sanitization from Market', ylabel='count'>
```

Out[28]:



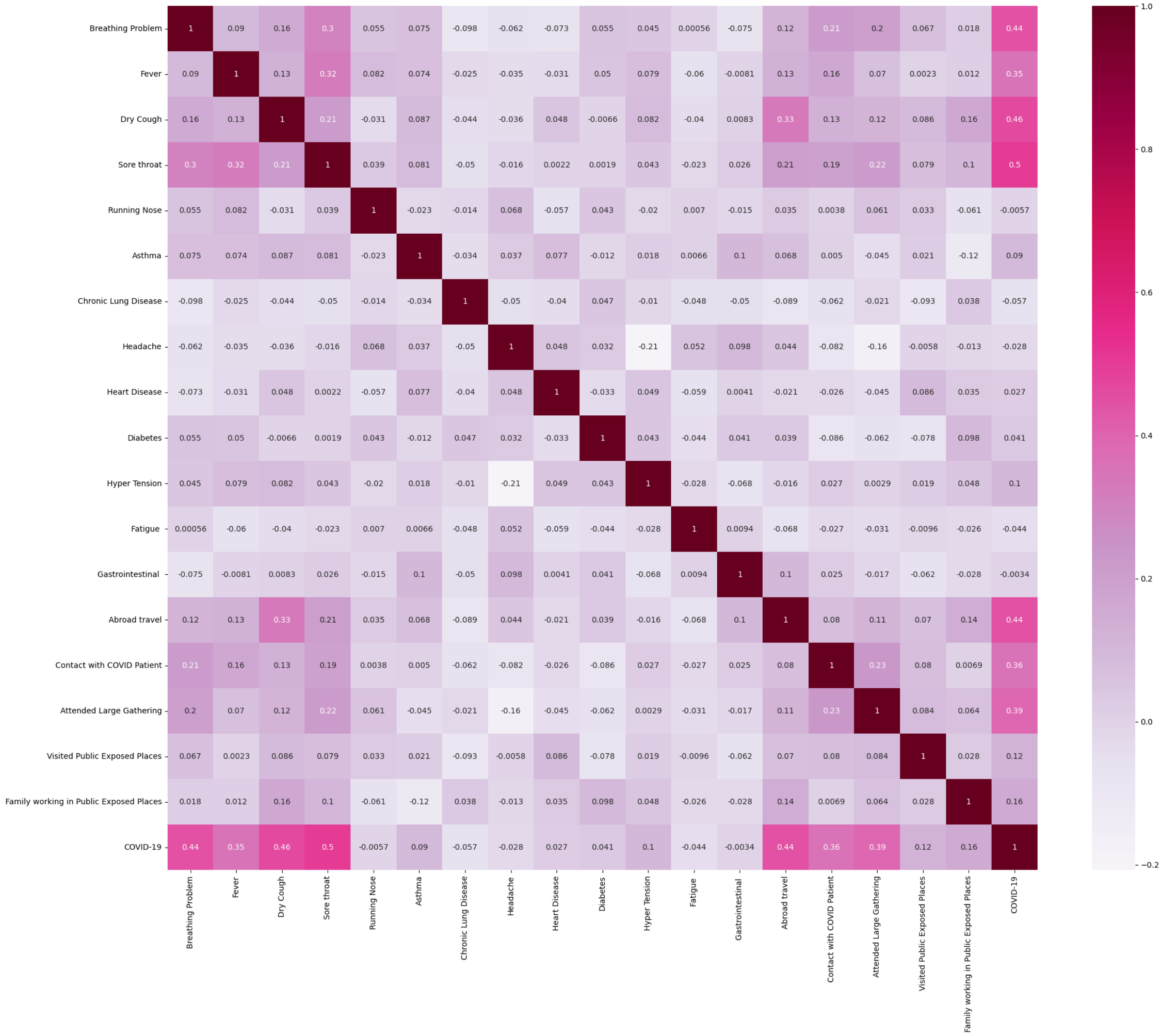
In [29]:
```
covid_data=covid_data.drop('Wearing Masks',axis=1)
covid_data=covid_data.drop('Sanitization from Market',axis=1)
```

In [30]:
```
covid_data.columns
```

Out[30]:
```
Index(['Breathing Problem', 'Fever', 'Dry Cough', 'Sore throat',
       'Running Nose', 'Asthma', 'Chronic Lung Disease', 'Headache',
       'Heart Disease', 'Diabetes', 'Hyper Tension', 'Fatigue ',
       'Gastrointestinal ', 'Abroad travel', 'Contact with COVID Patient',
       'Attended Large Gathering', 'Visited Public Exposed Places',
       'Family working in Public Exposed Places', 'COVID-19'],
      dtype='object')
```

In [31]:
```
plt.figure(figsize=(25,20))
sns.heatmap(covid_data.corr(), annot=True, cmap="PuRd")
```

Out[31]:
```
<Axes: >
```



In [32]:
```
x=covid_data.drop('COVID-19',axis=1)
y=covid_data['COVID-19']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20, random_state = 101)
```

In [33]:
```
accuracies = {}
algo_time={}
r2_scores={}
mean_squared_errors={}
roc_auc_scores={}
```

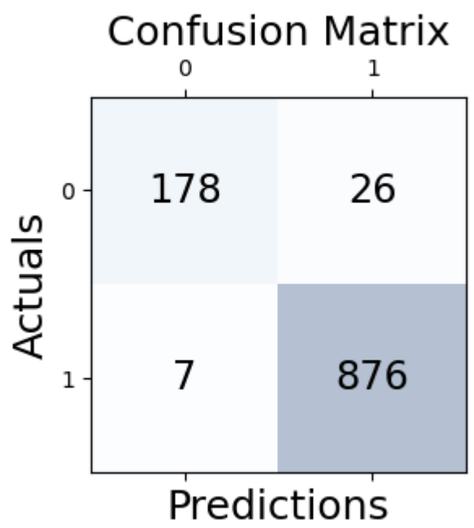
In [34]:
```
def print_performance2(yt,clf,clf_name):
    y_pred=clf.predict(x_test)
    roc_auc_scores[clf_name]=roc_auc_score(yt,y_pred)*100
    mean_squared_errors[clf_name]=mean_squared_error(yt,y_pred)*100
    r2_scores[clf_name]=r2_score(yt,y_pred)*100
    accuracies[clf_name]=clf.score(x_train,y_train)*100
    print('ROC_AUC value :',roc_auc_scores[clf_name],"%",'\n')
    print("Mean Squared Error :",mean_squared_errors[clf_name],"%")
    print("\nR2 score is :",r2_scores[clf_name],"%")
    print("\nAccuracy Score :",accuracies[clf_name],"%")
    print('\nClassification Report : ','\n',classification_report(yt,y_pred))

    confusionmatrix=confusion_matrix(yt,y_pred)

    fig, ax = plt.subplots(figsize=(3, 3))
    ax.matshow(confusionmatrix, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(confusionmatrix.shape[0]):
        for j in range(confusionmatrix.shape[1]):
            ax.text(x=j, y=i,s=confusionmatrix[i, j], va='center', ha='center', size='xx-large')

    plt.xlabel('Predictions', fontsize=18)
    plt.ylabel('Actuals', fontsize=18)
    plt.title('Confusion Matrix', fontsize=18)
```

LOGISTIC REGRESSION

In [ ]:
```
```

In [35]:
```
print("LOGISTIC REGRESSION")
start = time.time()
lr = LogisticRegression()
lr.fit(x_train, y_train)
end = time.time()

print_performance2(y_test,lr,'LOGISTIC REGRESSION')
#acc = lr.score(x_train, y_train)*100
#accuracies['LOGISTIC REGRESSION'] = acc
algo_time['LOGISTIC REGRESSION']=end-start
```

```
LOGISTIC REGRESSION
ROC_AUC value : 93.23107498945218 %

Mean Squared Error : 3.035878564857406 %

R2 score is : 80.08627006861634 %

Accuracy Score : 97.03243616287095 %

Classification Report :
              precision    recall  f1-score   support

           0       0.96      0.87      0.92       204
           1       0.97      0.99      0.98       883

    accuracy                           0.97      1087
   macro avg       0.97      0.93      0.95      1087
weighted avg       0.97      0.97      0.97      1087
```



K-NEAREST NEIGHBOURS

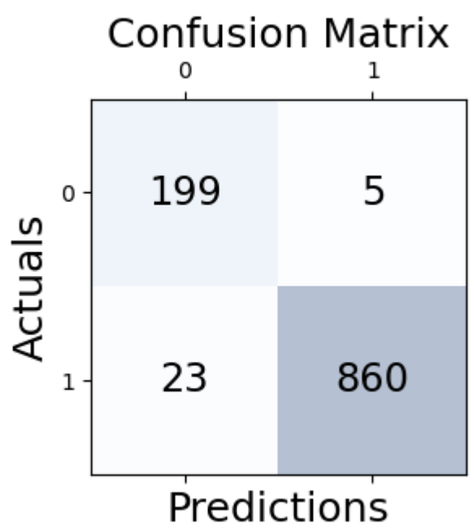
In [36]:
```
start = time.time()
knn = KNeighborsClassifier()
# assigning the dictionary of variables whose optimium value is to be retrieved
param_grid = {'n_neighbors' : np.arange(1,50)}
knn_cv = GridSearchCV(knn, param_grid, cv=5)
# training the model with the training data and best parameter
knn_cv.fit(x_train,y_train)
end=time.time()
algo_time['K-NEAREST NEIGHBOURS']=end-start
```

In [37]:
```
# finding out the best parameter chosen to train the model
print("The best paramter we have is: {}" .format(knn_cv.best_params_))

# finding out the best score the chosen parameter achieved
print("The best score we have achieved is: {}" .format(knn_cv.best_score_))
```

```
The best paramter we have is: {'n_neighbors': 2}
The best score we have achieved is: 0.9809068423210718
```

In [38]:
```
print("K-NEAREST NEIGHBOURS")
print_performance2(y_test,knn_cv,'K-NEAREST NEIGHBOURS')
#acc = knn_cv.score(x_train, y_train)*100
#accuracies['K-NEAREST NEIGHBOURS'] = acc
```

```
K-NEAREST NEIGHBOURS
ROC_AUC value : 97.47213154797593 %

Mean Squared Error : 2.5758969641214353 %

R2 score is : 83.10350187640175 %

Accuracy Score : 98.3666896710375 %

Classification Report :
              precision    recall  f1-score   support

           0       0.90      0.98      0.93       204
           1       0.99      0.97      0.98       883

    accuracy                           0.97      1087
   macro avg       0.95      0.97      0.96      1087
weighted avg       0.98      0.97      0.97      1087
```



RANDOM FOREST

In [39]:
```
rf_start=time.time()
rfc=RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x_train, y_train)
rf_end=time.time()
algo_time['RANDOM FOREST TREE']=rf_end-rf_start
```

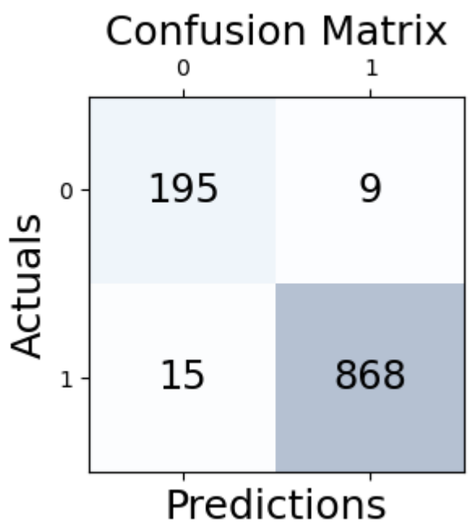
```
/Users/bhavinpatel/anaconda3/lib/python3.11/site-packages/sklearn/model_selection/_validation.py:547: FitFailedWarning:
100 fits failed out of a total of 300.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
100 fits failed with the following error:
Traceback (most recent call last):
  File "/Users/bhavinpatel/anaconda3/lib/python3.11/site-packages/sklearn/model_selection/_validation.py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Users/bhavinpatel/anaconda3/lib/python3.11/site-packages/sklearn/base.py", line 1467, in wrapper
    estimator._validate_params()
  File "/Users/bhavinpatel/anaconda3/lib/python3.11/site-packages/sklearn/base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "/Users/bhavinpatel/anaconda3/lib/python3.11/site-packages/sklearn/utils/_param_validation.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of RandomForestClassifier must be an int in the range [1, inf), a float in the range (0.0, 1.
0], a str among {'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/Users/bhavinpatel/anaconda3/lib/python3.11/site-packages/sklearn/model_selection/_search.py:1051: UserWarning: One or more of the test scores are non-finite: [       nan
nan 0.95445022 0.95329947 0.95445022 0.95329947
        nan        nan 0.9689433  0.9689433  0.9689433  0.9689433
        nan        nan 0.96986363 0.97055328 0.96986363 0.97055328
        nan        nan 0.97400394 0.97423303 0.97400394 0.97423303
        nan        nan 0.98136529 0.98136529 0.98136529 0.98136529
        nan        nan 0.95675013 0.9576702  0.95675013 0.9576702
        nan        nan 0.968483   0.9689433  0.968483   0.9689433
        nan        nan 0.97055328 0.97078343 0.97055328 0.97078343
        nan        nan 0.97216354 0.97354258 0.97216354 0.97354258
        nan        nan 0.98136529 0.98044469 0.98136529 0.98044469]
  warnings.warn(
```

In [40]:
```
# finding out the best parameter chosen to train the model
print("The best paramter we have is: {}" .format(CV_rfc.best_params_))

# finding out the best score the chosen parameter achieved
print("The best score we have achieved is: {}" .format(CV_rfc.best_score_*100))
```

```
The best paramter we have is: {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 200}
The best score we have achieved is: 98.13652897371796
```

In [41]:
```
print("RANDOM FOREST TREE")
print_performance2(y_test,CV_rfc,'RANDOM FOREST TREE')
#acc = CV_rfc.score(x_train, y_train)*100
#accuracies['RANDOM FOREST TREE'] = acc
```

```
RANDOM FOREST TREE
ROC_AUC value : 96.94474052361602 %

Mean Squared Error : 2.2079116835326587 %

R2 score is : 85.51728732263008 %

Accuracy Score : 98.38969404186795 %

Classification Report :
              precision    recall  f1-score   support

           0       0.93      0.96      0.94       204
           1       0.99      0.98      0.99       883

    accuracy                           0.98      1087
   macro avg       0.96      0.97      0.96      1087
weighted avg       0.98      0.98      0.98      1087
```



GRADIENT BOOSTING CLASSIFIER

Decision Tree

Naive bayes

colors = ["purple", "green", "orange", "blue", "red", "yellow", "magenta"]

sns.set_style("whitegrid") plt.figure(figsize=(10,5)) plt.yticks(np.arange(0,100,10)) plt.ylabel("Accuracy %") plt.xlabel("Algorithms") plt.xticks(rotation=90) ax = sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()), palette=colors) plt.show()

In [42]:
```
!conda install -c plotly plotly
!pip install colorama
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: -
The environment is inconsistent, please check the package plan carefully
The following packages are causing the inconsistency:

  - defaults/osx-arm64::_anaconda_depends==2023.09=py311_openblas_1
  - defaults/osx-arm64::imbalanced-learn==0.10.1=py311hca03da5_1
done


==> WARNING: A newer version of conda exists. <==
  current version: 23.7.4
  latest version: 24.3.0

Please update conda by running

    $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

     conda install conda=24.3.0



## Package Plan ##

  environment location: /Users/bhavinpatel/anaconda3

  added / updated specs:
    - plotly


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2024.3.11  |       hca03da5_0         128 KB
    certifi-2024.2.2           |  py311hca03da5_0         161 KB
    openssl-3.0.13             |       h1a28f6b_0         5.0 MB
    plotly-5.21.0              |             py_0         7.2 MB  plotly
    scikit-learn-1.3.0         |  py311h7aedaa7_1         8.8 MB
    ------------------------------------------------------------
                                           Total:        21.3 MB

The following NEW packages will be INSTALLED:

  scikit-learn       pkgs/main/osx-arm64::scikit-learn-1.3.0-py311h7aedaa7_1

The following packages will be UPDATED:

  ca-certificates                    2023.12.12-hca03da5_0 --> 2024.3.11-hca03da5_0
  certifi                        2023.11.17-py311hca03da5_0 --> 2024.2.2-py311hca03da5_0
  openssl                              3.0.12-h1a28f6b_0 --> 3.0.13-h1a28f6b_0
  plotly             pkgs/main/osx-arm64::plotly-5.9.0-py3~ --> plotly/noarch::plotly-5.21.0-py_0


Proceed ([y]/n)? ^C

CondaSystemExit:
Operation aborted.  Exiting.

Requirement already satisfied: colorama in /Users/bhavinpatel/anaconda3/lib/python3.11/site-packages (0.4.6)
```

In [43]:
```
import plotly.express as px
fig = px.bar(x=list(accuracies.keys()), y=list(accuracies.values()))
fig.update_traces(marker_color='teal', marker_line_color='rgb(8,48,107)', marker_line_width=1.5)
fig.update_layout(title="Accuracy Comparision", xaxis_title="Model", yaxis_title="Accuracy")
fig.show()
```

In [44]:
```
fig = px.bar(x=list(algo_time.keys()), y=list(algo_time.values()))
fig.update_traces(marker_color='teal', marker_line_color='rgb(8,48,107)', marker_line_width=1.5)
fig.update_layout(title="Algorithm Time Comparision", xaxis_title="Model", yaxis_title="")
fig.show()
```

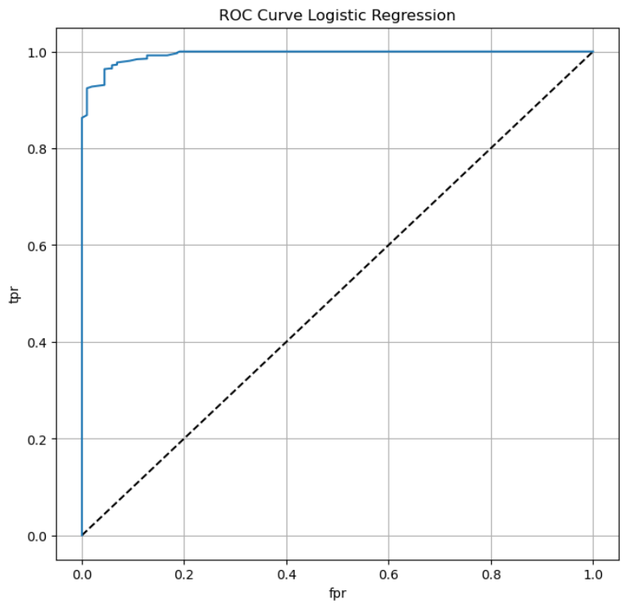
In [45]:
```
fig = px.bar(x=list(r2_scores.keys()), y=list(r2_scores.values()))
fig.update_traces(marker_color='teal', marker_line_color='rgb(8,48,107)', marker_line_width=1.5)
fig.update_layout(title="R2 Score Comparision", xaxis_title="Model", yaxis_title="R2 Scores")
fig.show()
```

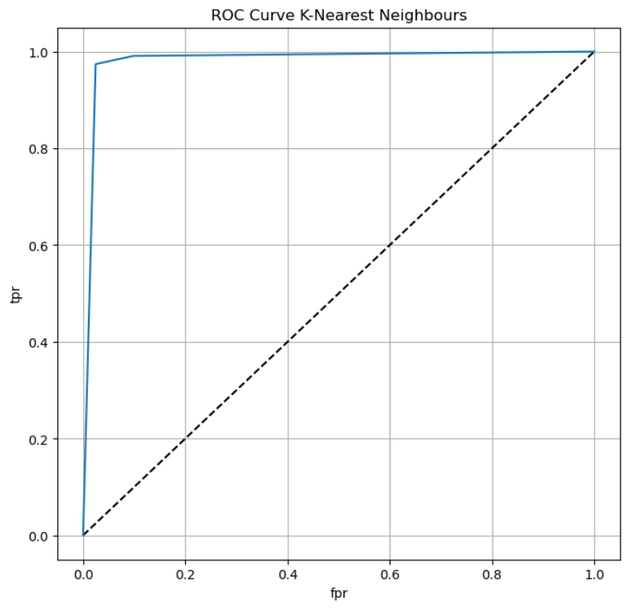
In [46]:
```
fig = px.bar(x=list(mean_squared_errors.keys()), y=list(mean_squared_errors.values()))
fig.update_traces(marker_color='teal', marker_line_color='rgb(8,48,107)', marker_line_width=1.5)
fig.update_layout(title="Mean Squared Error Comparision", xaxis_title="Model", yaxis_title="Mean Squared Error")
fig.show()
```

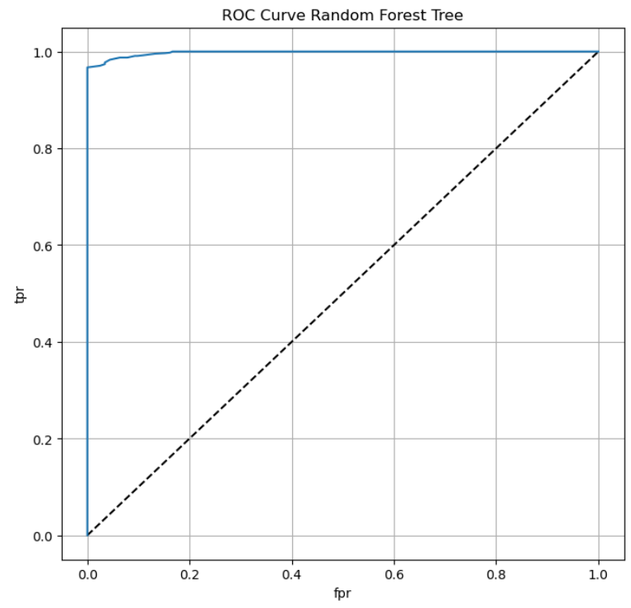
In [47]:
```
fig = px.bar(x=list(roc_auc_scores.keys()), y=list(roc_auc_scores.values()))
fig.update_traces(marker_color='teal', marker_line_color='rgb(8,48,107)', marker_line_width=1.5)
fig.update_layout(title="ROC Score Comparision", xaxis_title="Model", yaxis_title="ROC Scores")
fig.show()
```

In [48]:
```
import plotly.graph_objects as go
Algos=list(roc_auc_scores.keys())

fig = go.Figure(data=[
    go.Bar(name='Accuracies', x=Algos, y=list(accuracies.values())),
    go.Bar(name='R2 scores', x=Algos, y=list(r2_scores.values())),
    go.Bar(name='Mean Squared Errors', x=Algos, y=list(mean_squared_errors.values())),
    go.Bar(name='ROC Auc Scores', x=Algos, y=list(roc_auc_scores.values()))
])
# Change the bar mode
fig.update_layout(barmode='group')
fig.show()
```

In [49]:
```
from sklearn.metrics import roc_curve
plt.figure(figsize=(25,16))
# Logistic Regression Classification
Y_predict1_proba = lr.predict_proba(x_test)
Y_predict1_proba = Y_predict1_proba[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, Y_predict1_proba)
plt.subplot(441)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='ANN')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC Curve Logistic Regression')
plt.grid(True)

Y_predict1_proba = knn_cv.predict_proba(x_test)
Y_predict1_proba = Y_predict1_proba[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, Y_predict1_proba)
plt.subplot(442)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='ANN')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC Curve K-Nearest Neighbours')
plt.grid(True)

Y_predict1_proba = CV_rfc.predict_proba(x_test)
Y_predict1_proba = Y_predict1_proba[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, Y_predict1_proba)
plt.subplot(443)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='ANN')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC Curve Random Forest Tree')
plt.grid(True)

plt.subplots_adjust(top=2, bottom=0.08, left=0.10, right=1.4, hspace=0.45, wspace=0.45)
plt.show()
```







In [ ]:
```
import colorama
from colorama import Fore

print("COVID PREDICTION BASED ON ML ALGORITHMS")
print("Enter 1 for Yes and 0 for No")
Breathing_Problem = int(input("Does the patient have breathing problem ? "))
Fever = int(input("Does the patient have fever ? "))
Dry_Cough = int(input("Does the patient have dry cough ? "))
Sore_throat = int(input("Does the patient have sore throat ? "))
Running_Nose = int(input("Does the patient have running nose ? "))
Asthma = int(input("Does the patient have any record of asthma ? "))
Chronic_Lung_Disease = int(input("Does the patient have any records of chronic lung disease ? "))
Headache = int(input("Is the patient having headche ? "))
Heart_Disease = int(input("Does the patient have any record of any heart disease ? "))
Diabetes = int(input("Does the patient have diabetes ? "))
Hyper_Tension = int(input("Does the patient have hyper tension ? "))
Fatigue = int(input("Does the patient experience fatigue ? "))
Gastrointestinal = int(input("Does the patient have any gastrointestinal disorders ? "))
Abroad_travel = int(input("Has the patient travelled abroad recently ? "))
Contact_with_COVID_Patient = int(input("Was the patient in contact with a covid patient recently ? "))
Attended_Large_Gathering = int(input("Did the patient attend any large gathering event recently ? "))
Visited_Public_Exposed_Places = int(input("Did the patient visit any public exposed places recently ? "))
Family_working_in_Public_Exposed_Places = int(input("Does the patient have any family member working in public exposed places ? "))

patient = [[Breathing_Problem,Fever,Dry_Cough,Sore_throat,Running_Nose,Asthma,Chronic_Lung_Disease,Headache,Heart_Disease,Diabetes,Hyper_Tension,Fatigue,Gastrointestinal,Abroad_
result = knn_cv.predict(patient)
print("\nResults : ",result)

if result == 1:
    print(Fore.RED + 'You may be affected with COVID-19 virus! Please get RTPCR test ASAP and stay in Quarantine for 14 days!')
    print()
else :
    print(Fore.GREEN + 'You do not have any symptoms of COVID-19. Stay home! Stay safe!')
    print()
```

In [ ]:

Contribution Code :

*

In [51]:
```
covid_data = pd.read_csv("Covid Dataset.csv")
```

In [52]:
```
covid_data
```

Out[52]:

	Breathing Problem	Fever	Dry Cough	Sore throat	Running Nose	Asthma	Chronic Lung Disease	Headache	Heart Disease	Diabetes	...	Fatigue	Gastrointestinal	Abroad travel	Contact with COVID Patient	Attended Large Gathering	Visited Public Exposed Places	Family working in Public Exposed Places	Wearing Masks	Sanitization from Market	COVID-19
0	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	...	Yes	Yes	No	Yes	No	Yes	Yes	No	No	Yes
1	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	...	Yes	No	No	No	Yes	Yes	No	No	No	Yes
2	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	...	Yes	Yes	Yes	No	No	No	No	No	No	Yes
3	Yes	Yes	Yes	No	No	Yes	No	No	Yes	Yes	...	No	No	Yes	No	Yes	Yes	No	No	No	Yes
4	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	...	No	Yes	No	Yes	No	Yes	No	No	No	Yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5429	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No	No	...	Yes	Yes	No	No	No	No	No	No	No	Yes
5430	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	...	Yes	No	No	No	No	No	No	No	No	Yes
5431	Yes	Yes	Yes	No	No	No	No	No	Yes	No	...	No	No	No	No	No	No	No	No	No	No
5432	Yes	Yes	Yes	No	Yes	No	No	Yes	Yes	No	...	No	No	No	No	No	No	No	No	No	No
5433	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	...	Yes	No	No	No	No	No	No	No	No	No

5434 rows × 21 columns

In [53]:
```
covid_data.shape
```

Out[53]:
```
(5434, 21)
```

In [54]:
```
covid_data.columns
```

Out[54]:
```
Index(['Breathing Problem', 'Fever', 'Dry Cough', 'Sore throat',
       'Running Nose', 'Asthma', 'Chronic Lung Disease', 'Headache',
       'Heart Disease', 'Diabetes', 'Hyper Tension', 'Fatigue ',
       'Gastrointestinal ', 'Abroad travel', 'Contact with COVID Patient',
       'Attended Large Gathering', 'Visited Public Exposed Places',
       'Family working in Public Exposed Places', 'Wearing Masks',
       'Sanitization from Market', 'COVID-19'],
      dtype='object')
```

In [55]:
```
covid_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5434 entries, 0 to 5433
Data columns (total 21 columns):
 #   Column                                    Non-Null Count  Dtype 
---  ------                                    --------------  ----- 
 0   Breathing Problem                         5434 non-null   object
 1   Fever                                     5434 non-null   object
 2   Dry Cough                                 5434 non-null   object
 3   Sore throat                               5434 non-null   object
 4   Running Nose                              5434 non-null   object
 5   Asthma                                    5434 non-null   object
 6   Chronic Lung Disease                      5434 non-null   object
 7   Headache                                  5434 non-null   object
 8   Heart Disease                             5434 non-null   object
 9   Diabetes                                  5434 non-null   object
 10  Hyper Tension                             5434 non-null   object
 11  Fatigue                                   5434 non-null   object
 12  Gastrointestinal                          5434 non-null   object
 13  Abroad travel                             5434 non-null   object
 14  Contact with COVID Patient                5434 non-null   object
 15  Attended Large Gathering                  5434 non-null   object
 16  Visited Public Exposed Places             5434 non-null   object
 17  Family working in Public Exposed Places   5434 non-null   object
 18  Wearing Masks                             5434 non-null   object
 19  Sanitization from Market                  5434 non-null   object
 20  COVID-19                                  5434 non-null   object
dtypes: object(21)
memory usage: 891.6+ KB
```

In [56]:
```
covid_data.describe().T
```

Out[56]:

	count	unique	top	freq
Breathing Problem	5434	2	Yes	3620
Fever	5434	2	Yes	4273
Dry Cough	5434	2	Yes	4307
Sore throat	5434	2	Yes	3953
Running Nose	5434	2	Yes	2952
Asthma	5434	2	No	2920
Chronic Lung Disease	5434	2	No	2869
Headache	5434	2	Yes	2736
Heart Disease	5434	2	No	2911
Diabetes	5434	2	No	2846
Hyper Tension	5434	2	No	2771
Fatigue	5434	2	Yes	2821
Gastrointestinal	5434	2	No	2883
Abroad travel	5434	2	No	2983
Contact with COVID Patient	5434	2	Yes	2726
Attended Large Gathering	5434	2	No	2924
Visited Public Exposed Places	5434	2	Yes	2820
Family working in Public Exposed Places	5434	2	No	3172
Wearing Masks	5434	1	No	5434
Sanitization from Market	5434	1	No	5434
COVID-19	5434	2	Yes	4383

In [57]:
```
covid_data.head()
```

Out[57]:

	Breathing Problem	Fever	Dry Cough	Sore throat	Running Nose	Asthma	Chronic Lung Disease	Headache	Heart Disease	Diabetes	...	Fatigue	Gastrointestinal	Abroad travel	Contact with COVID Patient	Attended Large Gathering	Visited Public Exposed Places	Family working in Public Exposed Places	Wearing Masks	Sanitization from Market	COVID-19
0	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	...	Yes	Yes	No	Yes	No	Yes	Yes	No	No	Yes
1	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	...	Yes	No	No	No	Yes	Yes	No	No	No	Yes
2	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	...	Yes	Yes	Yes	No	No	No	No	No	No	Yes
3	Yes	Yes	Yes	No	No	Yes	No	No	Yes	Yes	...	No	No	Yes	No	Yes	Yes	No	No	No	Yes
4	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	...	No	Yes	No	Yes	No	Yes	No	No	No	Yes

5 rows × 21 columns

In [58]:
```
covid_data.columns
```

Out[58]:
```
Index(['Breathing Problem', 'Fever', 'Dry Cough', 'Sore throat',
       'Running Nose', 'Asthma', 'Chronic Lung Disease', 'Headache',
       'Heart Disease', 'Diabetes', 'Hyper Tension', 'Fatigue ',
       'Gastrointestinal ', 'Abroad travel', 'Contact with COVID Patient',
       'Attended Large Gathering', 'Visited Public Exposed Places',
       'Family working in Public Exposed Places', 'Wearing Masks',
       'Sanitization from Market', 'COVID-19'],
      dtype='object')
```

In [59]:
```
covid_data.groupby('COVID-19').size()
```

Out[59]:
```
COVID-19
No     1051
Yes    4383
dtype: int64
```

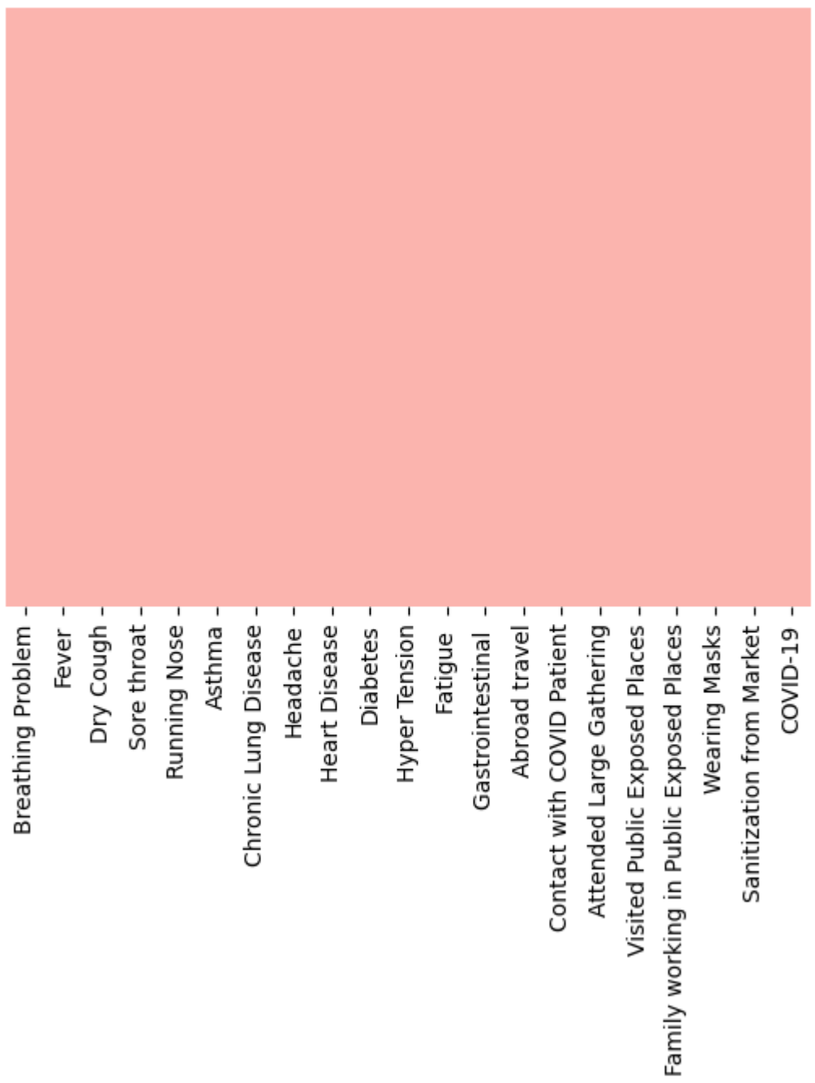
In [60]:
```
# create a table with data missing
missing_values=covid_data.isnull().sum() # missing values

percent_missing = covid_data.isnull().sum()/covid_data.shape[0]*100 # missing value %

value = {
    'missing_values ':missing_values,
    'percent_missing %':percent_missing
}
frame=pd.DataFrame(value)
frame
```

Out[60]:

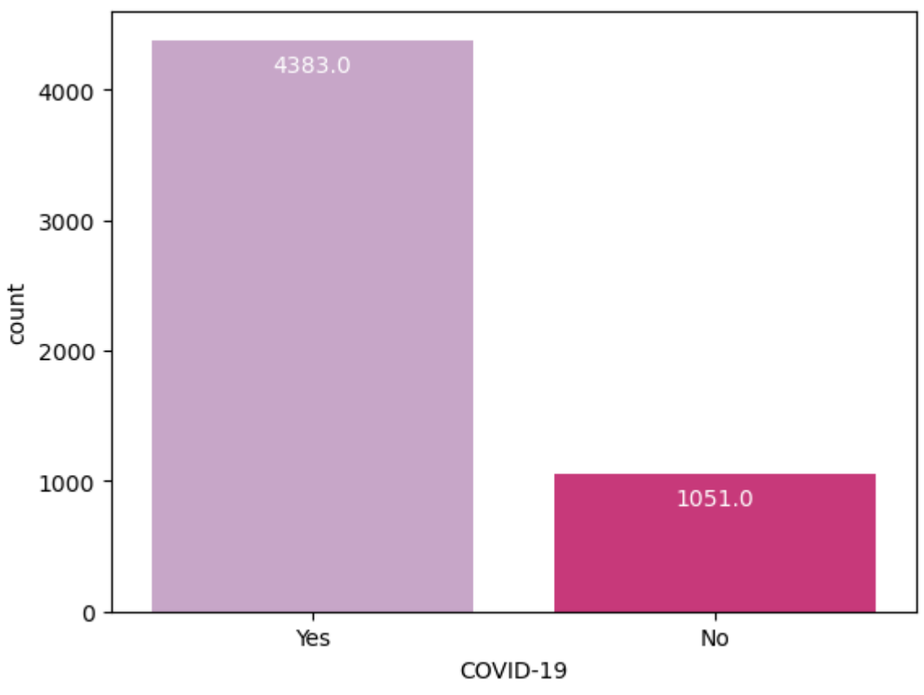
	missing_values	percent_missing %
Breathing Problem	0	0.0
Fever	0	0.0
Dry Cough	0	0.0
Sore throat	0	0.0
Running Nose	0	0.0
Asthma	0	0.0
Chronic Lung Disease	0	0.0
Headache	0	0.0
Heart Disease	0	0.0
Diabetes	0	0.0
Hyper Tension	0	0.0
Fatigue	0	0.0
Gastrointestinal	0	0.0
Abroad travel	0	0.0
Contact with COVID Patient	0	0.0
Attended Large Gathering	0	0.0
Visited Public Exposed Places	0	0.0
Family working in Public Exposed Places	0	0.0
Wearing Masks	0	0.0
Sanitization from Market	0	0.0
COVID-19	0	0.0

In [61]:
```
sns.heatmap(covid_data.isnull(),yticklabels=False,cbar=False,cmap='Pastel1')
```

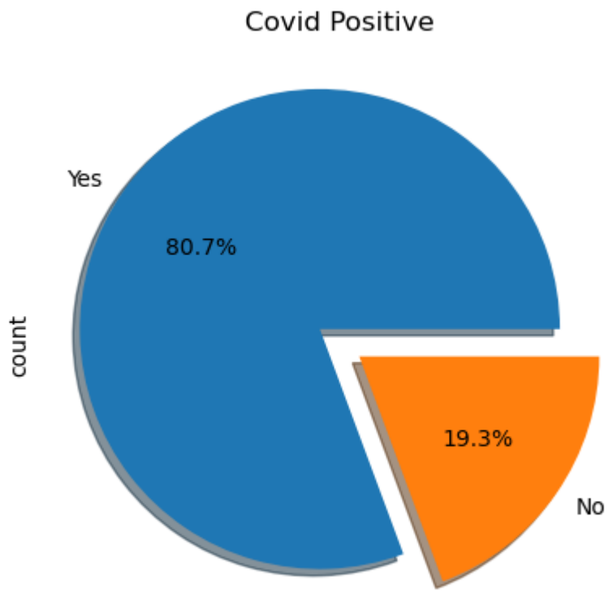
Out[61]:
```
<Axes: >
```



In [62]:
```
ax = sns.countplot(x='COVID-19',data=covid_data, palette="PuRd")
for p in ax.patches:
     ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.4, p.get_height()+100), ha='center', va='top', color='white', size=10)
plt.show()
```

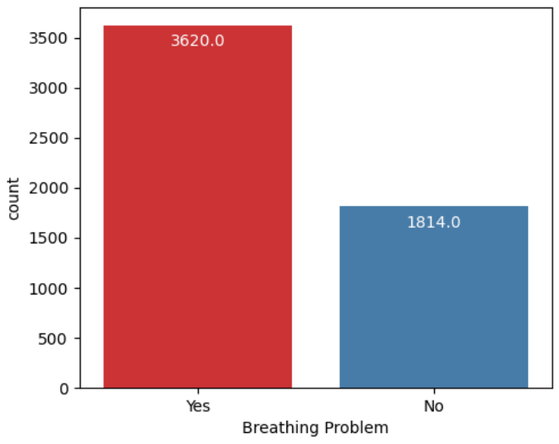


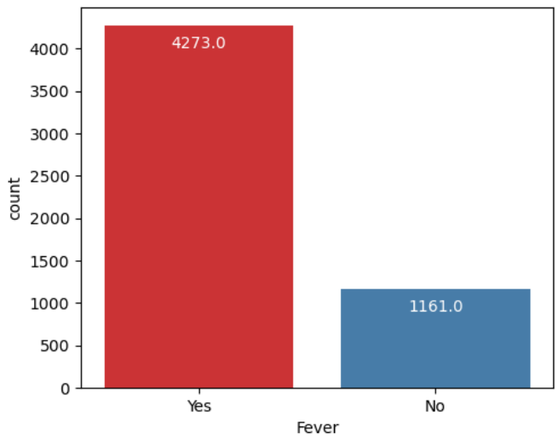
In [63]:
```
country_counts = covid_data["COVID-19"].value_counts()

# Define explode values
explode = [0.1] * len(country_counts)

# Plot the pie chart
country_counts.plot.pie(explode=explode, autopct='%1.1f%%', shadow=True)
plt.title('Covid Positive')
plt.show()
```

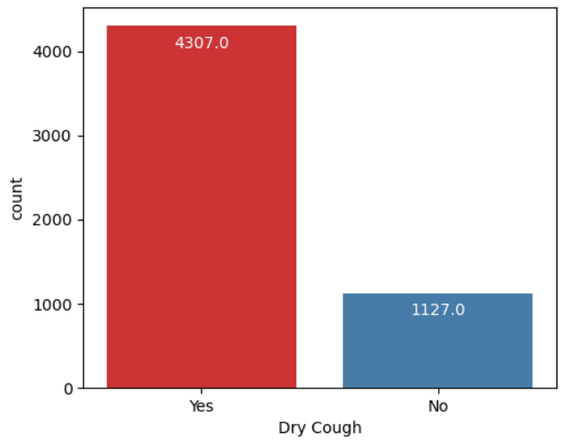


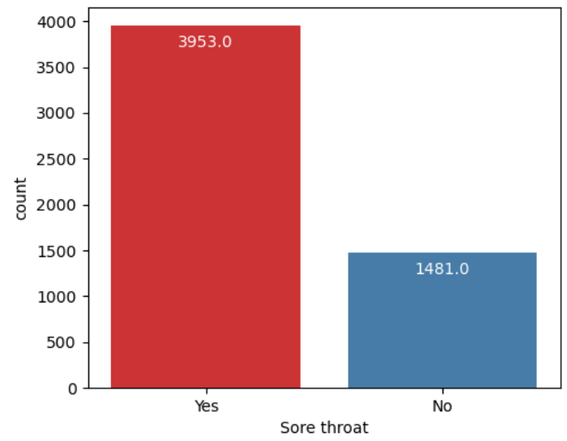
In [64]:
```
columns = ['Breathing Problem', 'Fever', 'Dry Cough', 'Sore throat',
       'Running Nose', 'Asthma', 'Chronic Lung Disease', 'Headache',
       'Heart Disease', 'Diabetes', 'Hyper Tension', 'Fatigue ',
       'Gastrointestinal ', 'Abroad travel', 'Contact with COVID Patient',
       'Attended Large Gathering', 'Visited Public Exposed Places',
       'Family working in Public Exposed Places', 'Wearing Masks',
       'Sanitization from Market']
```

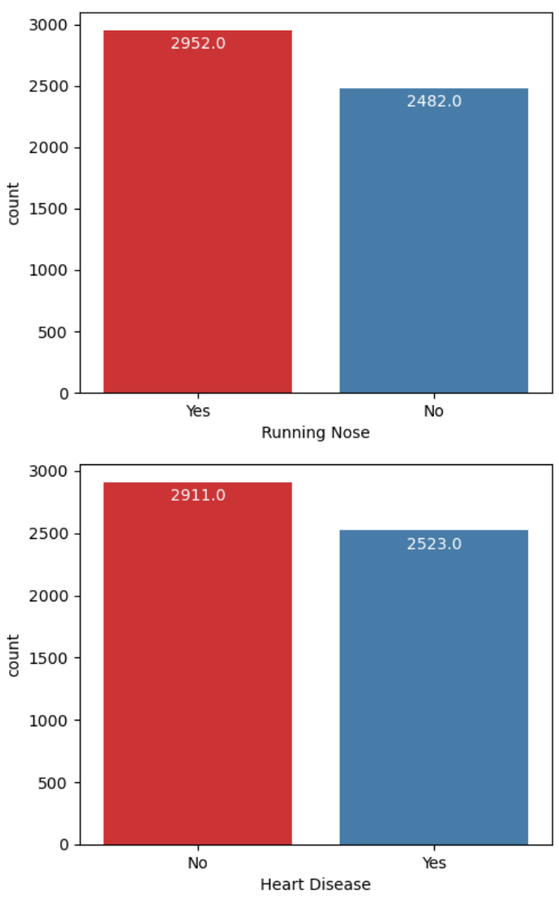
In [65]:
```
fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(20, 20))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Plot countplots for each column
for i, column in enumerate(columns[:-2]):  # Exclude the last two columns
    sns.countplot(x=column, data=covid_data, palette="Set1", ax=axes[i])
    for p in axes[i].patches:
        axes[i].annotate(f'\n{p.get_height()}', (p.get_x()+0.4, p.get_height()+100), ha='center', va='top', color='white', size=10)

# Hide the extra subplots
for j in range(len(columns[:-2]), len(axes)):
    axes[j].axis('off')

# Adjust layout
plt.tight_layout()
plt.show()
```

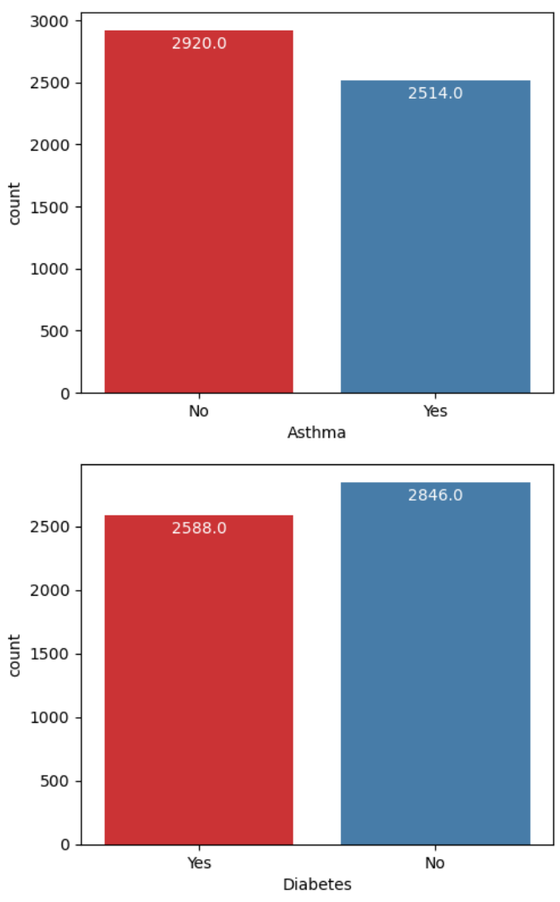


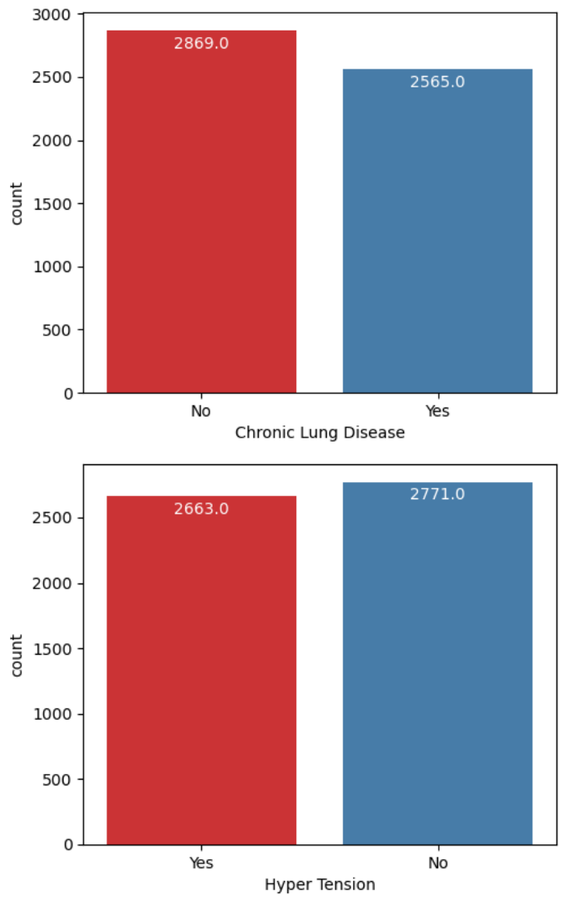


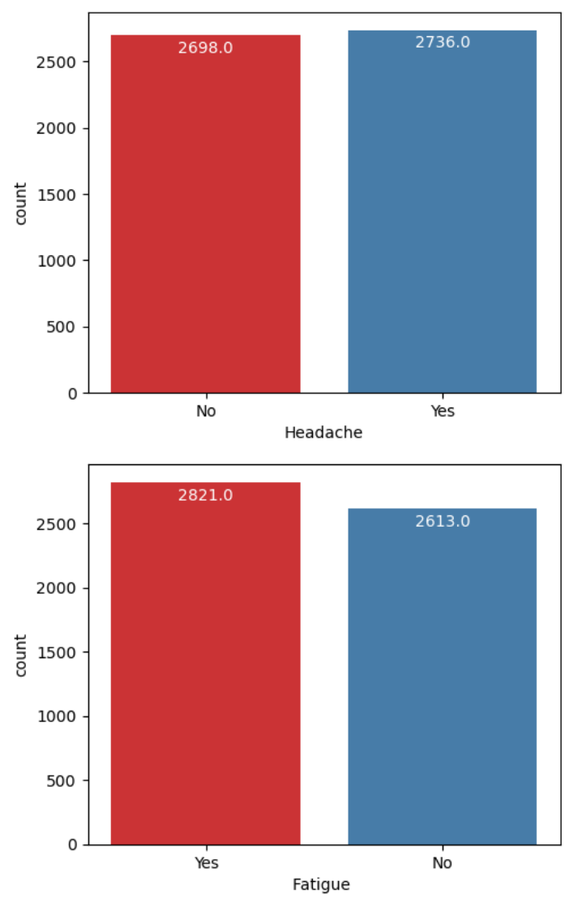


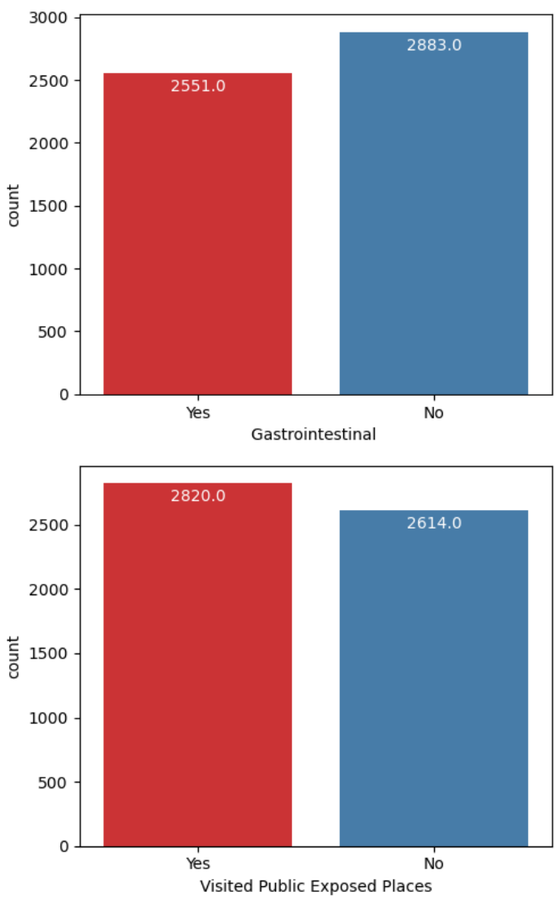


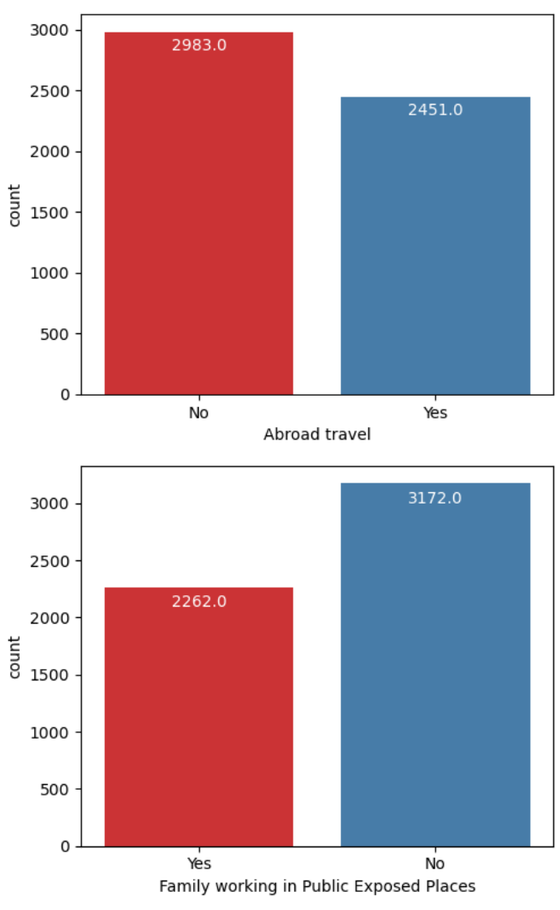


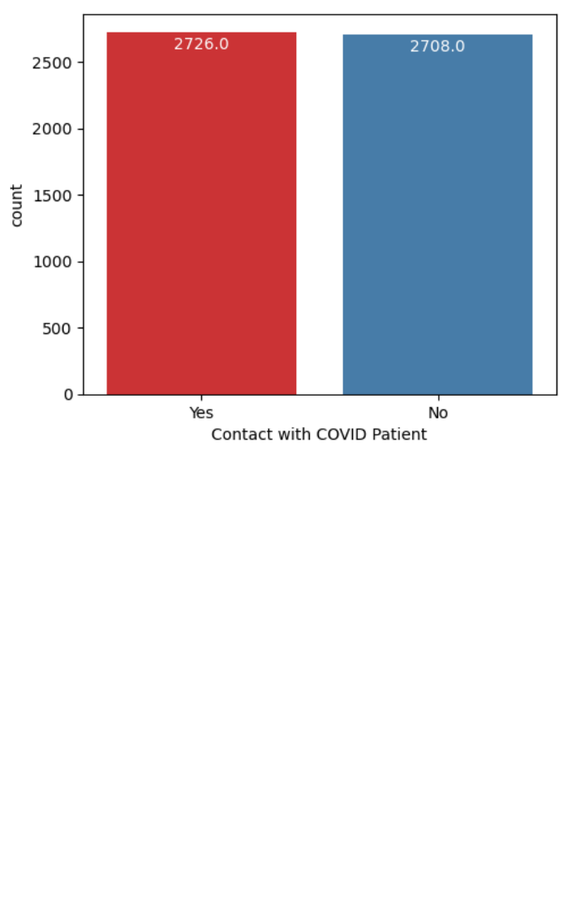


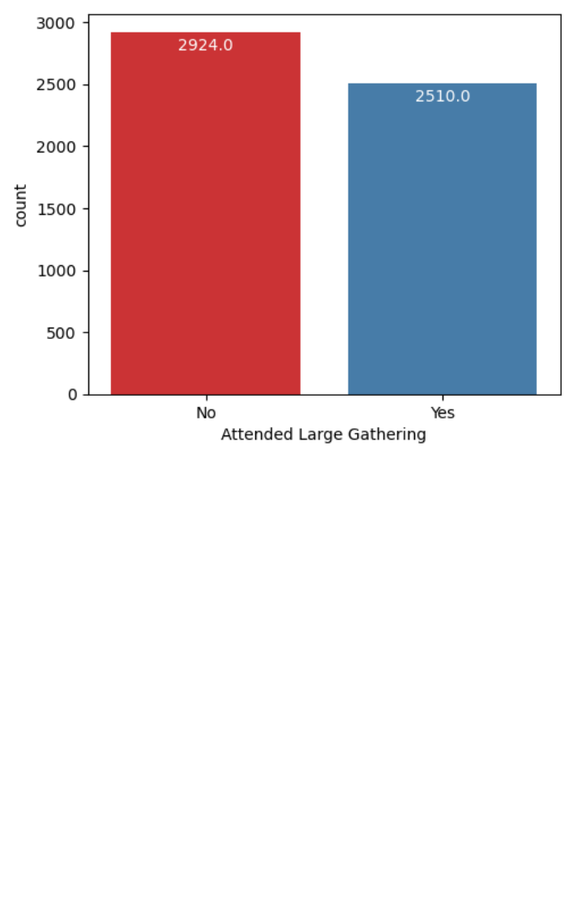




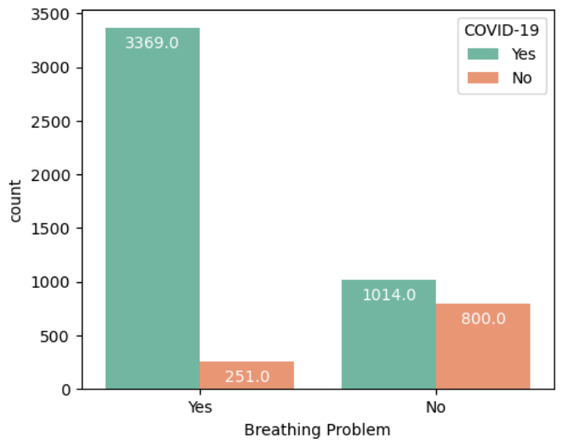


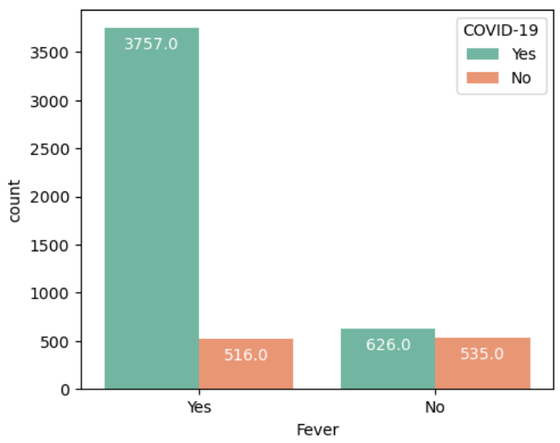


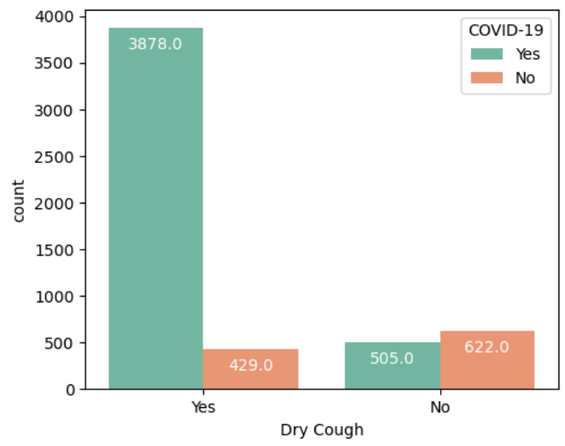


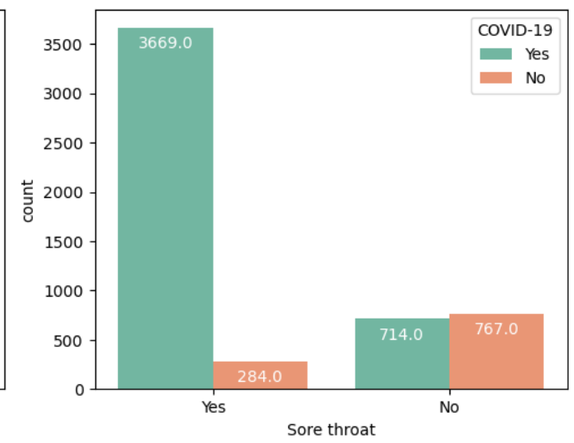


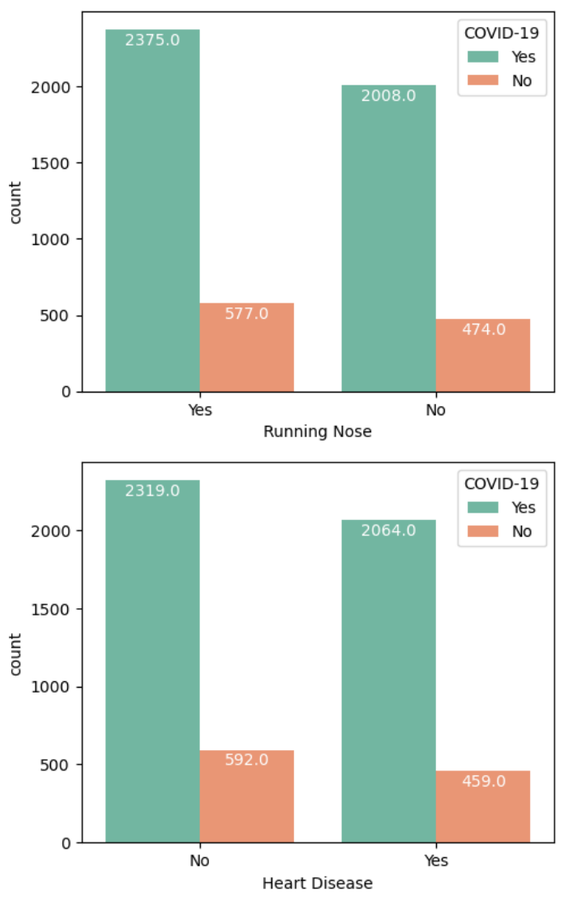
In [66]:
```
fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(20, 20))

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Plot countplots for each column
for i, column in enumerate(columns):
    sns.countplot(x=column, hue='COVID-19', data=covid_data, palette="Set2", ax=axes[i])
    for p in axes[i].patches:
        axes[i].annotate(f'\n{p.get_height()}', (p.get_x()+0.2, p.get_height()+100), ha='center', va='top', color='white', size=10)

# Hide the extra subplots
for j in range(len(columns), len(axes)):
    axes[j].axis('off')

# Adjust layout
plt.tight_layout()
plt.show()
```

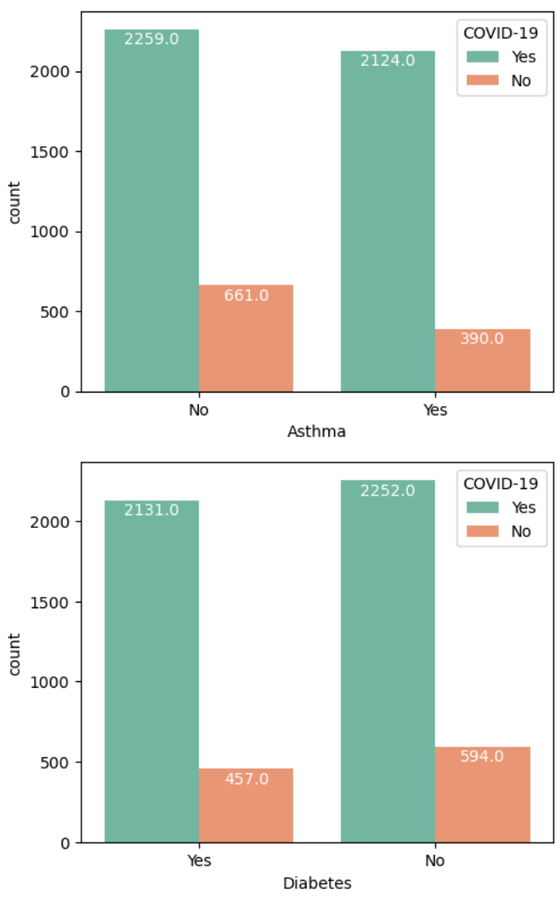


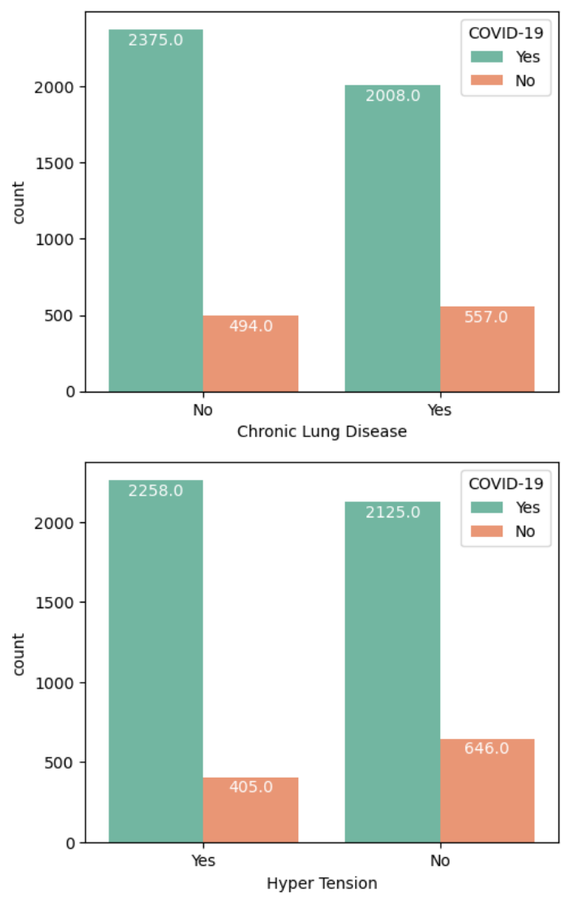


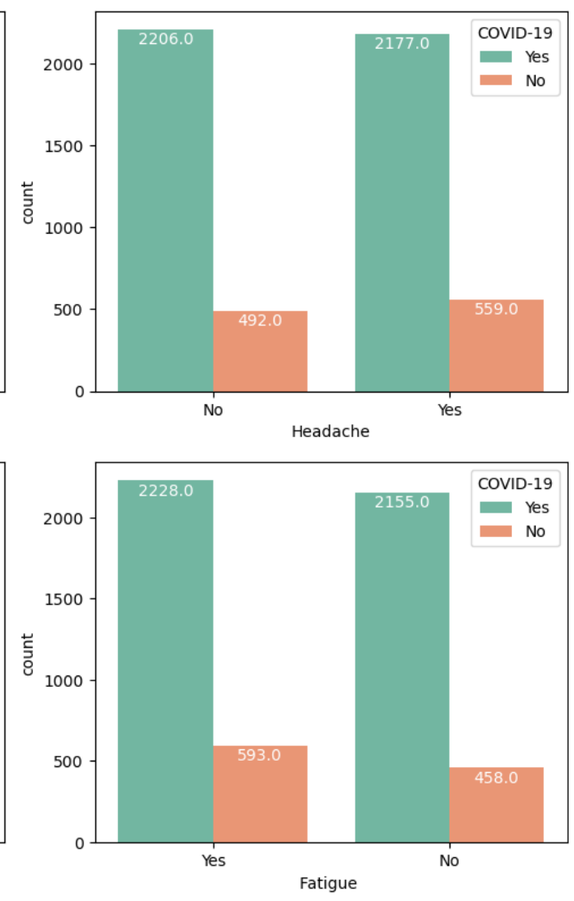


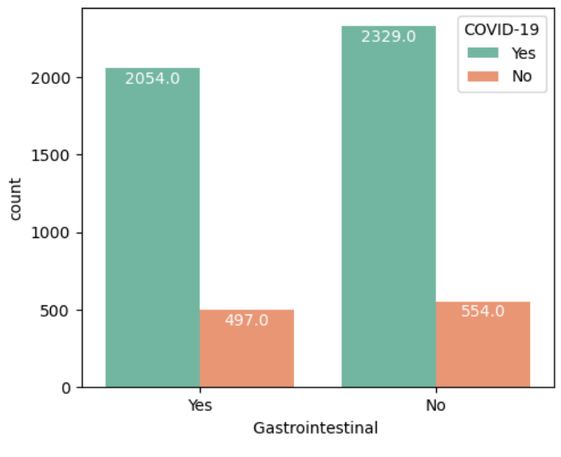


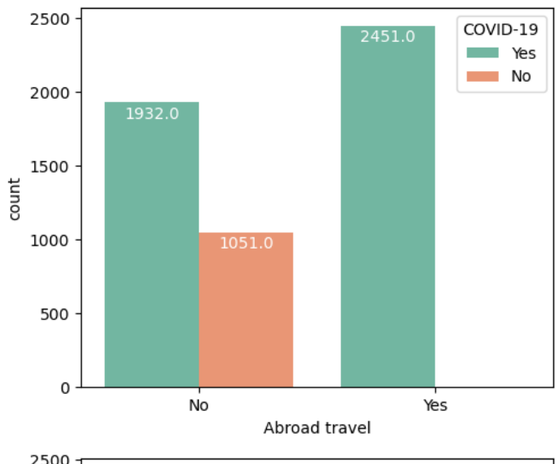


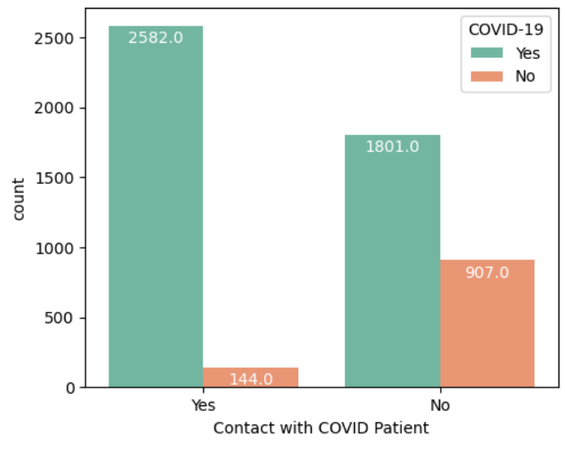


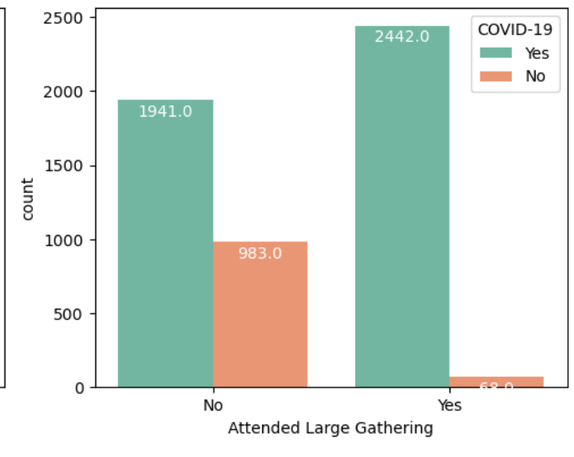


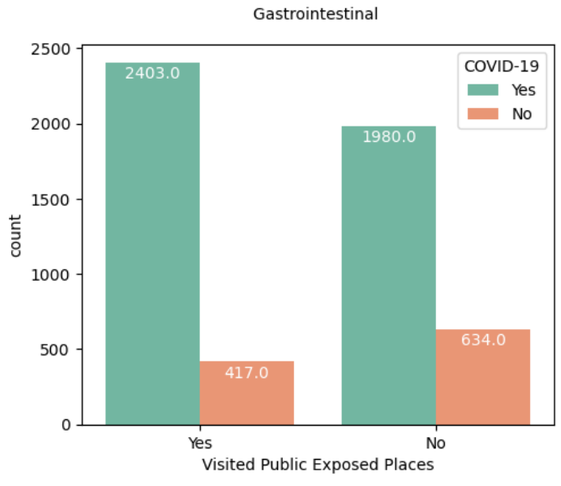


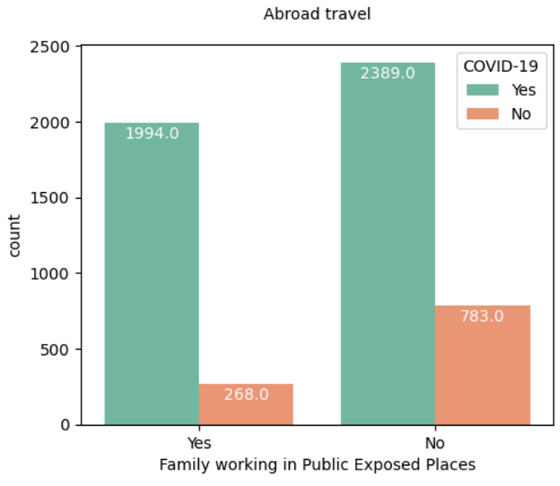


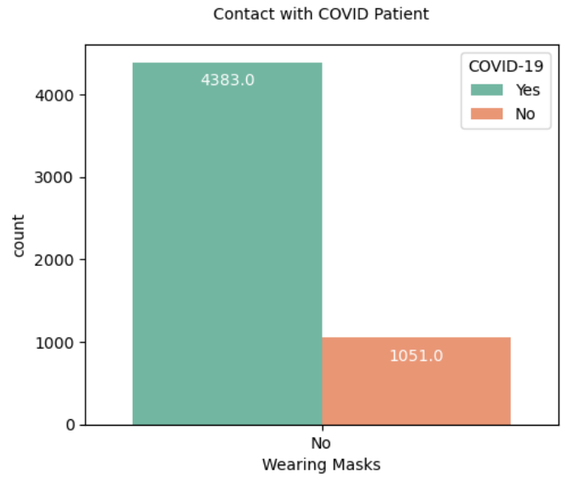


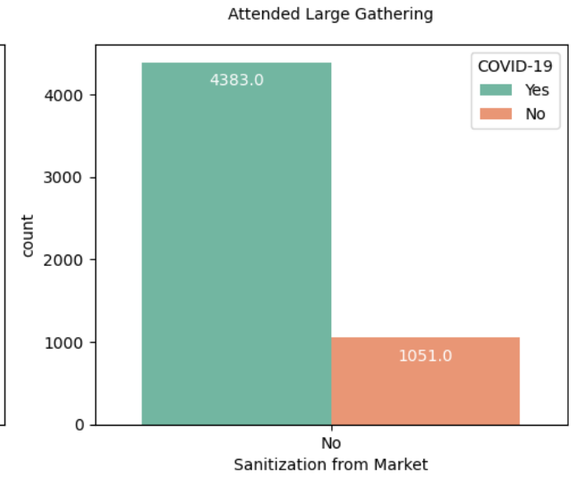












In [67]:
```
columns_to_encode = ['Breathing Problem', 'Fever', 'Dry Cough', 'Sore throat',
                     'Running Nose', 'Asthma', 'Chronic Lung Disease', 'Headache',
                     'Heart Disease', 'Diabetes', 'Hyper Tension', 'Abroad travel',
                     'Contact with COVID Patient', 'Attended Large Gathering',
                     'Visited Public Exposed Places', 'Family working in Public Exposed Places',
                     'Wearing Masks', 'Sanitization from Market', 'Gastrointestinal ', 'Fatigue ']

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply Label Encoding to each column
for column in columns_to_encode:
    covid_data[column] = label_encoder.fit_transform(covid_data[column])

# Encode the 'COVID-19' column separately if needed
covid_data['COVID-19'] = label_encoder.fit_transform(covid_data['COVID-19'])
```

In [68]:
```
covid_data.head()
```

Out[68]:

	Breathing Problem	Fever	Dry Cough	Sore throat	Running Nose	Asthma	Chronic Lung Disease	Headache	Heart Disease	Diabetes	...	Fatigue	Gastrointestinal	Abroad travel	Contact with COVID Patient	Attended Large Gathering	Visited Public Exposed Places	Family working in Public Exposed Places	Wearing Masks	Sanitization from Market	COVID-19
0	1	1	1	1	1	0	0	0	0	1	...	1	1	0	1	0	1	1	0	0	1
1	1	1	1	1	0	1	1	1	0	0	...	1	0	0	0	1	1	0	0	0	1
2	1	1	1	1	1	1	1	1	0	1	...	1	1	1	0	0	0	0	0	0	1
3	1	1	1	0	0	1	0	0	1	1	...	0	0	1	0	1	1	0	0	0	1
4	1	1	1	1	1	0	1	1	1	1	...	0	1	0	1	0	1	0	0	0	1

5 rows × 21 columns

In [69]:
```
pd.set_option('display.max_columns', None)
covid_data
```

Out[69]:

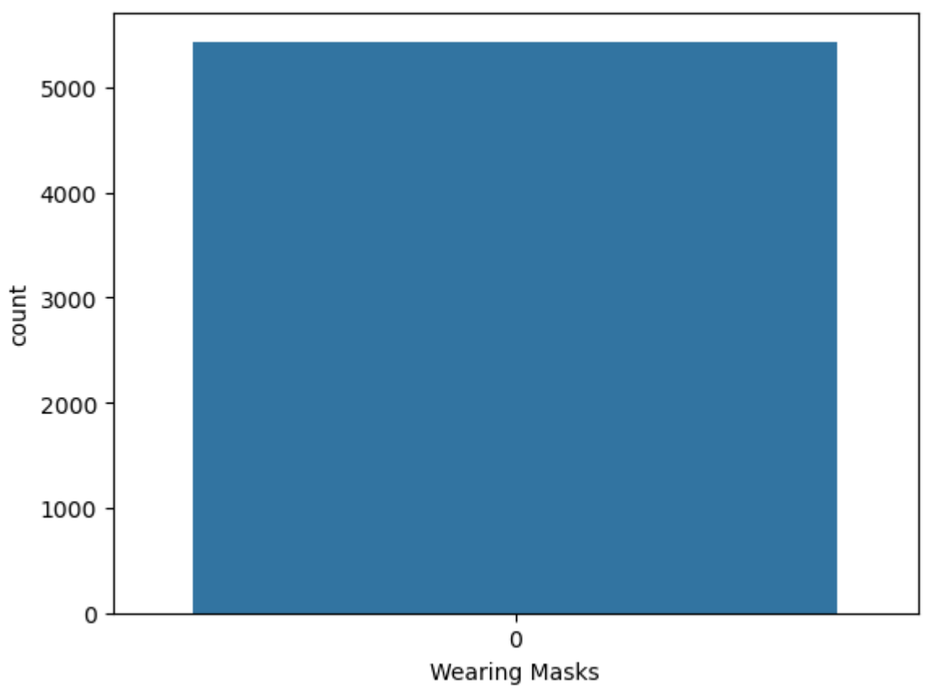
	Breathing Problem	Fever	Dry Cough	Sore throat	Running Nose	Asthma	Chronic Lung Disease	Headache	Heart Disease	Diabetes	Hyper Tension	Fatigue	Gastrointestinal	Abroad travel	Contact with COVID Patient	Attended Large Gathering	Visited Public Exposed Places	Family working in Public Exposed Places	Wearing Masks	Sanitization from Market	COVID-19
0	1	1	1	1	1	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
1	1	1	1	1	0	1	1	1	0	0	0	1	0	0	0	1	1	0	0	0	1
2	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0	0	0	0	0	0	1
3	1	1	1	0	0	1	0	0	1	1	0	0	0	1	0	1	1	0	0	0	1
4	1	1	1	1	1	0	1	1	1	1	1	0	1	0	1	0	1	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5429	1	1	0	1	1	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	1
5430	1	1	1	0	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	1
5431	1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
5432	1	1	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
5433	1	1	1	0	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0

5434 rows × 21 columns

In [70]:
```
print(covid_data['Wearing Masks'].value_counts())
sns.countplot(x='Wearing Masks',data=covid_data)
```

```
Wearing Masks
0    5434
Name: count, dtype: int64
<Axes: xlabel='Wearing Masks', ylabel='count'>
```

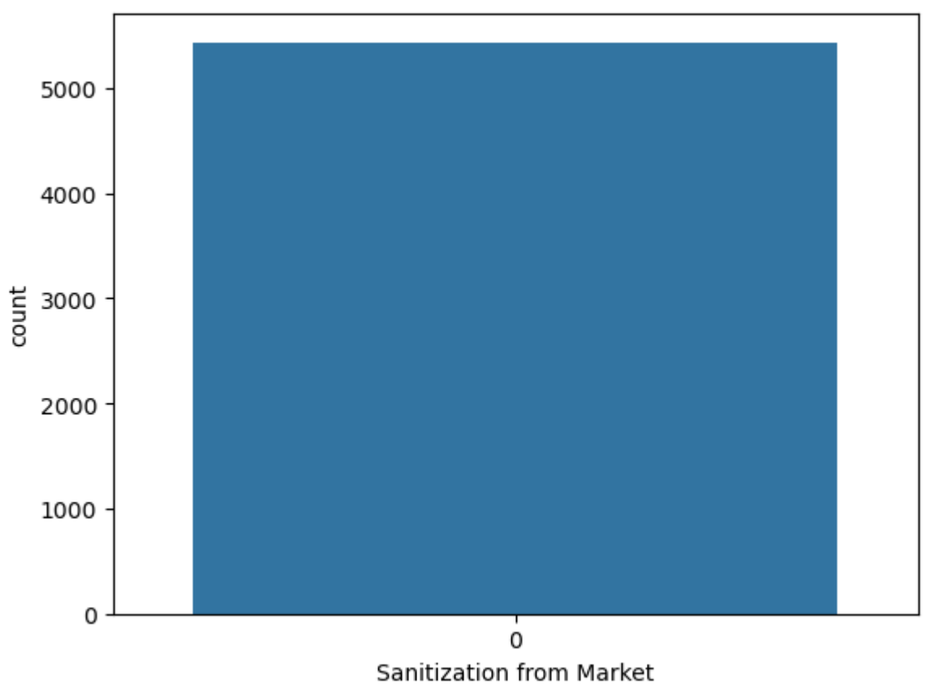
Out[70]:



In [71]:
```
print(covid_data['Sanitization from Market'].value_counts())
sns.countplot(x='Sanitization from Market',data=covid_data)
```

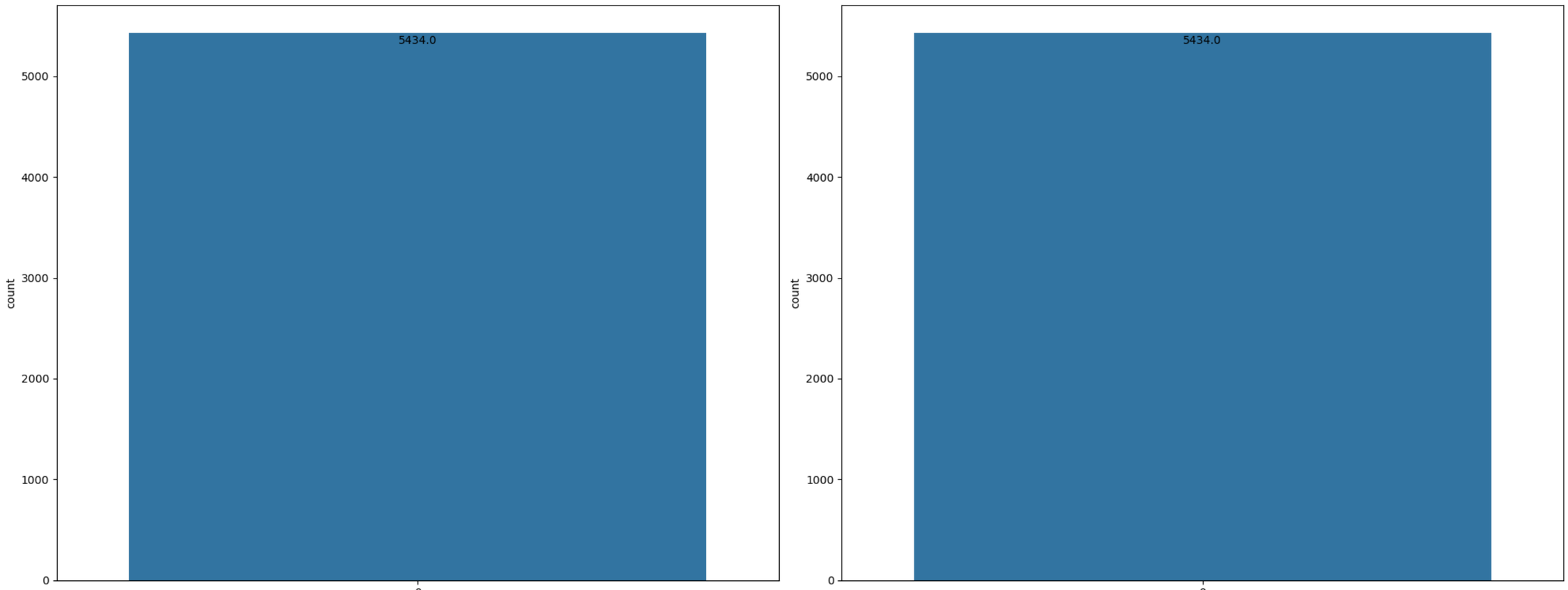
```
Sanitization from Market
0    5434
Name: count, dtype: int64
<Axes: xlabel='Sanitization from Market', ylabel='count'>
```

Out[71]:



In [72]:
```
columns_to_plot = ['Wearing Masks', 'Sanitization from Market']

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=len(columns_to_plot), figsize=(20, 8))

# Plot countplots for each column
for i, column in enumerate(columns_to_plot):
    # Plot value counts
    print(covid_data[column].value_counts())
    sns.countplot(x=column, data=covid_data, ax=axes[i])

    # Optionally, annotate the bars with counts
    for p in axes[i].patches:
        axes[i].annotate(f'\n{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height() + 100),
                         ha='center', va='top', color='black', size=10)
covid_data = covid_data.drop(columns_to_plot, axis=1)
# Show the plots
plt.tight_layout()
plt.show()
```

```
Wearing Masks
0    5434
Name: count, dtype: int64
Sanitization from Market
0    5434
Name: count, dtype: int64
```



In [73]:
```
covid_data.columns
```

Out[73]:
```
Index(['Breathing Problem', 'Fever', 'Dry Cough', 'Sore throat',
       'Running Nose', 'Asthma', 'Chronic Lung Disease', 'Headache',
       'Heart Disease', 'Diabetes', 'Hyper Tension', 'Fatigue ',
       'Gastrointestinal ', 'Abroad travel', 'Contact with COVID Patient',
       'Attended Large Gathering', 'Visited Public Exposed Places',
       'Family working in Public Exposed Places', 'COVID-19'],
      dtype='object')
```

In [75]:
```
right_triangle = np.tril(covid_data.corr())

plt.figure(figsize=(30, 20))

sns.heatmap(covid_data.corr(), vmin=-1, vmax=1, annot=True, square=True, fmt='0.3f',
            annot_kws={'size':10}, cmap="Spectral", mask=right_triangle)



plt.show()
```

In [76]:
```
x=covid_data.drop('COVID-19',axis=1)
y=covid_data['COVID-19']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20, random_state = 101)
```

In [77]:
```
accuracies = {}
algo_time={}
r2_scores={}
mean_squared_errors={}
roc_auc_scores={}
```

Performance Matrix of Classification model :

In [78]:
```
def print_performance2(yt,clf,clf_name):
    y_pred=clf.predict(x_test)
    roc_auc_scores[clf_name]=roc_auc_score(yt,y_pred)*100
    mean_squared_errors[clf_name]=mean_squared_error(yt,y_pred)*100
    r2_scores[clf_name]=r2_score(yt,y_pred)*100
    accuracies[clf_name]=clf.score(x_train,y_train)*100
    print('ROC_AUC value :',roc_auc_scores[clf_name],"%",'\n')
    print("Mean Squared Error :",mean_squared_errors[clf_name],"%")
    print("\nR2 score is :",r2_scores[clf_name],"%")
    print("\nAccuracy Score :",accuracies[clf_name],"%")
    print('\nClassification Report : ','\n',classification_report(yt,y_pred))

    confusionmatrix=confusion_matrix(yt,y_pred)

    fig, ax = plt.subplots(figsize=(3, 3))
    ax.matshow(confusionmatrix, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(confusionmatrix.shape[0]):
        for j in range(confusionmatrix.shape[1]):
            ax.text(x=j, y=i,s=confusionmatrix[i, j], va='center', ha='center', size='xx-large')

    plt.xlabel('Predictions', fontsize=18)
    plt.ylabel('Actuals', fontsize=18)
    plt.title('Confusion Matrix', fontsize=18)
```

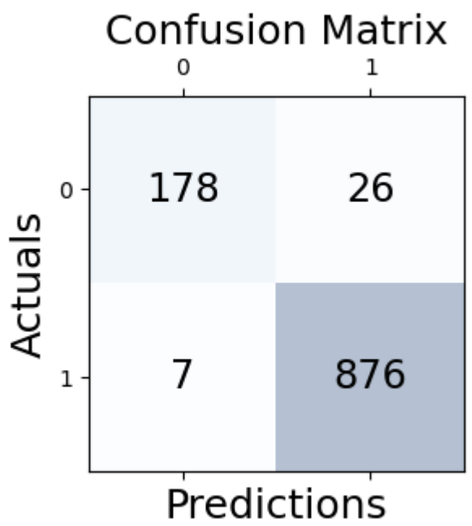
Performance Matrix for DNN(perceptron) :

In [79]:
```
def print_performance_nn(yt,clf,clf_name):
    y_pred=clf.predict(x_test)
    roc_auc_scores[clf_name]=roc_auc_score(yt,y_pred)*100
    mean_squared_errors[clf_name]=mean_squared_error(yt,y_pred)*100
    r2_scores[clf_name]=r2_score(yt,y_pred)*100
    _, accuracies[clf_name]=model.evaluate(x_test,y_test)
    print('ROC_AUC value :',roc_auc_scores[clf_name],"%",'\n')
    print("Mean Squared Error :",mean_squared_errors[clf_name],"%")
    print("\nR2 score is :",r2_scores[clf_name],"%")
    print("\nAccuracy Score :",accuracies[clf_name],"%")
```

LOGISTIC REGRESSION

In [85]:
```
print("LOGISTIC REGRESSION")
start = time.time()
lr = LogisticRegression()
lr.fit(x_train, y_train)
end = time.time()

print_performance2(y_test,lr,'LOGISTIC REGRESSION')
#acc = lr.score(x_train, y_train)*100
#accuracies['LOGISTIC REGRESSION'] = acc
algo_time['LOGISTIC REGRESSION']=end-start
```

```
LOGISTIC REGRESSION
ROC_AUC value : 93.23107498945218 %

Mean Squared Error : 3.035878576338291 %

R2 score is : 80.08627044011503 %

Accuracy Score : 97.03243616287095 %

Classification Report :
               precision    recall  f1-score   support

         0.0       0.96      0.87      0.92       204
         1.0       0.97      0.99      0.98       883

    accuracy                           0.97      1087
   macro avg       0.97      0.93      0.95      1087
weighted avg       0.97      0.97      0.97      1087
```



K-NEAREST NEIGHBOURS

In [86]:
```
start = time.time()
knn = KNeighborsClassifier()
# assigning the dictionary of variables whose optimium value is to be retrieved
param_grid = {'n_neighbors' : np.arange(1,50)}
knn_cv = GridSearchCV(knn, param_grid, cv=5)
# training the model with the training data and best parameter
knn_cv.fit(x_train,y_train)
end=time.time()
algo_time['K-NEAREST NEIGHBOURS']=end-start
```

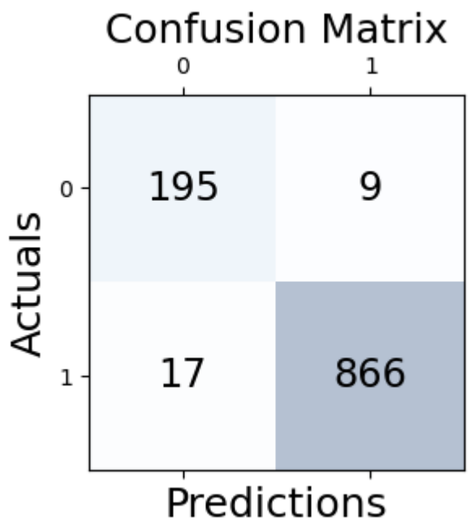
In [87]:
```
# finding out the best parameter chosen to train the model
print("The best paramter we have is: {}" .format(knn_cv.best_params_))

# finding out the best score the chosen parameter achieved
print("The best score we have achieved is: {}" .format(knn_cv.best_score_))
```

```
The best paramter we have is: {'n_neighbors': 2}
The best score we have achieved is: 0.9804454849675277
```

In [88]:
```
print("K-NEAREST NEIGHBOURS")
print_performance2(y_test,knn_cv,'K-NEAREST NEIGHBOURS')
```

```
#acc = knn_cv.score(x_train, y_train)*100
#accuracies['K-NEAREST NEIGHBOURS'] = acc
```

```
K-NEAREST NEIGHBOURS
ROC_AUC value : 96.83149024049031 %

Mean Squared Error : 2.391904406249523 %

R2 score is : 84.31039489221183 %

Accuracy Score : 98.20565907522429 %

Classification Report :
              precision    recall  f1-score   support

         0.0       0.92      0.96      0.94       204
         1.0       0.99      0.98      0.99       883

    accuracy                           0.98      1087
   macro avg       0.95      0.97      0.96      1087
weighted avg       0.98      0.98      0.98      1087
```



RANDOM FOREST

In [89]:
```
import warnings
warnings.filterwarnings('ignore')
```

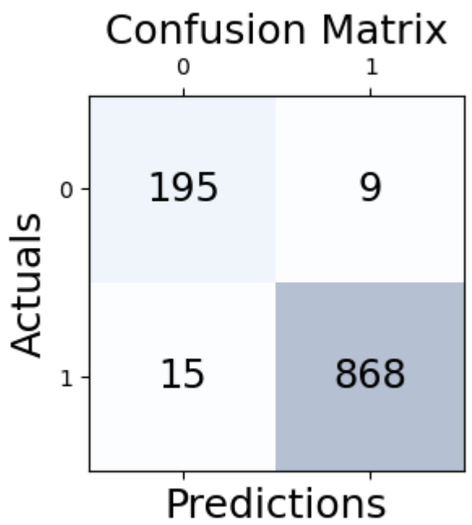
In [90]:
```
rf_start=time.time()
rfc=RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x_train, y_train)
rf_end=time.time()
algo_time['RANDOM FOREST TREE']=rf_end-rf_start
```

In [91]:
```
# finding out the best parameter chosen to train the model
print("The best paramter we have is: {}" .format(CV_rfc.best_params_))

# finding out the best score the chosen parameter achieved
print("The best score we have achieved is: {}" .format(CV_rfc.best_score_*100))
```

```
The best paramter we have is: {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 200}
The best score we have achieved is: 98.13652897371796
```

In [92]:
```
print("RANDOM FOREST TREE")
print_performance2(y_test,CV_rfc,'RANDOM FOREST TREE')
#acc = CV_rfc.score(x_train, y_train)*100
#accuracies['RANDOM FOREST TREE'] = acc
```

```
RANDOM FOREST TREE
ROC_AUC value : 96.94474052361602 %

Mean Squared Error : 2.2079117596149445 %

R2 score is : 85.51728759281093 %

Accuracy Score : 98.38969404186795 %

Classification Report :
              precision    recall  f1-score   support

         0.0       0.93      0.96      0.94       204
         1.0       0.99      0.98      0.99       883

    accuracy                           0.98      1087
   macro avg       0.96      0.97      0.96      1087
weighted avg       0.98      0.98      0.98      1087
```



xgb :

In [93]:
```
import xgboost as xgb

xgb_model = xgb.XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    use_label_encoder=True,
    n_estimators=1000,  # Reduced number of trees
    max_depth=10        # Reduced depth of each tree
)
xgb_model.fit(x_train, y_train)

# Evaluate the model
```

```
accuracy = xgb_model.score(x_test, y_test)
accuracy
```

Out[93]:
```
0.9751609935602575
```

DNN Preceptron :

In [80]:
```
x_train = x_train.astype('float32')
y_train = y_train.astype('float32')
x_test = x_test.astype('float32')
y_test = y_test.astype('float32')
```

In [82]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
# Neural network architecture
model = Sequential([
    Dense(128, input_dim=x_train.shape[1], activation='relu'),
    Dropout(0.3),  # Dropout layer after the first Dense layer
    Dense(64, activation='relu'),
    Dropout(0.5),  # Dropout layer after the second Dense layer
    Dense(32, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.4),  # Dropout layer after the second Dense layer

    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])
model.fit(x_train, y_train, epochs=35, batch_size=32, validation_split=0.2)
loss, accuracy = model.evaluate(x_test, y_test)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 1s 888us/step - binary_accuracy: 0.8017 - loss: 0.7862 - val_binary_accuracy: 0.9333 - val_loss: 0.3764
Epoch 2/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 475us/step - binary_accuracy: 0.9217 - loss: 0.3591 - val_binary_accuracy: 0.9621 - val_loss: 0.2118
Epoch 3/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 479us/step - binary_accuracy: 0.9534 - loss: 0.2146 - val_binary_accuracy: 0.9655 - val_loss: 0.1460
Epoch 4/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 479us/step - binary_accuracy: 0.9637 - loss: 0.1551 - val_binary_accuracy: 0.9690 - val_loss: 0.1148
Epoch 5/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 487us/step - binary_accuracy: 0.9644 - loss: 0.1302 - val_binary_accuracy: 0.9770 - val_loss: 0.0968
Epoch 6/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 466us/step - binary_accuracy: 0.9773 - loss: 0.0990 - val_binary_accuracy: 0.9736 - val_loss: 0.0873
Epoch 7/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 471us/step - binary_accuracy: 0.9693 - loss: 0.0927 - val_binary_accuracy: 0.9747 - val_loss: 0.0828
Epoch 8/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 472us/step - binary_accuracy: 0.9670 - loss: 0.0953 - val_binary_accuracy: 0.9667 - val_loss: 0.0845
Epoch 9/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 474us/step - binary_accuracy: 0.9810 - loss: 0.0721 - val_binary_accuracy: 0.9747 - val_loss: 0.0760
Epoch 10/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 479us/step - binary_accuracy: 0.9752 - loss: 0.0761 - val_binary_accuracy: 0.9770 - val_loss: 0.0727
Epoch 11/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 469us/step - binary_accuracy: 0.9802 - loss: 0.0658 - val_binary_accuracy: 0.9747 - val_loss: 0.0685
Epoch 12/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 497us/step - binary_accuracy: 0.9781 - loss: 0.0669 - val_binary_accuracy: 0.9770 - val_loss: 0.0663
Epoch 13/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 470us/step - binary_accuracy: 0.9788 - loss: 0.0597 - val_binary_accuracy: 0.9770 - val_loss: 0.0654
Epoch 14/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 472us/step - binary_accuracy: 0.9790 - loss: 0.0638 - val_binary_accuracy: 0.9770 - val_loss: 0.0613
Epoch 15/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 489us/step - binary_accuracy: 0.9792 - loss: 0.0622 - val_binary_accuracy: 0.9770 - val_loss: 0.0600
Epoch 16/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 474us/step - binary_accuracy: 0.9820 - loss: 0.0591 - val_binary_accuracy: 0.9724 - val_loss: 0.0639
Epoch 17/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 481us/step - binary_accuracy: 0.9855 - loss: 0.0495 - val_binary_accuracy: 0.9770 - val_loss: 0.0590
Epoch 18/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 499us/step - binary_accuracy: 0.9834 - loss: 0.0501 - val_binary_accuracy: 0.9770 - val_loss: 0.0589
Epoch 19/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 481us/step - binary_accuracy: 0.9801 - loss: 0.0517 - val_binary_accuracy: 0.9713 - val_loss: 0.0610
Epoch 20/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 482us/step - binary_accuracy: 0.9808 - loss: 0.0516 - val_binary_accuracy: 0.9770 - val_loss: 0.0551
Epoch 21/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 490us/step - binary_accuracy: 0.9818 - loss: 0.0537 - val_binary_accuracy: 0.9770 - val_loss: 0.0572
Epoch 22/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 493us/step - binary_accuracy: 0.9849 - loss: 0.0511 - val_binary_accuracy: 0.9770 - val_loss: 0.0538
Epoch 23/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 503us/step - binary_accuracy: 0.9814 - loss: 0.0527 - val_binary_accuracy: 0.9770 - val_loss: 0.0555
Epoch 24/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 492us/step - binary_accuracy: 0.9832 - loss: 0.0510 - val_binary_accuracy: 0.9747 - val_loss: 0.0543
Epoch 25/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 491us/step - binary_accuracy: 0.9832 - loss: 0.0481 - val_binary_accuracy: 0.9816 - val_loss: 0.0526
Epoch 26/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 480us/step - binary_accuracy: 0.9754 - loss: 0.0568 - val_binary_accuracy: 0.9828 - val_loss: 0.0542
Epoch 27/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 492us/step - binary_accuracy: 0.9830 - loss: 0.0501 - val_binary_accuracy: 0.9782 - val_loss: 0.0535
Epoch 28/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 470us/step - binary_accuracy: 0.9854 - loss: 0.0448 - val_binary_accuracy: 0.9770 - val_loss: 0.0506
Epoch 29/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 473us/step - binary_accuracy: 0.9860 - loss: 0.0480 - val_binary_accuracy: 0.9782 - val_loss: 0.0507
Epoch 30/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 510us/step - binary_accuracy: 0.9830 - loss: 0.0459 - val_binary_accuracy: 0.9770 - val_loss: 0.0511
Epoch 31/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 484us/step - binary_accuracy: 0.9787 - loss: 0.0438 - val_binary_accuracy: 0.9816 - val_loss: 0.0494
Epoch 32/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 497us/step - binary_accuracy: 0.9839 - loss: 0.0404 - val_binary_accuracy: 0.9816 - val_loss: 0.0490
Epoch 33/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 491us/step - binary_accuracy: 0.9841 - loss: 0.0386 - val_binary_accuracy: 0.9770 - val_loss: 0.0499
Epoch 34/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 495us/step - binary_accuracy: 0.9841 - loss: 0.0392 - val_binary_accuracy: 0.9770 - val_loss: 0.0522
Epoch 35/35
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 482us/step - binary_accuracy: 0.9831 - loss: 0.0410 - val_binary_accuracy: 0.9782 - val_loss: 0.0520
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 241us/step - binary_accuracy: 0.9826 - loss: 0.0420
Test Accuracy: 0.977000892162323
```

Decision Tree

In [83]:
```
print_performance_nn(y_test,model,'Perceptron')
```

```
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 707us/step
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 243us/step - binary_accuracy: 0.9826 - loss: 0.0420
ROC_AUC value : 99.78154908622565 %

Mean Squared Error : 1.5028960071504116 %

R2 score is : 90.14181080371722 %

Accuracy Score : 0.977000892162323 %
```

In [99]:
```
accuracies
```

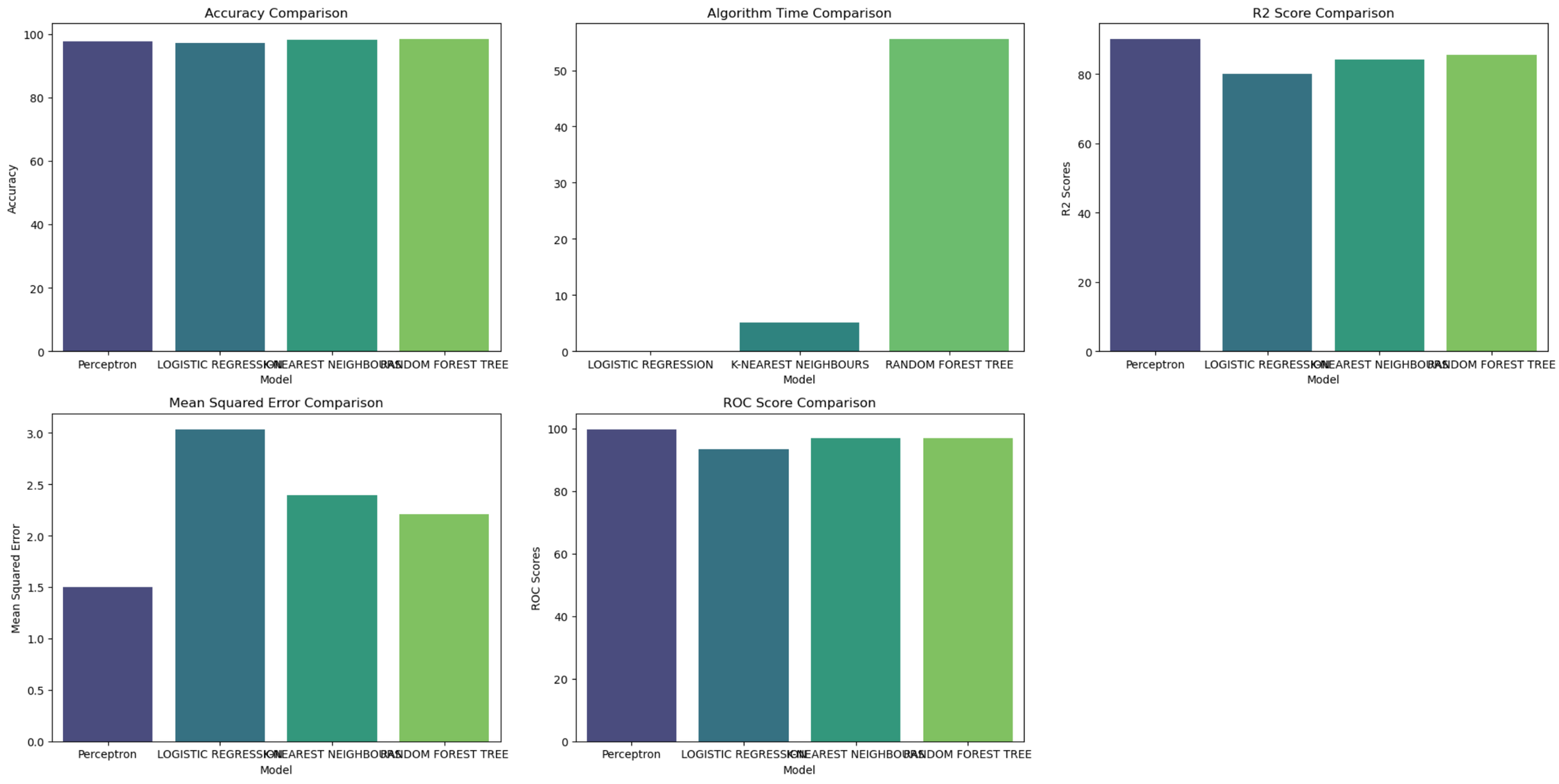
Out[99]:
```
{'Perceptron': 97.7000892162323,
 'LOGISTIC REGRESSION': 97.03243616287095,
 'K-NEAREST NEIGHBOURS': 98.20565907522429,
 'RANDOM FOREST TREE': 98.38969404186795}
```

In [100…
```
# Define the dictionaries containing the data
data_dicts = [
```

```
    ("Accuracy Comparison", accuracies, "Accuracy"),
    ("Algorithm Time Comparison", algo_time, ""),
    ("R2 Score Comparison", r2_scores, "R2 Scores"),
    ("Mean Squared Error Comparison", mean_squared_errors, "Mean Squared Error"),
    ("ROC Score Comparison", roc_auc_scores, "ROC Scores")
]
```

In [101…

```
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(20, 10))  # Adjust figsize to accommodate 5 graphs in a line

# Iterate over data dictionaries and create plots
for i, (title, data_dict, ylabel) in enumerate(data_dicts):
    # Extract keys and values from the data dictionary
    keys = list(data_dict.keys())
    values = list(data_dict.values())

    # Determine the position in the subplot grid
    row = i // 3  # Row index
    col = i % 3   # Column index

    # Create the bar plot using Seaborn
    sns.barplot(x=keys, y=values, ax=axes[row, col], palette="viridis")
    axes[row, col].set_title(title)
    axes[row, col].set_xlabel("Model")
    axes[row, col].set_ylabel(ylabel)

# Hide the empty subplots
for i in range(len(data_dicts), 6):
    row = i // 3
    col = i % 3
    axes[row, col].axis('off')

# Show the plots
plt.tight_layout()
plt.show()
```



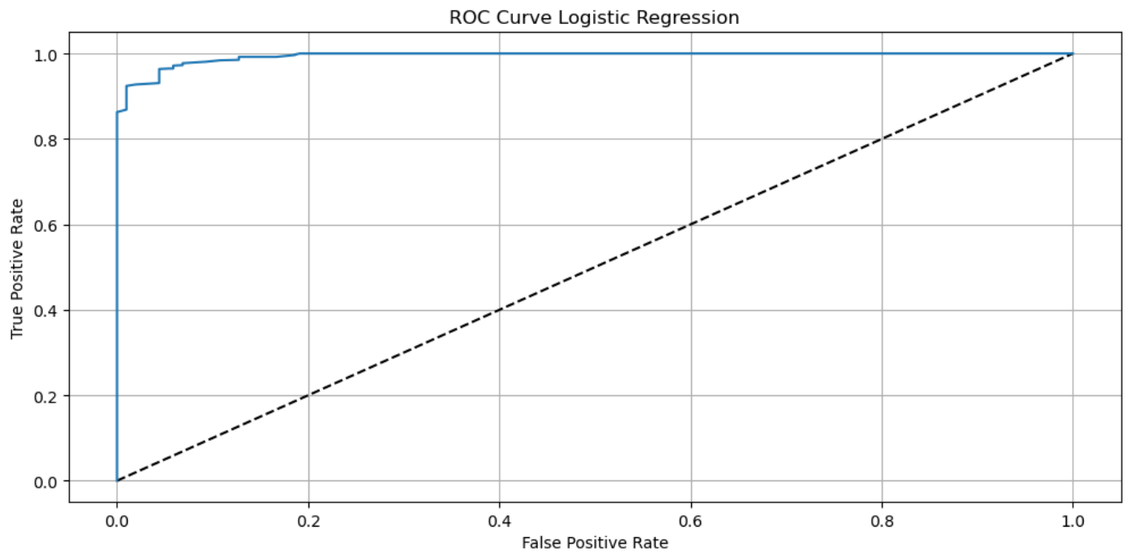
In [105…

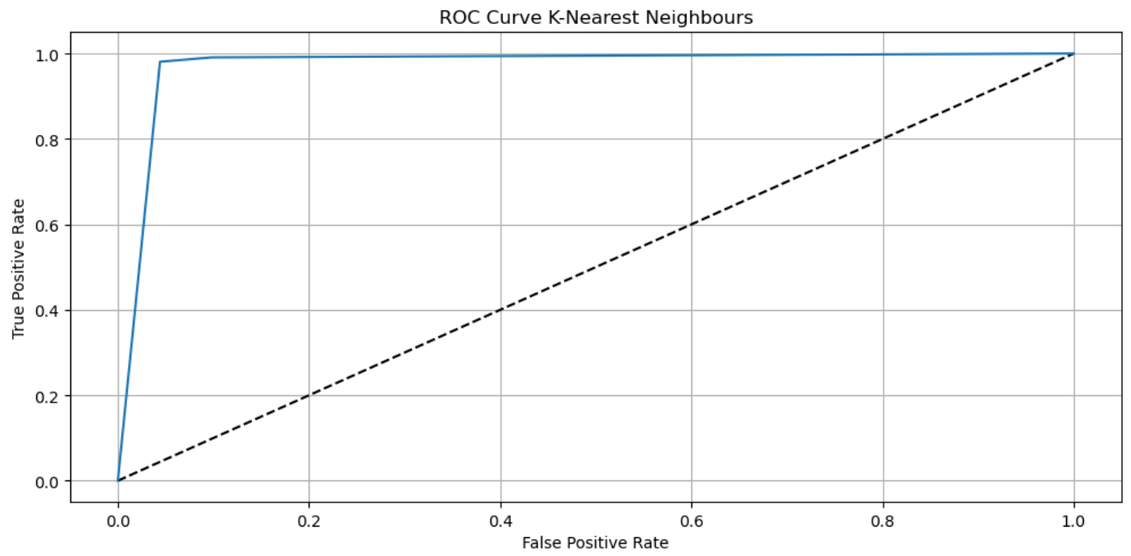
```
Algos=list(roc_auc_scores.keys())

fig = go.Figure(data=[
    go.Bar(name='Accuracies', x=Algos, y=list(accuracies.values())),
    go.Bar(name='R2 scores', x=Algos, y=list(r2_scores.values())),
    go.Bar(name='Mean Squared Errors', x=Algos, y=list(mean_squared_errors.values())),
    go.Bar(name='ROC Auc Scores', x=Algos, y=list(roc_auc_scores.values()))
])
# Change the bar mode
fig.update_layout(barmode='group')
fig.show()
```

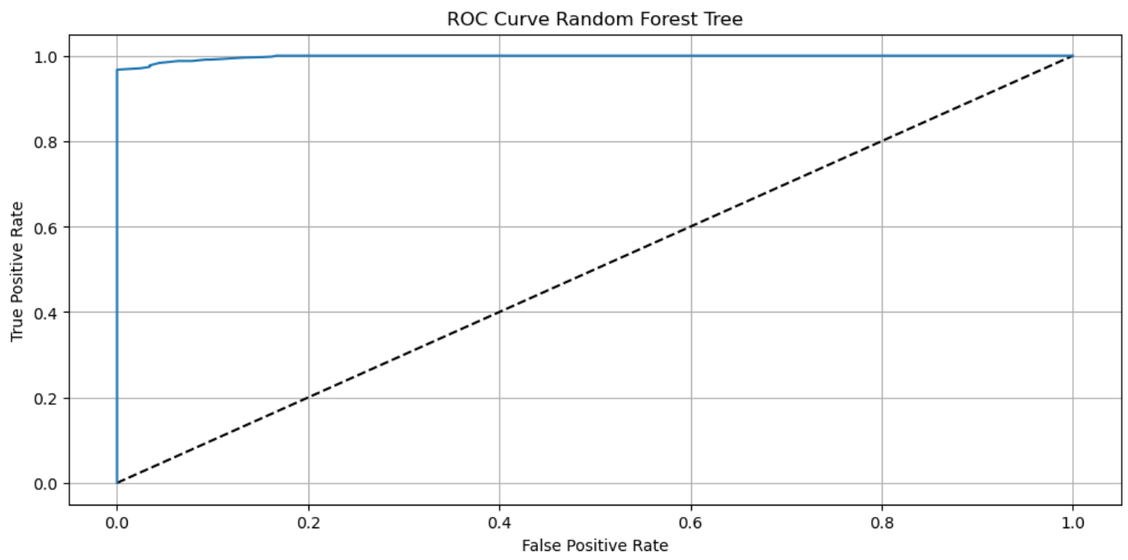
In [106…

```
from sklearn.metrics import roc_curve
#plt.figure(figsize=(25,16))
plt.figure(figsize=(20, 10))

models = [lr, knn_cv, CV_rfc]
model_names = ['Logistic Regression', 'K-Nearest Neighbours', 'Random Forest Tree']

for i, model in enumerate(models, start=1):
```

```
    plt.subplot(2, 2, i)

    # Predict probabilities
    Y_predict_proba = model.predict_proba(x_test)
    Y_predict_proba = Y_predict_proba[:, 1]

    # Calculate ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, Y_predict_proba)

    # Plot ROC curve
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr, label='ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve {model_names[i-1]}')
    plt.grid(True)

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







In [ ]:

```
import colorama
from colorama import Fore

print("COVID PREDICTION BASED ON ML ALGORITHMS")

# Dictionary to store questions and user responses
questions = {
    "Breathing Problem": "Does the patient have breathing problem ?",
    "Fever": "Does the patient have fever ?",
    "Dry Cough": "Does the patient have dry cough ?",
    "Sore Throat": "Does the patient have sore throat ?",
    "Running Nose": "Does the patient have running nose ?",
    "Asthma": "Does the patient have any record of asthma ?",
    "Chronic Lung Disease": "Does the patient have any records of chronic lung disease ?",
    "Headache": "Is the patient having headache ?",
    "Heart Disease": "Does the patient have any record of any heart disease ?",
    "Diabetes": "Does the patient have diabetes ?",
    "Hyper Tension": "Does the patient have hyper tension ?",
    "Fatigue": "Does the patient experience fatigue ?",
    "Gastrointestinal": "Does the patient have any gastrointestinal disorders ?",
    "Abroad Travel": "Has the patient travelled abroad recently ?",
    "Contact with COVID Patient": "Was the patient in contact with a covid patient recently ?",
    "Attended Large Gathering": "Did the patient attend any large gathering event recently ?",
    "Visited Public Exposed Places": "Did the patient visit any public exposed places recently ?",
    "Family Working in Public Exposed Places": "Does the patient have any family member working in public exposed places ?"
}

# Collect user responses
patient_data = {}
for symptom, question in questions.items():
    response = int(input(question + " (Enter 1 for Yes and 0 for No): "))
    patient_data[symptom] = response

# Predict using the model
result = knn_cv.predict([list(patient_data.values())])[0]

# Display result
if result == 1:
    print(Fore.RED + 'You may be affected with COVID-19 virus! Please get RTPCR test ASAP and stay in Quarantine for 14 days!')
else:
    print(Fore.GREEN + 'You do not have any symptoms of COVID-19. Stay home! Stay safe!')
```

Results :

1. Accuracy Improvement:
   - Adding the DNN perceptron model to the COVID-19 test model, together with RF, KNN, and LR, considerably improved its accuracy. The improved model outperformed prior versions in terms of accuracy, showing that it has higher predictive potential for classifying COVID-19 test results.
2. Enhanced Metrics:
   - In addition to accuracy, the changed code significantly increased recall, MSE (mean squared error), and F1 score. This indicates that the modified model not only excelled at correctly detecting positive and negative situations, but also performed better overall across many evaluation criteria.

Observations :

1. Model Diversity:
   - The addition of the DNN perceptron model broadened the model ensemble, allowing it to capture more complex patterns and correlations in the COVID-19 test dataset. This diversification most likely contributed to the improved performance seen across various metrics.
2. Code Standardization Impact:
   - The work to modify the code in accordance with coding standards not only increased readability but may also have contributed to the model's overall enhancement. Clear and consistent code allows for simpler collaboration, maintenance, and future project extension, enabling continual development in the model's performance and reliability.

---

*

Conclusion and Future Direction :

Learnings :

- This research initiative has provided useful insights into the use of machine learning techniques to address public health concerns.
- The study emphasizes the necessity of interdisciplinary collaboration among data scientists, healthcare practitioners, and policymakers in using technology to address healthcare emergencies. Furthermore, the research emphasizes the importance of strong data preparation, feature selection, and model evaluation procedures in ensuring predictive models' reliability and generalizability.

Results Discussion :

- The study's findings suggest that machine learning algorithms can detect COVID-19 with high accuracy, with Random Forest appearing as the most effective option.
- However, these findings must be interpreted in light of the study's limitations, which include dataset biases and model assumptions. Furthermore, the debate emphasizes the importance of constant validation and modification of the prediction model in order to preserve its efficacy in real-world scenarios.

Limitations :

- Despite the hopeful results, the study's limitations should be addressed. These include the use of retrospective data, which might bring biases and mistakes into the predictive model.
- Furthermore, the generalizability of the findings may be limited by the dataset's representativeness and the research population's unique characteristics. Furthermore, the study's emphasis on binary classification of COVID-19 cases may ignore subtle differences in disease severity and prognosis, necessitating additional research.

Future Extension :

- To address the identified limitations and enhance the model's utility, several avenues for future research can be explored. These include the incorporation of longitudinal data to capture temporal trends in COVID-19 transmission and disease progression.
- Additionally, the integration of multimodal data sources, such as clinical imaging and laboratory tests, could provide comprehensive insights into disease dynamics.
- Furthermore, the development of interpretable machine learning models and decision support systems can facilitate their adoption in clinical practice, enabling informed decision-making by healthcare providers. Overall, the study sets the stage for continued innovation and collaboration in leveraging technology to combat public health challenges.

- The paper highlights the potential of machine learning techniques for COVID-19 identification, with promising accuracy rates obtained through thorough experimentation. Moving forward, more study is needed to improve the model's performance and overcome current shortcomings. Future research could look into the use of new data sources, such as genomic sequencing or wearable device data, to improve prediction skills. Furthermore, engagement with healthcare institutions and regulatory organizations is critical for validating the model's clinical value and facilitating its integration into healthcare systems. Furthermore, the model must be continuously monitored and adapted to account for changing virus patterns and variations.

References:

[1]: Batista, A. F. M., Miraglia, J. L., Donato, T. H. R., & Filho, A. D. P. C. (2020). COVID-19 diagnosis prediction in emergency care patients: a machine learning approach. medRxiv. https://www.medrxiv.org/content/10.1101/2020.04.04.20052092v2.full.pdf