Analysis of Adverse Reactions from COVID-19 Vaccines and Study of various Factors contributing to the Side Effects

Apoorv Awasthi[1], Dallas Hutchinson[1], Gayathri Gurram[1], Navya Shruthi Potana[1], Sai Anmisha Doddamreddy[1], Vijayani Bomma[1]

[1]Indiana University-Purdue University, Indianapolis, IN 46202, USA

aawasth@iu.edu, dakhutch@iu.edu, ggurram@iu.edu, npotana@iu.edu, sdoddamr@iu.edu, vbomma@iu.edu

Abstract. This project analyzes the relationships between the various adverse effects and the factors causing the adverse effects in relation to the Covid-19 Vaccine. The goal is to identify the relationships between COVID-19 vaccines and its adverse effects in order to provide safe vaccines to individuals. We noticed through data analysis that elderly people experienced severe side effects. We were able to predict who was at risk of experiencing severe side effects based on age, gender, vaccine manufacturer, and medical history. This analysis could provide appropriate information to health care professionals about the factors to be considered while the administration of different types of vaccines for each individual.

Keywords: Covid-19, adverse effects, severity, symptoms, life threatening, hospitalization, disability, birth defect, death, data analysis, data analysis, data visualization, SQL, Python, machine learning, Tableau.

1. Project Scope

1.1 Introduction

The severe acute respiratory syndrome coronavirus-2 (SARS-CoV-2) has infected millions of people worldwide, triggering the coronavirus disease (COVID-19) outbreak, due to the high incidence and long incubation periods, the spread of this viral disease was very rapid. On December 31, 2019, the World Health Organization (WHO) received the very first confirmation of a potential coronavirus outbreak. Considering the impending threat, several countries took steps to cut transmissions which in turn reduced COVID-19's burden, but this resulted in massive financial losses. As a result, the quest for vaccination against SARS-CoV-2 became the world's top research priority to help restore normalcy. Covid-19 mortality rates continue rising despite the availability of vaccinations from firms like Pfizer & Moderna. The main goal of developing vaccines is to aid a person in fighting a condition yet ensuring that they will not be harmed by any adverse responses that may occur after vaccination. 'Commonly reported symptoms included pruritus, rash, itchy and scratchy sensations in the throat, and mild respiratory symptoms' (Tom Shimabukuro, 2021). 'Rare cases of myocarditis(inflammation of the heart muscle) and pericarditis (inflammation of the outer lining of the heart) in adolescents and young adults have been reported more often after getting the second dose than after the first dose of either the Pfizer-BioNTech or Moderna COVID-19 vaccines.' (Centers For Disease Control, 2021). As a result, broadening the understanding of the components would potentially aid in the development of vaccination with fewer or no side effects.
Is vaccination truly successful in all people? Specific attributes were chosen as data, which again are evaluated to produce statistical and extract information that can then be used to provide insight on anticipating the negative impacts.

1.2 Aim

We aim to determine the association between COVID-19 vaccinations and their adverse effects on offering individuals safe vaccines. We'd research the subjects based on attributes like age, gender, vaccine manufacturer, and past medical history for this. We also want to know if a person would have any major side effects depending on the characteristics mentioned above. We would also like to determine if there is a substantial difference in age between persons who have adverse effects and those who do not.

1.3 Research Questions

1. Is there a relationship between the number of days to show symptoms and the number of symptoms caused by the vaccines?
2. Can we predict the severity of adverse effects based on factors such as age, gender, vaccine manufacturer, and medical history?
3. Is there a significant difference in age between patients who experienced no severe effects and patients who did?

Based on our first research question, our hypothesis is stated below:
Null hypothesis: There is no relationship between the number of days to show symptoms and the number of symptoms caused by the vaccines.
Alternate hypothesis: There is a significant relationship between the number of days to show symptoms and the number of symptoms caused by the vaccines.

Based on our third research question, our hypothesis is stated below:
Null hypothesis: There is no significant difference between patients who experienced no severe effects and patients who did?
Alternate hypothesis: There is a significant difference between patients who experienced no severe effects and patients who did?

1.4 Purpose

This research would help determine how the type of covid vaccine administered and other demographic features of an individual influence the severity of side effects. The Kaggle dataset will help us figure out if any elements lead to adverse effects, and if so, what they are. This study would also help determine which age groups are more prone to experience serious side effects from covid vaccination. This analysis would also ascertain if there is a correlation between the number of days it takes for symptoms to appear after vaccination and the number of symptoms induced by vaccinations. This research would assist healthcare professionals in predicting which individuals would experience severe side effects, allowing the medical provider to make the best decision when administering the vaccine.

2. Methodology

2.1 Steps of the project

The main focus of our project was to compare data on factors that influence the degree of adverse effects using database technologies such as SQL and Python. The tools we have used are Python, Jupyter notebook, phpMyAdmin, SQL and Tableau.

Our project consists of 4 stages, which include:

1. Data Collection
2. Data Extraction and Storage
3. Data Visualization
4. Data Analysis

- Statistical Analysis: Descriptive data analysis and exploratory data analysis, Model Development & Visualization - Heat Map, Histograms, Box plot, Linear regression, Logistic regression, Gradient boost classification, random forest classification. We checked the results of our models using Classification reports and a Confusion matrix. Then checked which model was best using the receiver operating characteristic curve.
- Tools: Microsoft Excel, Python, SQL, Tableau.

Each stage will be discussed in detail further in this report.

2.2 Original Team Members and Responsibilities

Our team consists of people from different backgrounds. Below was the list of team members and tasks assigned to everyone before the commencement of the project.

Name	Background	Responsibility
Apoorv Awasthi	B.Tech in Chemical Engineering	Data Analysis, Data Cleaning, Statistical Analysis, Final Report
Dallas Hutchinson	B.S. in Biobehavioral Health	Project Management, Statistical Analysis, Data Analysis, Final Report
Gayathri Gurram	Pharm-D	Data Cleaning, Data Imputation, Final Report
Navya Shruthi Potana	Pharm-D	Data Analysis, Data Visualization, Final Report
Sai Anmisha Doddamreddy	Bachelors In Dentistry	Data Visualization, Data Collection, Final Report
Vijayani Bomma	Bachelors In Dentistry	Data Collection, Data Cleaning, Final Report

Fig. 1. Project Member's Original Responsibilities

2.3 Actual Contributions from Individual Team Members:

Initially, we did an excellent job of splitting down responsibilities so that we could work as best as possible. However, the above table had few changes as team members were interested in undertaking other tasks after the lecture sessions due to their interest in a specific topic. As a result, the responsibilities were redistributed based on each team member's talents and interests. The following is an updated table of how we worked:

Name	Background	Responsibility
Apoorv Awasthi	B.Tech in Chemical Engineering	Data Cleaning, Statistical Analysis, Data Analysis, Machine Learning integration, Project Presentation, Final Report
Dallas Hutchinson	B.S. in Biobehavioral Health	Project Management, Statistical Analysis, Data Analysis, Machine Learning integration, Project Presentation, Final Report
Gayathri Gurram	Pharm-D	Data Cleaning, Data Visualization, Data Analysis, Proof-reading, Final Report
Navya Shruthi Potana	Pharm-D	Data Collection, Data Imputation, Data Visualization, Final Report
Sai Anmisha Doddamreddy	Bachelors In Dentistry	Statistical Analysis, Data Visualization, Finding and Citing Sources, Final Report
Vijayani Bomma	Bachelors In Dentistry	SQL Joins and Data Import, Data Cleaning, Data Analysis, Final Report

Fig. 2. Project Member's Updated Responsibilities

2.4 Project Challenges:

We encountered certain challenges while working on the project. Data Cleaning was the first hurdle. It was time-consuming. As we moved on to the following step, we noticed there was still a lot of data to clean up based on the project requirement. So, we had allotted the greatest amount of time for data cleaning in order to increase data quality.

We had 3 CSV files, in which 1 CSV was having a large file size. Importing the large file into phpMyAdmin became a bit difficult for our team. We resolved this issue and successfully imported the file by contacting the Teaching Assistant(TA). Identifying the datatypes of each attribute and converting them to the appropriate one was another challenge encountered while working on the project. Because the majority of the attributes were present in different data types. There were few values in our dataset that were insignificant. Limiting those became a challenge for us. We solved this problem by referring to various materials to establish the limits. Running the codes on Jupyterhub took longer at times.

Finally, working on a long project with 6 members is a tough challenge. As everyone has different conflicting schedules, it was hard at times to find times to meet all at once. Our group adapted by splitting some of the project goals down into groups of 2 and conquering tasks that way. This produced quicker results where we all came together and presented what we had been working on for the past week.

3. Data Collection:

The Data set is taken from Kaggle.
(https://www.kaggle.com/ayushggarg/covid19-vaccine-adverse-reactions?select=2021VAERSDATA.csv).

This data set has information regarding the various factors which influence covid-19 vaccination. The VAERS (Vaccine Adverse Event Reporting System) conducts routine reports about vaccines and all associated side effects.

The dataset consists of three CSV files. The files have the following names:

2021VAERSDATA

2021VAERSSYMPTOMS

2021VAERSVAX

4. Data Extraction and Storage

4.1 Data Extraction

Selecting the attributes from our three files was the first step in the data extraction and storage stage. We looked over all of the attributes in each file and chose the ones that best fit.

The attributes we selected are below:

- VAERS_ID
- STATE
- AGE_YRS
- SEX
- DIED
- L_THREAT
- HOSPITAL
- DISABLE
- VAX_DATE
- ONSET_DATE
- NUMDAYS
- HISTORY

- BIRTH_DEFECT
- SYMPTOM1
- SYMPTOM2
- SYMPTOM3
- SYMPTOM4
- SYMPTOM5
- VAX_TYPE
- VAX_MANU

But the most important attributes used for our analysis are listed below:

- AGE_YRS - age of the person.
- SEX - Gender of the person.
- L_THREAT - Column that tells us whether the person got any life-threatening symptom or not
- DIED - Column that tells us whether the person died or not.
- HOSPITAL - Column that tells us whether the person got hospitalized or not.
- BIRTH_DEFECT - Column that tells us whether the person got a birth defect or not.
- DISABLE - Column that tells us whether the person got disabled or not.
- NUMDAYS - No. of days between vaccination and the onset of symptoms.
- VAX_MANU - Vaccine manufacturer name.
- HISTORY - Column that tells us whether the person had any serious medical history or not.
- SYMPTOM_COUNT - Column made up after counting columns containing strings for symptom1,2,3,4 and 5 for a particular VAERS_ID. It tells us about the total no. of symptoms a person suffered after vaccination.

4.2 Data Import:

After the SYMPTOMS_TEXT column was deleted from the VAERS DATA in Microsoft excel, three .csv files were imported to the SQL database and named the data as “I501Fa21grp07_db” in phpMyAdmin which was provided with shared access to all the team members. We imported three csv files with the following attributes VAERS_ID, STATE, AGE_YRS, SEX, DIED, L_THREAT, DISABLE, VAX_DATE, ONSET_DATE, NUMDAYS, OTHER_MEDS, CUR_ILL, HISTORY, BIRTH_DEFECT, RECOVD, ONSET_DATE, ER_VISIT, HOSPITAL, HOSPDAYS, ALLERGIES, SYMPTOM1, SYMPTOM2, SYMPTOM3, SYMPTOM4, SYMPTOM5, VAX_TYPE, VAX_MANU with data types VARCHAR, FLOAT, BOOLEAN, INT. Making the VAERS_ID the Primary key. Some part of data cleaning was performed in SQL such as dropping of rows with vaccines other than COVID-19 using ALTER TABLE queries. Later we imported the data to the Python database in the jupyter hub and completed the remaining part of the cleaning and rest of the project here.

```
myvars = {}
with open("dakhutch-mysql-password") as myfile:
    for line in myfile:
        name, var = line.partition(":")[::2]
        myvars[name.strip()] = var.strip()
myvars.keys()
conn = MySQLdb.connect(host="localhost", user=myvars['DB username'], passwd=myvars['DB password'], db='I501Fa21grp07_db')
cursor = conn.cursor()
cursor.execute('select * from VAERSDATA2');
vaers_data = cursor.fetchall()
cursor.execute('select * from VAERSSYMPTOMS2');
vaers_symptoms = cursor.fetchall()
cursor.execute('select * from VAERSVAX2');
vaers_vax = cursor.fetchall()
cursor.execute('select * from BIRTH_DEFECT');
birth_defect = cursor.fetchall()
```

```
data = pd.DataFrame(vaers_data)
symptoms = pd.DataFrame(vaers_symptoms)
vax = pd.DataFrame(vaers_vax)
birth_defect = pd.DataFrame(birth_defect)
```

Fig. 3. Importing the dataset

4.3 Data Cleaning and modification:

Modifications and Transformation to the dataset will be done. Some of the attributes like V_FUNDBY, LAB_DATA, V_ADMINBY, X_STAY, ALLERGIES were not included for analysis. All blank cells were dropped and duplicate values were determined and removed using VAERS_ID as the unique attribute for the dataset. Unfortunately, the BIRTH_DEFECT column was deleted inadvertently so we have retrieved the column from the original data set. SYMPTOM_COUNT was added as another column(attribute) which was the count of attributes such as SYMPTOM1, SYMPTOM2, SYMPTOM3, SYMPTOM4, SYMPTOM5. Removed irrelevant columns such as ALLERGIES, RECOVD, HOSPDAYS, CUR_ILL, ER_VISIT, etc as they were not concerned with the aim and research questions of our project. Replaced all the null values in num days by “np. nan” and then dropped the rows NA values of Sundays. Also dropped all the rows of data where num days > 45 days considering the window period between the first vaccine administered date and the publication date. And rows of data where the symptoms count is >25 as there is a negligible number of patients who have symptoms more than 25 in our data set considering the toll for analysis of such negligible data we have cut short our target to 25. Few attributes - SEX, HISTORY, BIRTH_DEFECT are given boolean data types based on a valid entry for analysis. Code for data cleaning can be found below:

```
def convert_to_boolean(colnum):
    for i in range (0,34121):
        if vaers_data[i,colnum] == 'Y':
            vaers_data[i,colnum] = 1
        else:
            vaers_data[i,colnum] = 0
```

```
# Convert Y/N attributes to boolean values

# Convert died column
convert_to_boolean(4)
# Convert life threatening column
convert_to_boolean(5)
# Convert er_visit column
convert_to_boolean(6)
# Convert hospital column
convert_to_boolean(7)
# Convert disabled column
convert_to_boolean(9)
```

Fig.4. Converting columns to boolean values

```
# Convert hospdays empty values to 0
for i in range (0,34121):
    if vaers_data[i,8] == '':
        vaers_data[i,8] = 0

# Convert recovered empty values to 'U' or unknown
for i in range (0,34121):
    if vaers_data[i,10] == '':
        vaers_data[i,10] = 'U'

# Convert other_meds column to boolean based on set of input strings
for i in range (0,34121):
    if vaers_data[i,14] == 'None' or vaers_data[i,14] == 'none' or vaers_data[i,14] == 'unknown' or vaers_data[i,14] == 'Unknown'
        vaers_data[i,14] = 0
    else:
        vaers_data[i,14] = 1

# Convert cur_ill column to boolean based on set of input strings
for i in range (0,34121):
    if vaers_data[i,15] == 'None' or vaers_data[i,15] == 'none' or vaers_data[i,15] == 'No' or vaers_data[i,15] == 'no' or vaers_
        vaers_data[i,15] = 0
    else:
        vaers_data[i,15] = 1
```

Fig 5. Converting columns to boolean values based on string inputs

```
# Convert medical history column to boolean based on set of input strings
for i in range (0,34121):
    if (vaers_data[i,16] == 'Comments: List of non-encoded Patient Relevant History: Patient Other Relevant History 1: None' or
    vaers_data[i,16] == 'Medical History/Concurrent Conditions: No adverse event (No reported medical history)' or
    vaers_data[i,16] == 'None' or vaers_data[i,16] == 'none' or vaers_data[i,16] == 'N/A' or
    vaers_data[i,16] == 'n/a' or vaers_data[i,16] == '' or vaers_data[i,16] == 'no' or
    vaers_data[i,16] == 'None.' or vaers_data[i,16] == 'NONE' or vaers_data[i,16] == 'No'):
        vaers_data[i,16] = 0
    else:
        vaers_data[i,16] = 1
```

Fig. 6. Converting medical history column to a boolean value based on common string inputs

```
#dropping allergies, hospdays, recovd, other_meds, cur_ill, er_visit, vax_date, onset_date columns
covid_df = covid_df.drop(['allergies','hospdays','recovd','other_meds','cur_ill','er_visit','vax_date','onset_date'], axis=1)

#replacing null values in numdays by np.nan
covid_df['numdays'] = covid_df['numdays'].replace('', np.nan)

# Remove rows with NA value for numdays
covid_df2 = covid_df2.dropna(subset = ['numdays'])
covid_df2['numdays'] = covid_df2['numdays'].astype('int64')

# Drop all rows of data where numdays > 45. Based on logic from when
# vaccines were released and dataset was published
covid_df2.drop(covid_df2[covid_df2['numdays'] > 45].index, inplace=True)

# Drop all rows of data where symptom count > 25. Based on logic from first hand
# experience on extreme case of vaccine reactions
covid_df2.drop(covid_df2[covid_df2['symptom_count'] > 25].index, inplace=True)
```

Fig. 7. Dropping columns and unnecessary rows

5. Data Analysis

5.1 Getting the Symptom Count variable from the Symptom table

We had 4 tables: vaers_data, vaers_symptoms, vaers_vax and birth_defect. As we have already explained the data cleaning part of vaers_data and vaers_vax, let us look at what we did to get the symptom count for a particular person. This is how the 'vaers_symptoms' table looked like:

VAERS_ID	SYMPTOM1	SYMPTOM2	SYMPTOM3	SYMPTOM4	SYMPTOM5
916600	Dysphagia	Epiglottitis			
916601	Anxiety	Dyspnoea			
916602	Chest discomfort	Dysphagia	Pain in extremity	Visual impairment	
916603	Dizziness	Fatigue	Mobility decreased		
916604	Injection site erythema	Injection site pruritus	Injection site swelling	Injection site warmth	
916606	Pharyngeal swelling				
916607	Abdominal pain	Chills	Sleep disorder		
916608	Diarrhoea	Nasal congestion			
916609	Vaccination site erythema	Vaccination site pruritus	Vaccination site swelling		
916610	Rash	Urticaria			
916611	Blood pressure decreased	Chest pain	Chills	Confusional state	Decreased appetite
916611	Dyspnoea	Fatigue	Feeling abnormal	Head discomfort	Headache
916611	Heart rate decreased	Heart rate increased	Hypertension	Injection site pain	Musculoskeletal chest pain
916611	Nausea	Pain	Pain in extremity	Paraesthesia oral	Pyrexia
916611	SARS-CoV-2 antibody test	SARS-CoV-2 test negative			
916613	Abdominal pain upper	Dizziness	Dysgeusia		
916614	Blood pressure increased	Chest discomfort	Heart rate increased		
916615	Injection site erythema	Injection site pruritus	Injection site swelling	Lymph node pain	Lymphadenopathy
916617	Chills	Dizziness	Injection site pain	Myalgia	Pyrexia
916618	Injection site pain	Pain			
916619	Injection site pain	Menorrhagia			
916620	Arthralgia	Chills	Headache	Mobility decreased	Myalgia
916620	Nausea	Pain in extremity	Pyrexia		
916621	Chills	Fatigue	Headache	Myalgia	
916622	Headache	Heart rate increased	Injection site erythema	Injection site pain	Injection site swelling

Fig. 8. Vaers_symptoms table in phpmyadmin

As we see, there were all the string values in the symptom columns. One more problem was that if a person is having more than 5 symptoms, then that person's VAERS_ID is repeated. So, we had duplicate rows for the same VAERS_ID in many cases. To resolve these issues, we first created a NumPy array of vaers_symptoms. Then, we used for loop and counted for all the string values in all the rows, and thus we obtained the no. of string values in each row. We named this list as count. We appended this count list into our vaers_symptom array as a new column and deleted all other columns except VAERS_ID and symptom count.

```
# Start to clean the second csv file, VAERSSYMPTOMS
vaers_symptoms = np.array(vaers_symptoms)
```

```
# We need to derive the total number of symptoms each patient experiences
# Making count list which counts
# total no. of symptoms for each rows
count = []
for i in range(0,48110):
    x = 0
    for j in range(1,6):
        if vaers_symptoms[i,j] != '':
            x = x+1
    count.append(x)
len(count)
```

```
48110
```

```
# creating count into vertical array
count = np.array(count)
count = np.reshape(count,(48110,1))
count = np.array(count)
```

```
# appending count array to the symptoms array
vaers_symptoms2 = np.append(vaers_symptoms, count, axis = 1)
vaers_symptoms2.shape
```

```
(48110, 7)
```

```
# Deleting string symptoms columns and only keeping the VAERS_ID and Symptom count
vaers_symptoms3 = np.delete(vaers_symptoms2,[1,2,3,4,5],1)
vaers_symptoms3.shape
```

```
(48110, 2)
```

Fig. 9. Adding Count column

Now to resolve the issue of multiple IDs, we created an empty dictionary. Then we added the values and keys in the dictionary where values = 'VAERS_ID' and keys = 'symptom count' based on the vaers_symptoms NumPy array. While adding the values and keys we ensure that if the VAERS_ID value is repeated, then the symptom count gets added to the repeated VAERS_ID. After getting the final dictionary, we convert it into a pandas data frame and name the data frame as symptoms. You can find the code below in Figure 10:

```
# Converting the data in the array to int data type
vaers_symptoms3 = vaers_symptoms3.astype(np.int)

# Creating a dictionary and removing multiple values of VAERS_ID
# and adding the symptom count of rows with same VAERS_ID
dict1 = {}
for i in vaers_symptoms3:
    if i[0] not in dict1:
        dict1[i[0]] = i[1]
    else:
        dict1[i[0]] = dict1[i[0]] + i[1]

# Checking if the length reduced or not
print(len(dict1))
```

```
34121
```

```
# Change symptoms from numpy array to a pandas df
vaers_symptoms4 = np.array(list(dict1.items()))
vaers_symptoms4.shape
symptoms = pd.DataFrame(vaers_symptoms4)
symptoms.head()
```

	0	1
0	916600	2
1	916601	2
2	916602	4
3	916603	3
4	916604	4

Fig. 10. Making new table by name of symptom count

5.2 Data Visualization

After doing the data cleaning and then merging all 4 csv files and removing all the unnecessary rows, we got our final data frame:

```
covid_df.head()
```

	vaers_id	state	age_yrs	sex	died	l_threat	hospital	disable	numdays	history	birth_defect	vax_manu	symptom_count
0	916600	TX	33.0	F	0	0	0	0	2	0	0	MODERNA	2
1	916601	CA	73.0	F	0	0	0	0	0	1	0	MODERNA	2
2	916602	WA	23.0	F	0	0	0	0	0	0	0	PFIZER\BIONTECH	4
3	916603	WA	58.0	F	0	0	0	0	0	1	0	MODERNA	3
4	916604	TX	47.0	F	0	0	0	0	7	0	0	MODERNA	4

Fig. 11. Final data frame after using merge function

We have three continuous attributes - ‘age_yrs’, ‘numdays’, and symptom_count. To understand these variables, we decided to make some visualizations. First, we decided to make a histogram of ‘age_yrs’.

```python
# Age distribution visual
plt.hist(covid_df2['age_yrs'],color="red",bins = 50, alpha=0.7)
plt.title('Age Distribution of Patients')
plt.xlabel('Age (years)')
plt.ylabel('Frequency count')
plt.show()
```

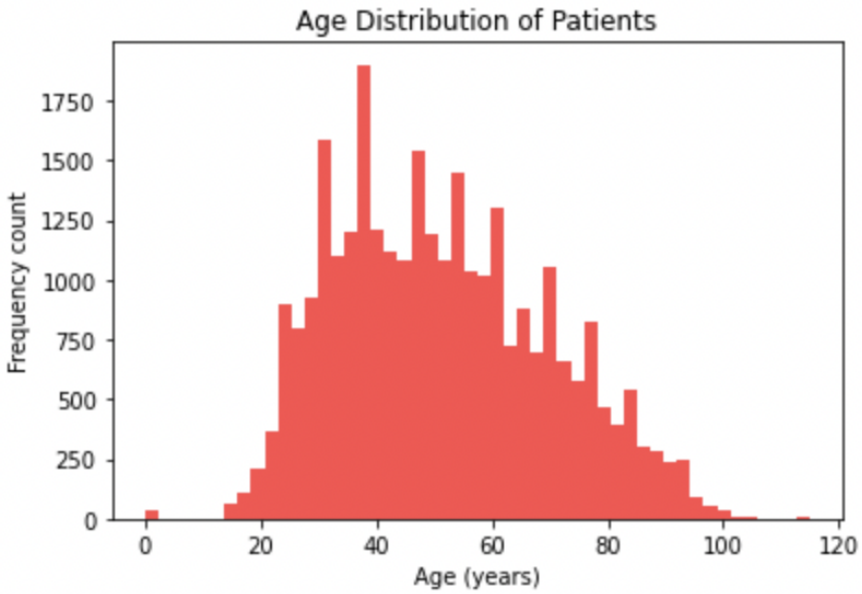

Fig. 12. Histogram of age

As we see in the histogram, the histogram is right-skewed. This tells us that there are more people having age on the higher side which makes sense because the dataset that we have chosen was made in the early days of COVID-19 vaccination and there were many older people who got vaccinated first.

We also made a box plot to check if the symptom count is different for males and females.

```python
# Gender vs. Symptom count boxplot
sns.set(rc={'figure.figsize':(10,10)})
sns.set_style("whitegrid")
sns.boxplot(x='sex', y='symptom_count', data=covid_df2)
plt.title('Distribution of symptom counts for males and females')
plt.xlabel('Sex')
plt.ylabel('Symptom count')
plt.show()
```

Fig. 13. Python code for boxplot

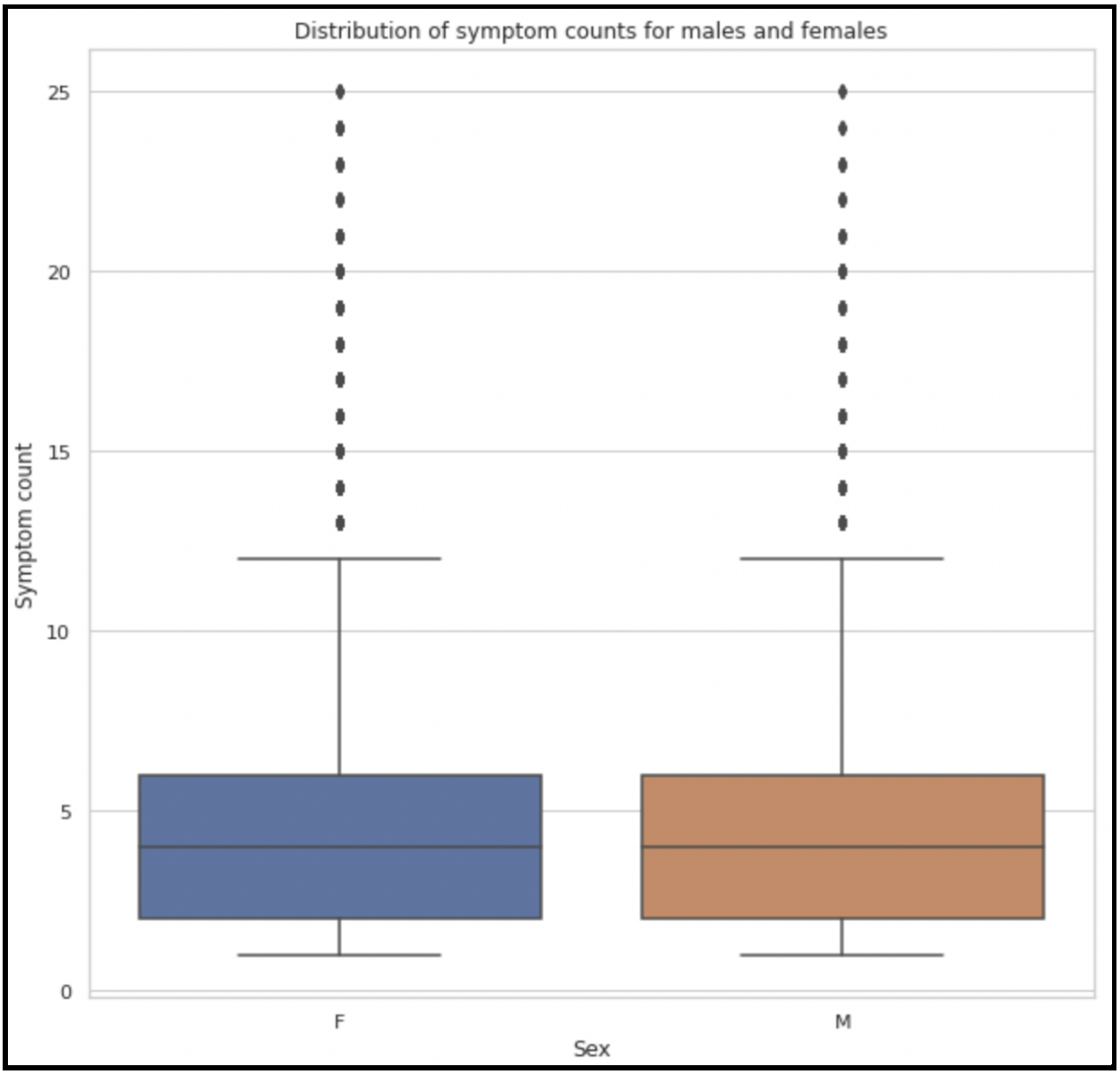

Fig. 14. Boxplot showing symptom count for male and female

We observed that both the boxplots are very similar and thus we can say that the symptom counts for both males and females are not different. The bar in the middle of the box indicates the median value and the outside ends of the box correspond to the 25th and 75th percentile values. The "whiskers" are the maximum and minimum values minus any outliers, pictured with individual dots.

```
# Distribution of symptom count in our dataset
plt.style.use('fivethirtyeight')
plt.hist(covid_df2['symptom_count'], color='darkgreen')
plt.xlabel('Number of symptoms experienced')
plt.ylabel('Frequency count')
plt.title('Symptom Count Distribution')
plt.show()
```

Fig. 15. Code for histogram of symptom count

5.3 First Research Question

To answer our first research question, we first needed to understand two variables: 'numdays' and 'symptom_count'. Symptom count was a derived attribute calculated by summing the individual symptoms each patient experienced as a result of the vaccine. Its minimum possible value is 1 since all patients in the dataset had been experiencing adverse effects. Numdays is the difference in days between onset of symptoms and vaccination date. Its minimum possible value is also 1 because the adverse effects from COVID-19 vaccines were predominantly felt a day after vaccination.

Both of these variables were of type integer and contained a decent number of outliers. We noticed symptom count values in the 60s-100s, which seemed totally unrealistic for a vaccine reaction. As a team, we discussed how we might cut off these outliers and two team members, who both had prior clinical work experience, noted a serious reaction to a vaccine may produce a symptom count in the low 20s. We concluded to set a conservative threshold of 25 symptoms. Similarly, the numdays variable saw outliers over 500 which could not be possible based on when the COVID-19 vaccines were released. We decided to set a conservative threshold of 45 days for this variable based on logic from the date vaccines were released to the public to when this dataset was published.

From there we utilized the pandas. DataFrame.describe() function to produce summary statistics on these two continuous variables. The code below shows these results.

```
covid_df2[['symptom_count', 'numdays']].describe()
```

Out[194]:

	symptom_count	numdays
count	29273.000000	29273.000000
mean	4.732621	2.700372
std	3.463876	5.093808
min	1.000000	0.000000
25%	2.000000	0.000000
50%	4.000000	1.000000
75%	6.000000	3.000000
max	25.000000	45.000000

Fig. 16. Code and output for the describe function

Here we see an average symptom count of 4.7 and 2.7 of days between vaccination and onset, respectively. Also, most of the values for both variables fall on the low end as seen by the 75th percentile. Next, we generated distribution histograms to visualize the data. See Figure 17 below. These visuals confirm what the summary statistics provided. Most of our dataset falls below 10 symptoms and 10 numdays.

```
# Distribution of symptom count in our dataset
plt.style.use('fivethirtyeight')
plt.hist(covid_df2['symptom_count'], color='darkgreen')
plt.xlabel('Number of symptoms experienced')
plt.ylabel('Frequency count')
plt.title('Symptom Count Distribution')
plt.show()
```

Fig. 17. Python code producing symptom count histogram

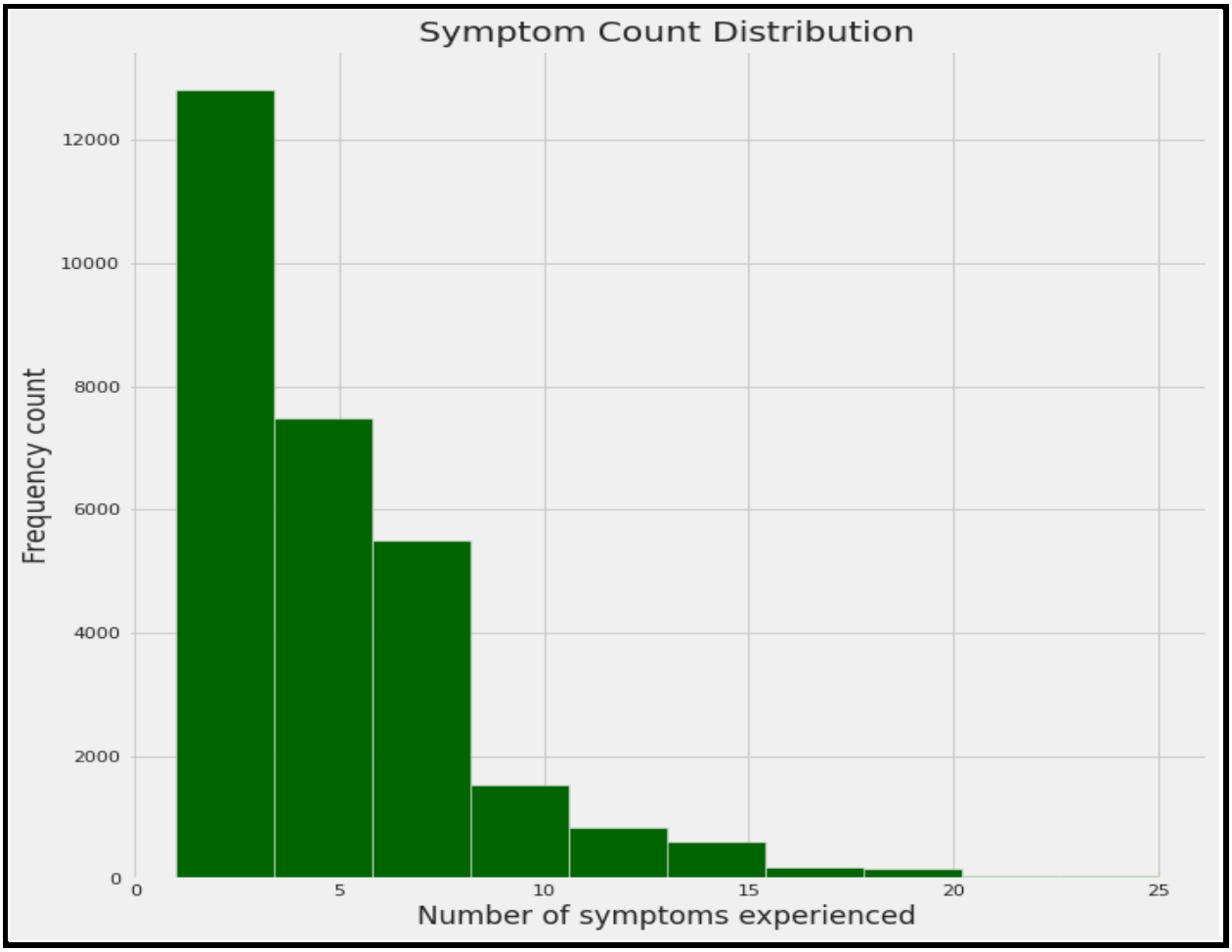

Fig. 18. Output for symptom count histogram

```
# Distribution of 'numdays' in our dataset
plt.style.use('fivethirtyeight')
plt.hist(covid_df2['numdays'], color='purple')
plt.xlabel('Number of days between vaccination and onset of symptoms')
plt.ylabel('Frequency count')
plt.title('Numdays Distribution')
plt.show()
```

Fig. 19. Python code producing “numdays” histogram

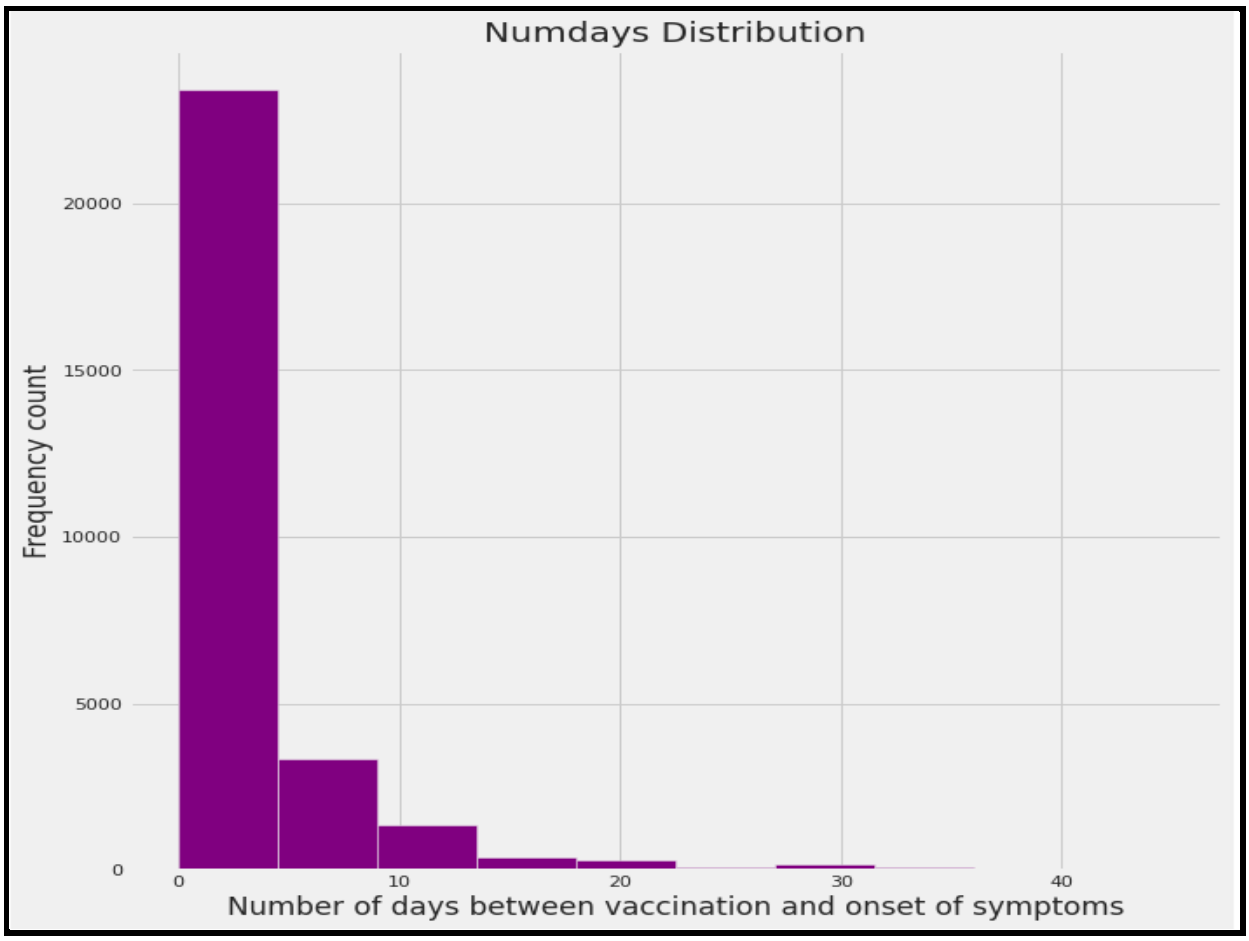

Fig. 20. Output for "numdays" histogram

Our next step was to test for normality among both variables. A p-value that is lower than 0.05 with the stats.normaltest indicates to reject the null hypothesis that the distribution is Gaussian. A p-value that is higher than 0.05 means you fail to reject the null hypothesis and assume a Gaussian distribution. Notice below both of our p-values are close to 0.0 so we reject the null hypothesis and assume a non-Gaussian distribution for both variables.

```
from scipy import stats
stats.normaltest(covid_df2['numdays'])
```

```
NormaltestResult(statistic=21296.971938060684, pvalue=0.0)
```

```
stats.normaltest(covid_df2['symptom_count'])
```

```
NormaltestResult(statistic=10260.897636006532, pvalue=0.0)
```

Fig. 21. Normality tests for 'numdays' and 'symptom_count'

Since our data did not follow a normal distribution, the Spearman rank and Kendall's Tau correlation coefficient must be used to test for a significant relationship. We performed both statistical tests to increase evaluation using two metrics.

```
stats.spearmanr(covid_df2['numdays'],covid_df2['symptom_count'])

SpearmanrResult(correlation=-0.02407431382474003, pvalue=3.798920506484018e-05)
```

```
stats.kendalltau(covid_df2['numdays'],covid_df2['symptom_count'])

KendalltauResult(correlation=-0.018973098419589215, pvalue=2.8124276366500103e-05)
```

Fig. 22. Spearman and Kendall Tau correlation tests

Notice both p-values come back below our α level of 0.05 which indicates significance. Thus, we reject our null hypothesis for research question 1 and claim evidence suggests there is a relationship between "numdays" and symptom count. Kendall's Tau result is slightly more significant so we used that test as our evaluation metric. Hence, there is a significant relationship between the number of days between vaccination and onset and symptom count. The correlation coefficient of -0.019 means there is an inverse relationship present, so as num days increases, symptom count decreases. However, it is important to note the relationship here is very weak as the correlation coefficient is close to 0.

On top of testing for a significant relationship, we wanted to use machine learning techniques to predict symptom count based upon the "numdays" variable. To accomplish this, linear regression was needed since both of our variables are continuous. Scikit-learn is a very powerful package in Python for machine learning modeling and evaluation. We imported the LinearRegression function from sklearn.linear_model to call an instance of the model. A random seed was placed beforehand to ensure replication of the results as possible. We split our DataFrame into the feature variable and target variable. Here, "numdays" is the feature vector and "symptom_count" is the target we are trying to predict. Train_test_split from scikit-learn was used to randomly split all the data into a training set and a testing set, where the training set accounted for 80% of all rows and the testing set accounted for the other 20%. The random_state parameter simulates some randomness in the model itself.

The .fit function fits the model to our data based on the training feature variable and the target variable. Linear regression works by fitting a straight line to the data which minimizes the total distance between the data points and the line of fit. The difference between the data and the line of fit is the error and once the error is minimized as much as possible, the linear regression line of best fit is created. Now it's time to predict symptom count using the linear regression model and our test "numdays" set. The test set target variable (symptom count) is also named expected_lin since those are the values we expect to see coming from the data. See Figure 23 below for relevant Python code.

```
# Create a linear regression model. IV = 'numdays' DV = 'symptom_count'
# Can we predict symptom count based upon the number of days from vaccination to onset?

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
import random

random.seed(100)
linreg = LinearRegression()

X_lin = covid_df2[['numdays']]
y_lin = covid_df2['symptom_count']

X_train_lin, X_test_lin, y_train_lin, y_test_lin = train_test_split(X_lin, y_lin, test_size=0.20, random_state=0)
linreg.fit(X_train_lin,y_train_lin)
predicted_lin = linreg.predict(X_test_lin)
expected_lin = y_test_lin
```

Fig. 23. Python code to produce linear regression model

```
# Evaluation of the Linear Regression Model
# The coefficient
print("Coefficients: \n", linreg.coef_)
# The mean squared error
print("Root mean squared error: %.2f" % np.sqrt(metrics.mean_squared_error(expected_lin, predicted_lin)))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.5f" % metrics.r2_score(expected_lin, predicted_lin))
```

```
Coefficients:
 [-0.01809771]
Root mean squared error: 3.41
Coefficient of determination: -0.00045
```

Fig. 24. Evaluation metrics of the linear regression model

Figure 24 above shows the relevant evaluation metrics for our linear regression model. The coefficient of -0.018 identifies the slope of the regression and explains how the independent variable influences the dependent. This means as "numdays" increases by 1, symptom count decreases by 0.018. The RMSE metric captures the average error from our model and is in the same units as the target variable, so our model is off by 3.41 symptoms on average. The coefficient of determination, or R^2, is the strength of the linear relationship and in our scenario, means a weak inverse relationship is present. We created a visual using plt.scatter() and plt.plot() to graphically represent the regression results.

```
# Create a graph of numdays vs. symptom count + regression line of fit

plt.style.use('fivethirtyeight')
plt.scatter(X_test,y_test)
plt.plot(X_test,predicted, color='darkblue', lw=3)
plt.xlabel('Number of days between vaccination and onset of symptoms')
plt.ylabel('Number of symptoms experienced')
plt.title('How does number of days between vaccination and onset of symptoms\n affect symptom count?')
plt.annotate("r-squared = {:.5f}".format(metrics.r2_score(expected, predicted)), (35, 23))
plt.show()
```

Fig. 25. Python code to create scatter and plot graph

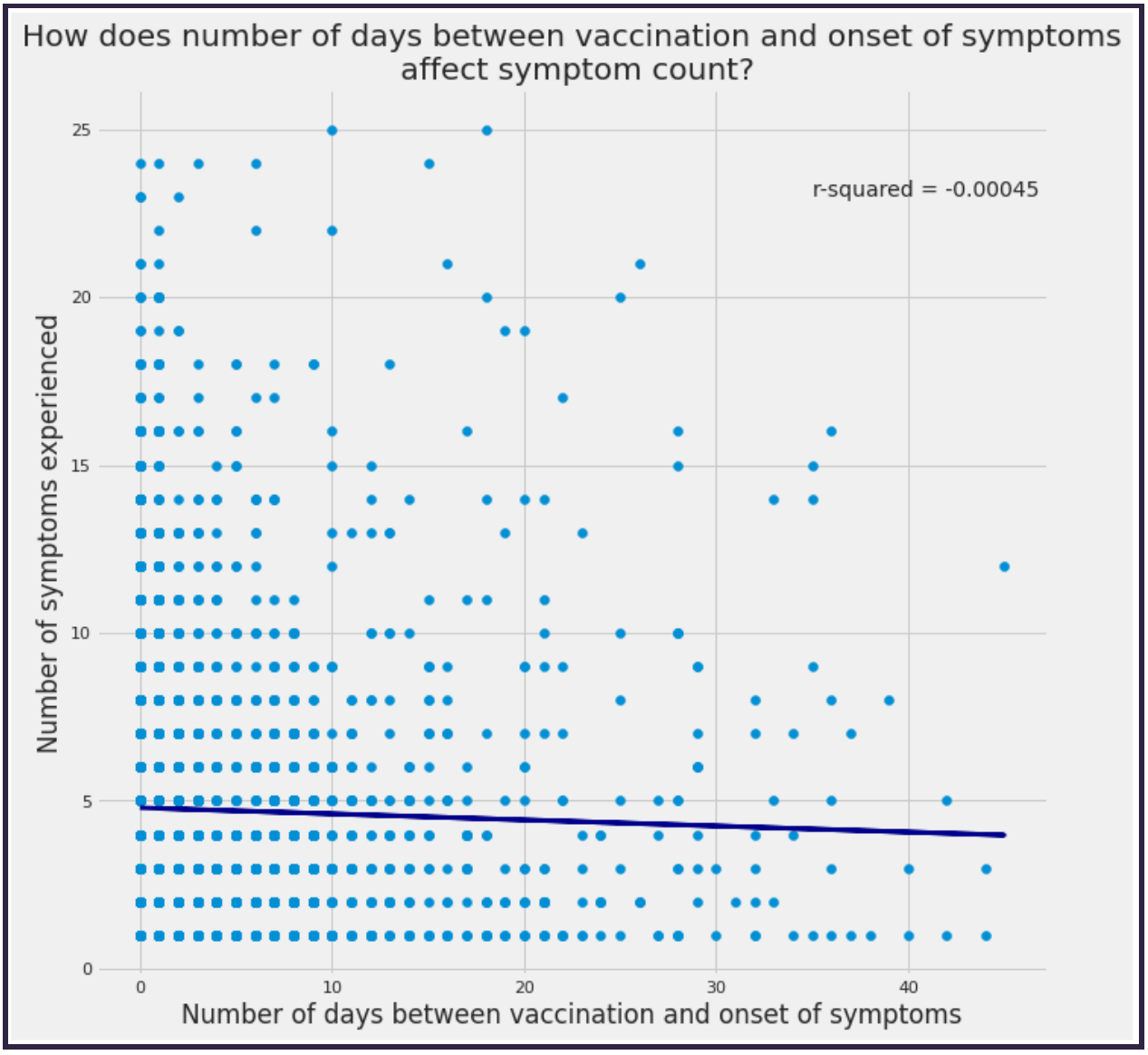

Fig. 26. Output from figure 25 code

Figure 26 above shows the relationship between our feature and target variables. Notice how the regression line of best fit does not correspond well with the data. This finding led us to try a polynomial regression model with the same two variables. Possibly the linear relationship could be explained by a higher order fit of degree 3 (cubic fit). To accomplish this, PolynomialFeatures and Pipeline were imported from the scikit-learn package (note degree=3 in the parameter). The same feature and target variables were converted into Numpy arrays in order for the Pipeline using PolynomialFeatures to work. The data was then reshaped and sorted to produce similar predicted values as were made using linear regression of degree 1. This can be seen below in Figure 27.

```
# Create a polynomial regression model based on degree=3

from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

X_poly = np.array(covid_df2['numdays'].values)
y_poly = np.array(covid_df2['symptom_count'].values)

# creating pipeline and fitting it on data
Input=[('polynomial',PolynomialFeatures(degree=3)),('modal',LinearRegression())]
pipe=Pipeline(Input)
pipe.fit(X_poly.reshape(-1,1),y_poly.reshape(-1,1))
poly_pred=pipe.predict(X_poly.reshape(-1,1))

#sorting predicted values with respect to predictor
sorted_zip = sorted(zip(X_poly,poly_pred))
x_poly, poly_pred = zip(*sorted_zip)
```

Fig. 27. Python code to produce polynomial regression model of degree 3

```
# Evaluation of the Polynomial (cubic) Regression Model

# The mean squared error
print("Root mean squared error: %.2f" % np.sqrt(metrics.mean_squared_error(y_poly, poly_pred)))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.5f" % metrics.r2_score(y_poly, poly_pred))
```

```
Root mean squared error: 3.47
Coefficient of determination: -0.00261
```

Fig. 28. Evaluation metrics for polynomial regression model

The evaluation of our polynomial regression model is seen above in Figure 28. Our RMSE value increased to 3.47 as well as the coefficient of determination increased to -0.0026. These changes are extremely minor and the cubic fit regression does not seem to be fitting well to our data still. We will visualize both models on the same scatter plot to better understand their similarities and differences (Figure 29).

```
# Plotting predictions for linear and cubic fit regression

plt.style.use('fivethirtyeight')
plt.scatter(X_test,y_test)
plt.plot(x_poly,poly_pred,color='g',label='Polynomial Regression')
plt.plot(X_test,predicted, color='darkblue', lw=3, label='Linear Regression')
plt.xlabel('Number of days between vaccination and onset of symptoms')
plt.ylabel('Number of symptoms experienced')
plt.title('How does number of days between vaccination and onset of symptoms\n affect symptom count?')
plt.annotate("r-squared (linear) = {:.5f}".format(metrics.r2_score(expected, predicted)), (35, 22))
plt.annotate("r-squared (cubic) = {:.4f}".format(metrics.r2_score(y, poly_pred)), (35.9, 21))
plt.legend(loc='upper right', bbox_to_anchor=(1.08,.75))
plt.show()
```

Fig. 29. Python code to scatter plot with overlapping regression models

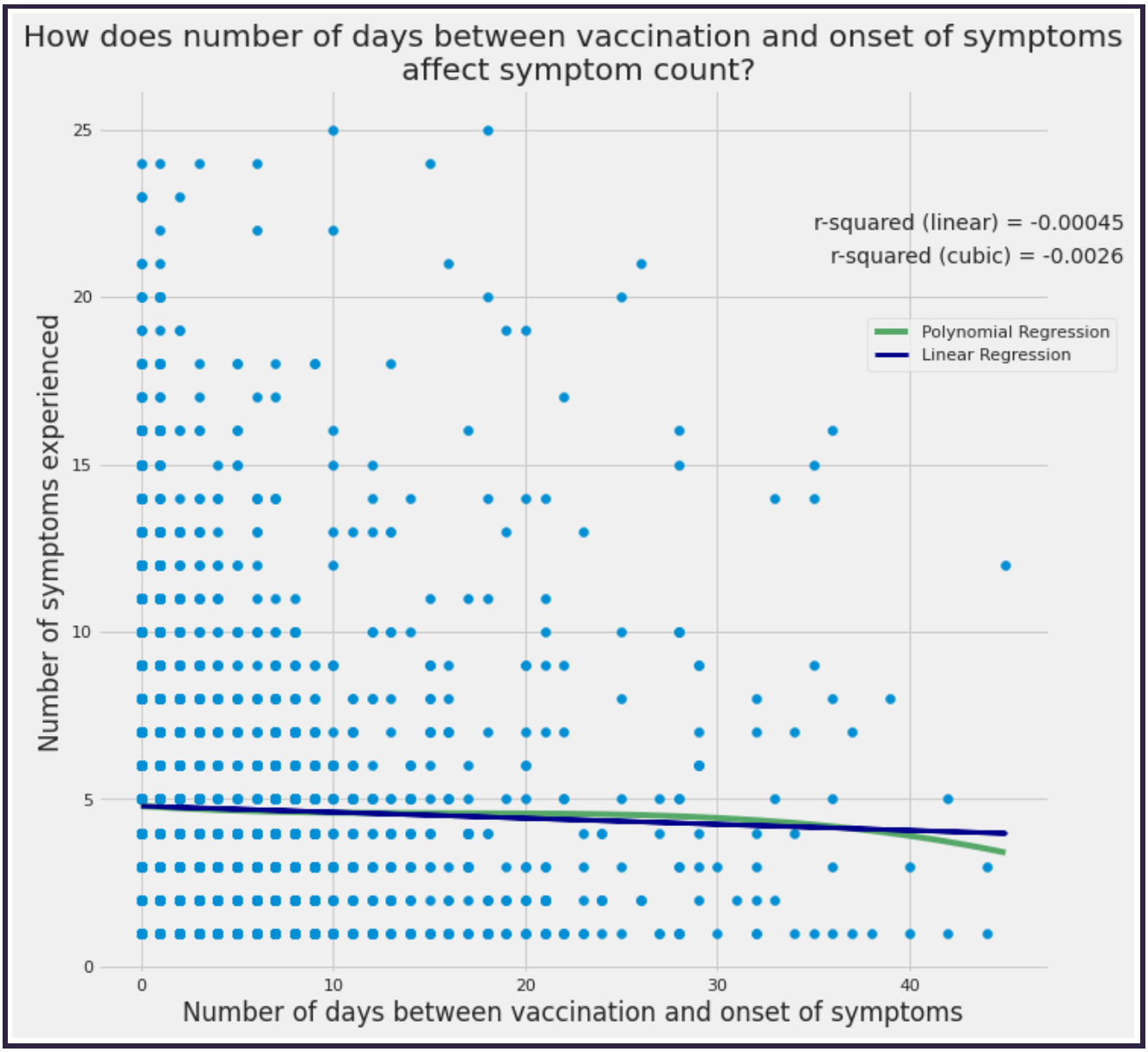

Fig. 30. Plot showing linear and polynomial regression lines

Above, in figure 30 we can see the predictive functions for the linear regression (dark blue) and polynomial regression (green) models. The polynomial regression model only begins to deviate from the first-degree linear model as "numdays" reach> 40 days. This is likely due to the immensely high concentration of data in the lower left quadrant of the plot, where "numdays" and "symptom_count" are < 10. We can conclude that linear regression modeling is not the appropriate technique for predicting symptom count using the number of days from vaccination to onset as the feature vector.

5.4 Second Research Question

Now for the second research question, we need a target variable. For the target variable, we decided to use the five columns - 'l_threat', 'died', 'hospital', 'disable', and 'birth_defect'. These five columns have boolean values of 0 and 1 which signify that if a person is having these serious effects, the value is 1, or else if the person is not having that severity, the value is 0. We created a new column by the name of 'Count' which sums the values of these five columns. And then based on the value of 'Count' we divided the data into seven classes.

1. If Count = 0, then Class = 'No Severity' or 0.
2. If Count = 1, then Class depends on the column which contributes towards this value. So we get 5 classes from this: Class = 'Life threatening' or 1, 'Hospitalized' or 2, Class = 'disable' or 3, Class = 'birth defect' or 4 and Class = 'died' or 6.
3. If Count > 1, then Class = 'Very Serious' or 5.

So, now we have 7 classes. We use if-else statements to create a list of classes of all the rows and then append that list by the name of 'Target' to the data frame.

```
# Adding new column 'count' which count the no. of severity criteria - l_threat,died,hospital,dis
covid_df["Count"] = covid_df[['l_threat','died','hospital','disable','birth_defect']].sum(axis=1)
```

```
covid_df['Count'].value_counts()
```

```
0    27012
1     5431
2     1219
3      142
4        4
Name: Count, dtype: int64
```

Fig. 31. Value counts of the column 'Count'

```
# Creating the target variable by giving a type a severity a value between 0 and 6
Target = []

for i in range(0,29273):
    if covid_df2['Count'][i] == 0:
        Target.append(0)
    elif covid_df2['Count'][i] == 1:
        if covid_df2['l_threat'][i] == 1:
            Target.append(1)
        elif covid_df2['died'][i] == 1:
            Target.append(6)
        elif covid_df2['hospital'][i] == 1:
            Target.append(2)
        elif covid_df2['disable'][i] == 1:
            Target.append(3)
        elif covid_df2['birth_defect'][i] == 1:
            Target.append(4)
    else:
        Target.append(5)

```

```
# Adding Target list to dataframe as a new column - Target
covid_df2['Target'] = Target
```

```
covid_df2['Target'].value_counts()
```

```
0    23329
2     2515
6     1359
5     1246
3      448
1      340
4       36
Name: Target, dtype: int64
```

Fig. 32. Creating the Target column

Then, we created a list of labels of those classes and made a new column by the name ‘severity_name’ using those labels.

```
# Names of Target variables
NTarget_Names = ['No Severity','Life Threatening','Hospitalized','Disabled','Birth Defect','Very Serious',
                 'Died']
```

```
# Creating a new column by name of severity_name to identify the type of severity
covid_df2['severity_name'] = np.array([NTarget_Names[i] for i in NTarget])
```

```
# Should match up with 'Target' column
covid_df2['severity_name'].value_counts()
```

```
No Severity         23329
Hospitalized         2515
Died                 1359
Very Serious         1246
Disabled              448
Life Threatening      340
Birth Defect           36
Name: severity_name, dtype: int64
```

Fig. 33. Creating column ‘severity name’ using labels for target variable

Now for the input variables, we are using ‘age_yrs’, ‘sex’, ‘vax_manu’, and ‘history’. As ‘sex’ and ‘vax_manu’ had strings, we converted the values into integers.

1. Assigned ‘sex’ column a number and created the new list by sex_num - Male is assigned to 0, female assigned to 1
2. Assigned ‘vax_manu’ a number and created the new list by vaxmanu - Moderna assigned to 0, Pfizer assigned to 1, Janssen assigned to 2

Then we appended both the lists to our data frame and kept the column names as ‘sex_data’ and ‘vax_manu_data’.

```
# Assigning sex column a number and creating the new list by sex_num
# Male is assigned to 0, female assigned to 1
sex_num = []
for i in range(0,29273):
    if covid_df2['sex'][i] == 'M':
        sex_num.append(0)
    else:
        sex_num.append(1)
```

```
# Creating new column by name of sex_data and using the newly created list sex_data
covid_df2['sex_data'] = sex_num
```

```
# Assigning vax_manu a number and creating the new list by vaxmanu
# Moderna assigned to 0, Pfizer assigned to 1, Janssen assigned to 2
vaxmanu = []
for i in range(0,29273):
    if covid_df2['vax_manu'][i] == 'MODERNA':
        vaxmanu.append(0)
    elif covid_df2['vax_manu'][i] == 'PFIZER\BIONTECH':
        vaxmanu.append(1)
    elif covid_df2['vax_manu'][i] == 'JANSSEN':
        vaxmanu.append(2)
```

```
# Adding the new column vax_manu_data taking values from newly created list - vaxmanu.
covid_df2['vax_manu_data'] = vaxmanu
```

Fig. 34. Assigning numbers to ‘sex’ and ‘vax_manu’ for analysis

Then, we made a new data frame by selecting the input variables - ‘age_yrs’, ‘sex_data’, ‘vax_manu_data’ and ‘history’ and the Target variable - ‘severity_name’.

```
# Creating a new dataframe using the columns required and the target variable (severity_name)
covid_df_target = covid_df2[['age_yrs','sex_data','vax_manu_data','history','severity_name']]
```

```
covid_df_target.head()
```

	age_yrs	sex_data	vax_manu_data	history	severity_name
0	33.0	1	0	0	No Severity
1	73.0	1	0	1	No Severity
2	23.0	1	1	0	No Severity
3	58.0	1	0	1	No Severity
4	47.0	1	0	0	No Severity

Fig. 35. Data frame for analysis

We used two classification models:

1. Random forest classification model
2. Gradient boosting classification model.

Using random forest classification model:

We imported 'train_test_split' from 'sklearn.ensemble'. We used 'train_test_split' to split our data into train and test data. We kept the test size = 0.5. Then we imported 'RandomForestClassifier' from 'sklearn.ensemble' and we used the train data to train our model. Then we also calculated the accuracy score and out-of-bag score which came as 0.79 and 0.794 respectively.

```
# Generating the test and train sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(covid_df_target[['age_yrs','sex_data','vax_manu_data',
                                                                     'history']], NTarget , test_size=0.5,
                                                    stratify=NTarget, random_state=123456)

```

```
# Random Forest classification
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, oob_score=True, random_state=123456)
rf.fit(X_train, y_train)
```

```
RandomForestClassifier(oob_score=True, random_state=123456)
```

```
# Accuracy evaluation
from sklearn.metrics import accuracy_score
predicted = rf.predict(X_test)
accuracy = accuracy_score(y_test, predicted)
print(f'Out-of-bag score estimate: {rf.oob_score_:.3}')
print(f'Mean accuracy score: {accuracy:.3}')
```

```
Out-of-bag score estimate: 0.79
Mean accuracy score: 0.794
```

Fig. 36. Code for Random forest model and for accuracy score

Using Gradient boosting classification model:

We imported 'train_test_split' from 'sklearn.ensemble'. We used 'train_test_split' to split our data into train and test data. We kept the test size = 0.5. Then we imported 'GradientBoostingClassifier' from 'sklearn.ensemble' and we used the train data to train our model. Then we also calculated the accuracy score which came as 0.743

```python
# Using gradient boosting classifier
from sklearn.ensemble import GradientBoostingClassifier

clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0)
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
score = clf.score(X_test, y_test)
print(score)
```

```
0.7433900389424062
```

Fig. 37. Code for Gradient boosting classification and for accuracy score

Results of models:

We imported 'metrics' from 'sklearn' to get the classification report of both the models.

```python
# Metrics classification report
from sklearn import metrics
print(metrics.classification_report(y_test, predicted))

# Note the low scores for all classes except class 0
```

```
              precision    recall  f1-score   support

           0       0.84      0.97      0.90     11665
           1       0.00      0.00      0.00       170
           2       0.25      0.09      0.14      1258
           3       0.00      0.00      0.00       224
           4       0.00      0.00      0.00        18
           5       0.14      0.03      0.05       623
           6       0.30      0.20      0.24       679

    accuracy                           0.79     14637
   macro avg       0.22      0.19      0.19     14637
weighted avg       0.71      0.79      0.74     14637
```

Fig. 38. Classification report for random forest model

```python
# Classification report for gradient boosting classifier

print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.88      0.90      0.89     11665
           1       0.00      0.00      0.00       170
           2       0.00      0.00      0.00      1258
           3       0.02      0.08      0.03       224
           4       0.00      0.00      0.00        18
           5       0.00      0.00      0.00       623
           6       0.23      0.59      0.33       679

    accuracy                           0.74     14637
   macro avg       0.16      0.22      0.18     14637
weighted avg       0.71      0.74      0.72     14637
```

Fig. 39. Classification report for gradient boosting model

This report tells us that both our models are only able to predict the first class (0) accurately. This class is the only one that is contributing towards the high accuracy of our models. Other classes have low F-1 scores, precision, and recall.

Then we plotted the confusion matrix:

```python
# Heat map for actual and predicted labels

cm = metrics.confusion_matrix(y_test, predicted)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = f'Mean accuracy score: {accuracy:.3}'
plt.title(all_sample_title, size = 15);
```

```python
# Heat map for actual and predicted values for gradient boosting classification

cm = metrics.confusion_matrix(y_test, predictions)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```

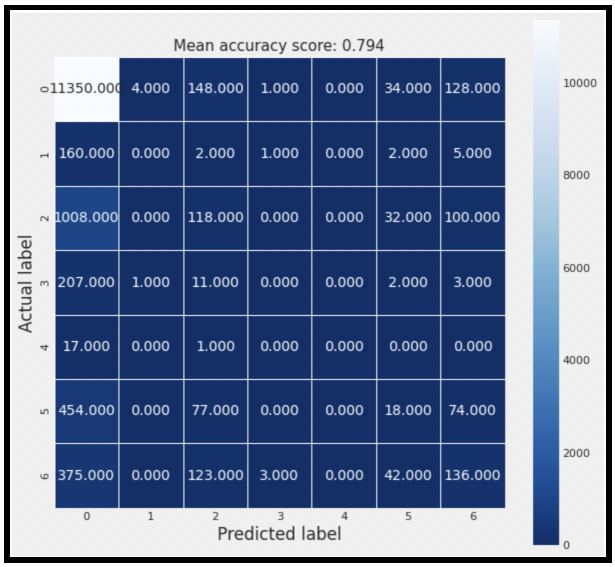

Fig. 40. Confusion matrix for Random Forest Classification

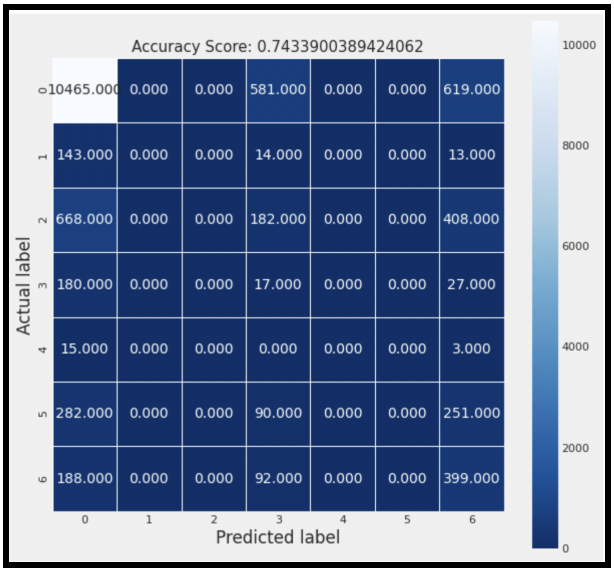

Fig. .41. Confusion matrix for Gradient Boosting Classification

This confusion matrix also shows us the same results. Our models can predict the first class accurately but not the other classes.

Let us look at the ROC curves of both models:

We import 'scikitplot' to make the ROC (Receiver Operating Characteristic) curves.

```
# ROC curve for random forest model

plt.rcParams['figure.figsize'] = [10, 10]

predicted_probas = rf.predict_proba(X_test)

import scikitplot as skplt
skplt.metrics.plot_roc(y_test, predicted_probas)

plt.show()
```

```
# ROC curve for gradient boosting classifier
plt.rcParams['figure.figsize'] = [10, 10]

predicted_probas1 = clf.predict_proba(X_test)

skplt.metrics.plot_roc(y_test, predicted_probas)
plt.show()
```

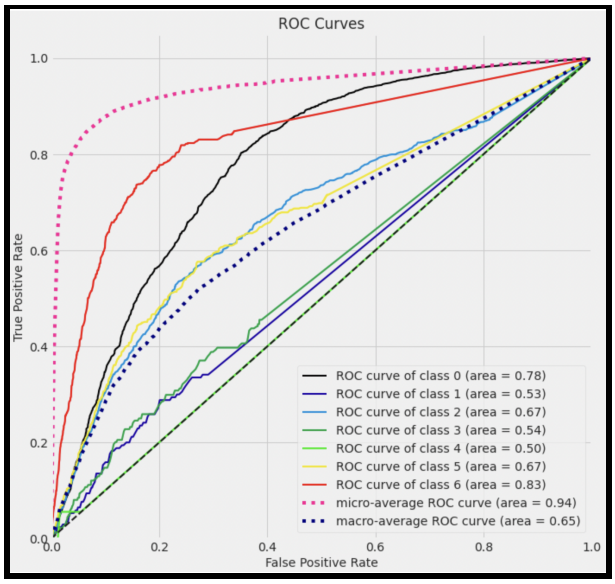

Fig. 42. ROC for Random Forest Classification

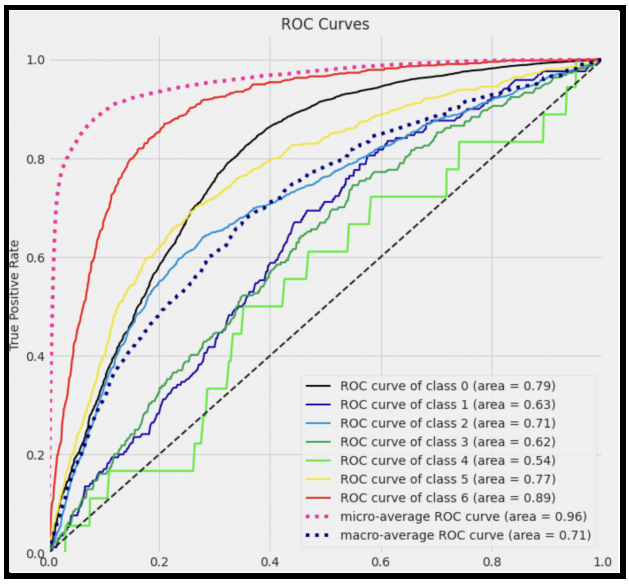

Fig. 43. ROC for Gradient Boosting classification

ROC curves also tell us the same thing. For both the models, the area under the curve for the first class is very high as compared to the area under the curve for the other six classes.

Problem with our model:

We had a high no. of rows for the 'No Severity' class but very little no. of rows for the other 6 classes.

```
1  covid_df2['severity_name'].value_counts()
```

```
No Severity          23329
Hospitalized          2515
Died                  1359
Very Serious          1246
Disabled               448
Life Threatening       340
Birth Defect            36
Name: severity_name, dtype: int64
```

Fig. 44. Value counts of 'severity name' column

So, we decided to make the other 6 classes as a single class by the name of 'Severe Side effects'. We did this by creating a list by checking the value of the 'Count' column.

If Count = 0, then Class = 'No Severity' (0)

If Count > 0, then class = 'Severe side effects' (1)

```
#Creating new list based on the no. of count. If count = 0, then Severity = 0. else Severity = 1.
Sever = []
for i in covid_df['Count']:
    if i == 0:
        Sever.append(0)
    else:
        Sever.append(1)
```

```
#Adding Sever list to dataframe as a new column - severity
covid_df['severity'] = Sever
```

```
# Creating array of Severity column
Nseverity = np.array(covid_df2['severity'])
```

```
# Names of severity variables
Nseverity_Names = ['No Severity','Severe Side-effects']
```

```
# Creating a new column by name of severity_type to identify the type of severity
covid_df2['severity_type'] = np.array([Nseverity_Names[i] for i in Nseverity])
```

Fig. 45. Creating new column 'severity_type' based on value of 'Count' column

We used three classification models:

1. Logistic regression model
2. Random forest classification model
3. Gradient boosting classification model

Using Random forest classification model:

We imported 'train_test_split' from 'sklearn.ensemble'. We used 'train_test_split' to split our data into train and test data. We kept the test size = 0.5. Then we imported 'RandomForestClassifier' from 'sklearn.ensemble' and we used the train data to train our model. Then we also calculated the accuracy score and out-of-bag score which came as 0.836 and 0.833 respectively.

Using logistic regression model:

We imported 'train_test_split' from 'sklearn.ensemble'. We used 'train_test_split' to split our data into train and test data. We kept the test size = 0.5. Then we imported 'LogisticRegression' from 'sklearn.linear_model' and we used the train data to train our model. Then we also calculated the accuracy score which came out as 0.834.

Using Gradient boosting classification model:

We imported 'train_test_split' from 'sklearn.ensemble'. We used 'train_test_split' to split our data into train and test data. We kept the test size = 0.5. Then we imported 'GradientBoostingClassifier' from 'sklearn.ensemble' and we used the train data to train our model. Then we also calculated the accuracy score which came out as 0.84.

Results of Models:

Classification Report:

	precision	recall	f1-score	support
No Severity	0.85	0.96	0.90	4666
Severe Side-effects	0.68	0.36	0.47	1189
accuracy			0.83	5855
macro avg	0.77	0.66	0.69	5855
weighted avg	0.82	0.83	0.81	5855

Fig. 46. Classification report for Logistic Regression

	precision	recall	f1-score	support
No Severity	0.86	0.94	0.90	4666
Severe Side-effects	0.64	0.40	0.50	1189
accuracy			0.83	5855
macro avg	0.75	0.67	0.70	5855
weighted avg	0.82	0.83	0.82	5855

Fig. 47. Classification report for Random Forest classification

	precision	recall	f1-score	support
No Severity	0.86	0.95	0.90	4666
Severe Side-effects	0.68	0.41	0.51	1189
accuracy			0.84	5855
macro avg	0.77	0.68	0.71	5855
weighted avg	0.83	0.84	0.82	5855

Fig. 48. Classification report for Gradient Boosting Classification

Again, all the models can predict the 1st class (0) accurately but are not able to predict the 2nd class (1) accurately with low precision, recall, and f1 scores.

Confusion matrix:

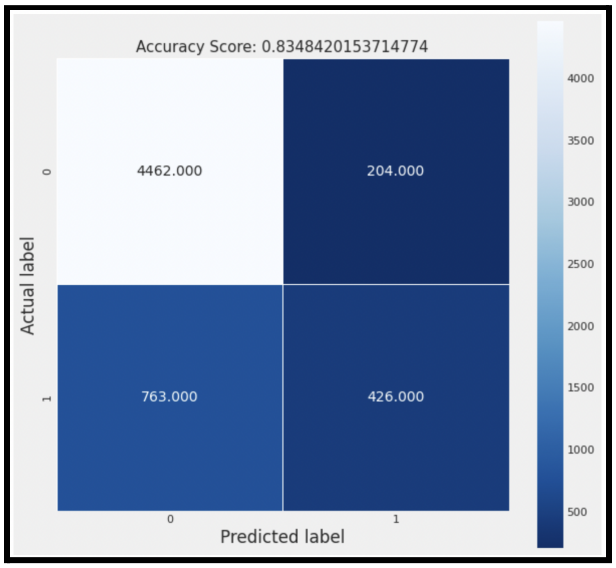

Fig. 49. Confusion matrix for Logistic regression

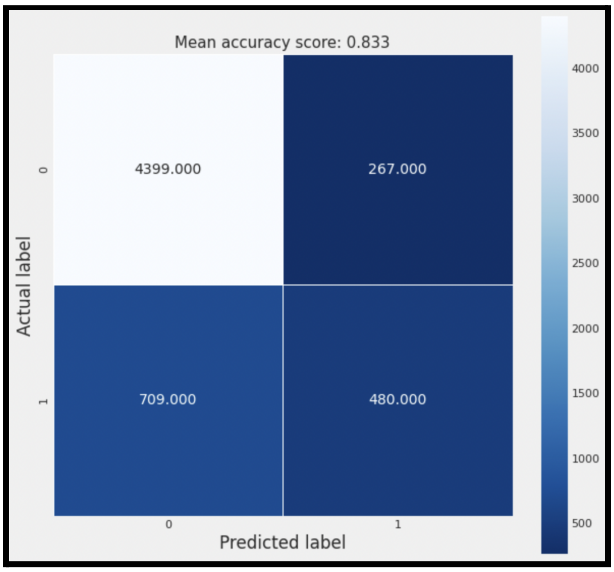

Fig. 50. Confusion matrix for Random Forest classification

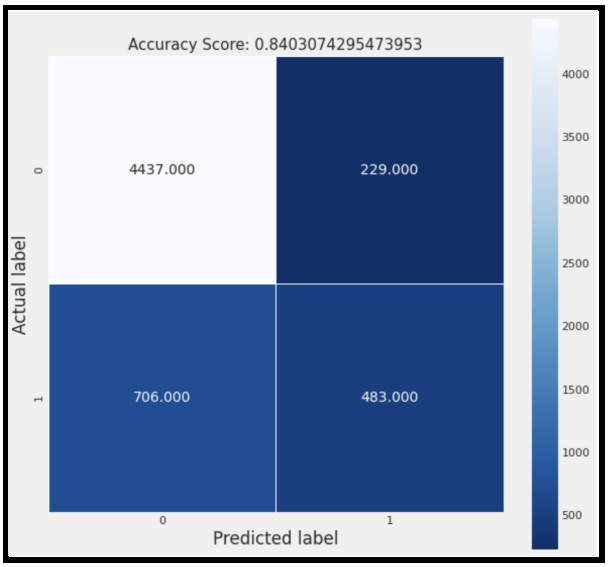

Fig. 51. Confusion matrix for Gradient boosting classification

Again, the confusion matrix also shows that all the models can predict the 1st class (0) accurately but are not able to predict the 2nd class (1).

ROC curves:

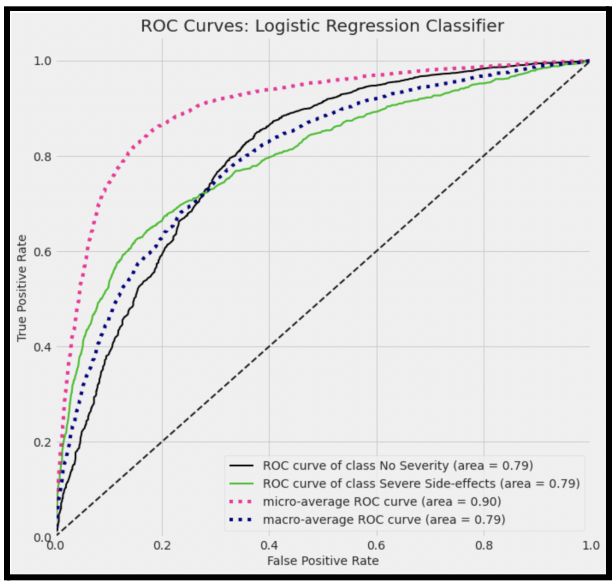

Fig. 52. ROC for Logistic regression

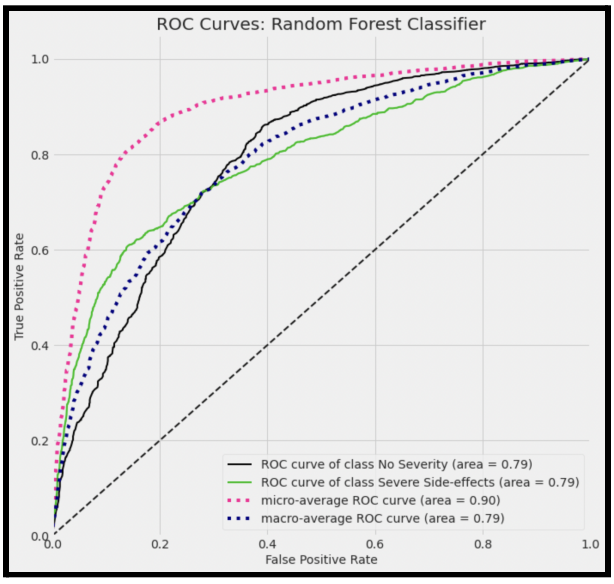

Fig. 53. ROC for Random forest classification

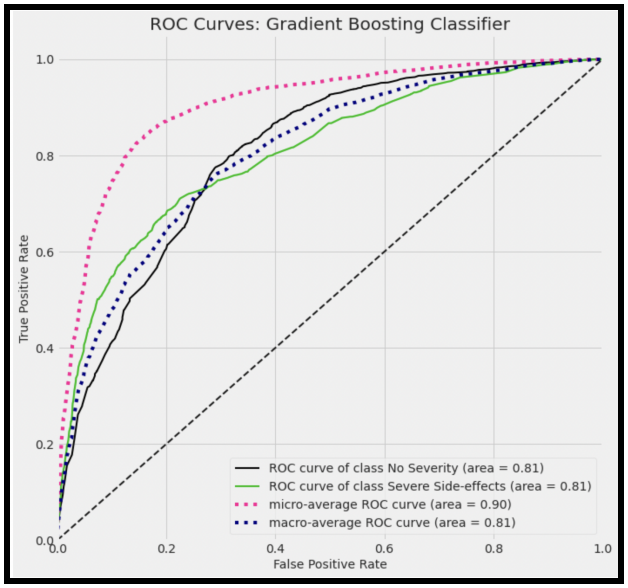

Fig. 54. Gradient boosting classifier

According to ROC curves, we see that the gradient boosting classification model gives the highest area under the curve and therefore is the best model.

Problem with our data:

We have a high no. of rows for class - 'No Severity' and very less no. of rows for 'Severe side effects'.

```
1  covid_df2['severity_type'].value_counts()
```

```
No Severity              23329
Severe Side-effects       5944
Name: severity_type, dtype: int64
```

Fig. 55. Value counts for 'severity_type'

To overcome this problem, we decided to equalize the no. of rows for both classes. For this, we created two separate data frames - one with all rows of class - 'No Severity' and one with all rows of class - 'Severe Side-effects'. After this, we selected random 5944 rows from the data frame which contained 'No Severity' data. Then we combined both the data frames into a new data frame that had an equal no. of rows for both the classes. We named it 'covid_New3'.

```
# Creating new dataframe - covid_New which only has rows which have severity_type as No severity
covid_New = covid_df2[covid_df2['severity_type'] == 'No Severity']
```

```
#covid_New.head()
```

```
covid_New['severity_type'].value_counts()
```

```
No Severity    23329
Name: severity_type, dtype: int64
```

```
# Selecting random 5944 rows from covid_New data frame. (As severity values are 5944)
covid_New = covid_New.sample(n=5944)
```

```
covid_New['severity_type'].value_counts()
```

```
No Severity    5944
Name: severity_type, dtype: int64
```

```
# Creating new dataframe - covid_New2 which only has rows which have severity_type as Severe Side-effects
covid_New2 = covid_df2[covid_df2['severity_type'] == 'Severe Side-effects']
```

```
covid_New2['severity_type'].value_counts()
# Both classes now have 5944 rows
```

```
Severe Side-effects    5944
Name: severity_type, dtype: int64
```

```
# Appending rows of covid_New2 to covid_New and creating a new dataframe - 'covid_New3'
covid_New3 = covid_New.append(covid_New2)
```

```
covid_New3['severity_type'].value_counts()
```

```
No Severity            5944
Severe Side-effects    5944
Name: severity_type, dtype: int64
```

Fig. 56. Equalizing rows for both classes

Now we used our classification model on this new data frame.
Models used:

1. Logistic regression
2. Random forest classification
3. Gradient boosting classification

Results of models:

Classification Report:

	precision	recall	f1-score	support
No Severity	0.72	0.74	0.73	2972
Severe Side-effects	0.73	0.72	0.72	2972
accuracy			0.73	5944
macro avg	0.73	0.73	0.73	5944
weighted avg	0.73	0.73	0.73	5944

Fig. 57. Classification report for Logistic Regression

	precision	recall	f1-score	support
No Severity	0.70	0.76	0.73	2972
Severe Side-effects	0.74	0.67	0.70	2972
accuracy			0.72	5944
macro avg	0.72	0.72	0.71	5944
weighted avg	0.72	0.72	0.71	5944

Fig. 58 . Classification rep5ort for Random forest classification

	precision	recall	f1-score	support
No Severity	0.71	0.81	0.76	2972
Severe Side-effects	0.78	0.67	0.72	2972
accuracy			0.74	5944
macro avg	0.75	0.74	0.74	5944
weighted avg	0.75	0.74	0.74	5944

Fig. 59. Classification report for Gradient boosting classification

According to the report, all the three models are now able to predict both the classes with an accuracy of around 0.73. For both the classes, the precision, recall and f1 score values are above 0.7.

Confusion matrix:

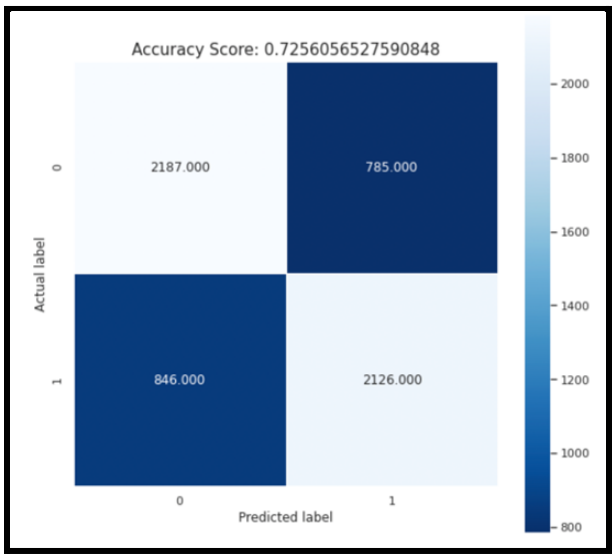

Fig. 60. Confusion matrix for Logistic regression

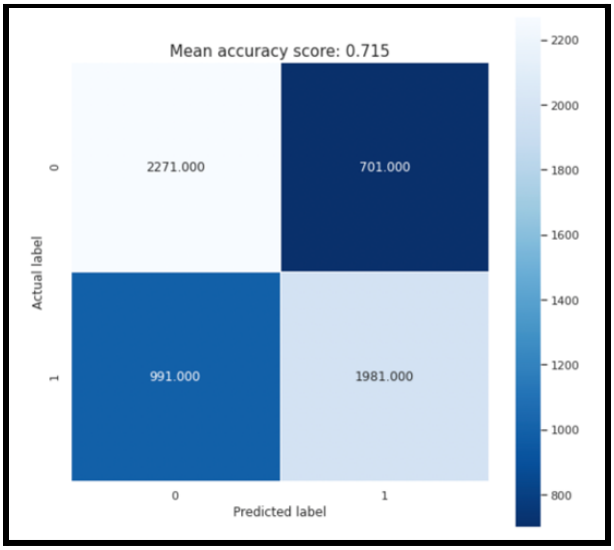

Fig. 61. Confusion matrix for Random forest classification

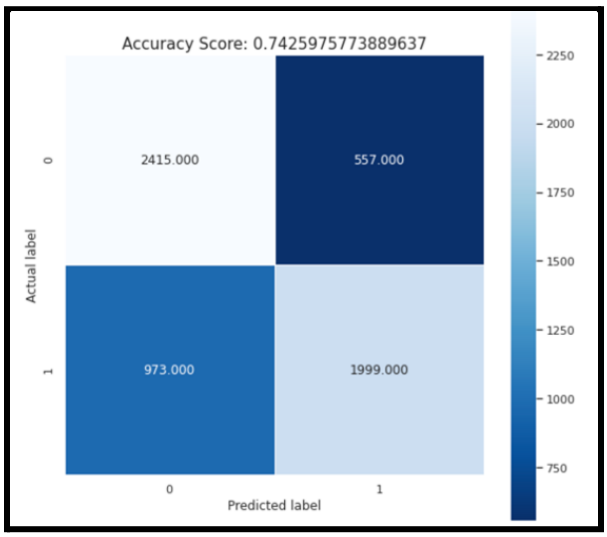

Fig. 62. Confusion matrix for Gradient boosting classification

Confusion matrix also shows that both the values are being predicted accurately in all the three models.

ROC curve comparison:

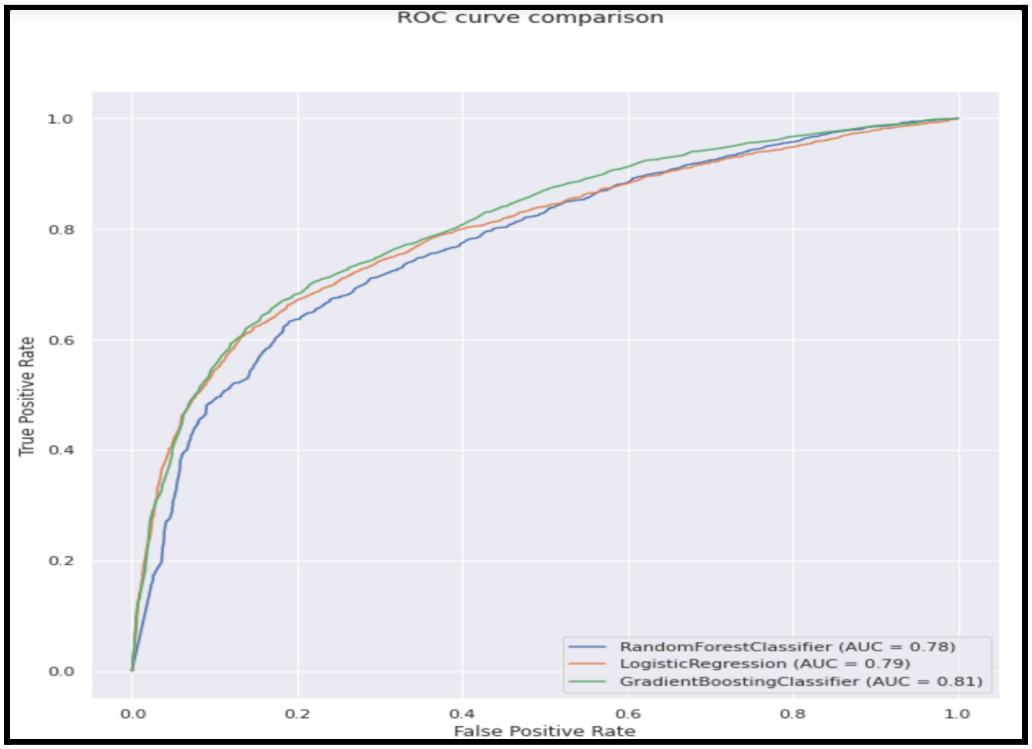

Fig. 63. ROC curve comparison for all the three models

ROC curves show that gradient boosting classification is the best model with the highest area under the curve of 0.81.

Making predictions:

We used gradient boosting classification model to make some predictions. We use the model.predict function to make predictions.

```
1 clf.predict([[90,0,0,1]])
```

```
array(['Severe Side-effects'], dtype=object)
```

```
1 clf.predict([[55,1,2,0]])
```

```
array(['No Severity'], dtype=object)
```

Fig. 64. Predictions for the gradient boosting model

5.5 Third Research Question

Moving to the 3rd research question, we needed to split our age data into two parts - Non Severity age and Severity age. For this, we used the following code:

```
In [709]:
1 covid_comp = covid_df2[['age_yrs','sex','severity','severity_type']]

In [710]:
1 covid_comp.head()

Out[710]:
```

	age_yrs	sex	severity	severity_type
0	33.0	F	0	No Severity
1	73.0	F	0	No Severity
2	23.0	F	0	No Severity
3	58.0	F	0	No Severity
4	47.0	F	0	No Severity

```
In [711]:
1 No_Severity_age = covid_comp[covid_comp['severity'] == 0]['age_yrs']
2 Severity_age = covid_comp[covid_comp['severity'] == 1]['age_yrs']
```

Fig. 65. Splitting the 'age_yrs' data for 'severity' and 'Non severity'

We have two sets of data - No_Severity_age (ages of people who suffered no severity) and Severity_age (ages of people who suffered severity).

We first did the normality test to check whether the data is normally distributed or not.
For normality test:
Null hypothesis: The data is normally distributed.
Alternate hypothesis: The data is not normally distributed.

```
In [711]:
1 No_Severity_age = covid_comp[covid_comp['severity'] == 0]['age_yrs']
2 Severity_age = covid_comp[covid_comp['severity'] == 1]['age_yrs']
3 #stats.ttest_ind(female_viq, male_viq)

In [731]:
1 type(Severity_age)

Out[731]: pandas.core.series.Series

In [712]:
1 #Normality test
2 from scipy import stats
3 stats.normaltest(No_Severity_age)

Out[712]: NormaltestResult(statistic=737.0026831441871, pvalue=9.160111947441657e-161)

In [713]:
1 stats.normaltest(Severity_age)

Out[713]: NormaltestResult(statistic=450.82866193775004, pvalue=1.2699888800188004e-98)
```

Fig. 66. Normality test for both the groups

For both the groups - No_Severity_age and Severity_age, we get the p-value which is less than 0.05. As this value is less than 0.05, we can reject the null hypothesis and conclude that both the groups are not normally distributed. As the data is not normally distributed, we had to do the non parametric test for comparing both the groups. As both the groups are independent from each other, we went with the mannwhitneyu test.

For Mannwhitneyu test:
Null hypothesis: There is no difference in age between the two groups.
Alternate hypothesis: There is a significant difference in age between the two groups.

```
In [714]:  1  stats.mannwhitneyu(No_Severity_age,Severity_age)

Out[714]: MannwhitneyuResult(statistic=32459234.0, pvalue=0.0)
```

Fig. 67. Mannwhitneyu test

As the p-value for the test came as very close to 0 and less than 0.05, we can reject the null hypothesis and state that there is a significant statistical difference between the ages of the two groups.

Using describe function on both the groups:

```
In [715]:  1  No_Severity_age.describe()

Out[715]: count    23329.000000
          mean        47.431866
          std         16.040632
          min          0.000000
          25%         35.000000
          50%         46.000000
          75%         58.000000
          max        115.000000
          Name: age_yrs, dtype: float64
```

Fig. 68. Description of 'No_Severity_age'

```
In [716]:  1  Severity_age.describe()

Out[716]: count    5944.000000
          mean       65.932705
          std        19.470628
          min         1.000000
          25%        52.000000
          50%        70.000000
          75%        81.000000
          max       106.000000
          Name: age_yrs, dtype: float64
```

Fig. 69. Description of 'Severity_age'

We can see that the mean of 'No_Severity_age' is 47.43 years whereas the mean of 'Severity_age' is 65.93 years. This shows that the people who suffer severe side effects are much older as compared to people with non severe side-effects.

Histogram and density plot of both the groups:

We plot the histogram and density plot in the same figure for both the groups using the 'distplot' function from the 'seaborn' library.

```
# Density plot of age split by no severity and severity groups

sns.distplot(No_Severity_age, hist=True, kde=True, bins=50, color = 'darkblue',
             hist_kws={'edgecolor':'black'},kde_kws={'linewidth': 4},label = 'No Severity')
sns.distplot(Severity_age, hist=True, kde=True, bins=50, color = 'red',
             hist_kws={'edgecolor':'black'},kde_kws={'linewidth': 4}, label = 'Severity')
plt.xlabel('Age (years)')
plt.legend()
plt.title('Density plot for age of patients with severity and No Severity')
```

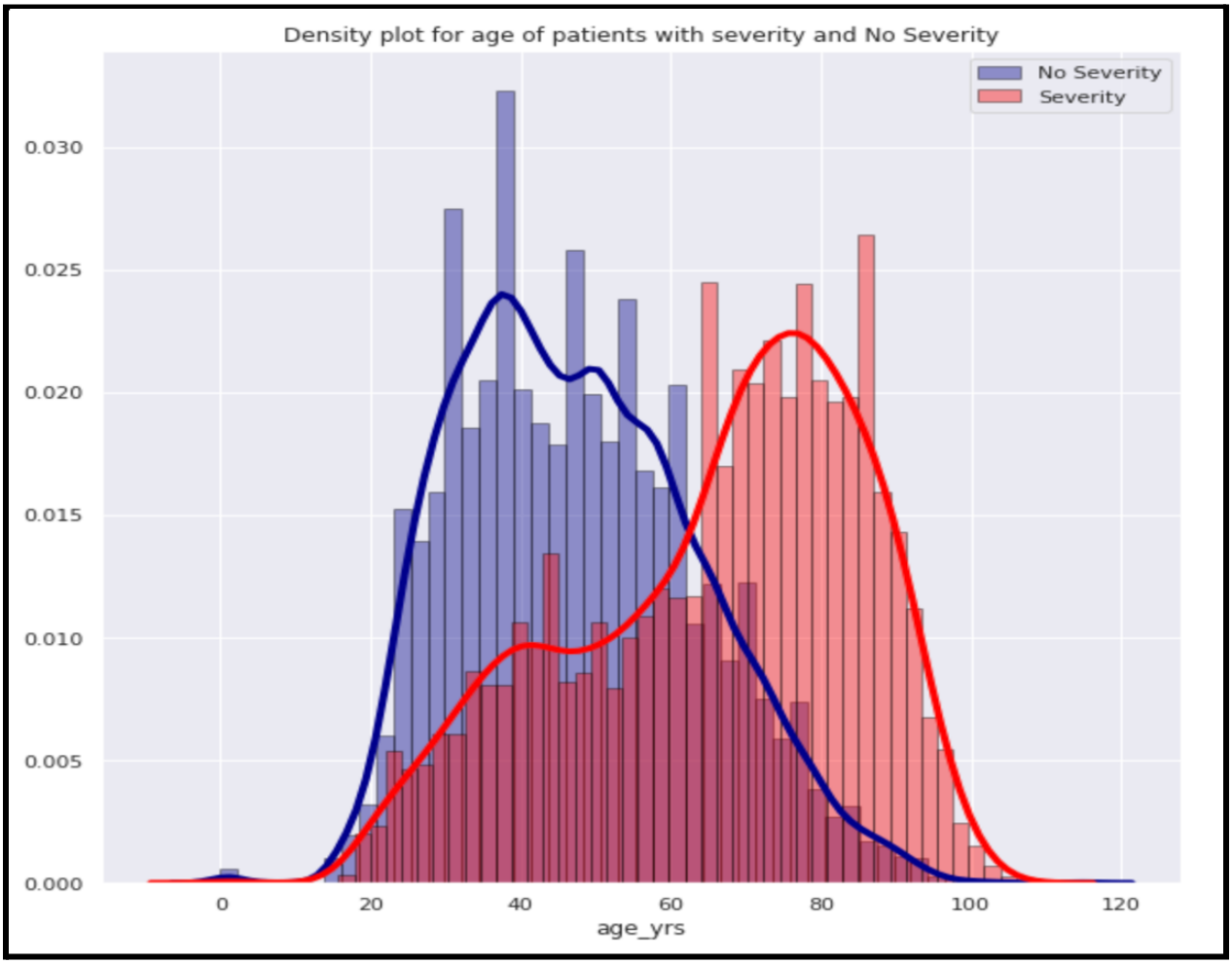

Fig. 70. Density and histogram plots for both the groups - 'Severity_age' and 'No_Severity_age'

The density plot also shows that there is a significant difference between both the groups and people with no severe side effects are a lot younger than people with severe side-effects.

6. Summary and Findings

We addressed all our three research questions and got the following findings:

- There is a significant relationship (inverse, very weak) between number of days from vaccination to onset and the symptom count.

- We were able to predict the severity of the adverse effect based on age, gender, vaccine manufacturer and medical history with an accuracy score of 0.74. We found Gradient Boosting Classification as the best model for prediction.
- There is a significant difference in age between patients who experienced severe reactions and the patients who did not experience any severe reactions. People having severe reactions were found much older than those people who did not suffer any severe side effects.

7. Limitations

As a team, we did our best to remain consistent and avoid mistakes in our methodology. For instance, if a row of data had a missing value in any column such as age, numdays, or symptom count it was removed from the dataset. With that being said, we would like to address a couple limitations that may be present in this study.

The first being our group's lack of experience in data science. While our team had solid background in healthcare and clinical studies, no one in our group had come from a coding background or completed an end-to-end analytics project to the extent this project required. While this created some struggle and frustration in the beginning of the project, we adjusted well throughout the semester and became more comfortable performing analysis. If our team were to start this project again, there's a high probability we would view the data from another perspective, be more efficient, and come to different conclusions.

Another limitation may be our handling of boolean values such as medical history. We converted anything resembling "None", N/A, "No" or an empty cell to a 0 and everything else that didn't fit the criteria to a 1. This method may have accidentally converted some data entries to a 1 when a 0 was warranted. This also brings up the issue of treating all medical history as equals.

8. Future Work

1. In a future analysis, it would be wise to develop a way to rank a patient's medical history using natural language processing.
2. Moreover, we can also predict no. of days a person is admitted in the hospital after facing severe effects based on a person's medical history, illness, age, sex and the type of vaccine given. We can do this using linear regression and also use the classification models.

A Appendix – SQL Queries

A.1 SQL queries for deleting rows other than Covid-19 Vaccine

```
DELETE FROM VAERSVAX WHERE VAX_TYPE = 'FLU4';
DELETE FROM VAERSVAX WHERE VAX_TYPE = 'FVARZOS';
DELETE FROM VAERSVAX WHERE VAX_TYPE != 'COVID19';
```

A.2 SQL queries for deleting columns that are not required for our project

```
ALTER TABLE VAERSVAX DROP VAX_LOT, DROP VAX_DOSE_SERIES, DROP VAX_ROUTE, DROP VAX_SITE, DROP VAX_NAME;
```

8. References

1. Centers for Disease Control and Prevention. (2021, February 22). *Symptoms of Covid-19*. Centers for Disease Control and Prevention. Retrieved September 21, 2021, from https://www.cdc.gov/coronavirus/2019-ncov/symptoms-testing/symptoms.html.

2. Centers for Disease Control and Prevention. (2021, September 14). *Selected adverse events reported AFTER COVID-19 Vaccination*. Centers for Disease Control and Prevention. Retrieved September 20, 2021, from https://www.cdc.gov/coronavirus/2019-ncov/vaccines/safety/adverse-events.html
3. *COVID-19 CORONAVIRUS PANDEMIC*. Worldometer. (2021, September 20). Retrieved September 20, 2021, from https://www.worldometers.info/coronavirus/?utm_campaign=homeAdvegas1%3F%22
4. Garg, A. (2021, March 31). *COVID-19 World Vaccine Adverse Reactions*. Kaggle. Retrieved September 27, 2021, from https://www.kaggle.com/ayushggarg/covid19-vaccine-adverse-reactions?select=2021VAERSDATA.csv.
5. Meo, S. A., Bukhari, I. A., Meo, A. S., & Klonoff, D. C. (2021, February 17). *COVID-19 vaccines: Comparison of biological, pharmacological characteristics and adverse effects of Pfizer/BioNTech and Moderna Vaccines*. European Review. Retrieved September 28, 2021, from https://www.europeanreview.org/article/24877.
6. Shimabukuro, T. (2021). Allergic reactions including anaphylaxis after receipt of the first dose of Pfizer-BioNTech COVID-19 vaccine - United States, December 14-23, 2020. *Am J Transplant*, *21*(3), 1332-1337. https://doi.org/10.1111/ajt.16516