California State University, Northridge

COVID-19 Prediction

Group number 1

COMP 541 #17129 Spring 2021

Students: Alisiya Balayan, Bishoy Abdelmalik, Arian Dehghani, Ariel Kohanim, Sergio Ramirez, Natalie Weingart

Professor: Taehyung Wang

Table of Contents

1. Project's objective

- Objective

  The project's objective is to determine when will be the optimal time for the U.S. to return to normal operations and end lockdowns without putting people's lives at risk.

- Success criteria

  The project's success criteria is having a road map of recovery to know when businesses and schools can return to normal operations. To obtain the roadmap of recovery, a prediction model will be built to understand COVID-19 trends.

- Goal

  To predict using machine learning models and data mining techniques when the U.S. population will achieve herd immunity.

2. Background information

- Tackled Problem

  Coronavirus disease (COVID-19) is an infectious disease caused by a newly discovered virus. It has affected every country and city around the world causing health, economic and psychological impacts. Therefore, predicting when the pandemic will come to an end by analyzing data about vaccinations, herd immunity, and hospitalizations is crucial for repairing our societies [1].

- Solution

  The solution we attempted in this project is to use data and models to predict when the pandemic will come to an end by analyzing data about vaccinations, herd immunity, and hospitalizations. The current assumption leading scientists have is that roughly 70% of the population needs to have COVID immunity to achieve herd immunity. Either through vaccination or by natural immunity, meaning through contracting the virus and developing antibodies. As a result our model will attempt to predict how many months are remaining given the current data for the country to achieve herd immunity.

3. Data mining scope

3.1. Datasets used

- COVID-19 World Vaccination Progress
  - This dataset tells us about: which country is using what vaccine, which country's vaccination program is more advanced, and where the rate of vaccinated people per day is higher in terms of percent from the entire population.
  - https://www.kaggle.com/gpreda/covid-world-vaccination-progress
- Novel Coronavirus 2019 Dataset
  - This dataset has daily level information on the number of affected cases, deaths and recovery from COVID-19. It provides us with data about each country and their cases, deaths and recoveries.
  - https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset
- Countries population by year 2020
  - This dataset provides us with the world population and top 20 countries' live clock. It contains population data for the past, present and future.
  - https://www.kaggle.com/eng0mohamed0nabil/population-by-country-2020
- Covid-19 Daily Vaccination
  - This data set contains vaccination data for countries showing how many people are vaccinated daily.
  - https://github.com/owid/covid-19-data/tree/master/public/data/vaccinations

3.2. Data mining technology

3.2.1. Programing languages and programs

3.2.1.1. Python - was used as the main programming language that was used to analyze and build models. As one of the most widely used data mining languages it allowed us to do everything from cleaning and data organization to applying machine learning algorithms.

3.2.1.2. Jupyter Notebook - was used to execute the python code and display the visualizations

3.2.1.3. GitHub (version control) - was used to stay organized and up to date with all the changes to the project.

3.2.2. Python libraries

3.2.2.1. Numpy - was used to make it easier to perform mathematical and logical operations on the data in our datasets.

3.2.2.2. Pandas - was used to allow us to explore and manipulate data in a very efficient manner and helped in ingesting data into python as well as display it and integrate with other visualization tools.

3.2.2.3. Matplotlib - allowed us to visualize the data as it allowed us to create all the needed graphs and plots for our data.

3.2.2.4. Seaborn - was used in visualizing the data in conjunction with matplotlib due to the fact that it provided easier implementation in some instances.

3.2.2.5. Sklearn - was used as our main machine learning library and used to implement the machine learning model.

4. Data exploration and data preprocessing

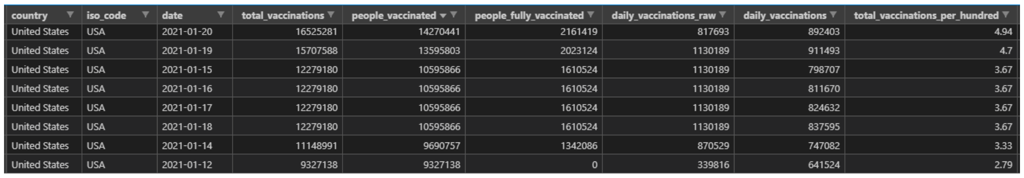
4.1. Data exploration

We explored our datasets using various techniques to examine relationships between attributes and discover the properties of the datasets.

4.2. Data Cleaning

Prior to processing the data, we cleaned and organized the datasets by removing unnecessary columns that we do not need. For example, in the dataset country_vaccinations.csv we removed columns that contained vaccine type, name, source as we are not going to analyze those values. Another thing we did was narrow down all datasets to only U.S. locations.
After selecting the columns that we are interested in working with we proceeded to make sure that our dataset was free of null and NaN values by finding all NaN and null values in the dataset and replacing them with nearest neighbor values. For example, there is no data on December 22, 2020 however vaccines were administered and we have data from the day before and after, so we decided to replace the null value with the nearest neighbor value.

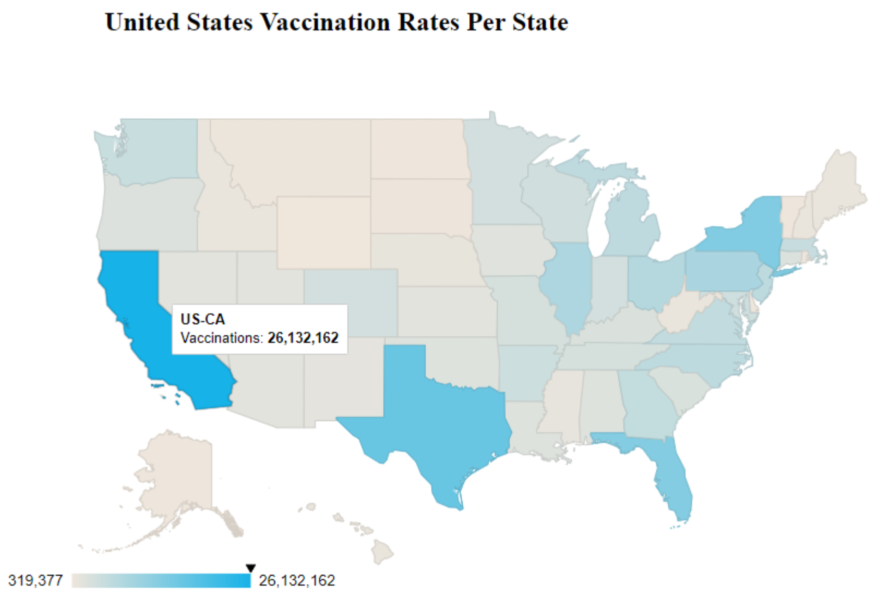
Below is the table that shows vaccination data in the U.S. [2]. Later we will use this cleaned dataset to further preprocess it by finding outliers, filling it missing values, etc.



country	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations	total_vaccinations_per_hundred
United States	USA	2021-01-20	16525281	14270441	2161419	817693	892403	4.94
United States	USA	2021-01-19	15707588	13595803	2023124	1130189	911493	4.7
United States	USA	2021-01-15	12279180	10595866	1610524	1130189	798707	3.67
United States	USA	2021-01-16	12279180	10595866	1610524	1130189	811670	3.67
United States	USA	2021-01-17	12279180	10595866	1610524	1130189	824632	3.67
United States	USA	2021-01-18	12279180	10595866	1610524	1130189	837595	3.67
United States	USA	2021-01-14	11148991	9690757	1342086	870529	747082	3.33
United States	USA	2021-01-12	9327138	9327138	0	339816	641524	2.79

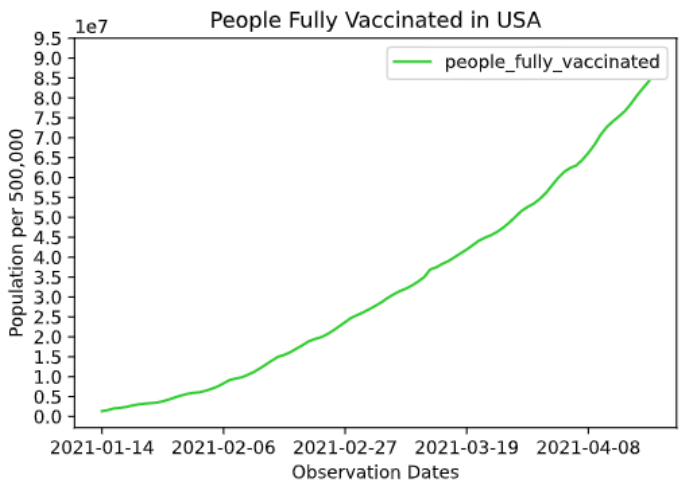
4.3. Clustering of vaccination data

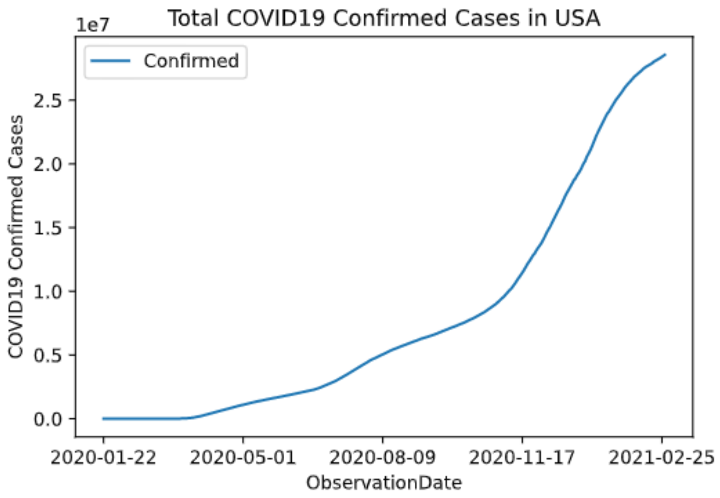
The figure above shows clustering of vaccination data per state where we can observe that California, Texas, Florida and New York have the most prominent clusters (i.e., shown in darker blue compared to the greyed out states). This clustering occurred for a few reasons, one being that these states have high populations and more vaccines are being delivered. Therefore, more people have access to vaccinations. However, to reach 70% immunity in these states, vaccination should continue to increase. For example, California's population is 39 million, and total vaccination (on 4/23/2021) was 26 million.



4.4. Datasets visualizations

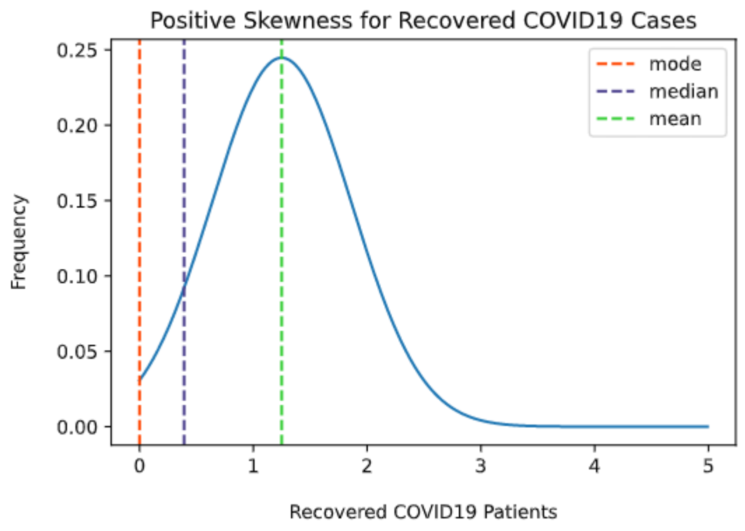
The figures below represent visualizations of our datasets where we are able to see total covid-19 cases, death, recoveries and vaccinations. Our datasets and graphs show data from January 21, 2020 until February 25, 2021 for Covid-19 cases, as well as vaccination data ranging from January 14, 2021 until April 23, 2021.

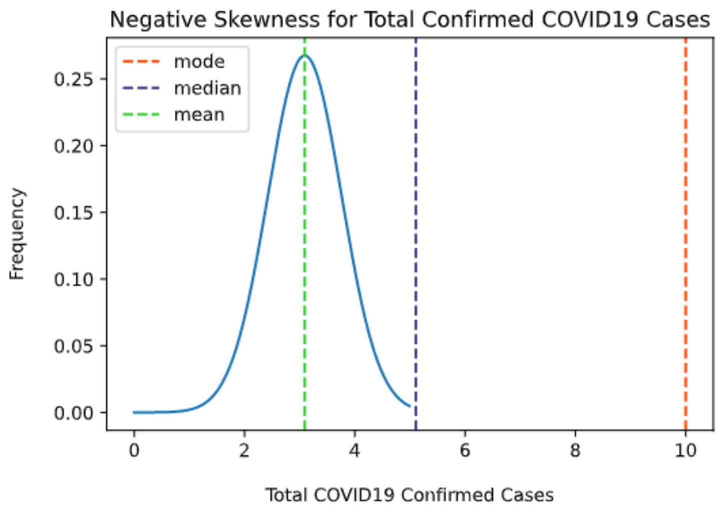


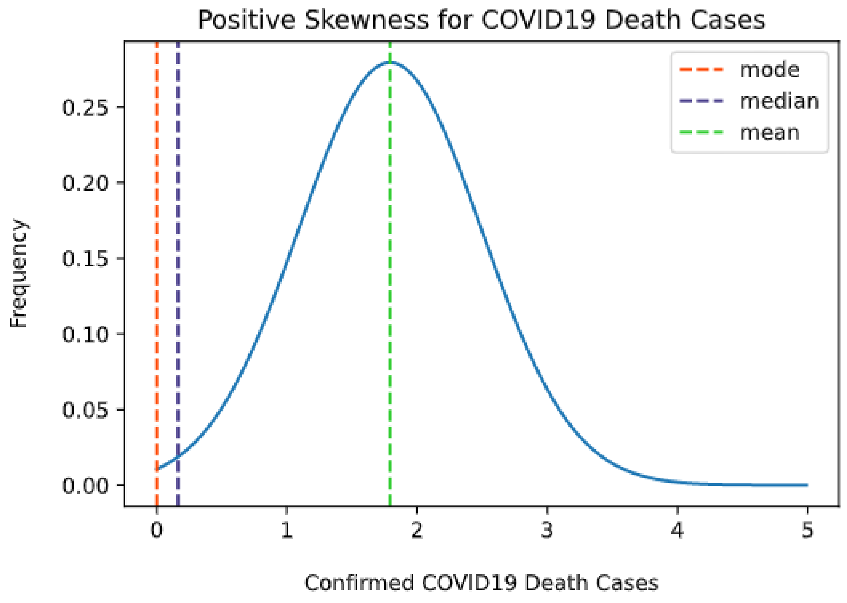


4.5. Skewness of Covid-19 cases data

We discovered the skewness in our covid-19 cases dataset by graphing the skewness graph of each feature in the dataset and observing it in comparison to the mean, mode and median of the dataset.



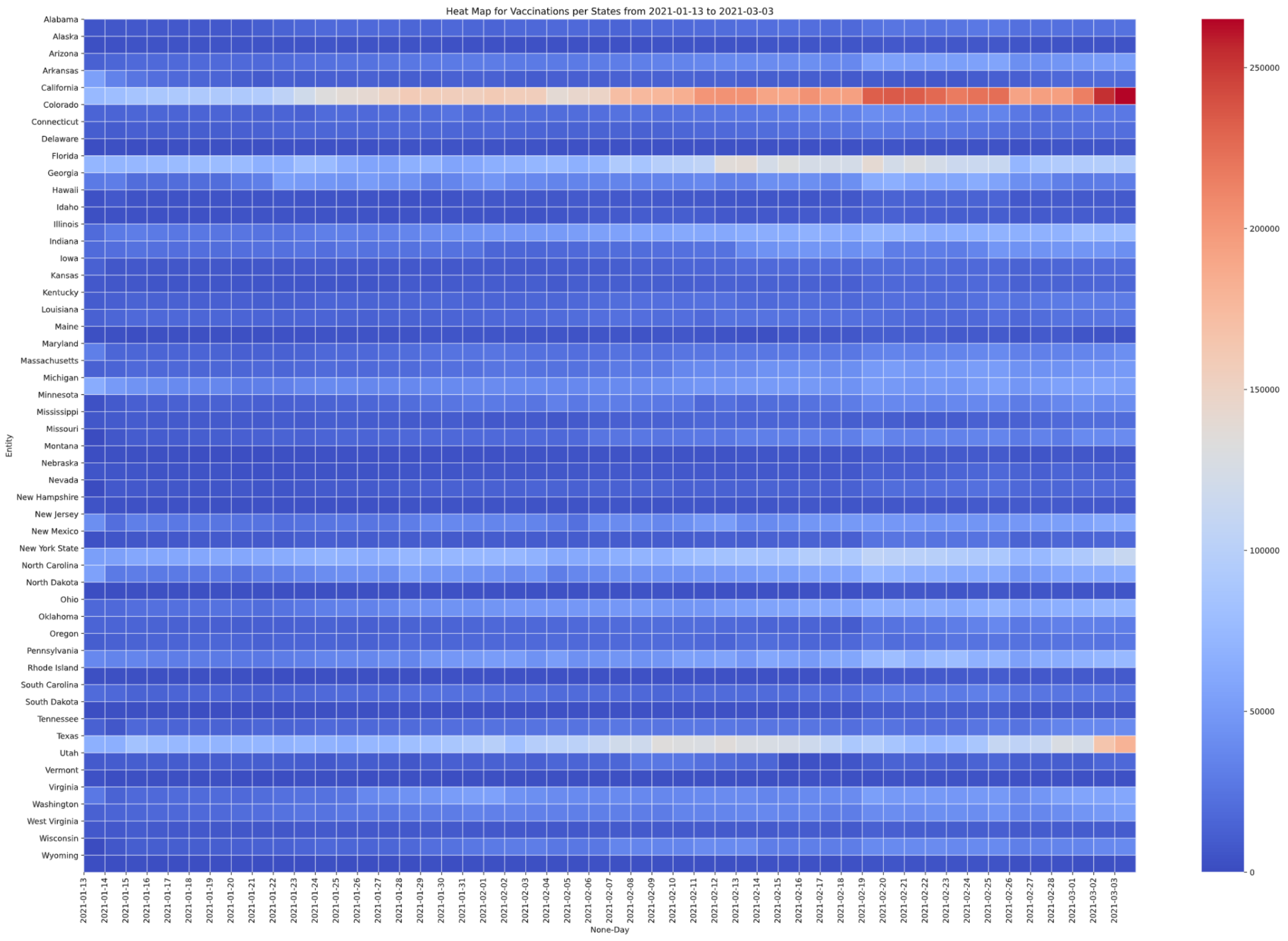




4.6. Kernel Density Estimate - was used to to estimate the probability density function of the random variables to make inferences about the population based on the finite data sample that we have.

4.7. Correlation between attributes by using heatmap

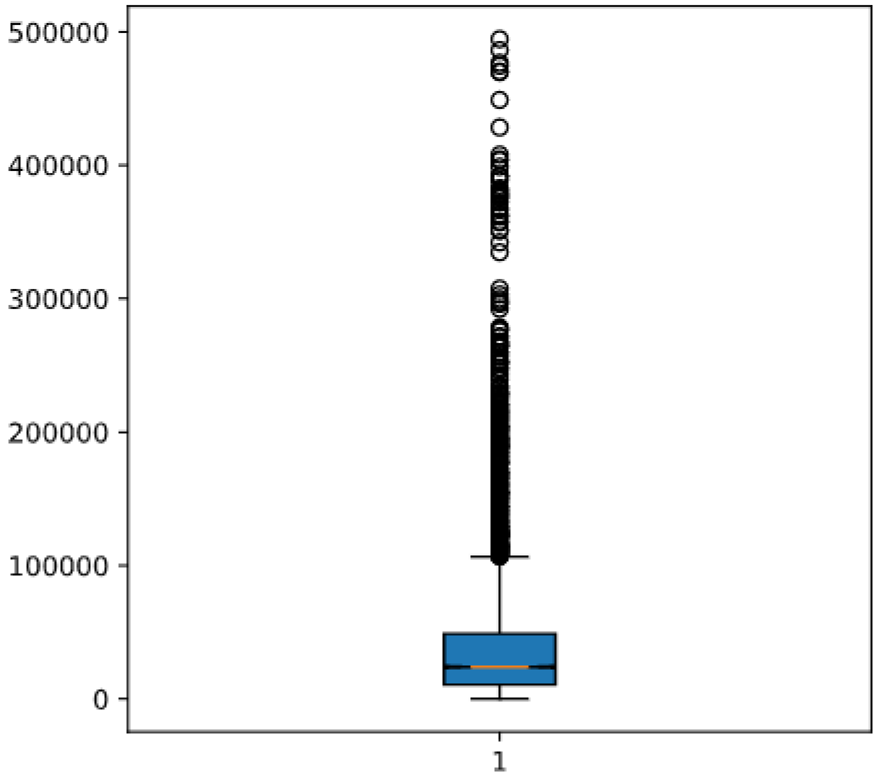
The figure below shows a correlation matrix using seaborn library between vaccinations in each state and the quantity of vaccine distribution over time. We can observe that in states like California and Texas where most cells are light and dark orange, vaccinations have been occurring a lot more compared to states like Alabama where most cells are blue. As of now, California vaccinated 26 million people, therefore there is a correlation between California vaccination and some of the recent months (March and April) where more people are vaccinated and it is continuing to increase.

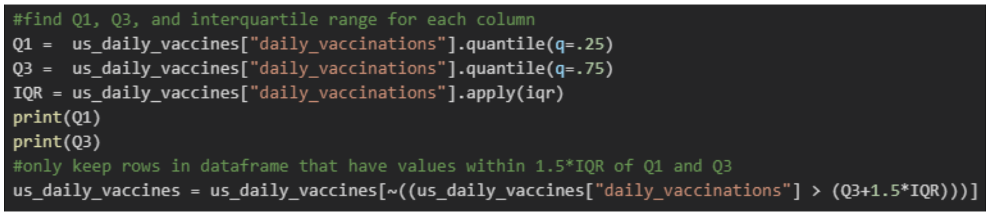


4.8. Data preprocessing and transformation

A. Outliers

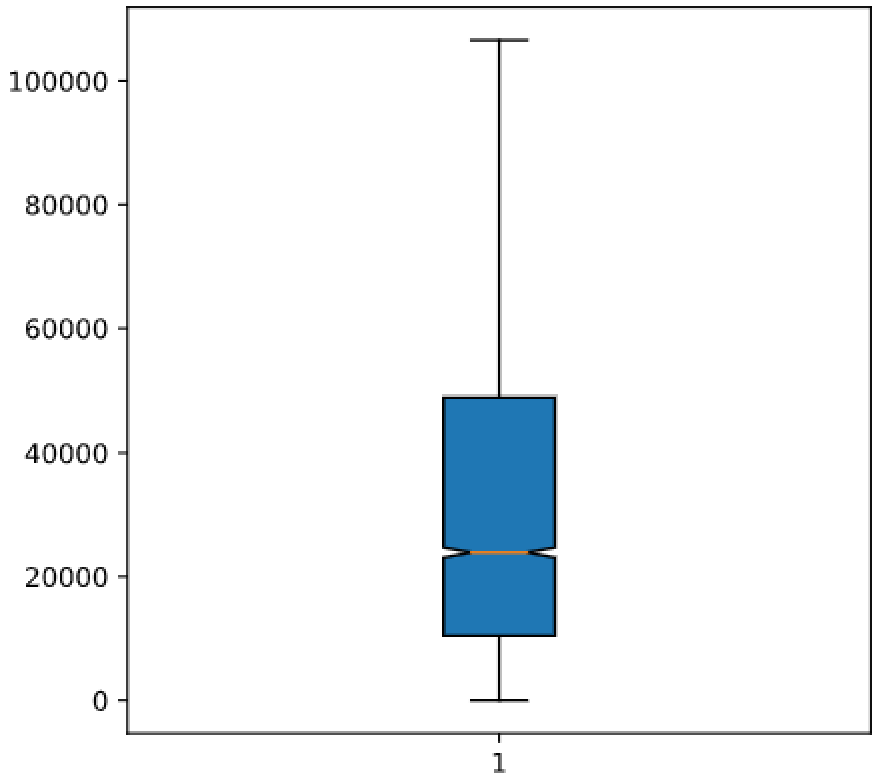
After cleaning the datasets, we graphed points using boxplot in matplotlib to identify if there are any outliers. After seeing the outliers in our datasets we used the IQR scores to remove outliers.





```
#find Q1, Q3, and interquartile range for each column
Q1 =  us_daily_vaccines["daily_vaccinations"].quantile(q=.25)
Q3 =  us_daily_vaccines["daily_vaccinations"].quantile(q=.75)
IQR = us_daily_vaccines["daily_vaccinations"].apply(iqr)
print(Q1)
print(Q3)
#only keep rows in dataframe that have values within 1.5*IQR of Q1 and Q3
us_daily_vaccines = us_daily_vaccines[~((us_daily_vaccines["daily_vaccinations"] > (Q3+1.5*IQR)))]
```

After using IQR test, this is the new output:



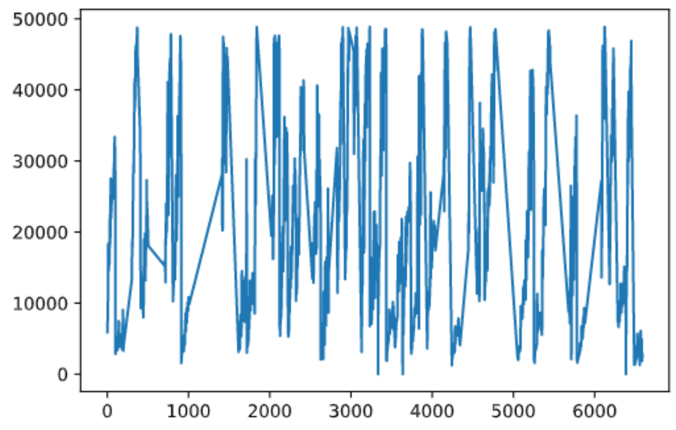
B. Discretize data

We performed discretization on our datasets because most of them contained dates that needed to be broken down into sections for easier evaluation. For example, our vaccination and covid19 cases datasets have data collected from each day, so we combined each month, and created a new csv with monthly covid19 cases instead.

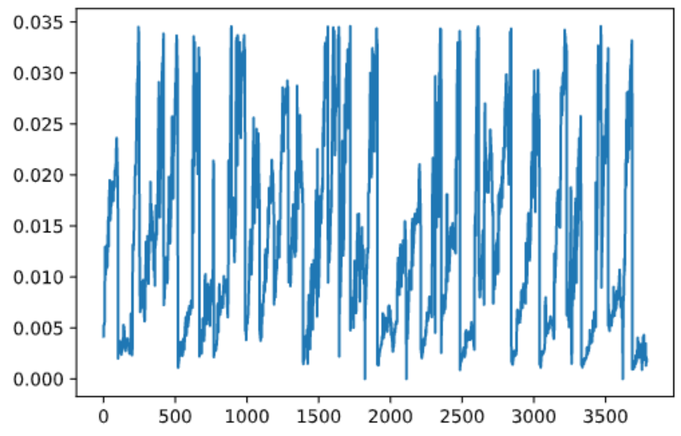
C. Normalize data

We performed data normalization to make our model faster and make calculations faster.
Below is the graph of the datasets before and after normalization.

Before:



After:



4.9. Data Quality Issues

A. Data is not up to date (lacks few months)

Some of our datasets are not up to date and are missing a couple of months. Also, different datasets contain information that is from inconsistent dates, so we had to match all datasets to be up to a certain time and month.

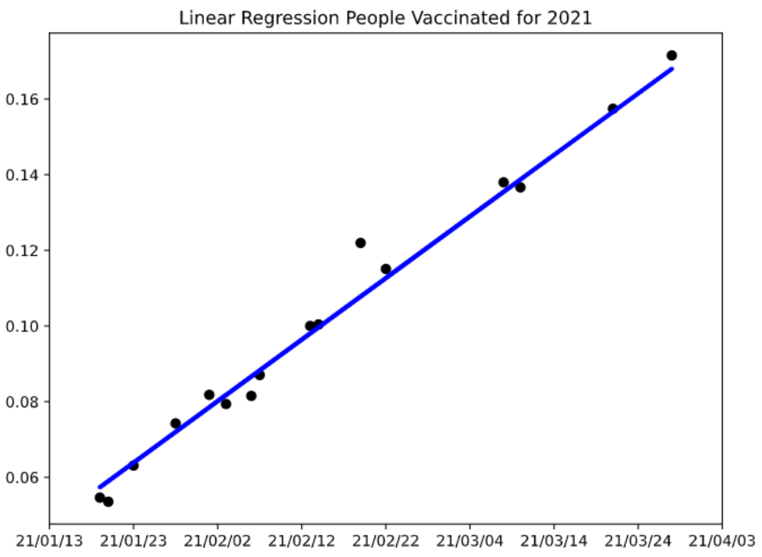
B. Inconsistency between datasets

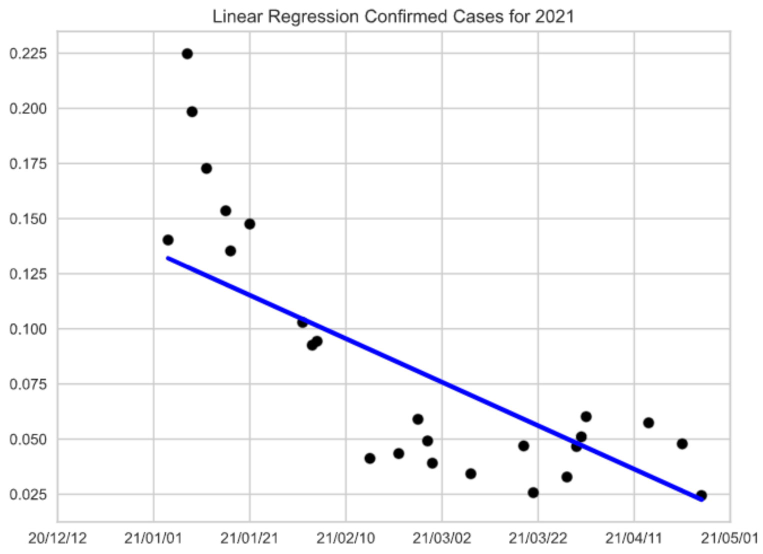
We have 2 vaccination datasets that have inconsistent data points due to the different way of collecting the data. That might have affected our results and might have caused some errors. We calculated error rates and divided our usable data to see how our model works on different data.

5. Modeling

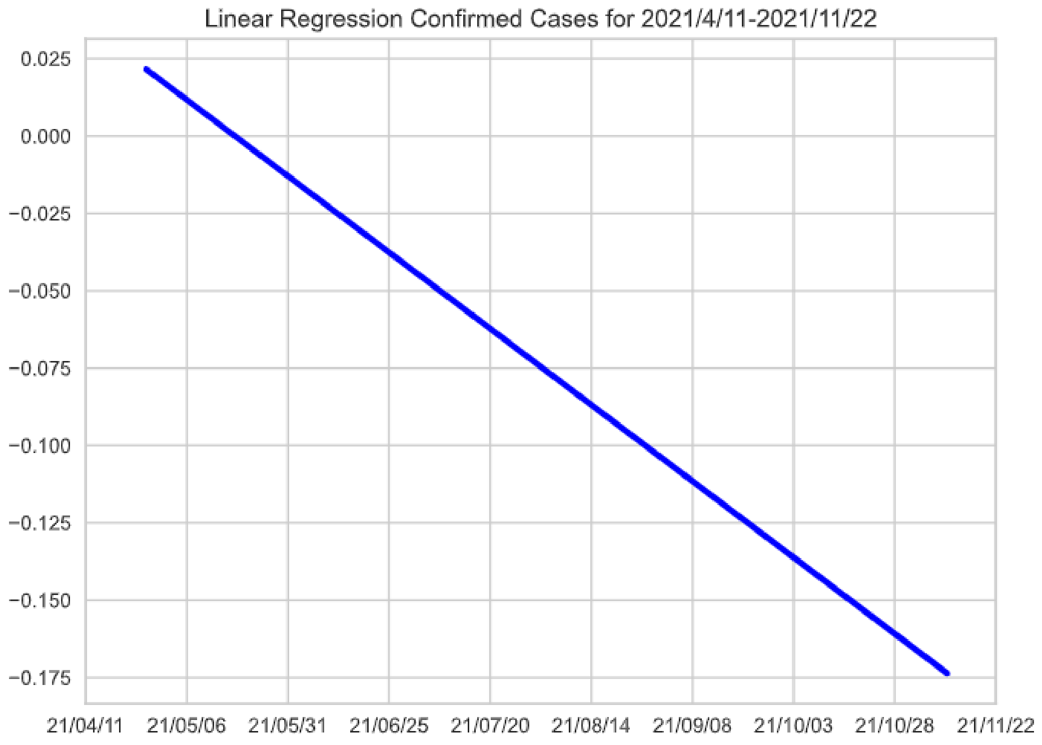
- Linear Regression

We used linear regression as a modeling technique to analyze the relationship between people getting fully vaccinated and confirmed COVID-19 cases. Linear regression is a linear approach to modelling the relationship between a scalar response and independent and dependent variables [3]. It is used to predict the value of a certain variable depending on the value of the other variable (hence, independent and dependent). Our independent variable is time, and dependent variables are people fully vaccinated, and confirmed COVID-19 cases. After independent and dependent variables have been identified, we attempted to make a model that fits linear equation relationship between the data.
We made linear regression models to see the relationship between time and vaccinations, and time and confirmed cases.





We used the regression model to predict the future of confirmed cases as you can see in the figure below. The linear regression shows us that by November of 2022, confirmed COVID-19 cases will significantly drop and vaccination rates will stabilize which means that we can safely come back in person. For example, CSUN plans to safely open more in-person classes by Spring 2022 which supports our results.



- Implementation issues and solutions

  We encountered a few problems with building the model that was solved by tweaking the datasets and normalizing it and matching the time intervals.

6. Assessment of results

Results from the linear regression model show that by November 2022, we will not have confirmed COVID-19 cases given that the rate of decline of new cases remains the same. In addition we can also find out that we will reach 70% of the population vaccinated in 13.7 months.

- Interpret models according to domain knowledge

6.1.1. Data mining success criteria

Our linear regression model aligns with the business goal and confirms our hypothesis. We wanted to see when we will reach 70% of immunity and our model predicted 13.7 months from today (May 10th). Data sources we selected for this project match the goals we have set as in our datasets containing valuable information about COVID-19 cases and vaccinations. The datasets had complete data, all missing values were filled by the nearest neighbor, null and NaN values were removed.

6.1.2. Test result

The result of the model predicted that in 13.7 months, the U.S. population will reach 70% immunity and according to the experts that is an acceptable percentage to get back safely in person. Also, CSUN's plans to open campus next year align with the results of our model as it confirms that we also got the same date. Another source that we found, also concluded similar results (source [4]).

7. Summary of learning experiences

7.1. Overall, we learned about data mining techniques such as CRISP-DM and how to apply it to projects. CRISP-DM model uses hierarchical breakdown that consists of different sets of tasks [5] and in our case CRISP-DM was applied into all 6 parts of our project. Lifecycle of our project was as follows: 1) we developed business understanding and goals of our idea, 2) we found datasets and tried to understand their contents, 3) we cleaned and prepared our datasets for further usage, 4) we built statistical models, 5) we evaluated our models. In our learning experience, we went through all stages of the CRISP-DM lifecycle while working on this project. During the implementation stage, we learned a lot about different statistical models and how each of them differs. After our final presentation, during the QA session, we learned that for our datasets and problems we were trying to solve we could also use the Lasso regression model.

8. References

[1] Patrícia Gonzalez-Dias et al. 2020. Methods for predicting vaccine immunogenicity and reactogenicity. (2020). Retrieved May 21, 2021 from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7062420/

[2] Edouard Mathieu. State-by-state data on COVID-19 vaccinations in the United States. Retrieved May 21, 2021 from https://ourworldindata.org/us-states-vaccinations

[3] Shubham JainI am currently pursuing my B.Tech in Ceramic Engineering from IIT (B.H.U) Varanasi. I am an aspiring data scientist and a ML enthusiast. I am really passionate about changing the world by using artificial intelligence. (2020, October 18). Linear, Ridge and Lasso Regression comprehensive guide for beginners. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/.

[4] Anon. Path to Normality - COVID-19 Vaccine Projections. Retrieved May 21, 2021 from https://covid19-projections.com/path-to-herd-immunity/

[5] Chapman, P. (n.d.). CRISP-DM 1.0 (Tech.).

9. Appendix

Attached is the code used during the project for cleaning processing data and creating the model and validating it.

Linear Regression and Model calculations and verification Code

May 21, 2021

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     import datetime as dt
     from datetime import timedelta
```

1 Importing datsets we are going to use

```
[2]: #Here we can just import all datsets we're going to use
     daily_vaccines = pd.read_csv('cleaned/us-daily-vaccines-discretize-normalized.
      ↪csv')
     display(daily_vaccines)
     covid_19_data = pd.read_csv('cleaned/Novel Corona Virus 2019 Dataset/
      ↪covid_19_data_normalized.csv')
     display(covid_19_data)
     #Should we transform dates or just drop/ignore?
     #We can transorm it using this
     #test_df['Date'].dt.strftime("%Y%m%d").astype(int)
     covid_19_world = pd.read_csv("cleaned/COVID-19 World Vaccination Progress/
      ↪country_vaccinations_normalized.csv")
     display(covid_19_world)
```

```
          Day  daily_vaccinations
0  2021-01-31            0.228328
1  2021-02-28            0.526468
2  2021-03-31            0.818961

    ObservationDate     Confirmed    Deaths  Recovered
0        2020-01-22  2.940369e-09  0.000000        0.0
1        2020-01-23  2.940369e-09  0.000000        0.0
2        2020-01-24  5.880739e-09  0.000000        0.0
3        2020-01-25  5.880739e-09  0.000000        0.0
4        2020-01-26  1.470185e-08  0.000000        0.0
..              …             …         …          …
462      2021-04-28  9.477060e-02  0.090126        0.0
463      2021-04-29  9.494173e-02  0.090260        0.0
```

```
464      2021-04-30  9.511204e-02  0.090423        0.0
465      2021-05-01  9.524525e-02  0.090500        0.0
466      2021-05-02  9.533128e-02  0.090551        0.0

[467 rows x 4 columns]
           date  total_vaccinations  people_vaccinated  \
0    2020-12-20            0.000533           0.000818
1    2020-12-21            0.000588           0.000904
2    2020-12-22            0.000588           0.000904
3    2020-12-23            0.000966           0.001483
4    2020-12-24            0.000966           0.001483
..          …                  …                 …
115  2021-04-14            0.186577           0.182325
116  2021-04-15            0.189953           0.185129
117  2021-04-16            0.193752           0.187954
118  2021-04-17            0.197190           0.190531
119  2021-04-18            0.200575           0.193111

     people_fully_vaccinated
0                   0.000000
1                   0.000000
2                   0.000000
3                   0.000000
4                   0.000000
..                       …
115                 0.200872
116                 0.205632
117                 0.211164
118                 0.216040
119                 0.220734

[120 rows x 4 columns]
```

2 Linear Regression us-daily-vaccines-normalized

[3]:
```
daily_vaccines = pd.read_csv('cleaned/us-daily-vaccines-normalized.csv')
daily_vaccines['Day'] = pd.to_datetime(daily_vaccines['Day'])
daily_vaccines['date_delta'] = (daily_vaccines['Day'] - daily_vaccines['Day'].
 ↪min())  / np.timedelta64(1,'D')
daily_vaccines.head()
```

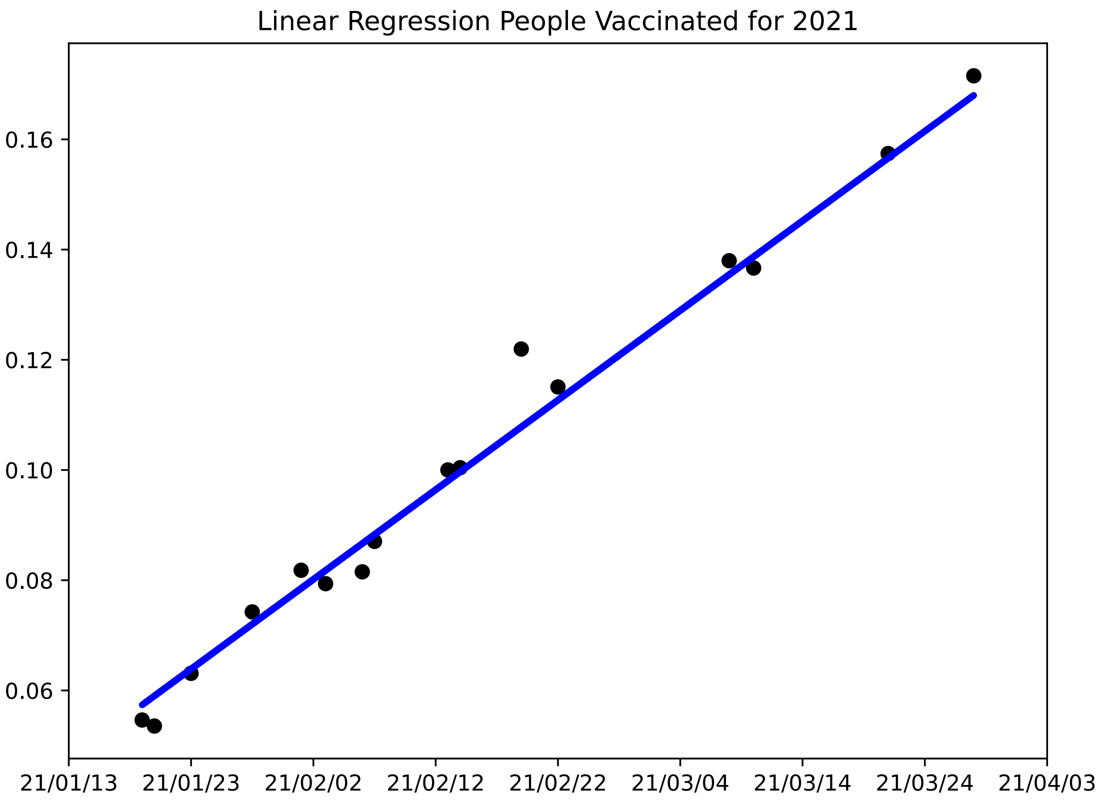
[3]:
```
         Day  daily_vaccinations  date_delta
0 2021-02-09            0.089862        27.0
1 2021-03-13            0.160251        59.0
2 2021-02-13            0.100012        31.0
3 2021-01-13            0.057914         0.0
```

```
4 2021-03-29            0.174406        75.0
```

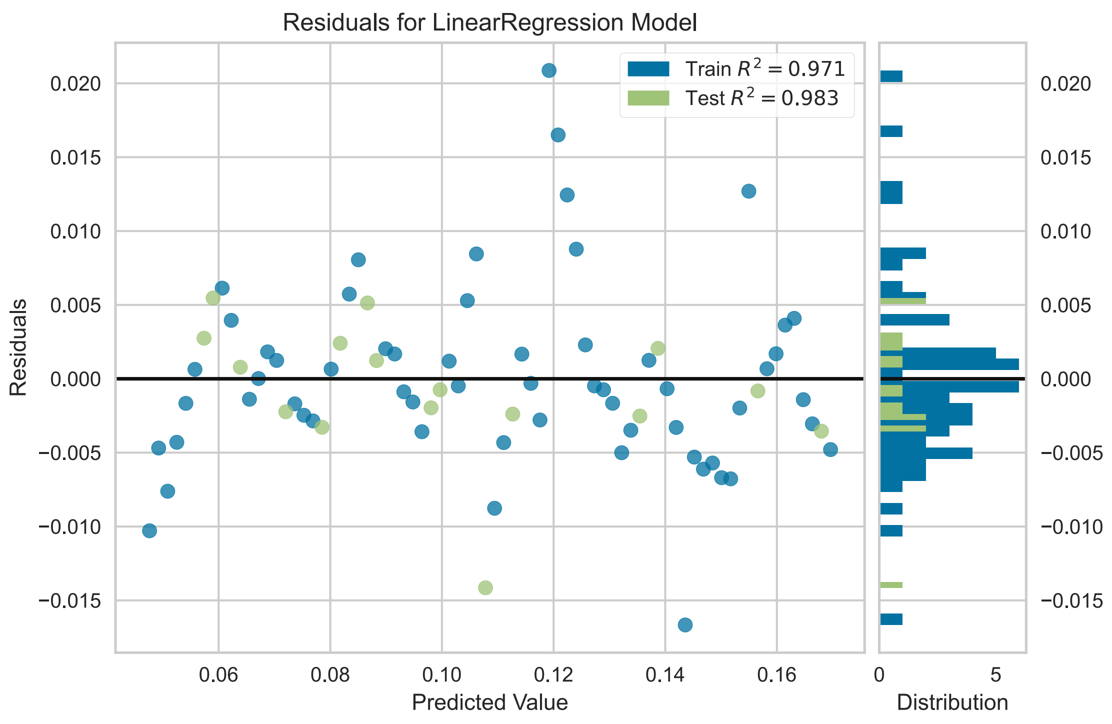
[4]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import sklearn.metrics as metrics
# daily_vaccines = daily_vaccines.drop('Entity',axis=1)
# covid_19_data = covid_19_data.drop("ObservationDate",axis=1)
#These will be the attributes we want to use
X = daily_vaccines.date_delta.values
#We want y to be our target. What's are target going to be? Idk yet lol
y = daily_vaccines.daily_vaccinations.values

X = X.reshape(len(X), 1)
y = y.reshape(len(y), 1)

#Here we can choose a random state so we get consistent results across all␣
 ↪systems.
#80-20 or 70-30 for (test/training is standard)
train_x, test_x, train_y, test_y = train_test_split(X,y,test_size = 0.
 ↪20,random_state=1001)
lr = LinearRegression()
lr.fit(train_x,train_y)
pred_y = lr.predict(test_x)

# print(pred_y)
mse = metrics.mean_squared_error(test_y, pred_y)
mse = f"{mse:.9f}"
print("mean squared error:"+mse)
# print("R-squared: {0}".format(metrics.r2_score(Y,y_pred1)))

# Plot outputs
plt.figure(figsize=(8, 6))

plt.scatter(test_x, test_y,  color='black')
plt.plot(test_x, pred_y, color='blue', linewidth=3)
# plt.xticks(X)
# plt.yticks(())
# print(plt.xticks())
# plt.xticks((plt.xticks()[0],str(plt.xticks()[0])))
xticks=plt.xticks()[0]
xticks_str=[]
first=daily_vaccines['Day'].min()
for tick in xticks:
    # d=dt.datetime.strptime(str(int(tick)),"%Y%m%d")
    xticks_str.append((first+ timedelta(days=int(tick))).strftime("%y/%m/%d"))
plt.xticks(xticks,xticks_str)
plt.title("Linear Regression People Vaccinated for 2021")

plt.show()
```

```
mean squared error:0.000020100
```



[5]:
```
from yellowbrick.regressor import ResidualsPlot
visualizer = ResidualsPlot(lr)
visualizer.fit(train_x, train_y)  # Fit the training data to the visualizer
visualizer.score(test_x, test_y)  # Evaluate the model on the test data
visualizer.show()                 # Finalize and render the figure
```



```
[5]: <matplotlib.axes._subplots.AxesSubplot at 0x13c6f896250>
```

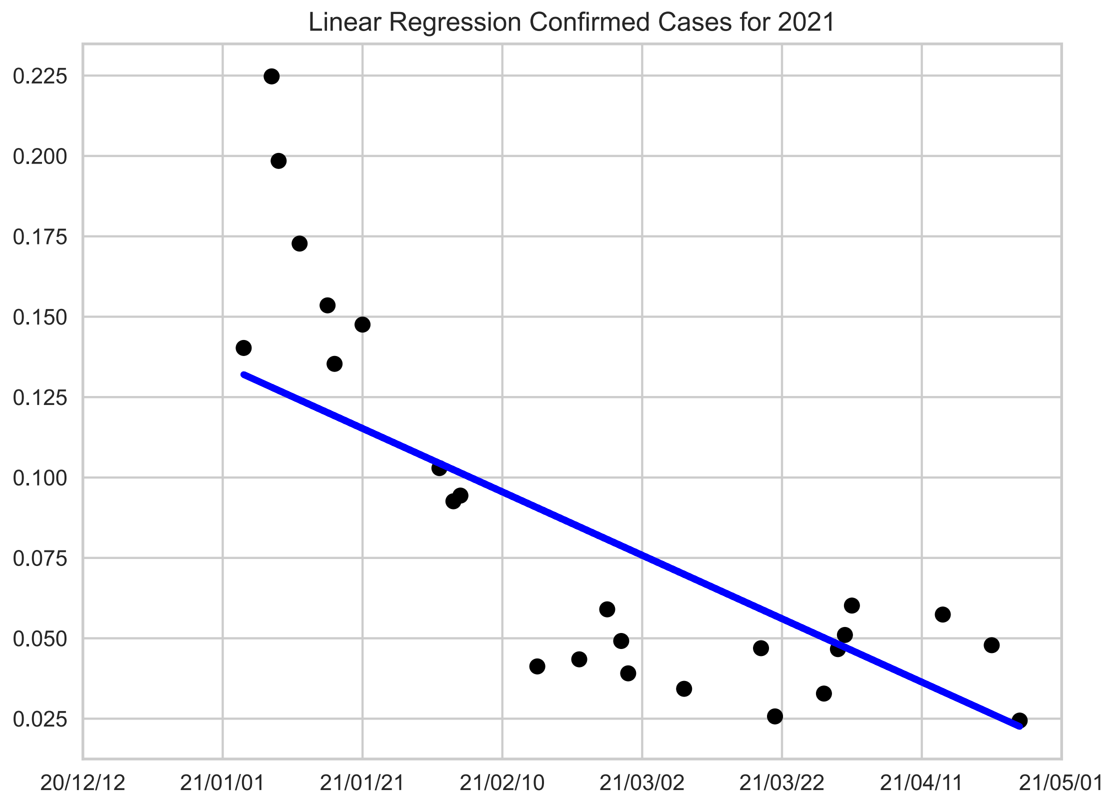
```
[6]: covid_19_data = pd.read_csv('cleaned/Novel Corona Virus 2019 Dataset/
      ↪covid_19_data_2021.csv')
     covid_19_data['ObservationDate'] = pd.
      ↪to_datetime(covid_19_data['ObservationDate'])
     covid_19_data['date_delta'] = (covid_19_data['ObservationDate'] -␣
      ↪covid_19_data['ObservationDate'].min())  / np.timedelta64(1,'D')
     diff=covid_19_data[["Confirmed"]].diff().fillna(0)
     covid_19_data["Confirmed_diff"]=diff["Confirmed"]
     covid_19_data.head()
```

```
[6]:   ObservationDate  Confirmed  Deaths  Recovered  date_delta  Confirmed_diff
     0      2021-01-01   20252991  354242          0         0.0             0.0
     1      2021-01-02   20553301  356749          0         1.0        300310.0
     2      2021-01-03   20762047  358196          0         2.0        208746.0
     3      2021-01-04   20946329  360288          0         3.0        184282.0
     4      2021-01-05   21181440  364002          0         4.0        235111.0
```

```
[7]: from sklearn.linear_model import LinearRegression
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import mean_squared_error, r2_score
     import sklearn.metrics as metrics
     from sklearn import preprocessing
```

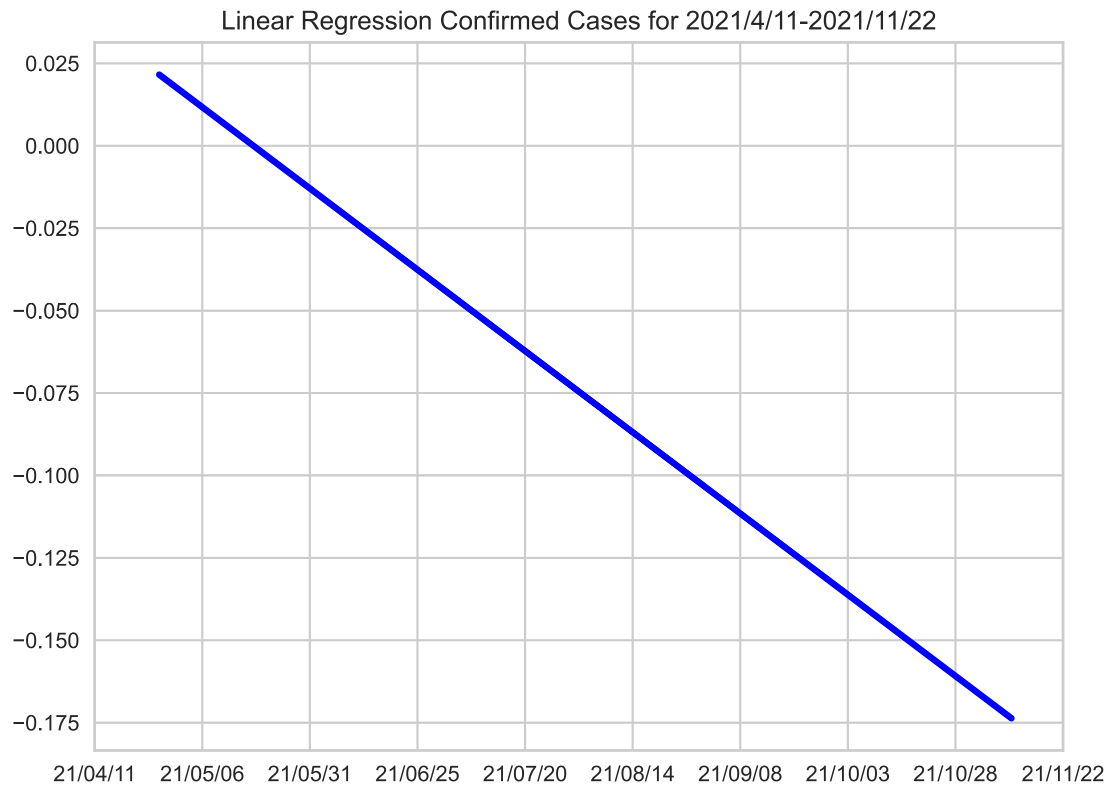
```
# daily_vaccines = daily_vaccines.drop('Entity',axis=1)
# covid_19_data = covid_19_data.drop("ObservationDate",axis=1)
#These will be the attributes we want to use
X = covid_19_data.date_delta.values
#We want y to be our target. What's are target going to be? Idk yet lol
# y = covid_19_data.Confirmed_diff.values
y=preprocessing.normalize([covid_19_data.Confirmed_diff.values])[0]


X = X.reshape(len(X), 1)
y = y.reshape(len(y), 1)

#Here we can choose a random state so we get consistent results across all␣
↪systems.
#80-20 or 70-30 for (test/training is standard)
train_x, test_x, train_y, test_y = train_test_split(X,y,test_size = 0.
↪20,random_state=1001)
lr = LinearRegression()
lr.fit(train_x,train_y)

pred_y = lr.predict(test_x)

mse = metrics.mean_squared_error(test_y, pred_y)
mse = f"{mse:.9f}"
print("mean squared error:"+mse)

# Plot outputs
plt.figure(figsize=(8, 6))

plt.scatter(test_x, test_y,  color='black')
plt.plot(test_x, pred_y, color='blue', linewidth=3)
# plt.xticks(X)
# plt.yticks(())
# print(plt.xticks())
# plt.xticks((plt.xticks()[0],str(plt.xticks()[0])))
xticks=plt.xticks()[0]
xticks_str=[]
first=covid_19_data['ObservationDate'].min()
for tick in xticks:
    # d=dt.datetime.strptime(str(int(tick)),"%Y%m%d")
    xticks_str.append((first+ timedelta(days=int(tick))).strftime("%y/%m/%d"))
plt.xticks(xticks,xticks_str)
plt.title("Linear Regression Confirmed Cases for 2021")

plt.show()
```

```
mean squared error:0.001214634
```



[8]:
```python
import time
start_time = time.time()

max_x=test_x.max()
test_x=np.array([[x] for x in range(int(max_x)+1,int(max_x)+200)])

pred_y = lr.predict(test_x)


# Plot outputs
plt.figure(figsize=(8, 6))

plt.plot(test_x, pred_y, color='blue', linewidth=3)

xticks=plt.xticks()[0]
xticks_str=[]
first=covid_19_data['ObservationDate'].min()
for tick in xticks:
    xticks_str.append((first+ timedelta(days=int(tick))).strftime("%y/%m/%d"))
```

```
plt.xticks(xticks,xticks_str)
plt.title("Linear Regression Confirmed Cases for 2021/4/11-2021/11/22")

plt.show()


print("Model took %s seconds " % (time.time() - start_time))
```



```
Model took 0.14151954650878906 seconds
```

[9]:
```
daily_vaccines = pd.read_csv('cleaned/compare/us-daily-vaccines-normalized.csv')
covid_19_data = pd.read_csv('cleaned/compare/covid_19_data.csv')
daily_vaccines.head()
```

[9]:
```
         Day  daily_vaccinations
0  1/22/2021            0.058307
1  1/23/2021            0.063109
2  1/24/2021            0.066902
3  1/25/2021            0.067127
4  1/26/2021            0.066949
```

```
[10]: covid_19_data['ObservationDate'] = pd.
       ↪to_datetime(covid_19_data['ObservationDate'])
      covid_19_data['date_delta'] = (covid_19_data['ObservationDate'] -␣
       ↪covid_19_data['ObservationDate'].min())  / np.timedelta64(1,'D')
      diff=covid_19_data[["Confirmed"]].diff().fillna(0)
      covid_19_data["Confirmed_diff"]=diff["Confirmed"]
      covid_19_data.head()
```

```
[10]:   ObservationDate  Confirmed  Deaths  Recovered  date_delta  Confirmed_diff
      0      2021-01-22   24902437  421849          0         0.0             0.0
      1      2021-01-23   25073050  425194          0         1.0        170613.0
      2      2021-01-24   25204112  427093          0         2.0        131062.0
      3      2021-01-25   25356081  429066          0         3.0        151969.0
      4      2021-01-26   25503621  433073          0         4.0        147540.0
```

```
[11]: from scipy import stats
      print("Pearson's correlation coefficient:"+str(stats.
       ↪pearsonr(covid_19_data["Confirmed_diff"],daily_vaccines["daily_vaccinations"]␣
       ↪)[0]))
```

```
Pearson's correlation coefficient:-0.6829676521183081
```

3 Linear Regression country_vaccinations_normalized

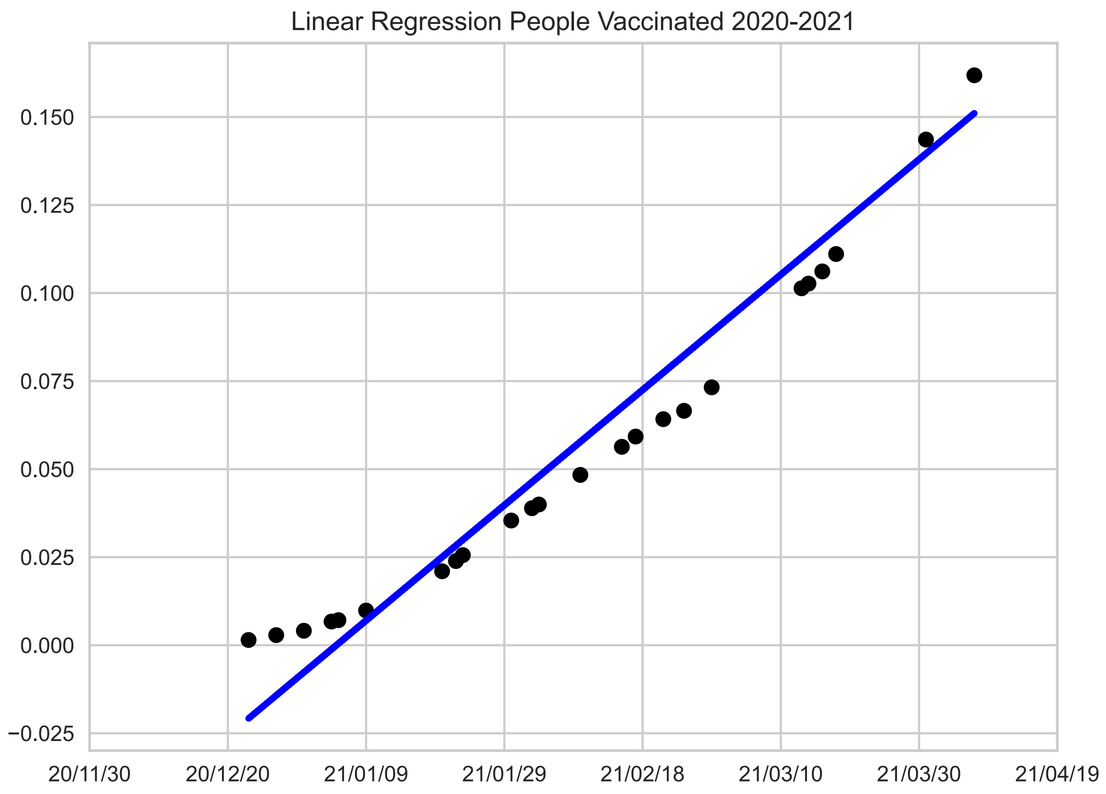
```
[12]: covid_19_world = pd.read_csv("cleaned/COVID-19 World Vaccination Progress/
       ↪country_vaccinations_normalized.csv")
      covid_19_world['date'] = pd.to_datetime(covid_19_world['date'])
      covid_19_world['date_delta'] = (covid_19_world['date'] - covid_19_world['date'].
       ↪min())  / np.timedelta64(1,'D')
      covid_19_world.head()
```

```
[12]:         date  total_vaccinations  people_vaccinated  people_fully_vaccinated  \
      0 2020-12-20            0.000533           0.000818                      0.0
      1 2020-12-21            0.000588           0.000904                      0.0
      2 2020-12-22            0.000588           0.000904                      0.0
      3 2020-12-23            0.000966           0.001483                      0.0
      4 2020-12-24            0.000966           0.001483                      0.0

         date_delta
      0         0.0
      1         1.0
      2         2.0
      3         3.0
      4         4.0
```

```
[ ]:
```

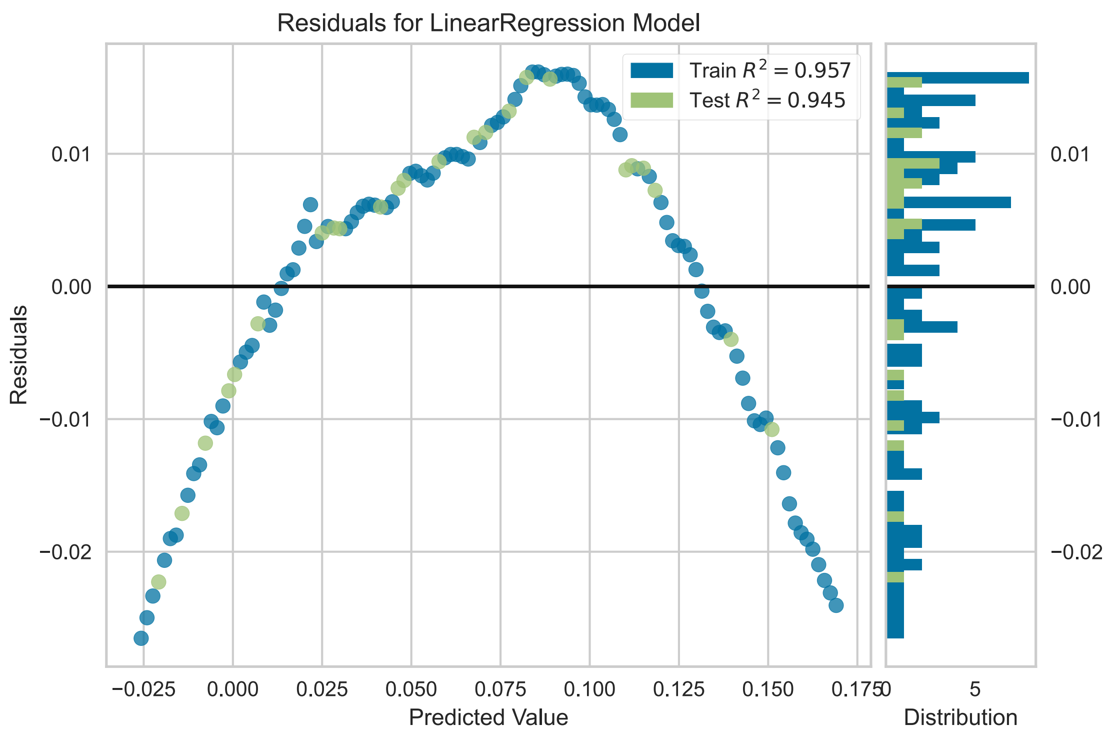
```
[13]: from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import sklearn.metrics as metrics
# daily_vaccines = daily_vaccines.drop('Entity',axis=1)
# covid_19_data = covid_19_data.drop("ObservationDate",axis=1)
#These will be the attributes we want to use
X = covid_19_world.date_delta.values
#We want y to be our target. What's are target going to be? Idk yet lol
y = covid_19_world.people_vaccinated.values

X = X.reshape(len(X), 1)
y = y.reshape(len(y), 1)

#Here we can choose a random state so we get consistent results across all␣
↪systems.
#80-20 or 70-30 for (test/training is standard)
train_x, test_x, train_y, test_y = train_test_split(X,y,test_size = 0.
↪20,random_state=1001)
lr = LinearRegression()
lr.fit(train_x,train_y)
pred_y = lr.predict(test_x)
# print(pred_y)
mse = metrics.mean_squared_error(test_y, pred_y)
mse = f"{mse:.9f}"
print("mean squared error:"+mse)
# print("R-squared: {0}".format(metrics.r2_score(Y,y_pred1)))

# Plot outputs
plt.figure(figsize=(8, 6))

plt.scatter(test_x, test_y,  color='black')
plt.plot(test_x, pred_y, color='blue', linewidth=3)
# plt.xticks(X)
# plt.yticks(())
# print(plt.xticks())
# plt.xticks((plt.xticks()[0],str(plt.xticks()[0])))
xticks=plt.xticks()[0]
xticks_str=[]
first=covid_19_world['date'].min()
for tick in xticks:
    # d=dt.datetime.strptime(str(int(tick)),"%Y%m%d")
    xticks_str.append((first+ timedelta(days=int(tick))).strftime("%y/%m/%d"))
plt.xticks(xticks,xticks_str)
plt.title("Linear Regression People Vaccinated 2020-2021")

plt.show()
```

```
mean squared error:0.000112214
```



```
[ ]:
```

```
[14]: from yellowbrick.regressor import ResidualsPlot
      visualizer = ResidualsPlot(lr)
      visualizer.fit(train_x, train_y)  # Fit the training data to the visualizer
      visualizer.score(test_x, test_y)  # Evaluate the model on the test data
      visualizer.show()                 # Finalize and render the figure
```



```
[14]: <matplotlib.axes._subplots.AxesSubplot at 0x13c6faee820>
```

3.1 Precentage of people fully vaccinated

```
[15]: # TODO total people fully vaccinated /(usa population - fully vaccinated)
      covid_19_world = pd.read_csv("cleaned/COVID-19 World Vaccination Progress/
       ↪country_vaccinations_grouped_by_month.csv")
```

```
[ ]:
```

```
[16]: diff=covid_19_world[["people_fully_vaccinated"]].diff().fillna(0)
      covid_19_world["fully_vaccinated_diff"]=diff["people_fully_vaccinated"]
```

```
[17]: covid_19_world.head()
```

```
[17]:          date  total_vaccinations  people_vaccinated  people_fully_vaccinated  \
      0  2020-12-31           2794588.0          2794588.0                      0.0
      1  2021-01-31          31123299.0         25201143.0                5657142.0
      2  2021-02-28          75236003.0         49772180.0               24779920.0
      3  2021-03-31         150273292.0         97593290.0               54607041.0
      4  2021-04-30         209406814.0        131247546.0               84263408.0

         fully_vaccinated_diff
```

```
0                  0.0
1            5657142.0
2           19122778.0
3           29827121.0
4           29656367.0
```

[18]:
```
average_diff=covid_19_world["fully_vaccinated_diff"].mean()
print("Average number of additional people fully vaccinated every moneth:␣
↪"+str(average_diff))
```

```
Average number of additional people fully vaccinated every moneth: 16852681.6
```

[19]:
```
population=pd.read_csv("cleaned/Countries population by year 2020/
↪population_by_country_2020.csv")
population.head()
```

[19]:
```
  Country (or dependency)  Population (2020)  Density (P/Km )  Med. Age
0           United States          330610570               36        38
```

[20]:
```
population_num=list(population["Population (2020)"])[0]
print(population_num)
```

```
330610570
```

[21]:
```
last_full_vaccinated=covid_19_world["people_fully_vaccinated"][len(covid_19_world["people_full
percentage=(last_full_vaccinated/population_num)*100
print(percentage)
```

```
25.487209316991894
```

[22]:
```
need=(population_num*70)/100
need
```

[22]:
```
231427399.0
```

[23]:
```
print("we will get to 70% in "+str(need/average_diff)+" months")
```

```
we will get to 70% in 13.732378294027699 months
```

clean and process data

May 21, 2021

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
```

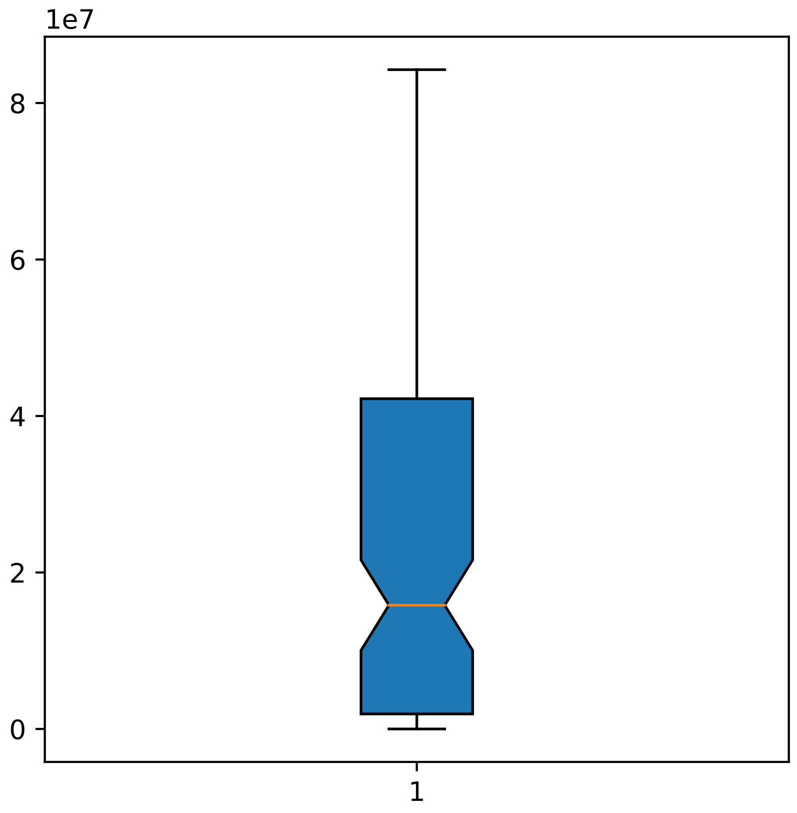
1 COVID-19 World Vaccination Progress/country_vaccinations.csv

[2]:
```
data = pd.read_csv("datasets/COVID-19 World Vaccination Progress/
 ↪country_vaccinations.csv")
usa_values=data.loc[data['iso_code']=='USA']
usa_values=usa_values.fillna(method="ffill")
# usa_values=usa_values.fillna(method="bfill")
usa_values=usa_values.fillna(0)
usa_values=usa_values.drop(columns=['vaccines', 'source_name','source_website'])

# printing nan
print(usa_values.isna().sum())
```

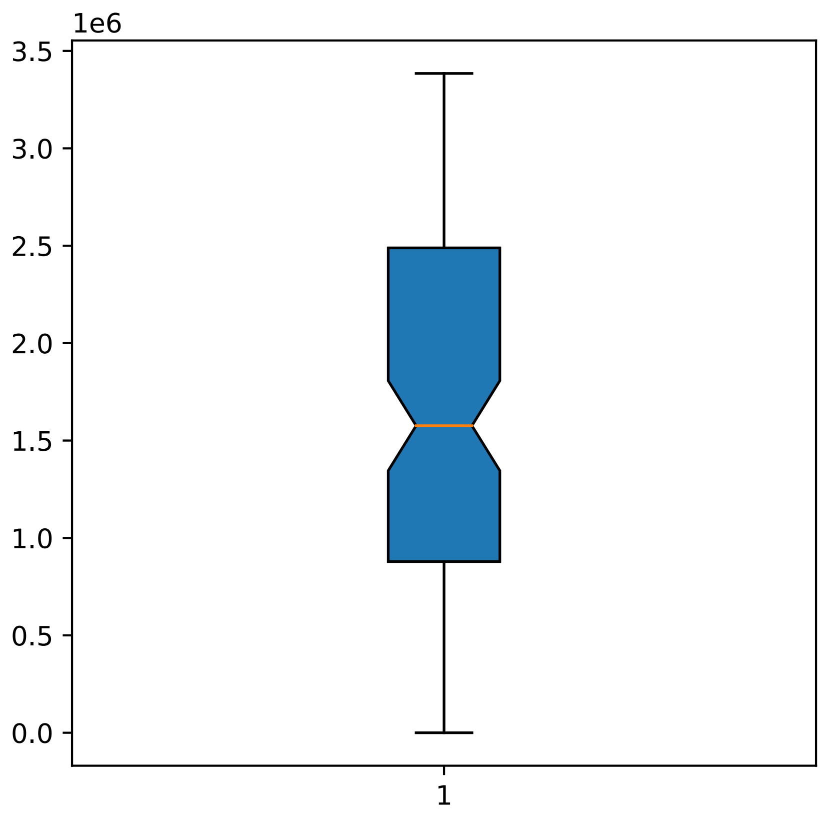
```
country                                0
iso_code                               0
date                                   0
total_vaccinations                     0
people_vaccinated                      0
people_fully_vaccinated                0
daily_vaccinations_raw                 0
daily_vaccinations                     0
total_vaccinations_per_hundred         0
people_vaccinated_per_hundred          0
people_fully_vaccinated_per_hundred    0
daily_vaccinations_per_million         0
dtype: int64
```

1.1 Box plot to detect outliers

[3]:
```
# No outliers in people fully vaccinated
plt.figure(figsize=(5, 5))
plt.boxplot(usa_values['people_fully_vaccinated'], notch=True,␣
 ↪patch_artist=True)
plt.show()
```



[4]:
```
# No outliers in daily vaccinations
plt.figure(figsize=(5, 5))
plt.boxplot(usa_values['daily_vaccinations'], notch=True, patch_artist=True)
plt.show()
```



```
[5]: usa_values.head()
```

```
[5]:          country iso_code        date  total_vaccinations  \
     12061  United States      USA  2020-12-20            556208.0
     12062  United States      USA  2020-12-21            614117.0
     12063  United States      USA  2020-12-22            614117.0
     12064  United States      USA  2020-12-23           1008025.0
     12065  United States      USA  2020-12-24           1008025.0

            people_vaccinated  people_fully_vaccinated  daily_vaccinations_raw  \
     12061           556208.0                      0.0                     0.0
     12062           614117.0                      0.0                 57909.0
     12063           614117.0                      0.0                 57909.0
     12064          1008025.0                      0.0                 57909.0
     12065          1008025.0                      0.0                 57909.0

            daily_vaccinations  total_vaccinations_per_hundred  \
     12061                 0.0                            0.17
     12062             57909.0                            0.18
     12063            127432.0                            0.18
     12064            150606.0                            0.30
```

```
12065          191001.0                           0.30

       people_vaccinated_per_hundred  people_fully_vaccinated_per_hundred  \
12061                           0.17                                  0.0
12062                           0.18                                  0.0
12063                           0.18                                  0.0
12064                           0.30                                  0.0
12065                           0.30                                  0.0

       daily_vaccinations_per_million
12061                             0.0
12062                           173.0
12063                           381.0
12064                           450.0
12065                           571.0
```

[6]:
```
usa_values=usa_values[['date','total_vaccinations','people_vaccinated','people_fully_vaccinate
usa_values.head()
```

[6]:
```
             date  total_vaccinations  people_vaccinated  \
12061  2020-12-20            556208.0           556208.0
12062  2020-12-21            614117.0           614117.0
12063  2020-12-22            614117.0           614117.0
12064  2020-12-23           1008025.0          1008025.0
12065  2020-12-24           1008025.0          1008025.0

       people_fully_vaccinated
12061                      0.0
12062                      0.0
12063                      0.0
12064                      0.0
12065                      0.0
```

[7]:
```
import os
if not os.path.exists('cleaned/COVID-19 World Vaccination Progress'):
    os.makedirs('cleaned/COVID-19 World Vaccination Progress')
f=open(f'cleaned/COVID-19 World Vaccination Progress/country_vaccinations.csv',␣
 ↪"w")
usa_values.to_csv(path_or_buf=f,index=False,line_terminator='\n')
f.close()
```

[8]:
```
usa_values['date'] = pd.to_datetime(usa_values.date)

groupedByMonth=usa_values.groupby(pd.Grouper(key='date', freq='1M')).max().
 ↪reset_index()
groupedByMonth
```

[8]:
```
         date  total_vaccinations  people_vaccinated  people_fully_vaccinated
0 2020-12-31           2794588.0          2794588.0                      0.0
1 2021-01-31          31123299.0         25201143.0                5657142.0
2 2021-02-28          75236003.0         49772180.0               24779920.0
3 2021-03-31         150273292.0         97593290.0               54607041.0
4 2021-04-30         209406814.0        131247546.0               84263408.0
```

[9]:
```
import os
if not os.path.exists('cleaned/COVID-19 World Vaccination Progress'):
    os.makedirs('cleaned/COVID-19 World Vaccination Progress')
f=open(f'cleaned/COVID-19 World Vaccination Progress/
 ↪country_vaccinations_grouped_by_month.csv', "w")
groupedByMonth.to_csv(path_or_buf=f,index=False,line_terminator='\n')
f.close()
```

[10]:
```
data = usa_values['total_vaccinations']
normalized_arr = preprocessing.normalize([data])
usa_values['total_vaccinations']=normalized_arr[0]
data = usa_values['people_vaccinated']
normalized_arr = preprocessing.normalize([data])
usa_values['people_vaccinated']=normalized_arr[0]
data = usa_values['people_fully_vaccinated']
normalized_arr = preprocessing.normalize([data])
usa_values['people_fully_vaccinated']=normalized_arr[0]
```

[11]:
```
usa_values.head()
```

[11]:
```
             date  total_vaccinations  people_vaccinated  \
12061  2020-12-20            0.000533           0.000818
12062  2020-12-21            0.000588           0.000904
12063  2020-12-22            0.000588           0.000904
12064  2020-12-23            0.000966           0.001483
12065  2020-12-24            0.000966           0.001483

       people_fully_vaccinated
12061                      0.0
12062                      0.0
12063                      0.0
12064                      0.0
12065                      0.0
```

[12]:
```
import os
if not os.path.exists('cleaned/COVID-19 World Vaccination Progress'):
    os.makedirs('cleaned/COVID-19 World Vaccination Progress')
f=open(f'cleaned/COVID-19 World Vaccination Progress/
 ↪country_vaccinations_normalized.csv', "w")
usa_values.to_csv(path_or_buf=f,index=False,line_terminator='\n')
```

```
f.close()
```

[13]:
```
usa_values['date'] = pd.to_datetime(usa_values.date)

groupedByMonth=usa_values.groupby(pd.Grouper(key='date', freq='1M')).max().
 ↪reset_index()
groupedByMonth.head()
```

[13]:
```
        date  total_vaccinations  people_vaccinated  people_fully_vaccinated
0 2020-12-31            0.002677           0.004112                 0.000000
1 2021-01-31            0.029811           0.037080                 0.014819
2 2021-02-28            0.072063           0.073232                 0.064913
3 2021-03-31            0.143936           0.143594                 0.143047
4 2021-04-30            0.200575           0.193111                 0.220734
```

[14]:
```
import os
if not os.path.exists('cleaned/COVID-19 World Vaccination Progress'):
    os.makedirs('cleaned/COVID-19 World Vaccination Progress')
f=open(f'cleaned/COVID-19 World Vaccination Progress/
 ↪country_vaccinations_normalized_grouped_by_month.csv', "w")
groupedByMonth.to_csv(path_or_buf=f,index=False,line_terminator='\n')
f.close()
```

[ ]:
```

```

2 us-daily-vaccines.csv

[15]:
```
us_daily_vaccines=pd.read_csv("datasets/us-daily-vaccines.csv")



states_to_exclude=["American Samoa","Bureau of Prisons","Dept of␣
 ↪Defense","District of Columbia","Federated States of␣
 ↪Micronesia","Guam","Indian Health Svc","Long Term Care","Marshall␣
 ↪Islands","Northern Mariana Islands","Puerto Rico","Republic of␣
 ↪Palau","Veterans Health","United States","Virgin Islands"]

us_daily_vaccines=us_daily_vaccines.drop(columns=['Code'])

count =0
for s in states_to_exclude:
    us_daily_vaccines=us_daily_vaccines[us_daily_vaccines['Entity']!
 ↪=states_to_exclude[count]]
    count+=1

# printing nan
print(us_daily_vaccines.isna().sum())
```

```
us_daily_vaccines.describe()
```

```
Entity                0
Day                   0
daily_vaccinations    0
dtype: int64
```

[15]:
```
       daily_vaccinations
count         5050.000000
mean         40493.919010
std          53026.532755
min              0.000000
25%          10468.250000
50%          23898.500000
75%          48924.000000
max         494575.000000
```

[16]:
```
us_daily_vaccines.head()
```

[16]:
```
    Entity         Day  daily_vaccinations
0  Alabama  2021-01-13                5906
1  Alabama  2021-01-14                7083
2  Alabama  2021-01-15                7478
3  Alabama  2021-01-16                7498
4  Alabama  2021-01-17                7509
```

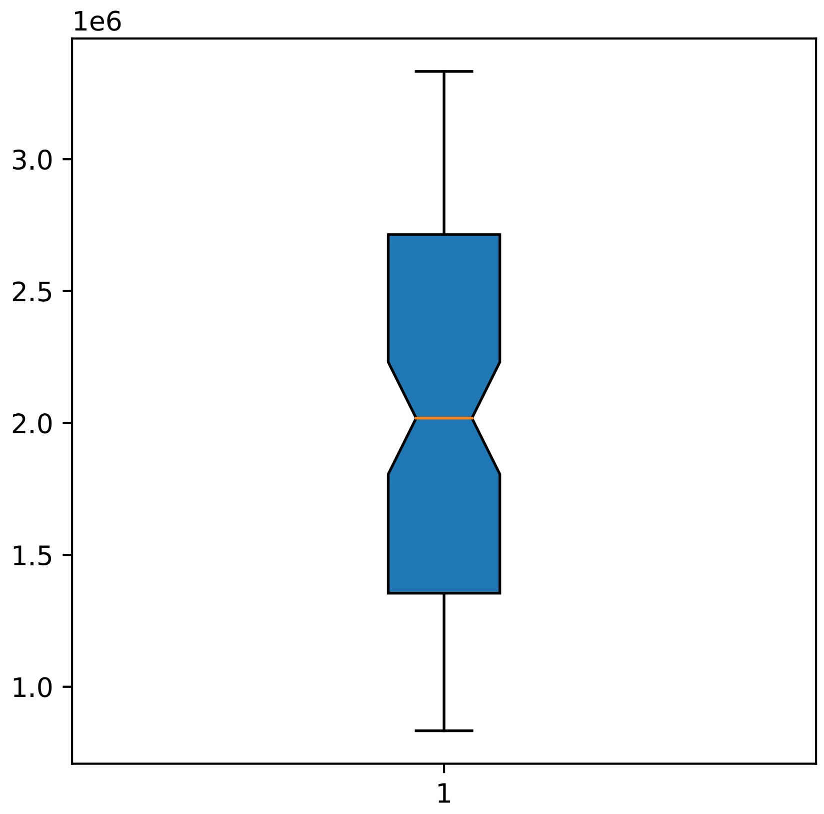
[17]:
```
days=set(us_daily_vaccines["Day"])
temp={"Day":[],"daily_vaccinations":[]}
for d in days:
    temp["Day"].append(d)
    sum=us_daily_vaccines[us_daily_vaccines["Day"]==d]["daily_vaccinations"].
 ↪sum()
    temp["daily_vaccinations"].append(sum)
us_daily_vaccines=pd.DataFrame(temp)
```

[ ]:
```

```

[ ]:
```

```

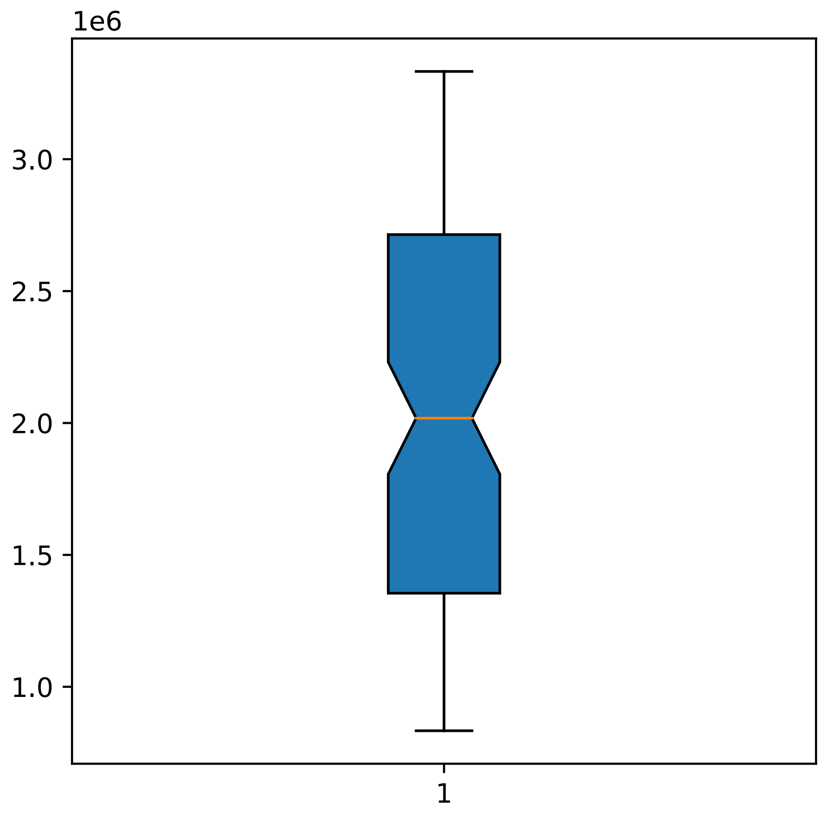
2.1 Boxplot to detect outliers.

[18]:
```
plt.figure(figsize=(5, 5))
plt.boxplot(us_daily_vaccines['daily_vaccinations'], notch=True,␣
 ↪patch_artist=True)
plt.show()
```



2.1.1 Hide outliers

```
[19]: plt.figure(figsize=(5, 5))
      plt.boxplot(us_daily_vaccines['daily_vaccinations'], notch=True,␣
       ↪patch_artist=True, showfliers=False)
      plt.show()
```



2.2 IQR range to find statistical dispersion.

[20]:
```
from scipy.stats import iqr

data = us_daily_vaccines['daily_vaccinations']
iqr(data, axis=0)
```

[20]:
```
1359464.0
```

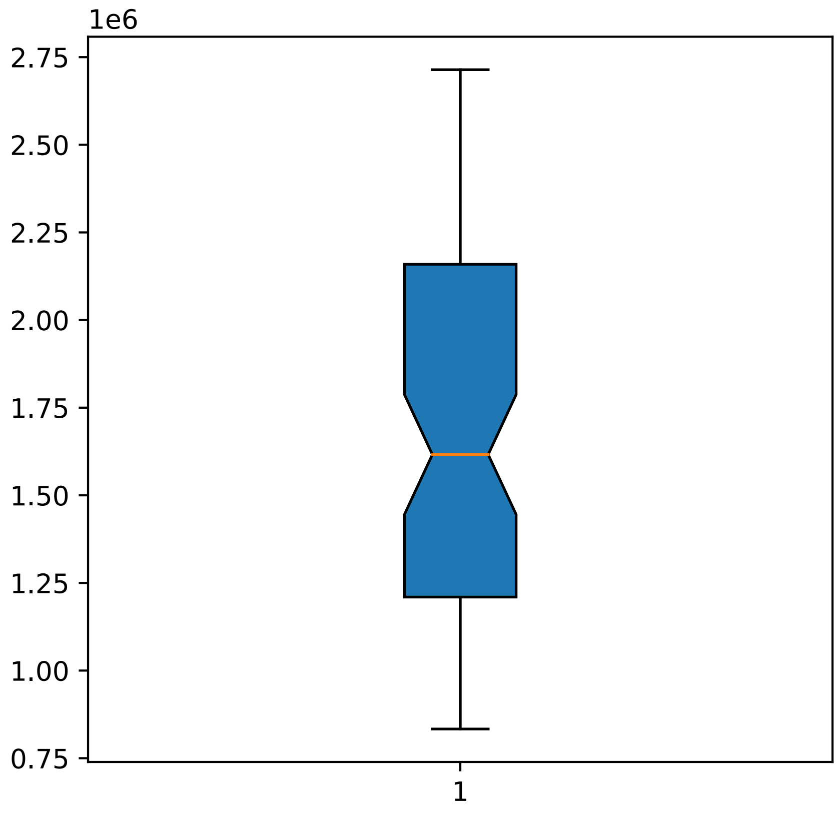
[21]:
```
'''
Warning!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
every time you run this block it will remove outliers assumed to be in Q3
'''
print(us_daily_vaccines.shape)
#find Q1, Q3, and interquartile range for each column
Q1 =  us_daily_vaccines["daily_vaccinations"].quantile(q=.25)
Q3 =  us_daily_vaccines["daily_vaccinations"].quantile(q=.75)
IQR = us_daily_vaccines["daily_vaccinations"].apply(iqr)
print(Q1)
print(Q3)
```

```
#only keep rows in dataframe that have values within 1.5*IQR of Q1 and Q3
us_daily_vaccines =␣
 ↪us_daily_vaccines[~((us_daily_vaccines["daily_vaccinations"] > (Q3+1.
 ↪5*IQR)))]

print(us_daily_vaccines.shape)
```

```
(101, 2)
1354719.0
2714183.0
(76, 2)
```

2.3 Boxplot graph without outliers

[22]:
```
plt.figure(figsize=(5, 5))
plt.boxplot(us_daily_vaccines['daily_vaccinations'], notch=True,␣
 ↪patch_artist=True)
plt.show()
```



2.4 group by month whole USA (discretize data)

[23]:
```
us_daily_vaccines['Day'] = pd.to_datetime(us_daily_vaccines.Day)

groupedByMonth=us_daily_vaccines.groupby(pd.Grouper(key='Day', freq='1M')).
↪sum().reset_index()
groupedByMonth.head()
```

[23]:
```
         Day  daily_vaccinations
0 2021-01-31            18665480
1 2021-02-28            43038036
2 2021-03-31            66948906
```

[24]:
```
import os
if not os.path.exists('cleaned'):
    os.makedirs('cleaned')
f=open(f'cleaned/us-daily-vaccines-discretize-not-normalized.csv', "w")
groupedByMonth.to_csv(path_or_buf=f,index=False,line_terminator='\n')
f.close()
```

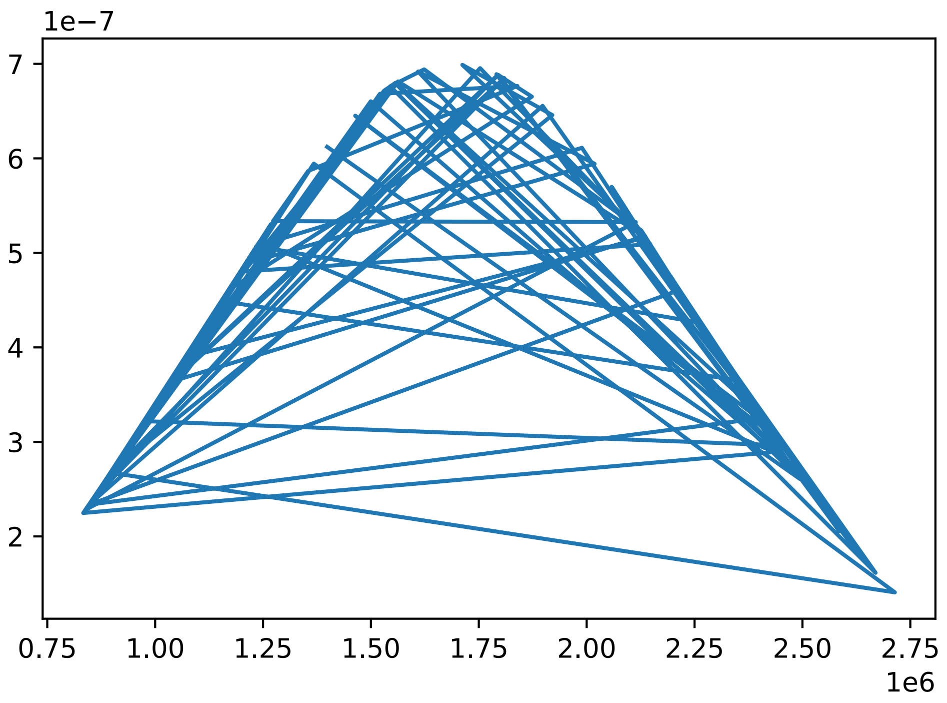
[25]:
```
data = groupedByMonth['daily_vaccinations']
normalized_arr = preprocessing.normalize([data])
groupedByMonth['daily_vaccinations'] = normalized_arr[0]
```

[26]:
```
import os
if not os.path.exists('cleaned'):
    os.makedirs('cleaned')
f=open(f'cleaned/us-daily-vaccines-discretize-normalized.csv', "w")
groupedByMonth.to_csv(path_or_buf=f,index=False,line_terminator='\n')
f.close()
```

2.5 Graph normal distribution of the dataset

[27]:
```
from scipy.stats import norm
import statistics

data = us_daily_vaccines['daily_vaccinations']
mean = statistics.mean(data)
sd = statistics.stdev(data)
# norm.pdf(data, mean, sd)
# plt.plot(norm.pdf(data, mean, sd))
# plt.show()
plt.plot(data, norm.pdf(data, mean, sd))
# plt.show()
```

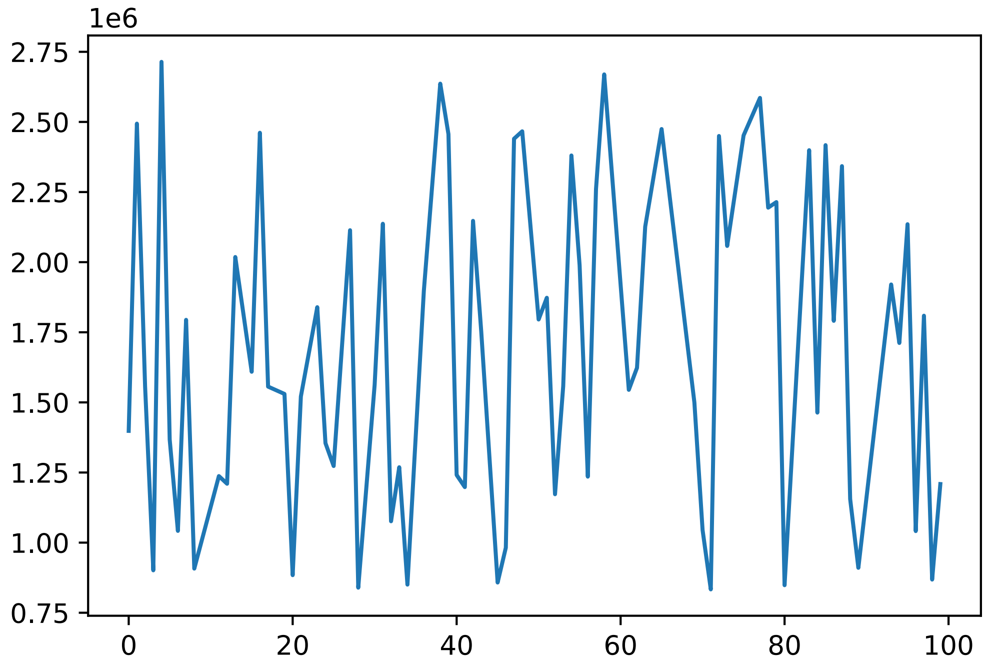
[27]:
```
[<matplotlib.lines.Line2D at 0x19f0eaa1b20>]
```

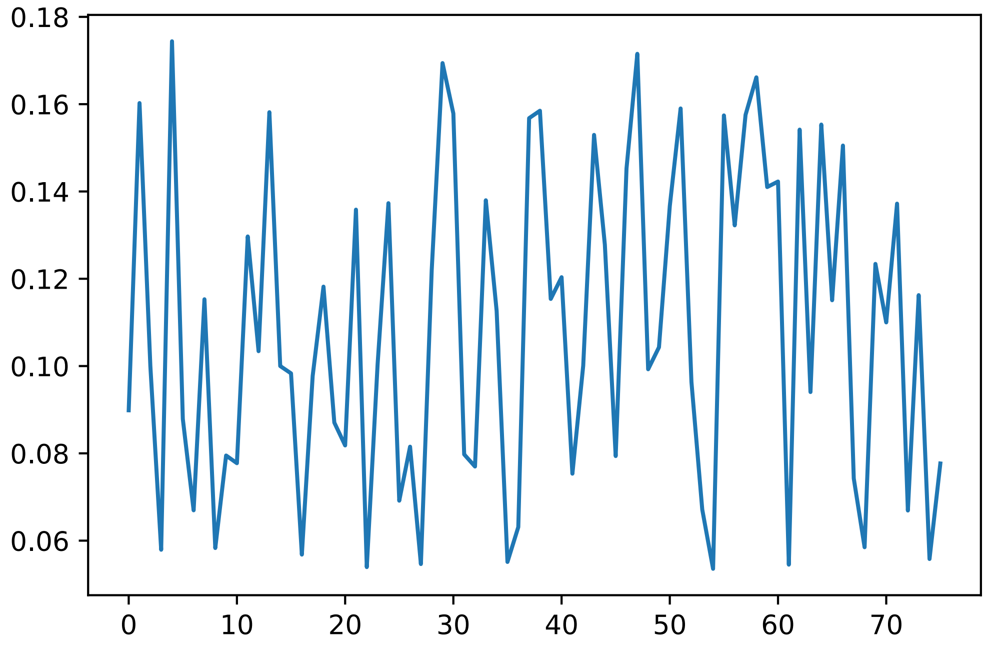


2.6 Preprocess and normalize the data

[28]:
```python
data = us_daily_vaccines['daily_vaccinations']
normalized_arr = preprocessing.normalize([data])
print(normalized_arr)

plt.plot( us_daily_vaccines['daily_vaccinations'])
plt.show()
plt.plot(normalized_arr[0])
plt.show()
```

```
[[0.08986206 0.16025117 0.10001154 0.05791397 0.17440574 0.08788024
  0.06694866 0.11529672 0.05830735 0.07950367 0.0777454  0.12969219
  0.1034085  0.15816108 0.100003   0.09832152 0.05680405 0.09772958
  0.11819671 0.08705042 0.08181083 0.13583293 0.0539385  0.10042086
  0.13731232 0.06915358 0.08152732 0.05463788 0.1219528  0.16941013
  0.15784712 0.07975434 0.07697939 0.13797835 0.11264161 0.05511936
  0.06310858 0.15678874 0.15849374 0.11537794 0.1203544  0.07534533
  0.10010652 0.15295882 0.12780457 0.07937709 0.14525573 0.17153358
  0.09926372 0.10431196 0.13665195 0.15901641 0.09636411 0.06712735
  0.05354493 0.15742276 0.13223408 0.15754858 0.16614593 0.14100711
  0.14227529 0.05450159 0.15416942 0.09404758 0.15532676 0.11507645
  0.15051621 0.07425585 0.05848926 0.12340283 0.11000187 0.13720758
  0.06690168 0.11624529 0.05579342 0.07767581]]
```





[29]:
```
import os
if not os.path.exists('cleaned'):
    os.makedirs('cleaned')
```

```
f=open(f'cleaned/us-daily-vaccines.csv', "w")
us_daily_vaccines.to_csv(path_or_buf=f,index=False,line_terminator='\n')
f.close()
```

```
[30]: us_daily_vaccines['daily_vaccinations'] = normalized_arr[0]
      import os
      if not os.path.exists('cleaned'):
          os.makedirs('cleaned')
      f=open(f'cleaned/us-daily-vaccines-normalized.csv', "w")
      us_daily_vaccines.to_csv(path_or_buf=f,index=False,line_terminator='\n')
      f.close()
```

```
[ ]:
```

3 Novel Corona Virus 2019 Dataset/covid_19_data.csv

```
[31]: data = pd.read_csv("datasets/Novel Corona Virus 2019 Dataset/covid_19_data.csv")
      data=data[data["Country/Region"]=="US"]
      display(data)




      # printing nan
      print(data.isna().sum())
```

```
           SNo ObservationDate Province/State Country/Region  \
31          32      01/22/2020     Washington             US
70          71      01/23/2020     Washington             US
119        120      01/24/2020     Washington             US
120        121      01/24/2020        Chicago             US
161        162      01/25/2020     Washington             US
…         …             …             …              …
285273  285274      05/02/2021       Virginia             US
285283  285284      05/02/2021     Washington             US
285286  285287      05/02/2021  West Virginia             US
285288  285289      05/02/2021      Wisconsin             US
285289  285290      05/02/2021        Wyoming             US

                Last Update  Confirmed   Deaths  Recovered
31          1/22/2020 17:00        1.0      0.0        0.0
70            1/23/20 17:00        1.0      0.0        0.0
119           1/24/20 17:00        1.0      0.0        0.0
120           1/24/20 17:00        1.0      0.0        0.0
161           1/25/20 17:00        1.0      0.0        0.0
…                       …         …        …          …
285273  2021-05-03 04:20:39   661314.0  10791.0        0.0
```

```
285283  2021-05-03 04:20:39   404709.0   5499.0         0.0
285286  2021-05-03 04:20:39   153918.0   2686.0         0.0
285288  2021-05-03 04:20:39   661685.0   7567.0         0.0
285289  2021-05-03 04:20:39    58142.0    707.0         0.0

[25174 rows x 8 columns]
```

```
SNo                0
ObservationDate    0
Province/State     0
Country/Region     0
Last Update        0
Confirmed          0
Deaths             0
Recovered          0
dtype: int64
```

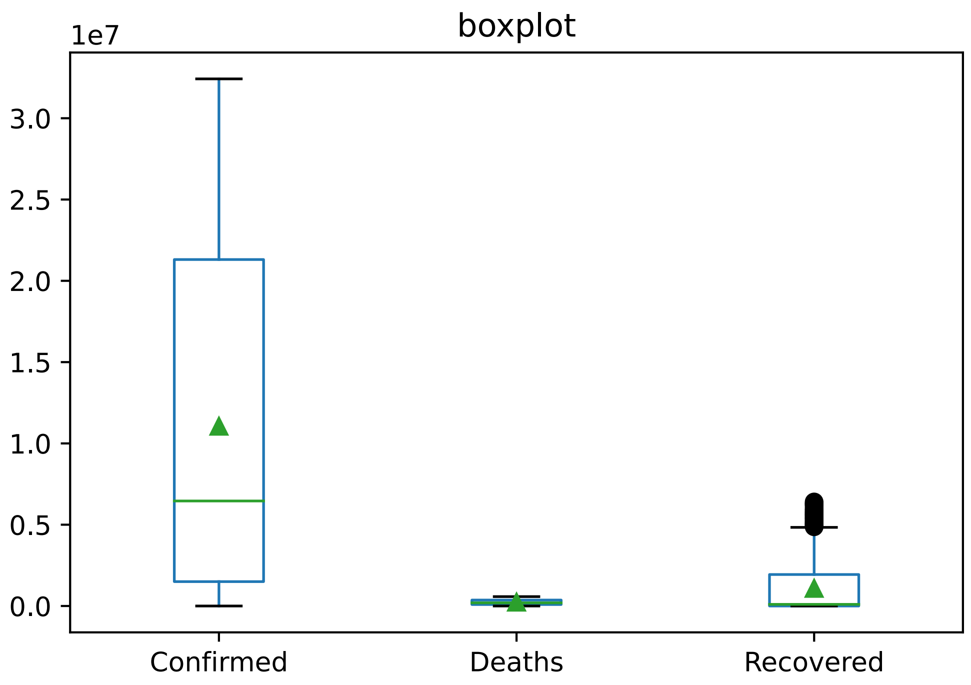
combine the values of all the states for every day

[32]:
```python
from datetime import datetime

dates=list(data['ObservationDate'].unique())
dates.sort(key=lambda date: datetime.strptime(date, "%m/%d/%Y"))

cleaned = pd.DataFrame({'ObservationDate':[],'Confirmed':[],'Deaths':
↪[],'Recovered':[]})
for d in dates:
    temp=data[data["ObservationDate"]==d]
    d=datetime.strptime(d, "%m/%d/%Y")
    d=d.strftime('%Y-%m-%d')
    cleaned.loc[len(cleaned.index)] = [d, temp['Confirmed'].
↪sum(),temp['Deaths'].sum(),temp['Recovered'].sum()]
```

3.1 Boxplots to detect outliers.

[33]:
```python
plt.figure(figsize=(10, 10))
ax = cleaned[['Confirmed', 'Deaths', 'Recovered']].plot(kind='box',␣
↪title='boxplot', showmeans=True)
plt.show()
```

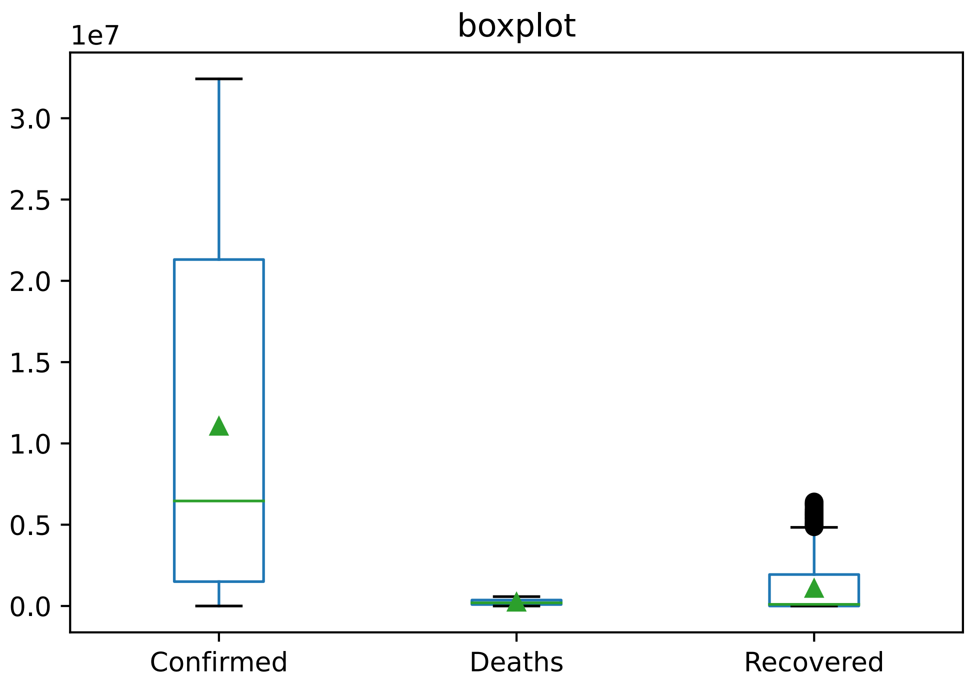
```
<Figure size 720x720 with 0 Axes>
```



3.2 Boxplot graph without outliers

[35]:
```
plt.figure(figsize=(10, 10))
ax = cleaned[['Confirmed', 'Deaths', 'Recovered']].plot(kind='box',␣
 ↪title='boxplot', showmeans=True)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



[36]:
```
import os
if not os.path.exists('cleaned/Novel Corona Virus 2019 Dataset'):
    os.makedirs('cleaned/Novel Corona Virus 2019 Dataset')
f=open(f'cleaned/Novel Corona Virus 2019 Dataset/covid_19_data.csv', "w")
cleaned.to_csv(path_or_buf=f,index=False,line_terminator='\n')
f.close()
```

3.3 group by month (discretize data)

[37]:
```
cleaned['ObservationDate'] = pd.to_datetime(cleaned.ObservationDate)

groupedByMonth=cleaned.groupby(pd.Grouper(key='ObservationDate', freq='1M')).
 ↪max().reset_index()
```

[38]:
```
import os
if not os.path.exists('cleaned/Novel Corona Virus 2019 Dataset'):
    os.makedirs('cleaned/Novel Corona Virus 2019 Dataset')
f=open(f'cleaned/Novel Corona Virus 2019 Dataset/covid_19_data_grouped_by_month.
 ↪csv', "w")
groupedByMonth.to_csv(path_or_buf=f,index=False,line_terminator='\n')
f.close()
```

3.4 normalize

[39]:
```
cleaned["Confirmed"] = (preprocessing.normalize([cleaned["Confirmed"]]))[0]
print(cleaned["Confirmed"])
cleaned["Deaths"] = preprocessing.normalize([cleaned["Deaths"]])[0]
print(cleaned["Deaths"])
cleaned["Recovered"] = preprocessing.normalize([cleaned["Recovered"]])[0]
print(cleaned["Recovered"])
```

```
0      2.940369e-09
1      2.940369e-09
2      5.880739e-09
3      5.880739e-09
4      1.470185e-08
           …
462    9.477060e-02
463    9.494173e-02
464    9.511204e-02
465    9.524525e-02
466    9.533128e-02
Name: Confirmed, Length: 467, dtype: float64
0      0.000000
1      0.000000
2      0.000000
3      0.000000
4      0.000000
         …
462    0.090126
463    0.090260
464    0.090423
465    0.090500
466    0.090551
Name: Deaths, Length: 467, dtype: float64
0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
      …
462    0.0
463    0.0
464    0.0
465    0.0
466    0.0
Name: Recovered, Length: 467, dtype: float64
```

```
[40]: import os
      if not os.path.exists('cleaned/Novel Corona Virus 2019 Dataset'):
          os.makedirs('cleaned/Novel Corona Virus 2019 Dataset')
      f=open(f'cleaned/Novel Corona Virus 2019 Dataset/covid_19_data_normalized.csv',␣
       ↪"w")
      cleaned.to_csv(path_or_buf=f,index=False,line_terminator='\n')
      f.close()
```

4 Countries population by year 2020

```
[41]: data = pd.read_csv("datasets/Countries population by year 2020/
       ↪population_by_country_2020.csv")
      data=data[data["Country (or dependency)"]=="United States"]
      data=data[["Country (or dependency)","Population (2020)","Density (P/Km²)","Med.
       ↪ Age"]]
      data.head()
```

```
[41]:   Country (or dependency)  Population (2020)  Density (P/Km²) Med. Age
      2           United States          330610570               36       38
```

```
[42]: import os
      if not os.path.exists('cleaned/Countries population by year 2020'):
          os.makedirs('cleaned/Countries population by year 2020')
      f=open(f'cleaned/Countries population by year 2020/population_by_country_2020.
       ↪csv', "w")
      data.to_csv(path_or_buf=f,index=False,line_terminator='\n')
      f.close()
```