DM Lab Assignment
PART - C

Report Submitted By:

Pranav Hegde - 1MS18IS412
Kumar Hegde - 1MS18IS406
Bhaskar .C - 1MS18IS402

Table of Contents:

Abstract

Coronavirus is a family of viruses that are named after their spiky crown. The novel coronavirus, also known as SARS-CoV-2, is a contagious respiratory virus that first reported in Wuhan, China. On 2/11/2020, the World Health Organization designated the name COVID-19 for the disease caused by the novel coronavirus. This notebook aims at exploring COVID-19 through data analysis and projections.

Introduction

Novel Corona Virus 2019 Dataset

Dataset_Source:-https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset?select=COVID19_line_list_data.csv

2019 Novel Coronavirus (2019-nCoV) is a virus (more specifically, a coronavirus) identified as the cause of an outbreak of respiratory illness first detected in Wuhan, China. Early on, many of the patients in the outbreak in Wuhan, China reportedly had some link to a large seafood and animal market, suggesting animal-to-person spread.

However, a growing number of patients reportedly have not had exposure to animal markets, indicating person-to-person spread is occurring. At this time, it's unclear how easily or sustainably this virus is spreading between people CDC

This dataset has daily level information on the number of affected cases, deaths and recovery from 2019 novel coronavirus. Please note that this is a time series data and so the number of cases on any given day is the cumulative number.

Approach Used:

Regression analysis is a set of statistical methods used for the estimation of relationships between a dependent variable and one or more independent variables. It can be utilized to assess the strength of the relationship between variables and for modeling the future relationship between them.
Regression analysis includes several variations, such as linear, multiple linear, and nonlinear. The most common models are simple linear and multiple linear. Nonlinear regression analysis is commonly used for more complicated data sets in which the dependent and independent variables show a nonlinear relationship.

Formula to Find The Regression Model : $Y_i=f(X_i, \beta)+e_i$.

Data Analysis:

Initial Setup:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import pandas as pd
import random
import math
import time
#-------------------------------------
from sklearn.linear_model import LinearRegression, BayesianRidge
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error
#-------------------------------------------------------------------
import datetime
import operator
plt.style.use('fivethirtyeight')
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Data Description :

```
latest_data.head()
```

	FIPS	Admin2	Province_State	Country_Region	Last_Update	Lat	Long_	Confirmed	Deaths	Recovered	Active	Combined_Key	Incidence_Rate	Case-Fatality_Ratio
0	NaN	NaN	NaN	Afghanistan	2020-10-21 04:24:14	33.93911	67.709953	40369	1501	33790	5068.0	Afghanistan	103.669971	3.714349
1	NaN	NaN	NaN	Albania	2020-10-21 04:24:14	41.15330	20.168300	17651	458	10225	6968.0	Albania	613.350476	2.594754
2	NaN	NaN	NaN	Algeria	2020-10-21 04:24:14	28.03390	1.659600	54829	1873	38346	14610.0	Algeria	125.034654	3.416075
3	NaN	NaN	NaN	Andorra	2020-10-21 04:24:14	42.50630	1.521800	3623	62	2273	1288.0	Andorra	4689.057141	1.711289
4	NaN	NaN	NaN	Angola	2020-10-21 04:24:14	-11.20270	17.873900	8049	251	3037	4761.0	Angola	24.490155	3.118400

```
confirmed_df.head()
```

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	12/24/20	12/25/20	12/26/20	12/27/20	12/28/20	12/29/20	12/30/20	12/31/2
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	50655	50810	50886	51039	51280	51350	51405	5152
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	54827	55380	55755	56254	56572	57146	57727	5831
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	...	97007	97441	97857	98249	98631	98988	99311	9961
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	...	7699	7756	7806	7821	7875	7919	7983	804
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	...	17029	17099	17149	17240	17296	17371	17433	1755

5 rows × 351 columns

```
us_medical_data.head()
```

	Province_State	Country_Region	Last_Update	Lat	Long_	Confirmed	Deaths	Recovered	Active	FIPS	Incident_Rate	People_Tested	People_Hospitalized	Mortality_Rate	UID	ISO3
0	Alabama	US	2020-10-21 04:30:32	32.3182	-86.9023	175210	2805	74238.0	97485.0	1.0	3559.482255	1265575.0	NaN	1.607192	84000001	USA
1	Alaska	US	2020-10-21 04:30:32	61.3707	-152.4044	11391	67	6681.0	4643.0	2.0	1557.115420	546525.0	NaN	0.588184	84000002	USA
2	American Samoa	US	2020-10-21 04:30:32	-14.2710	-170.1320	0	0	NaN	0.0	60.0	0.000000	1616.0	NaN	NaN	16	ASM
3	Arizona	US	2020-10-21 04:30:32	33.7298	-111.4312	232937	5837	38705.0	188395.0	4.0	3200.248066	1647345.0	NaN	2.505828	84000004	USA
4	Arkansas	US	2020-10-21 04:30:32	34.9697	-92.3731	100441	1728	90283.0	8430.0	5.0	3328.281094	1231652.0	NaN	1.720413	84000005	USA

Data Preprocessing:

In this processing we will get all the dates of the out break of pandemic and daily Increases and averages.

```
confirmed = confirmed_df.loc[:, cols[4]:cols[-1]]
deaths = deaths_df.loc[:, cols[4]:cols[-1]]
recoveries = recoveries_df.loc[:, cols[4]:cols[-1]]
```

```
dates = confirmed.keys()
world_cases = []
total_deaths = []
mortality_rate = []
recovery_rate = []
total_recovered = []
total_active = []

for i in dates:
    confirmed_sum = confirmed[i].sum()
    death_sum = deaths[i].sum()
    recovered_sum = recoveries[i].sum()

    # confirmed, deaths, recovered, and active
    world_cases.append(confirmed_sum)
    total_deaths.append(death_sum)
    total_recovered.append(recovered_sum)
    total_active.append(confirmed_sum-death_sum-recovered_sum)

    # calculate rates
    mortality_rate.append(death_sum/confirmed_sum)
    recovery_rate.append(recovered_sum/confirmed_sum)
```

Increasing the Daily Cases Of Corona Virus Accoding to the Basis of Perticular regions.

```
def daily_increase(data):
    d = []
    for i in range(len(data)):
        if i == 0:
            d.append(data[0])
        else:
            d.append(data[i]-data[i-1])
    return d

def moving_average(data, window_size):
    moving_average = []
    for i in range(len(data)):
        if i + window_size < len(data):
            moving_average.append(np.mean(data[i:i+window_size]))
        else:
            moving_average.append(np.mean(data[i:len(data)]))
    return moving_average

# window size
window = 7

# confirmed cases
world_daily_increase = daily_increase(world_cases)
world_confirmed_avg= moving_average(world_cases, window)
world_daily_increase_avg = moving_average(world_daily_increase, window)

# deaths
world_daily_death = daily_increase(total_deaths)
world_death_avg = moving_average(total_deaths, window)
world_daily_death_avg = moving_average(world_daily_death, window)


# recoveries
world_daily_recovery = daily_increase(total_recovered)
world_recovery_avg = moving_average(total_recovered, window)
world_daily_recovery_avg = moving_average(world_daily_recovery, window)


# active
world_active_avg = moving_average(total_active, window)
```

```
days_since_1_22 = np.array([i for i in range(len(dates))]).reshape(-1, 1)
world_cases = np.array(world_cases).reshape(-1, 1)
total_deaths = np.array(total_deaths).reshape(-1, 1)
total_recovered = np.array(total_recovered).reshape(-1, 1)
```

Visualization:

The data is further processed for better visualization of datetime.

The different regressions are used for visualization they are listed below

```
start = '1/22/2020'
start_date = datetime.datetime.strptime(start, '%m/%d/%Y')
future_forcast_dates = []
for i in range(len(future_forcast)):
    future_forcast_dates.append((start_date + datetime.timedelta(days=i)).strftime('%m/%d/%Y'))
```

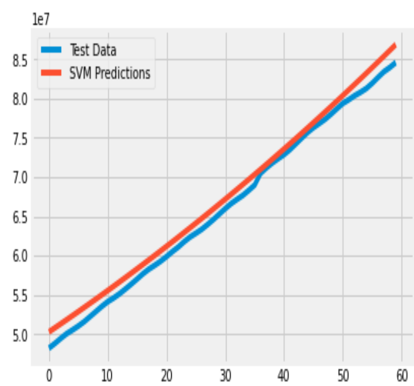
```
# slightly modify the data to fit the model better (regression models cannot pick the pattern)
X_train_confirmed, X_test_confirmed, y_train_confirmed, y_test_confirmed = train_test_split(days_since_1_22[50:], world_cases[50:], test_size=0.2, shuffle=False)
```

SVM Regressor

```
# svm_confirmed = svm_search.best_estimator_
svm_confirmed = SVR(shrinking=True, kernel='poly',gamma=0.01, epsilon=1,degree=3, C=0.1)
svm_confirmed.fit(X_train_confirmed, y_train_confirmed)
svm_pred = svm_confirmed.predict(future_forcast)
```

```
# check against testing data
svm_test_pred = svm_confirmed.predict(X_test_confirmed)
plt.plot(y_test_confirmed)
plt.plot(svm_test_pred)
plt.legend(['Test Data', 'SVM Predictions'])
print('MAE:', mean_absolute_error(svm_test_pred, y_test_confirmed))
print('MSE:',mean_squared_error(svm_test_pred, y_test_confirmed))
```

```
MAE: 1326548.419556894
MSE: 1965217939383.6392
```



Polynomial Regression

```
# transform our data for polynomial regression
poly = PolynomialFeatures(degree=4)
poly_X_train_confirmed = poly.fit_transform(X_train_confirmed)
poly_X_test_confirmed = poly.fit_transform(X_test_confirmed)
poly_future_forcast = poly.fit_transform(future_forcast)

bayesian_poly = PolynomialFeatures(degree=5)
bayesian_poly_X_train_confirmed = bayesian_poly.fit_transform(X_train_confirmed)
bayesian_poly_X_test_confirmed = bayesian_poly.fit_transform(X_test_confirmed)
bayesian_poly_future_forcast = bayesian_poly.fit_transform(future_forcast)
```

```
# polynomial regression
linear_model = LinearRegression(normalize=True, fit_intercept=False)
linear_model.fit(poly_X_train_confirmed, y_train_confirmed)
test_linear_pred = linear_model.predict(poly_X_test_confirmed)
linear_pred = linear_model.predict(poly_future_forcast)
print('MAE:', mean_absolute_error(test_linear_pred, y_test_confirmed))
print('MSE:',mean_squared_error(test_linear_pred, y_test_confirmed))
```

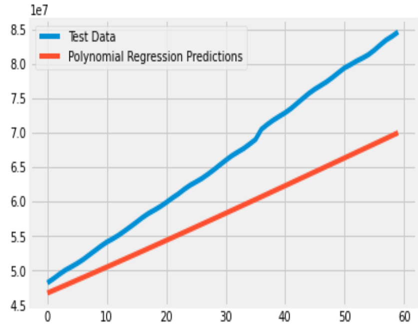
```
MAE: 7945015.295903881
MSE: 79185296204042.3
```

```
print(linear_model.coef_)
```

```
[[-1.38287451e+06  3.14448519e+04 -2.02859763e+02  3.88791621e+00
  -5.32255097e-03]]
```

```
plt.plot(y_test_confirmed)
plt.plot(test_linear_pred)
plt.legend(['Test Data', 'Polynomial Regression Predictions'])
```

```
<matplotlib.legend.Legend at 0x7fe6e68e3f10>
```



Bayesian ridge polynomial regression

```
# bayesian ridge polynomial regression
tol = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2]
alpha_1 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
alpha_2 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
lambda_1 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
lambda_2 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
normalize = [True, False]

bayesian_grid = {'tol': tol, 'alpha_1': alpha_1, 'alpha_2' : alpha_2, 'lambda_1': lambda_1, 'lambda_2' : lambda_2,
                 'normalize' : normalize}

bayesian = BayesianRidge(fit_intercept=False)
bayesian_search = RandomizedSearchCV(bayesian, bayesian_grid, scoring='neg_mean_squared_error', cv=3, return_train_score=True, n_jobs=-1, n_iter=40, verbose=1)
bayesian_search.fit(bayesian_poly_X_train_confirmed, y_train_confirmed)
```

```
Fitting 3 folds for each of 40 candidates, totalling 120 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    2.0s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:    2.1s finished
RandomizedSearchCV(cv=3, estimator=BayesianRidge(fit_intercept=False),
                   n_iter=40, n_jobs=-1,
                   param_distributions={'alpha_1': [1e-07, 1e-06, 1e-05, 0.0001,
                                                    0.001],
                                        'alpha_2': [1e-07, 1e-06, 1e-05, 0.0001,
                                                    0.001],
                                        'lambda_1': [1e-07, 1e-06, 1e-05,
                                                     0.0001, 0.001],
                                        'lambda_2': [1e-07, 1e-06, 1e-05,
                                                     0.0001, 0.001],
                                        'normalize': [True, False],
                                        'tol': [1e-06, 1e-05, 0.0001, 0.001,
                                                0.01]},
                   return_train_score=True, scoring='neg_mean_squared_error',
                   verbose=1)
```

```
bayesian_search.best_params_
```

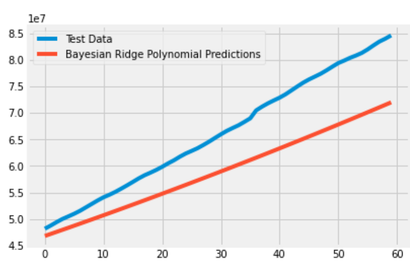
```
{'tol': 0.001,
 'normalize': False,
 'lambda_2': 1e-06,
 'lambda_1': 0.001,
 'alpha_2': 0.001,
 'alpha_1': 1e-07}
```

```
bayesian_confirmed = bayesian_search.best_estimator_
test_bayesian_pred = bayesian_confirmed.predict(bayesian_poly_X_test_confirmed)
bayesian_pred = bayesian_confirmed.predict(bayesian_poly_future_forcast)
print('MAE:', mean_absolute_error(test_bayesian_pred, y_test_confirmed))
print('MSE:',mean_squared_error(test_bayesian_pred, y_test_confirmed))
```

```
MAE: 7142471.68637624
MSE: 63025963254224.75
```

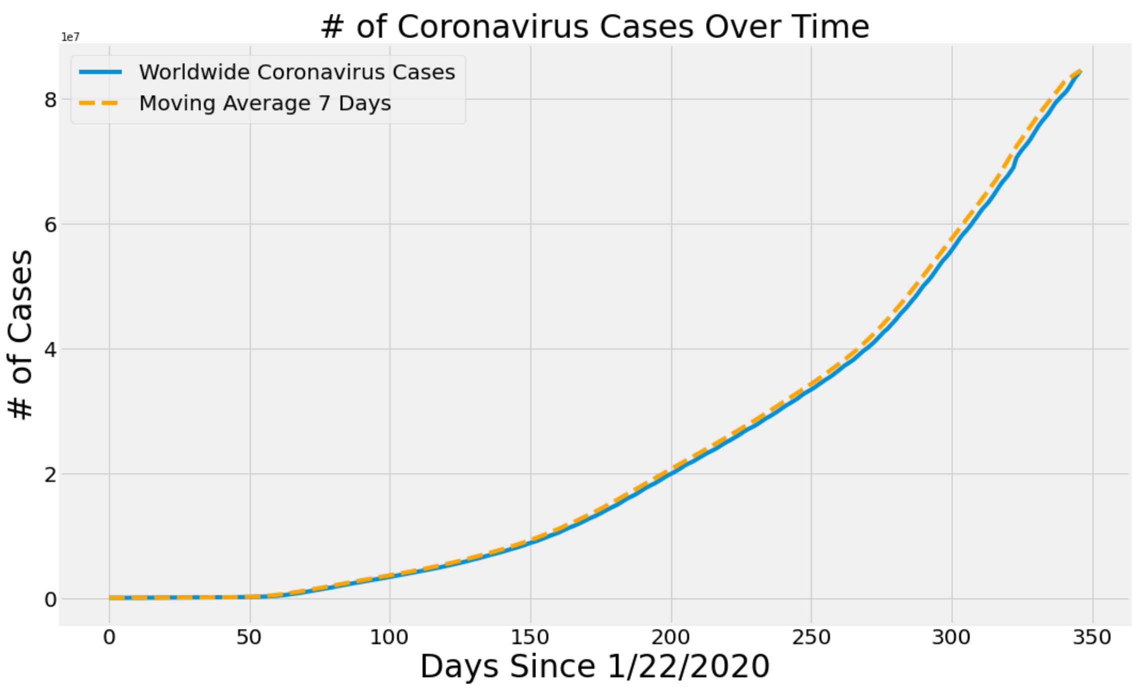
```
plt.plot(y_test_confirmed)
plt.plot(test_bayesian_pred)
plt.legend(['Test Data', 'Bayesian Ridge Polynomial Predictions'])
```

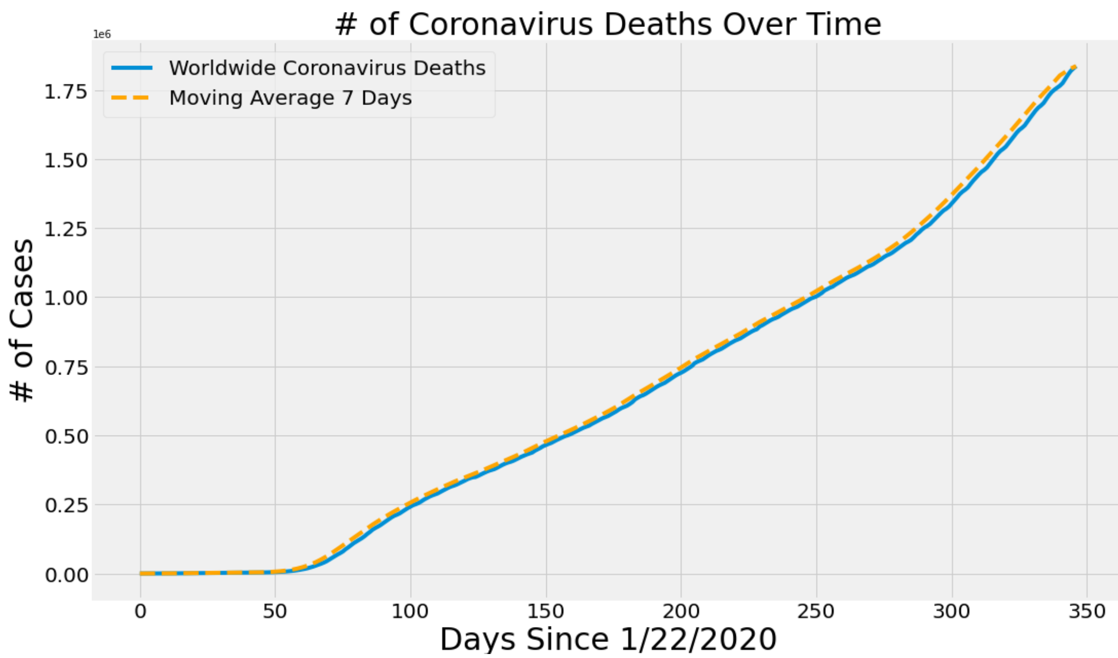
```
<matplotlib.legend.Legend at 0x7fe6e6b11ad0>
```



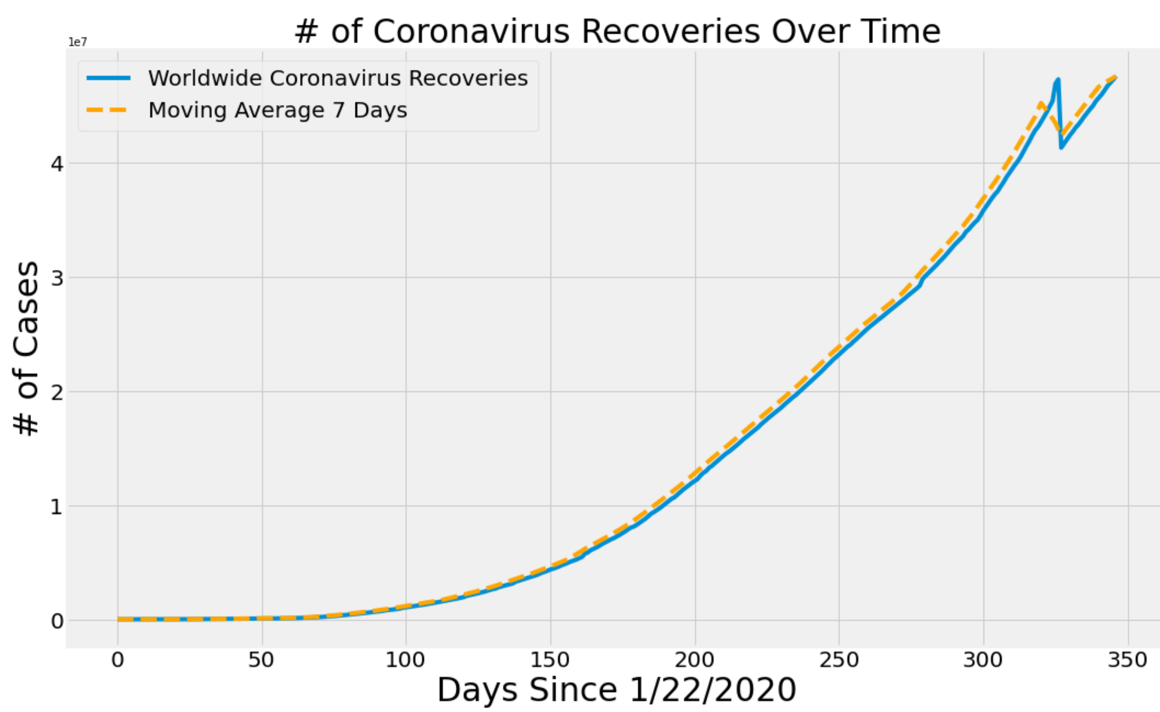
Here be the Example Of the Confirmed cases And Graphing the number of confirmed cases, active cases, deaths, recoveries, mortality rate and recovery rate.

```python
adjusted_dates = adjusted_dates.reshape(1, -1)[0]
plt.figure(figsize=(16, 10))
plt.plot(adjusted_dates, world_cases)
plt.plot(adjusted_dates, world_confirmed_avg, linestyle='dashed', color='orange')
plt.title('# of Coronavirus Cases Over Time', size=30)
plt.xlabel('Days Since 1/22/2020', size=30)
plt.ylabel('# of Cases', size=30)
plt.legend(['Worldwide Coronavirus Cases', 'Moving Average {} Days'.format(window)], prop={'size': 20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()

plt.figure(figsize=(16, 10))
plt.plot(adjusted_dates, total_deaths)
plt.plot(adjusted_dates, world_death_avg, linestyle='dashed', color='orange')
plt.title('# of Coronavirus Deaths Over Time', size=30)
plt.xlabel('Days Since 1/22/2020', size=30)
plt.ylabel('# of Cases', size=30)
plt.legend(['Worldwide Coronavirus Deaths', 'Moving Average {} Days'.format(window)], prop={'size': 20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()

plt.figure(figsize=(16, 10))
plt.plot(adjusted_dates, total_recovered)
plt.plot(adjusted_dates, world_recovery_avg, linestyle='dashed', color='orange')
plt.title('# of Coronavirus Recoveries Over Time', size=30)
plt.xlabel('Days Since 1/22/2020', size=30)
plt.ylabel('# of Cases', size=30)
plt.legend(['Worldwide Coronavirus Recoveries', 'Moving Average {} Days'.format(window)], prop={'size': 20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()

plt.figure(figsize=(16, 10))
plt.plot(adjusted_dates, total_active)
plt.plot(adjusted_dates, world_active_avg, linestyle='dashed', color='orange')
plt.title('# of Coronavirus Active Cases Over Time', size=30)
plt.xlabel('Days Since 1/22/2020', size=30)
plt.ylabel('# of Active Cases', size=30)
plt.legend(['Worldwide Coronavirus Active Cases', 'Moving Average {} Days'.format(window)], prop={'size': 20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.show()
```

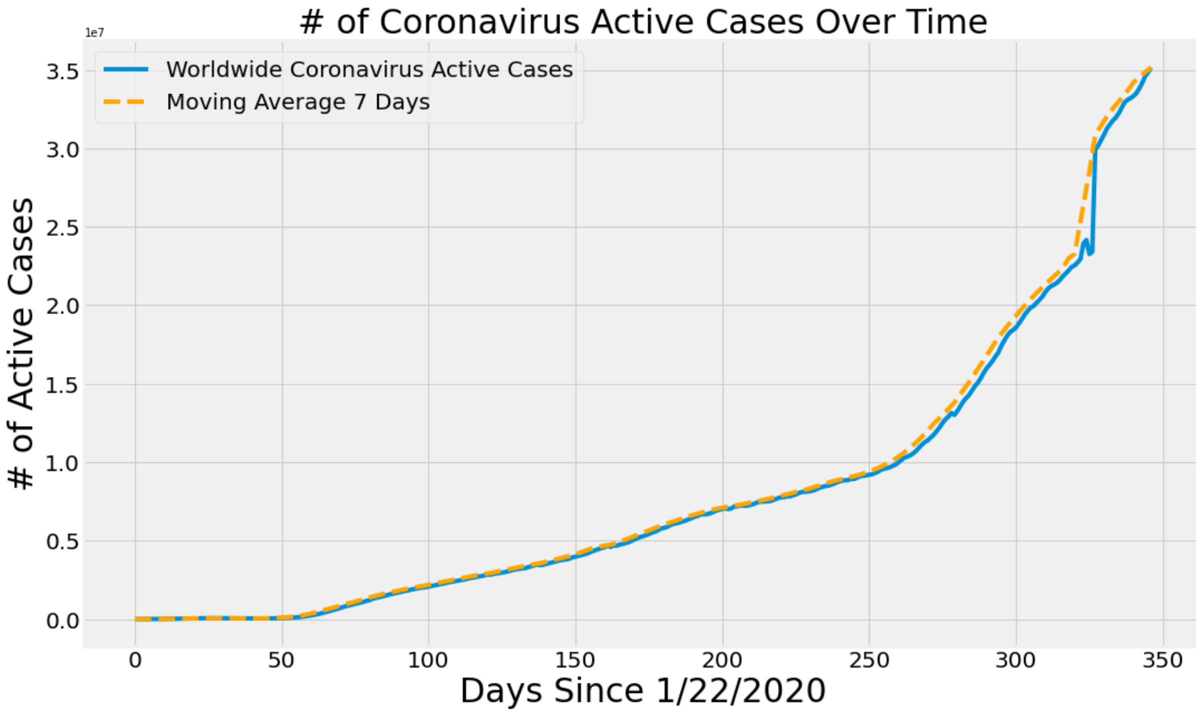




# of Coronavirus Deaths Over Time
1e6
Worldwide Coronavirus Deaths
Moving Average 7 Days
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
# of Cases
0
50
100
150
200
250
300
350
Days Since 1/22/2020



# of Coronavirus Recoveries Over Time
1e7
Worldwide Coronavirus Recoveries
Moving Average 7 Days
4
3
2
1
0
# of Cases
0
50
100
150
200
250
300
350
Days Since 1/22/2020



# of Coronavirus Active Cases Over Time
1e7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
# of Active Cases
Worldwide Coronavirus Active Cases
Moving Average 7 Days
0
50
100
150
200
250
300
350
Days Since 1/22/2020

Conclusion:

After referring the different techniques and using those on dataset conclude that we got the main features of the dataset those are confirmed cases, active cases, deaths, recoveries, mortality rate and recovery rate.
The features are graphed and made easy to understand the current situation

References:

- The data set was taken from the - https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset?select=COVID19_line_list_data.csv
- Also from - https://www.kaggle.com/bhushanaditya/covid-19-data-visualization