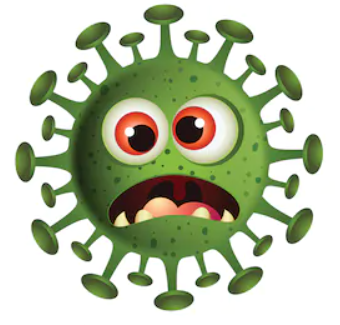
**United States COVID – 19 Dataset**

**Introduction**

Corona virus disease (COVID-19) is an infectious disease caused by a newly discovered corona virus. The virus mainly originated in China and spread across the world in the early 2020. Most people who are falling sick with the virus have experienced some simple symptoms and have recovered without any special medical treatment. But the elderly and people who have low immune system have been hit hardly by the virus. The COVID-19 virus is mainly transmitted through infected person when droplets of coughs & sneezes of the infected person pass from one person to another. These droplets are too heavy and quickly fall on floors or surfaces making it very easily transmittable.



As per the world wide COVID – 19’s latest report as on 16/12/2020 the total of 72.8 million cases are reported in which 41.3 million people have recovered successfully & 1.62 million people have lost their life. In the United States alone the total of 16,602,319 cases are reported in which around 301,762 people have lost their life during this pandemic.

**UnitedStates\_COVID\_19\_dataset – Problem Statement**

This is the data repository for the 2019 Novel Corona virus Visual Dashboard operated by the Johns Hopkins University Center for Systems Science and Engineering (JHU CSSE). Currently (on the date 20 May 2020), this leads to a total of 300K+ Deaths across the globe, including 90K+ deaths alone in USA. The dataset is provided to “**Identify the Deaths and Recovered Cases”**.

**Data Description**

The important columns in the Dataset are as below.

1. Province\_State - The name of the State within the USA.
2. Country\_Region - The name of the Country (US).
3. Last\_Update - The most recent date the file was pushed.
4. Lat - Latitude.
5. Long\_ - Longitude.
6. Confirmed - Aggregated confirmed case count for the state.
7. Deaths - Aggregated Death case count for the state.
8. Recovered - Aggregated Recovered case count for the state.
9. Active - Aggregated confirmed cases that have not been resolved (Active = Confirmed - Recovered - Deaths).
10. FIPS - Federal Information Processing Standards code that uniquely identifies counties within the USA.
11. Incident\_Rate - confirmed cases per 100,000 persons.
12. People\_Tested - Total number of people who have been tested.
13. People\_Hospitalized - Total number of people hospitalized.
14. Mortality\_Rate - Number recorded deaths \* 100/ Number confirmed cases.
15. UID - Unique Identifier for each row entry.
16. ISO3 - Officialy assigned country code identifiers.
17. Testing\_Rate - Total number of people tested per 100,000 persons.
18. Hospitalization\_Rate - Total number of people hospitalized \* 100/ Number of confirmed cases.

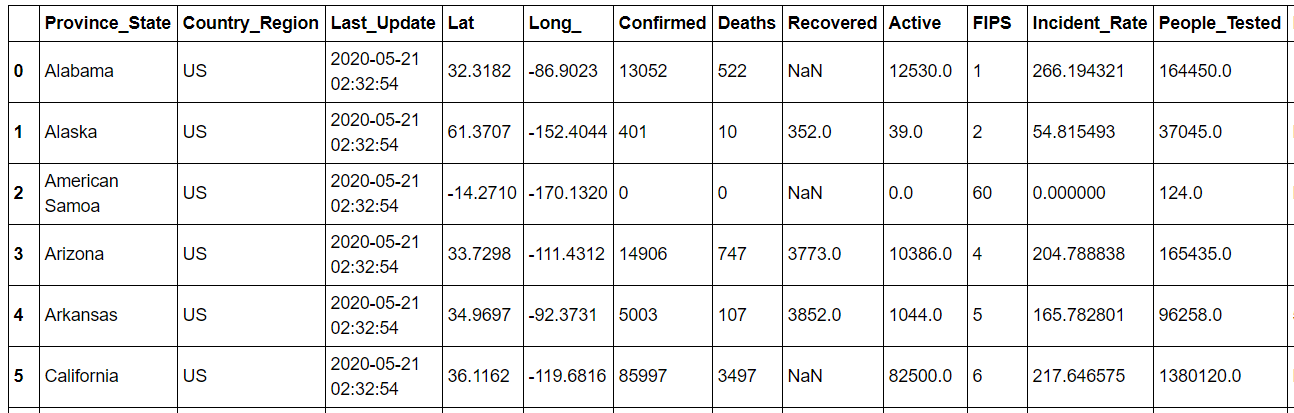
Now let’s load the dataset and do the analysis.

**Importing Libraries**

* **import** pandas as pd
* **import** numpy as np
* **import** seaborn as sns
* **import** matplotlib.pyplot as plt
* **import** warnings
* warnings.filterwarnings('ignore')

**Loading the dataset**

* ds=pd.read\_csv('csse\_covid\_19\_daily\_reports\_us.csv')
* ds



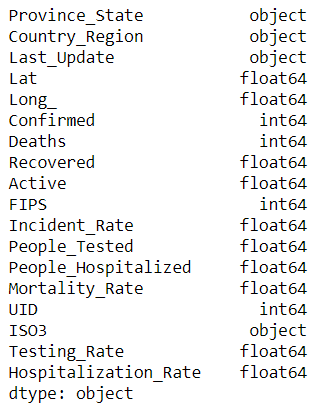
* #Checking the shape of dataset
* ds.shape

(58, 18)

The above dataset has 58 rows and 18 columns in the dataset.

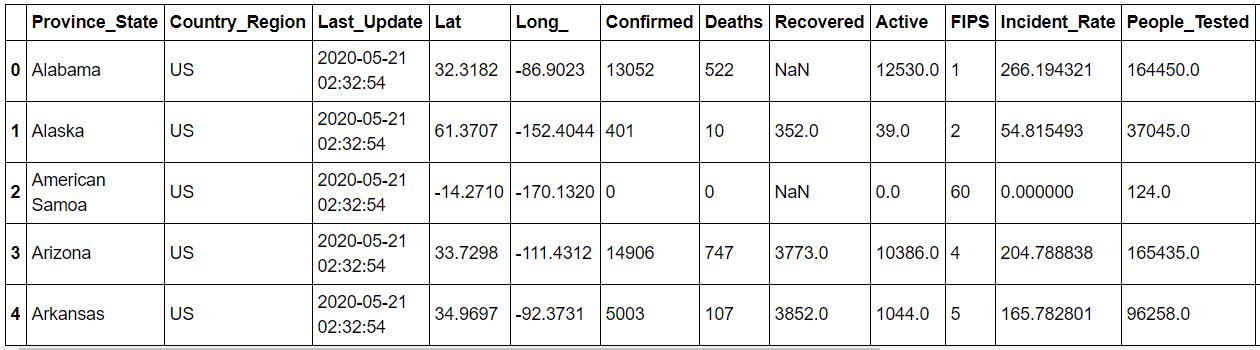
**Checking the data type of dataset.**

* #Checking the data types of dataset
* ds.dtypes

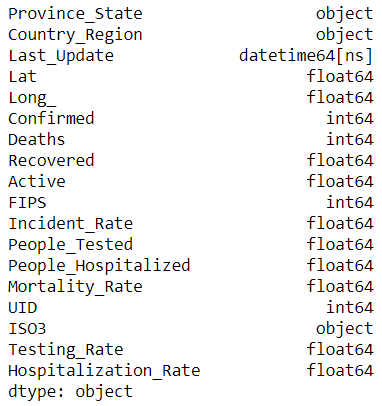


In the dataset “Last\_Update” column is in date type format but in the data type it is updated in “object” format. So converting the “Last\_Update” into “datetime64[ns]” format.

* #Changing the data types of Last\_Update column using pd.to\_datetime
* ds['Last\_Update'] = pd.to\_datetime(ds['Last\_Update'])
* ds.head()



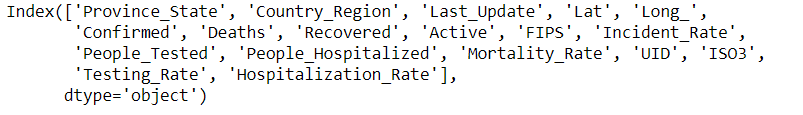
* #Checking the data types of dataset
* ds.dtypes



The “Last\_Update” column is now changed to datetime64[ns] format.

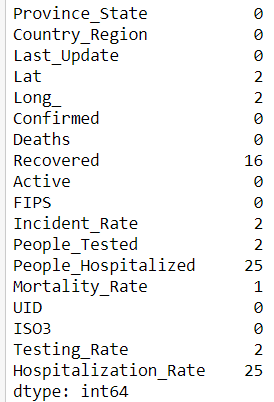
* #Checking the columns available in the dataset.

* ds.columns



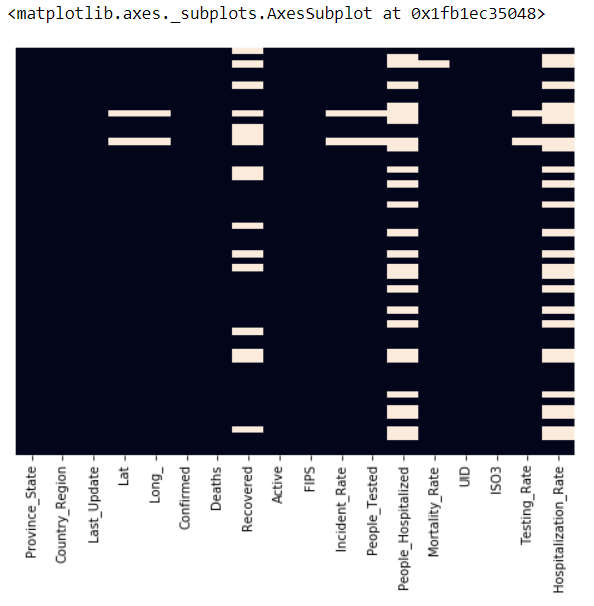
**Tracing Null values in data frame.**

* #Changing the null values in data frame.
* df=pd.DataFrame(ds)
* df.isnull().sum()

****

**Tracing Null values with the help of heat map.**

* #seeing missing value via visualization
* plt.figure(figsize=(8, 6))
* sns.heatmap(df.isnull(),yticklabels=False, cbar=False)

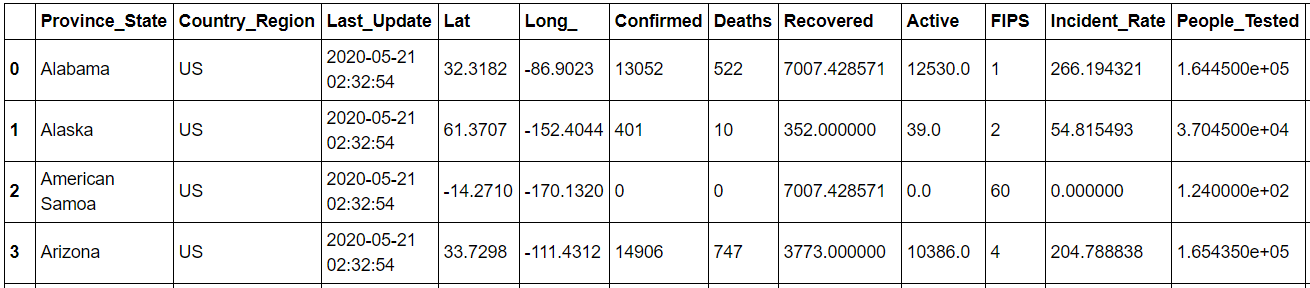


The “Lat, Long\_, People\_Tested, Incident\_Rate, Recovered, People\_Hospitalized, Hospitalization\_Rate, Mortality\_Rate, Testing\_Rate” columns have null values in the dataset.

**Replacing Null values with Simple Imputer**

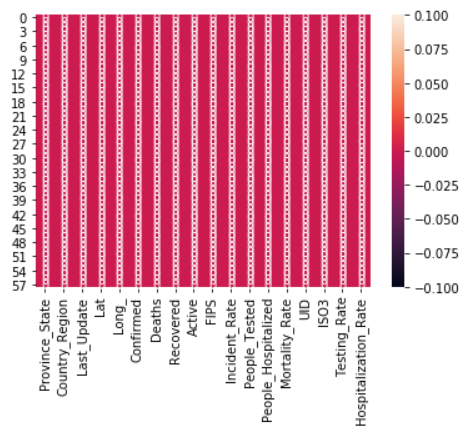
**Simple Imputer** – is a scikit-learn class which is used for handling the missing data in the dataset. It replaces the null values with a specified placeholder using strategy such as mean, media and most\_frequent.

* #imputer works for replacing the null data values
* **from** sklearn.impute **import** SimpleImputer
* imp=SimpleImputer(strategy="most\_frequent")
* df['Lat']=imp.fit\_transform(df['Lat'].values.reshape(-1, 1))
* df['Long\_']=imp.fit\_transform(df['Long\_'].values.reshape(-1, 1))
* #handling null data/value or imputing null
* imp=SimpleImputer(strategy="median")
* df['People\_Hospitalized']=imp.fit\_transform(df['People\_Hospitalized'].values.reshape(-1, 1))
* df['Hospitalization\_Rate']=imp.fit\_transform(df['Hospitalization\_Rate'].values.reshape(-1, 1))
* df['Incident\_Rate']=imp.fit\_transform(df['Incident\_Rate'].values.reshape(-1, 1))
* #handling null data/value or imputing null
* imp=SimpleImputer(strategy="mean")
* df['Recovered']=imp.fit\_transform(df['Recovered'].values.reshape(-1, 1))
* df['People\_Tested']=imp.fit\_transform(df['People\_Tested'].values.reshape(-1, 1))
* df['Mortality\_Rate']=imp.fit\_transform(df['Mortality\_Rate'].values.reshape(-1, 1))
* df['Testing\_Rate']=imp.fit\_transform(df['Testing\_Rate'].values.reshape(-1, 1))
* df



**Data set after implementing simple imputer.**

* #Lets check whether the null values in the dataset after implementing Simple Imputer Streategy
* sns.heatmap(df.isnull(), annot=True)
* plt.show()

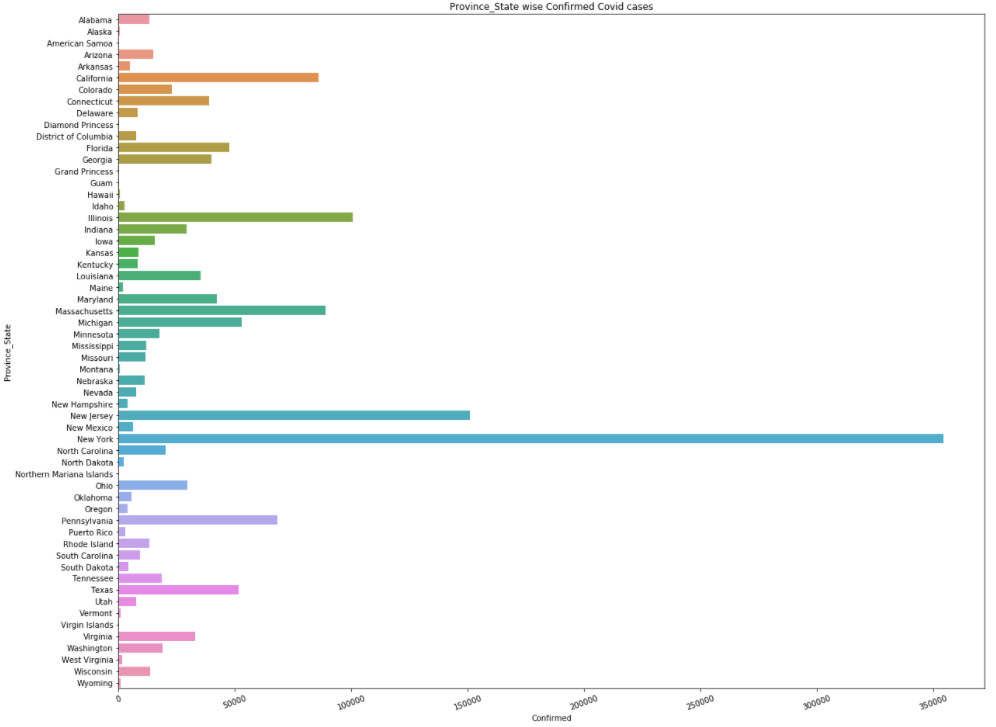
****

**Exploratory Data Analysis (EDA)**

Using EDA we can understand the relation between different columns.

**1.** **Province\_State wise Confirmed Covid cases**

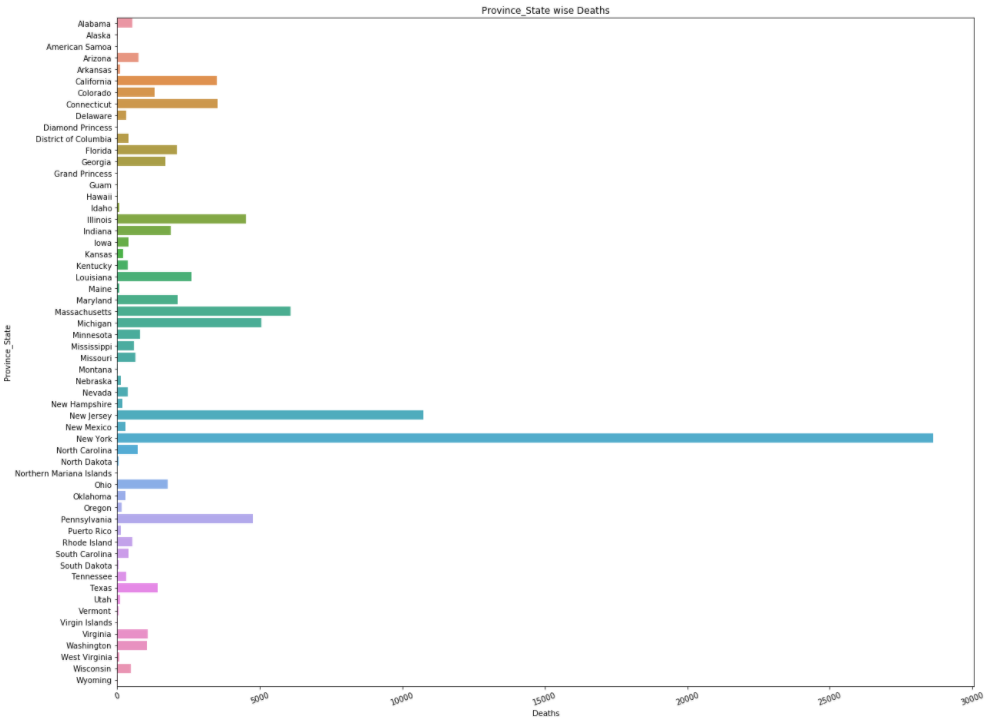
* #Province\_State wise Confirmed Covid cases
* plt.figure(figsize=(20, 16))
* sns.barplot(x='Confirmed', y='Province\_State', data=df)
* plt.title("Province\_State wise Confirmed Covid cases")
* plt.xticks(rotation=20)
* plt.show()



In the above graph we can see that “Ney York” has the highest number of Covid patients with 354,370 confirmed cases.

**2. Province\_State wise total number of Deaths**

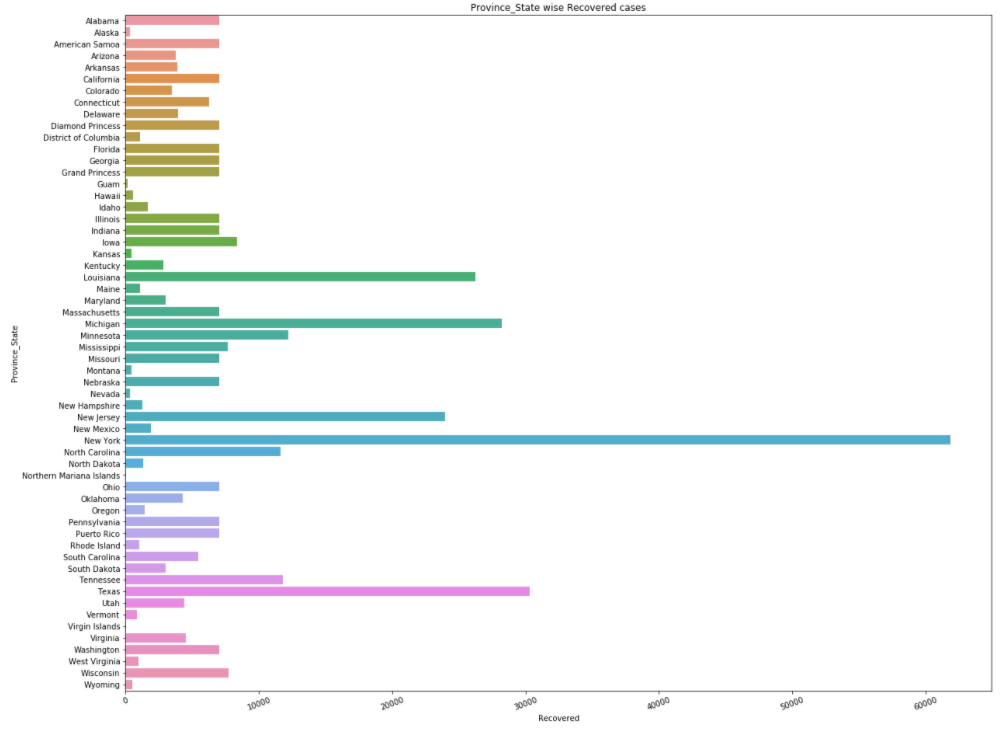
* #Province\_State wise Deaths
* plt.figure(figsize=(20, 16))
* sns.barplot(x='Deaths', y='Province\_State', data=df)
* plt.title("Province\_State wise Deaths")
* plt.xticks(rotation=20)
* plt.show()



In the above graph we can see that “Ney York” has the highest number of deaths with total of 28,636 diseased.

**3. Province\_State wise total number of Recovered cases**

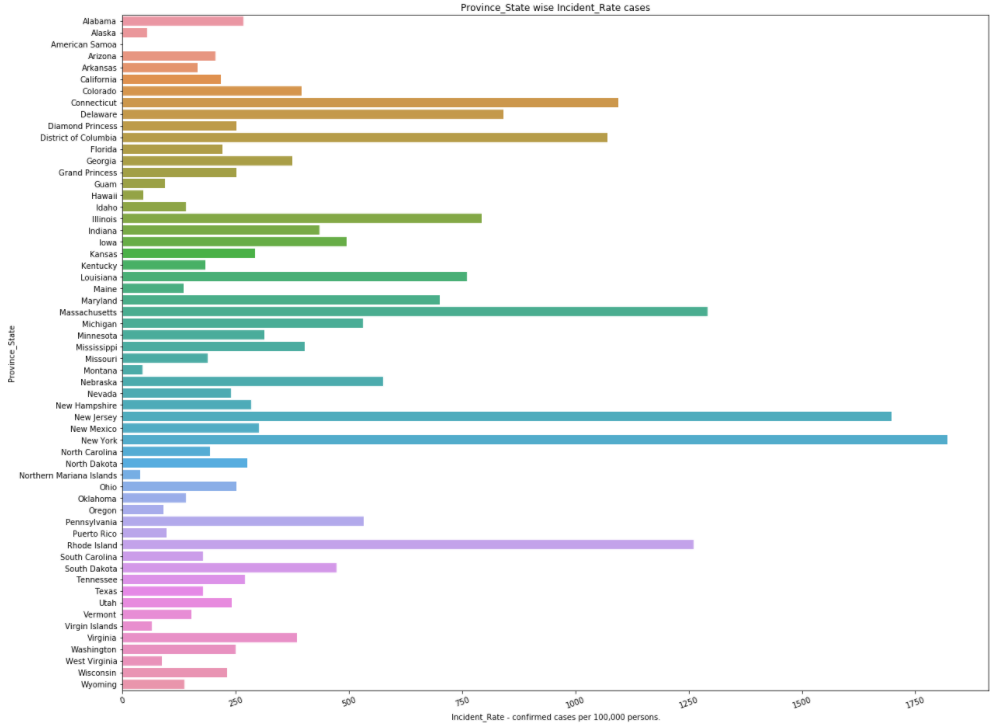
* #Province\_State wise Recovered
* plt.figure(figsize=(20, 16))
* sns.barplot(x='Recovered', y='Province\_State', data=df)
* plt.title("Province\_State wise Recovered cases")
* plt.xticks(rotation=20)
* plt.show()



In the above graph we can see that “Ney York” has the highest number of total recovered cases with total of 61,886.

**4. Province\_State wise total number of Incident\_Rate**

* #Province\_State wise Incident\_Rate
* plt.figure(figsize=(20, 16))
* sns.barplot(x='Incident\_Rate', y='Province\_State', data=df)
* plt.title("Province\_State wise Incident\_Rate cases")
* plt.xlabel("Incident\_Rate - confirmed cases per 100,000 persons.")
* plt.xticks(rotation=20)
* plt.show()

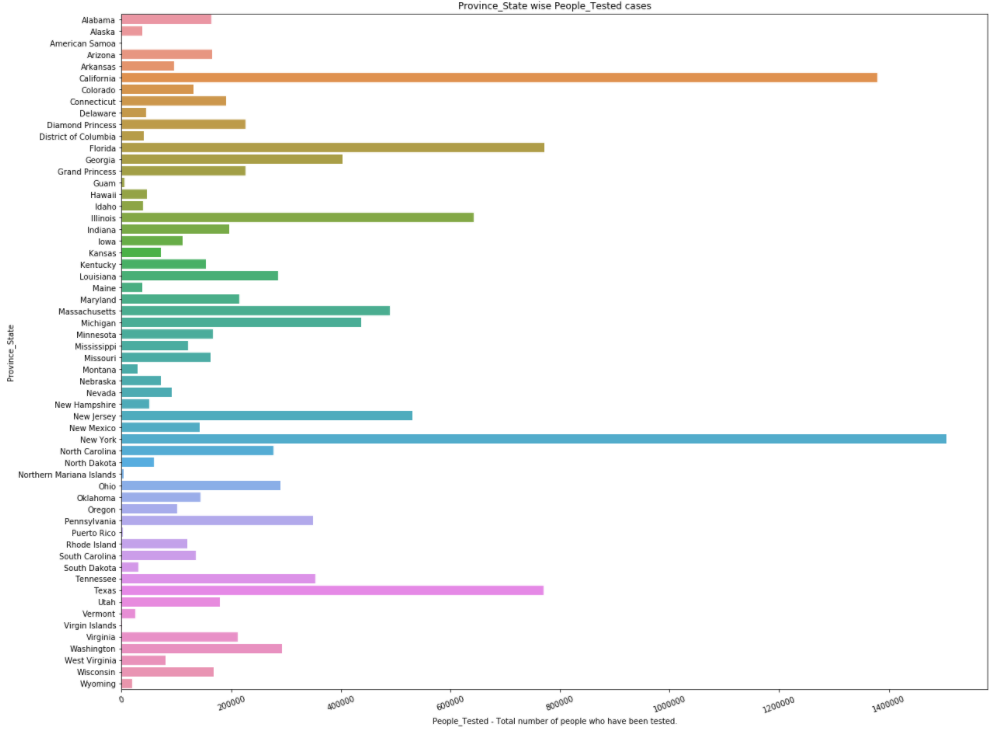


In the above graph we can see that “Ney York” has the highest number of Incident\_Rate

with total of 1821.62. Incident\_Rate stands for “confirmed cases per 100,000 persons”.

**5. Province\_State wise total number of People\_Tested cases.**

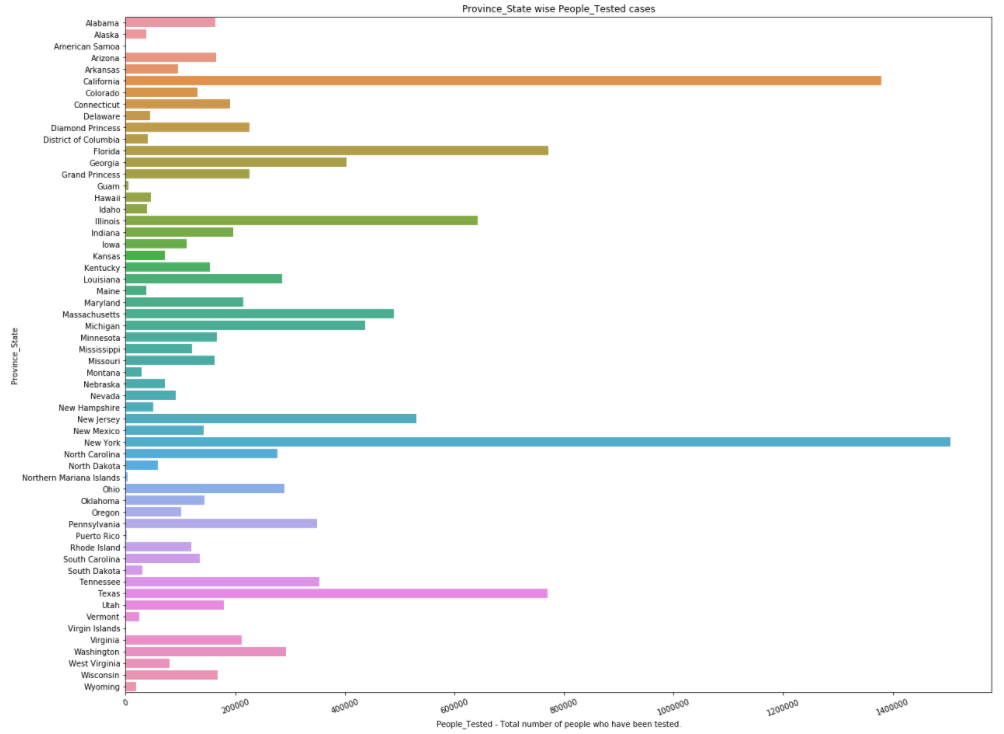
* #Province\_State wise People\_Tested cases
* plt.figure(figsize=(20, 16))
* sns.barplot(x='People\_Tested', y='Province\_State', data=df)
* plt.title("Province\_State wise People\_Tested cases")
* plt.xlabel("People\_Tested - Total number of people who have been tested.")
* plt.xticks(rotation=20)
* plt.show()



In the above graph we can see that “Ney York” has the highest number of People\_Tested cases with total of 1,505,836.

**6. Province\_State wise total number of People\_Hospitalized cases.**

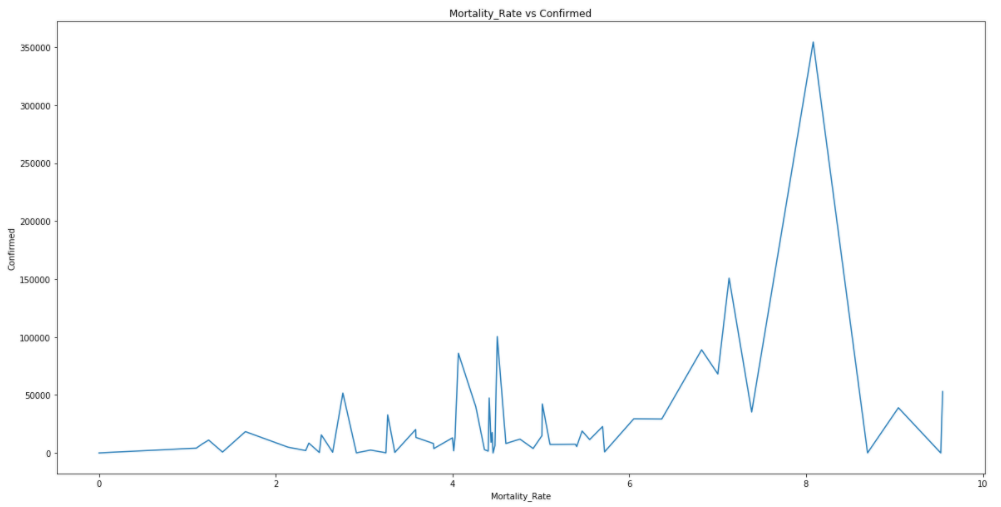
* #Province\_State wise People\_Hospitalized cases
* plt.figure(figsize=(20, 16))
* sns.barplot(x='People\_Hospitalized', y='Province\_State', data=df)
* plt.title("Province\_State wise People\_Hospitalized cases")
* plt.xlabel("People\_Hospitalized - Total number of people hospitalized.")
* plt.xticks(rotation=20)
* plt.show()



In the above graph we can see that “Ney York” has the highest number of People\_Hospitalizedcases with total of 76,410.

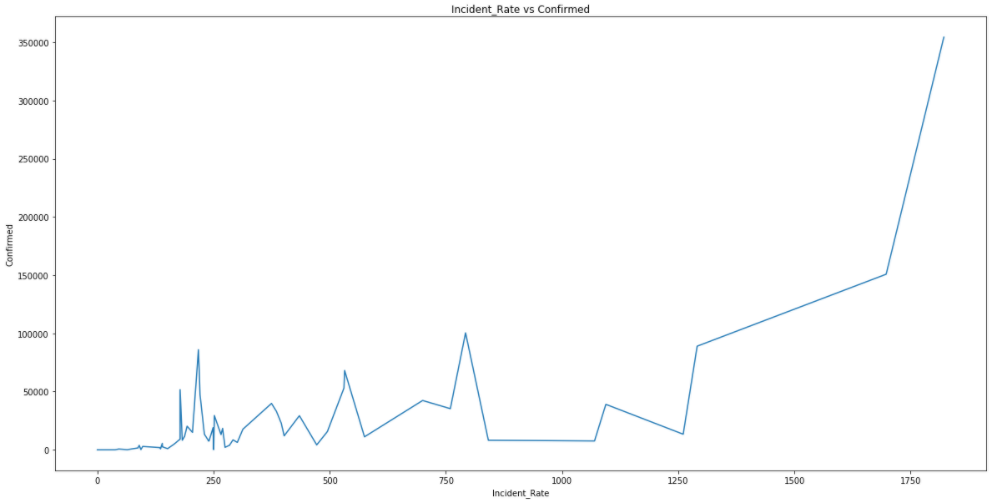
**7. Analysis of Mortality\_Rate with Confirmed cases using line plot.**

* #Analysis of Mortality\_Rate with Confirmed cases using line plot.
* plt.figure(figsize=(20, 10))
* sns.lineplot(x="Mortality\_Rate", y="Confirmed", data=df)
* #sns.countplot(x ='trestbps', hue = "age", data = df)
* plt.title("Mortality\_Rate vs Confirmed")
* plt.show()

****

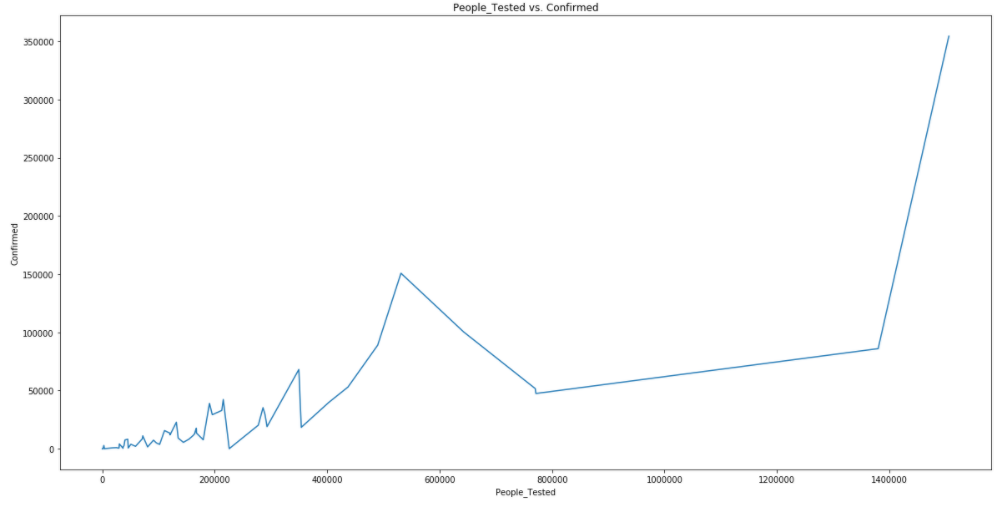
**8. Analysis of Incident\_Rate with Confirmed cases using line plot.**

* #Analysis of Incident\_Rate with Confirmed cases using line plot.
* plt.figure(figsize=(20, 10))
* sns.lineplot(x="Incident\_Rate", y="Confirmed", data=df)
* #sns.countplot(x ='trestbps', hue = "age", data = df)
* plt.title("Incident\_Rate vs Confirmed")
* plt.show()

****

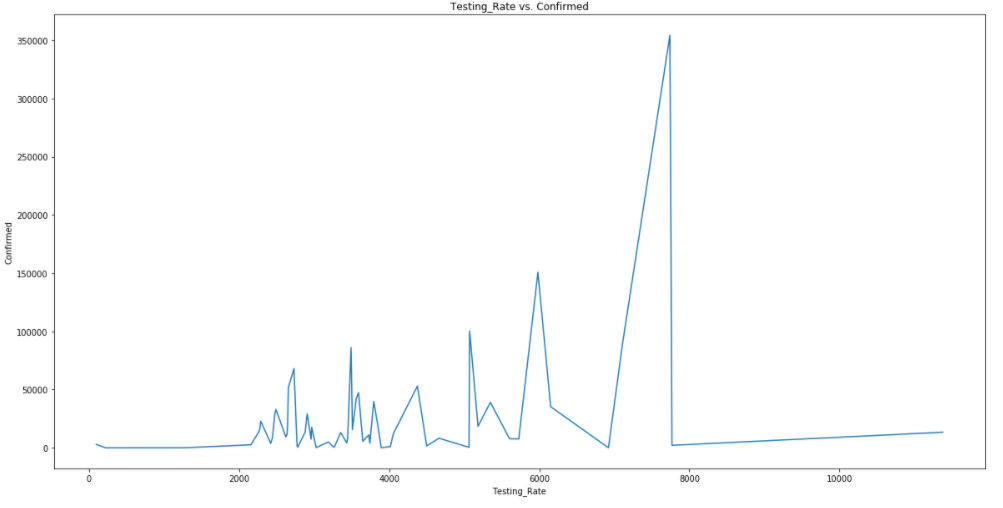
**8. Analysis of People\_Tested with Confirmed cases using line plot.**

* #Analysis of People\_Tested with Confirmed cases using line plot.
* plt.figure(figsize=(20, 10))
* sns.lineplot(x="People\_Tested", y="Confirmed", data=df)
* #sns.countplot(x ='trestbps', hue = "age", data = df)
* plt.title("People\_Tested vs. Confirmed")
* plt.show()



**9. #Analysis of Testing\_Rate with Confirmed cases using line plot.**

* #Analysis of Testing\_Rate with Confirmed cases using line plot.
* plt.figure(figsize=(20, 10))
* sns.lineplot(x="Testing\_Rate", y="Confirmed", data=df)
* #sns.countplot(x ='trestbps', hue = "age", data = df)
* plt.title("Testing\_Rate vs. Confirmed")
* plt.show()

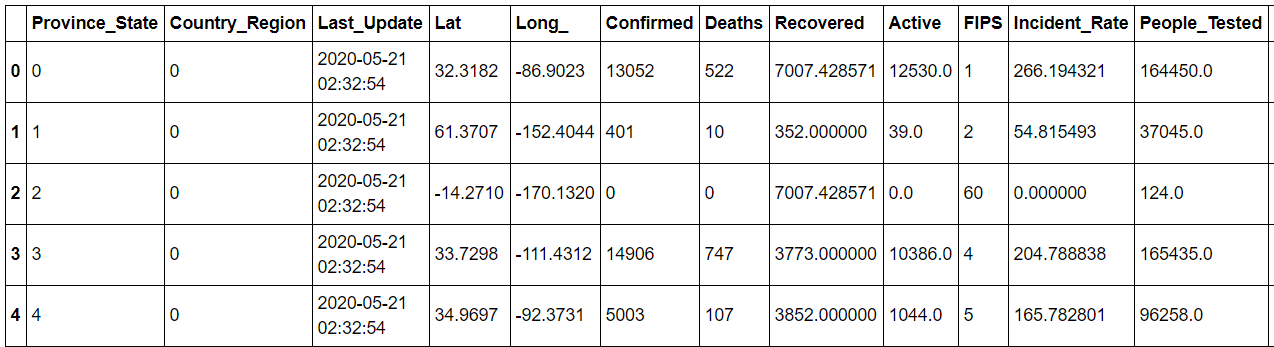


Now let’s replace the categorical values (alphabetic values) to numeric values using Label Encoder.

**Label Encoder** - refers to converts the labels into numeric form i.e the columns which are in alphabetical or categorical values are assigned with numbers so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated during the process.

Import label encoder for converting categorical data (object) into numeric(int 64) data.

* # Import label encoder
* # Now lets replace the categorical values (alphabetic values) to numeric values using Label Encoder
* **from** sklearn **import** preprocessing
* # label\_encoder object knows how to understand word labels.
* label\_encoder = preprocessing.LabelEncoder()
* # Lets Encode labels in columns
* df['Province\_State']= label\_encoder.fit\_transform(df['Province\_State'])
* df['Country\_Region']= label\_encoder.fit\_transform(df['Country\_Region'])
* df['ISO3']= label\_encoder.fit\_transform(df['ISO3'])
* df.head()

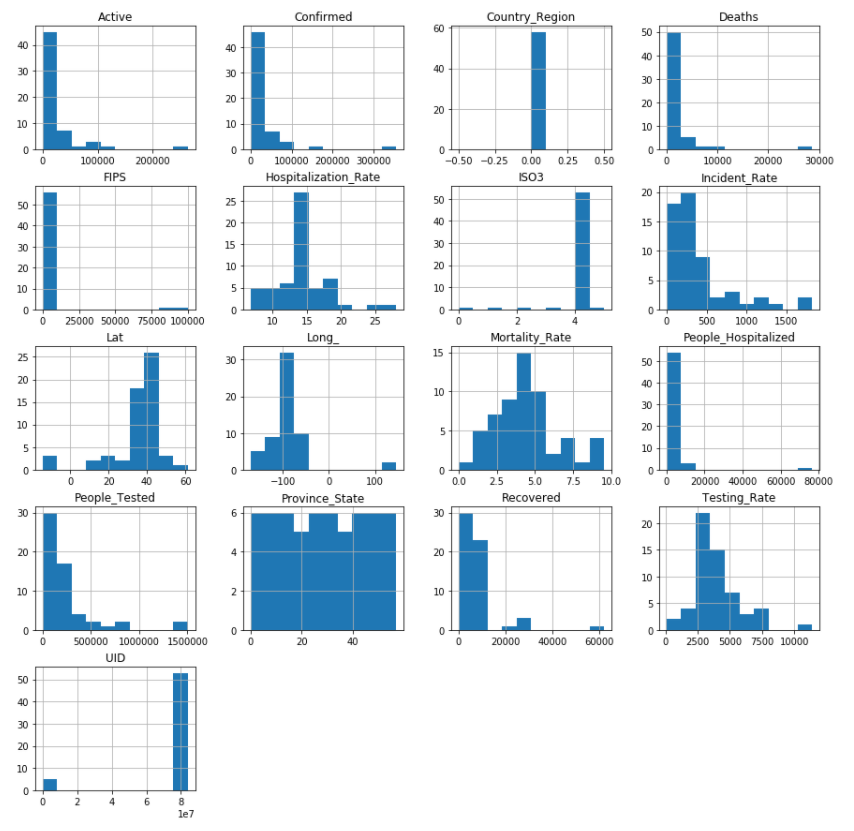


The object type columns “Province\_State, Country\_Region and ISO3” are converted into numeric type columns.

**Plotting Histogram**

A histogram is used to represent data provided in a form of certain groups using various columns of the dataset. It is accurate method for the graphical representation of numerical data distribution in the given dataset.

* df.hist(figsize=(15,15))



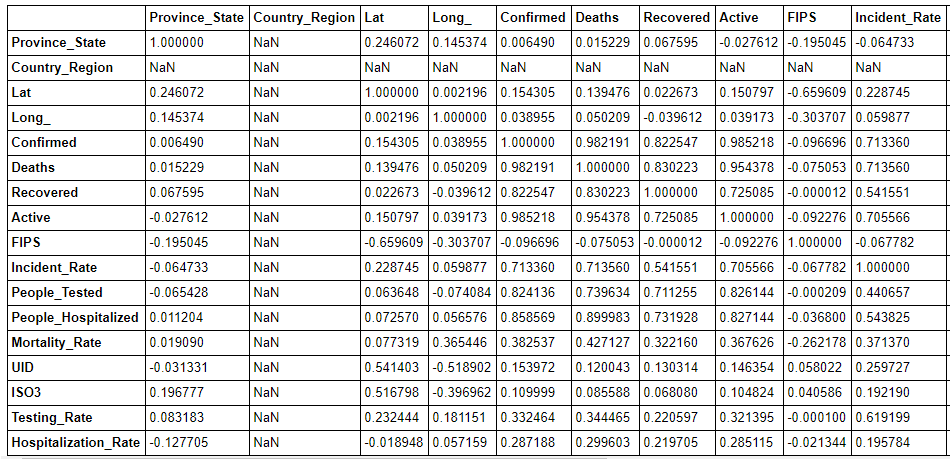
The above graph shows the graphical representation of the different columns in the form of histogram.

**Correlation Factor**

The statistical relationship between two variables is referred to as their correlation. The correlation factor represents the relation between columns in a given dataset. A correlation can be positive, meaning both variables are moving in the same direction or it can be negative, meaning that when one variable's value increasing, the other variable’s value is decreasing.

Now let’s check the correlation factor of the df dataset

* #The Correlation factor
* dfcor=df.corr()
* dfcor

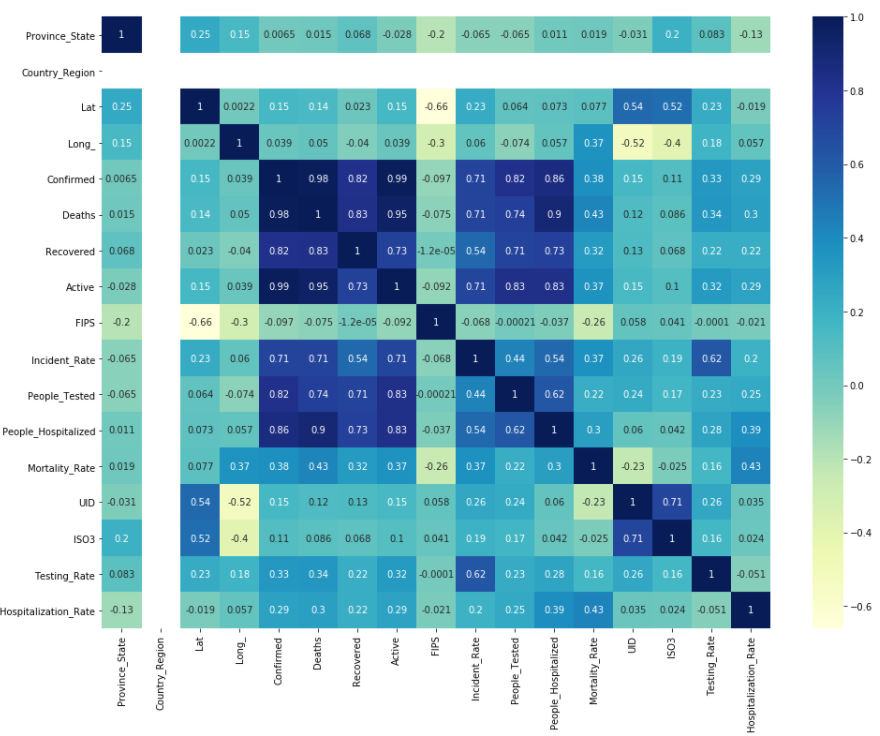


**Correlation Matrix**

A correlation matrix is a tabular data representing the ‘correlations’ between pairs of variables in a given dataset. It is also a very important pre-processing step in Machine Learning pipelines. The Correlation matrix is a data analysis representation that is used to summarize data to understand the relationship between various different variables of the given dataset.

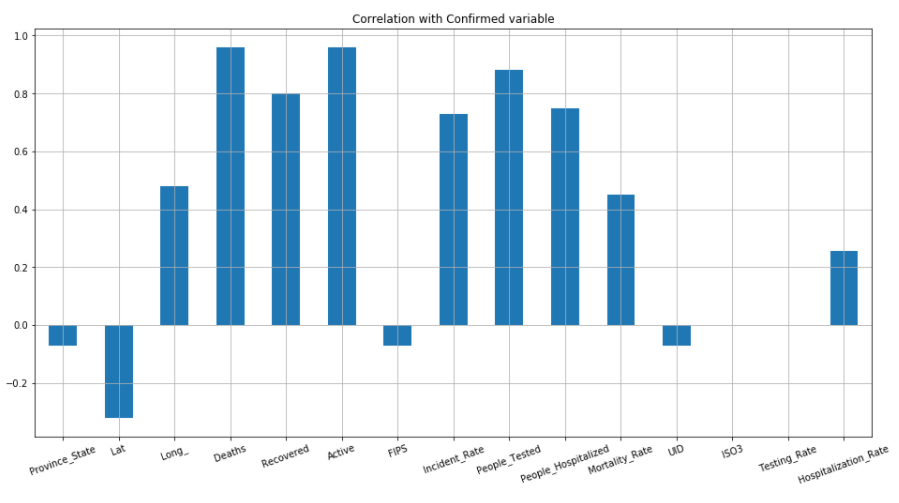
**Correlation factor with visualization / Correlation matrix**

* #Now lets check the Correlation factor with visualization
* plt.figure(figsize=(16, 12))
* sns.heatmap(dfcor, cmap='YlGnBu', annot=True)



**Checking Correlation with target (Confirmed) variable**

* #checking Correlation with Confirmed variable
* plt.figure(figsize=(16, 8))
* df.drop('Confirmed', axis=1).corrwith(df['Confirmed']).plot(kind='bar', grid=True)
* plt.xticks(rotation=20)
* plt.title("Correlation with Confirmed variable")

****

**Dropping ineffective columns from dataset.**

* #Dropping the 'Country\_Region' & 'Last\_Update' column from the datset
* df=df.drop(['Country\_Region'], axis=1)
* df=df.drop(['Last\_Update'], axis=1)

In the above dataset we can drop the 'Country\_Region' and 'Last\_Update' columns from the dataset. The 'Country\_Region' mentioned in the dataset is USA which is common for all the Provinces/States and also it is not adding any value inputs in the correlation matrix. Also we can drop the 'Last\_Update’ column as the date across all the columns remains same.

**Outliers**

An outlier is a data point in a data set which is distant or far from all other observations available. It is a data point which lies outside the overall distribution which is available in the dataset.

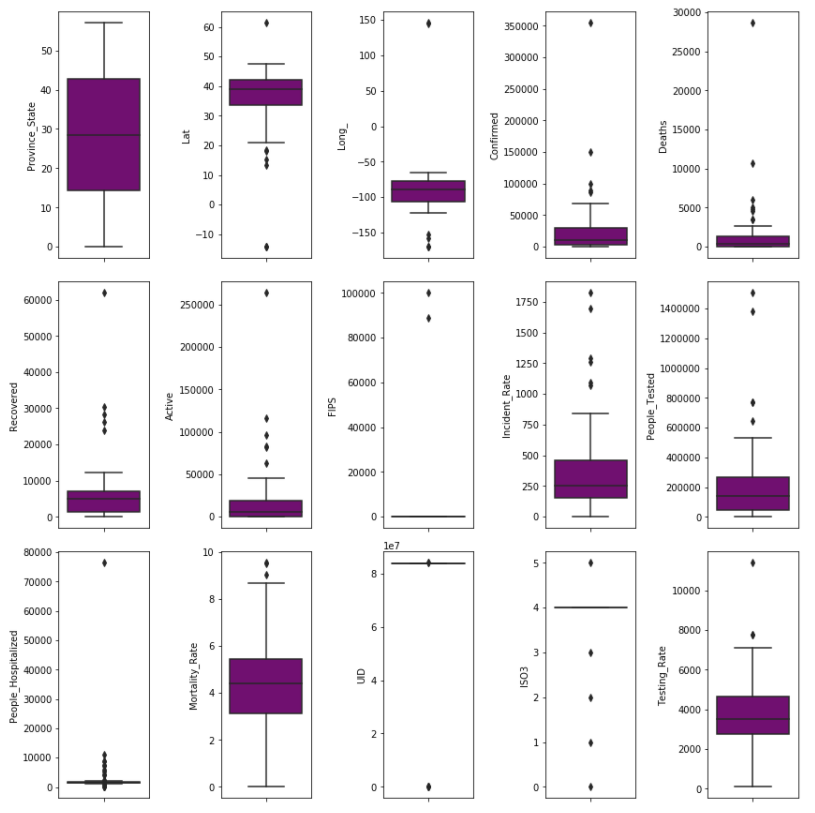
**Plotting outliers in the Dataset**

* #Plotting outliers in the df Dataset
* columns1=['Province\_State', 'Lat', 'Long\_',
* 'Confirmed', 'Deaths', 'Recovered', 'Active', 'FIPS', 'Incident\_Rate',
* 'People\_Tested', 'People\_Hospitalized', 'Mortality\_Rate', 'UID', 'ISO3',
* 'Testing\_Rate', 'Hospitalization\_Rate']

* ncol=5
* nrow=5


* plt.figure(figsize=(12, 20))
* **for** i **in** range(0, len(columns1)):
* plt.subplot(nrow, ncol, i+1)
* sns.boxplot(df[columns1[i]], color='purple', orient='v')
* plt.tight\_layout()

**Representation of outliers in Box plot**

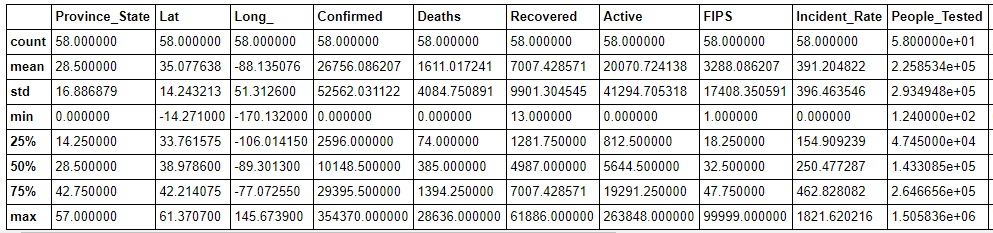
****

The above graph shows that outliers are present in “Lat, Long\_, Confirmed, Deaths, Recovered, Active, FIPS, Incident\_Rate, People\_Tested, People\_Hospitalized, Mortality\_Rate, UID, ISO3, Testing\_Rate” columns.

**Summary Statistics**

The describe() function computes a summary of statistics pertaining to the Data Frame columns. This function gives the mean, count, max, standard deviation and IQR values of the dataset in a simple understandable way.

* df.describe()

****

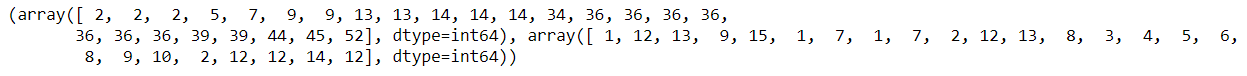
In the above columns we can see the value of Summary statistics.

1. The total count of all the columns in 58
2. In the Confirmed column maximum confirmed cases are 354,370 & a minimum case confirmed is 0.
3. In the Deaths column maximum deaths confirmed are 28,636 & minimum deaths confirmed are 0.
4. In the Recovered column maximum recovered cases are 61,886 & minimum recovered cases are 13.
5. In the Active column maximum active cases are 263,848 & minimum active case is 1.
6. In the Incident\_Rate column maximum incident rate recorded is 1821.62 & minimum incident rate recorded is 0.
7. In the People\_Hospitalized column maximum people hospitalized are 76,410 & minimum people hospitalized are 65.
8. In the Mortality\_Rate column maximum Mortality\_Rate recorded is 9.5455 & minimum Mortality\_Rate recorded is 0.
9. Describe table gives the min, max, std, count values of the different columns across the dataset & also it provides IQR values in 25%, 50% and 75% of the columns.

**Z – Score**

A Z-score is a numerical measurement that describes a value's relationship to the mean of a group of values in the dataset. Z-score is measured in terms of [standard deviations](https://www.investopedia.com/terms/s/standarddeviation.asp) from the mean.

* #Z score
* **from** scipy.stats **import** zscore
* z=np.abs(zscore(df))
* z
* threshold=3
* **print**(np.where(z>3))



* #Checking the shape of df dataset & df\_new datset
* df\_new=df[(z<3).all(axis=1)]
* **print**(df.shape, '\t', df\_new.shape)



* #shape of df dataset
* df=df\_new
* **print**(df.shape)



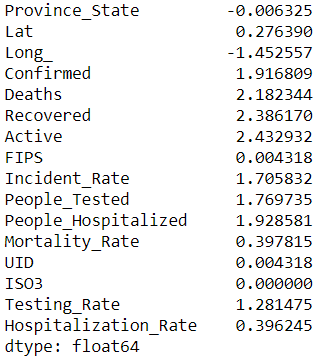
**To check distribution of Skewness**

**Skewness**

Skewness refers to distortion or asymmetry in a symmetrical bell curve, or [normal distribution](https://www.investopedia.com/terms/n/normaldistribution.asp) in a set of data. Besides positive and negative skew, distributions can also be said to have zero or undefined skew. The skewness value can be positive, zero, negative, or undefined.

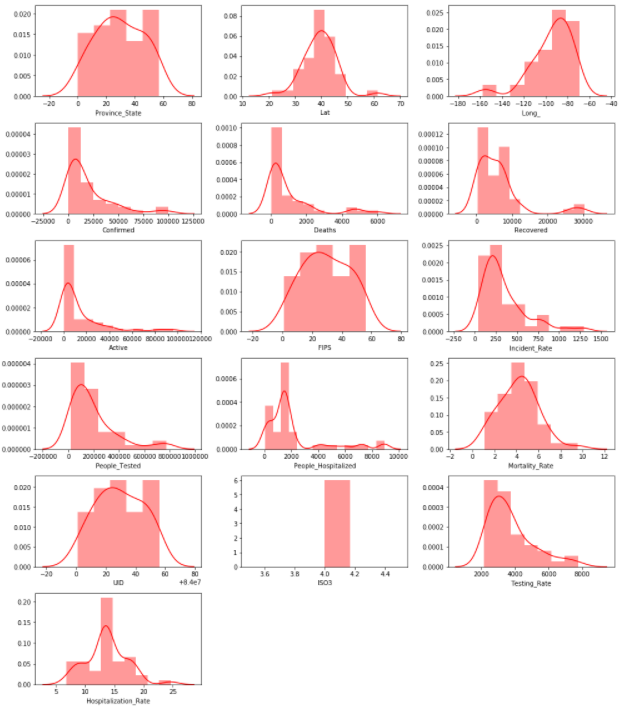
Check skewness in datset.

* df.skew()



* #To check skewness of dataset with visualization
* ncol=3
* nrow=7
* plt.figure(figsize=(14, 18))
* **for** i **in** range(0, len(columns1)):
* plt.subplot(nrow, ncol, i+1)
* sns.distplot(df[columns1[i]], color='red')
* plt.tight\_layout()

**Checking skewness using distplot**



As per the graphical representation of the dataset the columns “Confirmed, Deaths, Recovered, Active, Incident\_Rate, People\_Tested, People\_Hospitalized, Testing\_Rate” are Right skewed.

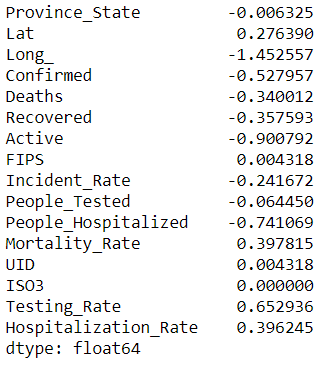
**Treating Skewness**

The log-transformed data follows a normal or near normal distribution while trating skewness. In this case, the log-transformation method removes or reduces skewness.

* #treating skewness via log method
* **for** col **in** df.columns:
* **if** df[col].skew()>0.55:
* df[col]=np.log1p(df[col])

  #After removing skewness

* df.skew()



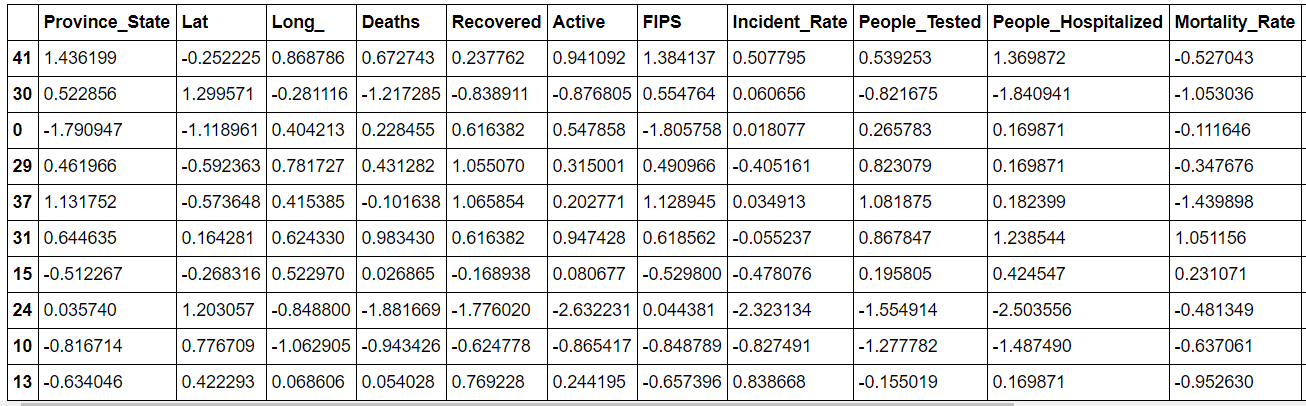
**Model Training**

Now separating the output variable “y” from the input variable df\_x

* #seprating into input and output variables
* df\_x=df.drop(columns=['Confirmed'])
* y=pd.DataFrame(df['Confirmed'])

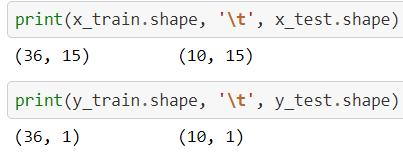
* #scaling the input variable
* **from** sklearn.preprocessing **import** StandardScaler
* sc=StandardScaler()
* x=sc.fit\_transform(df\_x)
* x=pd.DataFrame(x, columns=df\_x.columns)

* x.sample(10)



Splitting the data into training and testing data

* #breaking input and output into target variable
* **from** sklearn.model\_selection **import** train\_test\_split
* x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, random\_state=42)



**Using Regressor models for model building.**

Different type of Regreesor models used in model building.

**1. Linear Regression –** The linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The case of one explanatory variable is called simple linear regression; for more than one variable, the process is known as multiple linear regression.

2. **Lasso** - Lasso regression is a linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a mid point, like the mean. The lasso procedure encourages simple and sparse models. In machine learning model, lasso is a regression analysis method that performs variable selection and regularization in order to bring out the prediction accuracy and interpretability of the statistical model it produces.

**3. Ridge** - Ridge regression is used to create a model when the number of predictor variables in a set exceeds the number of observations in the given dataset, or when a data set has multicollinearity (correlations between predictor variables) in it.

**4. ElasticNet** - In statistics the fitting of linear or logistic regression models, the elastic net regularize regression method that linearly combines the L₁ and L₂ penalties of both lasso and ridge methods.

**5. SVR** - As in classification, support vector regression (SVR) is characterized by the use of kernels, sparse solution, and VC control of the margin and the number of support vectors. Support Vector Regression (SVR) is quite different than all other Regression models. The Support Vector Machine (SVM) algorithm, a classification category algorithm is used to predict a continuous variable.

**6. KNeighbors Regressor** – It is a Regression based on k-nearest neighbors. In the KNeighbors model target is predicted by local interpolation of the targets which associated to the nearest neighbors in the training set.

7. **Decision Tree Regressor** - Decision tree learning is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a decision tree to go from observations about an item to conclusions about the item's target value.

**8. Random Forest Regressor -** Random Forest uses multiple decision trees as base learning models in the dataset. Random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting in the dataset. The main concept of Random Forest is to combine multiple decision trees in determining the final result rather than relying on individual decision trees.

Importing all the necessary libraries for model building.

* #importing library
* **from** sklearn.linear\_model **import** LinearRegression, Lasso, Ridge, ElasticNet
* **from** sklearn.svm **import** SVR
* **from** sklearn.neighbors **import** KNeighborsRegressor
* **from** sklearn.tree **import** DecisionTreeRegressor
* **from** sklearn.ensemble **import** RandomForestRegressor
* #Importing error Metrics
* **from** sklearn.metrics **import** mean\_absolute\_error, mean\_squared\_error, r2\_score

**Using algorithms via for loop for models “LinearRegression, RandomForestRegressor, DecisionTreeRegressor, KNeighborsRegressor, SVR, Lasso, Ridge and ElasticNet”**

* #Using algorithms via for loop for all models

* model=[LinearRegression(), RandomForestRegressor(), DecisionTreeRegressor(), KNeighborsRegressor(), SVR(), Lasso(), Ridge(), ElasticNet()]
* **for** m **in** model:
* m.fit(x\_train, y\_train)
* **print**('Score of', m, 'is: ', m.score(x\_train, y\_train))
* predm=m.predict(x\_test)
* **print**('\n')
* **print**('Error:')
* **print**('Mean absolute error:', mean\_absolute\_error(y\_test, predm))
* **print**('Mean squared error:', mean\_squared\_error(y\_test, predm))
* **print**('Root mean squared error:', np.sqrt(mean\_squared\_error(y\_test, predm)))
* **print**('r2\_score:', r2\_score(y\_test, predm))
* **print**('\n')
* **print**('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')
* **print**('\n')

**Output**

* Score of LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False) **is**:  0.9999991457165923

* Error:
* Mean absolute error: 0.0016017254094544242
* Mean squared error: 4.166420971502749e-06
* Root mean squared error: 0.002041181268653705
* r2\_score: 0.9999910538693358

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Score of RandomForestRegressor(bootstrap=True, ccp\_alpha=0.0, criterion='mse',
* max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,
* max\_samples=None, min\_impurity\_decrease=0.0,
* min\_impurity\_split=None, min\_samples\_leaf=1,
* min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,
* n\_estimators=100, n\_jobs=None, oob\_score=False,
* random\_state=None, verbose=0, warm\_start=False) **is**:  0.99347013

* Error:
* Mean absolute error: 0.21288533115313707
* Mean squared error: 0.06931822723694563
* Root mean squared error: 0.2632835491194724
* r2\_score: 0.8511600429933043

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Score of DecisionTreeRegressor(ccp\_alpha=0.0, criterion='mse', max\_depth=None,
* max\_features=None, max\_leaf\_nodes=None,
* min\_impurity\_decrease=0.0, min\_impurity\_split=None,
* min\_samples\_leaf=1, min\_samples\_split=2,
* min\_weight\_fraction\_leaf=0.0, presort='deprecated',
* random\_state=None, splitter='best') **is**:  1.0

* Error:
* Mean absolute error: 0.38083908181997755
* Mean squared error: 0.2020194108100649
* Root mean squared error: 0.4494656948089196
* r2\_score: 0.5662243306265349

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Score of KNeighborsRegressor(algorithm='auto', leaf\_size=30, metric='minkowski',
* metric\_params=None, n\_jobs=None, n\_neighbors=5, p=2,
* weights='uniform') **is**:  0.8789246583461614
* Error:
* Mean absolute error: 0.29052782851927217
* Mean squared error: 0.10404187743384233
* Root mean squared error: 0.3225552316020348
* r2\_score: 0.7766014916795883

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Score of SVR(C=1.0, cache\_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
* kernel='rbf', max\_iter=-1, shrinking=True, tol=0.001, verbose=False) **is**:  0.9383081775738186

* Error:
* Mean absolute error: 0.20734343405293582
* Mean squared error: 0.07612578788082267
* Root mean squared error: 0.27590902102110154
* r2\_score: 0.8365428625785246

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Score of Lasso(alpha=1.0, copy\_X=True, fit\_intercept=True, max\_iter=1000,
* normalize=False, positive=False, precompute=False, random\_state=None,
* selection='cyclic', tol=0.0001, warm\_start=False) **is**:  0.5665861052231055

* Error:
* Mean absolute error: 0.4670661836287895
* Mean squared error: 0.32247058583878185
* Root mean squared error: 0.5678649362645856
* r2\_score: 0.3075918117740494

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Score of Ridge(alpha=1.0, copy\_X=True, fit\_intercept=True, max\_iter=None,
* normalize=False, random\_state=None, solver='auto', tol=0.001) **is**:  0.9979907705607476

* Error:
* Mean absolute error: 0.03785502757953374
* Mean squared error: 0.0018773095141137973
* Root mean squared error: 0.04332792995417387
* r2\_score: 0.9959690448168295
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Score of ElasticNet(alpha=1.0, copy\_X=True, fit\_intercept=True, l1\_ratio=0.5,
* max\_iter=1000, normalize=False, positive=False, precompute=False,
* random\_state=None, selection='cyclic', tol=0.0001, warm\_start=False) **is**:  0.8179884993200188

* Error:
* Mean absolute error: 0.29449663207633103
* Mean squared error: 0.14911345732392228
* Root mean squared error: 0.3861521168191653
* r2\_score: 0.6798238867051805

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Cross Validation**

Cross validation helps to find out the over fitting and under fitting of the model.In the cross validation the model is made to run on different subsets of the dataset which will get multiple measures of the model. If we take 5 folds, the data will be divided into 5 pieces where each part being 20% of full dataset. While running the Cross validation the 1st part (20%) of the 5 parts will be kept out as a hold out set for validation and everything else is used for training data. This way we will get the first estimate of the model quality of the dataset. In the similar way further iterations are made for the second 20% of the dataset is held as a hold out set and remaining 4 parts are used for training data during process. This way we will get the second estimate of the model quality of the dataset. These steps are repeated during the cross validation process to get the remaining estimate of the model quality.

* #cross validating the models
* **from** sklearn.model\_selection **import** cross\_val\_score
* model=[LinearRegression(), RandomForestRegressor(), DecisionTreeRegressor(), KNeighborsRegressor(), SVR(), Lasso(), Ridge(), ElasticNet()]
* **for** m **in** model:
* score=cross\_val\_score(m, x, y, cv=5, scoring='r2')
* **print**('Score of', m, 'is:')
* **print**('\n')
* **print**('Score :', score)
* **print**('Mean Score :', score.mean())
* **print**('Standard deviation:', score.std())
* **print**('\n')
* **print**('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')
* **print**('\n')

**Output of Cross Validation**

* Score of LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False) **is**:

* Score : [0.99999674 0.99999833 0.99999619 0.99999587 0.9999987 ]
* Mean Score : 0.9999971665039139
* Standard deviation: 1.1402189057623085e-06

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Score of RandomForestRegressor(bootstrap=True, ccp\_alpha=0.0, criterion='mse',
* max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,
* max\_samples=None, min\_impurity\_decrease=0.0,
* min\_impurity\_split=None, min\_samples\_leaf=1,
* min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,
* n\_estimators=100, n\_jobs=None, oob\_score=False,
* random\_state=None, verbose=0, warm\_start=False) **is**:

* Score : [0.94615613 0.94607959 0.95503381 0.91736481 0.90459791]
* Mean Score : 0.933846448939667
* Standard deviation: 0.019376293328378254

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Score of DecisionTreeRegressor(ccp\_alpha=0.0, criterion='mse', max\_depth=None,
* max\_features=None, max\_leaf\_nodes=None,
* min\_impurity\_decrease=0.0, min\_impurity\_split=None,
* min\_samples\_leaf=1, min\_samples\_split=2,
* min\_weight\_fraction\_leaf=0.0, presort='deprecated',
* random\_state=None, splitter='best') **is**:

* Score : [0.83052753 0.91975779 0.9538462  0.37356152 0.88556073]
* Mean Score : 0.7926507534434293
* Standard deviation: 0.21346797099445375

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Score of KNeighborsRegressor(algorithm='auto', leaf\_size=30, metric='minkowski',
* metric\_params=None, n\_jobs=None, n\_neighbors=5, p=2,
* weights='uniform') **is**:

* Score : [0.70829497 0.82317126 0.87699348 0.67116561 0.63865145]
* Mean Score : 0.7436553548076642
* Standard deviation: 0.09124996059233804

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Score of SVR(C=1.0, cache\_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
* kernel='rbf', max\_iter=-1, shrinking=True, tol=0.001, verbose=False) **is**:

* Score : [0.26451803 0.89736985 0.74282536 0.89824838 0.59808719]
* Mean Score : 0.6802097618835694
* Standard deviation: 0.23586597596629302

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Score of Lasso(alpha=1.0, copy\_X=True, fit\_intercept=True, max\_iter=1000,
* normalize=False, positive=False, precompute=False, random\_state=None,
* selection='cyclic', tol=0.0001, warm\_start=False) **is**:

* Score : [0.2932363  0.29331226 0.39169325 0.49336903 0.31889428]
* Mean Score : 0.35810102346776784
* Standard deviation: 0.0766394864624593

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Score of Ridge(alpha=1.0, copy\_X=True, fit\_intercept=True, max\_iter=None,
* normalize=False, random\_state=None, solver='auto', tol=0.001) **is**:

* Score : [0.99722439 0.99407441 0.994608   0.99454346 0.99706049]
* Mean Score : 0.9955021501916509
* Standard deviation: 0.001352887654030823

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Score of ElasticNet(alpha=1.0, copy\_X=True, fit\_intercept=True, l1\_ratio=0.5,
* max\_iter=1000, normalize=False, positive=False, precompute=False,
* random\_state=None, selection='cyclic', tol=0.0001, warm\_start=False) **is**:

* Score : [0.72007181 0.68306412 0.75137387 0.77997247 0.68883132]
* Mean Score : 0.7246627188810676
* Standard deviation: 0.03690003748690955

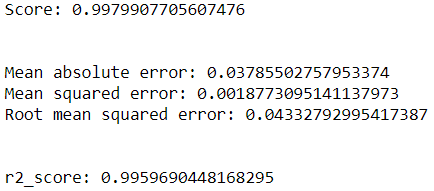
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In the above models we can see that “Linear Regression, Ridge, Random Forest Regressor” are giving the best results for the model. However the Ridge model is performing better compared to other models with respect to the Accuracy score, Cross val score and other parameters. So we can select the Ridge as a finalized model.

**Finalized Model**

* #Using Ridge with best reults
* **from** sklearn.linear\_model **import** Ridge
* rd=Ridge()
* rd.fit(x\_train, y\_train)
* **print**('Score:', rd.score(x\_train, y\_train))
* predrd=rd.predict(x\_test)
* **print**('\n')
* **print**('Mean absolute error:', mean\_absolute\_error(y\_test, predrd))
* **print**('Mean squared error:', mean\_squared\_error(y\_test, predrd))
* **print**('Root mean squared error:', np.sqrt(mean\_squared\_error(y\_test, predrd)))
* **print**('\n')
* **print**('r2\_score:', r2\_score(y\_test, predrd))

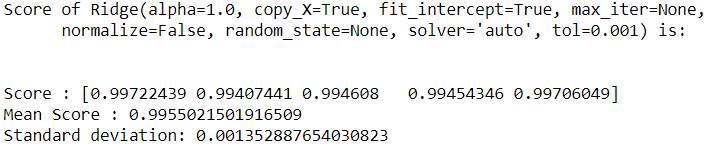
**Output**

****

**Cross Validation of finalized model**

* #cross validating the Ridge model
* score=cross\_val\_score(rd, x, y, cv=5, scoring='r2')
* **print**('Score of', rd, 'is:')
* **print**('\n')
* **print**('Score :', score)
* **print**('Mean Score :', score.mean())
* **print**('Standard deviation:', score.std())

**Output**

****

**Saving the finalized Ridge model**

* # Saving the finalised Ridg model with accuracy score of - 0.9979
* **from** sklearn.externals **import** joblib
* #save the model as a pickle in a file
* joblib.dump(rd, 'covid\_19\_daily\_reports\_us.pkl')

****

**Observations**

1. The Ridge model is giving the r2 score of 0.99 during the model building.

2. The Mean absolute error is calculated as 0.03785 for the Ridge model.

3. The Mean squared error: is determined as 0.00187730 for the given model.

4. The Root mean squared error is calculated as 0.0433279 for Ridge.

5. With score of 0.9979, r2 score of 0.9959690 and during the cross validation the mean score of 0.995502150 makes this Ridge aggressor a perfect model for the given dataset.

6. The ridge is not only giving the perfect score but it is performing way better than other regression models for given dataset.

**NOTE** – The “UnitedStates\_COVID\_19\_dataset” regression problem detailed code can be found in Github repository.

**Github Repository Link** - https://github.com/Carneiro22/Evaluation-Projects/blob/main/Evaluation%20Project%209%20-%20UnitedStates\_COVID\_19\_dataset.ipynb