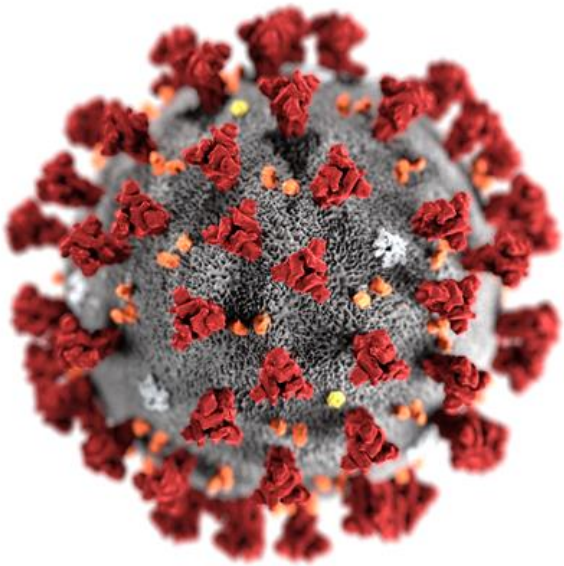
A Project on

Corona Management System (C.M.S)



By: Preetkamal Singh Sandhu

INTRODUCTION TO PYTHON AND CSV FILES

Python is a multi-purpose, object-oriented, **High-Level Programming language**, with applications in multiple areas, including scripting, machine learning, data sciences, scientific learning, cloud computing and artificial intelligence.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages

The best part is that it is an **Open-Source** programming language.

It was developed by **Guido Van Rossum in the year 1991**.

It remained the most popular language of 2019, and it is going to flourish exponentially in upcoming years because of its versatility & flexibility.

So many communities are forced to use python these days, such as:

Network Engineers, Software Engineers, Data Analysts, Mathematicians, Scientists, Accountants and Website & App Developers

Data science and machine learning are the main reasons, why programmers are learning this language.

- Python offers a different framework and libraries, for example, **Matplotlib, Pandas,** and **NumPy**.
- The syntax in Python helps programmers to do coding in fewer steps than Java or C++.
- It is easy and fun to do programming in Python language.
- Python is widely used in large organizations because of its many programming patterns. About 14% of programmers use it on operating systems such as UNIX, Linux, Windows and Mac OS.
- Large companies and programmers use Python because it has created a feature for itself in software development, such as interactive, Modular Dynamic and you will get the large functionality in less code.
- You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Comma-Separated Values (CSV) Files:

A CSV is a comma-separated values file, which allows data to be saved in a tabular format. CSVs look like a garden-variety spreadsheet but with a **.csv extension**.

- CSV files can be used with almost any spreadsheet program, such as Microsoft Excel or Google Spreadsheets.

- They differ from other spreadsheet file types because you can only have a single sheet in a file, they cannot save cell, column, or row. Also, you cannot not save formulas in this format.

These files serve a number of **different purposes**. They help companies export a high volume of data to a more concentrated database.

They also serve some other primary functions:

- CSV files are plain-text files, making them **easier for the website developer to create**.

- Since they're plain text, they're easier to import into a spreadsheet or another storage database, regardless of the specific software you're using.

- To better organize **large amounts of data.**

Since CSV files are easy to organize, ecommerce business owners can manipulate these files in many different ways. CSV files are mostly used for importing and exporting important information, such as customer or order data, to and from the database.

INTRODUCTION

TO

THE PROJECT

This project is based on the processing and merging of datasets to calculate the needed measures and prepare them for an analysis.

In this project, we are going to work with the **COVID-19 dataset** for various states of India, which consists of the data related to the cumulative number of confirmed cases, per day, deaths and active cases in each State.

Also, we have another dataset consists of various life factors, like the number of Hospitals, Beds, Ventilators, Funds available for the **public good**.

This project is broadly divided into **Four categories** i.e., Reading of the data from the dataset, Visualization of the data through different data libraries, Sorting and Manipulation of the data and at the last Updating the dataset.

The software being used in this project is **Python**, along with **CSV files**. Further we will use various Python libraries such as **Pandas**, **Numpy**, etc. for handling the dataset.

Data Visualization is the first step towards getting an insight into a large data set. So, we are using **Matplotlib** library to analyse and visualize the coronavirus dataset.

No special hardware is being used for the execution of this project.

This dataset is interesting because it is highly structured and meaningfully related to **Real-World Figures**.

The reason for taking up this project is that new cases of Coronavirus are increasing rapidly at astonishing rates in India; more than 11 Million people have developed this infection and out of these, around 1.5 lakh people have died of this disease in India.

There is an immediate requirement to store such a large amount of data of these cases, using different data storage technologies. It helps to computationally analyse to reveal patterns, trends, associations and differences.

It can also help in revealing the insights into the spread and control of this virus. This data can be used gainfully to minimise the risk of spreading this virus. The data obtained can further be trained over again for developing future preventive methods.

This information can be effectively used for **Case Identification** and helping to allocate the resources for better protection of public health.

Technology is vital in the fight against coronavirus and future pandemics. Predicting the flow of a pandemic, various emerging technologies can quickly and effectively analyse data to help humans on the frontline figure out the best preparation and response to this and future pandemics.

And finally, perhaps the most important factor to remember is that driving insight from **Data Is All About Actions**.

Most of the analytics team's time and effort are typically on data collection, cleaning and visualization, but the **biggest impact is made by interpretation and action**.

When a piece of analysis is finished, it is important that the analytics teams are thinking about what the insights feed into the business.

By establishing a normal baseline, agencies can compare the volume of virus-related calls for service against previous days. With a combination of internal and external data, agencies can see the frequency of COVID-19-related calls, confirmed cases within a community and **hot spots** or areas where cases are growing.

Agencies can then share these reports with pertinent personnel, from decision-makers to first responders. If an agency places more units in an area where COVID-19 cases are increasing, it **may reduce response times** and help responders mentally prepare for a surge in calls and how to prepare for a patient with obvious symptoms.

Ultimately, the pandemic has shone a light on the **need for accurate and reliable data** and the importance of data-driven decisions that are made based on analyses that are scientifically rigorous and robust.

SOURCE CODE

FOR

THE PROJECT

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


pd.set_option('display.max_columns',None)
pd.set_option('display.width', 5000)


line1="------------------------------------------------------------------------------ \n"
line2="\n------------------------------------------------------------------------------ \n"
line3=("--------------------------------------------------------------------------------")
line4="\n--------------------------------------------------------------------------------"


def menu():
    print("\n \n \t \t \t \t  ###################################### \n")
    print("\t \t \t \t     CORONA MANAGEMENT SYSTEM (C.M.S) \n")
    print("\t \t \t \t  ###################################### \n")
    print(line1)
    print("\t 1.Read Data From The CSV Files \n")
    print("\t 2.Data Visualization \n")
    print("\t 3.Data Sorting And Manipulation \n")
    print("\t 4.Update Data \n")
    print(line3)

menu()


def Error():
    print("\n \n \t \t \t \t INVALID  INPUT ! ")
    print(line2)
    input("\t Press Enter to Re-execute:")
    print(line2)


def ReadData():
    print(line2)
    print("\t --> Read The Following CSV Files: \n")
    print("\t 1. 'Cases' Data \n")
    print("\t 2. 'Resources' Data \n")
    print(line3)
    opt=input("Input Here: ")
    if opt=="1":
        df=pd.read_csv("Cases.csv")
        print(line2)
        print(df)
    elif opt=="2":
        df=pd.read_csv("Resources.csv")
        print(line2)
        print(df)
    else:
        Error()
        ReadData()


def LineChart():
    print(line2)
    df1=pd.read_csv("Cases.csv")
    df2=pd.read_csv("Resources.csv")
    clmn1=df1.columns[1:]
    clmn2=df2.columns[1:]
    print(" --> FIELDS Available : \n")
    for cl1 in clmn1:
        print(cl1)
    for cl2 in clmn2:
        print(cl2)
    print(line4)
```

```
    opt=input("Input Field(String):  STATE v/s : ")
    plt.xlabel("State")
    plt.xticks(rotation='vertical')



    if opt in clmn1:
        print(line1)
        plt.plot(df1['State'],df1[opt],color="red")
        plt.ylabel(opt)
        title="State v/s "+opt
        plt.title(title)
        plt.grid()
        plt.show()

    elif opt in clmn2:
        print(line1)
        plt.plot(df2['State'],df2[opt],color="blue")
        plt.ylabel(opt)
        title="State v/s "+opt
        plt.title(title)
        plt.grid()
        plt.show()

    else:
        Error()
        LineChart()



def BarPlot():
    print(line2)
    df1=pd.read_csv("Cases.csv")
    df2=pd.read_csv("Resources.csv")
    clmn1=df1.columns[1:]
    clmn2=df2.columns[1:]
    print(" --> FIELDS Available : \n")
    for cl1 in clmn1:
        print(cl1)
    for cl2 in clmn2:
        print(cl2)
    print(line4)
    opt=input("Input Field(String):  STATE v/s : ")
    plt.xlabel("State")
    plt.xticks(rotation='vertical')

    if opt in clmn1:
        print(line1)
        plt.bar(df1['State'],df1[opt],color="red")
        plt.ylabel(opt)
        title="State v/s "+opt
        plt.title(title)
        plt.grid()
        plt.show()

    elif opt in clmn2:
        print(line1)
        plt.bar(df2['State'],df2[opt],color="red")
        plt.ylabel(opt)
        title="State v/s "+opt
        plt.title(title)
        plt.grid()
        plt.show()

    else:
        Error()
        BarPlot()
```

```
def PieChart():
    print(line2)
    df1=pd.read_csv("Cases.csv")
    df2=pd.read_csv("Resources.csv")
    clmn1=df1.columns[1:]
    clmn2=df2.columns[1:]
    print(" --> FIELDS Available : \n")
    for cl1 in clmn1:
        print(cl1)
    for cl2 in clmn2:
        print(cl2)
    print(line4)
    opt=input("Input Field(String):  STATE v/s : ")


    if opt in clmn1:
        print(line1)
        plt.pie(df1[opt],autopct="%3d%%")
        title="State v/s "+opt
        plt.title(title)
        plt.legend(df1['State'],title='State',loc='lower left')
        plt.show()


    elif opt in clmn2:
        print(line1)
        plt.pie(df2[opt],autopct="%3d%%")
        title="State v/s "+opt
        plt.title(title)
        plt.legend(df2['State'],title='State',loc='lower left')
        plt.show()


    else:
        Error()
        BarPlot()


def ScatterChart():
    print(line2)
    print("\t --> Select A File For Scatter Chart: \n")
    print("\t 1. 'Cases' Data \n")
    print("\t 2. 'Resources' Data \n")
    print("\t 3.  Both \n")
    print(line3)

    df1=pd.read_csv("Cases.csv")
    df2=pd.read_csv("Resources.csv")
    st=df1['State']
    cf=df1['Confirmed']
    rc=df1['Recovered']
    ac=df1['Active']
    dt=df1['Deaths']
    adc=df1['Avg.DailyCases']
    rcr=df1['Recovery(%)']
    hsp=df2['Hospitals']
    vtl=df2['Ventilators']
    bds=df2['Beds']
    icu=df2['ICU_Beds']
    fdr=df2['Funds_Received(in Crores)']

    ax=plt.gca()
    plt.xlabel("State")
    plt.xticks(rotation='vertical')
    opt=input("Input Here: ")
```

```
    if opt=='1':
        print(line1)
        ax.scatter(st,cf,color='red',label="State wise Confirmed")
        ax.scatter(st,rc,color='green',label="State wise Recovered")
        ax.scatter(st,ac,color='orange',label="State wise Active")
        ax.scatter(st,dt,color='black',label="State wise Deaths")
        ax.scatter(st,adc,color='brown',label="State wise Avg. Daily Cases")
        ax.scatter(st,rcr,color='cyan',label="State wise Recovery Rate(%)")
        plt.legend()
        plt.title("Scatter Chart For 'Cases' Data")
        plt.grid()
        plt.show()


    elif opt=='2':
        print(line1)
        ax.scatter(st,hsp,color='cyan',label="State wise Hospitals Available")
        ax.scatter(st,vtl,color='green',label="State wise Ventilators Available")
        ax.scatter(st,bds,color='orange',label="State wise Beds Available")
        ax.scatter(st,icu,color='black',label="State wise ICU Beds Available")
        ax.scatter(st,fdr,color='red',label="State wise Funds Received(in Crores)")
        plt.legend()
        plt.title("Scatter Chart For 'Resources' Data")
        plt.grid()
        plt.show()


    elif opt=='3':
        print(line1)
        ax.scatter(st,cf,color='red',label="State wise Confirmed")
        ax.scatter(st,rc,color='green',label="State wise Recovered")
        ax.scatter(st,ac,color='orange',label="State wise Active")
        ax.scatter(st,dt,color='black',label="State wise Deaths")
        ax.scatter(st,adc,color='brown',label="State wise Avg. Daily Cases")
        ax.scatter(st,rcr,color='cyan',label="State wise Recovery Rate(%)")
        ax.scatter(st,hsp,color='grey',label="State wise Hospitals Available")
        ax.scatter(st,vtl,color='violet',label="State wise Ventilators Available")
        ax.scatter(st,bds,color='tan',label="State wise Beds Available")
        ax.scatter(st,icu,color='lawngreen',label="State wise ICU Beds Available")
        ax.scatter(st,fdr,color='deepskyblue',label="State wise Funds Received")
        plt.legend()
        plt.title("Complete Scatter Chart")
        plt.grid()
        plt.show()


    else:
        Error()
        ScatterChart()


def CompareTwoStates():
    print(line2)
    print(" --> STATES Available \n")
    df1=pd.read_csv("Cases.csv")
    df2=pd.read_csv("Resources.csv")
    st=df1['State'].to_string(index=False)
    print(st)
    print(line2)
    st1=input("Input State 1(String): ")
    st2=input("Input State 2(String): ")
    if (st1 in st) and (st2 in st) and (st1 != st2) and st1 !=' ' and st2 !=' ' :
        print(line2)
        clmn1=df1.columns[1:]
        clmn2=df2.columns[1:]
        print(" --> FIELDS Available : \n")
```

```
            for cl1 in clmn1:
                print(cl1)
            for cl2 in clmn2:
                print(cl2)
            print(line4)
            field=input("Input A Field(String): ")


            if field in clmn1:
                print(line1)
                plt.bar([st1],[df1.loc[df1['State'] == st1, field].iloc[0]])
                plt.bar([st2],[df1.loc[df1['State'] == st2, field].iloc[0]])
                plt.xlabel('State')
                plt.ylabel(field)
                title=st1+' v/s '+st2+' For '+field
                plt.title(title)
                plt.grid()
                plt.show()


            elif field in clmn2:
                print(line1)
                plt.bar([st1],[df2.loc[df2['State'] == st1, field].iloc[0]])
                plt.bar([st2],[df2.loc[df2['State'] == st2, field].iloc[0]])
                plt.xlabel('State')
                plt.ylabel(field)
                title=st1+' v/s '+st2+' For '+field
                plt.title(title)
                plt.grid()
                plt.show()


            else:
                Error()
                CompareTwoStates()

        else:
            Error()
            CompareTwoStates()



def DataVisualisation():
    print(line2)
    print("\t --> Select A Type of Visualisation: \n")
    print("\t 1.Line Chart \n")
    print("\t 2.Bar Plot \n")
    print("\t 3.Pie Chart \n")
    print("\t 4.Complete Scatter Chart \n")
    print("\t 5.Compare Two States \n")
    print(line3)
    opt=input("Input Here: ")
    if opt=='1':
        LineChart()
    elif opt=='2':
        BarPlot()
    elif opt=='3':
        PieChart()
    elif opt=='4':
        ScatterChart()
    elif opt=='5':
        CompareTwoStates()
    else:
        Error()
        DataVisualisation()
```

```
def DataManipulation():
    print(line2)
    print("\t --> Select an Option: \n")
    print("\t 1.Sort Data \n")
    print("\t 2.Read A Specific Column \n")
    print("\t 3.Read Specific Top and Bottom Records \n")
    print("\t 4.Read a Specific Cell Value \n")
    print("\t 5.Get Sum of Data for a Specific Column\n")
    print(line1)
    df1=pd.read_csv("Cases.csv",index_col=0)
    df2=pd.read_csv("Resources.csv",index_col=0)
    clmn1=df1.columns
    clmn2=df2.columns
    opt=input("Input Here: ")


    if opt=='1':
        print(line2)
        print(" --> Select A Field To Sort : \n")
        for cl1 in clmn1:
            print(cl1)
        for cl2 in clmn2:
            print(cl2)
        print(line4)
        field=input("Input Field(String): ")

        if field in clmn1:
            print(line2)
            print("\t --> Select A Sorting Order : \n")
            print("\t 1.Ascending Order")
            print("\t 2.Descending Order")
            print(line4)
            odr=input("Input Here: ")

            if odr=='1':
                print(line1)
                df1.sort_values([field],inplace=True,ascending=True)
                print(df1)
            elif odr=='2':
                df1.sort_values([field],inplace=True,ascending=False)
                print(df1)
            else:
                Error()
                DataManipulation()


        elif field in clmn2:
            print(line2)
            print("\t --> Select A Sorting Order : \n")
            print("\t 1.Ascending Order")
            print("\t 2.Descending Order")
            print(line4)
            odr=input("Input Here: ")
            if odr=='1':
                print(line1)
                df2.sort_values([field],inplace=True,ascending=True)
                print(df2)
            elif odr=='2':
                df2.sort_values([field],inplace=True,ascending=False)
                print(df2)
            else:
                Error()
                DataManipulation()

        else:
            Error()
            DataManipulation()
```

```
elif opt=='2':
    print(line2)
    print(" --> Select A Specific Column : \n")
    for cl1 in clmn1:
        print(cl1)
    for cl2 in clmn2:
        print(cl2)
    print(line4)
    field=input("Input Column(String): ")

    if field in clmn1:
        print(line2)
        df3=pd.read_csv("Cases.csv",usecols=['State',field])
        print(df3)
    elif field in clmn2:
        print(line2)
        df4=pd.read_csv("Resources.csv",usecols=['State',field])
        print(df4)
    else:
        Error()
        DataManipulation()


elif opt=='3':
    print(line2)
    print("\t --> Select A CSV File : \n")
    print("\t 1. 'Cases' File \n")
    print("\t 2. 'Resources' File")
    print(line4)
    df1=pd.read_csv("Cases.csv",index_col=0)
    df2=pd.read_csv("Resources.csv",index_col=0)
    csv=input("Input Here: ")
    if csv=='1':
        print(line1)
        top=input("\t Input Number of Records To Display From Top: ")
        bottom=input("\t Input Number of Records To Display From Bottom: ")
        if top.isdigit() and bottom.isdigit():
            print(line2)
            print(" --> First",top,"Records : \n")
            print(df1.head(int(top)))
            print('\n')
            print(" --> Last",bottom,"Records : \n")
            print(df1.tail(int(bottom)))
            print('\n')
        else:
            Error()
            DataManipulation()


    elif csv=='2':
        print(line1)
        top=input("\t Input Number of Records To Display From Top: ")
        bottom=input("\t Input Number of Records To Display From Bottom: ")
        if top.isdigit() and bottom.isdigit():
            print(line2)
            print(" --> First",top,"Records : \n")
            print(df2.head(int(top)))
            print('\n')
            print(" --> Last",bottom,"Records : \n")
            print(df2.tail(int(bottom)))
            print('\n')
        else:
            Error()
            DataManipulation()
    else:
        Error()
        DataManipulation()
```

```
elif opt=='4':
    print(line2)
    print(" --> STATES Available \n")
    df1=pd.read_csv("Cases.csv",index_col=0)
    df2=pd.read_csv("Resources.csv",index_col=0)
    df=pd.read_csv("Cases.csv")
    st=df['State'].to_string(index=False)
    print(st)
    print(line4)
    st1=input("Input A State(String): ")

    if (st1 in st) and st1 !='':
        print(line2)
        print(" --> FIELDS Available \n")
        for cl1 in clmn1:
            print(cl1)
        for cl2 in clmn2:
            print(cl2)
        print(line4)
        field=input("Input A Field(String): ")
        if field in clmn1:
            print(line2)
            print("\t --> ",field,"For",st1,"is:",df1.loc[st1,field])
            print(line2)
        elif field in clmn2:
            print(line2)
            print("\t --> ",field,"For",st1,"is:",df2.loc[st1,field])
            print(line2)
        else:
            Error()
            DataManipulation()

    else:
        Error()
        DataManipulation()


elif opt=='5':
    print(line2)
    print(" --> FIELDS Available : \n")
    for cl1 in df1.columns[1:-1]:
        print(cl1)
    for cl2 in clmn2:
        print(cl2)
    print(line4)
    field=input("Input A Field(String): ")
    if field in df1.columns[1:-1]:
        print(line2)
        total=df1[field].sum()
        print("\t --> Total Sum For",field,"Is:",total)
        print(line2)
    elif field in clmn2:
        print(line2)
        total=df2[field].sum()
        print("\t --> Total Sum For",field,"Is:",total)
        print(line2)


    else:
        Error()
        DataManipulation()

else:
    Error()
    DataManipulation()
```

```
def UpdateData():
    print(line2)
    print(" --> STATES Available \n")
    df=pd.read_csv("Cases.csv")
    df1=pd.read_csv("Cases.csv",index_col=0)
    df2=pd.read_csv("Resources.csv",index_col=0)
    st=df['State'].to_string(index=False)
    print(st)
    print(line4)
    st1=input("Input A State Here(String): ")

    if (st1 in st) and st1 !='' :
        print(line2)
        print(" --> Enter A Field To Update : \n")
        clmn1=df1.columns[1:]
        clmn2=df2.columns[1:]
        for cl1 in clmn1:
            print(cl1)
        for cl2 in clmn2:
            print(cl2)
        print(line4)
        field=input("Input Here(String): ")

        if field in clmn1:
            print(line2)
            old=df1.loc[st1,field]
            print("\t Current Value:",old)
            print("\n")
            new=input("Input A New Value: ")

            if new.isdigit() and old != int(new):
                print(line2)
                df1.loc[st1,field]=int(new)
                df1.to_csv("Cases.csv")
                print("\t \t Data Updated Successfully ! \n")

            elif type(eval(new))==float and old != float(new):
                print(line2)
                df1.loc[st1,field]=float(new)
                df1.to_csv("Cases.csv")
                print("\t \t Data Updated Successfully ! \n")
            else:
                Error()
                UpdateData()

        elif field in clmn2:
            print(line2)
            old=df2.loc[st1,field]
            print("\t Current Value:",old)
            print("\n")
            new=input("Input A New Value: ")
            if new.isdigit() and old != int(new):
                print(line2)
                df2.loc[st1,field]=int(new)
                df2.to_csv("Resources.csv")
                print("\t \t Data Updated Successfully ! \n")

            else:
                Error()
                UpdateData()

        else:
            Error()
            UpdateData()

    else:
        Error()
        UpdateData()
```

```
def MainOption():
    opt=input("Input Here: ")

    if opt=='1':
        ReadData()

    elif opt=='2':
        DataVisualisation()

    elif opt=='3':
        DataManipulation()

    elif opt=='4':
        UpdateData()

    else:
        Error()
        menu()
        MainOption()

MainOption()
```

WORKING

OF

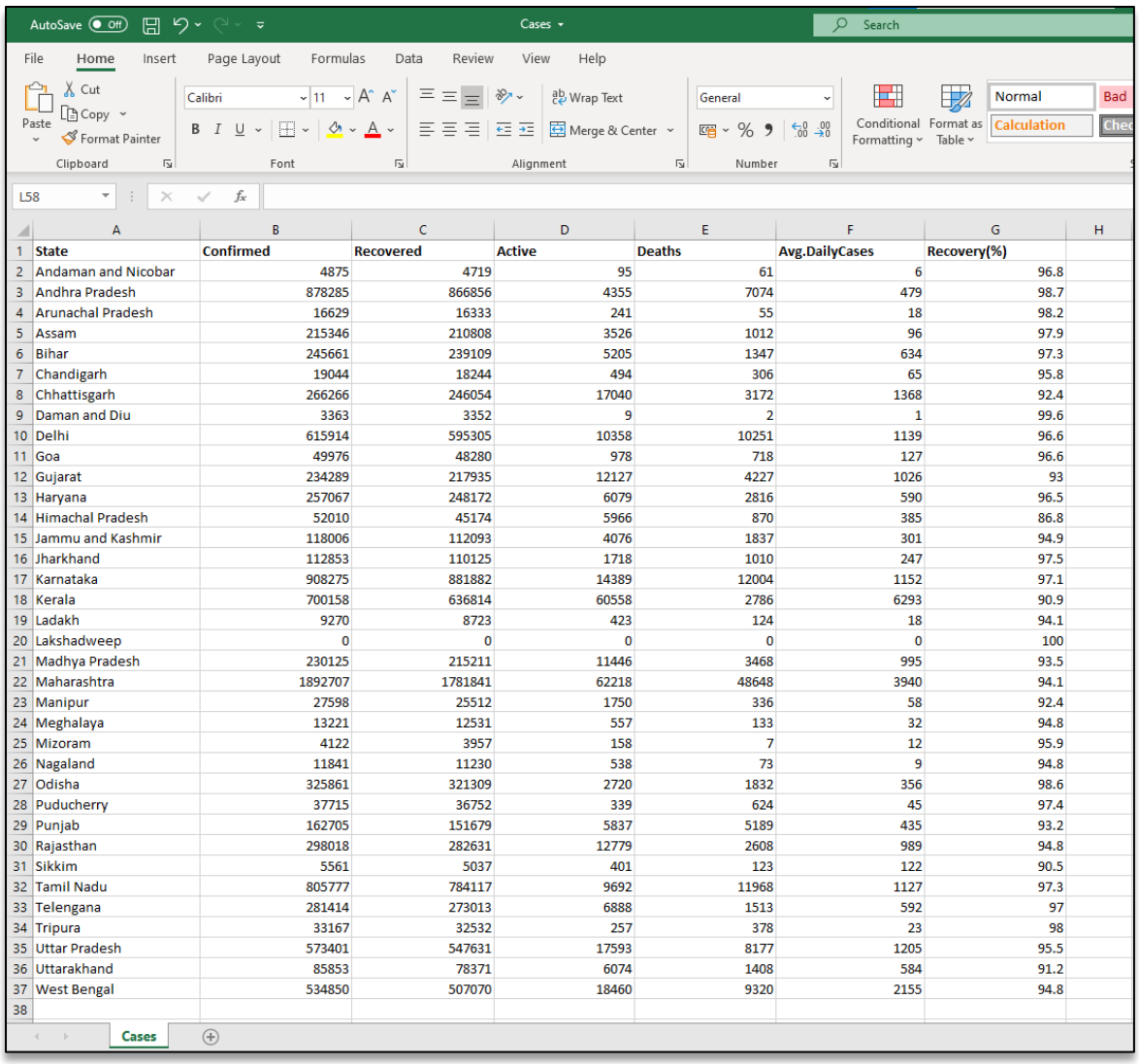
THE PROJECT

CSV DATA FILES USED

Two CSV data files are being incorporated for this project; namely, **Cases.csv** and **Resources.csv**

1. **Cases.csv**

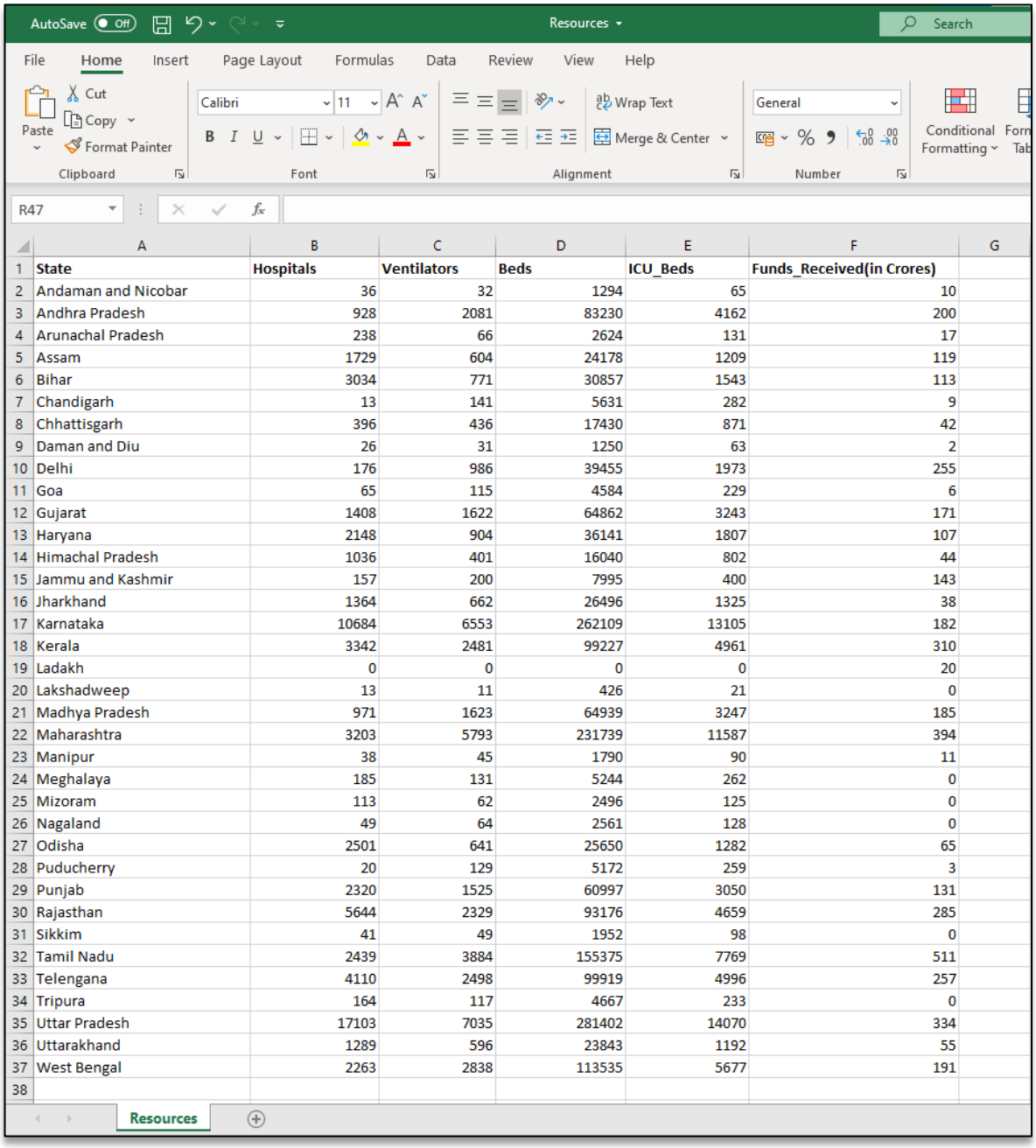
There are six fields in this CSV file i.e, **State**, **Confirmed**, **Recovered, Active, Deaths, Average Daily Cases** and **Recovery Percentage** of all the states of India as per the Government data.



	State	Confirmed	Recovered	Active	Deaths	Avg.DailyCases	Recovery(%)
2	Andaman and Nicobar	4875	4719	95	61	6	96.8
3	Andhra Pradesh	878285	866856	4355	7074	479	98.7
4	Arunachal Pradesh	16629	16333	241	55	18	98.2
5	Assam	215346	210808	3526	1012	96	97.9
6	Bihar	245661	239109	5205	1347	634	97.3
7	Chandigarh	19044	18244	494	306	65	95.8
8	Chhattisgarh	266266	246054	17040	3172	1368	92.4
9	Daman and Diu	3363	3352	9	2	1	99.6
10	Delhi	615914	595305	10358	10251	1139	96.6
11	Goa	49976	48280	978	718	127	96.6
12	Gujarat	234289	217935	12127	4227	1026	93
13	Haryana	257067	248172	6079	2816	590	96.5
14	Himachal Pradesh	52010	45174	5966	870	385	86.8
15	Jammu and Kashmir	118006	112093	4076	1837	301	94.9
16	Jharkhand	112853	110125	1718	1010	247	97.5
17	Karnataka	908275	881882	14389	12004	1152	97.1
18	Kerala	700158	636814	60558	2786	6293	90.9
19	Ladakh	9270	8723	423	124	18	94.1
20	Lakshadweep	0	0	0	0	0	100
21	Madhya Pradesh	230125	215211	11446	3468	995	93.5
22	Maharashtra	1892707	1781841	62218	48648	3940	94.1
23	Manipur	27598	25512	1750	336	58	92.4
24	Meghalaya	13221	12531	557	133	32	94.8
25	Mizoram	4122	3957	158	7	12	95.9
26	Nagaland	11841	11230	538	73	9	94.8
27	Odisha	325861	321309	2720	1832	356	98.6
28	Puducherry	37715	36752	339	624	45	97.4
29	Punjab	162705	151679	5837	5189	435	93.2
30	Rajasthan	298018	282631	12779	2608	989	94.8
31	Sikkim	5561	5037	401	123	122	90.5
32	Tamil Nadu	805777	784117	9692	11968	1127	97.3
33	Telengana	281414	273013	6888	1513	592	97
34	Tripura	33167	32532	257	378	23	98
35	Uttar Pradesh	573401	547631	17593	8177	1205	95.5
36	Uttarakhand	85853	78371	6074	1408	584	91.2
37	West Bengal	534850	507070	18460	9320	2155	94.8

2. **“Resources.csv”**

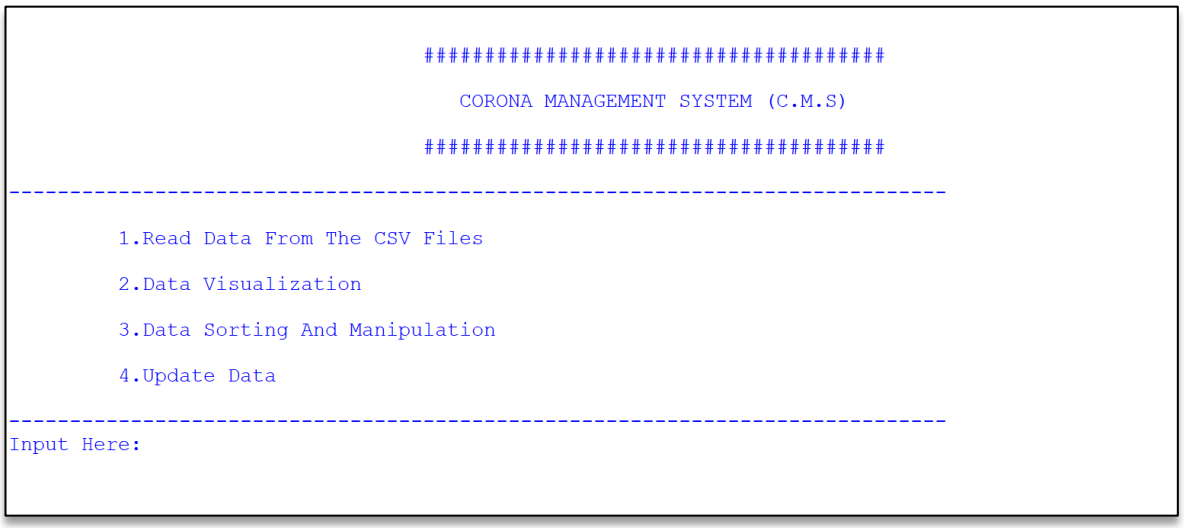
This CSV file contains different data related to the resources available in the country in order to handle the various situations like availability of **Hospitals, Beds, Ventilators** and **Funds** available with the Government.



	A	B	C	D	E	F	G
1	State	Hospitals	Ventilators	Beds	ICU_Beds	Funds_Received(in Crores)	
2	Andaman and Nicobar	36	32	1294	65	10	
3	Andhra Pradesh	928	2081	83230	4162	200	
4	Arunachal Pradesh	238	66	2624	131	17	
5	Assam	1729	604	24178	1209	119	
6	Bihar	3034	771	30857	1543	113	
7	Chandigarh	13	141	5631	282	9	
8	Chhattisgarh	396	436	17430	871	42	
9	Daman and Diu	26	31	1250	63	2	
10	Delhi	176	986	39455	1973	255	
11	Goa	65	115	4584	229	6	
12	Gujarat	1408	1622	64862	3243	171	
13	Haryana	2148	904	36141	1807	107	
14	Himachal Pradesh	1036	401	16040	802	44	
15	Jammu and Kashmir	157	200	7995	400	143	
16	Jharkhand	1364	662	26496	1325	38	
17	Karnataka	10684	6553	262109	13105	182	
18	Kerala	3342	2481	99227	4961	310	
19	Ladakh	0	0	0	0	20	
20	Lakshadweep	13	11	426	21	0	
21	Madhya Pradesh	971	1623	64939	3247	185	
22	Maharashtra	3203	5793	231739	11587	394	
23	Manipur	38	45	1790	90	11	
24	Meghalaya	185	131	5244	262	0	
25	Mizoram	113	62	2496	125	0	
26	Nagaland	49	64	2561	128	0	
27	Odisha	2501	641	25650	1282	65	
28	Puducherry	20	129	5172	259	3	
29	Punjab	2320	1525	60997	3050	131	
30	Rajasthan	5644	2329	93176	4659	285	
31	Sikkim	41	49	1952	98	0	
32	Tamil Nadu	2439	3884	155375	7769	511	
33	Telengana	4110	2498	99919	4996	257	
34	Tripura	164	117	4667	233	0	
35	Uttar Pradesh	17103	7035	281402	14070	334	
36	Uttarakhand	1289	596	23843	1192	55	
37	West Bengal	2263	2838	113535	5677	191	
38							

Introduction to the Main Menu

Execution of the program module results in the following menu, displaying the various options and commands available in the program.



- The First option is to **Read Data** from various CSV data files available.

- The Second option available is offering **Data Visualization** and it is one of the most important feature of this project. It provides the option to visualize the data in different type of plots like **Line Plot, Bar Plot, Scatter Charts** etc.

- The Third option contain various options to **Sort and Manipulate** the data as per the user's choice.

- The fourth option helps to **Update** the field values in any CSV data file as per the changing figures and values of the data .

Option 1: Read Data From CSV File

```
--------------------------------------------------------------------------------

        1.Read Data From The CSV Files

        2.Data Visualization

        3.Data Sorting And Manipulation

        4.Update Data

--------------------------------------------------------------------------------
Input Here: 1
--------------------------------------------------------------------------------

        --> Read The Following CSV Files:

        1. 'Cases' Data

        2. 'Resources' Data

--------------------------------------------------------------------------------
Input Here:
```

- After choosing this option, it will show various CSV files available where we need to input the required option to get the desired output as shown below.

```
Input Here: 1
--------------------------------------------------------------------------------
```

	State	Confirmed	Recovered	Active	Deaths	Avg.DailyCases	Recovery(%)
0	Andaman and Nicobar	4875	4719	95	61	6	96.8
1	Andhra Pradesh	878285	866856	4355	7074	479	98.7
2	Arunachal Pradesh	16629	16333	241	55	18	98.2
3	Assam	215346	210808	3526	1012	96	97.9
4	Bihar	245661	239109	5205	1347	634	97.3
5	Chandigarh	19044	18244	494	306	65	95.8
6	Chhattisgarh	266266	246054	17040	3172	1368	92.4
7	Daman and Diu	3363	3352	9	2	1	99.6
8	Delhi	615914	595305	10358	10251	1139	96.6
9	Goa	49976	48280	978	718	127	96.6
10	Gujarat	234289	217935	12127	4227	1026	93.0
11	Haryana	257067	248172	6079	2816	590	96.5
12	Himachal Pradesh	52010	45174	5966	870	385	86.8
13	Jammu and Kashmir	118006	112093	4076	1837	301	94.9
14	Jharkhand	112853	110125	1718	1010	247	97.5
15	Karnataka	908275	881882	14389	12004	1152	97.1
16	Kerala	700158	636814	60558	2786	6293	90.9
17	Ladakh	9270	8723	423	124	18	94.1
18	Lakshadweep	0	0	0	0	0	100.0
19	Madhya Pradesh	230125	215211	11446	3468	995	93.5
20	Maharashtra	1892707	1781841	62218	48648	3940	94.1
21	Manipur	27598	25512	1750	336	58	92.4
22	Meghalaya	13221	12531	557	133	32	94.8
23	Mizoram	4122	3957	158	7	12	95.9
24	Nagaland	11841	11230	538	73	9	94.8
25	Odisha	325861	321309	2720	1832	356	98.6
26	Puducherry	37715	36752	339	624	45	97.4
27	Punjab	162705	151679	5837	5189	435	93.2
28	Rajasthan	298018	282631	12779	2608	989	94.8
29	Sikkim	5561	5037	401	123	122	90.5
30	Tamil Nadu	805777	784117	9692	11968	1127	97.3
31	Telengana	281414	273013	6888	1513	592	97.0
32	Tripura	33167	32532	257	378	23	98.0
33	Uttar Pradesh	573401	547631	17593	8177	1205	95.5
34	Uttarakhand	85853	78371	6074	1408	584	91.2
35	West Bengal	534850	507070	18460	9320	2155	94.8

```
>>>
```

Option 2: Data Visualization

On selecting this option, we will further get five **Sub-Options**.

They contain different type of data visualization techniques available for plotting the data like:

- Line Chart
- Bar Plot
- Pie Chart
- Scatter Plot

The last option is to **compare** the data between two specific states for a specific field value.

This Menu looks as follows:

```
--------------------------------------------------------------------------------

        1.Read Data From The CSV Files

        2.Data Visualization

        3.Data Sorting And Manipulation

        4.Update Data

--------------------------------------------------------------------------------
Input Here: 2
--------------------------------------------------------------------------------

        --> Select A Type of Visualisation:

        1.Line Chart

        2.Bar Plot

        3.Pie Chart

        4.Complete Scatter Chart

        5.Compare Two States

--------------------------------------------------------------------------------
Input Here:
```

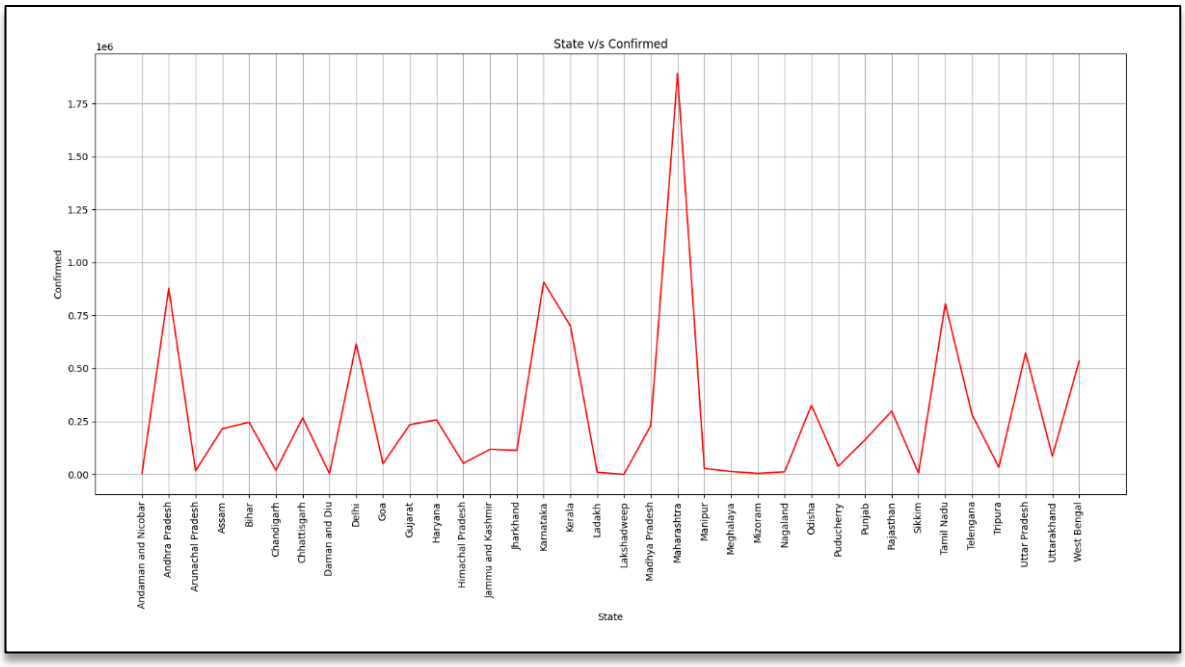
Sub-Option 1: **Line Chart**

```
--------------------------------------------------------------------------------

         --> Select A Type of Visualisation:

         1.Line Chart

         2.Bar Plot

         3.Pie Chart

         4.Complete Scatter Chart

         5.Compare Two States

--------------------------------------------------------------------------------
Input Here: 1

--------------------------------------------------------------------------------

 --> FIELDS Available :

Confirmed
Recovered
Active
Deaths
Avg.DailyCases
Recovery(%)
Hospitals
Ventilators
Beds
ICU_Beds
Funds_Received(in Crores)

--------------------------------------------------------------------------------
Input Field(String):  STATE v/s : Confirmed
```

- After choosing the option for Line Chart, we need to give input for the field against which data is to be plotted.

- For example, in this case we are plotting data for **State v/s Confirmed Cases** .

- So, on giving this input, we get the following **Line Chart** as the output:

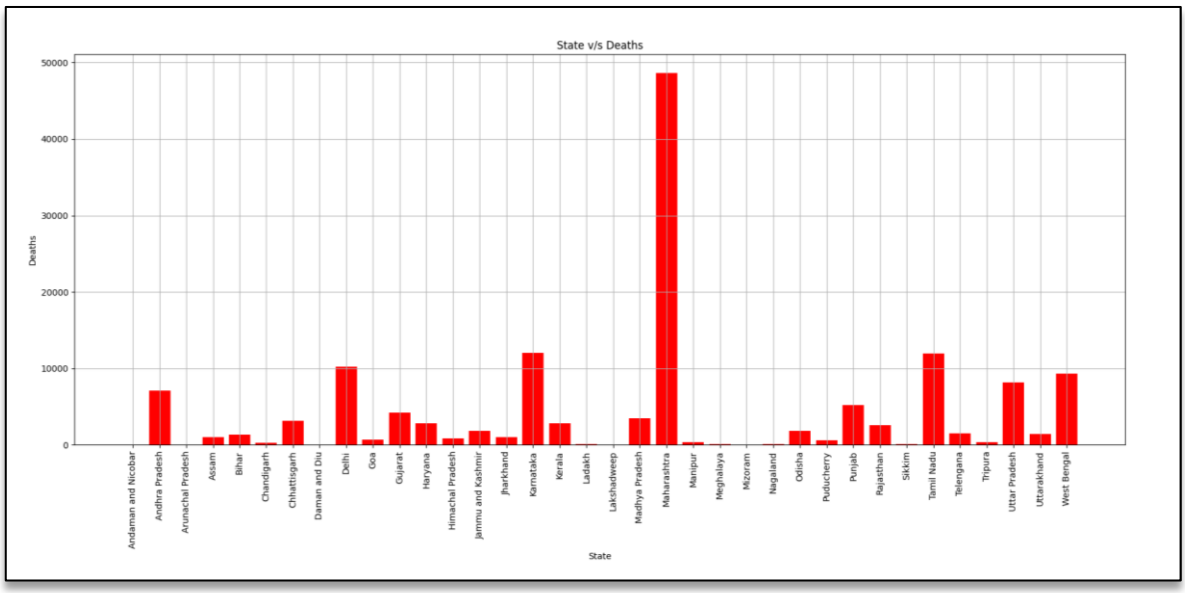

State v/s Confirmed
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Confirmed
Andaman and Nicobar
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chandigarh
Chhattisgarh
Daman and Diu
Delhi
Goa
Gujarat
Haryana
Himachal Pradesh
Jammu and Kashmir
Jharkhand
Karnataka
Kerala
Ladakh
Lakshadweep
Madhya Pradesh
Maharashtra
Manipur
Meghalaya
Mizoram
Nagaland
Odisha
Puducherry
Punjab
Rajasthan
Sikkim
Tamil Nadu
Telengana
Tripura
Uttar Pradesh
Uttarakhand
West Bengal
State

P.T.O

Sub-Option 2: **Bar Plot**

```
--------------------------------------------------------------------------------

        --> Select A Type of Visualisation:

        1.Line Chart

        2.Bar Plot

        3.Pie Chart

        4.Complete Scatter Chart

        5.Compare Two States

--------------------------------------------------------------------------------
Input Here: 2

--------------------------------------------------------------------------------

 --> FIELDS Available :

Confirmed
Recovered
Active
Deaths
Avg.DailyCases
Recovery(%)
Hospitals
Ventilators
Beds
ICU_Beds
Funds_Received(in Crores)

--------------------------------------------------------------------------------
Input Field(String):  STATE v/s : Deaths
```

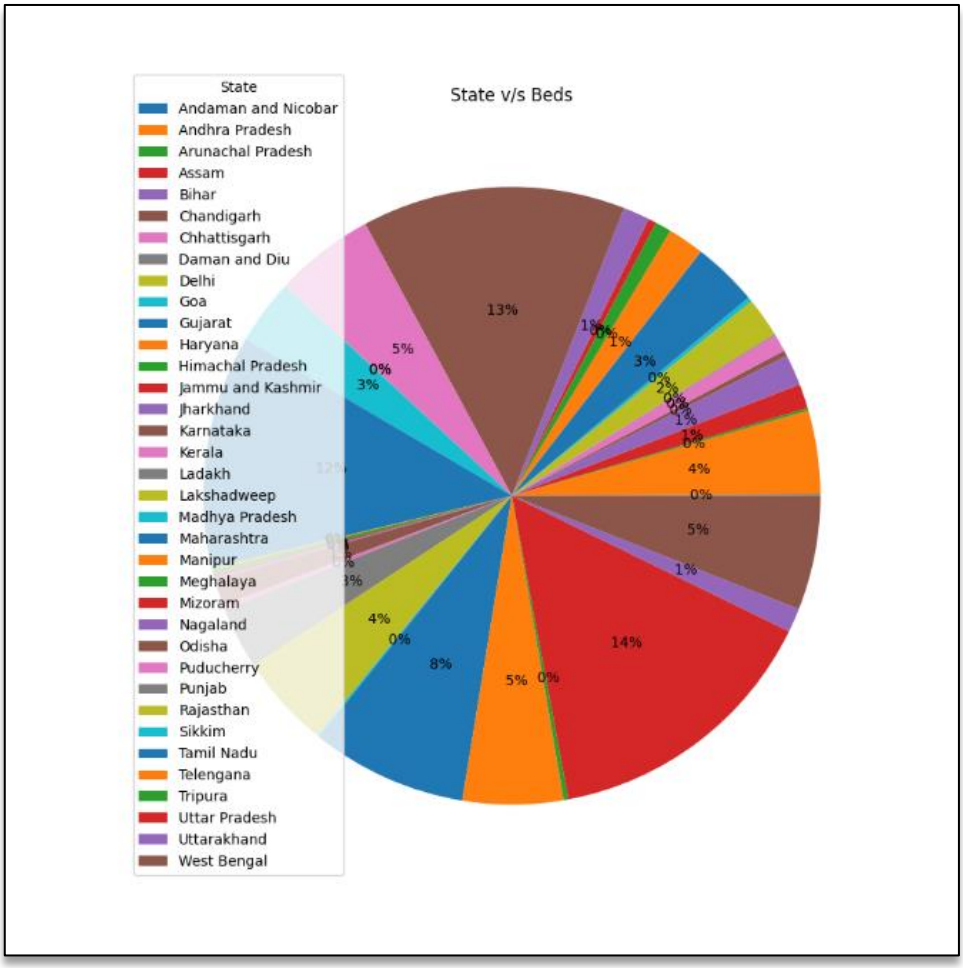
- After giving the required field value, we get the following **Bar Plot** as the output



Sub-Option 3: **Pie Chart**

```
--------------------------------------------------------------------------------

         --> Select A Type of Visualisation:

         1.Line Chart

         2.Bar Plot

         3.Pie Chart

         4.Complete Scatter Chart

         5.Compare Two States

--------------------------------------------------------------------------------
Input Here: 3

--------------------------------------------------------------------------------

 --> FIELDS Available :

Confirmed
Recovered
Active
Deaths
Avg.DailyCases
Recovery(%)
Hospitals
Ventilators
Beds
ICU_Beds
Funds_Received(in Crores)

--------------------------------------------------------------------------------
Input Field(String):  STATE v/s : Beds
```

- After giving the required input for the field value, For Example **Beds Data** in this case, we got the following **Pie Chart** as the output(on next page).


State v/s Beds
State
Andaman and Nicobar
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chandigarh
Chhattisgarh
Daman and Diu
Delhi
Goa
Gujarat
Haryana
Himachal Pradesh
Jammu and Kashmir
Jharkhand
Karnataka
Kerala
Ladakh
Lakshadweep
Madhya Pradesh
Maharashtra
Manipur
Meghalaya
Mizoram
Nagaland
Odisha
Puducherry
Punjab
Rajasthan
Sikkim
Tamil Nadu
Telengana
Tripura
Uttar Pradesh
Uttarakhand
West Bengal
13%
5%
0%
3%
1%
3%
4%
0%
5%
1%
14%
0%
5%
8%
4%
0%

P.T.O

Sub-Option 4: **Scatter Plot**

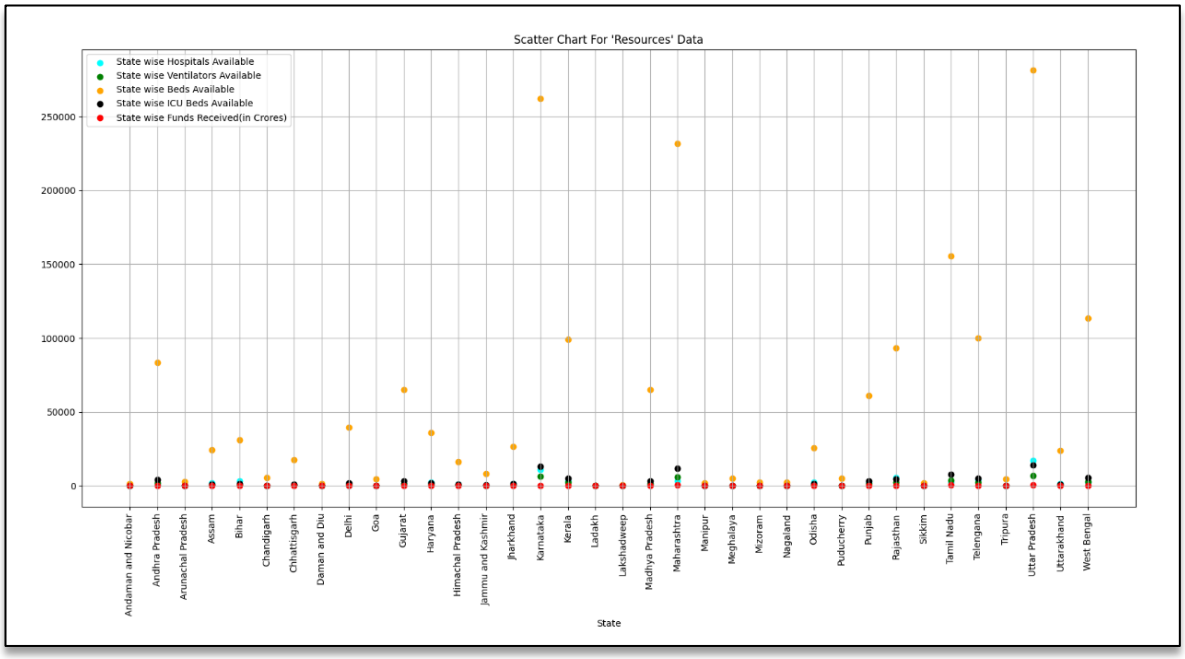
- Here, we have two options for visualizing the scatter plot.

- The First option is to plot the data for a single CSV file i.e either **Cases.csv** or **Resources.csv** file.

- The second option can be used to plot the data for both the CSV files in a single Scatter Plot.

```
--------------------------------------------------------------------------------

        --> Select A Type of Visualisation:

        1.Line Chart

        2.Bar Plot

        3.Pie Chart

        4.Complete Scatter Chart

        5.Compare Two States

--------------------------------------------------------------------------------
Input Here: 4

--------------------------------------------------------------------------------

        --> Select A File For Scatter Chart:

        1. 'Cases' Data

        2. 'Resources' Data

        3.  Both

--------------------------------------------------------------------------------
Input Here: 2
--------------------------------------------------------------------------------

>>>
```

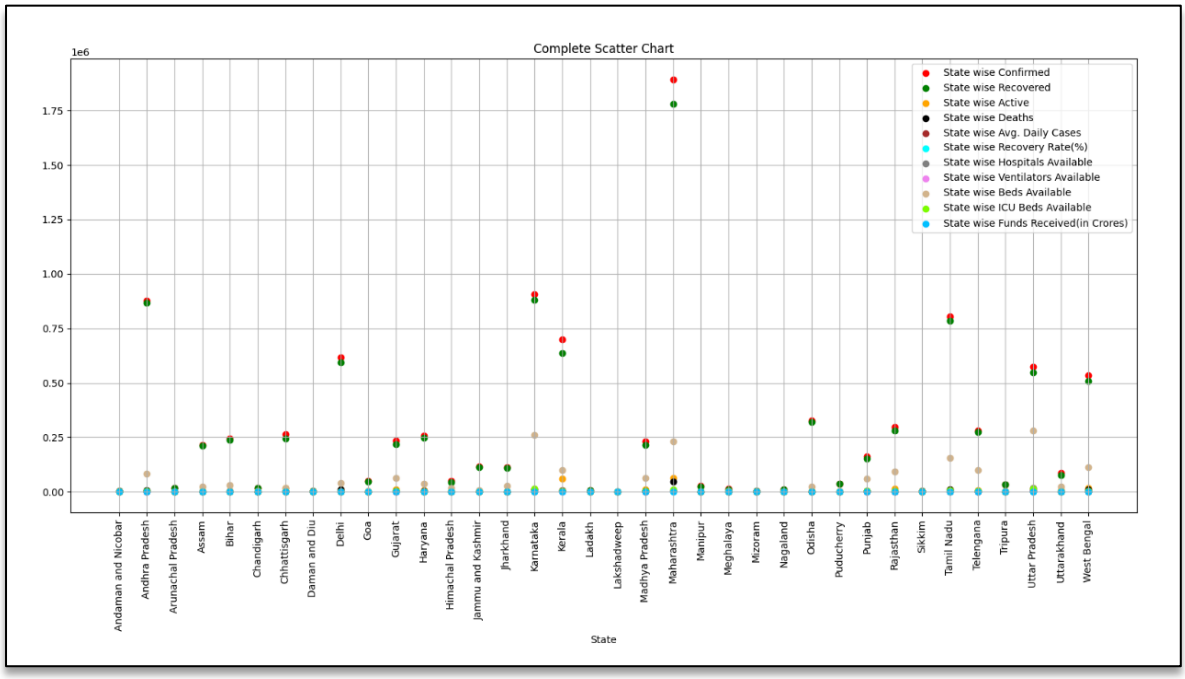
- Output for the Scatter Plot for a single CSV file is as on the next page:



- Now in order to display the scatter chart for both CSV files, we give the following input:

```
--------------------------------------------------------------------------------

        --> Select A Type of Visualisation:

        1.Line Chart

        2.Bar Plot

        3.Pie Chart

        4.Complete Scatter Chart

        5.Compare Two States

--------------------------------------------------------------------------------
Input Here: 4

--------------------------------------------------------------------------------

        --> Select A File For Scatter Chart:

        1. 'Cases' Data

        2. 'Resources' Data

        3.  Both

--------------------------------------------------------------------------------
Input Here: 3
--------------------------------------------------------------------------------

>>>
```

- Output for this **Scatter Plot** is as follows:



P.T.O

Sub-Option 5: **Comparing Two States**

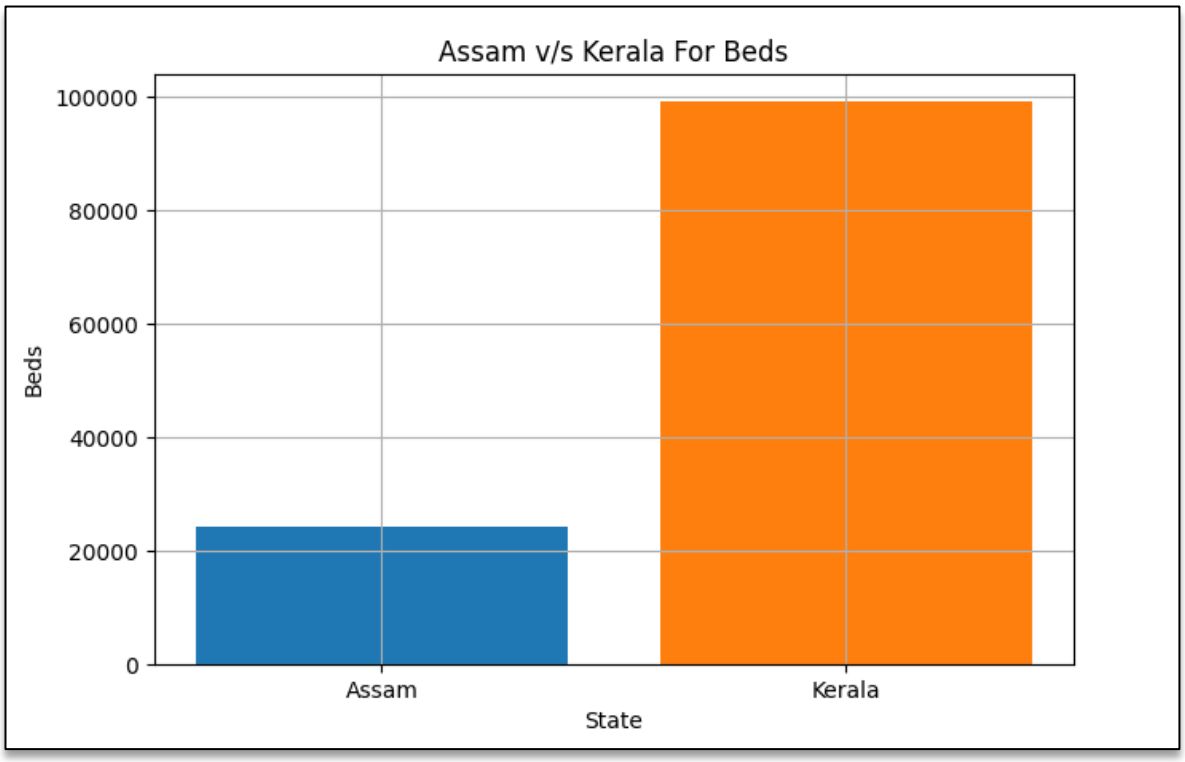
- This feature helps to **Compare Two Different States** on the basis of a specific field value.

```
--------------------------------------------------------------------------------

        --> Select A Type of Visualisation:

        1.Line Chart

        2.Bar Plot

        3.Pie Chart

        4.Complete Scatter Chart

        5.Compare Two States

--------------------------------------------------------------------------------
Input Here: 5
```

- We will get the list of all the States available in the CSV data file.
- It will prompt to give input of any two States as per the requirement of the user.
- Let us consider a case, in which we want to compare the data for the states of **Assam** and **Kerala**.
- After giving the input for the states, all the different fields available for comparison will be listed.
- We need to give the input for the field which is to be compared.
- For illustration, let us take the **Beds** column for the purpose of comparison.

```
            Nagaland
               Odisha
           Puducherry
               Punjab
            Rajasthan
               Sikkim
           Tamil Nadu
            Telengana
              Tripura
        Uttar Pradesh
          Uttarakhand
          West Bengal

--------------------------------------------------------------------------------

Input State 1(String): Assam
Input State 2(String): Kerala

--------------------------------------------------------------------------------

 --> FIELDS Available :

Confirmed
Recovered
Active
Deaths
Avg.DailyCases
Recovery(%)
Hospitals
Ventilators
Beds
ICU_Beds
Funds_Received(in Crores)

--------------------------------------------------------------------------------
Input A Field(String): Beds
```

- After giving the desired inputs, we get the following **Bar Plot** as the output:



Option 3: Data Sorting and Manipulation

This option contains various types of Commands/Sub-options to **manipulate the data** like:

- **Sorting the data** of a specific column in **Ascending or Descending** order.
- Reading the data of a **specific field** value.
- Using **head()** and **tail()** functions to read a specific number of rows from the top and bottom of the CSV data file.
- Get a specific **cell value** for a particular State against a particular field value.
- Getting the **Sum of the data values** for any desired field.

```
--------------------------------------------------------------------------------

        1.Read Data From The CSV Files

        2.Data Visualization

        3.Data Sorting And Manipulation

        4.Update Data

--------------------------------------------------------------------------------
Input Here: 3
--------------------------------------------------------------------------------

        --> Select an Option:

        1.Sort Data

        2.Read A Specific Column

        3.Read Specific Top and Bottom Records

        4.Read a Specific Cell Value

        5.Get Sum of Data for a Specific Column

--------------------------------------------------------------------------------

Input Here:
```

Sub-Option 1: **Sorting The Data**

- Here we can sort the data in a specific order that is **Ascending** or **Descending** order, as per the desired CSV file.

```
--------------------------------------------------------------------------------

        --> Select an Option:

        1.Sort Data

        2.Read A Specific Column

        3.Read Specific Top and Bottom Records

        4.Read a Specific Cell Value

        5.Get Sum of Data for a Specific Column

--------------------------------------------------------------------------------

Input Here: 1
```

- After giving an input for the field, we need to give the input for the **desired order** of sorting i.e., ascending or descending order.

```
--------------------------------------------------------------------------------

 --> Select A Field To Sort :

Confirmed
Recovered
Active
Deaths
Avg.DailyCases
Recovery(%)
Hospitals
Ventilators
Beds
ICU_Beds
Funds_Received(in Crores)

--------------------------------------------------------------------------------
Input Field(String): Recovered

--------------------------------------------------------------------------------

        --> Select A Sorting Order :

        1.Ascending Order
        2.Descending Order

--------------------------------------------------------------------------------
Input Here: 2
```

- Here, we are sorting the **Recovered** field in **descending order**. After giving the desired input, we get the following output:

```
        --> Select A Sorting Order :

        1.Ascending Order
        2.Descending Order

--------------------------------------------------------------------------------
Input Here: 2
                   Confirmed  Recovered  Active  Deaths  Avg.DailyCases  Recovery(%)
State
Maharashtra          1892707    1781841   62218   48648            3940         94.1
Karnataka             908275     881882   14389   12004            1152         97.1
Andhra Pradesh        878285     866856    4355    7074             479         98.7
Tamil Nadu            805777     784117    9692   11968            1127         97.3
Kerala                700158     636814   60558    2786            6293         90.9
Delhi                 615914     595305   10358   10251            1139         96.6
Uttar Pradesh         573401     547631   17593    8177            1205         95.5
West Bengal           534850     507070   18460    9320            2155         94.8
Odisha                325861     321309    2720    1832             356         98.6
Rajasthan             298018     282631   12779    2608             989         94.8
Telengana             281414     273013    6888    1513             592         97.0
Haryana               257067     248172    6079    2816             590         96.5
Chhattisgarh          266266     246054   17040    3172            1368         92.4
Bihar                 245661     239109    5205    1347             634         97.3
Gujarat               234289     217935   12127    4227            1026         93.0
Madhya Pradesh        230125     215211   11446    3468             995         93.5
Assam                 215346     210808    3526    1012              96         97.9
Punjab                162705     151679    5837    5189             435         93.2
Jammu and Kashmir     118006     112093    4076    1837             301         94.9
Jharkhand             112853     110125    1718    1010             247         97.5
Uttarakhand            85853      78371    6074    1408             584         91.2
Goa                    49976      48280     978     718             127         96.6
Himachal Pradesh       52010      45174    5966     870             385         86.8
Puducherry             37715      36752     339     624              45         97.4
Tripura                33167      32532     257     378              23         98.0
Manipur                27598      25512    1750     336              58         92.4
Chandigarh             19044      18244     494     306              65         95.8
Arunachal Pradesh      16629      16333     241      55              18         98.2
Meghalaya              13221      12531     557     133              32         94.8
Nagaland               11841      11230     538      73               9         94.8
Ladakh                  9270       8723     423     124              18         94.1
Sikkim                  5561       5037     401     123             122         90.5
Andaman and Nicobar     4875       4719      95      61               6         96.8
Mizoram                 4122       3957     158       7              12         95.9
Daman and Diu           3363       3352       9       2               1         99.6
Lakshadweep                0          0       0       0               0        100.0
>>>
```

P.T.O

Sub-Option 2: **Read A Specific Column**

```
--------------------------------------------------------------------------------

         --> Select an Option:

         1.Sort Data

         2.Read A Specific Column

         3.Read Specific Top and Bottom Records

         4.Read a Specific Cell Value

         5.Get Sum of Data for a Specific Column

--------------------------------------------------------------------------------

Input Here: 2
```

- This option is to be used when we require the data of a specific field for all the states.

```
--------------------------------------------------------------------------------

         --> Select an Option:

         1.Sort Data

         2.Read A Specific Column

         3.Read Specific Top and Bottom Records

         4.Read a Specific Cell Value

         5.Get Sum of Data for a Specific Column

--------------------------------------------------------------------------------

Input Here: 2

--------------------------------------------------------------------------------

 --> Select A Specific Column :

Confirmed
Recovered
Active
Deaths
Avg.DailyCases
Recovery(%)
Hospitals
Ventilators
Beds
ICU_Beds
Funds_Received(in Crores)

--------------------------------------------------------------------------------
Input Column(String): Hospitals
```

- For the purpose of illustration, here we have taken the **Hospitals** field into consideration.

- So, all data related to **Hospitals** field only is shown against each state.

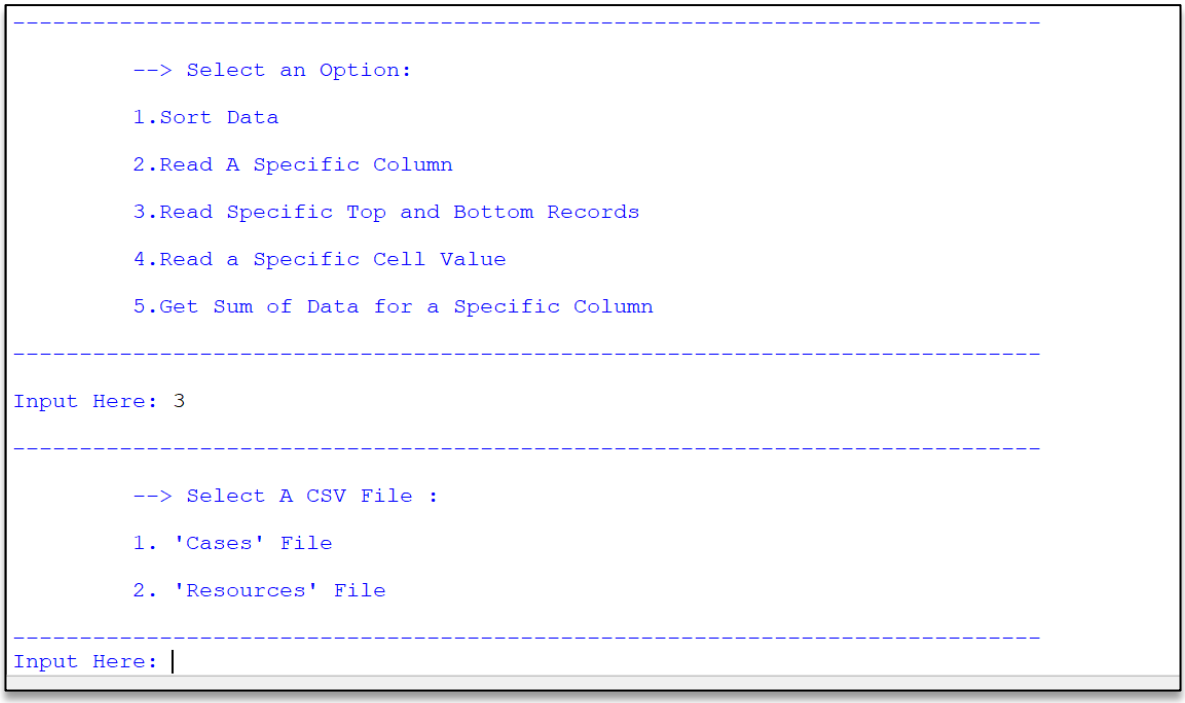
- The output for the given input is as shown on the next page:

```
Input Column(String): Hospitals

--------------------------------------------------

                  State  Hospitals
0   Andaman and Nicobar         36
1        Andhra Pradesh        928
2     Arunachal Pradesh        238
3                 Assam       1729
4                 Bihar       3034
5            Chandigarh         13
6          Chhattisgarh        396
7         Daman and Diu         26
8                 Delhi        176
9                   Goa         65
10              Gujarat       1408
11              Haryana       2148
12    Himachal Pradesh        1036
13   Jammu and Kashmir         157
14            Jharkhand       1364
15            Karnataka      10684
16               Kerala       3342
17               Ladakh          0
18          Lakshadweep         13
19       Madhya Pradesh        971
20          Maharashtra       3203
21              Manipur         38
22            Meghalaya        185
23              Mizoram        113
24             Nagaland         49
25               Odisha       2501
26           Puducherry         20
27               Punjab       2320
28            Rajasthan       5644
29               Sikkim         41
30           Tamil Nadu       2439
31            Telengana       4110
32              Tripura        164
33        Uttar Pradesh      17103
34          Uttarakhand       1289
35          West Bengal       2263
>>>
```

Sub-Option 3: **Read Specific Top And Bottom Records**

- This option makes the use of **head()** and **tail()** function of Python Pandas in order to display specific number of rows from top and bottom of the CSV data file.

- Here, we are using the **Cases.csv** file for the purpose of illustration.



- After this, we need to specify the number of rows from top and bottom to be displayed from the csv file. We get the following output for **3 and 4 Rows** from top and bottom respectively.

```
--------------------------------------------------------------------------------

        --> Select A CSV File :

        1. 'Cases' File

        2. 'Resources' File

--------------------------------------------------------------------------------
Input Here: 1
--------------------------------------------------------------------------------

        Input Number of Records To Display From Top: 3
        Input Number of Records To Display From Bottom: 4

--------------------------------------------------------------------------------

 --> First 3 Records :

                    Confirmed  Recovered  Active  Deaths  Avg.DailyCases  Recovery(%)
State
Andaman and Nicobar       4875       4719      95      61               6         96.8
Andhra Pradesh          878285     866856    4355    7074             479         98.7
Arunachal Pradesh        16629      16333     241      55              18         98.2


 --> Last 4 Records :

              Confirmed  Recovered  Active  Deaths  Avg.DailyCases  Recovery(%)
State
Tripura           33167      32532     257     378              23         98.0
Uttar Pradesh    573401     547631   17593    8177            1205         95.5
Uttarakhand       85853      78371    6074    1408             584         91.2
West Bengal      534850     507070   18460    9320            2155         94.8


>>>
```

P.T.O

Sub-Option 4: **Read A Specific Cell Value**

- This sub-option is extremely helpful if we desire to read the data value for a **specific field** and that for a **specific state**.

- For example, if we want to know the **number of Confirmed Cases in Kerala**, then we can use this option.

```
--------------------------------------------------------------------------------

        1.Read Data From The CSV Files

        2.Data Visualization

        3.Data Sorting And Manipulation

        4.Update Data

--------------------------------------------------------------------------------
Input Here: 3
--------------------------------------------------------------------------------

        --> Select an Option:

        1.Sort Data

        2.Read A Specific Column

        3.Read Specific Top and Bottom Records

        4.Read a Specific Cell Value

        5.Get Sum of Data for a Specific Column

--------------------------------------------------------------------------------
Input Here: 4
```

- We get the list of all the available states and fields in the CSV file.

- Let us say we need the number of **Active Cases in the state of Odisha**.

- The following inputs will help us to achieve the desired results:

```
          Tamil Nadu
           Telengana
             Tripura
       Uttar Pradesh
         Uttarakhand
         West Bengal

--------------------------------------------------------------------------------
Input A State(String): Odisha

--------------------------------------------------------------------------------

 --> FIELDS Available

Confirmed
Recovered
Active
Deaths
Avg.DailyCases
Recovery(%)
Hospitals
Ventilators
Beds
ICU_Beds
Funds_Received(in Crores)

--------------------------------------------------------------------------------
Input A Field(String): Active

--------------------------------------------------------------------------------

         -->  Active For Odisha is: 2720

--------------------------------------------------------------------------------

>>>
```

Sub-Option 5: **Getting Sum Of Data For A Specific Field**

- This option proves helpful if we want to get the **Sum of all the data values** for a specific field.
- After selecting this option, list of all the available fields is displayed.
- The user needs to input the field as per the requirement.
- Let us assume that we require the number of **total ventilators** available in the country i.e., in all the states.
- The desired input and output would be as shown:

```
         2.Read A Specific Column

         3.Read Specific Top and Bottom Records

         4.Read a Specific Cell Value

         5.Get Sum of Data for a Specific Column

--------------------------------------------------------------------------------

Input Here: 5

--------------------------------------------------------------------------------

 --> FIELDS Available :

Recovered
Active
Deaths
Avg.DailyCases
Hospitals
Ventilators
Beds
ICU_Beds
Funds_Received(in Crores)

--------------------------------------------------------------------------------
Input A Field(String): Ventilators

--------------------------------------------------------------------------------

         --> Total Sum For Ventilators Is: 47456

--------------------------------------------------------------------------------

>>>
```

Option 4: Update Data

- At the last, there is an option to **Update the Data**.

- This option is useful in order to update any field value in the desired CSV file.

- This option is very import as the **figures are going to change every day** and the **data needs to be updated on a regular basis** in order to keep a proper track of the situation.

```
                                ######################################
                                   CORONA MANAGEMENT SYSTEM (C.M.S)
                                ######################################
--------------------------------------------------------------------------------
        1.Read Data From The CSV Files
        2.Data Visualization
        3.Data Sorting And Manipulation
        4.Update Data
--------------------------------------------------------------------------------
Input Here: 4
```

- After selecting this option, all the available states are listed. After giving input of the desired state, we will get the list of fields and then we are to input the field which needs to be modified.

- Once we give these inputs, the system will show the **current data value** stored in that specific cell and prompts the user to **input the new value**.

- Let us say we need to update the data for availability of **ICU Beds in Punjab State**. We need to give the desired inputs as shown:

```
              Sikkim
          Tamil Nadu
           Telengana
             Tripura
       Uttar Pradesh
         Uttarakhand
         West Bengal

--------------------------------------------------------------------------------
Input A State Here(String): Punjab

--------------------------------------------------------------------------------

 --> Enter A Field To Update :

Recovered
Active
Deaths
Avg.DailyCases
Recovery(%)
Ventilators
Beds
ICU_Beds
Funds_Received(in Crores)

--------------------------------------------------------------------------------
Input Here(String): ICU_Beds

--------------------------------------------------------------------------------

         Current Value: 3050


Input A New Value: 4239
```

- We can also see the current cell value.

- Now we need to input the new field value.

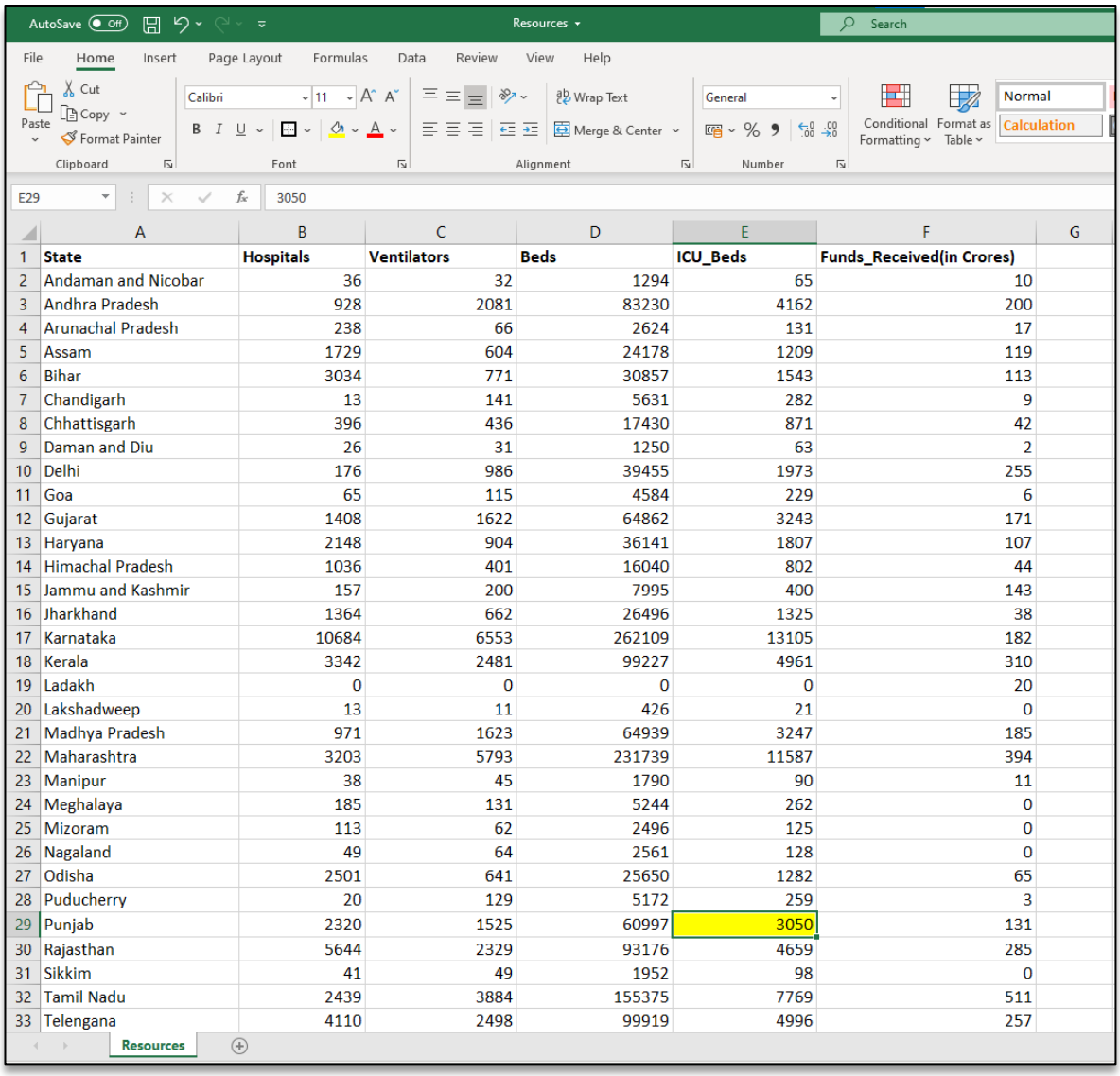
- Once we enter the new value, we get the message of "**Data Updated Successfully!**" and the field value in the CSV file gets updated.

- The final output would be as shown on the next page:

```
--------------------------------------------------------------------------------

 --> Enter A Field To Update :

Recovered
Active
Deaths
Avg.DailyCases
Recovery(%)
Ventilators
Beds
ICU_Beds
Funds_Received(in Crores)
Unnamed: 6
Unnamed: 7

--------------------------------------------------------------------------------
Input Here(String): ICU_Beds

--------------------------------------------------------------------------------

         Current Value: 3050


Input A New Value: 4239

--------------------------------------------------------------------------------

                Data Updated Successfully !

>>>
```

- We may also check the same in the actual CSV file before and after updating the data through the Python Program.

- Before updating the data value, original CSV data file looks as shown on the next page:



AutoSave Off | Resources | Search

File Home Insert Page Layout Formulas Data Review View Help

Paste | Cut | Copy | Format Painter | Clipboard | Calibri 11 | Font | Wrap Text | Merge & Center | Alignment | General | Number | Conditional Formatting | Format as Table | Normal | Calculation

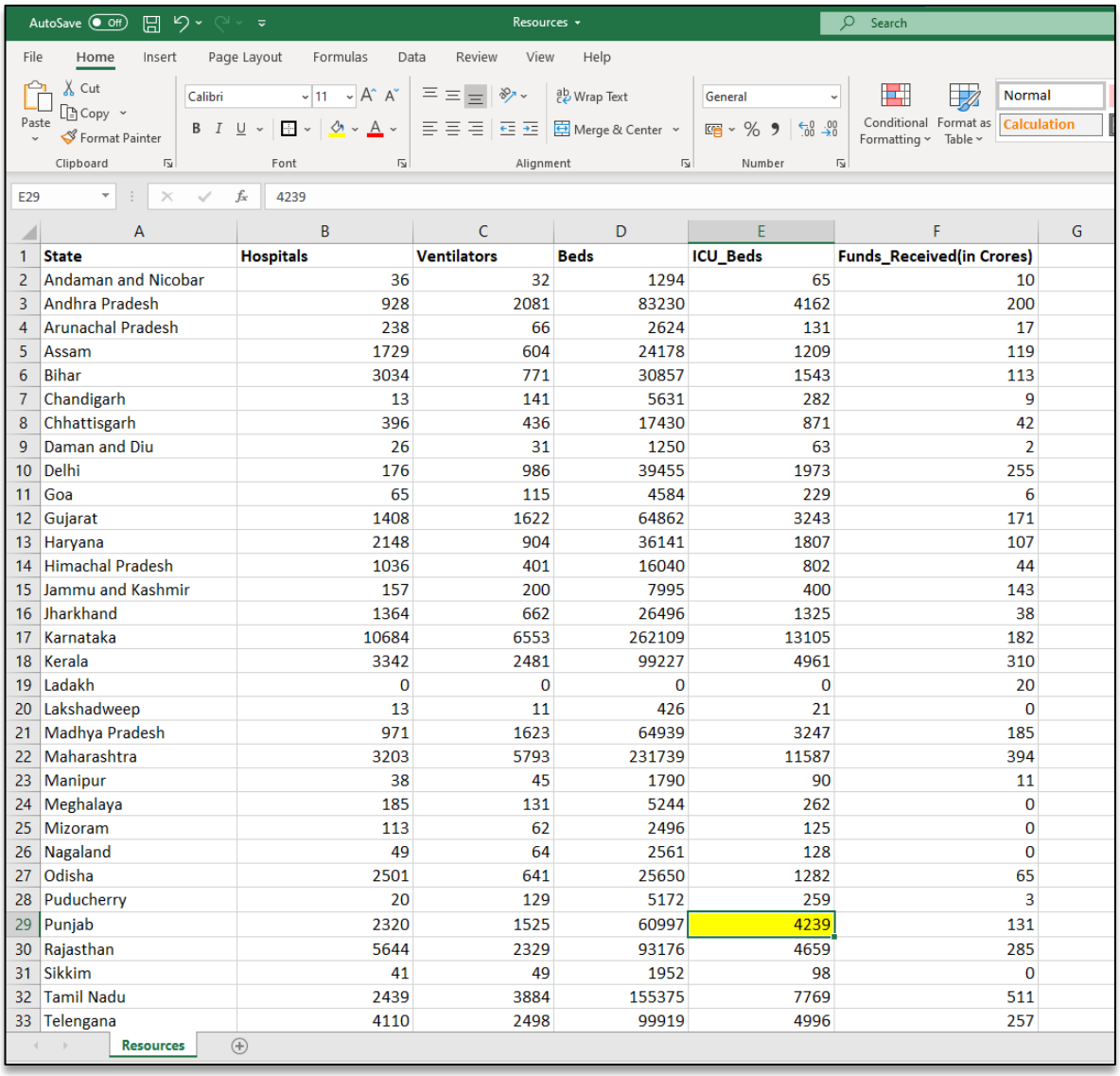
E29 | 3050

	A	B	C	D	E	F	G
1	**State**	**Hospitals**	**Ventilators**	**Beds**	**ICU_Beds**	**Funds_Received(in Crores)**	
2	Andaman and Nicobar	36	32	1294	65	10	
3	Andhra Pradesh	928	2081	83230	4162	200	
4	Arunachal Pradesh	238	66	2624	131	17	
5	Assam	1729	604	24178	1209	119	
6	Bihar	3034	771	30857	1543	113	
7	Chandigarh	13	141	5631	282	9	
8	Chhattisgarh	396	436	17430	871	42	
9	Daman and Diu	26	31	1250	63	2	
10	Delhi	176	986	39455	1973	255	
11	Goa	65	115	4584	229	6	
12	Gujarat	1408	1622	64862	3243	171	
13	Haryana	2148	904	36141	1807	107	
14	Himachal Pradesh	1036	401	16040	802	44	
15	Jammu and Kashmir	157	200	7995	400	143	
16	Jharkhand	1364	662	26496	1325	38	
17	Karnataka	10684	6553	262109	13105	182	
18	Kerala	3342	2481	99227	4961	310	
19	Ladakh	0	0	0	0	20	
20	Lakshadweep	13	11	426	21	0	
21	Madhya Pradesh	971	1623	64939	3247	185	
22	Maharashtra	3203	5793	231739	11587	394	
23	Manipur	38	45	1790	90	11	
24	Meghalaya	185	131	5244	262	0	
25	Mizoram	113	62	2496	125	0	
26	Nagaland	49	64	2561	128	0	
27	Odisha	2501	641	25650	1282	65	
28	Puducherry	20	129	5172	259	3	
29	Punjab	2320	1525	60997	3050	131	
30	Rajasthan	5644	2329	93176	4659	285	
31	Sikkim	41	49	1952	98	0	
32	Tamil Nadu	2439	3884	155375	7769	511	
33	Telengana	4110	2498	99919	4996	257	

Resources

P.T.O

- After updating the data, we can see the following changes in the CSV data file:



	A	B	C	D	E	F	G
1	**State**	**Hospitals**	**Ventilators**	**Beds**	**ICU_Beds**	**Funds_Received(in Crores)**	
2	Andaman and Nicobar	36	32	1294	65	10	
3	Andhra Pradesh	928	2081	83230	4162	200	
4	Arunachal Pradesh	238	66	2624	131	17	
5	Assam	1729	604	24178	1209	119	
6	Bihar	3034	771	30857	1543	113	
7	Chandigarh	13	141	5631	282	9	
8	Chhattisgarh	396	436	17430	871	42	
9	Daman and Diu	26	31	1250	63	2	
10	Delhi	176	986	39455	1973	255	
11	Goa	65	115	4584	229	6	
12	Gujarat	1408	1622	64862	3243	171	
13	Haryana	2148	904	36141	1807	107	
14	Himachal Pradesh	1036	401	16040	802	44	
15	Jammu and Kashmir	157	200	7995	400	143	
16	Jharkhand	1364	662	26496	1325	38	
17	Karnataka	10684	6553	262109	13105	182	
18	Kerala	3342	2481	99227	4961	310	
19	Ladakh	0	0	0	0	20	
20	Lakshadweep	13	11	426	21	0	
21	Madhya Pradesh	971	1623	64939	3247	185	
22	Maharashtra	3203	5793	231739	11587	394	
23	Manipur	38	45	1790	90	11	
24	Meghalaya	185	131	5244	262	0	
25	Mizoram	113	62	2496	125	0	
26	Nagaland	49	64	2561	128	0	
27	Odisha	2501	641	25650	1282	65	
28	Puducherry	20	129	5172	259	3	
29	Punjab	2320	1525	60997	4239	131	
30	Rajasthan	5644	2329	93176	4659	285	
31	Sikkim	41	49	1952	98	0	
32	Tamil Nadu	2439	3884	155375	7769	511	
33	Telengana	4110	2498	99919	4996	257	

References

Websites:

- https://www.mygov.in/covid-19/
- https://data.gov.in/
- https://www.python.org/
- https://www.bigcommerce.com/
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7204193/
- https://www.wef.org/coronavirus