IIITB
AIML Project Elective

Project Report

MANOCHAITANYA ANALYSIS

Code and results attached in the submission mail.

Under the guidance of,

- Prof. Ramesh Kestur, IIITB

.

A Project by,
Fahed Shaikh IMT2019079,
Rohit Oze, IMT2019072.
Daksh Agarwal, IMT2019505.

Table of Contents

1 OVERVIEW . . . 3
2 REQUIRED LIBRARIES . . . 3
3 READING DATA AND EDA . . . 3
4 TEXT PRE-PROCESSING . . . 4
5 CHOOSING DISTRICT . . . 6
6 OUTLIER WARNING MODEL . . . 7
7 RESAMPLING MONTHWISE . . . 8
8 PLOTTING GRAPHS OF TOTAL VISITED PATIENTS . . . 10
9 AUTO CORELATION AND PARTIAL AUTOCORELATION . . . 11
10 TIME SERIES GRAPH . . . 12
11 MAPE CALCULATIONS . . . 12
12 METHODS . . . 13
13 OBSERVATIONS . . . 14
14 FUTURE WORK . . . 14
15 FINAL MAPE VALUES . . . 14
16 PREVIOUSLY (Jan-Jun) . . . 15
17 PART-2 (Aug-Dec) . . . 15
18 LOOPING ALL THE DISTRICTS . . . 16
19 SPATIAL MAPPING . . . 16
20 COORDINATES . . . 17
21 FINAL DATASET . . . 18
22 CORRECTIONS . . . 18
23 SHP FILES . . . 19
24 VISUALIZING PREDICTIONS . . . 19
25 INSTALLING PYKRINGE AND INITIALIZING OUR MODEL . . . 20
26 PERFORMING GEOSPATIAL ANALYSIS WITH PYKRINGE . . . 20
27 INTERPOLATION . . . 21
28 FOLIUM HEATMAP PACKAGE . . . 22
29 HEATMAP INTERPOLATION USING SCIPY . . . 23
30 CONCLUSIONS AND CONTRIBUTIONS . . . 23

1 OVERVIEW

This project is an Analysis for forecasting of number of patient visits to Manochaitanya. Lack of resources like beds, medicines and ventilators etc., is a huge problem in our nation's medical infrastructure. This problem is usually seen in almost every government hospital. The resources cannot be distributed equally as they get over or under the need of requirement and get wasted at many centeres where there is not enough need of them. This analysis of patient visits we have done in this project helps us to-

- Understand the rush at a government hospital.
- Prioritize resource sharing so that no center gets to experience lack or resources.
- Minimize the wastage of resources.
- The time series analysis provides us with a proper graph to understand which month of the year which center has what amount of patient inflow.

2 REQUIRED LIBRARIES

1. **Pandas:** Pandas is a powerful data manipulation and analysis library in Python.
2. **Numpy:** NumPy is a fundamental library for numerical computing in Python.
3. **Matplotlib:** Matplotlib is a widely used plotting library in Python.
4. **os:** The import os statement is used in Python to import the os module, which provides a way to interact with the operating system. The os module provides various functions for performing operating system-related tasks such as file and directory operations, environment variables, process management, and more.
5. **datetime:** The datetime module in Python provides classes for manipulating dates and times. By importing only the datetime class, you can directly use it in your code without having to reference the module name.
6. **warnings:** provides functionality for issuing warnings in Python programs. Warnings are typically used to alert developers about potential issues or deprecated features in their code. By importing the warnings module, you can use its functions and classes to customize the behavior of warnings in your program

3 READING DATA AND EDA

We have uploaded the data file- Prepared Clinical Data, a file we recieved from our Professor, The dataset contains all the details of MnCs, from inpatient count to reason for their visit. The data required a lot of preprocessing, which was done. Here is how the dataset looked initially-

ReportId	StateId	DistrictId	DistrictName	TalukaId	MncHospitalId	MncVisiteDate	ReportingMonthyear	ReportingDate	old_smd_male	old_smd_female	new_smd_male	new_smd_female	old_cmd_male	old_cmd_female	new_cmd_male	new_cmd_female	old_a
21	17	3	Bangalore Urban	298.0	NaN	NaN	2017-04-01	2017-08-09	5	6	1	1	43	39	8	2	
22	17	45	Bbmp	297.0	NaN	NaN	2017-04-01	2017-10-06	0	0	0	0	0	1	0	2	
23	17	45	Bbmp	296.0	NaN	NaN	2017-04-01	2017-10-06	0	0	0	0	0	0	0	0	
24	17	45	Bbmp	295.0	NaN	NaN	2017-04-01	2017-10-06	0	0	0	0	1	0	0	0	
25	17	45	Bbmp	294.0	NaN	NaN	2017-04-01	2017-10-06	0	0	0	0	0	0	0	0	

We have described the data, took a note of the information stored in it,

	Unnamed: 0	ReportId	StateId	DistrictId	TalukaId	MncHospitalId	old_smd_male	old_smd_female	new_smd_male	new_smd_female	...	InPatient_12	FacilityId	IsMnc	total_male	total_femal
count	49335.000000	49335.000000	49335.0	49335.000000	49323.000000	10794.000000	49335.000000	49335.000000	49335.000000	49335.000000	...	49333.000000	38539.000000	49335.000000	49335.000000	49335.00000
mean	24667.000000	29532.193615	17.0	23.836607	142.113679	200.855012	6.387027	5.478869	1.436951	1.264032	...	0.008919	1420.966190	0.270923	39.519165	31.48570
std	14241.932102	21283.428045	0.0	13.076381	90.790445	156.538894	26.229063	20.829407	6.983477	6.144499	...	1.160917	991.173439	0.553331	132.402065	90.93202
min	0.000000	21.000000	17.0	1.000000	0.000000	101.000000	-3.000000	-8.000000	-3.000000	-2.000000	...	-9.000000	0.000000	0.000000	-13.000000	-11.00000
25%	12333.500000	12480.500000	17.0	15.000000	108.000000	151.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	518.000000	0.000000	4.000000	3.00000
50%	24667.000000	26518.000000	17.0	21.000000	147.000000	194.000000	1.000000	1.000000	0.000000	0.000000	...	0.000000	1341.000000	0.000000	10.000000	9.00000
75%	37000.500000	39852.500000	17.0	34.000000	210.000000	241.000000	4.000000	3.000000	1.000000	1.000000	...	0.000000	2499.000000	0.000000	26.000000	23.00000
max	49334.000000	77007.000000	17.0	45.000000	298.000000	4020.000000	1017.000000	941.000000	502.000000	509.000000	...	253.000000	2892.000000	3.000000	11849.000000	4317.00000

and especially, we would be focussing on the column called ***'TotalVisitedPatients'***-

```
data['TotalVisitedPatients'].describe()
```

```
count    49335.000000
mean        73.284828
std        226.122560
min        -20.000000
25%          7.000000
50%         21.000000
75%         50.000000
max      11860.000000
Name: TotalVisitedPatients, dtype: float64
```

We have also taken lowest and highest timestamps recorded in the dataset,

```
pd.Timestamp.min
```

```
Timestamp('1677-09-21 00:12:43.145224193')
```

```
pd.Timestamp.max
```

```
Timestamp('2262-04-11 23:47:16.854775807')
```

We made sure, there are no NULL values in the required information area,

```
data['TotalVisitedPatients'].isnull().sum()
```

```
0
```

```
data.TotalVisitedPatients.isna().any()
```

```
False
```

```
check_for_nan = data['TotalVisitedPatients'].isnull()
print (check_for_nan)
```

```
MncVisiteDate
NaT    False
NaT    False
NaT    False
NaT    False
NaT    False
       ...
NaT    False
NaT    False
NaT    False
NaT    False
NaT    False
Name: TotalVisitedPatients, Length: 49335, dtype: bool
```

4 TEXT PRE-PROCESSING

a) We began by converting the values in the ***'MncVisiteDate'*** column of the DataFrame 'data' to datetime format. the 'errors' parameter is set to 'coerce'. This means that any values that cannot be parsed as datetime will be set to NaT (Not a Time). The 'format' parameter specifies the expected format of the date in the column as $'\%Y - \%m - \%d'$, where $'\%Y'$ represents the year with century, $'\%m'$ represents the month, and $'\%d'$ represents the day.

b) We set the column ***'MncVisiteDate'*** as our index to make our calculations of sampling easier, this gives the index to be timestamp, so that we could resample our data into months, that would give as all the required values of *each month*, instead of individual dates.

```
data.set_index(('MncVisiteDate'), inplace = True)
data.index
```

```
DatetimeIndex(['NaT', 'NaT', 'NaT', 'NaT', 'NaT', 'NaT', 'NaT', 'NaT', 'NaT',
               'NaT',
               ...
               'NaT', 'NaT', 'NaT', 'NaT', 'NaT', 'NaT', 'NaT', 'NaT', 'NaT',
               'NaT'],
              dtype='datetime64[ns]', name='MncVisiteDate', length=49335, freq=None)
```

c) Now, as we can see, there is a lot of unwanted data in our dataset. So we filter out the dataset and create a new dataframe, ***'helpfuldata'***, which contains only the information we will be needing.-

```
helpfuldata = pd.DataFrame()
helpfuldata = data[["TotalVisitedPatients", "DistrictId", "TalukaId", "ReportingMonthyear"]]
helpfuldata
```

	TotalVisitedPatients	DistrictId	TalukaId	ReportingMonthyear
MncVisiteDate				
NaT	139.0	3	298.0	2017-04-01
NaT	6.0	45	297.0	2017-04-01
NaT	0.0	45	296.0	2017-04-01
NaT	3.0	45	295.0	2017-04-01
NaT	0.0	45	294.0	2017-04-01
...	...	...	...	...
NaT	0.0	29	196.0	2020-08-01
NaT	6.0	41	260.0	2020-08-01
NaT	57.0	37	238.0	2020-08-01
NaT	72.0	37	238.0	2020-08-01
NaT	3.0	29	196.0	2020-08-01

49335 rows × 4 columns

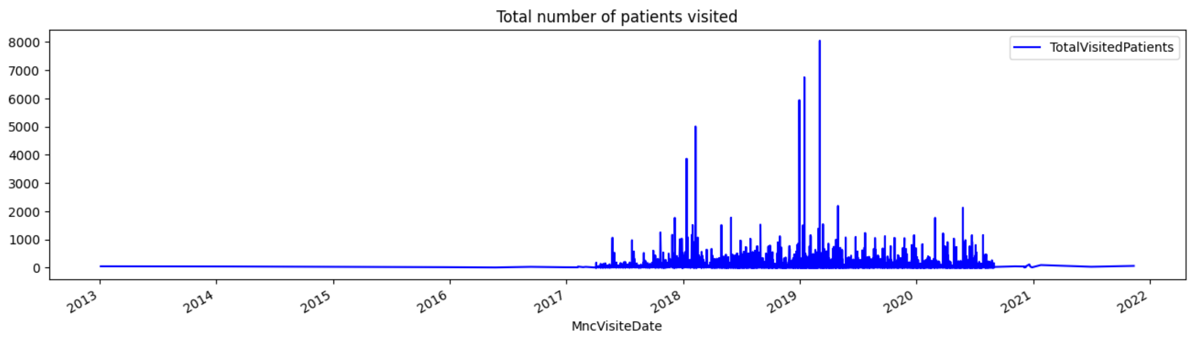
d) We noticed that the ***'TotalVisitedPatients'*** column had float values present in it, so we converted it into int, as the number of patients could never be in decimals and integers would make it easier for our calculations.

```
cols = ['TotalVisitedPatients']
helpfuldata[cols] = helpfuldata[cols].applymap(np.int64)
```

```
helpfuldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 49335 entries, NaT to NaT
Data columns (total 4 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   TotalVisitedPatients  49335 non-null  int64
 1   DistrictId            49335 non-null  int64
 2   TalukaId              49323 non-null  float64
 3   ReportingMonthyear    49335 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(1), int64(2)
memory usage: 1.9 MB
```

If we plot a graph between visit date and total visits, the graph looks something like this-



5 CHOOSING DISTRICT

We have worked by focussing on single particular district, and we choose it before running the code.

```
[ ]  print(helpfuldata['DistrictId'].max())
     print(helpfuldata['DistrictId'].min())

     45
     1
```

```
[ ]  dist = helpfuldata[helpfuldata['DistrictId']==3]
```

We perform a quick little EDA on this new dataset to get some idea on it.

```
[2047] mnc_monthly.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3338 entries, NaT to NaT
Data columns (total 4 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   TotalVisitedPatients  3338 non-null   int64
 1   DistrictId            3338 non-null   int64
 2   TalukaId              3338 non-null   float64
 3   ReportingMonthyear    3338 non-null   datetime64[ns]
dtypes: datetime64[ns](1), float64(1), int64(2)
memory usage: 130.4 KB
```

```
[2048] mnc_monthly.head()
```

MncVisiteDate	TotalVisitedPatients	DistrictId	TalukaId	ReportingMonthyear
NaT	139	3	298.0	2017-04-01
NaT	144	3	114.0	2017-04-01
NaT	238	3	113.0	2017-04-01
NaT	108	3	112.0	2017-04-01
NaT	154	3	115.0	2017-05-01

```
mnc_monthly.describe()
```

	TotalVisitedPatients	DistrictId	TalukaId
count	3338.000000	3338.0	3338.000000
mean	47.289694	3.0	131.040743
std	214.028838	0.0	83.369123
min	-8.000000	3.0	0.000000
25%	11.000000	3.0	113.000000
50%	29.000000	3.0	114.000000
75%	51.000000	3.0	114.000000
max	11860.000000	3.0	298.000000

6 OUTLIER WARNING MODEL

1. ***Percentile :*** The percent of population which lies below that value
2. ***Quantile :*** The cut points dividing the range of probability distribution into continuous intervals with equal probability. There are q-1 of q quantiles one of each k satisfying 0 ¡ k ¡ q
3. ***Quartile :*** Quartile is a special case of quantile, quartiles cut the data set into four equal parts i.e. q=4 for quantiles so we have First quartile Q1, second quartile Q2(Median) and third quartile Q3

Quartile First quartile The first quartile is determined by No of elements $\times(1/4)$. It is the rank in the population (from least to greatest values) at which approximately 1/4 of the values are less than the value of the first quartile.

```
[2184] Q0 = mnc_monthly.TotalVisitedPatients.quantile(0)
Q1 = mnc_monthly.TotalVisitedPatients.quantile(0.25)
Q3 = mnc_monthly.TotalVisitedPatients.quantile(0.75)
IQR = Q3 - Q1
```

```
[2051] print(IQR)
print(Q0)
print(Q1)
print(Q3)
```

```
40.0
-8.0
11.0
51.0
```

```
[2052] min_value = Q0
print(min_value)

max_value = Q3 + 1.5 * IQR
print(max_value)
```

```
-8.0
111.0
```

```
[2053] value ={}
```

```
[2054] if value == 0:
    print('Entering a zero value, confirm if zero is ok')
elif not bool(value):
    # Check if this field is empty
    print('This field can not be empty, please enter a value')
elif (value < min_value):
    print ("The number of patients visited is less than the least number of patients visited in the past. Please confirm")
elif (value > max_value):
    print ("The number of patients is much higher than the number of patients visited in the past. Please confirm")
```

```
This field can not be empty, please enter a value
```

7 RESAMPLING MONTHWISE

```
[2055] mnc_monthly.drop(['DistrictId', 'DistrictId','ReportingMonthyear', 'TalukaId'],axis = 1, inplace = True)
```

```
[2056] mnc_monthly.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3338 entries, NaT to NaT
Data columns (total 1 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   TotalVisitedPatients  3338 non-null   int64
dtypes: int64(1)
memory usage: 52.2 KB
```

```
mnc_monthly
```

MncVisiteDate	TotalVisitedPatients
NaT	139
NaT	144
NaT	238
NaT	108
NaT	154
...	...
NaT	164
NaT	47
NaT	0
NaT	64
NaT	71

3338 rows × 1 columns

Now that we are solely focussing on Total Visited Patients and all the other columns are dropped, we can resample this data in months to get the total visited patients in each month.

```
[2064] mnc_monthly = mnc_monthly.resample('M').sum()
```

```
[2065] mnc_monthly.head(20)
```

MncVisiteDate	TotalVisitedPatients
2017-04-30	163
2017-05-31	131
2017-06-30	179
2017-07-31	210
2017-08-31	322
2017-09-30	371
2017-10-31	313
2017-11-30	413
2017-12-31	379
2018-01-31	405
2018-02-28	421
2018-03-31	413
2018-04-30	597
2018-05-31	677
2018-06-30	1278
2018-07-31	642
2018-08-31	765
2018-09-30	622
2018-10-31	683
2018-11-30	722

Here is some EDA on this newly formed dataset-

```
[2066] mnc_monthly.describe()
```

	TotalVisitedPatients
count	41.000000
mean	562.243902
std	231.687805
min	131.000000
25%	379.000000
50%	642.000000
75%	710.000000
max	1278.000000

Now showing, dates with lowest and highest number of visits to the MnC would be-

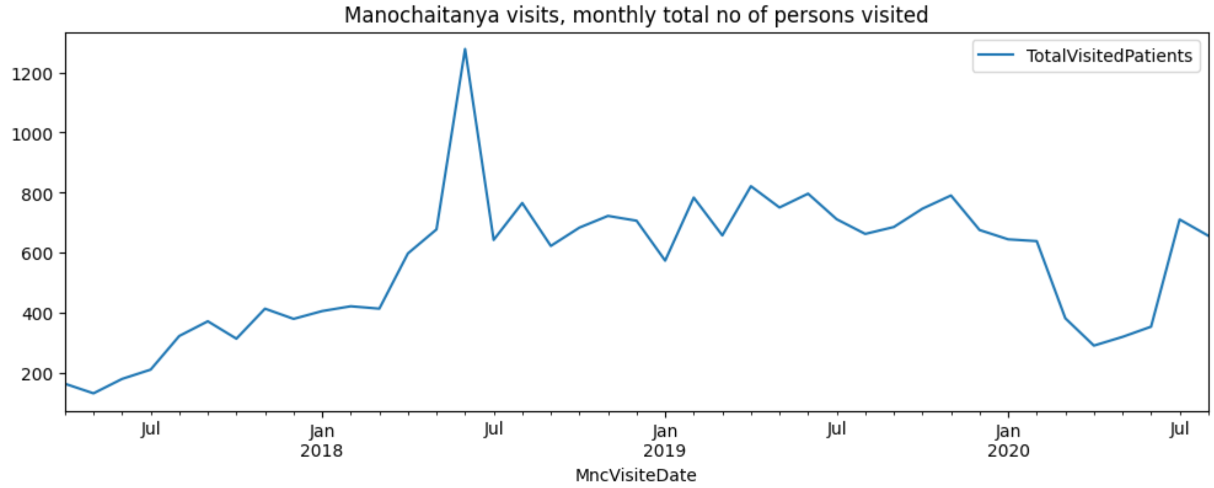
```
[2067] smallest10 = mnc_monthly.nsmallest(10, ['TotalVisitedPatients'])
smallest10
```

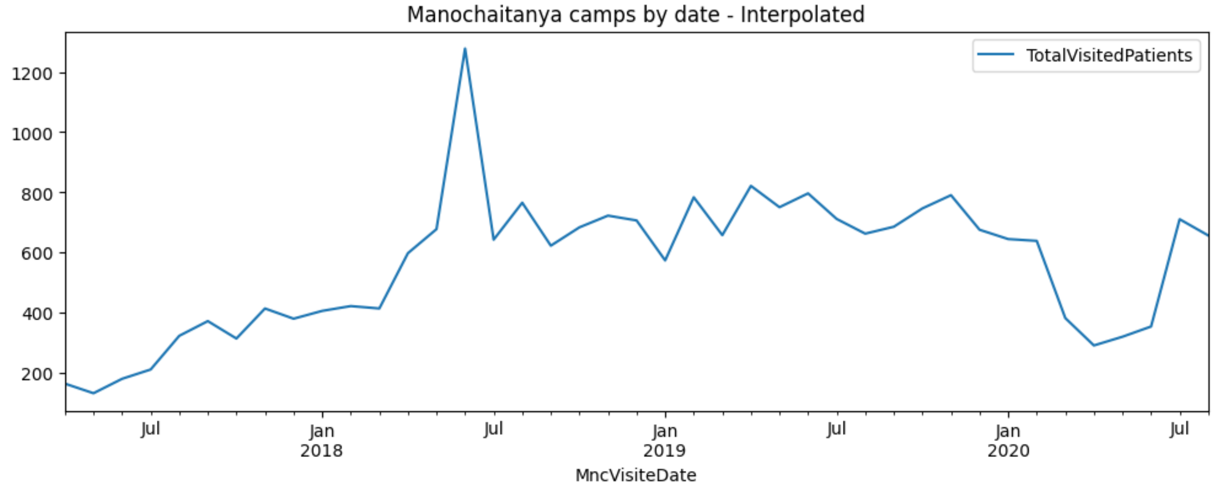
MncVisiteDate	TotalVisitedPatients
2017-05-31	131
2017-04-30	163
2017-06-30	179
2017-07-31	210
2020-04-30	290
2017-10-31	313
2020-05-31	319
2017-08-31	322
2020-06-30	353
2017-09-30	371

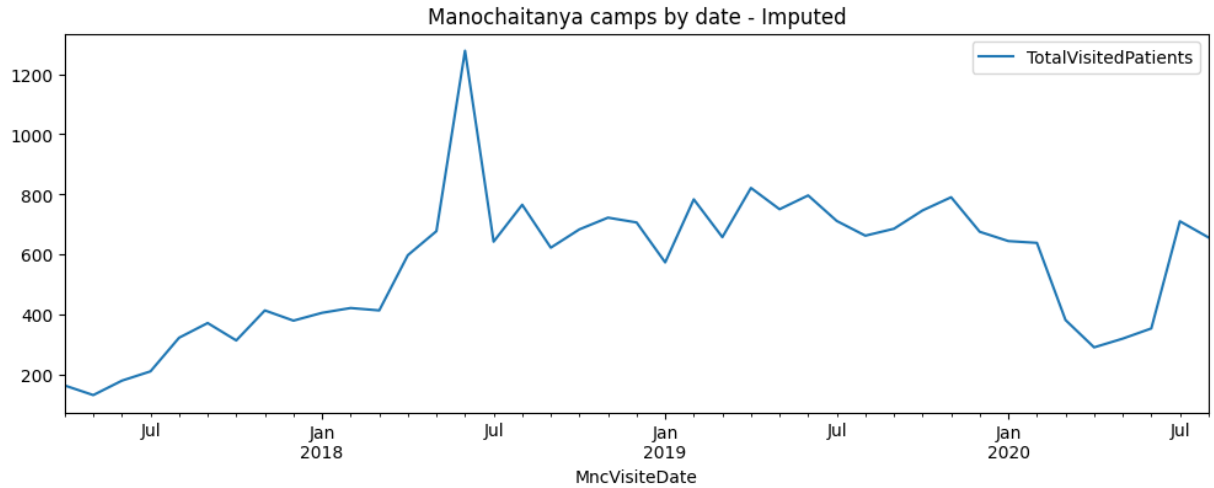
```
largest10 = mnc_monthly.nlargest(10, ['TotalVisitedPatients'])
largest10
```

MncVisiteDate	TotalVisitedPatients
2018-06-30	1278
2019-04-30	821
2019-06-30	796
2019-11-30	790
2019-02-28	783
2018-08-31	765
2019-05-31	750
2019-10-31	746
2018-11-30	722
2019-07-31	711

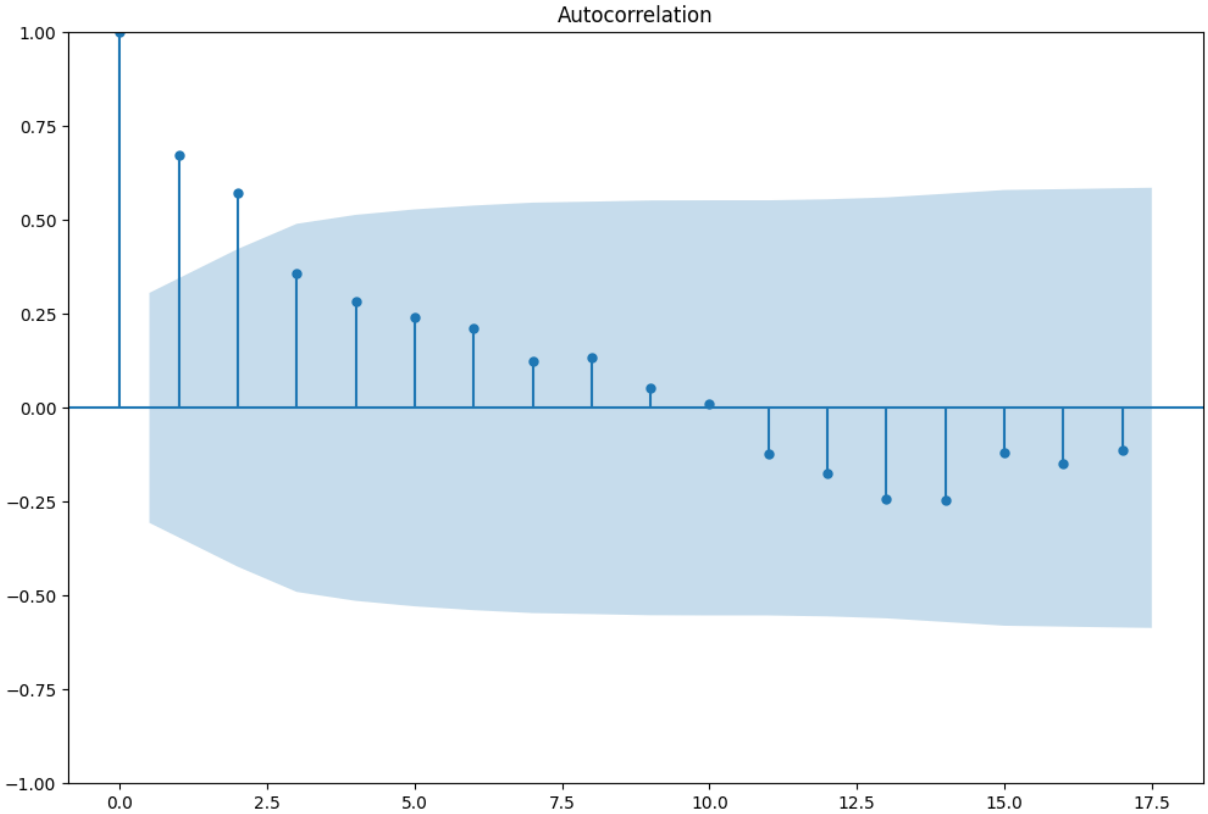
8 PLOTTING GRAPHS OF TOTAL VISITED PATIENTS

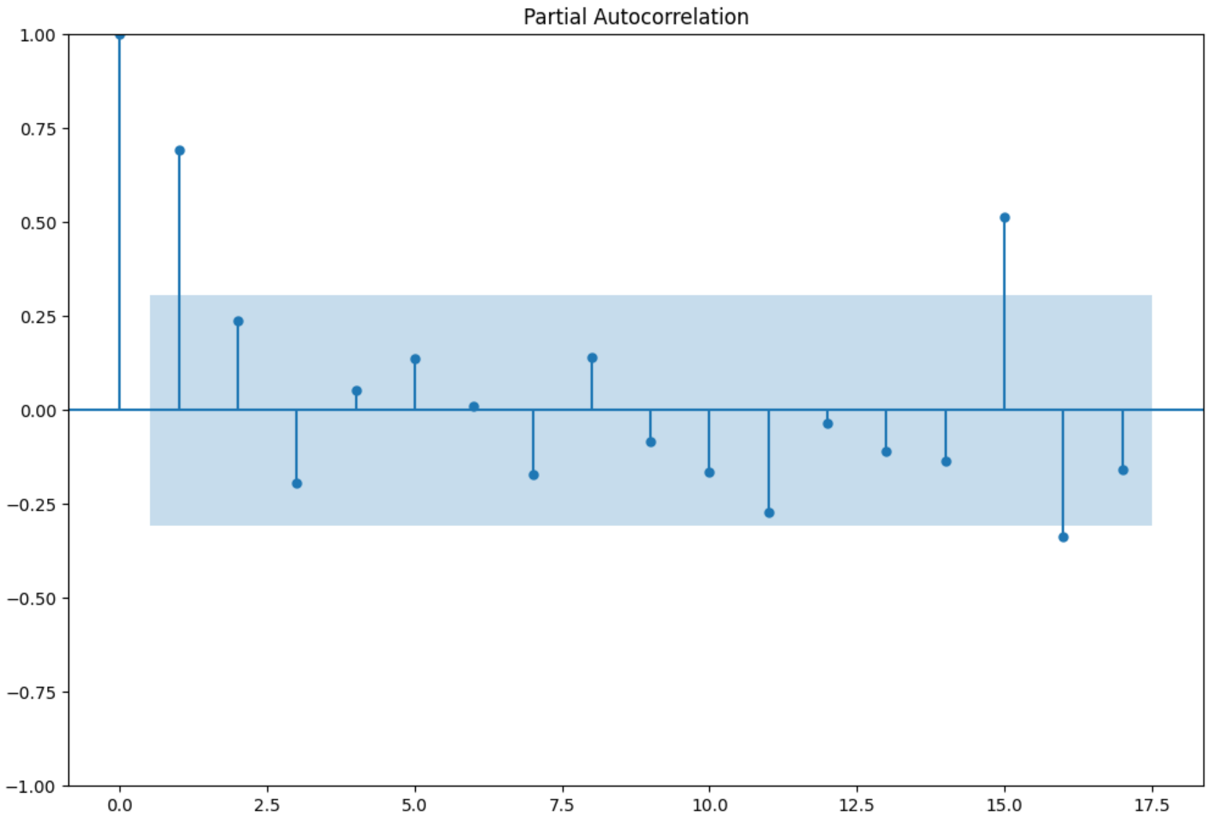




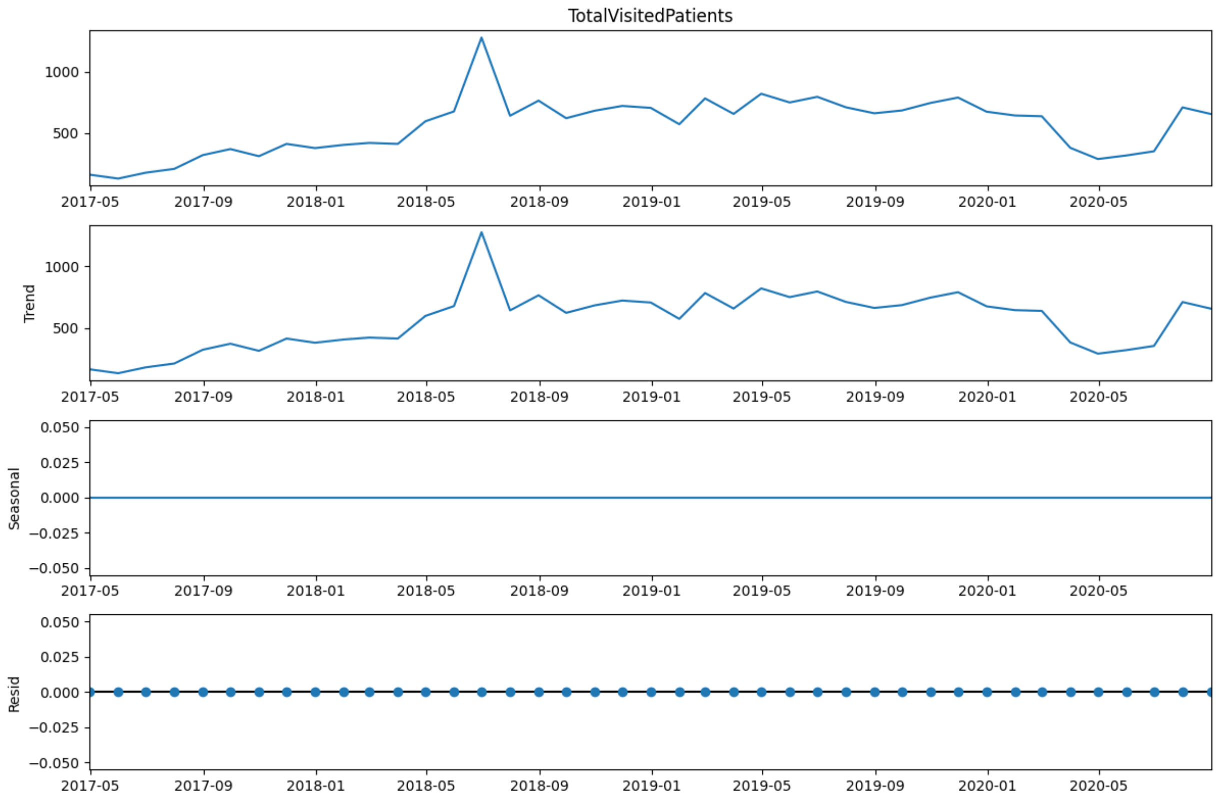


9 AUTO CORELATION AND PARTIAL AUTOCORELATION





10 TIME SERIES GRAPH



Time series of total patient visits to hospitals.
Decomposing the time series: It can be observed that there is a trend but there is no seasonality.

11 MAPE CALCULATIONS

We perform the following steps in each method to calculate the MAPE values-

MAPE CALCULATION METHODS

Method Name

a) Applying the Model

b) Plotting the graph

c) Calculation for MAPE (Mean absolute percentage error)

d) Cumulative Results for MAPE

c) Calculation for MAPE (Mean absolute percentage error)

$$\text{MAPE} = 100N \times \sum i = 1N \;|||\; xi - x^i xi \;|||$$

12 METHODS

1. NAVIE METHOD

[ ] 10 cells hidden

2. SIMPLE AVERAGE

[ ] 10 cells hidden

3. SIMPLE MOVING AVERAGE

[ ] 11 cells hidden

4. SIMPLE EXPONENTIAL SMOOTHING TECHNIQUE

The simplest of the exponentially smoothing methods is naturally called simple exponential smoothing (SES)13. This method is suitable for forecasting data with no clear trend or seasonal pattern.

[ ] 16 cells hidden

5. HOLT METHOD

[ ] 21 cells hidden

6. HOLT WINTERS ADDITIVE METHOD

[ ] 9 cells hidden

7. HOLT WINTERS MULTIPLICATIVE METHOD

[ ] 9 cells hidden

Before executing the regression modules, we had to perform a few calculations and tests. After which, we executed the following Regression Models. Here is a snapshot of the methods applied-

Regression Models

Stationary Test

[ ] 1 cell hidden

Box Cox transformation to make variance constant

[ ] 4 cells hidden

Graph After Box Cox transform

[ ] 3 cells hidden

Adjusting mnc_len

[ ] 1 cell hidden

Install pmdarima

[ ] 6 cells hidden

8. AR

[ ] 10 cells hidden

9. MA

[ ] 9 cells hidden

11. ARMA

[ ] 9 cells hidden

12. ARIMA

[ ] 9 cells hidden

13. SARIMA

[ ] 9 cells hidden

13 OBSERVATIONS

The following code has been applied on all the districts in Karnataka, as given with an ID from 1 to 45. It is to be noted that a few districts yield an error due to some unclean data present in them. They lack some important information, hence we cannot apply this code to them. Meanwhile, in this code, we have selected the important parameter - ***'TotalVisitePatients'***. To the districts giving an error, we can try running the code by changing this parameter to any other parameter like, ***'InPatients'*** etc,.

14 FUTURE WORK

- ***GeoSpatial Analysis-*** A heatmap can be made out of the data we have with us so that we get a clearer idea on the people rush at ManoChaitanya Centers across the state.
- ***Automation-*** We can apply a for loop to all the districts so that we don't have to change the district number every time we run the code.
- ***GridSearch-*** We can make use of GridSearch algorithm to find the appropriate p,d,q values from the autocorrelation and partial autocorrelation plots.
- ***Personalized code for each district-*** We can choose appropriate parameter instead of TotalVistePatients for each district to make the code more efficient.
- ***Plotting confidence intervals***
- ***Implementation of oos for regressive models***
- ***Compute MAPE for oos predictions***

15 FINAL MAPE VALUES

```
Here are the calculated MAPE Values for respective methods for district 3
```

	Method	MAPE
0	Naive method	63.51
0	Simple average method	54.89
0	Simple moving average forecast	65.69
0	Simple exponential smoothing forecast	73.34
0	Holt's exponential smoothing method	82.87
0	Holt Winters' additive method	95.36
0	Holt Winters' multiplicative method	87.62
0	Autoregressive (AR) method	39.83
0	Moving Average (MA) method	35.63
0	Autoregressive moving average (ARMA) method	37.65
0	Autoregressive integrated moving average (ARIM...	39.83
0	Seasonal autoregressive integrated moving aver...	135.44

A similar table for all the possible districts has been tagged in the mail in a zipfile with each image name indicating the district ID and it's table of MAPE values for the particular district's corresponding ID number.

16 PREVIOUSLY (Jan-Jun)

Here are the calculated MAPE Values for respective methods for district 3

	Method	MAPE
0	Naive method	63.51
0	Simple average method	54.89
0	Simple moving average forecast	65.69
0	Simple exponential smoothing forecast	73.34
0	Holt's exponential smoothing method	82.87
0	Holt Winters' additive method	95.36
0	Holt Winters' multiplicative method	87.62
0	Autoregressive (AR) method	39.83
0	Moving Average (MA) method	35.63
0	Autoregressive moving average (ARMA) method	37.65
0	Autoregressive integrated moving average (ARIM...	39.83
0	Seasonal autoregressive integrated moving aver...	135.44

17 PART-2 (Aug-Dec)

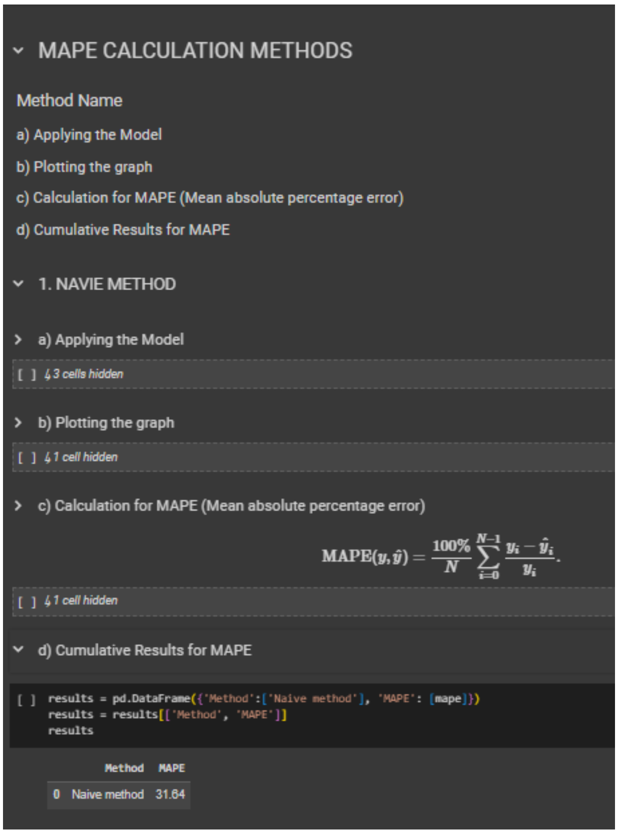
Part-2 Continued by making corrections in the previous work done, utilizing techniques like ***Stationary Test, BoxCox Transformation, Differnecing, Grid Search for PDQ values, Auto Arima etc,.*** to eradicate any sort of abnormalities, errors and non stationary data. After immense enhancements and data transformations, we finally started utilizing Time Series Models after making sure our data fits the stationary requirements. Here are the updated results for the above result shown earlier-

Here are the calculated MAPE Values for respective methods for district 3

	Method	MAPE
0	Naive method	25.82
0	Simple average method	18.63
0	Simple moving average forecast	20.38
0	Simple exponential smoothing forecast	24.68
0	Holt's exponential smoothing method	34.58
0	Holt Winters' additive method	43.94
0	Holt Winters' multiplicative method	28.34
0	Autoregressive (AR) method	24.78
0	Moving Average (MA) method	22.98
0	Autoregressive moving average (ARMA) method	25.00
0	Autoregressive integrated moving average (ARIM...	24.78
0	Seasonal autoregressive integrated moving aver...	51.29

18 LOOPING ALL THE DISTRICTS

A repetitive loop has been deployed which will basically calculate MAPE values for each and every district available in the given dataset.



Post to which, the loop eliminates all the methods which result in an MAPE value greater than 25%. Then, our function automatically figures out the best method which results in the lowest MAPE and uses Out Of Sampling for the upcoming month. With the predicted visit forecast for the upcoming month, we created a list ***"VisitPredictions"***

```
[32] VisitPredictions = {'1': 130.95658641465997, '2': 144.928571, '3': 249.487805, '12': 108.02373081019874, '13': 361.940741, '15': 116.26188046882066, '16': 44.02697495298788, '17': 161.902439, '18': 555.560976, '19': 598.177095,
                        '20': 77.90615999201721, '21': 278.815063, '23': 550.4540110181078, '25': 50.646623596081376, '27': 91.46934607107055, '28': 271.0057244023, '29': 239.107143, '30': 281.650809, '31': 242.14364, '32': 116.785185,
                        '33': 28.616630723877368, '34': 231.609756, '35': 255.77917458907893, '37': 263.6555665393537, '38': 79.52402938542872, '39': 516.527778, '41': 109.28572385427759, '42': 24.544101943043543,
                        '43': 661.8508997079608, '44': 30.333333, '45': 737.137931}
```

19 SPATIAL MAPPING

Copying the list into a new file, ***"Spatial Mapping"***, we utilized the following libraries in order to begin collection of the Geographical Co-Ordinates of our ManoChaitanya centers.

```
# Import the required library
import pandas as pd
from geopy.geocoders import Nominatim
import geopandas as gpd
from geopy.exc import GeocoderTimedOut, GeocoderUnavailable
```

```
# Initialize Nominatim API
geolocator = Nominatim(user_agent="MyApp")
```

20 COORDINATES

Here is a demonstration of the code made to work:-

```
location = geolocator.geocode("Karnataka")
print("The latitude of the location is:  ", location.latitude)
print("The longitude of the location is:  ", location.longitude)
```

Which gives us exact coordinates of the center of state Karnataka. Now with the following code, we grabbed the coordinates of all the locations where we have forecasted the next month predictions.

```
coords = pd.DataFrame(columns=['ID', 'Place','Latitude','Longitude'])

for i in range(len(names)):
    try:
        location = geolocator.geocode(names[i])
        coords.loc[len(coords)]={'ID':ids[i], 'Place': names[i],'Latitude':location.latitude, 'Longitude':location.longitude}
    except:
        continue
```

Finally, we have our data ready with each district connected to it's coordinated location.

	ID	Place	Latitude	Longitude
0	3	Bangalore Urban	13.000000	77.583333
1	45	Bbmp	13.058135	77.506462
2	44	Yadgir	16.767096	77.140398
3	43	Uttara Kannada	14.883333	74.583333
4	37	Raichur	16.083333	77.166667
5	35	Mysore	12.305183	76.655381
6	33	Koppal	15.348414	76.154742
7	30	Haveri	14.787482	75.399673
8	27	Gadag	15.421087	75.654559
9	20	Chitradurga	14.226644	76.400512
10	19	Chikmagalur	13.318014	75.773874
11	18	Chikkaballapur	13.099349	77.388632
12	16	Bijapur	18.793568	80.815939
13	15	Bidar	18.083333	77.333333
14	13	Bellary	15.143395	76.919388
15	12	Belgaum	15.857267	74.506934
16	1	Bagalkote	16.185317	75.696792
17	25	Dharwad	15.454050	75.006652
18	42	Udupi	13.341917	74.747323
19	41	Tumkur	13.340077	77.100621
20	39	Shimoga	13.932609	75.574978
21	38	Ramanagar	25.303693	87.660310
22	34	Mandya	12.523889	76.896198
23	32	Kolar	13.136720	78.133725
24	31	Kodagu	12.251925	75.741215
25	28	Gulbarga	17.166667	77.083333
26	21	Dakshina Kannada	12.932446	74.981313
27	17	Chamrajnagar	11.926994	76.942431
28	2	Bangalore Rural	13.001087	77.336123
29	29	Hassan	13.007082	76.099270
30	23	Davanagere	14.466127	75.920636

Now we need to link the districts with their predictions from the dictionary- ***"VisitPredictions"***. Here is our final dataset on which we will be working on Spatial Analysis.

21 FINAL DATASET

	ID	Place	Latitude	Longitude	VisitPredictions
0	3	Bangalore Urban	13.000000	77.583333	249.487805
1	45	Bbmp	13.058135	77.506462	737.137931
2	44	Yadgir	16.767096	77.140398	30.333333
3	43	Uttara Kannada	14.883333	74.583333	661.850900
4	37	Raichur	16.083333	77.166667	263.655567
5	35	Mysore	12.305183	76.655361	255.779175
6	33	Koppal	15.348414	76.154742	28.616631
7	30	Haveri	14.787482	75.399673	281.650809
8	27	Gadag	15.421087	75.654559	91.469346
9	20	Chitradurga	14.226644	76.400512	77.906160
10	19	Chikmagalur	13.318014	75.773874	598.177095
11	18	Chikkaballapur	13.099349	77.388632	555.560976
12	16	Bijapur	18.793568	80.815939	44.026975
13	15	Bidar	18.083333	77.333333	116.261880
14	13	Bellary	15.143395	76.919388	361.940741
15	12	Belgaum	15.857267	74.506934	108.023731
16	1	Bagalkote	16.185317	75.696792	130.956586
17	25	Dharwad	15.454050	75.006652	50.646624
18	42	Udupi	13.341917	74.747323	24.544102
19	41	Tumkur	13.340077	77.100621	109.285724
20	39	Shimoga	13.932609	75.574978	516.527778
21	38	Ramanagar	25.303693	87.660310	79.524029
22	34	Mandya	12.523889	76.896196	231.609756
23	32	Kolar	13.136720	78.133725	116.785185
24	31	Kodagu	12.251925	75.741215	242.143640
25	28	Gulbarga	17.166667	77.083333	271.005724
26	21	Dakshina Kannada	12.932446	74.981313	278.815063
27	17	Chamrajnagar	11.926994	76.942431	161.902439
28	2	Bangalore Rural	13.001087	77.336123	144.928571
29	29	Hassan	13.007082	76.099270	239.107143
30	23	Davanagere	14.468127	75.920636	550.454011

22 CORRECTIONS

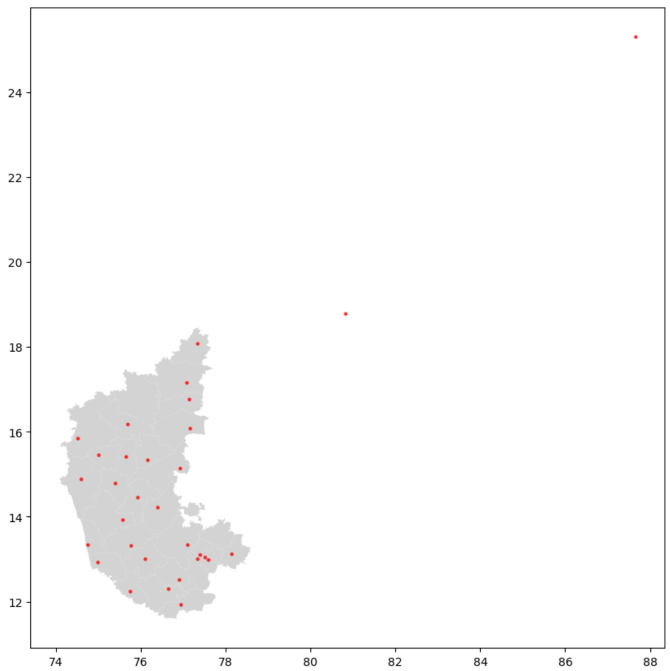


Figure 1: MisCalculated Coordinates

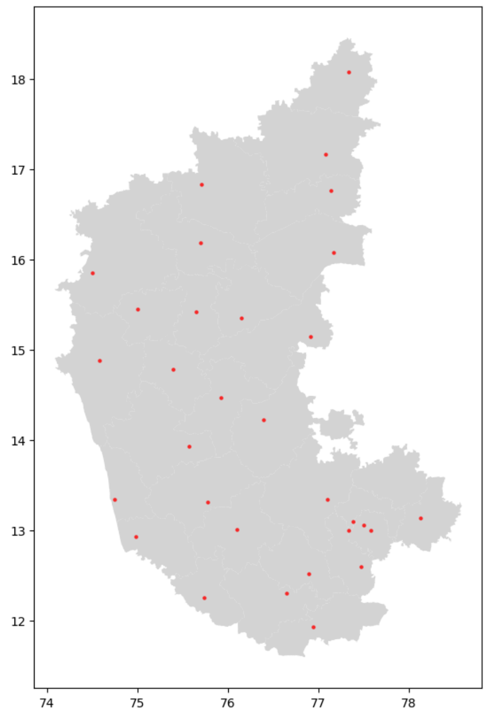


Figure 2: Coordinates after manual Correction

Upon initial plotting, we can notice that a few misreadings have been occurred in the coordinates, so had to sort them out manually. And this was done by the following piece of code.

```
# Update the coordinates for Ramanagara (ID=38) and Bijapur (ID=16)
ramanagara_index = data[data['ID'] == 38].index
data.loc[ramanagara_index, ['Latitude', 'Longitude']] = [12.6003, 77.4702]

bijapur_index = data[data['ID'] == 16].index
data.loc[bijapur_index, ['Latitude', 'Longitude']] = [16.8302, 75.7100]
```

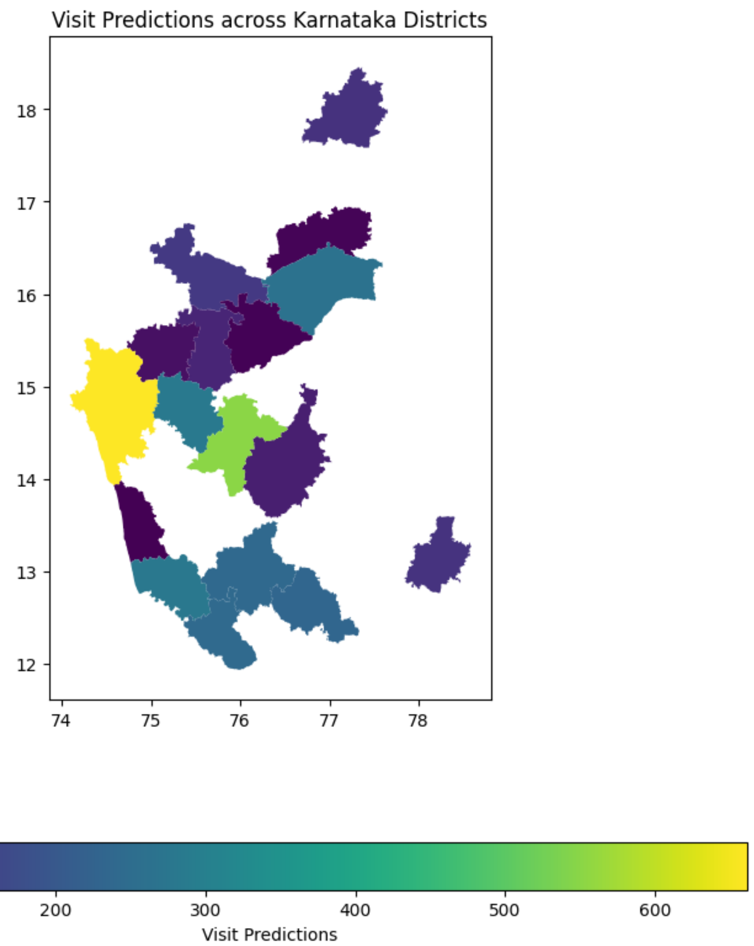
23 SHP FILES

We now downloaded the SHP files from the site https://geographicalanalysis.com/download-karnataka-shapefiles/. After which, we merged the ***SHP File*** with our data table, using the common column for ***District Name***.

```
sample_data
    Karnataka shapefile.cpg
    Karnataka shapefile.dbf
    Karnataka shapefile.prj
    Karnataka shapefile.qmd
    Karnataka shapefile.shp
    Karnataka shapefile.shx
    README.md
```

```
import geopandas as gpd
import matplotlib.pyplot as plt
from scipy.interpolate import griddata
import pandas as pd
import numpy as np

# Load Karnataka State Shapefile
karnataka = gpd.read_file('/content/sample_data/Karnataka shapefile.shp')
print(karnataka.columns)
```

24 VISUALIZING PREDICTIONS



1. This figure shows us that many districts from the state map are missing and do not have the predictions for upcoming month.

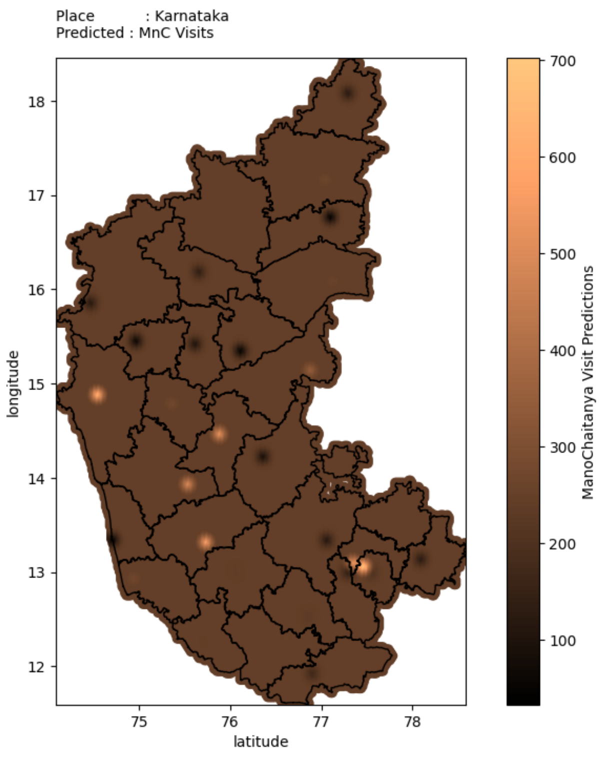
25 INSTALLING PYKRINGE AND INITIALIZING OUR MODEL

```
[20] from pykrige.rk import Krige
     from sklearn.model_selection import GridSearchCV
     import warnings
     param_dict = {
         "method": ["ordinary"],
         "variogram_model": ["linear", "power", "gaussian", "spherical"],
         "nlags": [4, 6, 8, 12],
         "weight": [True, False]
     }

     estimator = GridSearchCV(Krige(), param_dict, verbose=0, return_train_score=True)
     warnings.filterwarnings("ignore", message="n_closest_points will be ignored for UniversalKriging")
     estimator.fit(X=data[['Longitude', 'Latitude']].values, y=data['VisitPredictions'].values)

     if hasattr(estimator, 'best_score_'):
         print('best_score R² = {:.3f}'.format(estimator.best_score_))
         print('best_params = ', estimator.best_params_)
     best_parameters=estimator.best_params_

     best_score R² = -0.725
     best_params =  {'method': 'ordinary', 'nlags': 8, 'variogram_model': 'spherical', 'weight': True}

[ ]  import geopandas as gpd
     from pykrige.ok import OrdinaryKriging
     import numpy as np

     boundary = gpd.read_file("/content/sample_data/Karnataka shapefile.shp")
     min_lon, min_lat, max_lon, max_lat = boundary.total_bounds
     grid_lon = np.linspace(min_lon, max_lon, 300)
     grid_lat = np.linspace(min_lat, max_lat, 300)

     model = OrdinaryKriging(
         data['Longitude'], data['Latitude'],data['VisitPredictions'],
         variogram_model=best_parameters['variogram_model'],
         nlags=best_parameters['nlags'],
         weight=best_parameters['weight'],
         verbose=False)

     z, ss = model.execute('grid', grid_lon, grid_lat)
```

We installed PyKringe and fit our dataset through the estimator, the best ***R-squared score*** we could achieve was ***-0.725***. Which technically implies- a score of -0.725 suggests that the model's predictions are significantly worse than a horizontal line's predictions. It might imply that the chosen model doesn't capture the variance in the data at all and performs very poorly in explaining the variability of the dependent variable around its mean.

A score of -0.725 could indicate severe overfitting or a fundamental issue with the model selection or data quality. It's essential to investigate further, possibly by trying different models, adjusting parameters, or exploring the data to understand why the model performs so poorly. But just to visualize the scale of error we are dealing with, we plotted it onto the graph and it was something as shown.

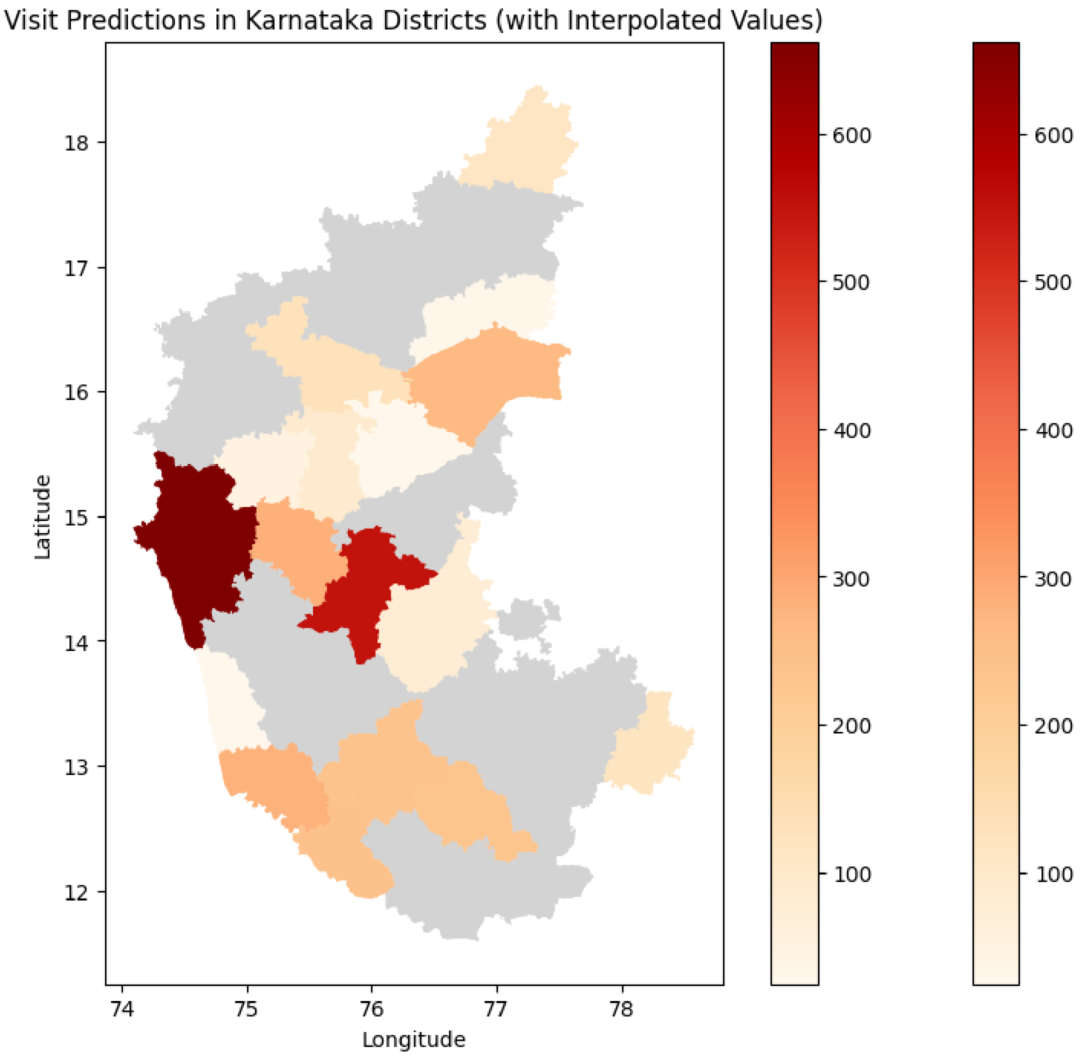
26 PERFORMING GEOSPATIAL ANALYSIS WITH PYKRINGE



27 INTERPOLATION

Since PyKringe model was not optimal for our dataset, we decided to make use of Interpolation to figure out the missing district values. We tried out the following code and it seemed an optimal approach. The result is also followed by the code snippet.

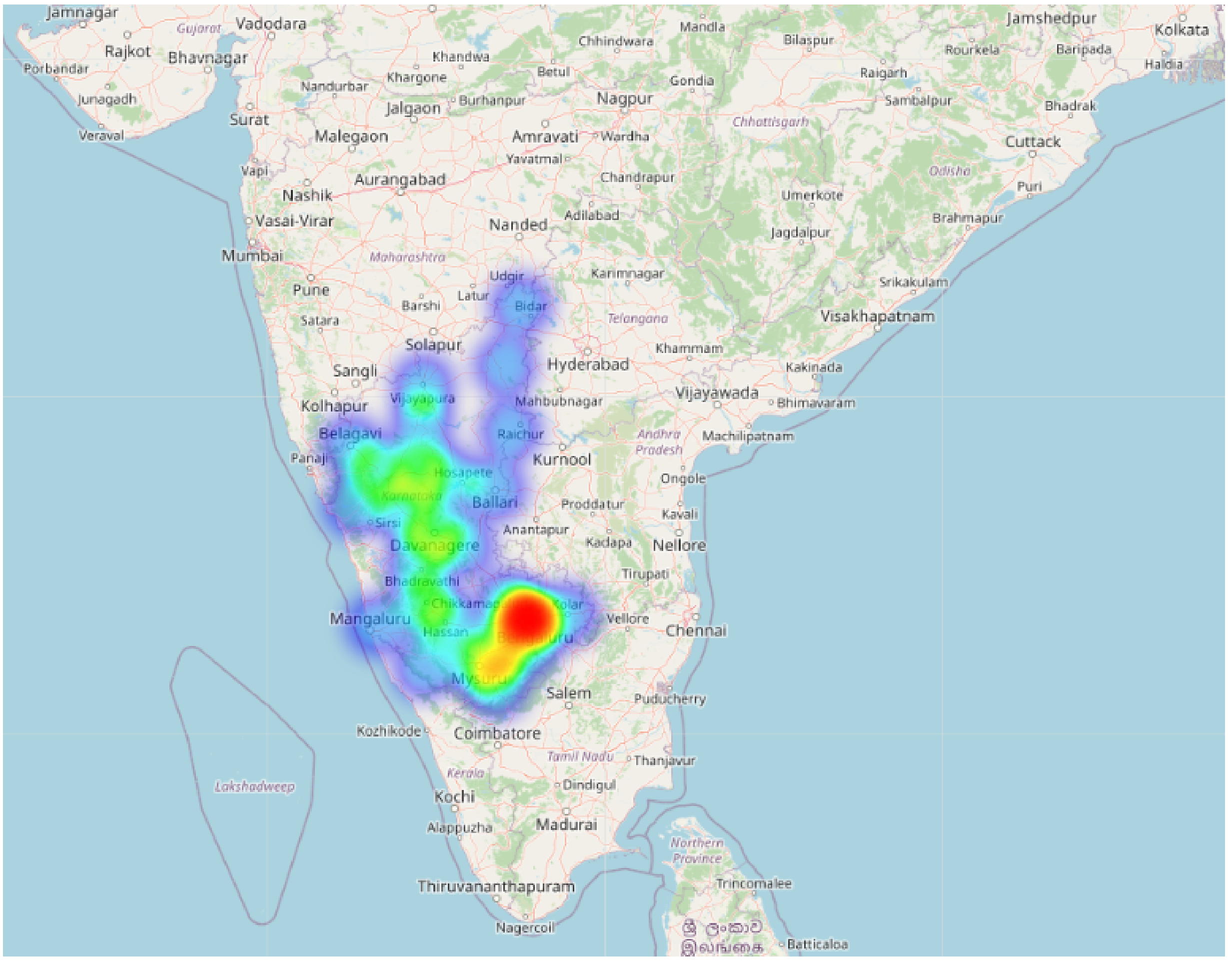
```
# Merge Data with Shapefile
merged_data = karnataka.merge(data, how='left', left_on='Dist_Name', right_on='Place')

# Plotting Visit Predictions on the Karnataka Map
fig, ax = plt.subplots(1, 1, figsize=(10, 8))
merged_data.plot(column='VisitPredictions', ax=ax, legend=True, cmap='OrRd', missing_kwds={'color': 'lightgrey'})
plt.title('Visit Predictions in Karnataka Districts')
plt.xlabel('Longitude')
plt.ylabel('Latitude')

# Interpolate Missing Values (Example using linear interpolation)
missing_data = merged_data[merged_data['VisitPredictions'].isnull()]  # Get missing data rows

# Perform interpolation for missing values (Example using linear interpolation)
if not missing_data.empty:
    points = merged_data[['Latitude', 'Longitude']].dropna()  # Use available lat/long values for interpolation
    values = merged_data['VisitPredictions'].dropna()

    # Interpolation function
    interpolate_predictions = griddata(
        points,
        values,
        (missing_data['Latitude'], missing_data['Longitude']),
        method='linear'
    )

    # Fill missing values in the dataframe
    missing_data['VisitPredictions'] = interpolate_predictions
    merged_data.update(missing_data)

# Plot the updated map with interpolated values
merged_data.plot(column='VisitPredictions', ax=ax, legend=True, cmap='OrRd', missing_kwds={'color': 'lightgrey'})
plt.title('Visit Predictions in Karnataka Districts (with Interpolated Values)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')

plt.show()
```



28 FOLIUM HEATMAP PACKAGE

We studied and researched for further more available interactive packages which would help us visualize the upcoming predictions and we came across this software called ***Folium***, where we built in interactive map for the given coordinates with next month's predictions. A file for the same has been attached in the submission. Here is a glimpse of code and the result.

```python
import folium
from folium.plugins import HeatMap
import pandas as pd

# Create a map centered around Karnataka
map_karnataka = folium.Map(location=[12.9, 76.5], zoom_start=7, tiles='Stamen Terrain')

# Define the grid for interpolation
x = np.array(df['Latitude'])
y = np.array(df['Longitude'])
z = np.array(df['VisitPredictions'])

# Generate a grid to interpolate
xi = np.linspace(min(x), max(x), 100)
yi = np.linspace(min(y), max(y), 100)
xi, yi = np.meshgrid(xi, yi)

# Interpolate missing values
zi = griddata((x, y), z, (xi, yi), method='linear')

# Handle NaN values by replacing them with zeros
zi = np.nan_to_num(zi, nan=0.0)

# Flatten the interpolated grid data
interpolated_data = []
for i in range(len(xi)):
    for j in range(len(yi)):
        interpolated_data.append([xi[i][j], yi[i][j], zi[i][j]])

# Adding HeatMap to the map
HeatMap(interpolated_data).add_to(map_karnataka)

# Save the map as an HTML file
map_karnataka.save('karnataka_heatmap_interpolated.html')
# Display the map
karnataka_map
```

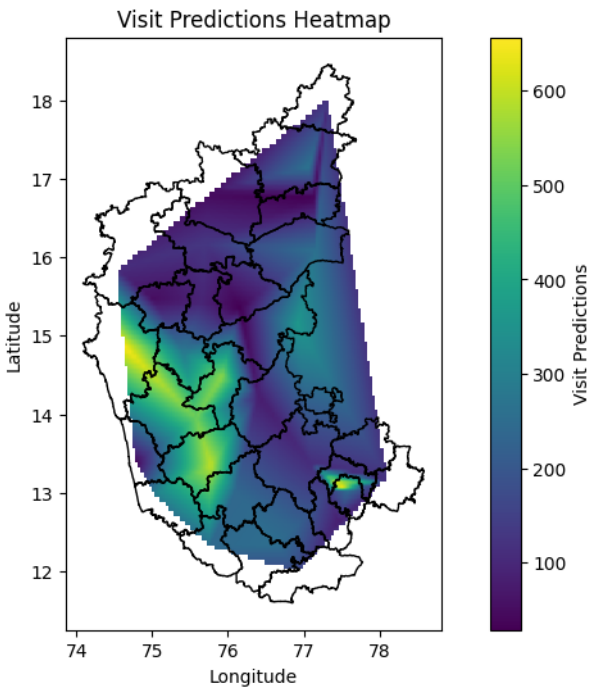


29 HEATMAP INTERPOLATION USING SCIPY

The code then prepares data for interpolation by extracting longitude, latitude, and predicted visit data. It sets up a grid for interpolation using NumPy's meshgrid function and performs the interpolation using SciPy's griddata function, employing linear interpolation.

Matplotlib is utilized to generate the heatmap visualization. It uses *plt.imshow*() to display the interpolated data as a heatmap with a 'viridis' colormap. The *plt.colorbar*() function adds a color bar showing the predicted visit values. The boundaries of the Karnataka shapefile are outlined on the heatmap using *map_df.plot*().

```python
import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import griddata

# Load the shapefile
shapefile_path = '/content/sample_data/Karnataka shapefile.shp'
map_df = gpd.read_file(shapefile_path)

# Interpolate missing data
x = np.array(data['Longitude'])
y = np.array(data['Latitude'])
z = np.array(data['VisitPredictions'])

# Create a grid to interpolate the data onto
x_grid, y_grid = np.meshgrid(np.linspace(x.min(), x.max(), 100), np.linspace(y.min(), y.max(), 100))

z_interp = griddata((x, y), z, (x_grid, y_grid), method='linear')

# Plot the heatmap
plt.figure(figsize=(10, 6))
plt.imshow(z_interp, extent=(x.min(), x.max(), y.min(), y.max()), origin='lower', cmap='viridis')
plt.colorbar(label='Visit Predictions')
map_df.plot(ax=plt.gca(), color='none', edgecolor='black')  # Plot the shapefile boundaries
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Visit Predictions Heatmap')
plt.show()
```



30 CONCLUSIONS AND CONTRIBUTIONS

We would like to Thank Professor ***Ramesh Kestur*** for providing us with an opportunity to work under him for this very intriguing real-life project for a 20 Credit course. This has enhanced our abilities in Time Series Forecasting, GeoSpatial Analysis. We also learnt to deal with highly non-stationary data and many data handling techniques. We faced many challenges and Spatio Temporal analysis was a major one of it. Forecasting values also made us encounter many NaN values in the predictions so had to limit the Out Of Sample forecasting to one month. Fahed contributed in Co-ordinate collection, SHP Files, Folium package Rohit Oze worked on Visualizing the predictions and PyKringe operations. Meanwhile Daksh Aggarwal contributed with the Interpolations and the SciPy heatmaps. While all three of us had to run the predictions for each district particularly to get exact predictions for the upcoming months.