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,
Saurabh Sharma, IMT2019078.

Contents

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
```

MncVisiteDate	TotalVisitedPatients	DistrictId	TalukaId	ReportingMonthyear
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

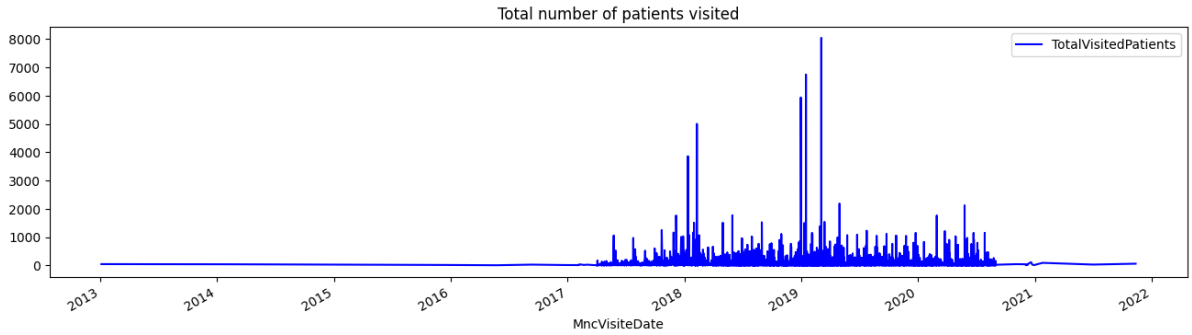
d) We noticed that the ***'TotalVisitedPatients'*** column had float values present in it, so we converted it into int, as the number of patients could never be in decimals and integers would make it easier for our calculations.

```
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.

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

```
print(IQR)
print(Q0)
print(Q1)
print(Q3)
```

```
40.0
-8.0
11.0
51.0
```

```
min_value = Q0
print(min_value)

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

```
-8.0
111.0
```

```
value ={}
```

```
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-

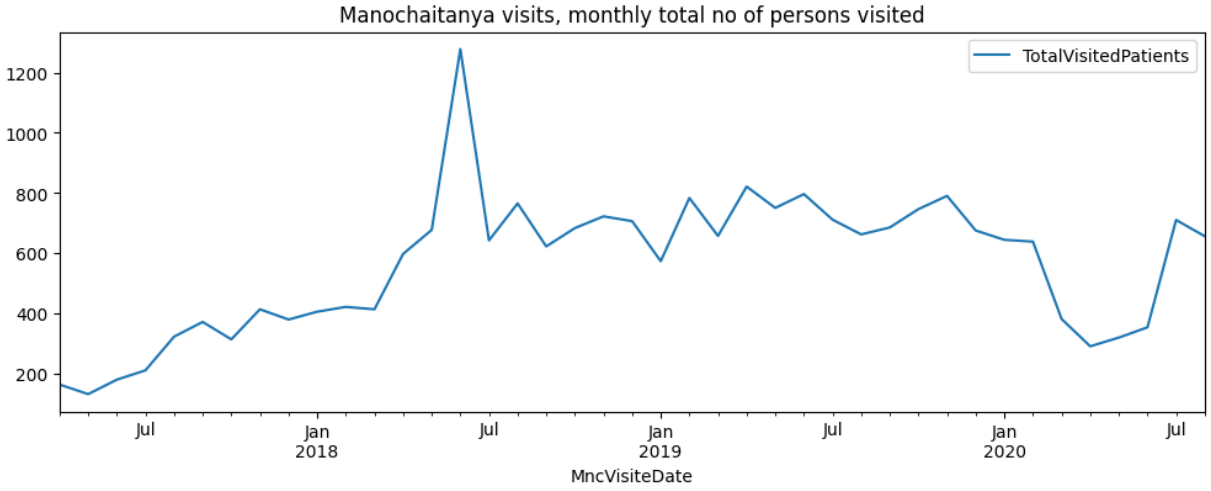
```
[2067] smallest10 = mnc_monthly.nsmallest(10, ['TotalVisitedPatients'])
       smallest10
```

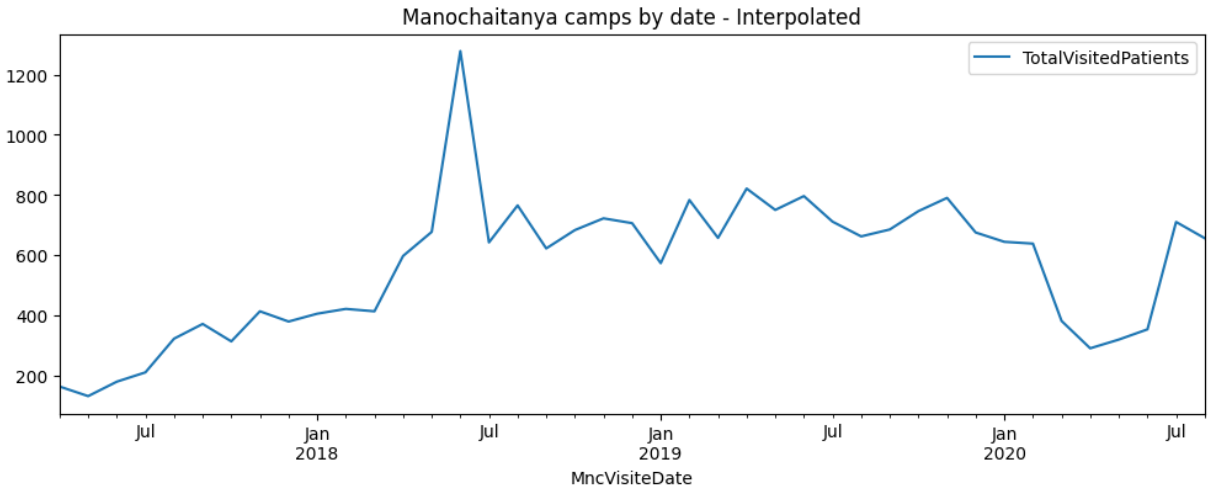
MncVisiteDate	TotalVisitedPatients
2017-05-31	131
2017-04-30	163
2017-06-30	179
2017-07-31	210
2020-04-30	290
2017-10-31	313
2020-05-31	319
2017-08-31	322
2020-06-30	353
2017-09-30	371

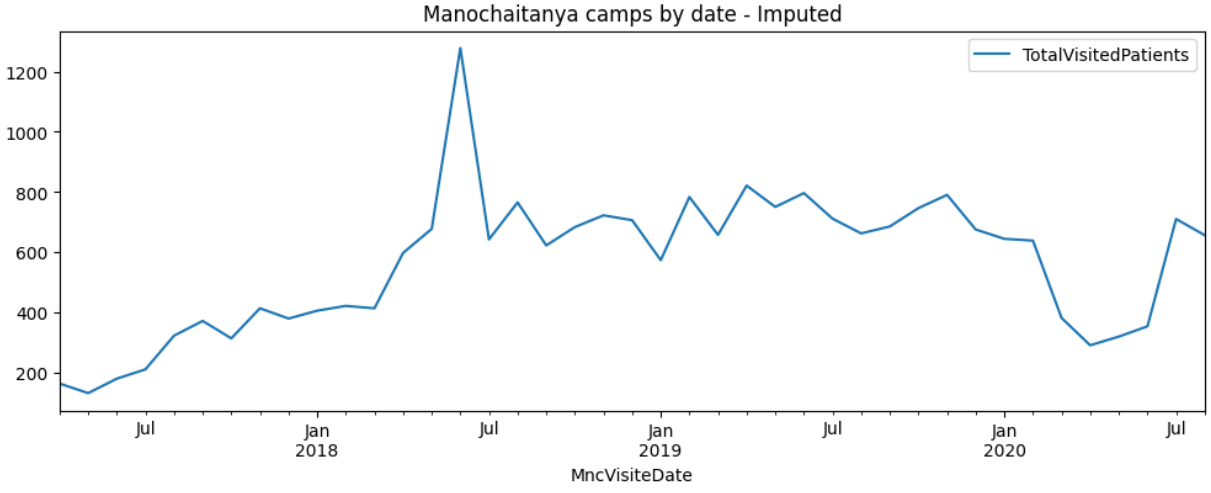
```
largest10 = mnc_monthly.nlargest(10, ['TotalVisitedPatients'])
largest10
```

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

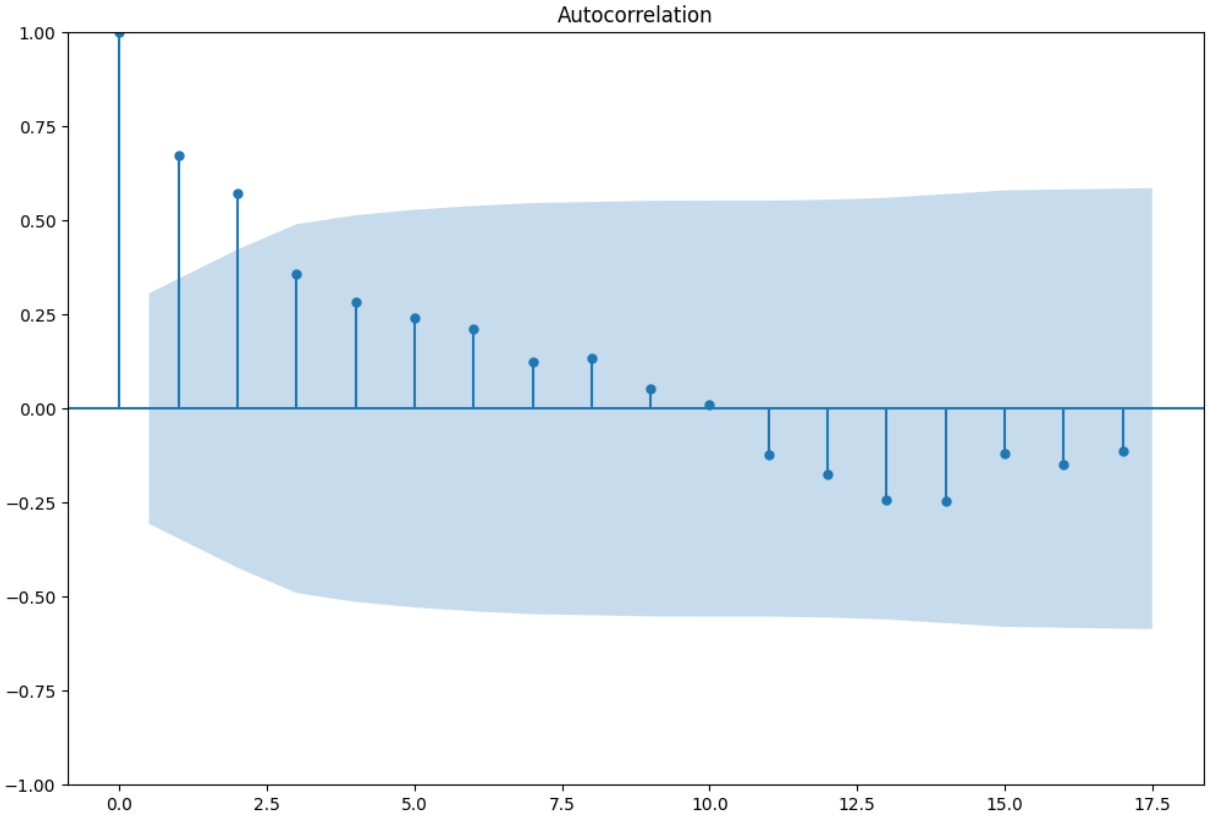
8 PLOTTING GRAPHS OF TOTAL VISITED PATIENTS

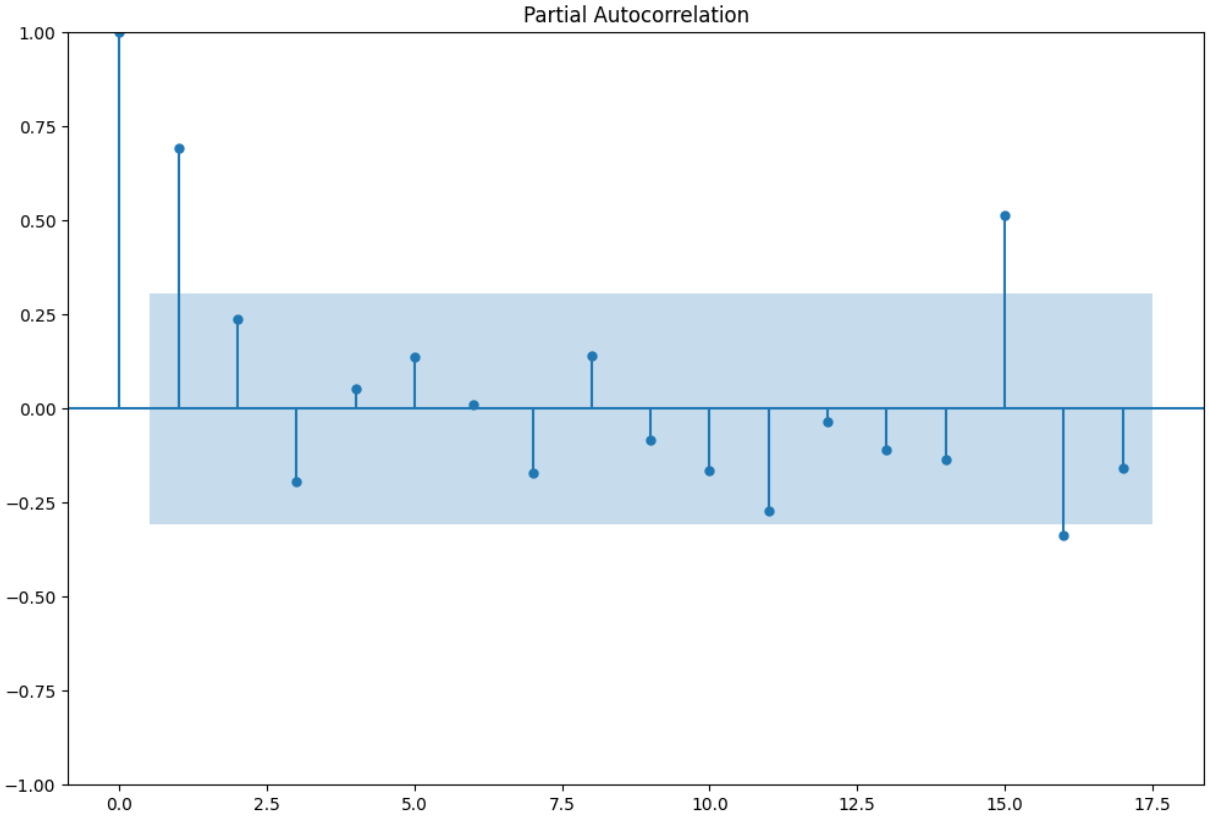




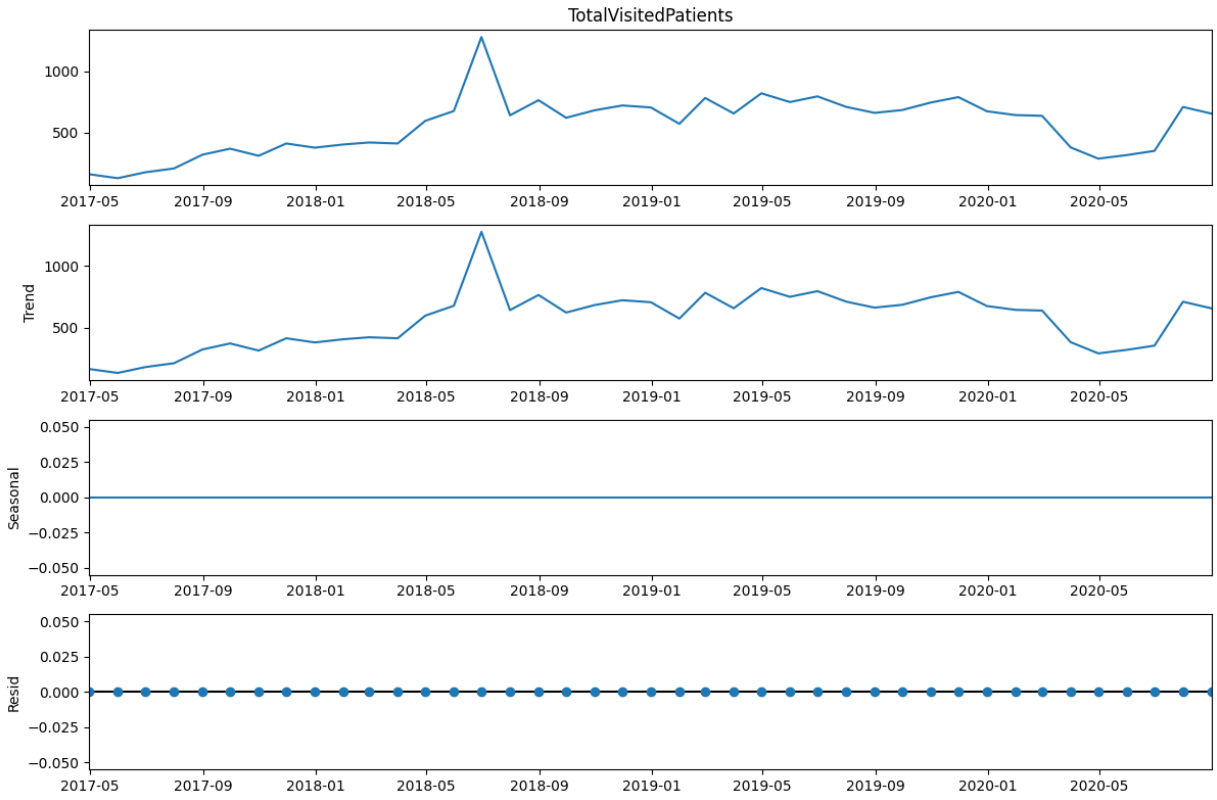


9 AUTO CORELATION AND PARTIAL AUTOCORELATION





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)

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

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 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.

14 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,.

15 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***

16 CONCLUSION

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. This has enhanced our abilities in Time Series Analysis and other new models.