

Deep Learning of Real-time Revenue and Employment in COVID with Data Mining and Machine Learning

Introduction

Covid-19 impact on many businesses has been sudden and severe with forcing to close down. A total of 19.6 total nonfarm U.S jobs have been lost which is twice the total job loss during great recession and an estimation of one quarter in labor force has been disrupted in first three months of lockdown. In the month of May, employment in public sector is increased by 3 million but small business revenue is more affected with high financial risk.

This project uses data from multiple sources and gives mechanisms through which Covid impacts our Economy from changes in consumer activity to business revenue losses to employment.

Throughout the project, it follows real life data science project cycle - Data Analytics life cycle (DALC) with phases like discovering, data preparation, model planning, model building, data visualization, interpreting results, operationalize in iterative manner for better results. Data Preparation includes acquisition, exploratory data analysis with visualization for preprocessing. Selecting models for the data by choose one or more analytical techniques/approaches and then building the model with training and testing data to evaluate the model performance. Interpreting model results using data visualizations to rebuild the model if necessary, to obtain desired results and then interpreting the final data reports or results and then the model/application can be useful for further analytical part or in business operations. Each phase focuses on analytical methods and data mining techniques required to achieve data- driven solutions for better decision-making in an organization as a data Scientist.

Statement of the project problem

This project is studied to understand the total impact of revenue on public sector and employment to mitigate the crisis with deep learning techniques using data mining approaches and machine learning. This study is aimed to,

1. Learn the COVID-19 impacts on private sector revenue in real-time - analyze the total small business revenue from the expenses spent in gradual timely manner across accommodation and food service, Merchant category, arts etc., in several Covid effected areas at national level to understand and make better decisions for new policies across industries.
2. Nonfarm US Employment flow estimation for next month across various industries - estimate the flow of employment in nonfarm public sector with deep insights from job posts across various industries at national level for better employment knowledge helpful for public in recession times.
3. Does lifting up lockdown and public mobility have impact on COVID infection cases? - analyze the nation level google mobility data of people to determine the number of covid infection cases, death records to discover the new patterns from peaks of covid cases helpful for business gains and public welfare.

Review of Literature

The tracker paperwork from Raj, John, Nathaniel, Michael and Opportunity Insights team (2020), the consumer spending from Affinity Solutions, Womble' business revenues and Burning glass job posts have been explained on limited dataset from these above-mentioned applications. This study includes additional research questions with larger dataset from multiple sources to find accurate answers considering the current and previous occupational Employment

statistics across several industries details from U.S Bureau of Labor Statistics that include nationwide records.

Another excellent study from Sara (2020), provides with detailed reports of Great Recession period and recovery impacts focusing on California's public sector jobs and local education. With available larger dataset, this study provides detailed reports of COVID impacts on Nationwide level public sector jobs and revenue.

From a theory paper called 'Vector Autoregressive models for Multivariate series', has given the introduction to VAR and VECM with in depth explanation of the models for multivariate time series estimations.

Another theory paper, 'Autoregression and vector error-correction models' has provided a clear understanding on when to choose VAR or VECM model with proper implementation of causality matrix and cointegration test.

Excellent theoretical study paper on 'Cointegration' by Bent (2019) has provided a clear understanding of cointegration and its important role in model fitting and predicting for time series analysis.

Also, another IMF working paper on theoretical study of 'Testing for cointegration using johansen methodology when variables are near integrated' by Erik. & Par (2007), has provided deep knowledge on properties and working of of Johansen's max eigen values and traces test for cointegration.

A theoretical study paper on 'Vector error correction model' providing a clear understanding of VECM model works in theory.

An article by Aishwarya (2018), given description in handling non-stationary time series with clear explanation in types of non - stationarities and methods to convert non-stationary times series into stationary

Objectives of the study

This project is studied with objectives to find the solutions to mitigate the COVID crisis in Private sector, the three main objectives are:

1. to analyze the total small business revenue from the expenses spent in gradual timely manner across accommodation and food service, Merchant category, arts etc., across national level along with open business percentage to understand and make better decisions for new policies across industries.
2. to estimate the flow of employment with deep insights from private payroll apps across various industries at national level for better employment knowledge helpful for public in recession times.
3. to analyze the nation level google mobility data of people and percent of covid infection cases, death records to discover the new patterns from peaks of covid cases helpful for business gains and public welfare.

Data Collection

The Datasets of in this study are taken from the sources Kaggle. Dataset taken from Kaggle, is created and updated by Opportunity Insight's Economic Tracker in real-time having least lag possible in current data and available data in dataset. These datasets include separate dataset for customer spending across various businesses like accommodation, food service, arts, entertainment, recreation, general merchandize and etc., Revenue from small business with opening businesses percentage datasets across nationwide, public mobility information datasets

in each geo location, COVID test cases data, infection cases data, deaths data across nation.

Employment levels datasets from a private apps in period from January 2020. These datasets are of size 15-32 KB each.

These time series datasets are of multiple .csv files with various attributes which are collected and combined as required for each learning objective using manual feature selection method and then processed to clean the missing values.

Time series dataset with daily granularity from 01-13-2020 to 11-15-2020 to analyze the small business revenues is collected as below

Time series Variable	Definition
Spend_all	Seasonally adjusted credit/debit spending in all merchant category codes smoothed to 7 day moving average
Spend_apg	Seasonally adjusted credit/debit spending in general merchandise smoothed to 7 day moving average
Spend_aer	Seasonally adjusted credit/debit spending in arts, entertainement & recreation smoothed to 7 day moving average
Spend_grf	Seasonally adjusted credit/debit spending in grocery and food smoothed to 7 day moving average
Spend_hcs	Seasonally adjusted credit/debit spending in health care and social assistance smoothed to 7 day moving average
Spend_acf	Seasonally adjusted credit/debit spending in accommodation &food services smoothed to 7 day moving average

Spend_tws	Seasonally adjusted credit/debit spending in transport & warehousing smoothed to 7 day moving average
Spend_retail_no_grocery	Seasonally adjusted credit/debit spending in retail with no grocery smoothed to 7 day moving average
Spend_retail_w_grocery	Seasonally adjusted credit/debit spending in retail with grocery smoothed to 7 day moving average
Merchants_all	Seasonally adjusted and smoothed to 7 day moving average of all small business open percent change
Merchants_ss40	Seasonally adjusted and smoothed to 7 day moving average of small business open percent change in transportation
Merchants_ss60	Seasonally adjusted and smoothed to 7 day moving average of small business open percent change in professional and business services
Merchants_ss65	Seasonally adjusted and smoothed to 7 day moving average of small business open percent change in education and health services
Merchants_ss70	Seasonally adjusted and smoothed to 7 day moving average of small business open percent change in leisure and hospitality
Revenue_all	Seasonally adjusted and smoothed to 7 day moving average of all small business net revenue percent change
Revenue_ss40	Seasonally adjusted and smoothed to 7 day moving average of small business net revenue percent change in transportation

Revenue_ss60	Seasonally adjusted and smoothed to 7 day moving average of small business net revenue percent change in professional and business services
Revenue_ss65	Seasonally adjusted and smoothed to 7 day moving average of small business net revenue percent change in education and health services
Revenue_ss70	Seasonally adjusted and smoothed to 7 day moving average of small business net revenue percent change in leisure and hospitality

Time series dataset with daily granularity from 01-27-2020 to 11-5-2020 to analyze the employment levels with covid-19 cases is collected as below

Time series Variable	Definition
Emp_combined	Seasonally adjusted credit/debit and smoothed to 7 day moving average of employment levels for all workers
Emp_combined_ss40	Seasonally adjusted credit/debit and smoothed to 7 day moving average of employment levels for workers in trade, transport and utilities
Emp_combined_ss60	Seasonally adjusted credit/debit and smoothed to 7 day moving average of employment levels for workers in professional and business services

Emp_combined_ss65	Seasonally adjusted credit/debit and smoothed to 7 day moving average of employment levels for workers in education and health services
Emp_combined_ss70	Seasonally adjusted credit/debit and smoothed to 7 day moving average of employment levels for workers in leisure and hospitality
case_rate	Confirmed covid-19 cases per 1,00,000 people with 7 day moving average
New_case_rate	New confirmed covid-19 cases per 1,00,000 people with 7 day moving average

Time series dataset with daily granularity from 01-27-2020 to 11-5-2020 to analyze the google mobility of people across and covid-19 cases is collected as below

Time series Variable	Definition
Gps_retail_and_recreation	Seasonally adjusted credit/debit and smoothed to 7 day moving average of time spent in retail and recreation
Gps_grocery_and_pharmacy	Seasonally adjusted credit/debit and smoothed to 7 day moving average of time spent in grocery and pharmacy
Gps_parks	Seasonally adjusted credit/debit and smoothed to 7 day moving average of time spent in parks
Gps_transit_stations	Seasonally adjusted credit/debit and smoothed to 7 day moving average of time spent in transit stations

Gps_workplaces	Seasonally adjusted credit/debit and smoothed to 7 day moving average of time spent in workplaces
Gps_residential	time spent in residential locations
Gps_away_from_home	Time spent outside of residential locations
case_rate	Confirmed covid-19 cases per 1,00,000 people with 7 day moving average
New_case_rate	New confirmed covid-19 cases per 1,00,000 people with 7 day moving average

Data Preparation and Cleaning

As all the data sources taken for any kind of analytics can have potential errors or missing values, data cleaning must be done before processing it to avoid anomalies. Data cleaning addresses many issues like miss specifications of the model, false conclusions from incorrect analysis like including errors in parameter estimation, etc.,.

Time series data is special type of data that includes hidden properties like trend, seasonality, cyclical and irregularity in an chronological order. Cleaning such datasets is slightly more complicated and needs more attention.

Time series data is a list of numbers or sequence of numbers taken at equally spaced consecutive points of time. This time series data consists of four elements - i) Trend (T) : Trend defines a long term pattern in the series which can either be positive or negative depending upon increasing or decreasing long term pattern, ii) Seasonality (S) : In a time series, seasonality occurs when it shows regular fluctuations during period or month or quarter in a year, iii) Cyclical (C) : Any sinusoidal or cos-sinusoidal patterns observed over a trend in the series are defined as cyclical

pattern and iv) Irregular (I) : Every time series composes of an unpredictable component called as irregular component which is a random variable and during time series analysis every component except random component is modelled to a point.

In this multivariate time series analysis, three steps are followed for data preparation and processing that included a) understanding the data, b) EDA – Inspection, data profiling, visualizations and c) Data cleaning – missing data, outlier detection and rectifying outliers.

For collecting the required time series variables from multiple datasets for each learning objective as shown in above, first each time series dataset is collected, understood and cleaned the missing values as shown below.

For the first learning objective i.e., to analyze the small business revenues, we used dataset of customer expenditure in various service sectors, dataset of open percentage of small business across various service sectors and also the dataset of revenue generated for small business in various service sectors. Each dataset is uploaded, verified for missing values and rectified them when identified as below.

❖ Collecting and uploading of customer expenditure dataset,

The screenshot shows a Jupyter Notebook interface with two code cells and their outputs.

In [2]:

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

#Reading Customer Expenditure Data from csv file using specified columns
afs_columns = ['freq', 'spend_all_inlow', 'spend_all_inmiddle', 'spend_all_inchigh', 'provisi
file_path = "/Users/harikaporsala/Desktop/Harika's_MS_Courses/INFO_5082/Project/Project_Data/At
afs_data = pd.read_csv(file_path)

afs_data= afs_data.drop(afs_columns, axis=1)

total_nrows = len(afs_data.index)

#printing the results of csv data reading
afs_data.tail(10)
```

Out [2]:

	year	month	day	spend_all	spend_app	spend_aer	spend_grf	spend_hcs	spend_acf	spend_tws	spend_retail_no_grocer
292	2020	10	31	-0.0479	-0.1020	-0.555	0.123	-0.142	-0.300	-0.474	0.089
293	2020	11	1	-0.0624	-0.1050	-0.568	0.118	-0.135	-0.302	-0.477	0.084
294	2020	11	2	-0.0498	-0.1200	-0.536	0.127	-0.105	-0.303	-0.463	0.079
295	2020	11	3	-0.0530	-0.1360	-0.529	0.133	-0.191	-0.303	-0.455	0.063
296	2020	11	4	-0.0615	-0.1500	-0.526	0.116	-0.176	-0.309	-0.457	0.053
297	2020	11	5	-0.0651	-0.1550	-0.531	0.102	-0.180	-0.318	-0.460	0.046
298	2020	11	6	-0.0660	-0.1360	-0.528	0.105	-0.169	-0.317	-0.456	0.039
299	2020	11	7	-0.0598	-0.1230	-0.526	0.100	-0.163	-0.307	-0.458	0.043
300	2020	11	8	-0.0446	-0.1240	-0.517	0.114	-0.152	-0.308	-0.457	0.045
301	2020	11	15	-0.0283	-0.0341	-0.564	0.155	-0.135	-0.306	-0.478	0.133

- ❖ Collecting and uploading of small business open percentage dataset,

In [3]:

```
#Reading Merchants/Business open percentage data from csv file using specified columns
wm_columns = ['merchants_inchigh', 'merchants_inclow', 'merchants_incmiddle']
file_path = "/Users/harikaporalia/Desktop/Harika's_MS_Courses/INFO_5082/Project/Project_Data/Wc"
wm_data = pd.read_csv(file_path,skiprows = [1,2,3],nrows=total_nrows)
wm_data=wm_data.drop(wm_columns,axis=1)

#printing the results of csv data reading
wm_data.head(10)
```

Out[3]:

	year	month	day	merchants_all	merchants_ss40	merchants_ss60	merchants_ss65	merchants_ss70
0	2020	1	13	-0.00689	-0.00576	-0.00921	-0.00734	-0.004830
1	2020	1	14	-0.00854	-0.00749	-0.01020	-0.00873	-0.005540
2	2020	1	15	-0.01010	-0.00955	-0.01070	-0.01010	-0.006760
3	2020	1	16	-0.01110	-0.01050	-0.01100	-0.01100	-0.007620
4	2020	1	17	-0.00864	-0.00804	-0.00910	-0.00864	-0.004850
5	2020	1	18	-0.00513	-0.00438	-0.00600	-0.00669	-0.001330
6	2020	1	19	-0.00395	-0.00335	-0.00761	-0.00484	0.000165
7	2020	1	20	-0.00129	-0.00104	-0.00471	-0.00545	0.002620
8	2020	1	21	-0.00166	-0.00188	-0.00264	-0.00338	0.002100
9	2020	1	22	-0.00340	-0.00279	-0.00601	-0.00370	0.000727

- ❖ Collecting and uploading of small business's revenue dataset,

In [4]:

```
#calling above function to read csv data with a file path
wr_columns = ['revenue_inchigh', 'revenue_inclow', 'revenue_incmiddle']
file_path = "/Users/harikaporalia/Desktop/Harika's_MS_Courses/INFO_5082/Project/Project_Data/Wc"
wr_data = pd.read_csv(file_path,skiprows = [1,2,3],nrows=total_nrows)
wr_data=wr_data.drop(wr_columns,axis=1)

#printing the results of csv data reading
wr_data.head(10)
```

Out[4]:

	year	month	day	revenue_all	revenue_ss40	revenue_ss60	revenue_ss65	revenue_ss70
0	2020	1	13	-0.020100	-0.01530	-0.017200	-0.03430	-0.01930
1	2020	1	14	-0.015100	-0.01190	-0.008520	-0.02730	-0.01290
2	2020	1	15	-0.011600	-0.00846	-0.000735	-0.02420	-0.00866
3	2020	1	16	-0.009310	-0.00955	0.001150	-0.02190	-0.00517
4	2020	1	17	-0.006110	-0.00629	-0.007700	-0.01850	0.00290
5	2020	1	18	-0.003070	-0.00402	-0.006720	-0.01210	0.00494
6	2020	1	19	-0.000157	-0.00644	-0.006160	-0.01370	0.01650
7	2020	1	20	0.003420	-0.00151	0.005200	-0.00882	0.01160
8	2020	1	21	0.005640	-0.00435	0.008550	0.00335	0.01440
9	2020	1	22	-0.001500	-0.00587	-0.004550	-0.00990	0.00684

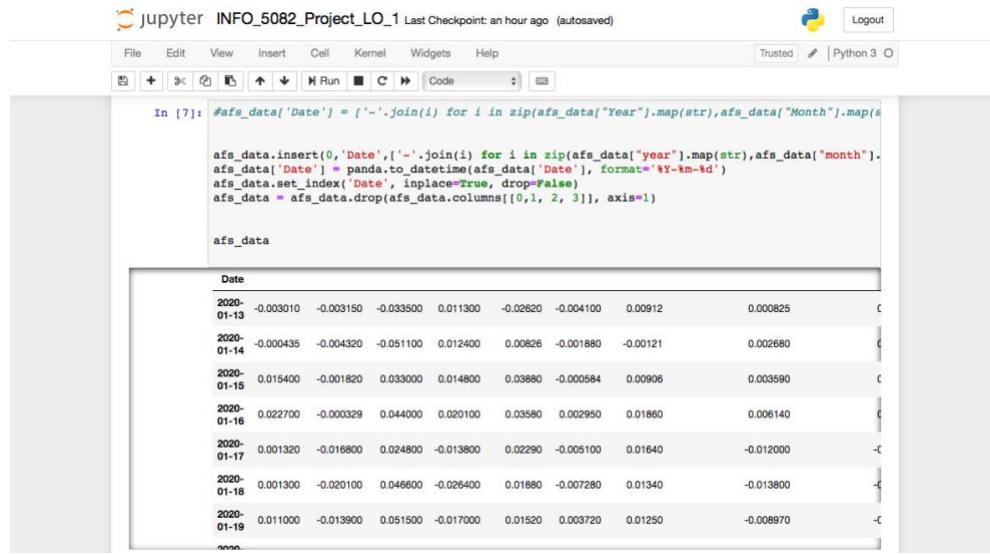
In [5]:

```
#Using functions to verify missing values in the dataset
def verifyMissingValuesInData(fileData):
    print(fileData.isnull().sum())
    return

#Calling above function to verify missing values and print results
```

For easy and better data visualizations three separate columns of Days, Month and Year in dataset are converted into an indexed column of Date with Datetime datatype as below.

- ❖ Datetime indexing for customer expenditure dataset,



In [7]:

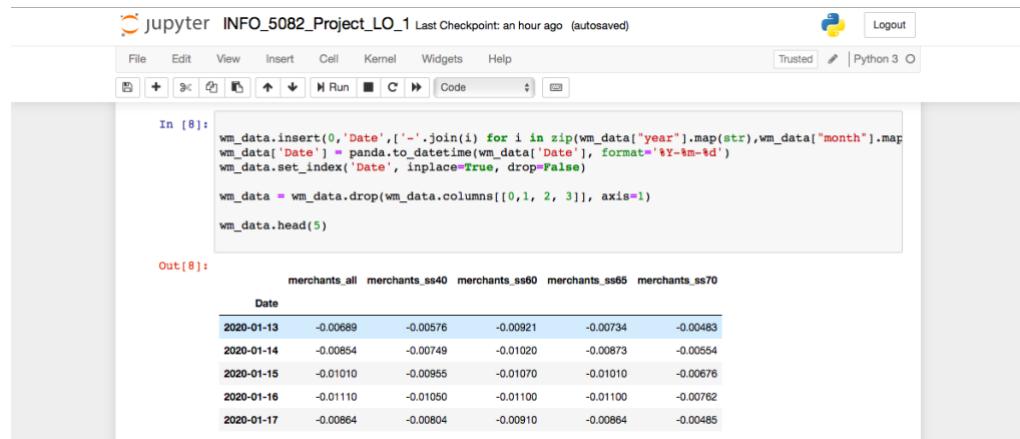
```
#afs_data['Date'] = '-'.join(i) for i in zip(afs_data["Year"].map(str),afs_data["Month"].map(str))
afs_data.insert(0,'Date',[-''.join(i) for i in zip(afs_data["year"].map(str),afs_data["month"].map(str))])
afs_data['Date'] = pd.to_datetime(afs_data['Date'], format='%Y-%m-%d')
afs_data.set_index('Date', inplace=True, drop=False)
afs_data = afs_data.drop(afs_data.columns[[0,1, 2, 3]], axis=1)

afs_data
```

Out[7]:

Date	2020-01-13	-0.003150	-0.033500	0.011300	-0.02620	-0.004100	0.00912	0.000825
2020-01-14	-0.000435	-0.004320	-0.051100	0.012400	0.00826	-0.001880	-0.00121	0.002680
2020-01-15	0.015400	-0.001820	0.033000	0.014800	0.03680	-0.000584	0.00906	0.003590
2020-01-16	0.022700	-0.000329	0.044000	0.020100	0.03680	0.002950	0.01860	0.006140
2020-01-17	0.001320	-0.016800	0.024800	-0.013800	0.02290	-0.005100	0.01640	-0.012000
2020-01-18	0.001300	-0.020100	0.046600	-0.026400	0.01860	-0.007280	0.01340	-0.013800
2020-01-19	0.011000	-0.013900	0.051500	-0.017000	0.01520	0.003720	0.01250	-0.008970
...								

- ❖ Datetime indexing for business open percentage dataset,



In [8]:

```
wm_data.insert(0,'Date',[-''.join(i) for i in zip(wm_data["year"].map(str),wm_data["month"].map(str))])
wm_data['Date'] = pd.to_datetime(wm_data['Date'], format='%Y-%m-%d')
wm_data.set_index('Date', inplace=True, drop=False)

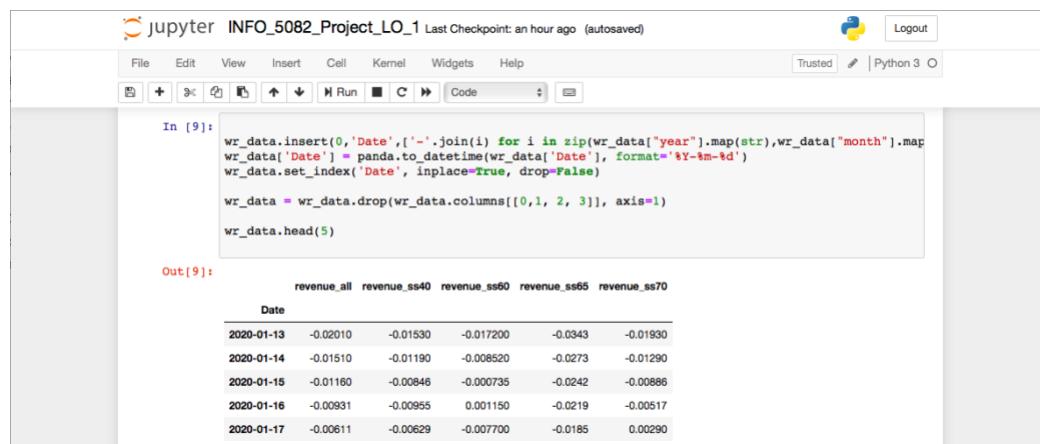
wm_data = wm_data.drop(wm_data.columns[[0,1, 2, 3]], axis=1)

wm_data.head(5)
```

Out[8]:

	merchants_all	merchants_ss40	merchants_ss60	merchants_ss65	merchants_ss70
Date					
2020-01-13	-0.00689	-0.00576	-0.00921	-0.00734	-0.00483
2020-01-14	-0.00854	-0.00749	-0.01020	-0.00873	-0.00554
2020-01-15	-0.01010	-0.00955	-0.01070	-0.01010	-0.00676
2020-01-16	-0.01110	-0.01050	-0.01100	-0.01100	-0.00762
2020-01-17	-0.00864	-0.00804	-0.00910	-0.00864	-0.00485

- ❖ Datetime indexing for business's revenue dataset,



In [9]:

```
wr_data.insert(0,'Date',[-''.join(i) for i in zip(wr_data["year"].map(str),wr_data["month"].map(str))])
wr_data['Date'] = pd.to_datetime(wr_data['Date'], format='%Y-%m-%d')
wr_data.set_index('Date', inplace=True, drop=False)

wr_data = wr_data.drop(wr_data.columns[[0,1, 2, 3]], axis=1)

wr_data.head(5)
```

Out[9]:

	revenue_all	revenue_ss40	revenue_ss60	revenue_ss65	revenue_ss70
Date					
2020-01-13	-0.02010	-0.01530	-0.017200	-0.0343	-0.01930
2020-01-14	-0.01510	-0.01190	-0.008520	-0.0273	-0.01290
2020-01-15	-0.01160	-0.00846	-0.000735	-0.0242	-0.00866
2020-01-16	-0.00931	-0.00955	0.001150	-0.0219	-0.00517
2020-01-17	-0.00611	-0.00629	-0.007700	-0.0185	0.00290

Similarly, for second learning objective i.e., for employment rate estimation, dataset of nationwide employment level rates recorded from 01-27-2020 to 11-05-2020 and the dataset of covid-19 infections rates in same period of time are collected, uploaded and verified for missingvalues to rectify.

- ❖ Collecting and uploading of nationwide employment rates' dataset,

```
In [215]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

#Reading Nationwide employment rate Data from csv file using specified columns
emp_columns = ['emp_combined_inclow', 'emp_combined_incmiddle', 'emp_combined_inchigh', 'emp_c
file_path = "/Users/harikaporalla/Desktop/Harika's_M5_Courses/INFO_5082/Project/Project_Data/Em
emp_data = pd.read_csv(file_path,skiprows = [i for i in range(1,14)])
emp_data= emp_data.drop(emp_columns,axis=1)

#printing the results of csv data reading
total_rows = len(emp_data.index)
emp_data.head(10)
```

Out[215]:

	year	month	day	emp_combined	emp_combined_ss40	emp_combined_ss60	emp_combined_ss65	emp_combined_ss70
0	2020	1	27	0.00518	.00334	.00421	.00762	.00462
1	2020	1	28	0.00542	.00337	.00429	.00809	.00493
2	2020	1	29	0.00563	.00338	.00435	.00854	.00524
3	2020	1	30	0.00582	.00332	.00439	.00897	.00557
4	2020	1	31	0.00600	.00328	.00443	.00937	.00592
5	2020	2	1	0.00617	.00325	.00446	.00974	.00626
6	2020	2	2	0.00633	.00323	.00449	.0101	.00664
7	2020	2	3	0.00651	.00322	.00455	.0105	.00705
8	2020	2	4	0.00671	.00321	.00463	.0109	.00754
9	non	2	5	0.00692	.00323	.00478	.0113	.00803

- ❖ Collecting and uploading of nationwide covid-19 infection rates' dataset,

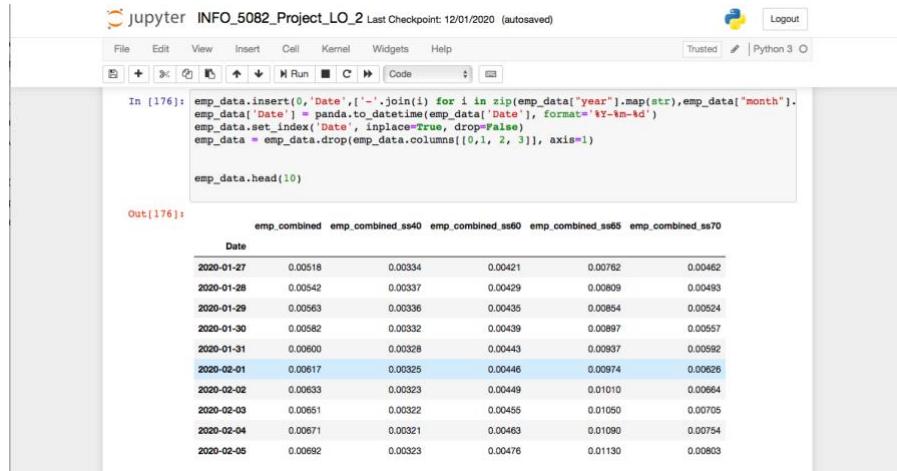
```
In [170]: #Reading Nationwide COVID-19 case rates' Data from csv file using specified columns
cov_case_cols = ['year', 'month', 'day', 'case_rate', 'new_case_rate']
file_path = "/Users/harikaporalla/Desktop/Harika's_M5_Courses/INFO_5082/Project/Project_Data/CC
covid_cases_df = pd.read_csv(file_path, usecols = cov_case_cols,skiprows = [1,2,3,4,5,6],nrc

#printing the results of csv data reading
covid_cases_df.tail(10)
```

Out[170]:

	year	month	day	case_rate	new_case_rate
274	2020	10	27	2675.0	22.0
275	2020	10	28	2700.0	22.8
276	2020	10	29	2727.0	23.5
277	2020	10	30	2757.0	24.1
278	2020	10	31	2783.0	24.4
279	2020	11	1	2805.0	25.0
280	2020	11	2	2833.0	25.8
281	2020	11	3	2861.0	26.6
282	2020	11	4	2894.0	27.7
283	2020	11	5	2931.0	29.1

- ❖ Datetime indexing for nationwide employment rates' dataset,



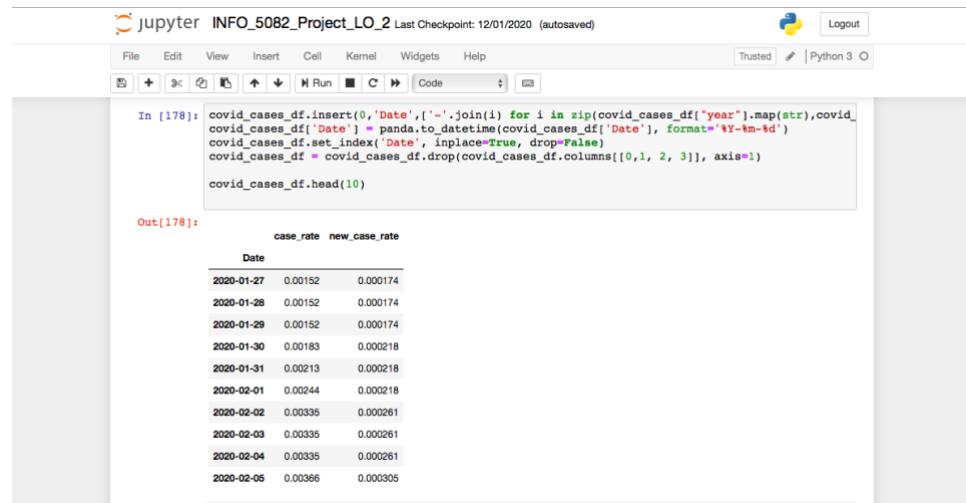
The screenshot shows a Jupyter Notebook interface with the title "jupyter INFO_5082_Project_LO_2 Last Checkpoint: 12/01/2020 (autosaved)". The code cell (In [176]) contains Python code to insert a 'Date' column into the 'emp_data' DataFrame by joining 'year' and 'month' columns, then setting it as the index. The output cell (Out[176]) displays the first 10 rows of the DataFrame, which includes columns for Date, emp_combined, emp_combined_ss40, emp_combined_ss60, emp_combined_ss65, and emp_combined_ss70.

```
In [176]: emp_data.insert(0,'Date','-' .join(i) for i in zip(emp_data['year'].map(str),emp_data['month']))
emp_data['Date'] = pd.to_datetime(emp_data['Date'], format='%Y-%m-%d')
emp_data.set_index('Date', inplace=True, drop=False)
emp_data = emp_data.drop(emp_data.columns[[0,1, 2, 3]], axis=1)

emp_data.head(10)
```

Date	emp_combined	emp_combined_ss40	emp_combined_ss60	emp_combined_ss65	emp_combined_ss70
2020-01-27	0.00518	0.00334	0.00421	0.00762	0.00462
2020-01-28	0.00542	0.00337	0.00429	0.00809	0.00493
2020-01-29	0.00563	0.00336	0.00435	0.00854	0.00524
2020-01-30	0.00582	0.00332	0.00439	0.00897	0.00557
2020-01-31	0.00600	0.00328	0.00443	0.00937	0.00592
2020-02-01	0.00617	0.00325	0.00446	0.00974	0.00626
2020-02-02	0.00633	0.00323	0.00449	0.01010	0.00664
2020-02-03	0.00651	0.00322	0.00455	0.01050	0.00705
2020-02-04	0.00671	0.00321	0.00463	0.01090	0.00754
2020-02-05	0.00692	0.00323	0.00476	0.01130	0.00803

- ❖ Datetime indexing for nationwide Covid-19 infection rates' dataset,



The screenshot shows a Jupyter Notebook interface with the title "jupyter INFO_5082_Project_LO_2 Last Checkpoint: 12/01/2020 (autosaved)". The code cell (In [178]) contains Python code to insert a 'Date' column into the 'covid_cases_df' DataFrame by joining 'year' and 'month' columns, then setting it as the index. The output cell (Out[178]) displays the first 10 rows of the DataFrame, which includes columns for Date, case_rate, and new_case_rate.

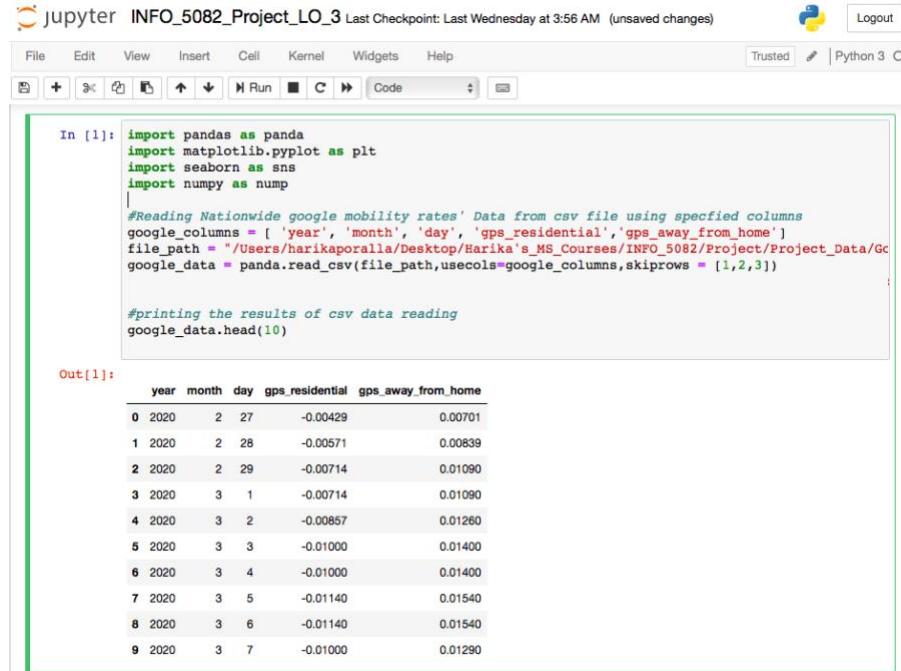
```
In [178]: covid_cases_df.insert(0,'Date','-' .join(i) for i in zip(covid_cases_df['year'].map(str),covid_cases_df['month']))
covid_cases_df['Date'] = pd.to_datetime(covid_cases_df['Date'], format='%Y-%m-%d')
covid_cases_df.set_index('Date', inplace=True, drop=False)
covid_cases_df = covid_cases_df.drop(covid_cases_df.columns[[0,1, 2, 3]], axis=1)

covid_cases_df.head(10)
```

Date	case_rate	new_case_rate
2020-01-27	0.00152	0.000174
2020-01-28	0.00152	0.000174
2020-01-29	0.00152	0.000174
2020-01-30	0.00183	0.000218
2020-01-31	0.00213	0.000218
2020-02-01	0.00244	0.000218
2020-02-02	0.00335	0.000261
2020-02-03	0.00335	0.000261
2020-02-04	0.00335	0.000261
2020-02-05	0.00366	0.000305

Similarly, for third learning objective i.e., to analyze the people's mobility rate with covid-19 case rate, dataset of nationwide google mobility rate from GPS values recorded from 02-27-2020 to 10-20-2020 and the dataset of covid-19 infections rates in same period of time are collected, uploaded and verified for missing values to rectify.

- ❖ Collecting and uploading of nationwide google mobility rates' dataset,



In [1]:

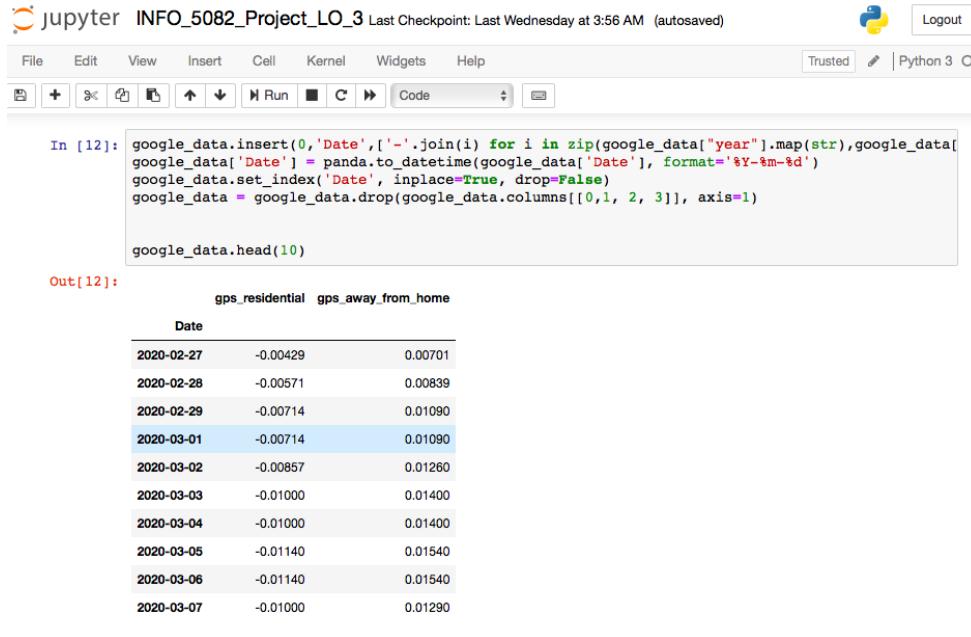
```
import pandas as panda
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as numnp
|
#Reading Nationwide google mobility rates' Data from csv file using specified columns
google_columns = ['year', 'month', 'day', 'gps_residential','gps_away_from_home']
file_path = "/Users/harikaporalia/Desktop/Harika's_MS_Courses/INFO_5082/Project/Project_Data/Gc"
google_data = panda.read_csv(file_path,usecols=google_columns,skiprows = [1,2,3])

#printing the results of csv data reading
google_data.head(10)
```

Out[1]:

	year	month	day	gps_residential	gps_away_from_home
0	2020	2	27	-0.00429	0.00701
1	2020	2	28	-0.00571	0.00839
2	2020	2	29	-0.00714	0.01090
3	2020	3	1	-0.00714	0.01090
4	2020	3	2	-0.00857	0.01260
5	2020	3	3	-0.01000	0.01400
6	2020	3	4	-0.01000	0.01400
7	2020	3	5	-0.01140	0.01540
8	2020	3	6	-0.01140	0.01540
9	2020	3	7	-0.01000	0.01290

- ❖ Datetime indexing for nationwide google mobility rates' dataset,



In [12]:

```
google_data.insert(0,'Date',['-'.join(i) for i in zip(google_data['year'].map(str),google_data['month'].map(str),google_data['day'].map(str))])
google_data['Date'] = panda.to_datetime(google_data['Date'], format='%Y-%m-%d')
google_data.set_index('Date', inplace=True, drop=False)
google_data = google_data.drop(google_data.columns[[0,1, 2, 3]], axis=1)

google_data.head(10)
```

Out[12]:

Date	gps_residential	gps_away_from_home
2020-02-27	-0.00429	0.00701
2020-02-28	-0.00571	0.00839
2020-02-29	-0.00714	0.01090
2020-03-01	-0.00714	0.01090
2020-03-02	-0.00857	0.01260
2020-03-03	-0.01000	0.01400
2020-03-04	-0.01000	0.01400
2020-03-05	-0.01140	0.01540
2020-03-06	-0.01140	0.01540
2020-03-07	-0.01000	0.01290

Nationwide covid-19 infection rates' dataset is also uploaded and indexed by Date column with Datetime date type.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) for analysis is an approach to gain maximum of insight into data, understand the uncovering structure of data, detect outliers and anomalies in the data and determine optimal factor settings for developing parsimonious models by using few quantitative techniques and mostly graphical structures for visualizing the hidden patterns in the dataset.

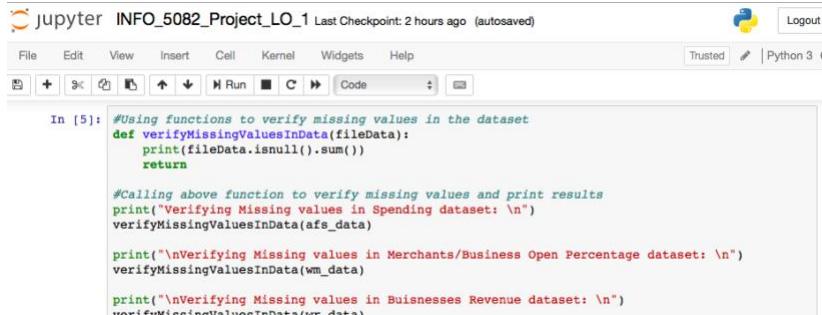
EDA helps in ensuring the results of the data analysis performed are the valid and applicable to given business problems. EDA Techniques include clustering, dimensional reduction in case of high-dimensional data, statistical summary of univariate, bivariate and multivariate visualizations for mapping and better understanding of variables' relationships to provide visual and numeric summaries of the data.

As mentioned in above, in this time series analysis, each dataset collected for all the three objectives are inspected first individually to verify and rectify the missing values and described to get insights for designing optimal model. After visualizing the summary of the dataset cleaning and preprocessing methods are applied as required to each dataset of all the objectives for analysis.

EDA for first Study objective

- Understanding the data:

Verification of null values to rectify and the description of summary are coded as below.



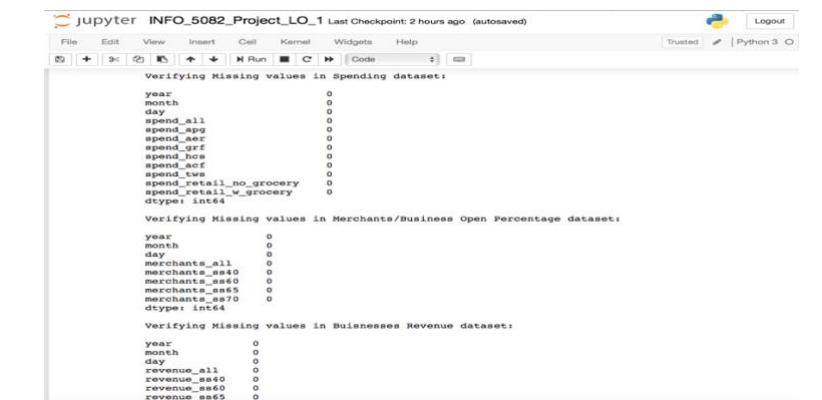
The screenshot shows a Jupyter Notebook interface with two code cells. The first cell contains Python code to define a function for verifying missing values and then calls it on three datasets: afs_data, wm_data, and wr_data. The output of the first cell shows the function definition and the results for each dataset, all indicating zero missing values.

```
In [5]: #Using functions to verify missing values in the dataset
def verifyMissingValuesInData(fileData):
    print(fileData.isnull().sum())
    return

#Calling above function to verify missing values and print results
print("Verifying Missing values in Spending dataset: \n")
verifyMissingValuesInData(afs_data)

print("\nVerifying Missing values in Merchants/Business Open Percentage dataset: \n")
verifyMissingValuesInData(wm_data)

print("\nVerifying Missing values in Businesses Revenue dataset: \n")
verifyMissingValuesInData(wr_data)
```

The screenshot shows the output of the code execution from the previous cell. It displays three sections of output, each titled "Verifying Missing values in [dataset]". The datasets listed are Spending, Merchants/Business Open Percentage, and Businesses Revenue. Each section shows a table of column names and their corresponding counts of missing values (0). The final output line shows the data type as int64.

```
Verifying Missing values in Spending dataset:
year          0
month         0
day           0
spend_all     0
spend_app     0
spend_aer     0
spend_csf     0
spend_hcs     0
spend_acf     0
spend_wes     0
spend_retail_no_grocery  0
spend_retail_w_grocery  0
dtype: int64

Verifying Missing values in Merchants/Business Open Percentage dataset:
year          0
month         0
day           0
merchants_all 0
merchants_ss40 0
merchants_ss60 0
merchants_ss65 0
merchants_ss80 0
dtype: int64

Verifying Missing values in Businesses Revenue dataset:
year          0
month         0
day           0
revenue_all   0
revenue_ss40  0
revenue_ss60  0
revenue_ss65  0
revenue_ss80  0
```

From the output generated there are no null/missing values in the dataset, so further describing the quantitative summaries as below.

- Summary of Customer expenditure dataset:

```
In [6]: #Using functions to describe the dataset
def descData(fileData):
    print("Dimension of the "+name+" dataset: \n",fileData.shape)
    print("Data Types of the "+name+" dataset: \n",fileData.dtypes)
    print("View of the first 10 records of "+name+" dataset: \n",fileData.head(10))
    print("\nSummary of the "+name+" dataset: \n",fileData.describe())
    return

#Calling above function to print dataset description
descData(afs_data,"Customer Spending")

descData(wm_data,"Merchants/Businesses Open Percentage")
descData(wr_data,"Businesses Revenue")
```

Dimension of the Customer Spending dataset:
(302, 12)

Data Types of the Customer Spending dataset:

year	int64
month	int64
day	int64
spend_all	float64
spend_apg	float64
spend_aer	float64
spend_grf	float64
spend_hcf	float64
spend_lnf	float64
spend_tws	float64
spend_retail_no_grocery	float64
spend_retail_v_grocery	float64

dtype: object

- Summary of Business open percentage dataset :

```
#Calling above function to print dataset description
descData(afs_data,"Customer Spending")

descData(wm_data,"Merchants/Businesses Open Percentage")
descData(wr_data,"Businesses Revenue")
```

Dimension of the Merchants/Businesses Open Percentage dataset:
(302, 8)

Data Types of the Merchants/Businesses Open Percentage dataset:

year	int64
month	int64
day	int64
merchants_all	float64
merchants_ss40	float64
merchants_ss60	float64
merchants_ss65	float64
merchants_ss70	float64

dtype: object

View of the first 10 records of Merchants/Businesses Open Percentage dataset:

	year	month	day	merchants_all	merchants_ss40	merchants_ss60
0	2020	1	13	-0.00689	-0.00576	-0.00921
1	2020	1	14	-0.00854	-0.00749	-0.01020
2	2020	1	15	-0.01010	-0.00955	-0.01070

The results generated have shown some outliers in this dataset which are further visualized and rectified by using linear interpolation method.

- Summary of Business open percentage dataset :

```
#Calling above function to print dataset description
descData(afs_data,"Customer Spending")

descData(wm_data,"Merchants/Businesses Open Percentage")

descData(wr_data,"Businesses Revenue")
```

Dimension of the Businesses Revenue dataset:
(302, 8)

Data Types of the Businesses Revenue dataset:

	year	int64
month	int64	
day	int64	
revenue_all	float64	
revenue_ss40	float64	
revenue_ss60	float64	
revenue_ss65	float64	
revenue_ss70	float64	
dtype:	object	

View of the first 10 records of Businesses Revenue dataset:

	year	month	day	revenue_all	revenue_ss40	revenue_ss60	revenue_ss65
0	2020	1	13	-0.020100	-0.01530	-0.017200	-0.03430
1	2020	1	14	-0.015100	-0.01190	-0.008520	-0.02730
2	2020	1	15	-0.011600	-0.00846	-0.000735	-0.02420
3	2020	1	16	0.000210	0.00055	0.001150	0.00180

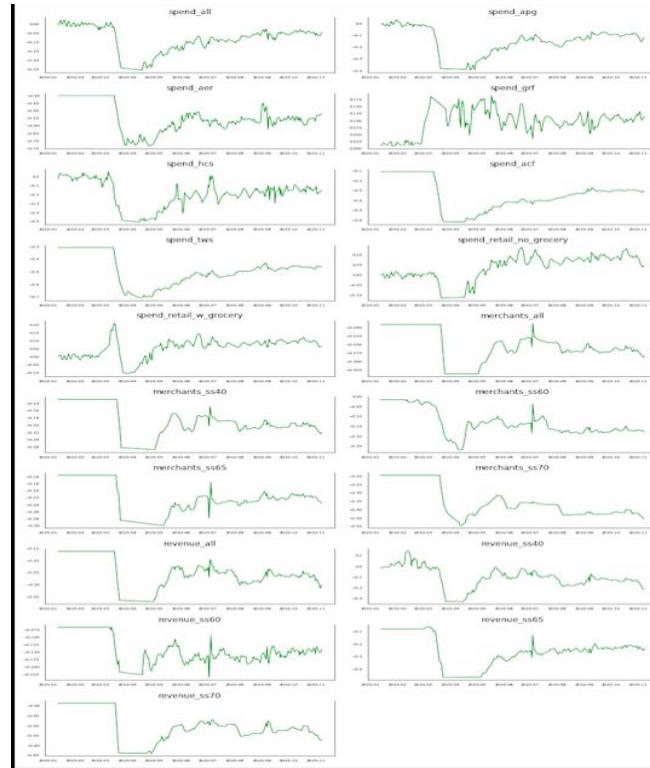
From these datasets required time series variables are combined together for the analysis to first objective,

```
In [13]: lol_data = pd.concat([afs_data, wm_data, wr_data], axis=1, sort=False)
lol_data
```

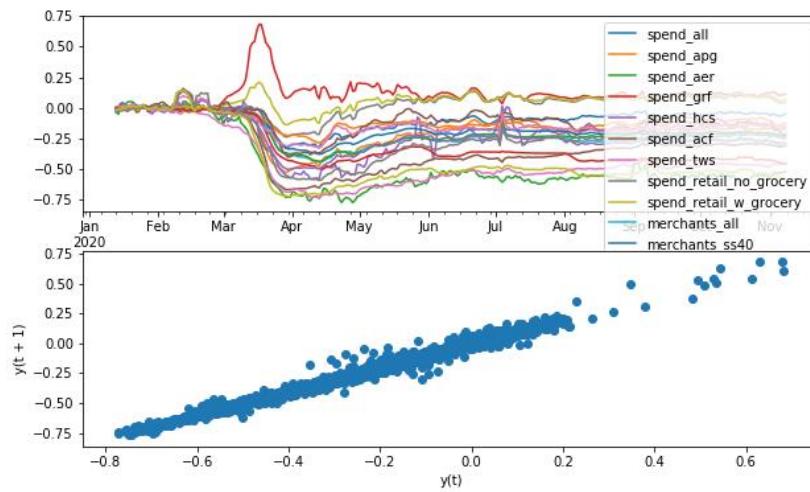
Date	spend_all	spend_apg	spend_aer	spend_grf	spend_hcs	spend_acf	spend_tws	spend_retail_no_grocery	spend_retail_w_grocery	mercha
2020-01-13	-0.003010	-0.003150	-0.033500	0.011300	-0.02620	-0.004100	0.00912	0.000825	0.00444	-0.0
2020-01-14	-0.000435	-0.004320	-0.051100	0.012400	0.00826	-0.001880	-0.00121	0.002680	0.00609	-0.0
2020-01-15	0.015400	-0.001820	0.033000	0.014800	0.03880	-0.000584	0.00906	0.003590	0.00757	-0.0
2020-01-16	0.022700	-0.000329	0.044000	0.020100	0.03580	0.002950	0.01860	0.006140	0.01120	-0.0
2020-01-17	0.001320	-0.016800	0.024800	-0.013800	0.02290	-0.005100	0.01640	-0.012000	-0.01260	-0.0
2020-01-18	0.001300	-0.020100	0.046600	-0.026400	0.01880	-0.007280	0.01340	-0.013800	-0.01830	-0.0
2020-01-19	0.001300	0.000000	0.001500	0.001700	0.001600	0.000700	0.001600	0.000700	0.001600	0.001600

Further dataset is rectified for null values and visualized for better understanding,

- Univariate plots for each time series variable in the dataset:



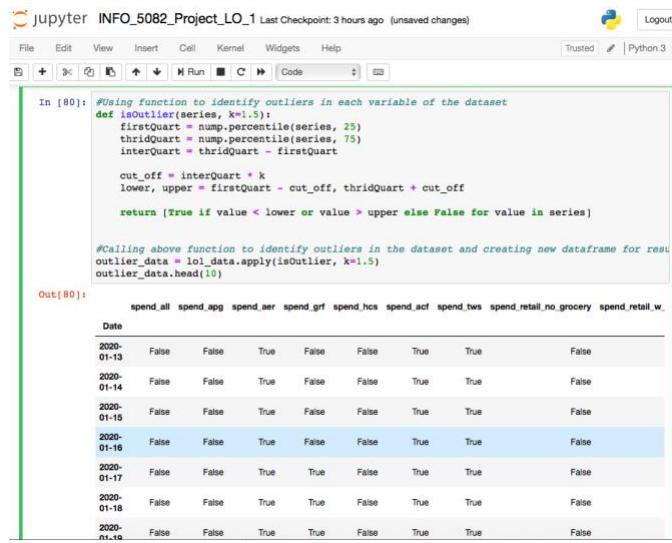
We can see the a slightly positive trend pattern in all variables from the month May, 2020. Further to understand the underlying structure pattern in the dataset, lag plot is drawn as below,



From the above plot of lag variables we can clearly see the outliers in the dataset. Also, we can clearly see the positive trend pattern in the dataset meaning each time series variable has linearity with its lagged variable. So, Vector Autocorrelation (VAR) can be applied if there is no cointegration among the variables in dataset.

- ❖ Data Cleaning for rectifying outliers:

As the outliers in each time series variable are identified as below.



```
#Using function to identify outliers in each variable of the dataset
def isOutlier(series, k=1.5):
    firstQuart = np.percentile(series, 25)
    thirdQuart = np.percentile(series, 75)
    interQuart = thirdQuart - firstQuart

    cut_off = interQuart * k
    lower, upper = firstQuart - cut_off, thirdQuart + cut_off

    return [True if value < lower or value > upper else False for value in series]

#Calling above function to identify outliers in the dataset and creating new dataframe for result
outlier_data = lol_data.apply(isOutlier, k=1.5)
outlier_data.head(10)
```

Date	spend_all	spend_apg	spend_aer	spend_grf	spend_hcs	spend_acf	spend_tws	spend_retail_no_grocery	spend_retail_w
2020-01-13	False	False	True	False	False	True	True	False	
2020-01-14	False	False	True	False	False	True	True	False	
2020-01-15	False	False	True	False	False	True	True	False	
2020-01-16	False	False	True	False	False	True	True	False	
2020-01-17	False	False	True	True	False	True	True	False	
2020-01-18	False	False	True	True	False	True	True	False	
2020-01-19	False	False	True	True	False	True	True	False	

From the results generated, time series variables ‘spend_aer’, ‘spend_acf’, ‘spend_tws’, ‘merchant_all’ and more have shown outliers. So these outlier data points are rectified by using ‘Linear Interpolate’ method as below

❖ Rectifying outliers:

```
In [23]: #Replacing above detected outliers values with 'NaN'
for column in lol_data:
    lol_data[column] = np.where(outlier_data[column] == True, 'NaN', lol_data[column])

#Converting the dataframe columns/variables to Numeric type
lol_data[lol_data.columns] = lol_data[lol_data.columns].apply(pd.to_numeric, errors='coerce')
lol_data.head(10)

Out[23]:
spend_all spend_apg spend_aer spend_grf spend_hcs spend_acf spend_tws spend_retail_no_grocery spend_retail_w_t
Date
2020-01-13 -0.003010 -0.003150 NaN 0.0113 -0.02620 NaN NaN 0.000825
2020-01-14 -0.000435 -0.004320 NaN 0.0124 0.00826 NaN NaN 0.002680
2020-01-15 0.015400 -0.001820 NaN 0.0148 0.03880 NaN NaN 0.003590
2020-01-16 0.022700 -0.000329 NaN 0.0201 0.03580 NaN NaN 0.006140
2020-01-17 0.001320 -0.016800 NaN NaN 0.02290 NaN NaN -0.012000
2020-01-18 0.001300 -0.020100 NaN NaN 0.01880 NaN NaN -0.013800
2020-01-19 0.011000 -0.013900 NaN NaN 0.01520 NaN NaN -0.008970
2020-01-20 0.025200 -0.001510 NaN 0.0120 0.01580 NaN NaN 0.004540
2020-01-21 0.012400 -0.012400 NaN NaN 0.02580 NaN NaN -0.004900
2020-01-22 -0.000322 -0.022900 NaN NaN 0.02720 NaN NaN -0.013100
```

```
In [24]: #Filling all the 'NaN' values using linear method for interpolation
lol_data = lol_data.interpolate(method='linear', axis=0).bfill().ffill()
lol_data.head(10)

Out[24]:
spend_all spend_apg spend_aer spend_grf spend_hcs spend_acf spend_tws spend_retail_no_grocery spend_retail_w_t
Date
2020-01-13 -0.003010 -0.003150 -0.397 0.011300 -0.02620 -0.107 -0.306 0.000825
2020-01-14 -0.000435 -0.004320 -0.397 0.012400 0.00826 -0.107 -0.306 0.002680
2020-01-15 0.015400 -0.001820 -0.397 0.014800 0.03880 -0.107 -0.306 0.003590
2020-01-16 0.022700 -0.000329 -0.397 0.020100 0.03580 -0.107 -0.306 0.006140
2020-01-17 0.001320 -0.016800 -0.397 0.018075 0.02290 -0.107 -0.306 -0.012000
2020-01-18 0.001300 -0.020100 -0.397 0.016050 0.01880 -0.107 -0.306 -0.013800
2020-01-19 0.011000 -0.013900 -0.397 0.014025 0.01520 -0.107 -0.306 -0.008970
2020-01-20 0.025200 -0.001510 -0.397 0.012000 0.01580 -0.107 -0.306 0.004540
2020-01-21 0.012400 -0.012400 -0.397 0.014720 0.02580 -0.107 -0.306 -0.004900
2020-01-22 -0.000322 -0.022900 -0.397 0.017440 0.02720 -0.107 -0.306 -0.013100
```

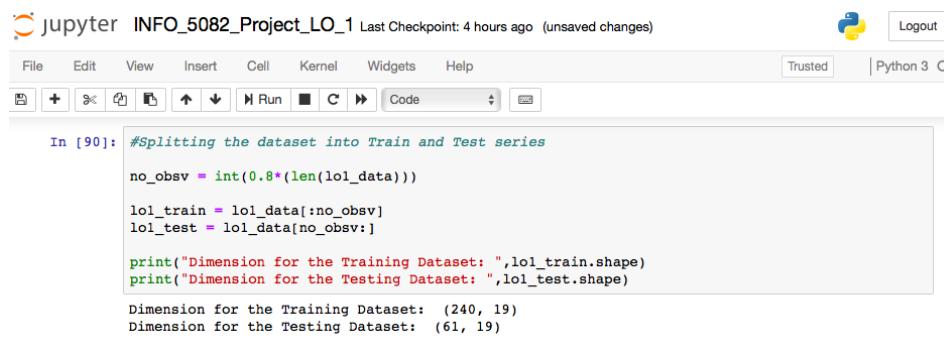
Now the outliers in dataset is cleaned and rectified, further the dataset is splitted into

Training and Test dataset.

❖ Splitting data into Train and Test data:

In this multivariate time series analysis, VAR model is fitted and trained with train dataset to forecast the test dataset and the actual test dataset is used to compare with predicted output for model accuracy.

For first objective's analysis dataset is split into (0.8) 80% for training and (0.2) 20% for testing as below.



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter INFO_5082_Project_LO_1 Last Checkpoint: 4 hours ago (unsaved changes)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help
- Cell Buttons:** Run, Cell, Kernel, Help, Code
- User Options:** Trusted, Python 3
- Cell Content (In [90]):**

```
#Splitting the dataset into Train and Test series
no_observ = int(0.8*(len(lol_data)))
lol_train = lol_data[:no_observ]
lol_test = lol_data[no_observ:]

print("Dimension for the Training Dataset: ",lol_train.shape)
print("Dimension for the Testing Dataset: ",lol_test.shape)
```
- Output:** Dimension for the Training Dataset: (240, 19)
Dimension for the Testing Dataset: (61, 19)

❖ Stationarity and making a stationary time series:

The assumption of summary statistics from observations are consistent require the time series to be stationary. This analysis consistency can be disturbed by the trend, seasonality or other time dependent patterns.

A stationary time series has the mean, variance and covariance that doesn't vary with time. Stationarity is an important element in time series analysis. There are three types stationarity, a) Strict stationary – this strict stationary series has the mean , variance and covariance that are not a function of time. For our analysis, we must convert a non-stationary time series into stationary, b) Trend Stationary – A series that has trend but no unit root is called Trend stationary which can be converted into Strict stationary by removing the trend and c) Difference Stationary – A strict stationary formed by using differencing is called Difference stationary.

A non-stationary time series can be converted into stationary by i) Differencing – here, to get rid of the varying mean we compute the difference between consecutive terms in series. ii) Seasonal Differencing – here, we compute the difference between two observations in same season over different period like difference between a observation in a month with its previous month. iii) Transformation – here, methods like log transformation, square root and power transform are used to stabilize a series with non-constant variance.

❖ Verifying and converting to stationarity:

To verify time series property of stationarity in the dataset variables, Augmented Dickey Fuller test function is applied as below.

```
In [89]: #Using function to run Augmented Dickey-Fuller Test (ADF Test) to check for stationarity in the
from statsmodels.tsa.stattools import adfuller

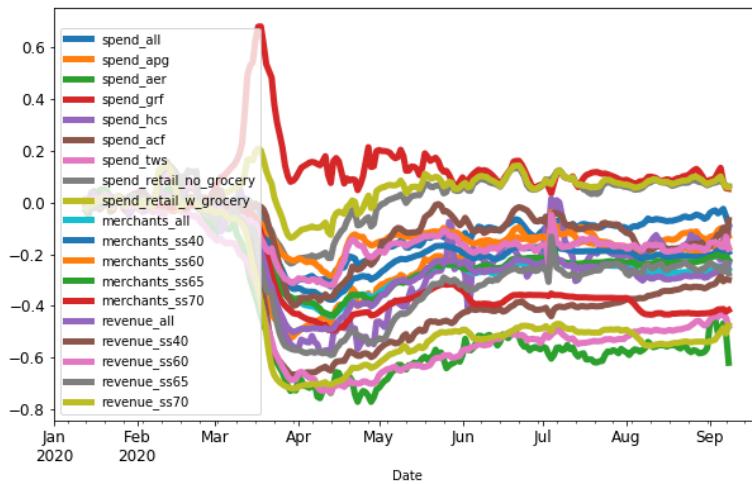
def Stationary_test_ADF(dataset):
    for i in dataset.columns:
        print("\n \nADF test for Column '"+i+":")
        print('-----')
        adf_test = adfuller(dataset[i], autolag="AIC")
        adf = pandas.Series(adf_test[0:4], index = ['Test Statistic ','p-value ','No_lags ',''
        for key, value in adf_test[4].items():
            adf['Critical Value for %s : '%key] = value
        print (adf)
        return

| Stationary_test_ADF(lol_train)

ADF test for Column 'spend_all':
-----
Test Statistic : -2.409741
p-value : 0.139006
No_lags : 12.000000
No_Obsv : 227.000000
Critical Value for 1% : -3.459490
Critical Value for 5% : -2.874358
Critical Value for 10% : -2.573602
dtype: float64

ADF test for Column 'spend_apg':
-----
Test Statistic : -2.019889
p-value : 0.277921
```

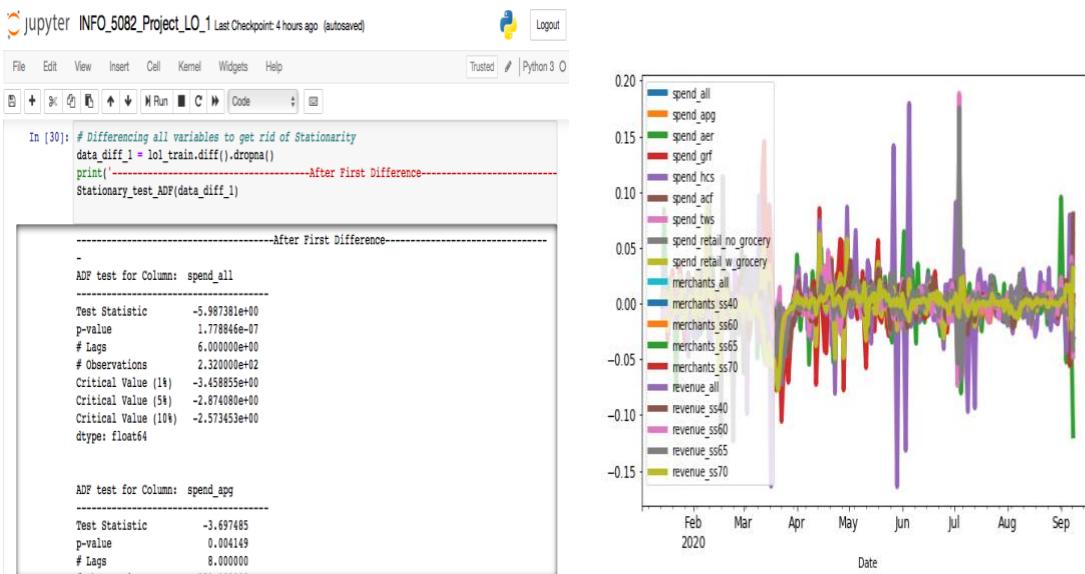
The ADF tests states, Null hypothesis – Data has the unit root meaning it is a non-stationary data when the p-value > 0.05 meaning weak evidence to reject null hypothesis.



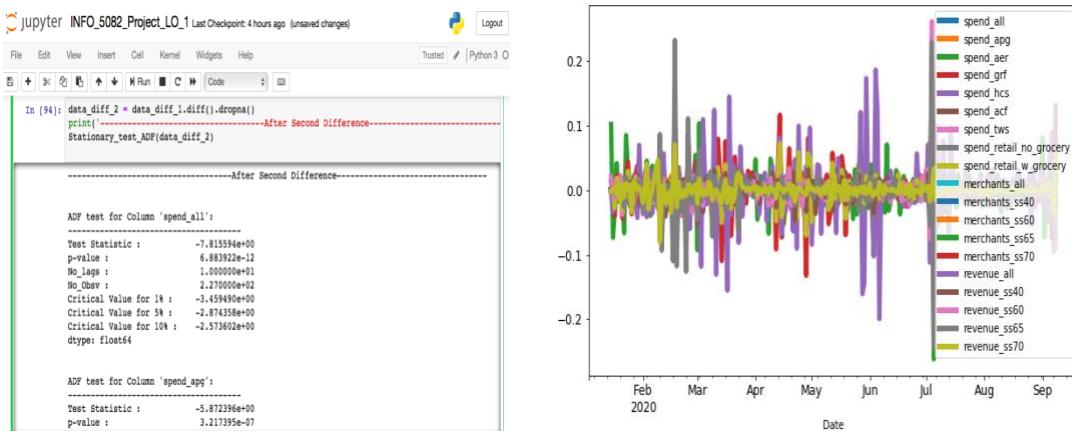
As the results shown that data is not stationary, to convert the time series dataset into stationary Differencing method has been implemented in two orders where the difference of consecutive terms are computed mathematically as,

$$y_t = y_t + y_{t-1}, \quad \text{where } y_t \text{ value of variable 'y' at a time 't'}$$

After first differencing, ADF test results shown as non-stationary, so second differencing is applied.



❖ ADF test results from second difference has shown as stationary.



Now the data is in stationary and ready for time series modelling.

❖ Cointegration:

It can be defined in simple as two time series variables x_t and y_t are said to be cointegrated if there exists a parameter ‘ α ’ such as $u_t = y_t - \alpha x_t$ is a stationary process.

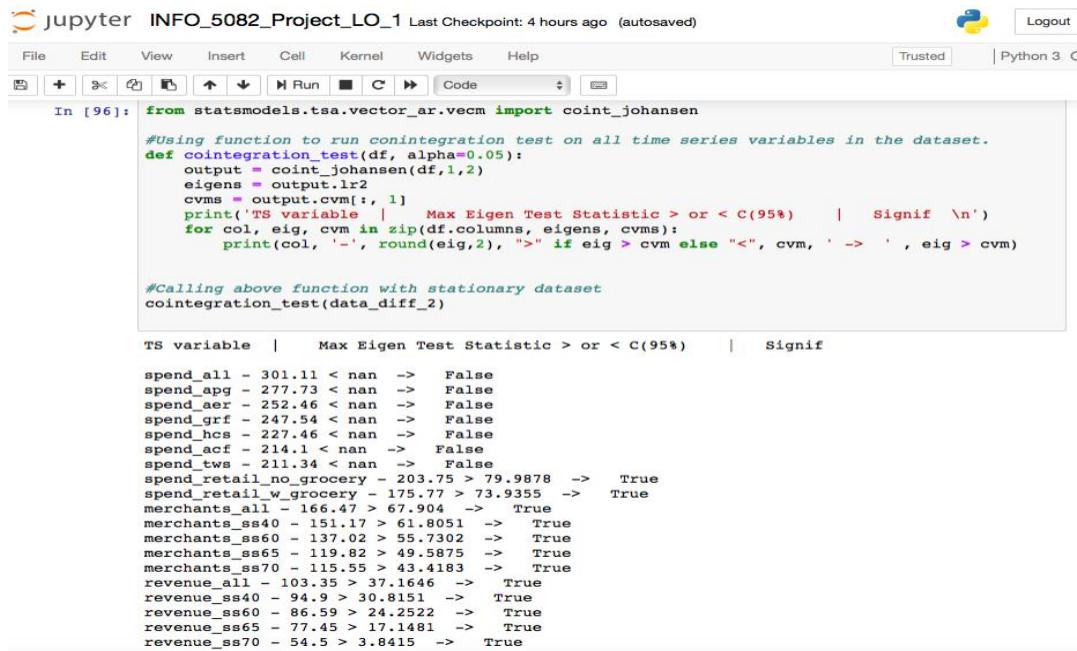
Cointegration for autoregression VAR(k) model in general is given as,

$$\Delta y_t = \Pi y_{t-1} + \sum_{j=1}^{k-1} \Gamma_j \Delta y_{t-j} + e_t$$

When $\Pi = 0$ means there is no cointegration, Π has full cointegrating rank ‘ r ’, then y_t must be stationary Π has less than full rank but not equal to zero then there is a conintegration.

❖ Cointegration Testing using Johansen’s function:

To select the appropriate model, dataset has been tested for cointegration among the variables to avoid the information loss using Johansen cointegration taking Eigen Test statistic with 95% confidence interval.



```

In [96]: from statsmodels.tsa.vector_ar.vecm import coint_johansen
          #Using function to run cointegration test on all time series variables in the dataset.
          def cointegration_test(df, alpha=0.05):
              output = coint_johansen(df, 1, 2)
              eigens = output.eigenvalues
              cvms = output.cvm[:, 1]
              print('TS variable | Max Eigen Test Statistic > or < C(95%) | Signif \n')
              for col, eig, cvm in zip(df.columns, eigens, cvms):
                  print(col, '-', round(eig, 2), ">" if eig > cvm else "<", cvm, ' -> ', eig > cvm)

          #Calling above function with stationary dataset
          cointegration_test(data_diff_2)

TS variable | Max Eigen Test Statistic > or < C(95%) | Signif
spend_all - 301.11 < nan -> False
spend_apg - 277.73 < nan -> False
spend_aer - 252.46 < nan -> False
spend_grf - 247.54 < nan -> False
spend_hcs - 227.46 < nan -> False
spend_acf - 214.1 < nan -> False
spend_tws - 211.34 < nan -> False
spend_retail_no_grocery - 203.75 > 79.9878 -> True
spend_retail_w_grocery - 175.77 > 73.9355 -> True
merchants_all - 166.47 > 67.904 -> True
merchants_ss40 - 151.17 > 61.8051 -> True
merchants_ss60 - 137.02 > 55.7302 -> True
merchants_ss65 - 119.82 > 49.5875 -> True
merchants_ss70 - 115.55 > 43.4183 -> True
revenue_all - 103.35 > 37.1646 -> True
revenue_ss40 - 94.9 > 30.8151 -> True
revenue_ss60 - 86.59 > 24.2522 -> True
revenue_ss65 - 77.45 > 17.1481 -> True
revenue_ss70 - 54.5 > 3.8415 -> True

```

Johansen’s cointegration test helps us to identify the statistically significant relation among time series variables in the dataset. The null hypothesis of Johansen test implies that the sum of eigen values or trace is equal to zero meaning there is no cointegration or linear trend in

the dataset. If eigen statistic value or trace is greater than critical value then we can reject the null hypothesis with given confidence level.

From the results generated, we can see that some variables are cointegrated that means in long run the time series variables are significant relationship this dataset when modelled using VECM model can have little loss of information than when modelled with VAR.

❖ Granger Causality Matrix:

This causality matrix reveals if any time series variables in the dataset are causing effect/consequences when forecasting a time series variable. The causation of a time series variable in the dataset by its own lagged variable and other variables is given by the p-value stating as when < 0.05 (significance) then dependent variable(y) is caused by another variable (x) and vice versa.

From the granger causality matrix applied we can see that there is some causality or linearity between some non-cointegrated variables, so this dataset is also modelled with VAR model.

	spend_all.x	spend_apg.x	spend_aer.x	spend_grf.x	spend_hcs.x	spend_acf.x	spend_tws.x	spend_ss40.y
spend_all.y	1.0000	0.0000	0.0040	0.0973	0.0205	0.0000	0.0000	
spend_apg.y	0.0000	1.0000	0.0322	0.0059	0.0000	0.0000	0.0000	
spend_aer.y	0.0052	0.0042	1.0000	0.0023	0.0250	0.0000	0.0001	
spend_grf.y	0.4246	0.0511	0.0003	1.0000	0.1422	0.0668	0.0671	
spend_hcs.y	0.0043	0.0178	0.0004	0.0121	1.0000	0.0057	0.0026	
spend_acf.y	0.0687	0.0000	0.0255	0.0259	0.0208	1.0000	0.0048	
spend_tws.y	0.0462	0.0204	0.0212	0.3717	0.0646	0.0000	1.0000	
spend_retail_no_grocery.y	0.4109	0.0000	0.0261	0.0000	0.0093	0.0000	0.0009	
spend_retail_w_grocery.y	0.0289	0.0000	0.0305	0.0000	0.0152	0.0000	0.0000	
merchants_all.y	0.0000	0.0000	0.0001	0.0180	0.0320	0.0000	0.0000	
merchants_ss40.y	0.0000	0.0000	0.0000	0.0521	0.0068	0.0000	0.0000	
merchants_ss60.y	0.1627	0.0297	0.0915	0.0000	0.0355	0.0000	0.0000	
merchants_ss65.y	0.0000	0.0001	0.0887	0.0003	0.0023	0.0000	0.0000	
merchants_ss70.y	0.0010	0.0000	0.0046	0.0000	0.0028	0.0000	0.0001	
revenue_all.y	0.1488	0.0000	0.0392	0.8131	0.0001	0.0000	0.0000	
revenue_ss40.y	0.0000	0.0002	0.0310	0.0352	0.0016	0.0001	0.0000	
revenue_ss60.y	0.0709	0.0000	0.0092	0.0000	0.0184	0.0000	0.0007	
revenue_ss65.y	0.0562	0.0042	0.5908	0.0010	0.0282	0.0000	0.0001	
revenue_ss70.y	0.0081	0.0000	0.0008	0.0002	0.0004	0.0000	0.0000	

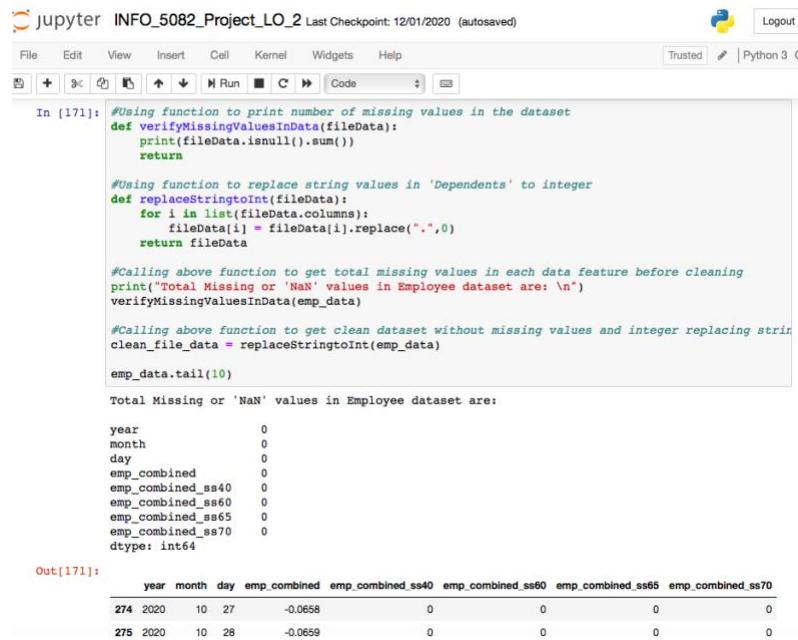
From the accuracy results of VAR and VECM model we can say which model is better for time series analysis of first study objective.

EDA for Second Study objective

- Understanding the data:

Verification of null values to rectify and the description of summary are coded as below.

- Employment rates' dataset:



```
#Using function to print number of missing values in the dataset
def verifyMissingValuesInData(fileData):
    print(fileData.isnull().sum())
    return

#Using function to replace string values in 'Dependents' to integer
def replaceStringToInt(fileData):
    for i in list(fileData.columns):
        fileData[i] = fileData[i].replace(".",0)
    return fileData

#Calling above function to get total missing values in each data feature before cleaning
print("Total Missing or 'NaN' values in Employee dataset are: \n")
verifyMissingValuesInData(emp_data)

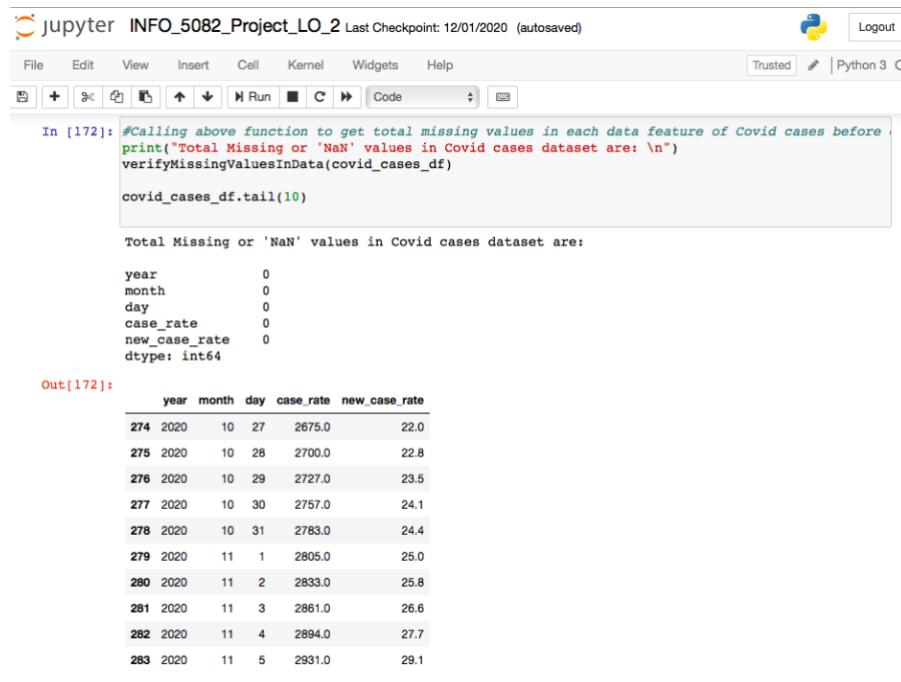
#Calling above function to get clean dataset without missing values and integer replacing string
clean_file_data = replaceStringToInt(emp_data)

emp_data.tail(10)
```

Total Missing or 'NaN' values in Employee dataset are:

year	month	day	emp_combined	emp_combined_ss40	emp_combined_ss60	emp_combined_ss65	emp_combined_ss70
274	2020	10	27	-0.0658	0	0	0
275	2020	10	28	-0.0659	0	0	0

- Covid-19 infection rates' dataset:



```
#Calling above function to get total missing values in each data feature of Covid cases before cleaning
print("Total Missing or 'NaN' values in Covid cases dataset are: \n")
verifyMissingValuesInData(covid_cases_df)

covid_cases_df.tail(10)
```

Total Missing or 'NaN' values in Covid cases dataset are:

year	month	day	case_rate	new_case_rate
274	2020	10	27	2675.0
275	2020	10	28	2700.0
276	2020	10	29	2727.0
277	2020	10	30	2757.0
278	2020	10	31	2783.0
279	2020	11	1	2805.0
280	2020	11	2	2833.0
281	2020	11	3	2861.0
282	2020	11	4	2894.0
283	2020	11	5	2931.0

From the output generated, there are no null/missing values in the dataset, so further describing the quantitative summaries as below.

- ❖ Summary of Employment rates' dataset:

```

In [174]: #Using functions to describe the dataset
def descData(fileData,name):
    print("\nDimension of the "+name+" dataset: \n",fileData.shape)
    print("\nData Types of the "+name+" dataset: \n",fileData.dtypes)
    print("\nView of the first 10 records of "+name+" dataset: \n",fileData.head(10))
    print("\nSummary of the "+name+" dataset: \n",fileData.describe())
    return

#Calling above function to print dataset description
descData(emp_data,"Employment")

emp_columns = ['emp_combined', 'emp_combined_ss40', 'emp_combined_ss60','emp_combined_ss65','emp_combined_ss70']

for each in emp_columns:
    emp_data[each] = panda.to_numeric(emp_data[each])

print("\n\n-----after conversion of dtypes-----")
descData(emp_data,"Employment")

```

Dimension of the Employment dataset:
 (284, 8)

 Data Types of the Employment dataset:
 year int64
 month int64
 day int64
 emp_combined float64
 emp_combined_ss40 object
 emp_combined_ss60 object
 emp_combined_ss65 object
 emp_combined_ss70 object
 dtype: object

 View of the first 10 records of Employment dataset:

	year	month	day	emp_combined	emp_combined_ss40	emp_combined_ss60
0	2020	1	27	0.00518	0.00334	0.00421

Since, this dataset has variables of object type conversion to numeric type is applied and the results are as below.

```

for each in emp_columns:
    emp_data[each] = panda.to_numeric(emp_data[each])

print("\n\n-----after conversion of dtypes-----")
descData(emp_data,"Employment")

```

-----after conversion of dtypes-----

 Dimension of the Employment dataset:
 (284, 8)

 Data Types of the Employment dataset:
 year float64
 month float64
 day float64
 emp_combined float64
 emp_combined_ss40 float64
 emp_combined_ss60 float64
 emp_combined_ss65 float64
 emp_combined_ss70 float64
 dtype: object

 View of the first 10 records of Employment dataset:

	year	month	day	emp_combined	emp_combined_ss40	emp_combined_ss60
0	2020	1	27	0.00518	0.00334	0.00421

❖ Summary of Covid-19 Infection rates' dataset :

```
In [175]: #Calling above function to print dataset description
descData(covid_cases_df, "Covid Cases")

Dimension of the Covid Cases dataset:
(284, 5)

Data Types of the Covid Cases dataset:
year      int64
month     int64
day       int64
case_rate float64
new_case_rate float64
dtype: object

View of the first 10 records of Covid Cases dataset:
   year  month  day  case_rate  new_case_rate
0  2020      1   27    0.00152      0.000174
1  2020      1   28    0.00152      0.000174
2  2020      1   29    0.00152      0.000174
3  2020      1   30    0.00183      0.000218
4  2020      1   31    0.00213      0.000218
5  2020      2    1    0.00244      0.000218
6  2020      2    2    0.00335      0.000261
7  2020      2    3    0.00335      0.000261
8  2020      2    4    0.00335      0.000261
9  2020      2    5    0.00366      0.000305

Summary of the Covid Cases dataset:
   year  month  day  case_rate  new_case_rate
count  284.0  284.00000  284.000000  284.000000  284.000000
mean  2020.0  6.024648  15.739437  969.992037  9.991574
std   0.0     2.701150  8.998374  895.132080  6.844277
min   2020.0  1.000000  1.000000  0.001520  0.00044
25%  2020.0  4.000000  8.000000  118.750000  6.432500
50%  2020.0  6.000000  16.000000  657.000000  9.240000
75%  2020.0  8.000000  24.000000  1771.500000  15.100000
```

The results generated have shown no outliers.

From these datasets required time series variables are combined together for the analysis

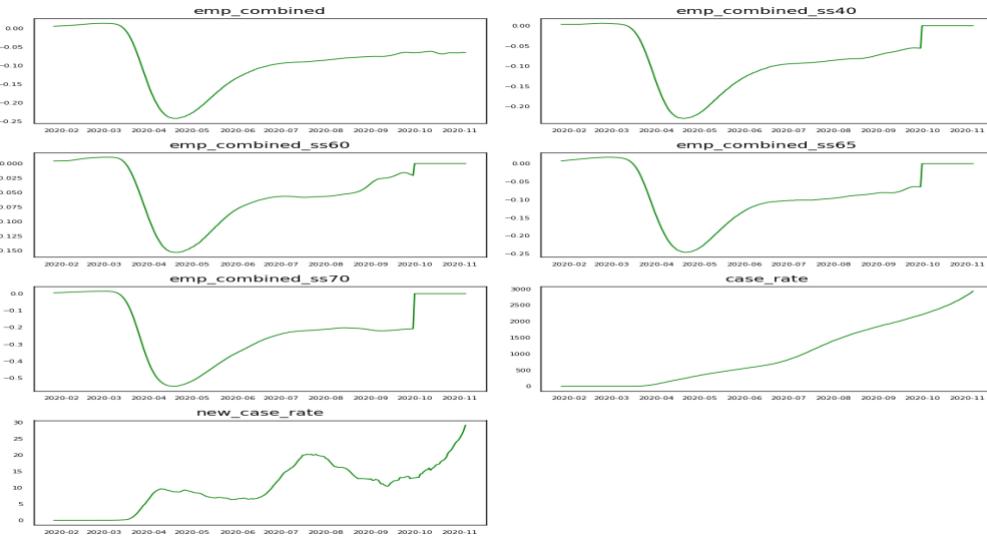
to second objective,

```
In [181]: lo2_data = pd.concat([emp_data, covid_cases_df], axis=1, sort=False)
lo2_data.head(10)

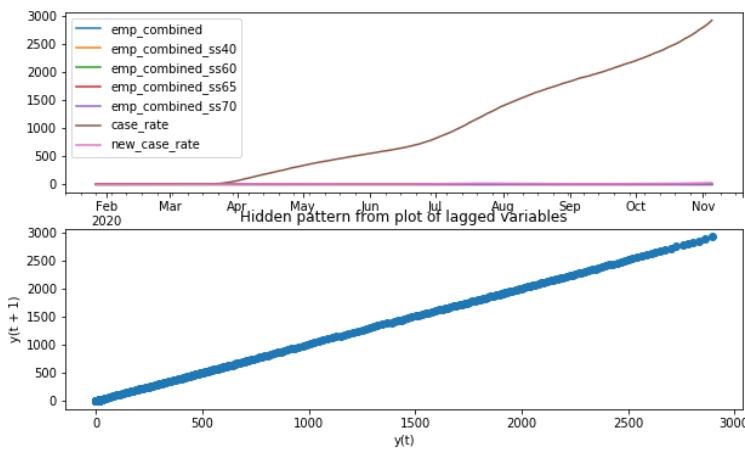
Out[181]:
   emp_combined  emp_combined_ss40  emp_combined_ss60  emp_combined_ss65  emp_combined_ss70  case_rate  new_ca
Date
2020-01-27    0.00518        0.00334        0.00421        0.00762        0.00462        0.00152    0.
2020-01-28    0.00542        0.00337        0.00429        0.00809        0.00493        0.00152    0.
2020-01-29    0.00563        0.00336        0.00435        0.00854        0.00524        0.00152    0.
2020-01-30    0.00582        0.00332        0.00439        0.00897        0.00557        0.00183    0.
2020-01-31    0.00600        0.00328        0.00443        0.00937        0.00592        0.00213    0.
2020-02-01    0.00617        0.00325        0.00446        0.00974        0.00626        0.00244    0.
2020-02-02    0.00633        0.00323        0.00449        0.01010        0.00664        0.00335    0.
2020-02-03    0.00651        0.00322        0.00455        0.01050        0.00705        0.00335    0.
2020-02-04    0.00671        0.00321        0.00463        0.01090        0.00754        0.00335    0.
2020-02-05    0.00682        0.00323        0.00476        0.01130        0.00803        0.00366    0.
```

- ❖ Univariate plots for each time series variable in the dataset:

Further dataset is visualized for better understanding



We can see the a slightly positive trend pattern in all variables from the month May, 2020. Further to understand the underlying structure pattern in the dataset, lag plot is drawn as below,

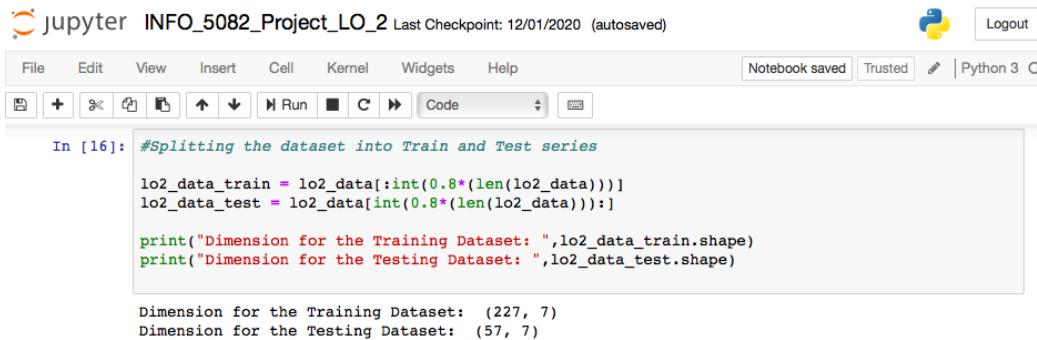


From the above plot of lag variables, there are no outliers in the dataset. Also, we can clearly see the positive trend pattern in the dataset meaning each time series variable has linearity

with its lagged variable. So, Vector Autocorrelation (VAR) can be applied if there is no cointegration among the variables in dataset.

- ❖ Splitting data into Train and Test data:

In this multivariate time series analysis for second study objective, dataset is split into (0.8) 80% for training and (0.2) 20% for testing as below.



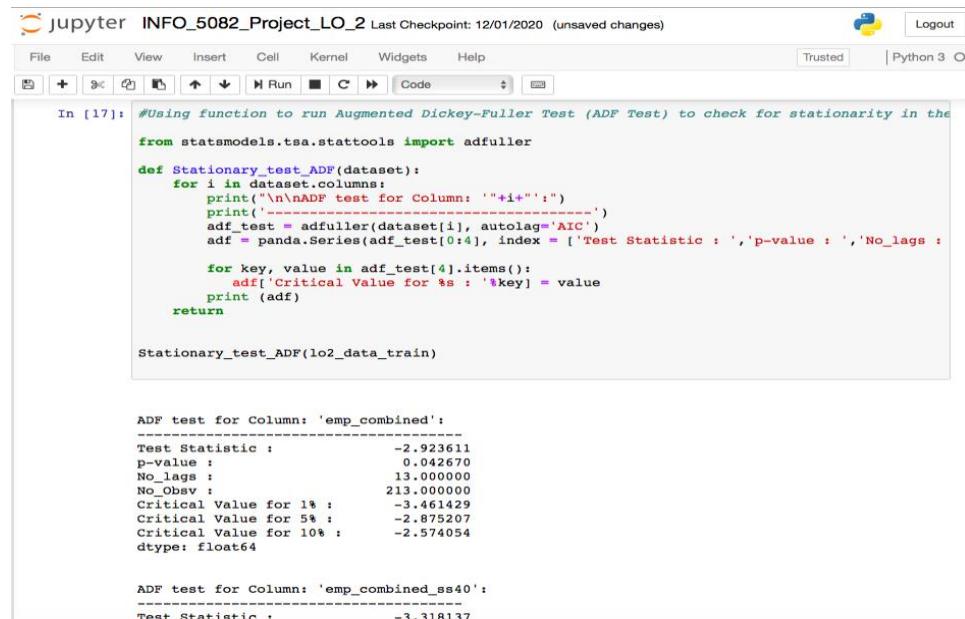
```
#Splitting the dataset into Train and Test series
lo2_data_train = lo2_data[:int(0.8*(len(lo2_data)))]
lo2_data_test = lo2_data[int(0.8*(len(lo2_data))):]

print("Dimension for the Training Dataset: ",lo2_data_train.shape)
print("Dimension for the Testing Dataset: ",lo2_data_test.shape)

Dimension for the Training Dataset: (227, 7)
Dimension for the Testing Dataset: (57, 7)
```

- ❖ Verifying and converting to stationarity:

To verify time series property of stationarity in the dataset variables, Augmented Dickey Fuller test function is applied on the train dataset as below.



```
#Using function to run Augmented Dickey-Fuller Test (ADF Test) to check for stationarity in the
from statsmodels.tsa.stattools import adfuller

def Stationary_test_ADF(dataset):
    for i in dataset.columns:
        print("\nADF test for Column: "+i+":")
        print('-----')
        adf_test = adfuller(dataset[i], autolag='AIC')
        adf = pandas.Series(adf_test[0:4], index = ['Test Statistic : ','p-value : ','No_lags : '])
        for key, value in adf_test[4].items():
            adf['Critical Value for %s : '%key] = value
        print(adf)
    return

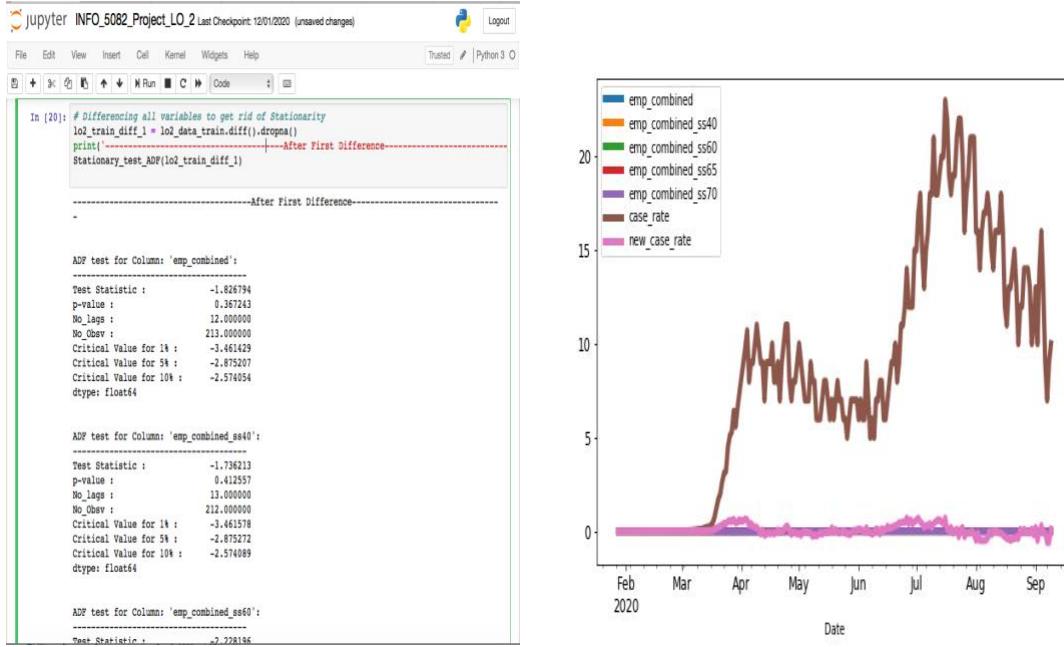
Stationary_test_ADF(lo2_data_train)

ADF test for Column: 'emp_combined':
-----
Test Statistic : -2.923611
p-value : 0.042670
No_lags : 13.000000
No_Obsv : 213.000000
Critical Value for 1% : -3.461429
Critical Value for 5% : -2.875207
Critical Value for 10% : -2.574054
dtype: float64

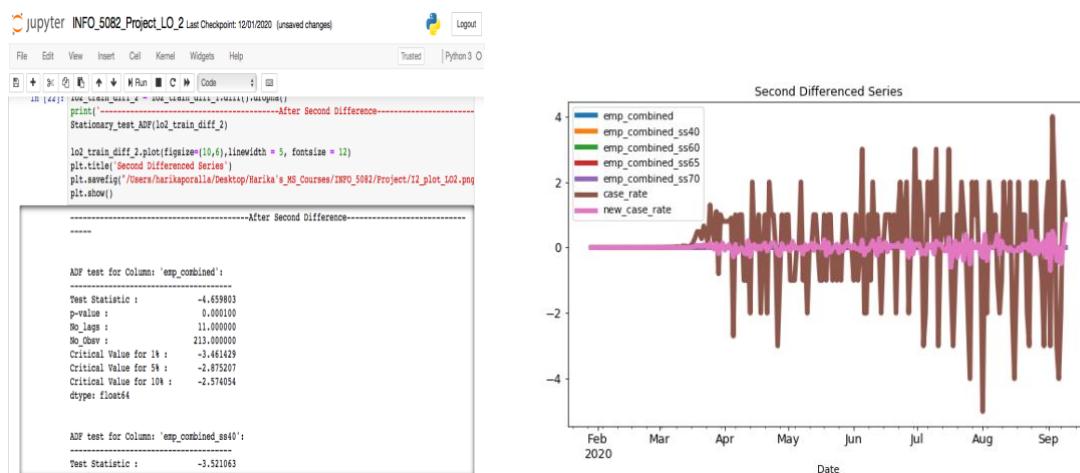
ADF test for Column: 'emp_combined_ss40':
-----
Test Statistic : -3.318137
```

As the results ADF test shown that data is not stationary, to convert the time series dataset into stationary Differencing method has been implemented in three orders.

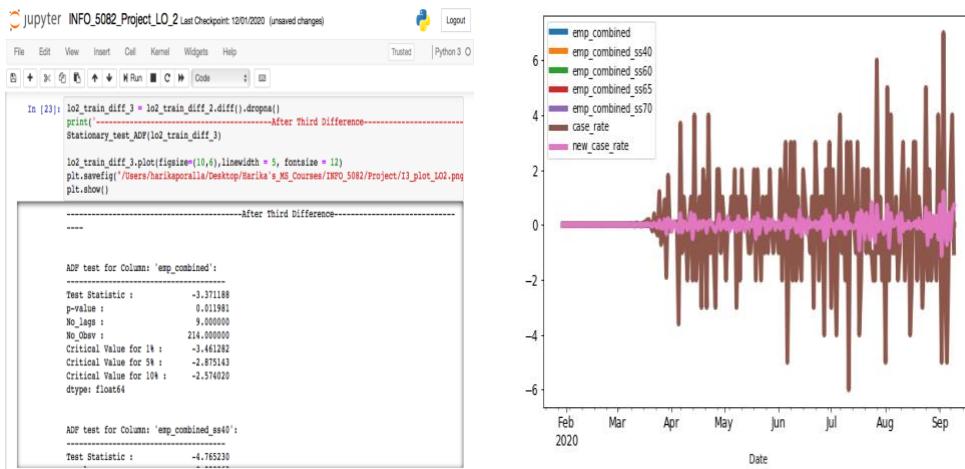
After first differencing, ADF test results shown as non-stationary, so second differencing is applied.



ADF test results from second difference are also not shown as stationary, so third differencing is performed.



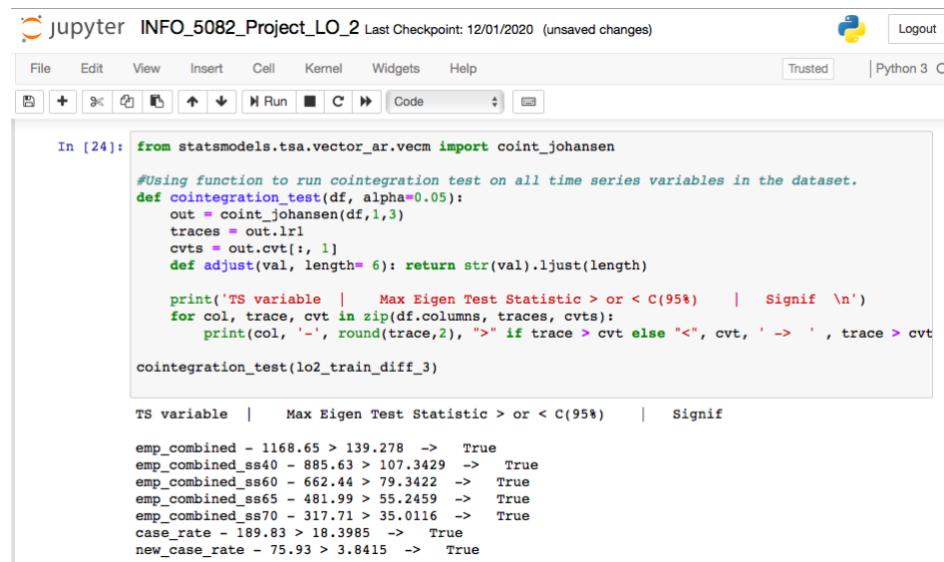
ADF results from third differencing has shown as stationary.



Now the data is in stationary and ready for time series modelling.

❖ Cointegration Testing using Johansen's function:

To select the appropriate model, dataset has been tested using Johansen's cointegration among the variables to avoid the information loss taking Trace Test statistic with 95% confidence interval.



From the results generated, we can see that all the time series variables are cointegrated meaning in long run the time series variables have significant correlations, so this dataset is modelled using VECM model to have little loss of information than when modelled with VAR.

❖ Granger Causality Matrix:

From the granger causality matrix applied we can see that there is causality or linearity between all the variables, so this dataset is with VECM model without ignoring any variables.

	emp_combined_x	emp_combined_ss40_x	emp_combined_ss60_x	emp_combined_ss65_x	emp_combined_ss70_x	c	w_case_rate_y
ip_combined_y	1.000	0.4892	0.1563	0.1351	0.028		
mbined_ss40_y	0.000	1.0000	0.0000	0.0000	0.0000		
mbined_ss60_y	0.000	0.0000	1.0000	0.0000	0.0000		
mbined_ss65_y	0.000	0.0000	0.0000	1.0000	0.0000		
mbined_ss70_y	0.000	0.0000	0.0000	0.0000	1.0000		
case_rate_y	0.000	0.0000	0.0000	0.0000	0.0000		
w_case_rate_y	0.015	0.0000	0.0000	0.0000	0.0000		

EDA for third study objective

- Understanding the data:

Verification of null values to rectify and the description of summary are coded as below.

- Google Mobility dataset:

```

In [7]: #Using function to print number of missing values in the dataset
def verifyMissingValuesInData(fileData):
    print(fileData.isnull().sum())
    return

#Calling above function to get total missing values in each data feature before cleaning
print("Total Missing or 'NaN' values in Google Mobility Data are: \n")
verifyMissingValuesInData(google_data)

google_data.tail(10)

Total Missing or 'NaN' values in Google Mobility Data are:
year          0
month         0
day           0
gps_residential      0
gps_away_from_home      0
dtype: int64

```

	year	month	day	gps_residential	gps_away_from_home
258	2020	11	11	0.0843	-0.0985
259	2020	11	12	0.0829	-0.0988
260	2020	11	13	0.0857	-0.0995
261	2020	11	14	0.0871	-0.1020
262	2020	11	15	0.0886	-0.1050
263	2020	11	16	0.0886	-0.1050
264	2020	11	17	0.0900	-0.1060
265	2020	11	18	0.0914	-0.1080

- Covid-19 infection rates' dataset:

The screenshot shows a Jupyter Notebook interface with the title "INFO_5082_Project_LO_3". The code cell In [7] contains Python code to verify missing values in a dataset named "covid_cases_df". The output cell Out[7] displays the last 10 rows of the dataset, which includes columns for year, month, day, case_rate, and new_case_rate. All values are non-null (0 or 16.0).

```

In [7]: #Calling above function to get total missing values in each data feature of Covid cases before
print("Total Missing or 'NaN' values in Covid-19 cases dataset are: \n")
verifyMissingValuesInData(covid_cases_df)

covid_cases_df.tail(10)

Total Missing or 'NaN' values in Covid-19 cases dataset are:
year          0
month         0
day           0
case_rate     0
new_case_rate 0
dtype: int64

Out[7]:
   year month day  case_rate  new_case_rate
258  2020    10   11      2362.0        16.0
259  2020    10   12      2377.0        15.3
260  2020    10   13      2393.0        15.8
261  2020    10   14      2411.0        16.1
262  2020    10   15      2431.0        16.5
263  2020    10   16      2452.0        17.0
264  2020    10   17      2468.0        17.1
265  2020    10   18      2483.0        17.2
266  2020    10   19      2502.0        17.9
267  2020    10   20      2521.0        18.2

```

From the output generated, there are no null/missing values in the dataset, so further describing the quantitative summaries as below.

❖ Summary of the Google Mobility dataset:

The screenshot shows a Jupyter Notebook interface with the title "INFO_5082_Project_LO_3". The code cell In [10] contains Python code to describe the "google_data" dataset, including its dimensions, data types, and first 10 records. The output cell Out[10] displays the summary statistics of the dataset, including count, mean, standard deviation, and quartiles for each column.

```

In [10]: #Using functions to describe the dataset
def descData(fileData,name):
    print("Dimensions of the "+name+" dataset: \n",fileData.shape)
    print("Data Types of the "+name+" dataset: \n",fileData.dtypes)
    print("View of the first 10 records of "+name+" dataset: \n",fileData.head(10))
    print("Summary of the "+name+" dataset: \n",fileData.describe())
    return

#Calling above function to print dataset description
descData(google_data,"Google Mobility")

Dimensions of the Google Mobility dataset:
(268, 5)

Data Types of the Google Mobility dataset:
year          int64
month         int64
day           int64
gps_residential    float64
gps_away_from_home float64
dtype: object

View of the first 10 records of Google Mobility dataset:
   year month day  gps_residential  gps_away_from_home
0  2020    2   27      -0.00429       0.00701
1  2020    2   28      -0.00571       0.00839
2  2020    2   29      -0.00714       0.01090
3  2020    3   1       -0.00714       0.01090
4  2020    3   2       -0.00857       0.01260
5  2020    3   3       -0.01000       0.01400
6  2020    3   4       -0.01000       0.01400
7  2020    3   5       -0.01140       0.01540
8  2020    3   6       -0.01140       0.01540
9  2020    3   7       -0.01000       0.01290

```

❖ Summary of Covid-19 Infection rates' dataset :

```
In [11]: #Calling above function to print dataset description
descData(covid_cases_df,"Covid Cases")

Dimension of the Covid Cases dataset:
(268, 5)

Data Types of the Covid Cases dataset:
year      int64
month     int64
day       int64
case_rate float64
new_case_rate float64
dtype: object

View of the first 10 records of Covid Cases dataset:
   year  month  day  case_rate  new_case_rate
0  2020      1    27  0.00152      0.000174
1  2020      1    28  0.00152      0.000174
2  2020      1    29  0.00152      0.000174
3  2020      1    30  0.00153      0.000216
4  2020      1    31  0.00113      0.000218
5  2020      2     1  0.00244      0.000218
6  2020      2     2  0.00335      0.000261
7  2020      2     3  0.00335      0.000261
8  2020      2     4  0.00335      0.000261
9  2020      2     5  0.00366      0.000305

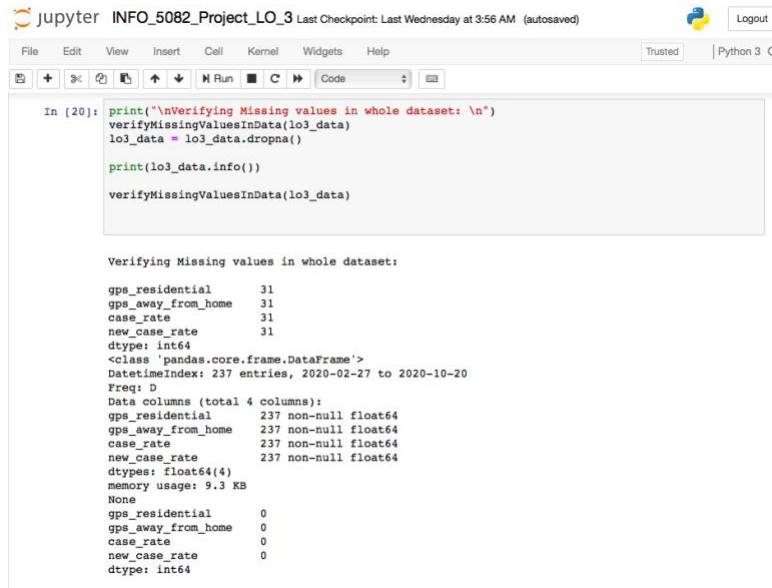
Summary of the Covid Cases dataset:
   year  month  day  case_rate  new_case_rate
count  268.0  268.000000  268.000000  268.000000
mean  2020.0  5.588645  15.555700  865.198278  9.205623
std   2.58887  2.588887  0.959018  0.016164  0.000494
min   2020.0  1.000000  1.000000  0.001520  0.000244
25%  2020.0  4.000000  8.000000  81.850000  6.360000
50%  2020.0  6.000000  15.000000  601.500000  8.955000
75%  2020.0  8.000000  17.000000  1211.000000  17.000000
```

The results generated have shown no outliers and these required time series variables from these datasets are combined together for the analysis of third study objective,

```
In [16]: lo3_data = pd.concat([google_data, covid_cases_df], axis=1, sort=False)
lo3_data.head(10)

Out[16]:
   Date  gps_residential  gps_away_from_home  case_rate  new_case_rate
2020-01-27        NaN             NaN  0.00152  0.000174
2020-01-28        NaN             NaN  0.00152  0.000174
2020-01-29        NaN             NaN  0.00152  0.000174
2020-01-30        NaN             NaN  0.00183  0.000218
2020-01-31        NaN             NaN  0.00213  0.000218
2020-02-01        NaN             NaN  0.00244  0.000218
2020-02-02        NaN             NaN  0.00335  0.000261
2020-02-03        NaN             NaN  0.00335  0.000261
2020-02-04        NaN             NaN  0.00335  0.000261
2020-02-05        NaN             NaN  0.00366  0.000305
```

- ❖ Null values are rectified as below.



```
In [20]: print("\nVerifying Missing values in whole dataset: \n")
verifyMissingValuesInData(io3_data)
io3_data = io3_data.dropna()

print(io3_data.info())

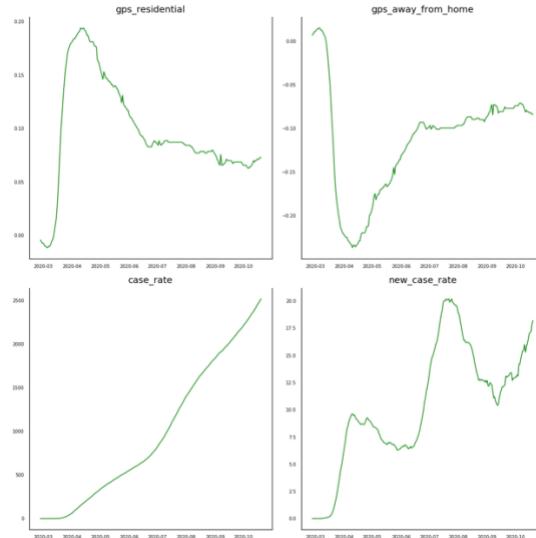
verifyMissingValuesInData(io3_data)
```

Verifying Missing values in whole dataset:

```
gps_residential      31
gps_away_from_home  31
case_rate            31
new_case_rate        31
dtype: int64
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 237 entries, 2020-02-27 to 2020-10-20
Freq: D
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype  
 0   gps_residential    237 non-null float64
 1   gps_away_from_home 237 non-null float64
 2   case_rate           237 non-null float64
 3   new_case_rate       237 non-null float64
dtypes: float64(4)
memory usage: 9.3 KB
None
gps_residential      0
gps_away_from_home  0
case_rate            0
new_case_rate        0
dtype: int64
```

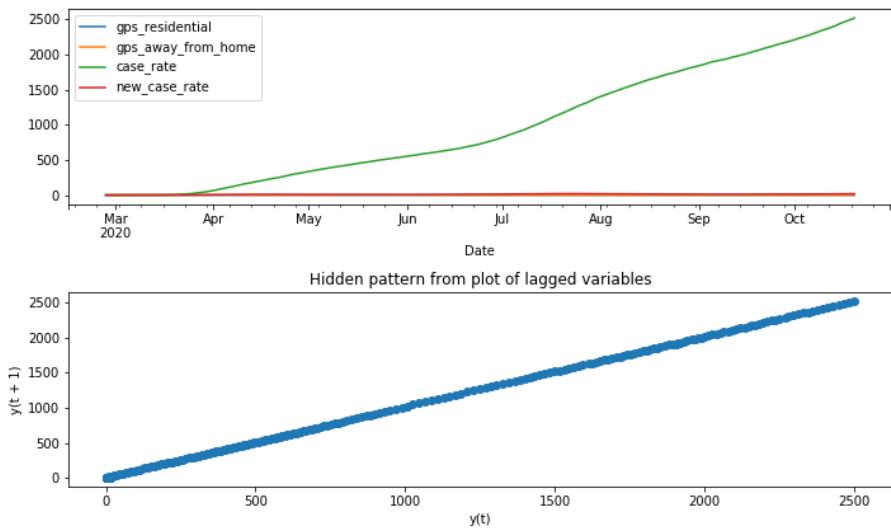
Further dataset is visualized for better understanding,

- ❖ Univariate plots for each time series variable in the dataset:



We can see the a slightly positive trend pattern in all variables except gps_residential from the month March, 2020.

Further to understand the underlying structure pattern in the dataset, lag plot is drawn as below,



From the above plot of lag variables, there are no outliers in the dataset. Also, we can clearly see the positive trend pattern in the dataset meaning each time series variable has linearity with its lagged variable. So, Vector Autocorrelation (VAR) can be applied if there is no cointegration among the variables in dataset.

❖ Splitting data into Train and Test data:

In this multivariate time series analysis for third study objective, dataset is split into (0.8) 80% for training and (0.2) 20% for testing as below.

```

In [29]: #Splitting the dataset into Train and Test series
lo3_data_train = lo3_data[:int(0.8*(len(lo3_data)))]
lo3_data_test = lo3_data[int(0.8*(len(lo3_data))):]

print("Dimension for the Training Dataset: ",lo3_data_train.shape)
print("Dimension for the Testing Dataset: ",lo3_data_test.shape)

Dimension for the Training Dataset: (189, 4)
Dimension for the Testing Dataset: (48, 4)

```

❖ Verifying and converting to stationarity:

To verify time series property of stationarity in the dataset variables, Augmented Dickey

Fuller test function is applied on the train dataset as below.

```
# Using function to run Augmented Dickey-Fuller Test (ADF Test) to check for stationarity in the
from statsmodels.tsa.stattools import adfuller

def Stationary_test_ADF(dataset):
    for i in dataset.columns:
        print('\nADF test for Column '+i+':\n')
        adf_test = adfuller(dataset[i], autolag='AIC')
        adf = pandas.Series(adf_test[0:4], index = ['Test Statistic :','p-value :','No_Lags :',
        'No_Obsv :'])
        for key, value in adf_test[4].items():
            adf['Critical Value ('+key+')'] = value
        print(adf)
    return

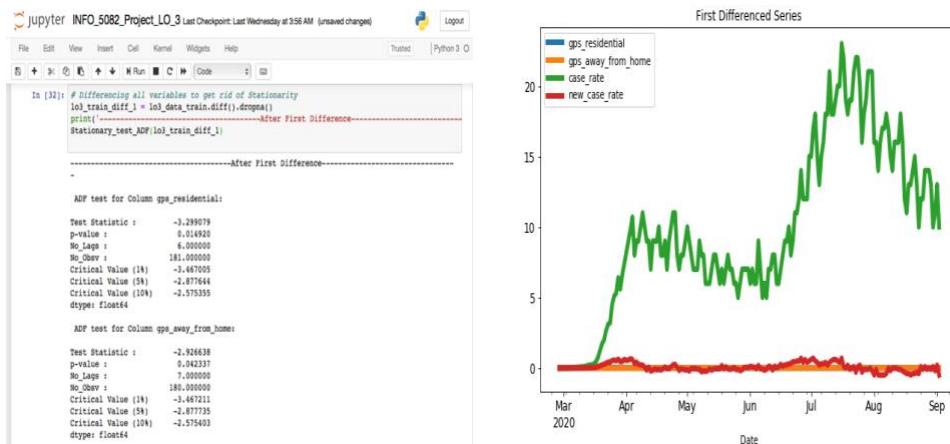
Stationary_test_ADF(lo3_data_train)

ADF test for Column gpa_residential:
Test Statistic : -3.051815
p-value : 0.030331
No_Lags : 8.000000
No_Observ : 180.000000
Critical Value (1%) -3.467211
Critical Value (5%) -2.877735
Critical Value (10%) -2.575403
dtype: float64

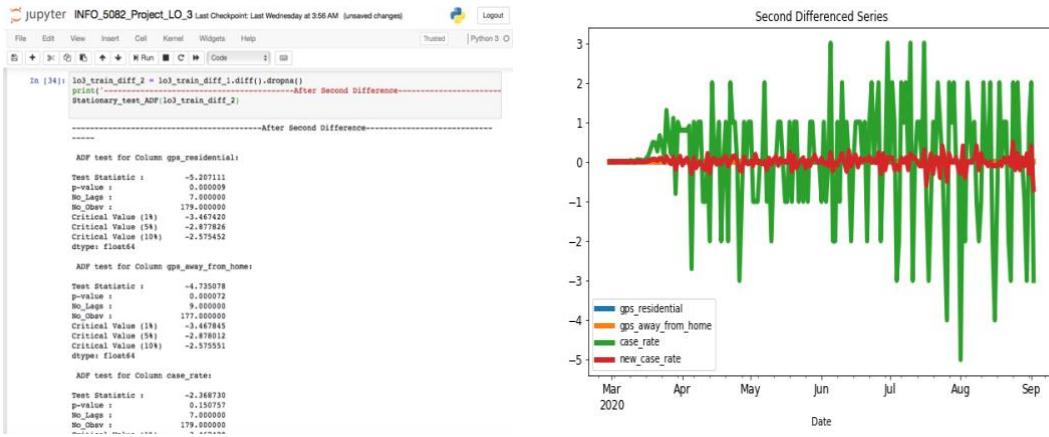
ADF test for Column gpa_away_from_home:
Test Statistic : -2.927372
p-value : 0.042256
No_Lags : 8.000000
No_Observ : 180.000000
```

As the results ADF test shown that data is not stationary, to convert the time series dataset into stationary Differencing method has been implemented in three orders.

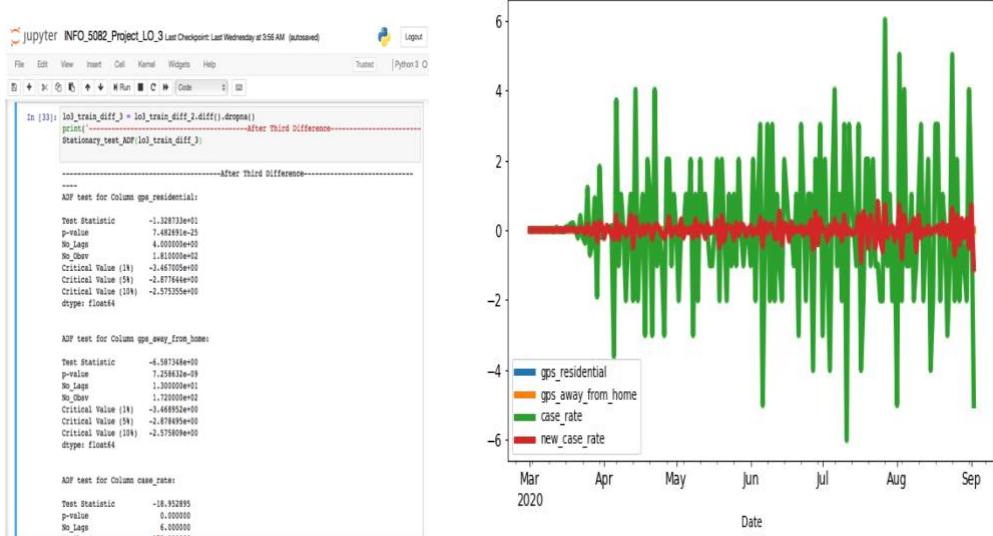
After first differencing, ADF test results shown as non-stationary, so second differencing is applied.



ADF test results from below second difference are also not shown as stationary, so third differencing is performed.



ADF results from third differencing has shown as stationary.



Now the data is in stationary and ready for time series modelling.

❖ Cointegration Testing using Johansen's function:

To select the appropriate model, dataset has been tested using Johansen's cointegration among the variables to avoid the information loss taking Eigen Test statistic with 95% confidence interval.

```

jupyter INFO_5082_Project_LO_3 Last Checkpoint: Last Wednesday at 3:56 AM (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 Logout

In [38]: from statsmodels.tsa.vector_ar.vecm import coint_johansen
#Using function to run cointegration test on all time series variables in the dataset.
def cointegration_test(df, alpha=0.05):
    out = coint_johansen(df, 1, 3)
    eigs = out.lr2
    cvms = out.cvm[:, 1]

    print('TS variable | Max Eigen Test Statistic > or < C(95%) | Signif')
    for col, eig, cvm in zip(df.columns, eigs, cvms):
        print(col, '-', round(eig, 2), ">" if eig > cvm else "<", cvm, ' -> ', eig > cvm)

cointegration_test(lo3_train_diff_3)

TS variable | Max Eigen Test Statistic > or < C(95%) | Signif
gps_residential - 145.29 > 30.8151 -> True
gps_away_from_home - 106.82 > 24.2522 -> True
case_rate - 93.82 > 17.1481 -> True
new_case_rate - 85.85 > 3.8415 -> True

```

From the results generated, we can see that all the time series variables are cointegrated meaning in long run the time series variables have significant correlations, so this dataset is modelled using VECM model to have little loss of information than when modelled with VAR.

❖ Granger Causality Matrix:

From the granger causality matrix applied we can see that there is no causality or linearity between mobility data columns with covid-19 infection case rates variables, so this dataset is not be useful for analyzing the people's mobility across nationwide with covid-19 case rates.

	gps_residential_x	gps_away_from_home_x	case_rate_x	new_case_rate_x
gps_residential_y	1.0000	0.0056	0.3679	0.0911
gps_away_from_home_y	0.0001	1.0000	0.3587	0.0835
case_rate_y	0.1637	0.2347	1.0000	0.0044
new_case_rate_y	0.6076	0.6384	0.1743	1.0000

❖ Model 2:

So, time series variables of people's google mobility in retail and recreation, pharmacy, parks, transit stations and workplaces along with previous covid-19 infection rates' dataset is taken for studying the third objective.

❖ Data preparation and cleaning

- Collection and uploading the google mobility dataset:

```
#calling above function to read csv data with a file path
google_columns = [ 'gps_residential','gps_away_from_home']
file_path = "/Users/harikaporalla/Desktop/Harika's_MS_Courses/INFO_5082/Project/Project_Data/Go
google_data = panda.read_csv(file_path,skiprows = [1,2,3])
google_data= google_data.drop(google_columns,axis=1)

#printing the results of csv data reading
google_data.head(10)
```

Out[2]:

	year	month	day	gps_retail_and_recreation	gps_grocery_and_pharmacy	gps_parks	gps_transit_stations	gps_workplaces
0	2020	2	27	0.0529	0.0157	0.121	0.0300	0.0214
1	2020	2	28	0.0614	0.0229	0.130	0.0357	0.0243
2	2020	2	29	0.0686	0.0286	0.127	0.0386	0.0243
3	2020	3	1	0.0771	0.0371	0.120	0.0414	0.0257
4	2020	3	2	0.0843	0.0443	0.121	0.0429	0.0257
5	2020	3	3	0.0943	0.0571	0.137	0.0443	0.0257
6	2020	3	4	0.0957	0.0629	0.154	0.0429	0.0271
7	2020	3	5	0.0957	0.0686	0.167	0.0400	0.0286
8	2020	3	6	0.0914	0.0686	0.170	0.0357	0.0286
9	2020	3	7	0.0886	0.0700	0.181	0.0357	0.0300

❖ Datetime indexing for above dataset:

```
In [8]: google_data.insert(0,'Date',['-' .join(i) for i in zip(google_data['year'].map(str),google_data['month'].map(str),google_data['day'].map(str))])
google_data['Date'] = panda.to_datetime(google_data['Date'], format='%Y-%m-%d')
google_data.set_index('Date', inplace=True, drop=False)
google_data = google_data.drop(google_data.columns[[0,1, 2, 3]], axis=1)

google_data.head(10)
```

Out[8]:

Date	gps_retail_and_recreation	gps_grocery_and_pharmacy	gps_parks	gps_transit_stations	gps_workplaces
2020-02-27	0.0529	0.0157	0.121	0.0300	0.0214
2020-02-28	0.0614	0.0229	0.130	0.0357	0.0243
2020-02-29	0.0686	0.0286	0.127	0.0386	0.0243
2020-03-01	0.0771	0.0371	0.120	0.0414	0.0257
2020-03-02	0.0843	0.0443	0.121	0.0429	0.0257
2020-03-03	0.0943	0.0571	0.137	0.0443	0.0257
2020-03-04	0.0957	0.0629	0.154	0.0429	0.0271
2020-03-05	0.0957	0.0686	0.167	0.0400	0.0286
2020-03-06	0.0914	0.0686	0.170	0.0357	0.0286
2020-03-07	0.0886	0.0700	0.181	0.0357	0.0300

EDA

- ❖ Null values verification:

The screenshot shows a Jupyter Notebook interface with the title "INFO_5082_Project_LO_3_model_2". In the code cell (In [4]), the following Python code is run:

```
#Using function to print number of missing values in the dataset
def verifyMissingValuesInData(fileData):
    print(fileData.isnull().sum())
    return

#Calling above function to get total missing values in each data feature before cleaning
print("Total Missing or 'NaN' values in Google Mobility Data are: \n")
verifyMissingValuesInData(google_data)

google_data.tail(10)
```

The output (Out[4]) shows the total missing or 'NaN' values in each data feature before cleaning:

```
Total Missing or 'NaN' values in Google Mobility Data are:
year          0
month         0
day           0
gps_retail_and_recreation  0
gps_grocery_and_pharmacy   0
gps_parks      0
gps_transit_stations     0
gps_workplaces    0
dtype: int64
```

Below this, a table of data is displayed:

	year	month	day	gps_retail_and_recreation	gps_grocery_and_pharmacy	gps_parks	gps_transit_stations	gps_workplaces
258	2020	11	11	-0.173	-0.0814	0.0486	-0.333	-0.261
259	2020	11	12	-0.180	-0.0857	0.0414	-0.333	-0.253
260	2020	11	13	-0.181	-0.0843	0.0171	-0.336	-0.254
261	2020	11	14	-0.184	-0.0843	-0.0129	-0.339	-0.254
262	2020	11	15	-0.187	-0.0843	-0.0429	-0.343	-0.256
263	2020	11	16	-0.189	-0.0829	-0.0529	-0.344	-0.256
264	2020	11	17	-0.191	-0.0829	-0.0671	-0.347	-0.257
...

- Summary of the above Dataset description:

The screenshot shows a Jupyter Notebook interface with the title "INFO_5082_Project_LO_3_model_2". In the code cell (In [6]), the following Python code is run:

```
#Using functions to describe the dataset
def descData(fileData,name):
    print("\ndimension of the "+name+" dataset: \n",fileData.shape)
    print("\nData Types of the "+name+" dataset: \n",fileData.dtypes)
    print("\nView of the first 10 records of "+name+" dataset: \n",fileData.head(10))
    print("\nSummary of the "+name+" dataset: \n",fileData.describe())
    return

#Calling above function to print dataset description
descData(google_data,"Google Mobility")
```

The output (Out[6]) shows the dimension, data types, view of the first 10 records, and summary statistics of the Google Mobility dataset:

```
Dimension of the Google Mobility dataset:
(268, 8)

Data Types of the Google Mobility dataset:
year          int64
month         int64
day           int64
gps_retail_and_recreation  float64
gps_grocery_and_pharmacy   float64
gps_parks      float64
gps_transit_stations     float64
gps_workplaces    float64
dtype: object

View of the first 10 records of Google Mobility dataset:
   year  month  day  gps_retail_and_recreation  gps_grocery_and_pharmacy \
0  2020      2    27            0.0529             0.0157
1  2020      2    28            0.0614             0.0229
2  2020      2    29            0.0686             0.0286
3  2020      3     1            0.0771             0.0371
4  2020      3     2            0.0843             0.0443
5  2020      3     3            0.0943             0.0571
6  2020      3     4            0.0957             0.0629
7  2020      3     5            0.0957             0.0686
   ... ... ... ... ... ...
```

Whole combined dataset for analysis and rectifying the null values:

jupyter INFO_5082_Project_LO_3_mod... Last Checkpoint: Last Wednesday at 3:57 AM (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 O

In [10]: `lo3_m2_data = pd.concat([google_data, covid_cases_df], axis=1, sort=False)`
lo3_m2_data.head(10)

Out[10]:

Date	gps_retail_and_recreation	gps_grocery_and_pharmacy	gps_parks	gps_transit_stations	gps_workplaces	case_rate	new_c
2020-01-27	NaN	NaN	NaN	NaN	NaN	NaN	0.00152
2020-01-28	NaN	NaN	NaN	NaN	NaN	NaN	0.00152
2020-01-29	NaN	NaN	NaN	NaN	NaN	NaN	0.00152
2020-01-30	NaN	NaN	NaN	NaN	NaN	NaN	0.00183
2020-01-31	NaN	NaN	NaN	NaN	NaN	NaN	0.00213
2020-02-01	NaN	NaN	NaN	NaN	NaN	NaN	0.00244
2020-02-02	NaN	NaN	NaN	NaN	NaN	NaN	0.00335
2020-02-03	NaN	NaN	NaN	NaN	NaN	NaN	0.00335
2020-02-04	NaN	NaN	NaN	NaN	NaN	NaN	0.00335
2020-02-05	NaN	NaN	NaN	NaN	NaN	NaN	0.00366

jupyter INFO_5082_Project_LO_3_model_2 Last Checkpoint: Last Wednesday at 3:57 AM (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 O

In [12]: `print("\nVerifying Missing values in whole dataset: \n")
verifyMissingValuesInData(lo3_m2_data)
lo3_m2_data = lo3_m2_data.dropna()

print(lo3_m2_data.info())

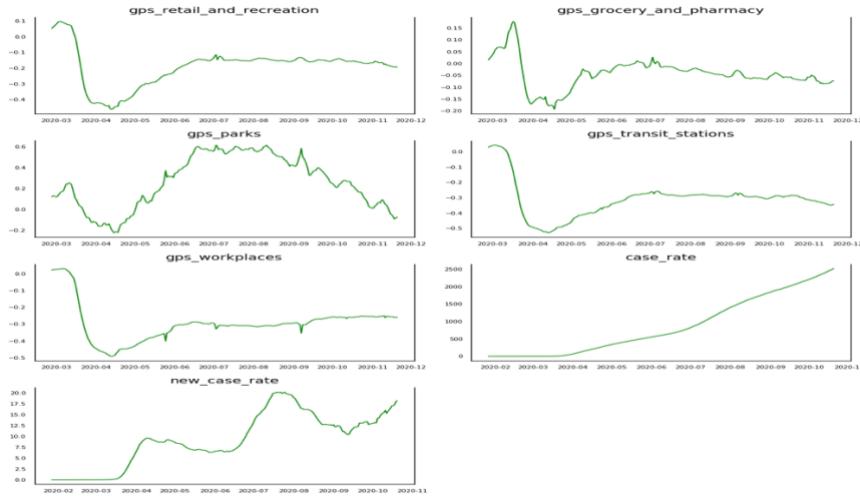
verifyMissingValuesInData(lo3_m2_data)`

Verifying Missing values in whole dataset:

```
gpe_retail_and_recreation    31
gpe_grocery_and_pharmacy     31
gpe_parks                     31
gpe_transit_stations          31
gpe_workplaces                 31
case_rate                      31
new_case_rate                  31
dtype: int64
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 137 entries, 2020-02-27 to 2020-10-20
Freq: D
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   gpe_retail_and_recreation  237 non-null float64
 1   gpe_grocery_and_pharmacy   237 non-null float64
 2   gpe_parks                  237 non-null float64
 3   gpe_transit_stations       237 non-null float64
 4   gpe_workplaces              237 non-null float64
 5   case_rate                  237 non-null float64
 6   new_case_rate               237 non-null float64
dtypes: float64(7)
memory usage: 14.8 KB
None
gpe_retail_and_recreation    0
gpe_grocery_and_pharmacy     0
gpe_parks                     0

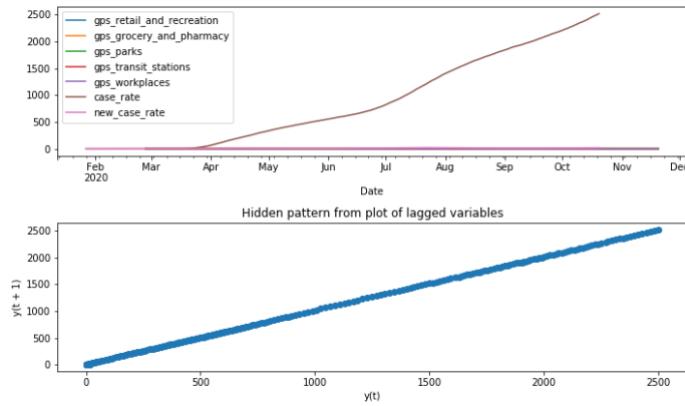
```

- Univariate plots of each time series variables:



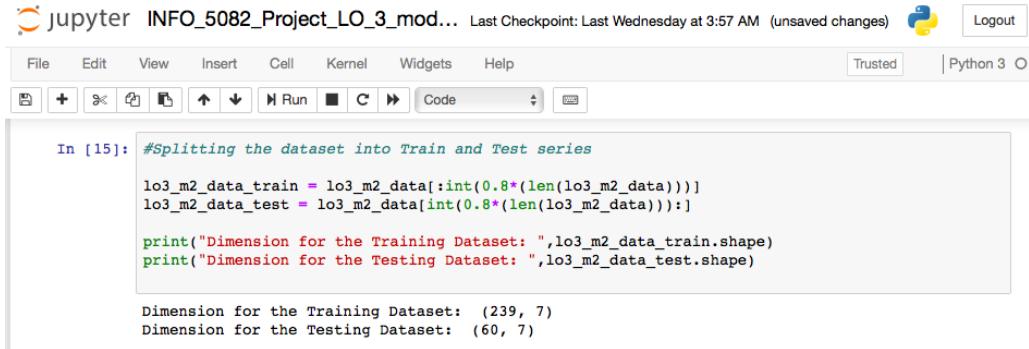
We can see the a slightly positive trend pattern in all variables except gps_residential from the month April, 2020.

- Plotting the lagged variables:



From the above plot of lag variables, there are no outliers in the dataset. Also, the positive trend pattern in the dataset shows the linearity with of time series variables with its lagged variable. Vector Autocorrelation (VAR) will be applied if there is no cointegration among the variables in dataset.

- ❖ Splitting of the dataset into training and testing



The screenshot shows a Jupyter Notebook interface with the title "jupyter INFO_5082_Project_LO_3_mod...". The code in cell In [15] is as follows:

```
#Splitting the dataset into Train and Test series
lo3_m2_data_train = lo3_m2_data[:int(0.8*(len(lo3_m2_data)))]
lo3_m2_data_test = lo3_m2_data[int(0.8*(len(lo3_m2_data))):]

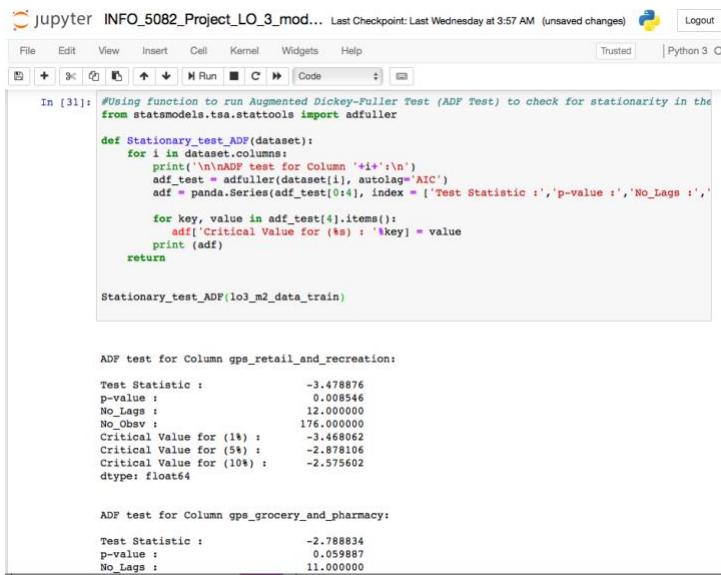
print("Dimension for the Training Dataset: ",lo3_m2_data_train.shape)
print("Dimension for the Testing Dataset: ",lo3_m2_data_test.shape)
```

The output of the code is:

```
Dimension for the Training Dataset: (239, 7)
Dimension for the Testing Dataset: (60, 7)
```

- ❖ Verifying and converting to stationarity:

Using Augmented Dickey Fuller test function to test the stationarity in the train dataset as below.



The screenshot shows a Jupyter Notebook interface with the title "jupyter INFO_5082_Project_LO_3_mod...". The code in cell In [31] is as follows:

```
#Using function to run Augmented Dickey-Fuller Test (ADF Test) to check for stationarity in the
from statsmodels.tsa.stattools import adfuller

def Stationary_test_ADF(dataset):
    for i in dataset.columns:
        print('\n\nADF test for Column '+i+'\n')
        adf_test = adfuller(dataset[i], autolag='AIC')
        adf = pandas.Series(adf_test[0:4], index = ['Test Statistic ','p-value ','No_Lags ',''])

        for key, value in adf_test[4].items():
            adf['Critical Value for (%) : ' + key] = value
        print (adf)
    return

Stationary_test_ADF(lo3_m2_data_train)
```

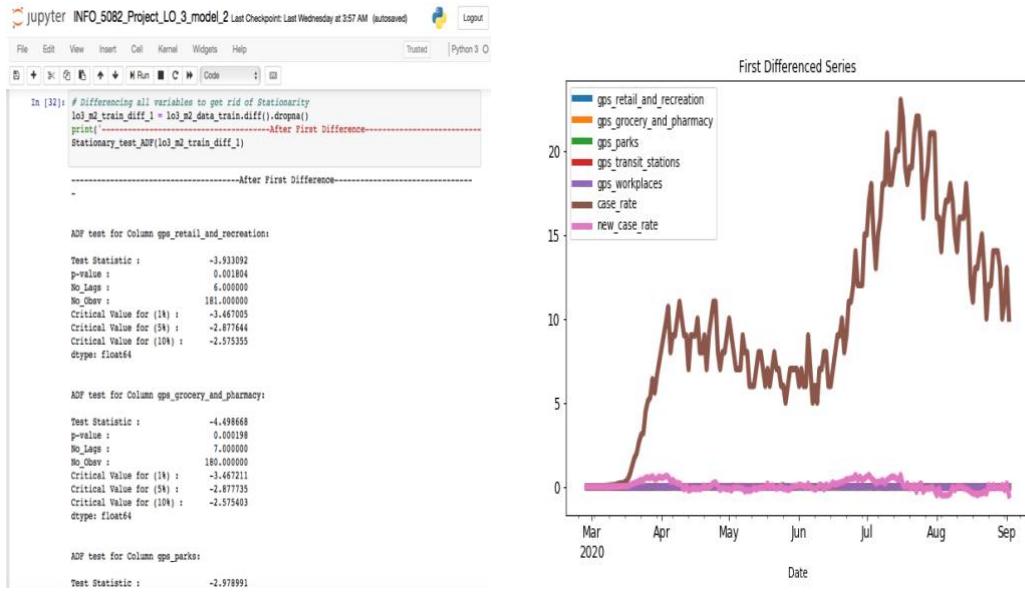
The output of the code shows the ADF test results for two columns:

```
ADF test for Column gps_retail_and_recreation:
Test Statistic : -3.478576
p-value : 0.008546
No_Lags : 12.000000
No_Obsv : 176.000000
Critical Value for (1%) : -3.468062
Critical Value for (5%) : -2.878106
Critical Value for (10%) : -2.575602
dtype: float64

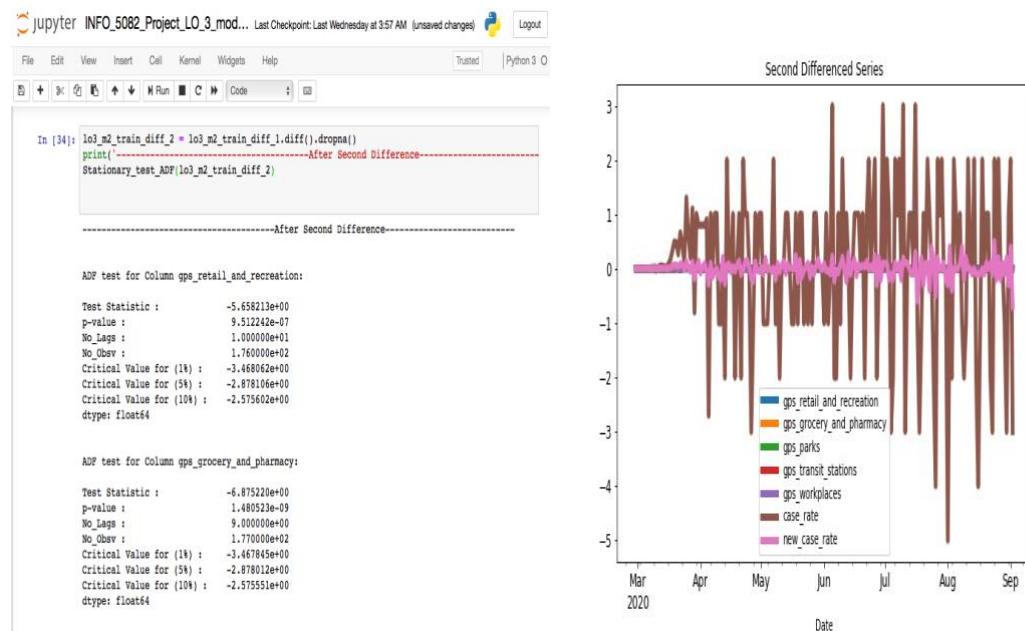
ADF test for Column gps_grocery_and_pharmacy:
Test Statistic : -2.788834
p-value : 0.059887
No_Lags : 11.000000
```

ADF test results have shown that data is not stationary, so dataset is converted into stationary using Differencing method in three orders.

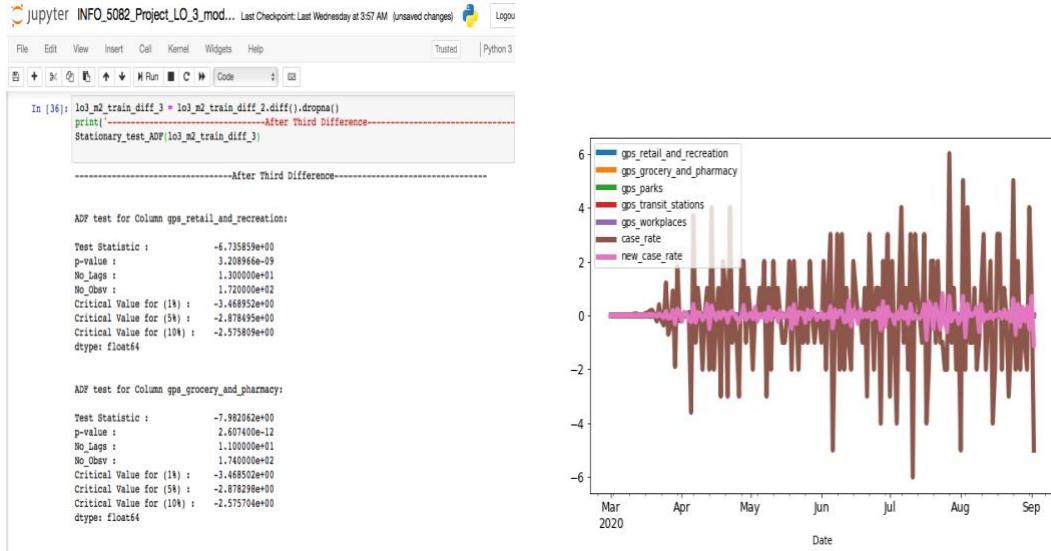
But first differencing ADF test results also have shown as non-stationary, so second differencing is applied.



Second differenced ADF test results from below also shown as non-stationary, so third differencing is applied.



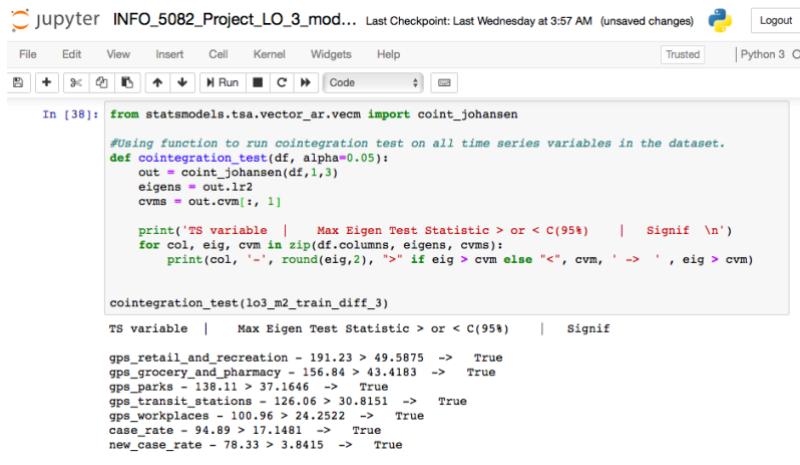
And ADF test results from third differencing has shown as stationary.



Now the data is in stationary and ready for time series modelling.

❖ Cointegration Testing using Johansen's function:

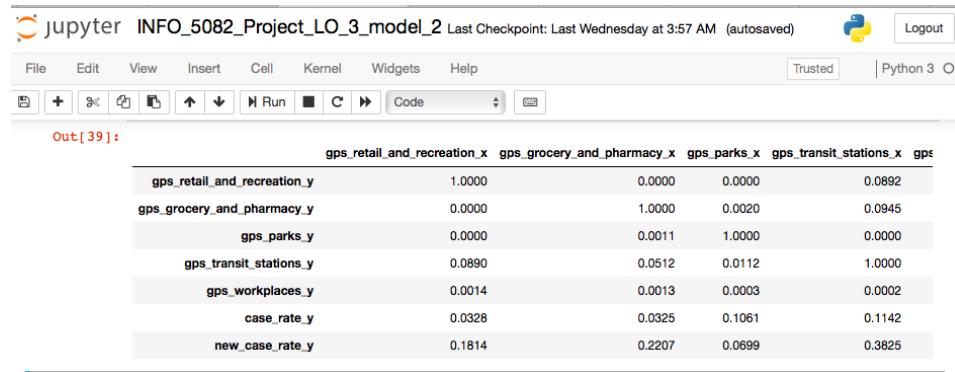
Dataset has been tested using Johansen's cointegration among the variables to avoid the information loss taking Eigen Test statistic with 95% confidence interval.



From above results, we can see that all the time series variables are cointegrated meaning in long run the time series variables have significant correlations, so this dataset is modelled using VECM model to have avoid the loss of information occurred VAR modeling.

❖ Granger Causality Matrix:

Granger's causality matrix is applied and observed some causalities or linearity between mobility data columns with covid-19 infection case rate variables, so this dataset is modelled using VECM for the analysis.



The screenshot shows a Jupyter Notebook interface with a toolbar at the top. The notebook title is "INFO_5082_Project_LO_3_model_2" and it indicates "Last Checkpoint: Last Wednesday at 3:57 AM (autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, Logout, and various cell type icons. Below the toolbar, a code cell is shown with the output labeled "Out[39]:". The output is a table representing a Granger Causality Matrix:

	gps_retail_and_recreation_x	gps_grocery_and_pharmacy_x	gps_parks_x	gps_transit_stations_x	gps_y
gps_retail_and_recreation_y	1.0000	0.0000	0.0000	0.0892	
gps_grocery_and_pharmacy_y	0.0000	1.0000	0.0020	0.0945	
gps_parks_y	0.0000	0.0011	1.0000	0.0000	
gps_transit_stations_y	0.0890	0.0512	0.0112	1.0000	
gps_workplaces_y	0.0014	0.0013	0.0003	0.0002	
case_rate_y	0.0328	0.0325	0.1061	0.1142	
new_case_rate_y	0.1814	0.2207	0.0699	0.3825	

Data Analytics

Time series is defined as a sequential value of a variable arranged in an orderly fashion spaced at equal time intervals. Mostly applied for Stock Market Forecasting, Workload Projections, Sales Forecasting and Economic Forecasting, etc., to get deep insights from underlying patterns in the data and also for model fitting for forecasting and monitoring.

There are so many techniques available for time series analysis, of which three main techniques to model fit a time series analysis are Box-Jenkins ARIMA Models, Box-Jenkins Multivariate Models and Holt-Winters Exponential Smoothing. Selection of the technique or method appropriate for analysis depends on the user's application and goals.

The AutoRegressive Moving Average Vector or VAR model can be applied for multivariate time series analysis where each times series variable is the vector of number of other times series variables in the dataset. Each variable in this analysis has a linearity with its own past lag values and also with the past lags of other variables.

VAR model with no mean and stationary multivariate time series is given as,
 $x_t = (x_{1t}, x_{2t}, x_{3t}, \dots, x_{nt})^T, -\infty < t < \infty$ (or) $x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \dots + \phi_p x_{t-p} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \dots - \theta_q a_{t-q}$, where x_t and a_t are $n \times 1$ column vectors and a_t represents the random component called white noise.

Here, $\phi_k = \{\phi_{k,jj}\}$ where $k = 1, 2, \dots, p$ (order of lags) and $\theta_k = \{\theta_{k,jj}\}$ where $k = 1, 2, \dots, q$ are the $n \times n$ autoregressive and moving average parameters. In above equation for each variable error or residual are represented by the past lag values of other variables so the these correlations can in general be spurious correlations because of any causal relationship.

When variables in a VAR are cointegrated we apply Vector error correction (VEC) model and VECM for two variables x and y is given as,

$\Delta y_t = \beta_{y0} + \beta_{y1}\Delta y_{t-1} + \dots + \beta_{yp}\Delta y_{t-p} + \gamma_{y1}\Delta x_{t-1} + \dots + \gamma_{yp}\Delta x_{t-p} - \lambda_y(y_{t-1} - \alpha_0 - \alpha_1 x_{t-1}) + v_t^y$ and

$\Delta x_t = \beta_{x0} + \beta_{x1}\Delta x_{t-1} + \dots + \beta_{xp}\Delta x_{t-p} + \gamma_{x1}\Delta y_{t-1} + \dots + \gamma_{xp}\Delta y_{t-p} - \lambda_x(y_{t-1} - \alpha_0 - \alpha_1 x_{t-1}) + v_t^x$,

Here, λ_x and λ_y are the error correction params that define how the variables x and y react in long run and $y_t = \alpha_0 + \alpha_1 x_t$ is the cointegration relationship.

For identifying the structural shocks and their dynamic additional assumptions are required. VAR can be used for two variables systems without further assumptions and can find the causality direction between x and y using granger's causality analysis. But VAR when applied for more than two variables and one lag the computational can be very complex. Thus, the automated tasks from python functions can make this task easy.

Granger's Causality: Granger's concept that "Correlation does not imply Causality" shows the directions and significance of causal relationships between variables x and y. It's null hypothesis is stated as "x does not cause y" which can be tested with p-value significance.

In this project, to analyze each study objective, multivariate time series analysis is performed by selecting appropriate models from VAR and VECM for fitting as per EDA results on taken data.

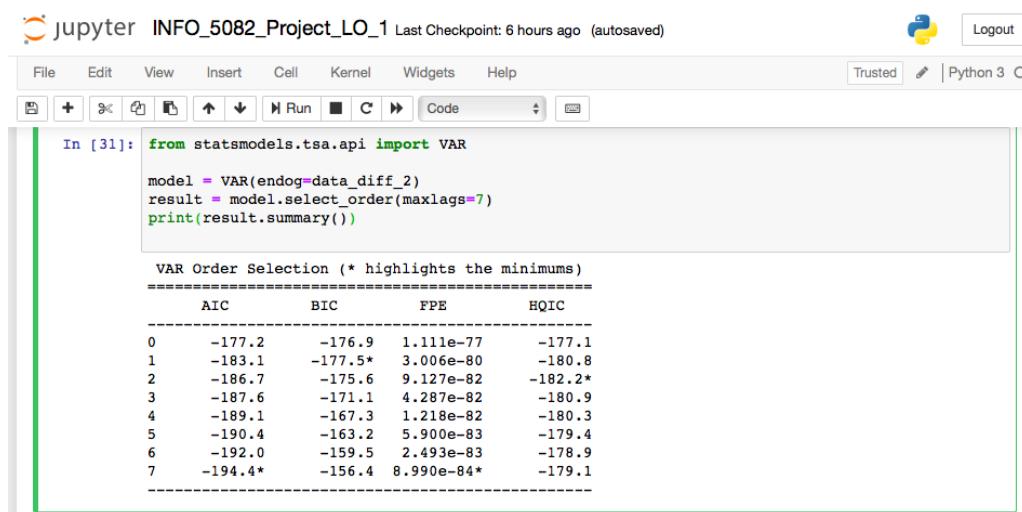
Data analysis for small business revenue

For the first study objective to analyze the small business revenues after the EDA results both the multivariate Autoregression model (VAR) and vector error correction models are used for model fitting to get the model with better accuracy.

❖ VAR model Analysis:

To gain the minimal information loss, Akaike Information criteria (AIC) is used in selecting the small order of lag for model fitting to forecast.

VAR model is iteratively fitted with increasing lag order till maxlag = 7 to pick the order with minimum AIC, as below



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter INFO_5082_Project_LO_1 Last Checkpoint: 6 hours ago (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Run, Cell, Kernel, Widgets, Help, Trusted, Python 3
- Code Cell:** In [31]:

```
from statsmodels.tsa.api import VAR
model = VAR(endog=data_diff_2)
result = model.select_order(maxlags=7)
print(result.summary())
```
- Output Cell:** VAR Order Selection (* highlights the minimums)
=====
- Data Output:**

	AIC	BIC	FPE	HQIC
0	-177.2	-176.9	1.111e-77	-177.1
1	-183.1	-177.5*	3.006e-80	-180.8
2	-186.7	-175.6	9.127e-82	-182.2*
3	-187.6	-171.1	4.287e-82	-180.9
4	-189.1	-167.3	1.218e-82	-180.3
5	-190.4	-163.2	5.900e-83	-179.4
6	-192.0	-159.5	2.493e-83	-178.9
7	-194.4*	-156.4	8.990e-84*	-179.1

From the results, model with lag order = 7 has the minimal AIC values. So, the VAR model is fitted with ‘lag order = 7’ and summary of output can be seen as below.

```
In [32]: #Fitting to a VAR model
model_fit = model.fit(7)

#Printing the summary of the VAR model results
model_fit.summary()

Out[32]: Summary of Regression Results
=====
Model: VAR
Method: OLS
Date: Tue, 08, Dec, 2020
Time: 10:23:58

No. of Equations: 19.0000 BIC: -156.415
Nobs: 231.000 HQIC: -179.053
Log likelihood: 18766.4 FPE: 8.98995e-84
AIC: -194.356 Det(Omega_mle): 1.50946e-87

Results for equation spend_all
=====
      coefficient    std. error      t-stat      prob
-----
const        -0.000087    0.000483     -0.180     0.857
L1.spend_all   -0.973415    0.147639     -6.593     0.000
L1.spend_apg   -0.137930    0.128787     -1.071     0.284
...
```

To check the correlation in the error residuals, durbin_watson function is used and the results are generated as below.

```
In [33]: from statsmodels.stats.stattools import durbin_watson
output_errors = durbin_watson(model_fit.resid)

print("-----Residuals----- \n")
for col, val in zip(lol_data.columns, output_errors):
    print(col + " -> " + str(val))

-----Residuals-----

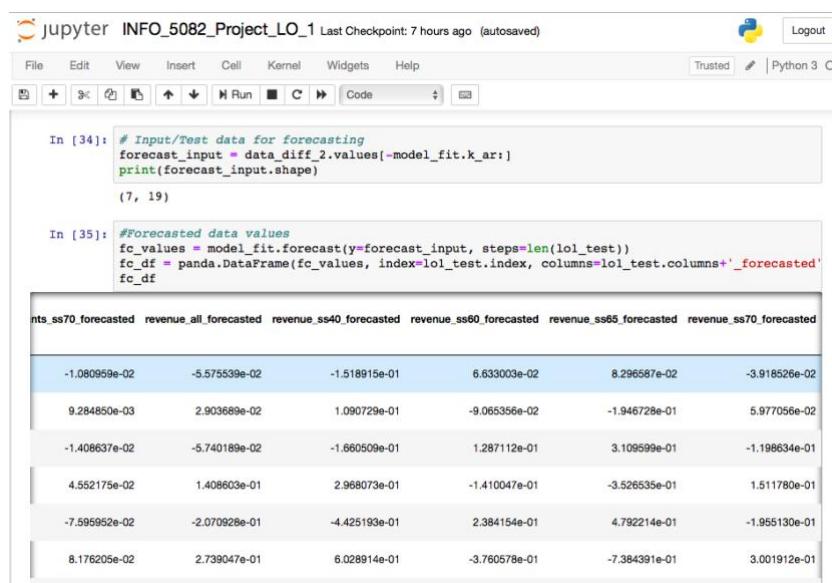
spend_all -> 2.1490602629639555
spend_apg -> 2.0181227598509364
spend_aer -> 2.0005948827831386
spend_grt -> 2.0483859313246334
spend_hci -> 2.0835529112995377
spend_acf -> 2.0214397734597225
spend_twi -> 2.3141742689059757
spend_retail_no_grocery -> 2.170134305022987
spend_retail_w_grocery -> 2.165872726246697
merchants_all -> 2.100116444447363
merchants_ss40 -> 2.0504474082381337
merchants_ss60 -> 2.0085631954983363
merchants_ss65 -> 1.9829072540961759
merchants_ss70 -> 2.1360733834061305
revenue_all -> 1.918972578195589
revenue_ss40 -> 2.161196480516894
revenue_ss60 -> 1.9794955574878432
revenue_ss65 -> 2.045670639337524
revenue_ss70 -> 1.8185221723747447
```

As per the durbin_watson residuals, if the error values are close to 4 then a negative correlation is left in the data, when the error values are close to 0 implies a positive correlation while error values close to 2 then there is no serial correlation in the dataset.

From above, the residuals are observed to close to 2 meaning there is no serial correlation pattern left in the dataset for further analysis.

❖ Model Forecasting the test outputs:

VAR model is provided with input data for essential past lag values and the forecasted data values generated below are based on differenced train data.



The screenshot shows a Jupyter Notebook interface with the title "jupyter INFO_5082_Project_LO_1 Last Checkpoint: 7 hours ago (autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. Below the toolbar, there are buttons for Run, Cell, Kernel, and Help. The code cell In [34] contains the following Python code:

```
# Input/Test data for forecasting
forecast_input = data_diff_2.values[-model_fit.k_ar:]
print(forecast_input.shape)
```

The output of this cell is (7, 19). The code cell In [35] contains the following Python code:

```
#Forecasted data values
fc_values = model_fit.forecast(y=forecast_input, steps=len(lol_test))
fc_df = pd.DataFrame(fc_values, index=lol_test.index, columns=lol_test.columns + '_forecasted'
fc_df
```

The output of this cell is a DataFrame with columns: nts_ss70_forecasted, revenue_all_forecasted, revenue_ss40_forecasted, revenue_ss60_forecasted, revenue_ss65_forecasted, and revenue_ss70_forecasted. The data is as follows:

	nts_ss70_forecasted	revenue_all_forecasted	revenue_ss40_forecasted	revenue_ss60_forecasted	revenue_ss65_forecasted	revenue_ss70_forecasted
-1.080959e-02	-5.575539e-02	-1.518915e-01	6.633003e-02	8.296587e-02	-3.918526e-02	
9.284850e-03	2.903689e-02	1.090729e-01	-9.065356e-02	-1.946728e-01	5.977056e-02	
-1.408637e-02	-5.740189e-02	-1.660509e-01	1.287112e-01	3.109599e-01	-1.198634e-01	
4.552175e-02	1.408603e-01	2.968073e-01	-1.410047e-01	-3.526535e-01	1.511780e-01	
-7.595952e-02	-2.070928e-01	-4.425193e-01	2.384154e-01	4.792214e-01	-1.955130e-01	
8.176205e-02	2.739047e-01	6.028914e-01	-3.760578e-01	-7.384391e-01	3.001912e-01	

So the values are inverted to remove the applied differencing to get original forecasted values as below

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter INFO_5082_Project_LO_1 Last Checkpoint: 7 hours ago (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3
- Code Cell:** In [38]:

```
# Inverting the Differencing Transformation
def invert_transformation(ds, df_forecast):
    for col in ds.columns:
        # Inverting 2nd Differencing
        df_forecast[str(col) + '_firstDiff'] = (ds[col].iloc[-1] - ds[col].iloc[-2]) + df_forecast[str(col) + '_firstDiff']

        # Inverting 1st Differencing
        df_forecast[str(col) + '_OrigForecast'] = ds[col].iloc[-1] + df_forecast[str(col) + '_firstDiff']

    return df_forecast

df_inverted = invert_transformation(lol_train, fc_df)

df_inverted
```
- Data Preview:** A table showing the first few rows of the inverted data frame. The columns are labeled: nue_ss60_firstDiff, revenue_ss60_OrigForecast, revenue_ss65_firstDiff, revenue_ss65_OrigForecast, revenue_ss70_firstDiff, and revenue_ss70_OrigForecast.

nue_ss60_firstDiff	revenue_ss60_OrigForecast	revenue_ss65_firstDiff	revenue_ss65_OrigForecast	revenue_ss70_firstDiff	revenue_ss70_OrigForecast
1.933003e-02	-1.596700e-01	4.196587e-02	-2.320341e-01	-8.185259e-03	-4.841853e-01
-7.132353e-02	-2.309935e-01	-1.527070e-01	-3.847411e-01	5.158530e-02	-4.326000e-01
5.738771e-02	-1.736058e-01	1.582530e-01	-2.264881e-01	-6.827810e-02	-5.008781e-01
-8.361704e-02	-2.572228e-01	-1.944006e-01	-4.208887e-01	8.289990e-02	-4.179782e-01
1.547984e-01	-1.024245e-01	2.848208e-01	-1.360679e-01	-1.126131e-01	-5.305913e-01
-2.212594e-01	-3.236838e-01	-4.536183e-01	-5.896862e-01	1.875781e-01	-3.430132e-01

❖ VECM Model Analysis:

Vector error correction model is fitted on the train data using lag order = 7 and cointegration rank = 3 with constant deterministic as the data has constant linearity pattern and the lower, upper and forecast values are predicted for test data with 0.05 significance as below,

The screenshot shows a Jupyter Notebook interface with the title "INFO_5082_Project_LO_1". The notebook has a "Trusted" status and is using Python 3. The code in cell In [42] imports `statsmodels.tsa.vector_ar` and fits a VECM model with 7 lags, 3 cointegrating ranks, and constant deterministic terms. The output (Out[42]) is a large array of numerical values representing the estimated parameters. Cell In [43] uses the fitted model to predict values for the test set, calculate confidence intervals, and print the results. The output (Out[43]) shows the forecast, lower, and upper bounds of the confidence intervals.

```
In [42]: from statsmodels.tsa.vector_ar import vecm
vecm_model = vecm.VECM(endog = lol_train, k_ar_diff = 7, coint_rank = 3, deterministic = "ci")
vecm_model_fit = vecm_model.fit()
vecm_model_fit.predict(steps=len(lol_test))

Out[42]: array([-1.06493944e-01, -1.39604077e-01, -6.15045410e-01, ...,
   -1.94793810e-01, -2.60006648e-01, -5.03160341e-01, ...,
   [-1.14975780e-01, -1.47382491e-01, -5.97090295e-01, ...,
   -2.21108951e-01, -3.19637292e-01, -4.77971424e-01], ...
   [-1.12629981e-01, -1.36876123e-01, -4.92703873e-01, ...,
   -1.57245366e-01, -2.00178994e-01, -5.63650833e-01],
   ...,
   [-1.53228148e+06, -1.22465981e+05, 5.07017947e+06, ...,
   6.51226109e+06, 1.44941051e+07, -8.80347800e+06], ...
   [ 2.16410307e+06, 1.72963688e+05, -7.16081996e+06, ...,
   -9.19752949e+06, -2.04706099e+07, 1.24335060e+07], ...
   [-3.05644999e+06, -2.44283627e+05, 1.01135139e+07, ...,
   1.29900417e+07, 2.89114679e+07, -1.75603433e+07]])

In [43]: forecast, lower, upper = vecm_model_fit.predict(len(lol_test), 0.05)
forecast_df = pd.DataFrame(forecast, index = lol_test.index, columns=lol_test.columns + '_pred'
lower_df = pd.DataFrame(lower, index = lol_test.index, columns=lol_test.columns + '_lower')
upper_df = pd.DataFrame(upper, index = lol_test.index, columns=lol_test.columns + '_upper')

print('lower bounds of confidence intervals:')
print(lower.round(3))
print('\npoint forecasts:')
print(forecast.round(3))
print('\nupper bounds of confidence intervals:')
print(upper.round(3))

lower bounds of confidence intervals:
[[-1.1400000e-01 -1.4900000e-01 -6.2800000e-01 ... -2.0500000e-01
 -2.7200000e-01 -5.0800000e-01]
```

Data Analysis for Employment Rate estimation

For the second study objective to analyze the estimate employment rates with covid-19 cases since the EDA results has shown linear pattern in the data and presence of cointegration among time series variables vector error correction model (VECM) is used for model fitting to get the model with no information loss and better accuracy.

❖ VECM Model Analysis:

vecm model from ‘statmodels.tsa.vector_ar’ is fitted on the train data using lag order = 7

and cointegration rank = 5 with constant deterministic as the data has constant linearity pattern

and forecasted values as below,

```
In [50]: vecm_model = vecm.VECM(endog = lo2_data_train, k_ar_diff = 7, coint_rank = 5, deterministic = 'c')
vecm_model_fit = vecm_model.fit()
vecm_model_fit.predict(steps=len(lo2_data_test))
```

```
[ 1.08868530e+01, -6.47495179e-02, -2.62621923e-02, -8.15334288e-02, -2.14592361e-01, 1.95528369e+03, 1.04224681e+01, [-7.49459519e-02, -6.38554780e-02, -2.69866282e-02, -8.18415755e-02, -2.11008215e-01, 1.96546436e+03, 1.00945111e+01, [-7.37527285e-02, -6.28237584e-02, -2.78916032e-02, -8.16399386e-02, -2.05064986e-01, 1.97199483e+03, 9.63535538e+00], [-7.26897789e-02, -6.19380591e-02, -2.84820842e-02, -8.10218858e-02, -1.96988061e-01, 1.97759754e+03, 9.37118042e+00], [-7.06419167e-02, -6.03182087e-02, -2.93842740e-02, -7.98700028e-02, -1.86427506e-01, 1.98349999e+03, 9.10850194e+00], [-6.75999273e-02, -5.83562921e-02, -2.98690745e-02, -7.84373902e-02, -1.73429338e-01, 1.99053805e+03, 8.62005042e+00], [-6.40531129e-02, -5.57320965e-02, -2.98332776e-02,
```

The predicted lower, upper forecasted values for test data with 0.05 significance are

given as below,

```
In [51]: forecast, lower, upper = vecm_model_fit.predict(len(lo2_data_test), 0.05)
forecast_df = pandas.DataFrame(forecast, index = lo2_data_test.index, columns=lo2_data_test.columns)
lower_df = pandas.DataFrame(lower, index = lo2_data_test.index, columns=lo2_data_test.columns)
upper_df = pandas.DataFrame(upper, index = lo2_data_test.index, columns=lo2_data_test.columns)
```

```
print('lower bounds of confidence intervals:')
print(lower.round(3))
print('\npoint forecasts:')
print(forecast.round(3))
print('\nupper bounds of confidence intervals:')
print(upper.round(3))
```

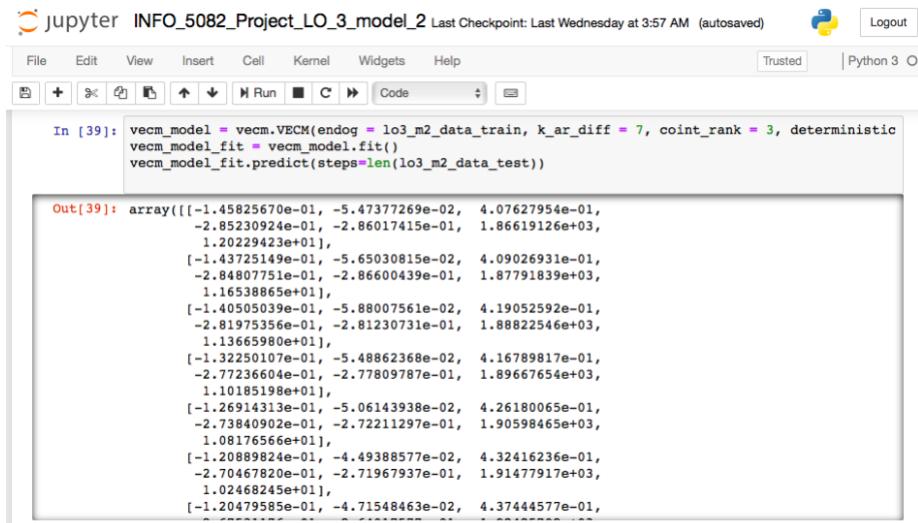
```
2.015217e+03 -1.205000e+00
[ 1.690000e-01 1.210000e-01 9.100000e-02 1.340000e-01 5.030000e-01
2.012597e+03 -2.004000e+00]
[ 1.960000e-01 1.440000e-01 1.090000e-01 1.600000e-01 5.630000e-01
2.006599e+03 -2.855000e+00]
[ 2.220000e-01 1.670000e-01 1.280000e-01 1.870000e-01 6.200000e-01
1.997891e+03 -3.788000e+00]
[ 2.480000e-01 1.900000e-01 1.480000e-01 2.140000e-01 6.740000e-01
1.988351e+03 -4.800000e+00]
[ 2.730000e-01 2.140000e-01 1.670000e-01 2.400000e-01 7.230000e-01
1.979518e+03 -5.870000e+00]
[ 2.970000e-01 2.370000e-01 1.860000e-01 2.650000e-01 7.680000e-01
1.971666e+03 -6.978000e+00]
[ 3.200000e-01 2.590000e-01 2.050000e-01 2.900000e-01 8.060000e-01
1.963430e+03 -8.117000e+00]
[ 3.400000e-01 2.810000e-01 2.220000e-01 3.120000e-01 8.360000e-01
1.952803e+03 -9.274000e+00]
[ 3.580000e-01 3.000000e-01 2.380000e-01 3.330000e-01 8.590000e-01
1.938731e+03 -1.045400e+01]
[ 3.730000e-01 3.170000e-01 2.520000e-01 3.510000e-01 8.730000e-01]
```

❖ Data Analysis for covid-19 infection rates with google mobility:

In the third study objective to analyze the covid-19 infection rates with people google mobility data, since the EDA results has shown linear pattern in the data and presence of cointegration among time series variables, again vector error correction model (VECM) is applied for model fitting to get the predictions with no information loss and better accuracy.

❖ VECM Model Analysis:

vecm model from ‘statmodels.tsa.vector_ar’ is fitted on the train data using lag order = 7 and cointegration rank = 3 with constant deterministic as the constant linearity pattern is observed in dataset and forecasted values as below,



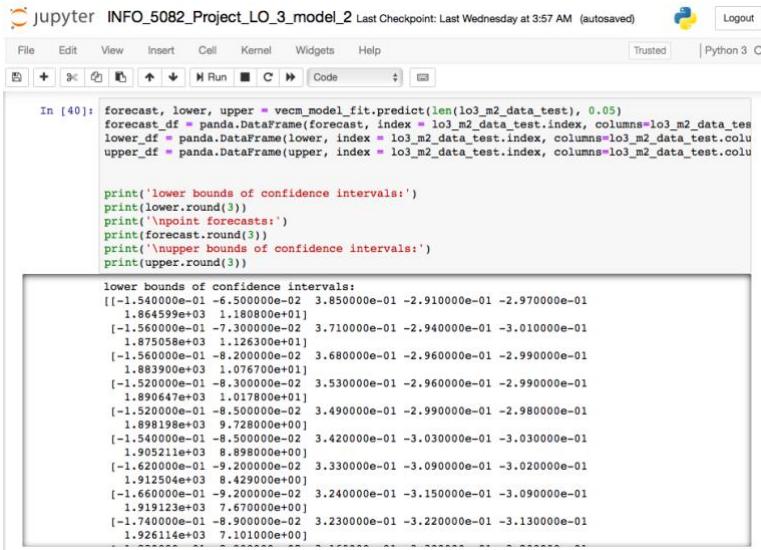
The screenshot shows a Jupyter Notebook interface with the title "INFO_5082_Project_LO_3_model_2". The notebook has a Python 3 kernel and is set to "Trusted". The code cell (In [39]) contains the following Python code:

```
vecm_model = vecm.VECM(endog = lo3_m2_data_train, k_ar_diff = 7, coint_rank = 3, deterministic='c')
vecm_model_fit = vecm_model.fit()
vecm_model_fit.predict(steps=len(lo3_m2_data_test))
```

The output cell (Out[39]) displays a large array of numerical values representing the forecasted values:

```
Out[39]: array([[-1.45825670e-01, -5.47377269e-02,  4.07627954e-01,
   -2.85230924e-01, -2.86017415e-01,  1.86619126e+03,
   1.20229423e+01],
  [-1.43725149e-01, -5.65030815e-02,  4.09026931e-01,
   -2.84807751e-01, -2.86600439e-01,  1.87791839e+03,
   1.16538865e+01],
  [-1.40505039e-01, -5.88007561e-02,  4.19052592e-01,
   -2.81975356e-01, -2.81230731e-01,  1.88822546e+03,
   1.13665980e+01],
  [-1.32250107e-01, -5.48862368e-02,  4.16789817e-01,
   -2.77236604e-01, -2.77809787e-01,  1.89667654e+03,
   1.10185198e+01],
  [-1.26914313e-01, -5.06143938e-02,  4.26180065e-01,
   -2.73840902e-01, -2.72211297e-01,  1.90598465e+03,
   1.08176566e+01],
  [-1.20889824e-01, -4.49388577e-02,  4.32416236e-01,
   -2.70467820e-01, -2.71967937e-01,  1.91477917e+03,
   1.02468245e+01],
  [-1.20479585e-01, -4.71548463e-02,  4.37444577e-01,
```

Also predicted lower, upper forecasted values for test data with 0.05 significance are given as below,



The screenshot shows a Jupyter Notebook interface with the title "jupyter INFO_5082_Project_LO_3_model_2 Last Checkpoint: Last Wednesday at 3:57 AM (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. A toolbar with various icons is at the top. The code cell contains Python code for forecasting and printing confidence intervals:

```
In [40]: forecast, lower, upper = vcm_model_fit.predict(len(lo3_m2_data_test), 0.05)
forecast_df = pandas.DataFrame(forecast, index = lo3_m2_data_test.index, columns=lo3_m2_data_test.columns)
lower_df = pandas.DataFrame(lower, index = lo3_m2_data_test.index, columns=lo3_m2_data_test.columns)
upper_df = pandas.DataFrame(upper, index = lo3_m2_data_test.index, columns=lo3_m2_data_test.columns)

print('lower bounds of confidence intervals:')
print(lower.round(3))
print('\npoint forecasts:')
print(forecast.round(3))
print('\nupper bounds of confidence intervals:')
print(upper.round(3))
```

The output cell displays the lower bounds of the confidence intervals for each data point in the test set. The output is a list of lists, where each inner list contains four numerical values representing the lower bound, forecast, upper bound, and a constant value (likely -1.0). The values are formatted in scientific notation.

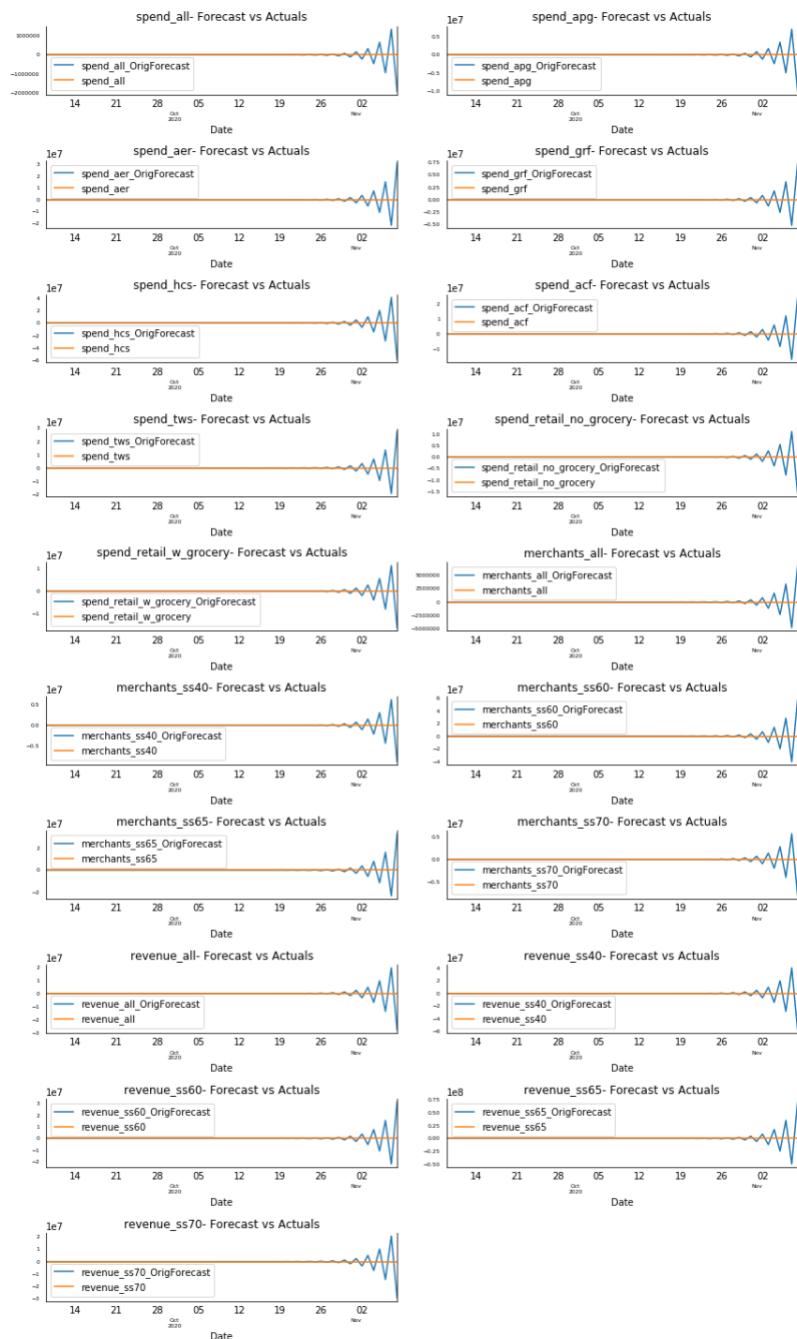
Index	Lower Bound	Forecast	Upper Bound	Constant
0	-1.54000e-01	-6.50000e-02	3.85000e-01	-2.97000e-01
1	1.86459e+03	1.18080e+01		
2	-1.56000e-01	-7.30000e-02	3.71000e-01	-2.94000e-01
3	1.87505e+03	1.12630e+01		
4	-1.56000e-01	-8.20000e-02	3.68000e-01	-2.96000e-01
5	1.88390e+03	1.07670e+01		
6	-1.52000e-01	-8.30000e-02	3.53000e-01	-2.96000e-01
7	1.89564e+03	1.01780e+01		
8	-1.52000e-01	-4.60000e-02	3.49000e-01	-2.99000e-01
9	1.89819e+03	9.72800e+00		
10	-1.54000e-01	-6.50000e-02	3.42000e-01	-3.03000e-01
11	1.90521e+03	8.89800e+00		
12	-1.62000e-01	-9.20000e-02	3.33000e-01	-3.09000e-01
13	1.91250e+03	8.42900e+00		
14	-1.66000e-01	-9.20000e-02	3.24000e-01	-3.15000e-01
15	1.91912e+03	7.67000e+00		
16	-1.74000e-01	-8.90000e-02	3.23000e-01	-3.22000e-01
17	1.92611e+03	7.10100e+00		

Data Visualization and Results Reports

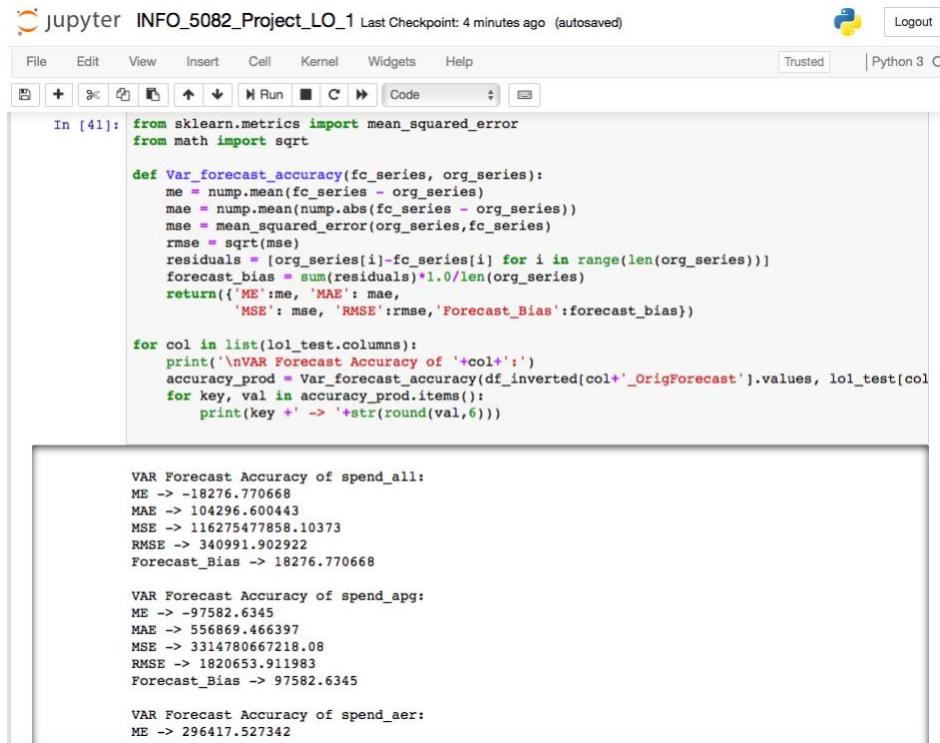
Data visualizations for first study objective – analyzing the small business revenues

❖ VAR model visualizations and evaluations:

- Forecasted values and Actual values are plotted as below,



- Evaluation of forecasted values for each time series variable is given as coded as below,



The screenshot shows a Jupyter Notebook interface with the title "jupyter INFO_5082_Project_LO_1 Last Checkpoint: 4 minutes ago (autosaved)". The notebook has a "Trusted" status and is running on "Python 3". The code in cell In [41] is as follows:

```

In [41]: from sklearn.metrics import mean_squared_error
         from math import sqrt

def Var_forecast_accuracy(fc_series, org_series):
    me = numpy.mean(fc_series - org_series)
    mae = numpy.mean(numpy.abs(fc_series - org_series))
    mse = mean_squared_error(org_series,fc_series)
    rmse = sqrt(mse)
    residuals = [org_series[i]-fc_series[i] for i in range(len(org_series))]
    forecast_bias = sum(residuals)*1.0/len(org_series)
    return({'ME':me, 'MAE': mae,
           'MSE': mse, 'RMSE':rmse,'Forecast_Bias':forecast_bias})

for col in list(lol_test.columns):
    print("\nVAR Forecast Accuracy of "+col+":")
    accuracy_prod = Var_forecast_accuracy(df_inverted[col+'_OrigForecast'].values, lol_test[col])
    for key, val in accuracy_prod.items():
        print(key + ' -> ' +str(round(val,6)))

```

The output of the code is displayed in the cell below:

```

VAR Forecast Accuracy of spend_all:
ME -> -18276.770668
MAE -> 104296.600443
MSE -> 116275477858.10373
RMSE -> 340991.902922
Forecast_Bias -> 18276.770668

VAR Forecast Accuracy of spend_apg:
ME -> -97582.6345
MAE -> 556869.466397
MSE -> 3314780667218.08
RMSE -> 1820653.911983
Forecast_Bias -> 97582.6345

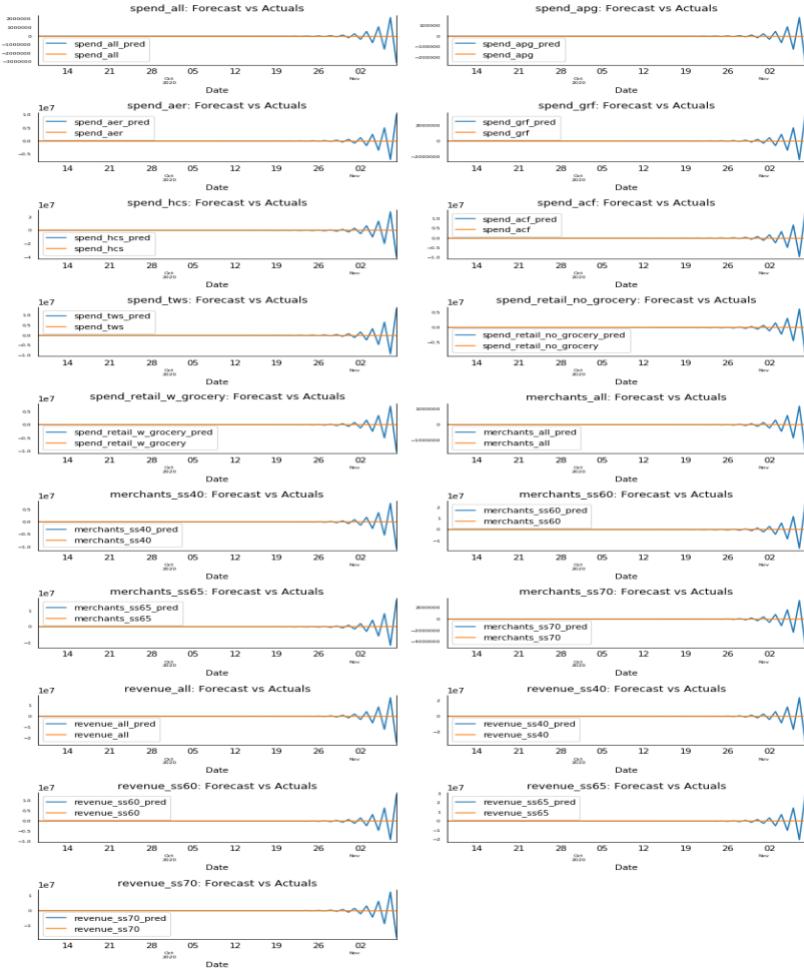
VAR Forecast Accuracy of spend_aer:
ME -> 296417.527342

```

Mean Forecast error (ME), Mean absolute error (MAE), Mean square error (MSE), root mean square error (RMSE) and bias are calculated and for 'revenue_all' RMSE = 5000351.65 and bias = 268006.16. So overall, this is not good model and the positive error from bias says that model has been under forecasted.

❖ VECM model visualizations and evaluations:

- Forecasted and actual values are plotted as below,



From the above graphs we cannot see much difference in forecasted values from VAR model.

Performing model evaluation using MAE, MSE , RMSE and bias as below.

The screenshot shows a Jupyter Notebook interface with the title "INFO_5082_Project_LO_1". The code cell contains a function to calculate forecast errors and metrics for multiple time series. The output cell displays the results for two series: "revenue_all" and "revenue_ss40".

```
def forecast_errors(fc_series,test_series):
    fc_errors = [test_series[i]-fc_series[i] for i in range(len(lol_test))]
    bias = sum(fc_errors)*1.0/len(lol_test)
    print(f'\n For "{test_series.name}"\n', '-'*50)
    print("\n Bias:",bias)

    mae = mean_absolute_error(test_series,fc_series)
    print("\n MAE:",mae)

    mse = mean_squared_error(test_series,fc_series)
    print("\n MSE:",mse)

    print("\n RMSE:",numpy.sqrt(mse))

    return

for i in list(lol_test.columns):
    forecast_errors(forecast_df[i+'_pred'],lol_test[i])
```

```
For "revenue_all"
-----
Bias: 231208.71274043826
MAE: 1352654.7052081372
MSE: 19077478779470.477
RMSE: 4367777.327139111

For "revenue_ss40"
-----
Bias: 328129.9827057099
```

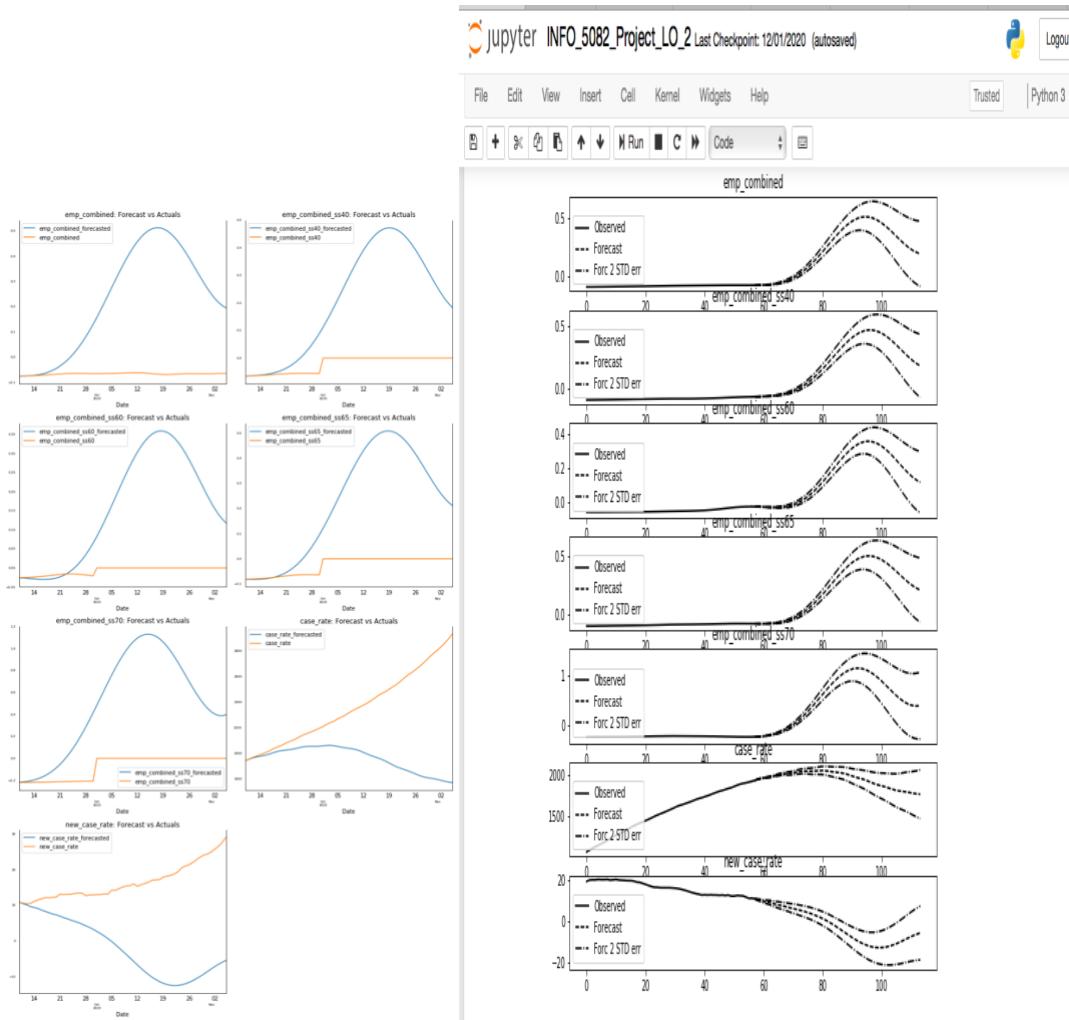
Considering the same 'revenue_all' variable, RMSE = 4367777.33 and bias = 231208.71

says the VECM model is relative better for analyzing the overall small business revenue.

Data visualizations for second study objective – estimation of employment rates

❖ VECM model visualizations and evaluation:

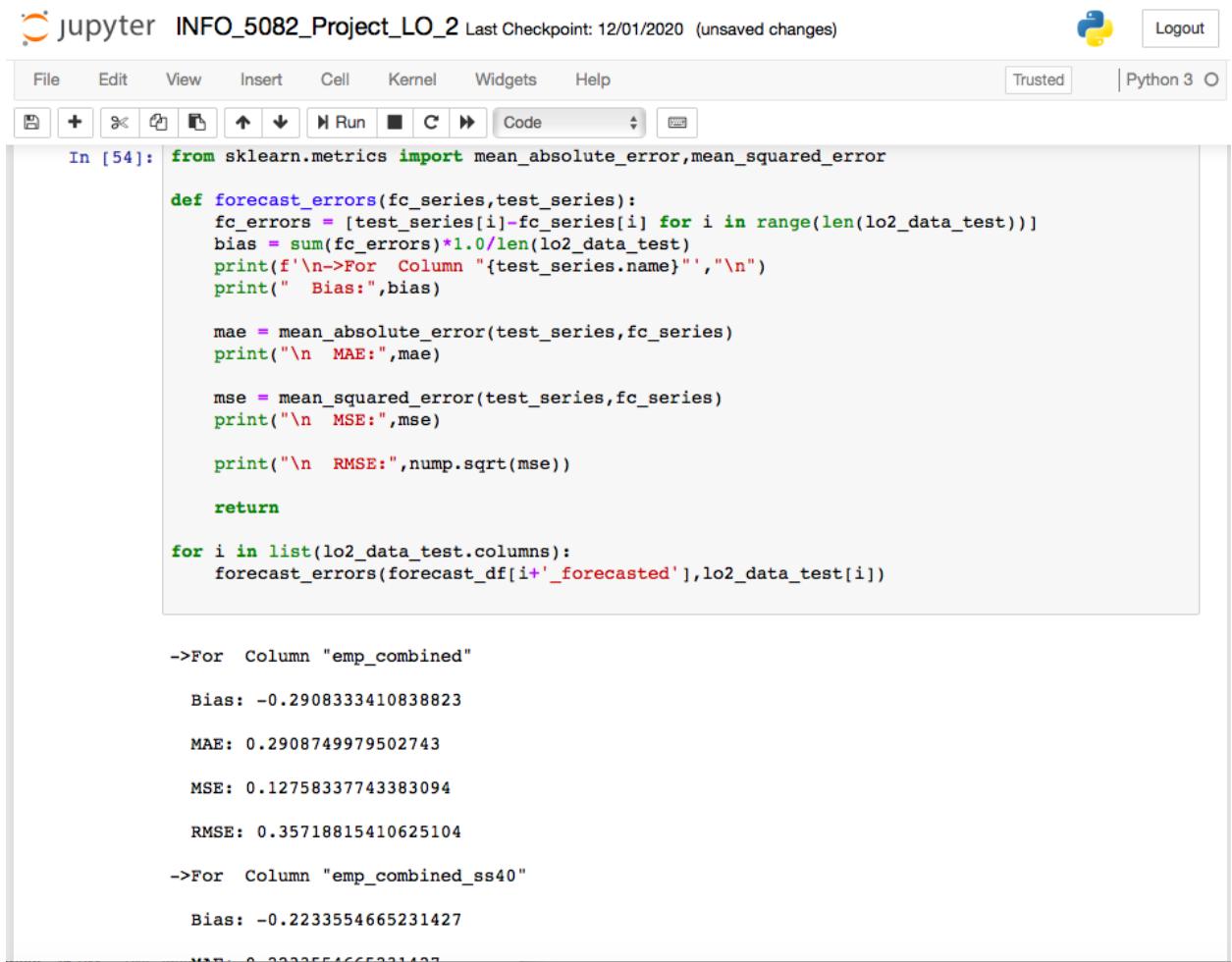
- Forecasted and actual values are plotted and also the upper and lower values of forecasted values are shown as below,



From the above graphs we can see that output error rate is much higher. And upper and lower forecasted values within standard deviations is predicted as the significance give is 0.05.

- Evaluating the forecast:

Performing model evaluation using MAE, MSE , RMSE and bias as below.



The screenshot shows a Jupyter Notebook interface with the title "jupyter INFO_5082_Project_LO_2 Last Checkpoint: 12/01/2020 (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code cell (In [54]) contains Python code for calculating forecast errors. The output cell shows the results for two columns: "emp_combined" and "emp_combined_ss40".

```

In [54]: from sklearn.metrics import mean_absolute_error,mean_squared_error

def forecast_errors(fc_series,test_series):
    fc_errors = [test_series[i]-fc_series[i] for i in range(len(lo2_data_test))]
    bias = sum(fc_errors)*1.0/len(lo2_data_test)
    print(f'\n->For Column "{test_series.name}"\n')
    print("  Bias:",bias)

    mae = mean_absolute_error(test_series,fc_series)
    print("\n  MAE:",mae)

    mse = mean_squared_error(test_series,fc_series)
    print("\n  MSE:",mse)

    print("\n  RMSE:",numpy.sqrt(mse))

    return

for i in list(lo2_data_test.columns):
    forecast_errors(forecast_df[i+'_forecasted'],lo2_data_test[i])

```

->For Column "emp_combined"
 Bias: -0.2908333410838823
 MAE: 0.2908749979502743
 MSE: 0.12758337743383094
 RMSE: 0.35718815410625104
 ->For Column "emp_combined_ss40"
 Bias: -0.2233554665231427
 MAE: 0.2233554665231427

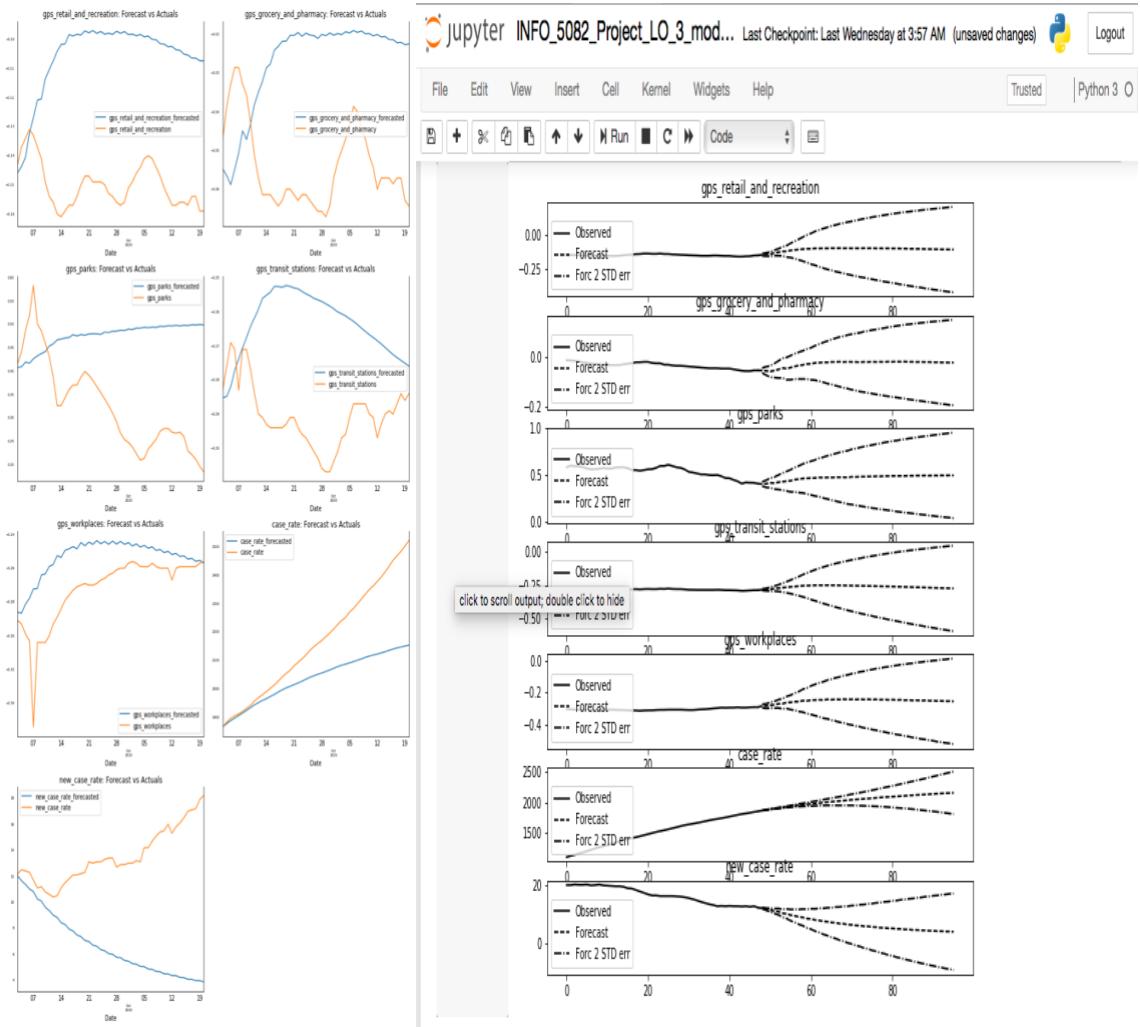
Considering the 'emp_combined' variable, RMSE = 0.3572 and bias = -0.2907 says the VECM model is has 35% accuracy and it is slightly over biased for analyzing the overall employment rates.

Data visualizations for third study objective – analyzing the people's mobility with covid-19

case rate

- ❖ VECM model visualizations and evaluation:

Visualizations for forecasted and actual values are plotted. Also the upper and lower values of forecasted values are plotted as below,



From the above graphs we can see that this model has better forecast. For the given 0.05 significance the upper and lower values for forecast are predicted.

- Evaluating the forecast:

Performing model evaluation using MAE, MSE , RMSE and bias as below.

```

In [70]: from sklearn.metrics import mean_absolute_error,mean_squared_error
def forecast_errors(fc_series,test_series):
    fc_errors = [test_series[i]-fc_series[i] for i in range(len(test_series))]
    bias = sum(fc_errors)*1.0/len(test_series)
    print(f"\n-> For Column '{test_series.name}':\n")
    print(f" Bias:{bias}\n")
    mae = mean_absolute_error(test_series,fc_series)
    print(f"\n MAE:{mae}\n")
    mse = mean_squared_error(test_series,fc_series)
    print(f"\n MSE:{mse}\n")
    rmse = np.sqrt(mse)
    print(f"\n RMSE:{rmse}\n")
    return
for i in list(lol_n2_data_test.columns):
    forecast_errors(forecast_df[i+"_forecasted"],lol_n2_data_test[i])

```

MAE: 0.0009921975257603
MSE: 0.03149951987849978
-> For Column "gpe_workplaces":
Bias: -0.0190231573640805
MAE: 0.019061215638379453
MSE: 0.000529375201096284
RMSE: 0.023726304453488504
-> For Column "case_rate":
Bias: 129.64682395324644
MAE: 129.64682395324644
MSE: 29071.641288987652
RMSE: 170.50407997754087
-> For Column "new_case_rate":
Bias: 6.809362156586476
MAE: 6.809362156586476
MSE: 63.79257916112217
RMSE: 7.987025676753654
MAE: 0.01187046184187318

Considering the ‘case_rate’ variable, RMSE = 170.50 and bias = 129.65 says that model is has a relatively better accuracy and it is under biased for analyzing the overall case_rates.

And considering the ‘new_case_rate’ variable, RMSE = 7.98 and bias = 6.802907 says that model is has a relatively better accuracy and it is under biased for analyzing and predicting new_case_rates in a day.

Conclusion

From the analysis of small business revenue data

Overall revenue in all small business is slowly increasing after sudden impact of covid-19 from March to April 2020.

By observing the granger's causality matrix, we can say the overall revenue generated in small businesses is very much impacted and caused by customers spending in almost every services like general merchandize, arts, etc., and also with opening businesses has shown impact with more revenue. It has been forecasted to maintain the same pattern till November 2020.

Revenue generated in transportation (SS 40) has shown similar impact and causality by customer spending and business opening except with professional and business services. It also ha been forecasted to maintain the same pattern till November 2020.

Revenue generated in Professional and business service (SS 60) has shown similar impact and causality by customer spending and business opening except with spending retails with no groceries and leisure and hospitality services. It also has been forecasted to maintain the same pattern till November 2020

Revenue generated in Education and health service (SS 65) has shown similar impact and causality by customer spending and business opening except with spending in arts and entertainment and leisure and hospitality services. It also has been forecasted to maintain the same pattern till November 2020.

Revenue generated in Leisure and hospitality service (SS 70) has shown similar impact and causality by customer spending and business opening except by transportation services. It also has been forecasted to maintain the same pattern till November 2020.

Both models VAR and VECM predicted with not so good accuracy, so this analysis need much more data for better understanding and predictions in small business revenues.

From the estimation of employment rates

Overall employment rate across the nation has also seen a sudden impact of covid-19 from in April 2020.

By observing the granger's causality matrix, we can say the overall employment rates has a very direct impact and caused by covid case rates along with daily new case rates. It is also impacted by employment rate in leisure and hospitality services and the model accuracy for overall employment is good but can improve its performance with more data.

Employment rate in transportation (SS 40), employment rate in Professional and business service (SS 60), employment rate in Education and health service (SS 65) and employment rate in Leisure and hospitality service (SS 70)has shown impact and causality by covid-19 cases and employment rate in all other services.

The VECM model predicted with a better accuracy rate, but can be improved by giving more data for better understanding and predictions for employment rate.

From the analysis of covid-19 rates with people's mobility

Overall covid-19 infections rate across the nation has been a increasing from the March 2020.

From model 1 where it has shown that people's mobility out of residential locations or inside residence has no impact on covid-19 which is strange.

From model 2, observing the granger's causality matrix, we can say the overall case_rates has a impact and caused by time spent in retail and recreational areas and in pharmacy.

The VECM model predicted with relatively better accuracy rate, but can be improved by giving more input data or including more variables of time spent in other locations like schools for better understanding and predictions for employment rate.

Bibliography

- Chetty, R., Friedman, J. N., Hendren, N., Stephene, M. (2020). *The Economic Impacts of COVID-19: Evidence from a New Public Database Built Using Private Sector Data.* Retrieved from: https://opportunityinsights.org/wp-content/uploads/2020/05/tracker_paper.pdf
- Sara. H. (2020). "Public Sector Impacts of the Great Recession and COVID-19". Retrieved from <https://laborcenter.berkeley.edu/public-sector-impacts-great-recession-and-covid-19/>
- Sorensen, S. B. (September 2020). ECONOMICS. *Cointegration*. Retrieved from <https://uh.edu/~bsorense/coint.pdf>
- Hjalmarsson, E., Osterholm, P. (June 2007). Rennhack., R. K. *Testing for Cointegration Using the Johansen Methodology when Variables are Near-Integrated*. Retrieved from <https://www.imf.org/external/pubs/ft/wp/2007/wp07141.pdf> WP/07/141
- Singh, A. (2018). Analytics Vidya. *A Gentle Introduction to Handling a Non-Stationary Time Series in Python*. Retrieved from <https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/>