Project Name – ETL (Extract, Transform and Load) technical report

Submitted by- Amber Sidhu, Raj Sisodia

Project Proposal – Currently, as world is facing a pandemic called Covid-19, the economies across the globe are fluctuating. There is a variation in the market which is leading to uncertainty across different sectors. These changes are leading to unemployment, lack of demand of the nations good's and services and decrease in productivity. In contrast to that our company figured a bunch of data which includes USA covid-19 and NYC stock exchange files and our team is tasked to migrate it to production data base.

Sources of Data – a) data. World b) Kaggle c) Yahoo finance website

Data Links are provided below:

https://www.kaggle.com/sudalairajkumar/covid19-in-usa#us_states_covid19_daily.csv

https://www.kaggle.com/dgawlik/nyse#prices.csv

https://ca.finance.yahoo.com/quote/SPGI/history?p=SPGI

Main Procedure-

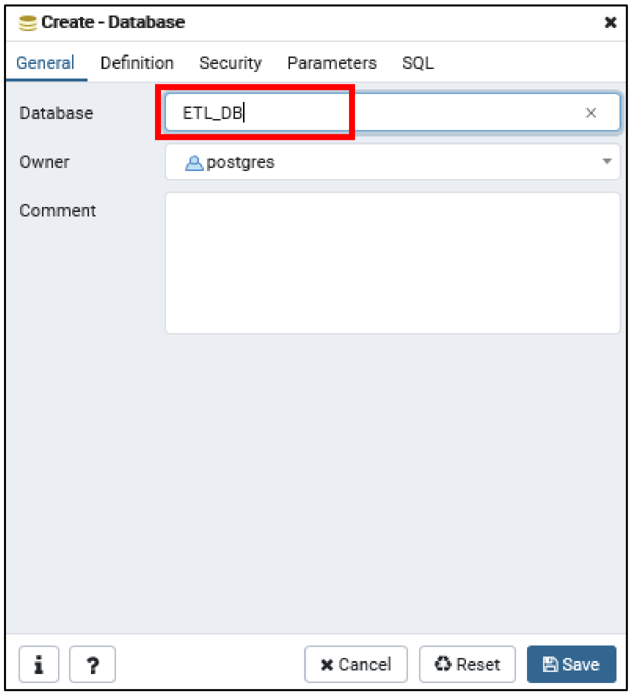
ETL - ETL stands for Extract, Transform and Load, which is a process used to collect data from various sources, transform the data depending on business rules/needs and load the data into a destination database.

The project started with lots of brainstorming within the team where we discussed different scenarios and ways to receive the data. We went through various websites like data.world, Kaggle and yahoo finance to finalise the data which looked productive and appealing to our eyes. The **USA Covid-19 csv** and **New-York stock exchange csv** files used for this project are taken from the above-mentioned sources of data. Even to make this project more challenging we scraped a stock table from yahoo finance website.

Extraction Process -

The individual csv files were visualised by team members to discover if the columns and rows are well structured with proper values. The team after mutual discussion finalised the columns in both the data sets and started working ahead.

The process started with creating a **database in pgAdmin named ETL_db**

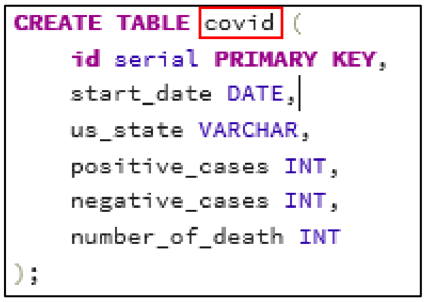

Create - Database

Database: ETL_DB

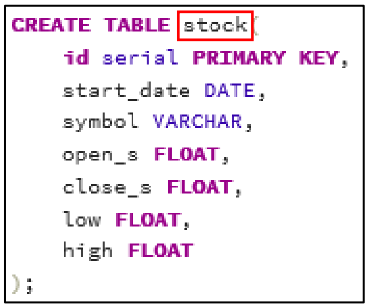
Owner: postgres

Comment:

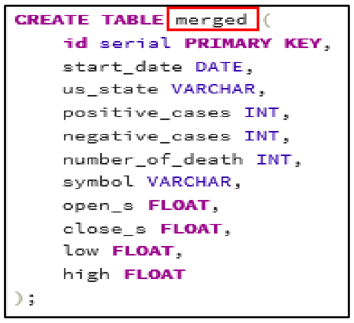
After which, tables are created to which the data is loaded after the execution and transformation process. In the below pictorial three tables are created in pgAdmin i.e. covid, stock and merged table.



```
CREATE TABLE covid (
    id serial PRIMARY KEY,
    start_date DATE,
    us_state VARCHAR,
    positive_cases INT,
    negative_cases INT,
    number_of_death INT
);
```



```
CREATE TABLE stock(
    id serial PRIMARY KEY,
    start_date DATE,
    symbol VARCHAR,
    open_s FLOAT,
    close_s FLOAT,
    low FLOAT,
    high FLOAT
);
```



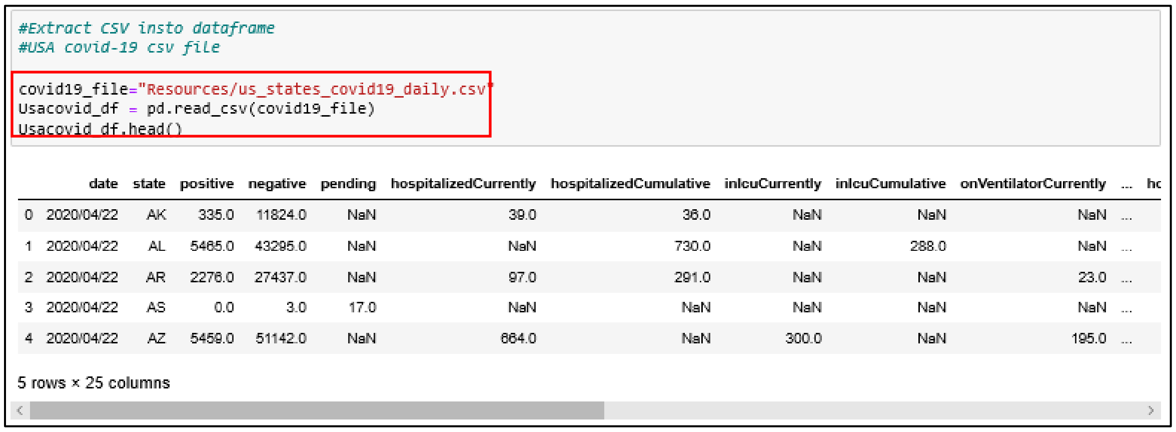
```
CREATE TABLE merged (
    id serial PRIMARY KEY,
    start_date DATE,
    us_state VARCHAR,
    positive_cases INT,
    negative_cases INT,
    number_of_death INT,
    symbol VARCHAR,
    open_s FLOAT,
    close_s FLOAT,
    low FLOAT,
    high FLOAT
);
```

Select* from covid, select* from stock and select* from merged commands are used to run and create tables in the database.

Jupyter Notebook

The next extraction process is to extract the excel csv files (**USA Covid-19 csv** and **New-York stock exchange csv**) in Jupyter notebook. It begins by importing the correct dependencies. For this project, the following dependencies were imported, import pandas as pd, from sqlalchemy import create_engine and import datetime as dt.

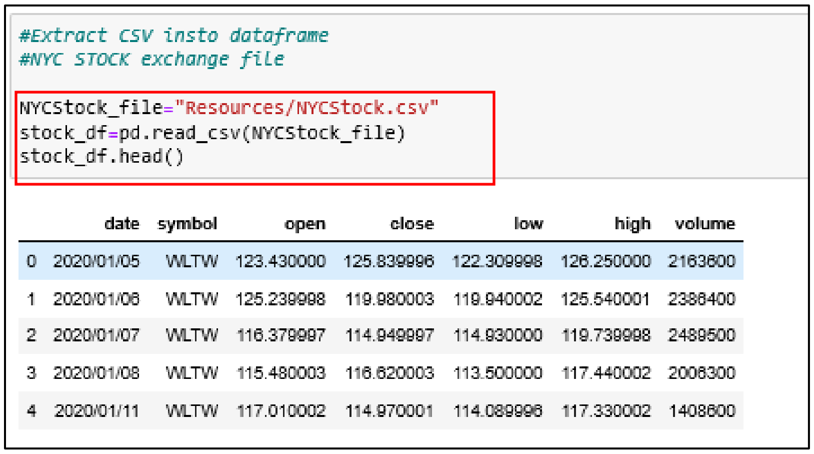
Extract the **USA covid-19 csv file** by giving the path where the file is situated and using **pd. read_csv file** command. Use**. head( )** command to visualize the data frame as shown in the picture,



```
#Extract CSV insto dataframe
#USA covid-19 csv file

covid19_file="Resources/us_states_covid19_daily.csv"
Usacovid_df = pd.read_csv(covid19_file)
Usacovid_df.head()
```

```
         date state positive negative pending hospitalizedCurrently hospitalizedCumulative inIcuCurrently inIcuCumulative onVentilatorCurrently ... hc
0  2020/04/22    AK    335.0  11824.0     NaN                  39.0                   36.0            NaN             NaN                   NaN ...
1  2020/04/22    AL   5465.0  43295.0     NaN                   NaN                  730.0            NaN           288.0                   NaN ...
2  2020/04/22    AR   2276.0  27437.0     NaN                  97.0                  291.0            NaN             NaN                  23.0 ...
3  2020/04/22    AS      0.0      3.0    17.0                   NaN                    NaN            NaN             NaN                   NaN ...
4  2020/04/22    AZ   5459.0  51142.0     NaN                 664.0                    NaN          300.0             NaN                 195.0 ...

5 rows × 25 columns
```

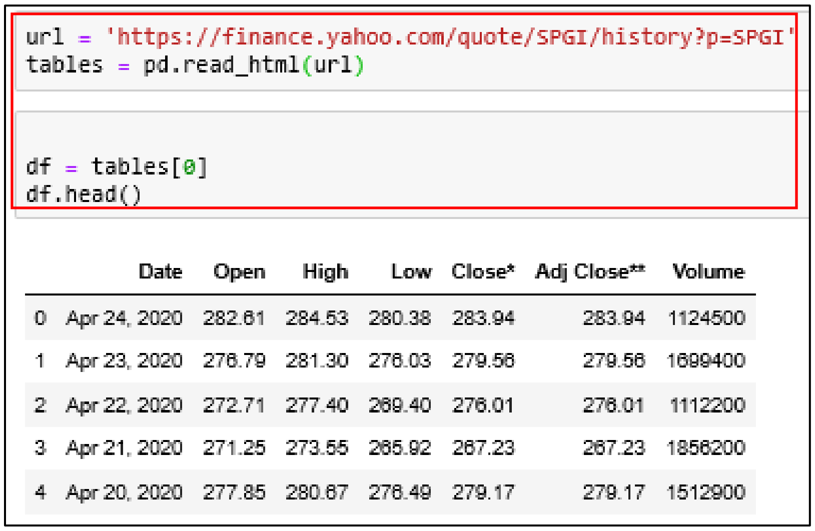
Now, extract the **New-York stock exchange csv** file by giving the path where the file is situated and using **pd. read_csv file command**. **Use. head( )** command to visualize the data frame as shown in the picture,



```
#Extract CSV insto dataframe
#NYC STOCK exchange file

NYCStock_file="Resources/NYCStock.csv"
stock_df=pd.read_csv(NYCStock_file)
stock_df.head()
```

	date	symbol	open	close	low	high	volume
0	2020/01/05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600
1	2020/01/06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400
2	2020/01/07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500
3	2020/01/08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300
4	2020/01/11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600

The second different approach is directly scraping the table from yahoo finance website. The website URL along with pd.read_html(URL) command is used to perform the task.



```
url = 'https://finance.yahoo.com/quote/SPGI/history?p=SPGI'
tables = pd.read_html(url)


df = tables[0]
df.head()
```

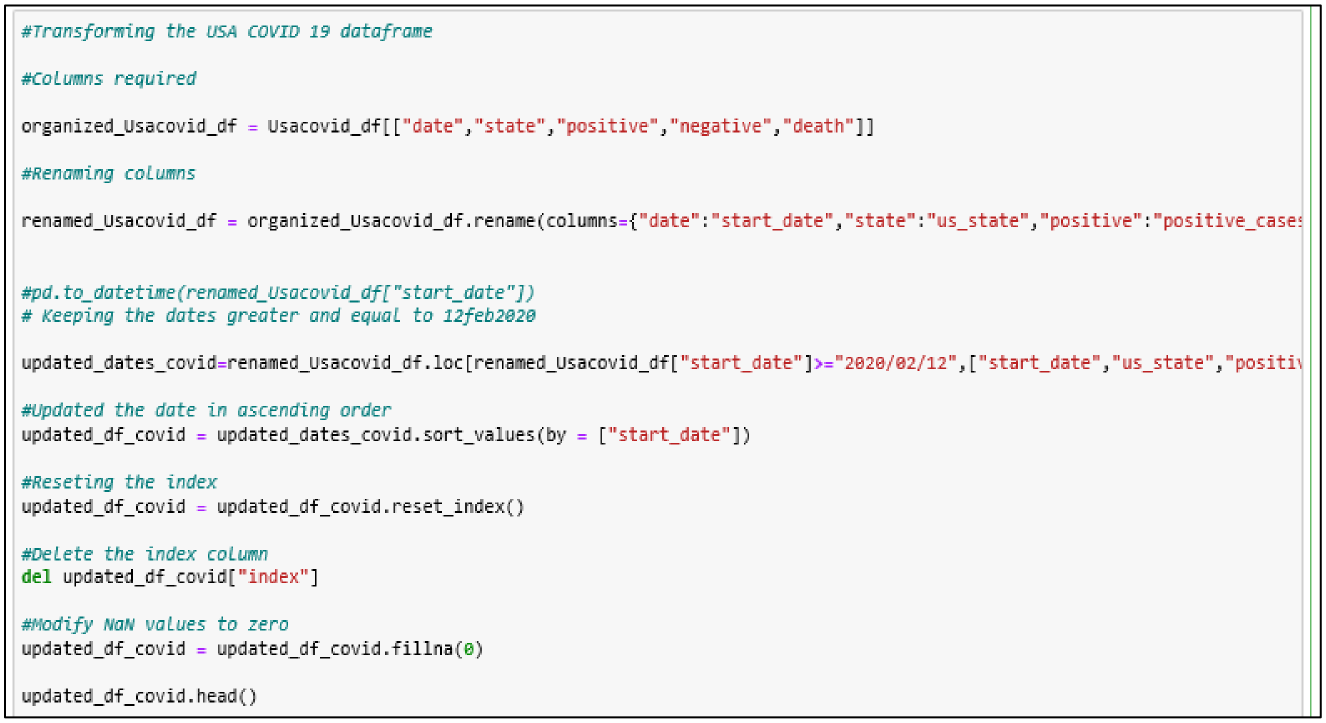
	Date	Open	High	Low	Close*	Adj Close**	Volume
0	Apr 24, 2020	282.61	284.53	280.38	283.94	283.94	1124500
1	Apr 23, 2020	276.79	281.30	276.03	279.56	279.56	1699400
2	Apr 22, 2020	272.71	277.40	269.40	276.01	276.01	1112200
3	Apr 21, 2020	271.25	273.55	265.92	267.23	267.23	1856200
4	Apr 20, 2020	277.85	280.67	276.49	279.17	279.17	1512900

Transformation Process – The main objective of the transformation process is to mould the data according to the specific needs. The data executed can provide unnecessary information which is not required. Transformation process in Jupyter notebook, can help to modify the data and clean according to the business requirement.

For the following two files imported, the transformation will be done separately.

In the **USA covid-19 data frame,** there are certain columns which are not required. Use **df [[]]** command to eliminate unnecessary columns and columns are renamed using **df. rename(columns={})** command.

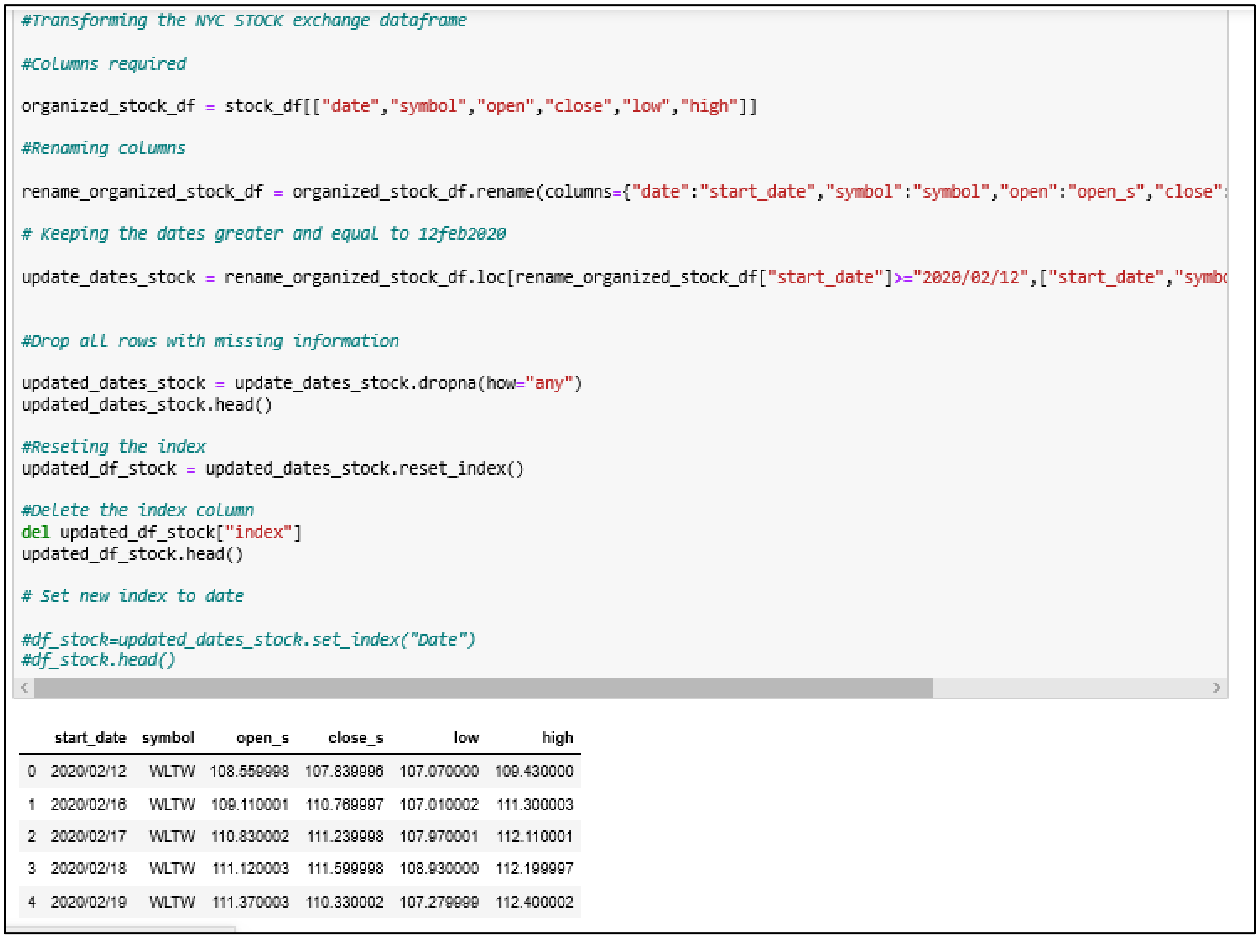
In the following data frame **start_date, us_state, positive_cases, negative_cases, number_of_death** columns are used. According to the project requirements the data is sorted on **dates** as the number of cases spiked in USA after February 28,2020. The dates are then arranged in ascending order to keep the dates in line and index is reset to have another column with serial numbers. In the data there were many rows with NaN values, all those values were converted to zero.



```
#Transforming the USA COVID 19 dataframe

#Columns required

organized_Usacovid_df = Usacovid_df[["date","state","positive","negative","death"]]

#Renaming columns

renamed_Usacovid_df = organized_Usacovid_df.rename(columns={"date":"start_date","state":"us_state","positive":"positive_cases


#pd.to_datetime(renamed_Usacovid_df["start_date"])
# Keeping the dates greater and equal to 12feb2020

updated_dates_covid=renamed_Usacovid_df.loc[renamed_Usacovid_df["start_date"]>="2020/02/12",["start_date","us_state","positiv

#Updated the date in ascending order
updated_df_covid = updated_dates_covid.sort_values(by = ["start_date"])

#Reseting the index
updated_df_covid = updated_df_covid.reset_index()

#Delete the index column
del updated_df_covid["index"]

#Modify NaN values to zero
updated_df_covid = updated_df_covid.fillna(0)

updated_df_covid.head()
```

In the **New-York stock exchange data frame CSV file,** there are certain columns which are not required. Use **df [[]]** command to eliminate unnecessary columns and columns are renamed using **df. rename(columns={})** command.

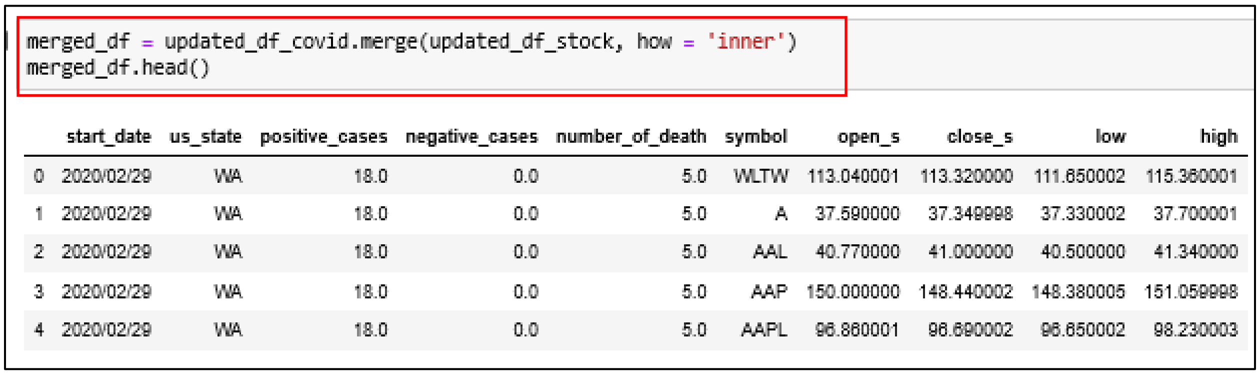
In the following data frame **start_date, Symbol, open_s, close_s, low, high** columns are used**.** The main objective is also to determine how the stock market fluctuated as the number of cases spiked in USA. So, data is sorted on **dates** to identify the stocks in USA after February 28,2020. The dates are then arranged in ascending order to keep the dates in line and index is reset to have another column with serial numbers. The stock csv is a big file with many rows with missing information, so all the rows with missing information is dropped from the dataset.



```
#Transforming the NYC STOCK exchange dataframe

#Columns required

organized_stock_df = stock_df[["date","symbol","open","close","low","high"]]

#Renaming columns

rename_organized_stock_df = organized_stock_df.rename(columns={"date":"start_date","symbol":"symbol","open":"open_s","close":

# Keeping the dates greater and equal to 12feb2020

update_dates_stock = rename_organized_stock_df.loc[rename_organized_stock_df["start_date"]>="2020/02/12",["start_date","symbo


#Drop all rows with missing information

updated_dates_stock = update_dates_stock.dropna(how="any")
updated_dates_stock.head()

#Reseting the index
updated_df_stock = updated_dates_stock.reset_index()

#Delete the index column
del updated_df_stock["index"]
updated_df_stock.head()

# Set new index to date

#df_stock=updated_dates_stock.set_index("Date")
#df_stock.head()
```

	start_date	symbol	open_s	close_s	low	high
0	2020/02/12	WLTW	108.559998	107.839996	107.070000	109.430000
1	2020/02/16	WLTW	109.110001	110.769997	107.010002	111.300003
2	2020/02/17	WLTW	110.830002	111.239998	107.970001	112.110001
3	2020/02/18	WLTW	111.120003	111.599998	108.930000	112.199997
4	2020/02/19	WLTW	111.370003	110.330002	107.279999	112.400002

Both the data(**USA Covid-19** and **New-York stock exchange**) are merged using inner joint and on date.

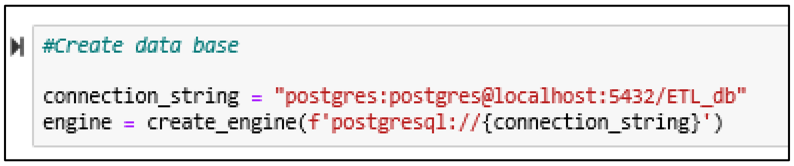


```
merged_df = updated_df_covid.merge(updated_df_stock, how = 'inner')
merged_df.head()
```

	start_date	us_state	positive_cases	negative_cases	number_of_death	symbol	open_s	close_s	low	high
0	2020/02/29	WA	18.0	0.0	5.0	WLTW	113.040001	113.320000	111.650002	115.360001
1	2020/02/29	WA	18.0	0.0	5.0	A	37.590000	37.349998	37.330002	37.700001
2	2020/02/29	WA	18.0	0.0	5.0	AAL	40.770000	41.000000	40.500000	41.340000
3	2020/02/29	WA	18.0	0.0	5.0	AAP	150.000000	148.440002	148.380005	151.059998
4	2020/02/29	WA	18.0	0.0	5.0	AAPL	96.860001	96.690002	96.650002	98.230003

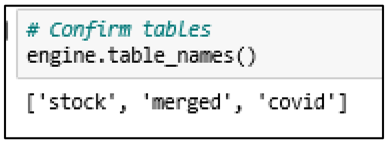
Loading Process – The process is to load the data after the transformation/modification process into the tables created in pgAdmin initially.

The process starts by creating connection string which consists of username and password along with local host details of the database followed by creating engine and giving the connection string to PostgreSQL.



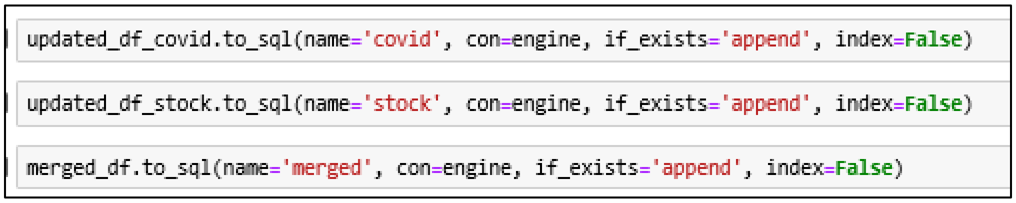
```
#Create data base

connection_string = "postgres:postgres@localhost:5432/ETL_db"
engine = create_engine(f'postgresql://{connection_string}')
```

Confirm if the connection is made by checking the tables using **engine.table_names()** command.



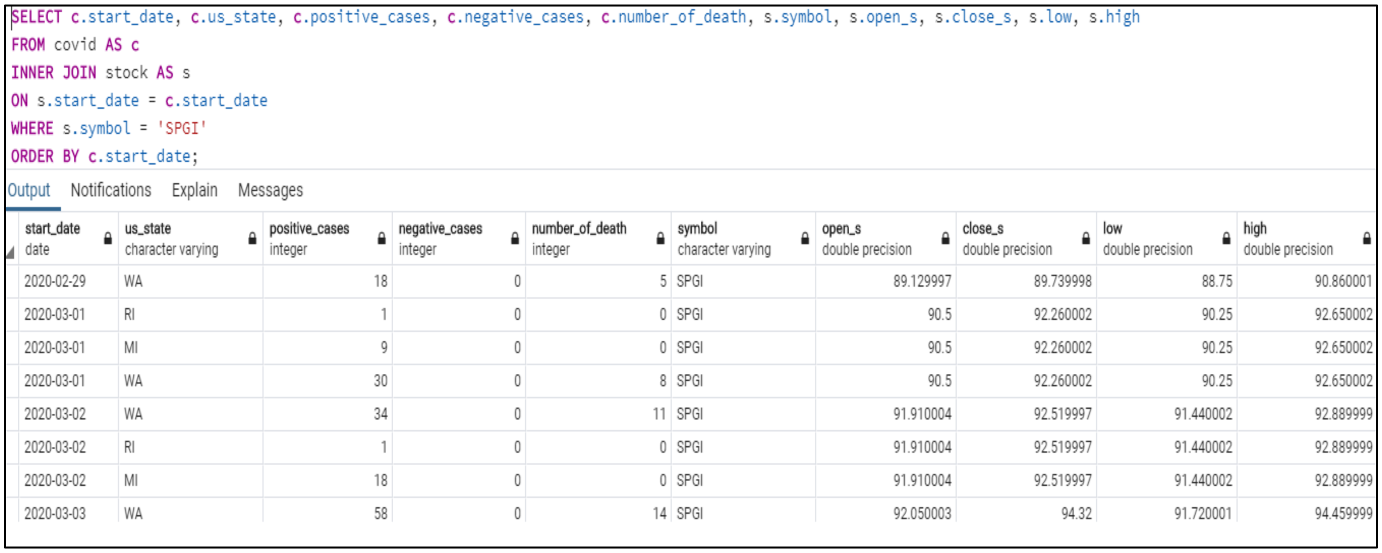
```
# Confirm tables
engine.table_names()

['stock', 'merged', 'covid']
```

Convert the individual tables formed in the jupyter notebook using pandas to sql using **.to_sql** command and append the data into sql tables.



```
updated_df_covid.to_sql(name='covid', con=engine, if_exists='append', index=False)

updated_df_stock.to_sql(name='stock', con=engine, if_exists='append', index=False)

merged_df.to_sql(name='merged', con=engine, if_exists='append', index=False)
```

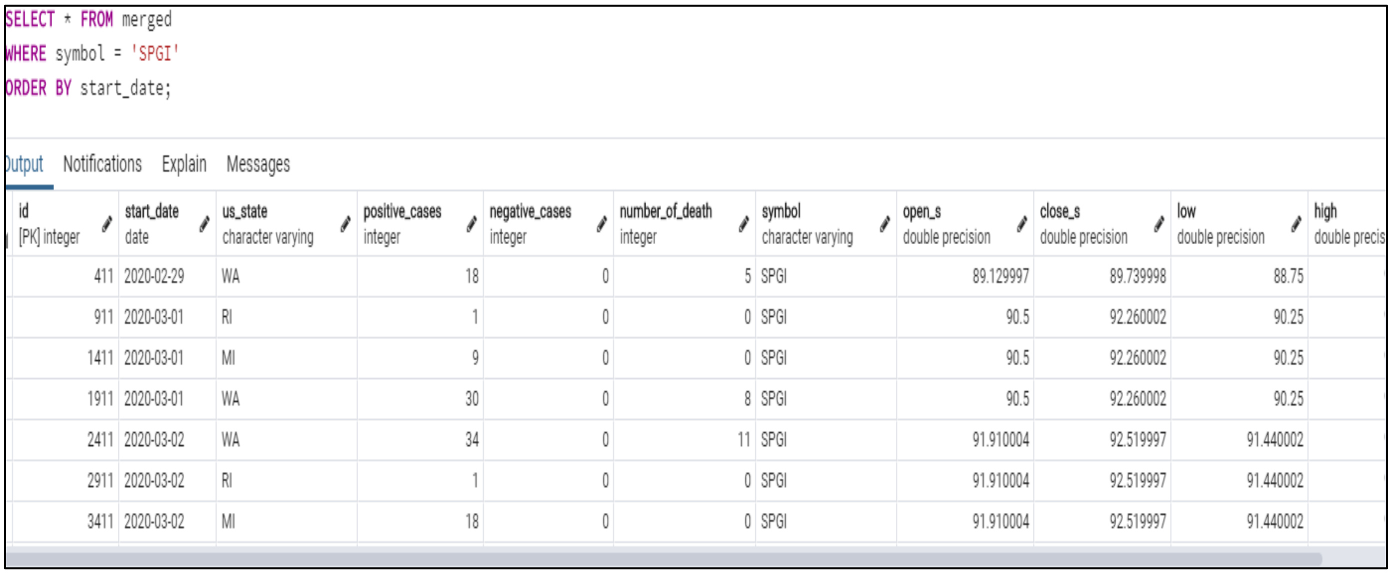
QC & Data Validation – Check the SQL database, if tables persists and data holds it value from the loading process by running the following query in Query Tool. Here we are checking data for S&P Global Inc. using its symbol "**SPGI**" to identify its stock price variation from dates starting from 28th Feb, the oldest date our combined data holds for covid-19 and stock tables.



```
SELECT c.start_date, c.us_state, c.positive_cases, c.negative_cases, c.number_of_death, s.symbol, s.open_s, s.close_s, s.low, s.high
FROM covid AS c
INNER JOIN stock AS s
ON s.start_date = c.start_date
WHERE s.symbol = 'SPGI'
ORDER BY c.start_date;
```

start_date date	us_state character varying	positive_cases integer	negative_cases integer	number_of_death integer	symbol character varying	open_s double precision	close_s double precision	low double precision	high double precision
2020-02-29	WA	18	0	5	SPGI	89.129997	89.739998	88.75	90.860001
2020-03-01	RI	1	0	0	SPGI	90.5	92.260002	90.25	92.650002
2020-03-01	MI	9	0	0	SPGI	90.5	92.260002	90.25	92.650002
2020-03-01	WA	30	0	8	SPGI	90.5	92.260002	90.25	92.650002
2020-03-02	WA	34	0	11	SPGI	91.910004	92.519997	91.440002	92.889999
2020-03-02	RI	1	0	0	SPGI	91.910004	92.519997	91.440002	92.889999
2020-03-02	MI	18	0	0	SPGI	91.910004	92.519997	91.440002	92.889999
2020-03-03	WA	58	0	14	SPGI	92.050003	94.32	91.720001	94.459999

We can validate the results with merged data table, by running following query. Both queries returned 1895 rows. Interesting to note that, above query returned result in 337msec whereas this one took 417msec.



```
SELECT * FROM merged
WHERE symbol = 'SPGI'
ORDER BY start_date;
```

id [PK] integer	start_date date	us_state character varying	positive_cases integer	negative_cases integer	number_of_death integer	symbol character varying	open_s double precision	close_s double precision	low double precision	high double precis
411	2020-02-29	WA	18	0	5	SPGI	89.129997	89.739998	88.75	
911	2020-03-01	RI	1	0	0	SPGI	90.5	92.260002	90.25	
1411	2020-03-01	MI	9	0	0	SPGI	90.5	92.260002	90.25	
1911	2020-03-01	WA	30	0	8	SPGI	90.5	92.260002	90.25	
2411	2020-03-02	WA	34	0	11	SPGI	91.910004	92.519997	91.440002	
2911	2020-03-02	RI	1	0	0	SPGI	91.910004	92.519997	91.440002	
3411	2020-03-02	MI	18	0	0	SPGI	91.910004	92.519997	91.440002	

This data is selected to determine the fluctuation in the stock market due to covid-19 pandemic. It is wisely said that any form of major crisis can result in fluctuation. The data can help investors to follow the stocks closely during the pandemic and can help the major industries to compare their results with their competitors.