Pandas Handbook

“ Reading a CSV file into a Pandas data frame, Retrieving data from Pandas data frames, Querying, sorting, and analyzing data, Merging, grouping, and aggregation of data, Extracting useful information from dates, Basic plotting using line and bar charts, Writing data frames to CSV files“

By
Ahmad Jawabreh

Dec, 2022

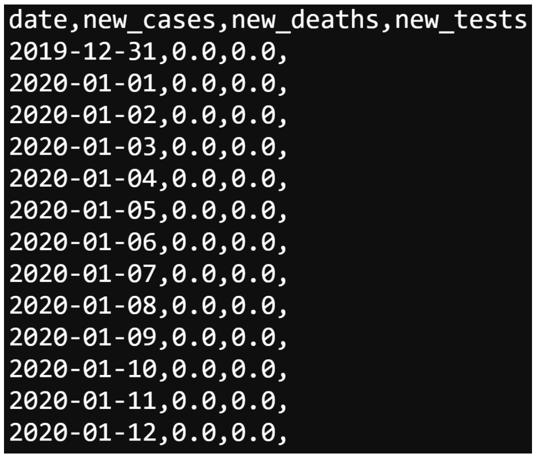
This handbook covers the following topics:

- Reading a CSV file into a Pandas data frame.
- Retrieving data from Pandas data frames.
- Querying, sorting, and analyzing data.
- Merging, grouping, and aggregation of data.
- Extracting useful information from dates.
- Basic plotting using line and bar charts.
- Writing data frames to CSV files.

1.0 Introduction

Pandas is a popular Python library used for working in tabular data (similar to the data stored in a spreadsheet). Pandas provides helper functions to read data from various file formats like CSV, Excel spreadsheets, HTML tables, JSON, SQL, and more.

et's download a file italy-covid-daywise.txt which contains day-wise Covid-19 data for Italy in the following format:



```
date,new_cases,new_deaths,new_tests
2019-12-31,0.0,0.0,
2020-01-01,0.0,0.0,
2020-01-02,0.0,0.0,
2020-01-03,0.0,0.0,
2020-01-04,0.0,0.0,
2020-01-05,0.0,0.0,
2020-01-06,0.0,0.0,
2020-01-07,0.0,0.0,
2020-01-08,0.0,0.0,
2020-01-09,0.0,0.0,
2020-01-10,0.0,0.0,
2020-01-11,0.0,0.0,
2020-01-12,0.0,0.0,
```

figure(1): italy-covid-daywis data sample

Data link:
https://gist.githubusercontent.com/aakashns/f6a004fa20c84fec53262f9a8bfee775/raw/f309558b1cf5103424cef58e2ecb8704dcd4d74c/italy-covid-daywise.csv

The data can be installed manually or using urllib library as following:

```
italy_covid_url =
'https://gist.githubusercontent.com/aakashns/f6a004fa20c84fec53262f9a8bfee775/raw/f309558b1cf5103424cef58e2ecb8704dcd4d74c/italy-covid-daywise.csv'
urlretrieve(italy_covid_url, 'italy-covid-daywise.csv')
```

2.0 Reading a CSV file using Pandas

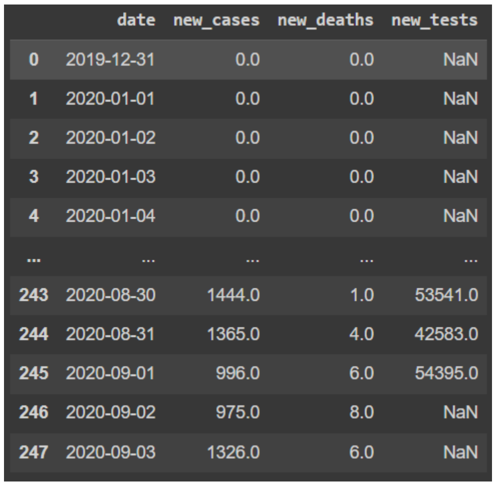
To read the file, we can use the read_csv method from Pandas. First, let's install the Pandas library.

import pandas as pd
covid_df = pd.read_csv('italy-covid-daywise.csv')

The data type of the covid_data_frame variable is pandas.core.frame.DataFrame which is a two-dimensional data structure composed of rows and columns.

Data from the file is read and stored in a DataFrame object - one of the core data structures in Pandas for storing and working with tabular data. We typically use the _df suffix in the variable names for dataframes.

Here is how the data stored in covid_df:



	date	new_cases	new_deaths	new_tests
0	2019-12-31	0.0	0.0	NaN
1	2020-01-01	0.0	0.0	NaN
2	2020-01-02	0.0	0.0	NaN
3	2020-01-03	0.0	0.0	NaN
4	2020-01-04	0.0	0.0	NaN
...	...	...	...	...
243	2020-08-30	1444.0	1.0	53541.0
244	2020-08-31	1365.0	4.0	42583.0
245	2020-09-01	996.0	6.0	54395.0
246	2020-09-02	975.0	8.0	NaN
247	2020-09-03	1326.0	6.0	NaN

figure(2): covid_df sneak peek

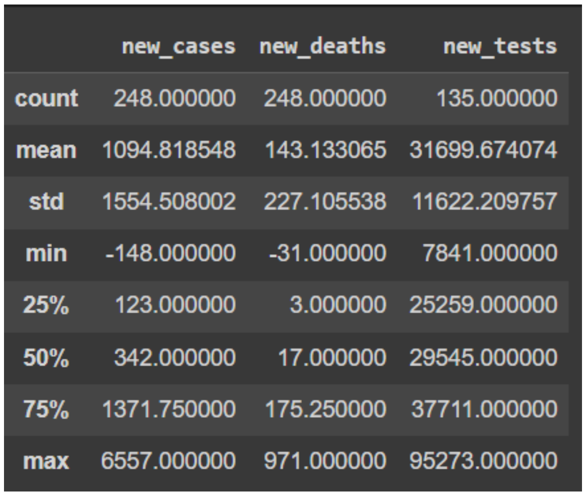
Here's what we can tell by looking at the dataframe:

The file provides four day-wise counts for COVID-19 in Italy
The metrics reported are new cases, deaths, and tests
Data is provided for 248 days: from Dec 12, 2019, to Sep 3, 2020

To get some information about the data that we have read, we can use the .info function which will give us some information about the data. covid_df.info()

It appears that each column contains values of a specific data type. You can view statistical information for numerical columns (mean, standard deviation, minimum/maximum values, and the number of non-empty values) using the .describe method.

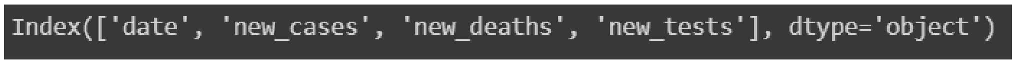
So if we run the commandcovid_df.describe(), we will see the below



	new_cases	new_deaths	new_tests
count	248.000000	248.000000	135.000000
mean	1094.818548	143.133065	31699.674074
std	1554.508002	227.105538	11622.209757
min	-148.000000	-31.000000	7841.000000
25%	123.000000	3.000000	25259.000000
50%	342.000000	17.000000	29545.000000
75%	1371.750000	175.250000	37711.000000
max	6557.000000	971.000000	95273.000000

figure(3): covid_df.describe() output

On the other hand, the columns property contains the list of columns within the data frame. covid_df.columns



```
Index(['date', 'new_cases', 'new_deaths', 'new_tests'], dtype='object')
```

figure(4): covid_df.columns output

You can also retrieve the number of rows and columns in the data frame using the .shape property. covid_df.shape

<u>Here's a summary of the functions & methods we've looked at so far:</u>

- pd.read_csv - Read data from a CSV file into a Pandas DataFrame object
- .info() - View basic infomation about rows, columns & data types
- .describe() - View statistical information about numeric columns
- .columns - Get the list of column names
- .shape - Get the number of rows & columns as a tuple

3.0 Retrieving data from a data frame

The first thing you might want to do is retrieve data from this data frame, e.g., the counts of a specific day or the list of values in a particular column. To do this, it might help to understand the internal representation of data in a data frame. Conceptually, you can think of a dataframe as a dictionary of lists: keys are column names, and values are lists/arrays containing data for the respective columns.

```
# Pandas format is similar to this
covid_data_dict = {
   'date':       ['2020-08-30', '2020-08-31', '2020-09-01', '2020-09-02', '2020-09-03'],
   'new_cases':  [1444, 1365, 996, 975, 1326],
   'new_deaths': [1, 4, 6, 8, 6],
   'new_tests': [53541, 42583, 54395, None, None]
}
```

Representing data in the above format has a few benefits:

- All values in a column typically have the same type of value, so it's more efficient to store them in a single array.
- Retrieving the values for a particular row simply requires extracting the elements at a given index from each column array.
- The representation is more compact (column names are recorded only once) compared to other formats that use a dictionary for each row of data (see the example below).

```
# Pandas format is not similar to this
covid_data_list = [
   {'date': '2020-08-30', 'new_cases': 1444, 'new_deaths': 1, 'new_tests': 53541},
   {'date': '2020-08-31', 'new_cases': 1365, 'new_deaths': 4, 'new_tests': 42583},
   {'date': '2020-09-01', 'new_cases': 996, 'new_deaths': 6, 'new_tests': 54395},
   {'date': '2020-09-02', 'new_cases': 975, 'new_deaths': 8 },
   {'date': '2020-09-03', 'new_cases': 1326, 'new_deaths': 6},
]
```

With the dictionary of lists analogy in mind, you can now guess how to retrieve data from a data frame. For example, we can get a list of values from a specific column using the [] indexing notation.

```
covid_data_dict['new_cases']
```

Each column is represented using a data structure called **Series**, which is essentially a numpy array with some extra methods and properties.

Like arrays, you can retrieve a specific value with a series using the indexing notation [].

- covid_df['new_cases'][246]
- covid_df['new_tests'][240]

Pandas also provides the **.at** method to retrieve the element at a specific row & column directly.

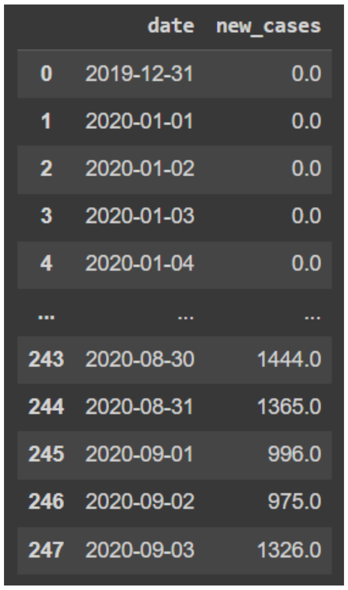
- covid_df.at[246, 'new_cases']
- covid_df.at[240, 'new_tests']

Instead of using the indexing notation [], Pandas also allows accessing columns as properties of the dataframe using the . notation. However, this method only works for columns whose names do not contain spaces or special characters.

- covid_df.new_cases

Further, you can also pass a list of columns within the indexing notation [] to access a subset of the data frame with just the given columns.

```
cases_df = covid_df[['date', 'new_cases']]
print(cases_df)
```



	date	new_cases
0	2019-12-31	0.0
1	2020-01-01	0.0
2	2020-01-02	0.0
3	2020-01-03	0.0
4	2020-01-04	0.0
...	...	...
243	2020-08-30	1444.0
244	2020-08-31	1365.0
245	2020-09-01	996.0
246	2020-09-02	975.0
247	2020-09-03	1326.0

figure(5): list of columns within the indexing notation output

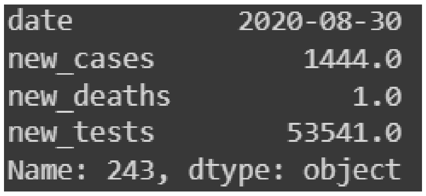
The new data frame cases_df is simply a "view" of the original data frame covid_df. Both point to the same data in the computer's memory. Changing any values inside one of them will also change the respective values in the other. Sharing data between data frames makes data manipulation in Pandas blazing fast. You needn't worry about the overhead of copying thousands or millions of rows every time you want to create a new data frame by operating on an existing one

Sometimes you might need a full copy of the data frame, in which case you can use the **copy** method. covid_df_copy = covid_df.copy()

The data within covid_df_copy is completely separate from covid_df, and changing values inside one of them will not affect the other.

Going back to data retrieval, we said that we can pass a list of columns within the indexing notation to access a subset of the data frame with just the given columns, but what if we want to do that for specific value, for example if we need the date and the new_case at record number 2?

In the case, we can use the **.loc** method, which is helping us to access a specific row of data, covid_df.loc[243], in this case we are retrieving all the information for record number 243 and this is how the result looks like.



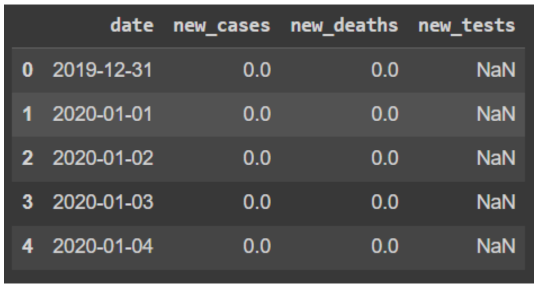
```
date            2020-08-30
new_cases           1444.0
new_deaths             1.0
new_tests          53541.0
Name: 243, dtype: object
```

figure(6): Retrieving Full Data at Specific Row Using .loc

But, we can also specify the columns at specific row, df.loc[2, ['date', 'new_cases']]
print(covid_df.loc[240, ['date', 'new_cases']])

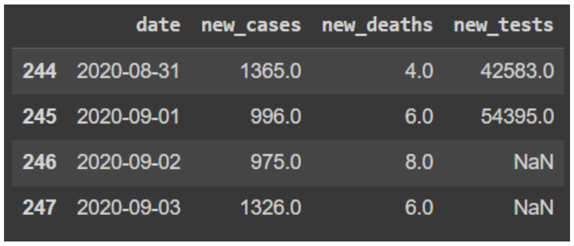
We can use the .head and .tail methods to view the first or last few rows of data.

- covid_df.head(5)
- covid_df.tail(4)



	date	new_cases	new_deaths	new_tests
0	2019-12-31	0.0	0.0	NaN
1	2020-01-01	0.0	0.0	NaN
2	2020-01-02	0.0	0.0	NaN
3	2020-01-03	0.0	0.0	NaN
4	2020-01-04	0.0	0.0	NaN

figure(7): DataFrame Head Sample



	date	new_cases	new_deaths	new_tests
244	2020-08-31	1365.0	4.0	42583.0
245	2020-09-01	996.0	6.0	54395.0
246	2020-09-02	975.0	8.0	NaN
247	2020-09-03	1326.0	6.0	NaN

figure(8): DataFrame Tail Sample

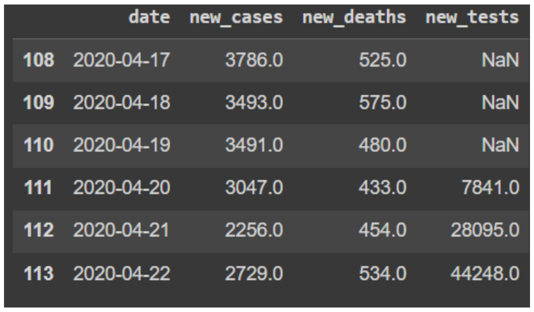
Notice above that while the first few values in the new_cases and new_deaths columns are 0, the corresponding values within the new_tests column are NaN. That is because the CSV file does not contain any data for the new_tests column for specific dates (you can verify this by looking into the file). These values may be missing or unknown.

The distinction between 0 and NaN is subtle but important. In this dataset, it represents that daily test numbers were not reported on specific dates. Italy started reporting daily tests on Apr 19, 2020. 93,5310 tests had already been conducted before Apr 19.

We can find the first index that doesn't contain a NaN value using a column's **first_valid_index** method.

covid_df.new_tests.first_valid_index(), which will be record number 111

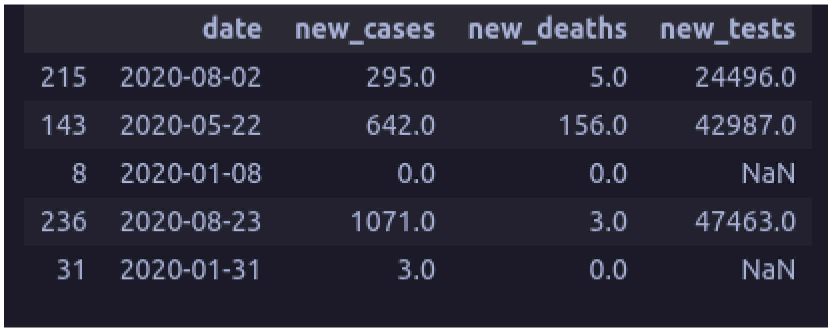
Let's look at a few rows before and after this index to verify that the values change from NaN to actual numbers. We can do this by passing a range to **.loc method**
covid_df.loc[108:113]



	date	new_cases	new_deaths	new_tests
108	2020-04-17	3786.0	525.0	NaN
109	2020-04-18	3493.0	575.0	NaN
110	2020-04-19	3491.0	480.0	NaN
111	2020-04-20	3047.0	433.0	7841.0
112	2020-04-21	2256.0	454.0	28095.0
113	2020-04-22	2729.0	534.0	44248.0

figure(9): Before and After Record NO.111

We can use the **.sample** method to retrieve a random sample of rows from the data frame.
covid_df.sample(5)



	date	new_cases	new_deaths	new_tests
215	2020-08-02	295.0	5.0	24496.0
143	2020-05-22	642.0	156.0	42987.0
8	2020-01-08	0.0	0.0	NaN
236	2020-08-23	1071.0	3.0	47463.0
31	2020-01-31	3.0	0.0	NaN

figure(10): returning set of records using .sample method

4.0 Analyzing data from data frames

Q: What are the total number of reported cases and deaths related to Covid-19 in Italy?

```
total_cases = covid_df.new_cases.sum()
total_deaths = covid_df.new_deaths.sum()

print(f"The number of reported cases is {format(int(total_cases))}, and
the number of reported deaths is {format(int(total_deaths))}")
```

Q: What is the overall death rate (ratio of reported deaths to reported cases)?

```
death_ratio = ((total_deaths/total_cases)*100)
print(f"{death_ratio}%")
```

Q: What is the overall number of tests conducted? A total of 935310 tests were conducted before daily test numbers were reported.

```
OLD_TESTS = 935310
reported_tests = covid_df.new_tests.sum()
total_tests = reported_tests + OLD_TESTS
print(total_tests)
```

5.0 Querying and sorting rows

Let's say we want to see the days where there are more than 1000 new cases. How can we do it ?

```
high_new_cases = covid_df.new_cases > 1000
print(high_new_cases)
```

The output of this code will be a series containing True False values, True indicated that this day contains more than 1000 new cases. But what if we want to see that same old DataFrame which is named in our code as covid_df and show only the days where they reported more than 1000 cases ?

```
high_cases = covid_df[covid_df.new_cases > 1000]
print(high_cases)
```

This above code shows only head and tail, But what if we want to see the full table ? Here we need to use another methodology as in the below code

```
from IPython.display import display
with pd.option_context('display.max_rows', 100):
   display(covid_df[covid_df.new_cases > 1000])
```

Now, let's try to determine the days when the ratio of cases reported to tests conducted is higher than the overall positive_rate if the positive rate is constant 0.05206657403227681.

```
POSITIVE_RATE = 0.05206657403227681
high_ratio_df = covid_df[(covid_df.new_cases / covid_df.new_tests) >
POSITIVE_RATE]
print(high_ratio_df)
```

Ok, now let’s say we want to add new column to the main DataFrame (the covid_df), this new column will be the positive rate which is positive_rate = new_cases / new_tests

```
covid_df['positive_rate'] = covid_df.new_cases / covid_df.new_tests
print(covid_df)
```

Same as before if we want to display the whole DataFrame we need to use display from IPython.display

If we want to remove this newly created columns, we use drop method as below

```
covid_df.drop(columns=['positive_rate'], inplace=True)
covid_df
```

When you set the inplace parameter to True that means you want to apply this operation to the original DataFrame, if not so it’s returning a new DataFrame with the changes you made.

6.0 Sorting rows using column values

Let's sort the rows in the covid_df to identify the days with the highest number of cases, then chain it with the head method to list just the first ten results.

```
covid_df.sort_values('new_cases', ascending=False).head(10)
```

If we want to sort all the rows we will use IPython.display as we did before

```
from IPython.display import display
with pd.option_context('display.max_rows', 1000):
   display(covid_df.sort_values('new_cases', ascending=False))
```

7.0 Working with Dates

When we are working with dates, the date data type is an object, so Pandas does not know that this column is a date.

That means we need to convert the date data to datetime type to work with it easily.

```
covid_df['year'] = pd.DatetimeIndex(covid_df.date).year
covid_df['month'] = pd.DatetimeIndex(covid_df.date).month
covid_df['day'] = pd.DatetimeIndex(covid_df.date).day
```

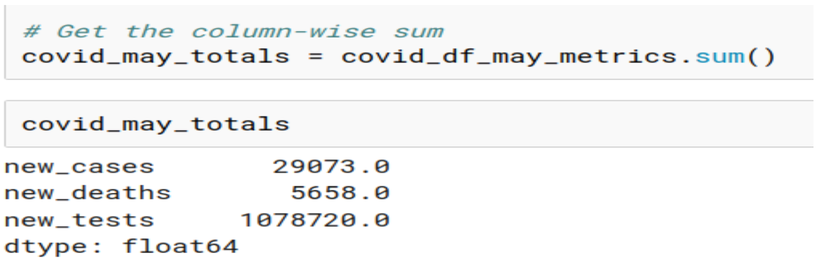
Now, let’s try to make some queries

```
# Query the rows for 2020
covid_df_may = covid_df[covid_df.year == 2020]
print(covid_df_may)
```

Or

```
# Query the rows for may
covid_df_may = covid_df[covid_df.year == 5]
print(covid_df_may)
```

```
# Get the column-wise sum
covid_may_totals = covid_df_may_metrics.sum()
```



```
# Get the column-wise sum
covid_may_totals = covid_df_may_metrics.sum()
```

```
covid_may_totals
```

```
new_cases        29073.0
new_deaths        5658.0
new_tests      1078720.0
dtype: float64
```

8.0 Grouping and aggregation

As a next step, we might want to summarize the day-wise data and create a new dataframe with month-wise data. We can use the **groupby** function to create a group for each month, select the columns we wish to aggregate, and aggregate them using the sum method.

```
covid_month_df = covid_df.groupby('month')[['new_cases', 'new_tests',
'new_deaths']].sum()
covid_month_df
```

Or instead of grouping with sum we can group with mean as below

```
covid_month_df = covid_df.groupby('month')[['new_cases', 'new_tests',
'new_deaths']].mean()
covid_month_df
```

9.0 Merging data from multiple sources

To determine other metrics like test per million, cases per million, etc., we require some more information about the country, viz. its population. Let's download another file locations.csv that contains health-related information for many countries, including Italy.

```
# merged_df = old_df.merge(new_df, on='column_name'
# the column name here represents the value which where the value in the
old dataframe equals the value in the new dataframe
merged_df = covid_df.merge(location_df, on='location')
merged_df
```

Let's make some operations on the merged dataframe

```
merged_df['CPM'] = merged_df.total_cases * 1e6 / merged_df.population
merged_df
```

10.0 Writing data back to files

After completing your analysis and adding new columns, you should write the results back to a file. Otherwise, the data will be lost when the Jupyter notebook shuts down. Before writing to file, let us first create a data frame containing just the columns we wish to record.

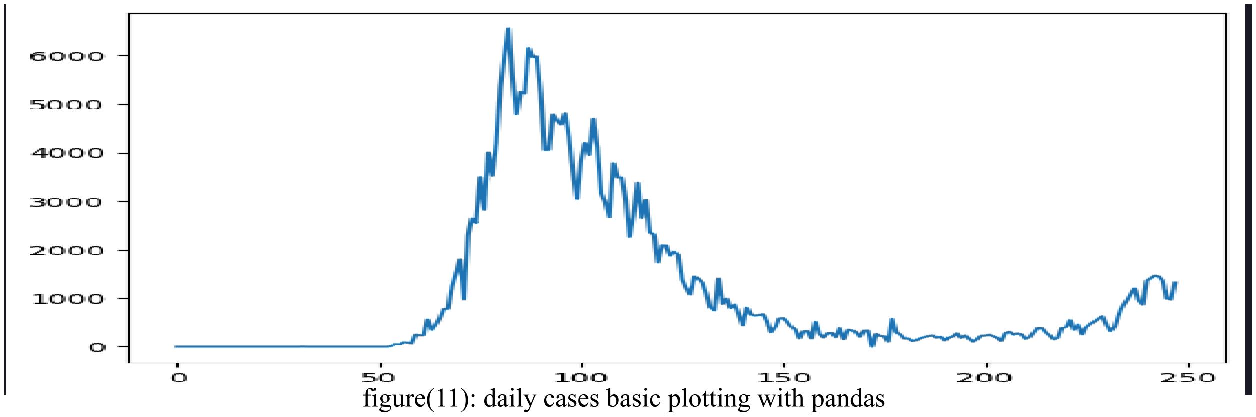
```
result_df = merged_df[['date',
                      'new_cases',
                      'total_cases',
                      'new_deaths',
                      'total_deaths',
                      'new_tests',
                      'total_tests',
                      'cases_per_million',
                      'deaths_per_million',
                      'tests_per_million']]
result_df.to_csv('results.csv', index=None)
# the index none here means that you dont want to include index
# if you want to do so, change it from None to True
```

11.0 Basic Plotting with pandas

We generally use a library like matplotlib or seaborn plot graphs within a Jupyter notebook. However, Pandas dataframes & series provide a handy .plot method for quick and easy plotting.

Let's plot a line graph showing how the number of daily cases varies over time.

```
covid_df.new_cases.plot()
```



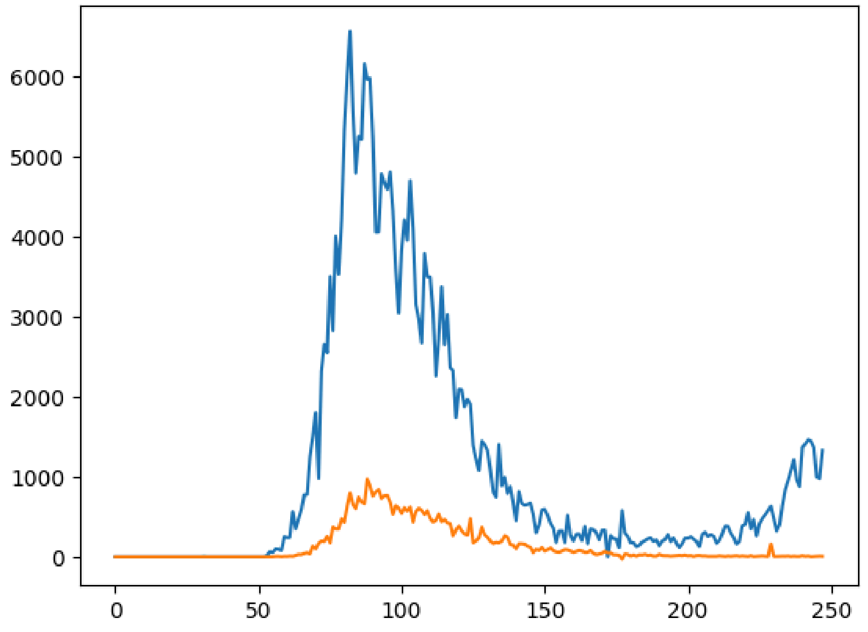
figure(11): daily cases basic plotting with pandas

While this plot shows the overall trend, it's hard to tell where the peak occurred, as there are no dates on the X-axis. We can use the date column as the index for the data frame to address this issue.

```
covid_df.set_index('date', inplace=True)
```

Now, let’s plot some columns, let’s assume we want to plot two columns on the same chart.

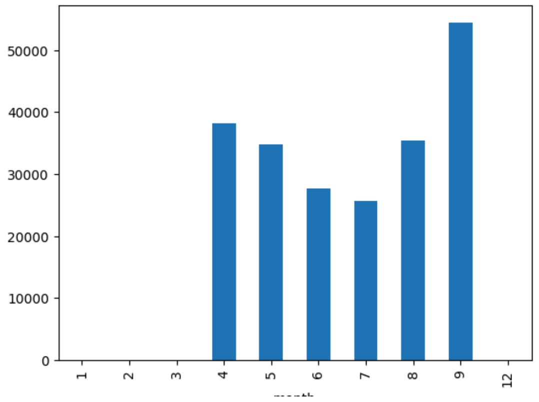
```
covid_df.new_cases.plot()
covid_df.new_deaths.plot();
```



figure(12): basic plotting with two columns

Last thing i want to mention, is we can have different types of charts

```
covid_month_df.new_tests.plot(kind='bar')
```



figure(13): basic plotting with bar chart

This chart can be as in this example bar or it can be one of the following

- line: which is the default one
- barh: a horizontal bar chart
- area: the a chart where the area will be filled with blue color

12.0 Exercises

Q1: How many countries does the dataframe contain?

```
num_countries = countries_df.location.nunique()
num_countries
# it's possible to use to below solution if you're sure that there is no
countries duplication
num_countries_2 = countries_df.location.count()
num_countries_2
```

Q2: How many countries does the dataframe contain?

```
continent = countries_df.continent.unique()
continent
```

Q3: What is the total population of all the countries listed in this dataset?

```
total_population = countries_df.population.sum()
total_population
```

Q: (Optional) What is the total population of all the countries listed in this dataset?

```
life_average = countries_df.life_expectancy.mean()
life_average
```

Q4: Create a dataframe containing 10 countries with the highest population.

```
top_ten_population = countries_df.nlargest(10, 'population')
top_ten_population
```

Q5: Add a new column in countries_df to record the overall GDP per country (product of population & per capita GDP).

```
countries_df['gdp'] = countries_df.population *
countries_df.gdp_per_capita
countries_df
```

Q: (Optional) Create a dataframe containing 10 countries with the lowest GDP per capita, among the counties with population greater than 100 million

```
lowest_gdp_per_capital_df = countries_df.nsmallest(10, 'gdp_per_capita')
lowest_gdp_per_capital_df
```

Q6: Create a data frame that counts the total countries in each continent?

```
countries_per_continent =
countries_df.groupby('continent').size().reset_index(name='total_countries
')
countries_per_continent
```

Q7: Create a data frame showing the total population of each continent.

```
continent_df =
countries_df.groupby('continent')['population'].sum().reset_index(name='to
tal_population')
continent_df
```

Q8: Count the number of countries for which the total_tests data is missing.

```
num_of_nan = covid_countries_df.total_tests.isna().sum()
num_of_nan
```

Q9: Merge countries_df with covid_data_df on the location column.

```
merged_df = covid_df.merge(covid_countries_df, on="location")
merged_df
```

Q10: Add columns tests_per_million, cases_per_million and deaths_per_million into combined_df.

```
merged_df['tests_per_million'] = merged_df['total_tests'] * 1e6 /
merged_df['population']
merged_df['cases_per_million'] = merged_df['total_cases'] * 1e6 /
merged_df['population']
merged_df['deaths_per_million'] = merged_df['deaths_cases'] * 1e6 /
merged_df['population']
merged_df
merged_df = covid_df.merge(covid_countries_df, on="location")
merged_df
```

Q11: Create a dataframe with 10 countries that have the highest number of tests per million people.

```
highest_number_of_tests_per_million = merged_df.nlargest(10,
'tests_per_million')
highest_number_of_tests_per_million
```