

python-pandas-data-analysis

June 29, 2022

1 Analyzing Tabular Data using Python and Pandas



<https://i.imgur.com/zfxLzEv.png>

This tutorial series is a beginner-friendly introduction to programming and data analysis using the Python programming language. These tutorials take a practical and coding-focused approach. The best way to learn the material is to execute the code and experiment with it yourself. Check out the full series here:

1. [First Steps with Python and Jupyter](#)
2. [A Quick Tour of Variables and Data Types](#)
3. [Branching using Conditional Statements and Loops](#)
4. [Writing Reusable Code Using Functions](#)
5. [Reading from and Writing to Files](#)
6. [Numerical Computing with Python and Numpy](#)
7. [Analyzing Tabular Data using Pandas](#)
8. [Data Visualization using Matplotlib & Seaborn](#)
9. [Exploratory Data Analysis - A Case Study](#)

This tutorial 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

1.0.1 How to run the code

This tutorial is an executable [Jupyter notebook](#) hosted on [Jovian](#). You can *run* this tutorial and experiment with the code examples in a couple of ways: *using free online resources* (recommended) or *on your computer*.

Option 1: Running using free online resources (1-click, recommended) The easiest way to start executing the code is to click the **Run** button at the top of this page and select **Run on Binder**. You can also select “Run on Colab” or “Run on Kaggle”, but you’ll need to create an account on [Google Colab](#) or [Kaggle](#) to use these platforms.

Option 2: Running on your computer locally To run the code on your computer locally, you’ll need to set up [Python](#), download the notebook and install the required libraries. We recommend using the [Conda](#) distribution of Python. Click the **Run** button at the top of this page, select the **Run Locally** option, and follow the instructions.

Jupyter Notebooks: This tutorial is a [Jupyter notebook](#) - a document made of *cells*. Each cell can contain code written in Python or explanations in plain English. You can execute code cells and view the results, e.g., numbers, messages, graphs, tables, files, etc., instantly within the notebook. Jupyter is a powerful platform for experimentation and analysis. Don’t be afraid to mess around with the code & break things - you’ll learn a lot by encountering and fixing errors. You can use the “Kernel > Restart & Clear Output” menu option to clear all outputs and start again from the top.

1.1 Reading a CSV file using Pandas

[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. Let’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  
2020-04-21,2256.0,454.0,28095.0  
2020-04-22,2729.0,534.0,44248.0  
2020-04-23,3370.0,437.0,37083.0  
2020-04-24,2646.0,464.0,95273.0  
2020-04-25,3021.0,420.0,38676.0  
2020-04-26,2357.0,415.0,24113.0  
2020-04-27,2324.0,260.0,26678.0  
2020-04-28,1739.0,333.0,37554.0  
...
```

This format of storing data is known as *comma-separated values* or CSV.

CSVs: A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. A CSV file typically stores tabular data (numbers and text) in plain text, in which case each line will have the same number of fields. ([Wikipedia](#))

We’ll download this file using the `urlretrieve` function from the `urllib.request` module.

```
[1]: from urllib.request import urlretrieve
```

```
[2]: italy_covid_url = 'https://gist.githubusercontent.com/aakashns/  
→f6a004fa20c84fec53262f9a8bfee775/raw/f309558b1cf5103424cef58e2ecb8704dc4d74c/  
→italy-covid-daywise.csv'  
  
urlretrieve(italy_covid_url, 'italy-covid-daywise.csv')
```

```
[2]: ('italy-covid-daywise.csv', <http.client.HTTPMessage at 0x7eff369affa0>)
```

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

```
[3]: !pip install pandas --upgrade --quiet
```

We can now import the `pandas` module. As a convention, it is imported with the alias `pd`.

```
[4]: import pandas as pd
```

```
[5]: covid_df = pd.read_csv('italy-covid-daywise.csv')
```

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.

```
[6]: type(covid_df)
```

```
[6]: pandas.core.frame.DataFrame
```

```
[7]: covid_df
```

```
[7]:      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
```

[248 rows x 4 columns]

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

Keep in mind that these are officially reported numbers. The actual number of cases & deaths may be higher, as not all cases are diagnosed.

We can view some basic information about the data frame using the `.info` method.

[8]: `covid_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 248 entries, 0 to 247
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   date        248 non-null    object  
 1   new_cases   248 non-null    float64 
 2   new_deaths  248 non-null    float64 
 3   new_tests   135 non-null    float64 
dtypes: float64(3), object(1)
memory usage: 7.9+ KB
```

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.

[9]: `covid_df.describe()`

```
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
```

The `columns` property contains the list of columns within the data frame.

[10]: `covid_df.columns`

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

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

[11]: `covid_df.shape`

```
[11]: (248, 4)
```

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

- `pd.read_csv` - Read data from a CSV file into a Pandas DataFrame object
- `.info()` - View basic information 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

1.1.1 Save and upload your notebook

Whether you're running this Jupyter notebook online or on your computer, it's essential to save your work from time to time. You can continue working on a saved notebook later or share it with friends and colleagues to let them execute your code. [Jovian](#) offers an easy way of saving and sharing your Jupyter notebooks online.

```
[12]: # Install the library
```

```
!pip install jovian --upgrade --quiet
```

```
[13]: import jovian
```

```
[14]: jovian.commit(project='python-pandas-data-analysis')
```

```
<IPython.core.display.Javascript object>

[jovian] Creating a new project "shubhammeena712/python-pandas-data-
analysis"
[jovian] Committed successfully! https://jovian.ai/shubhammeena712/python-
pandas-data-analysis
```

```
[14]: 'https://jovian.ai/shubhammeena712/python-pandas-data-analysis'
```

The first time you run `jovian.commit`, you'll be asked to provide an API Key to securely upload the notebook to your Jovian account. You can get the API key from your [Jovian profile page](#) after logging in / signing up.

`jovian.commit` uploads the notebook to your Jovian account, captures the Python environment, and creates a shareable link for your notebook, as shown above. You can use this link to share your work and let anyone (including you) run your notebooks and reproduce your work.

1.2 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.

```
[15]: # 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).

```
[16]: # 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.

```
[17]: covid_data_dict['new_cases']
```

```
[17]: [1444, 1365, 996, 975, 1326]
```

```
[18]: covid_df['new_cases']
```

```
[18]: 0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
...
243    1444.0
244    1365.0
245    996.0
246    975.0
247    1326.0
Name: new_cases, Length: 248, dtype: float64
```

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

```
[19]: type(covid_df['new_cases'])
```

```
[19]: pandas.core.series.Series
```

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

```
[20]: covid_df['new_cases'][246]
```

```
[20]: 975.0
```

```
[21]: covid_df['new_tests'][240]
```

```
[21]: 57640.0
```

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

```
[22]: covid_df.at[246, 'new_cases']
```

```
[22]: 975.0
```

```
[23]: covid_df.at[240, 'new_tests']
```

```
[23]: 57640.0
```

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.

```
[24]: covid_df.new_cases
```

```
[24]: 0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
...
243    1444.0
244    1365.0
245    996.0
246    975.0
247    1326.0
Name: new_cases, Length: 248, dtype: float64
```

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.

```
[25]: cases_df = covid_df[['date', 'new_cases']]
cases_df
```

```
[25]:      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
```

```
[248 rows x 2 columns]
```

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.

```
[26]: 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.

To access a specific row of data, Pandas provides the `.loc` method.

```
[27]: covid_df
```

```
[27]:      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
```

```
[248 rows x 4 columns]
```

```
[28]: covid_df.loc[243]
```

```
[28]:      date          2020-08-30
new_cases        1444.0
new_deaths        1.0
new_tests        53541.0
```

```
Name: 243, dtype: object
```

Each retrieved row is also a Series object.

```
[29]: type(covid_df.loc[243])
```

```
[29]: pandas.core.series.Series
```

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

```
[30]: covid_df.head(5)
```

```
[30]:      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
```

```
[31]: covid_df.tail(4)
```

```
[31]:      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
```

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.

```
[32]: covid_df.at[0, 'new_tests']
```

```
[32]: nan
```

```
[33]: type(covid_df.at[0, 'new_tests'])
```

```
[33]: numpy.float64
```

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. 93,531 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.

```
[34]: covid_df.new_tests.first_valid_index()
```

```
[34]: 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.

```
[35]: covid_df.loc[108:113]
```

```
[35]:      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
```

We can use the .sample method to retrieve a random sample of rows from the data frame.

```
[36]: covid_df.sample(10)
```

```
[36]:      date  new_cases  new_deaths  new_tests
132  2020-05-11    802.0     165.0  25823.0
55   2020-02-24     53.0      0.0       NaN
95   2020-04-04   4585.0     764.0       NaN
191  2020-07-09    193.0     15.0   29947.0
93   2020-04-02   4782.0     727.0       NaN
163  2020-06-11    202.0     71.0   32991.0
16   2020-01-16     0.0      0.0       NaN
4    2020-01-04     0.0      0.0       NaN
17   2020-01-17     0.0      0.0       NaN
219  2020-08-06    384.0     10.0   32169.0
```

Notice that even though we have taken a random sample, each row's original index is preserved - this is a useful property of data frames.

Here's a summary of the functions & methods we looked at in this section:

- covid_df['new_cases'] - Retrieving columns as a Series using the column name
- new_cases[243] - Retrieving values from a Series using an index
- covid_df.at[243, 'new_cases'] - Retrieving a single value from a data frame
- covid_df.copy() - Creating a deep copy of a data frame
- covid_df.loc[243] - Retrieving a row or range of rows of data from the data frame
- head, tail, and sample - Retrieving multiple rows of data from the data frame
- covid_df.new_tests.first_valid_index - Finding the first non-empty index in a series

Let's save a snapshot of our notebook before continuing.

```
[37]: import jovian
```

```
[38]: jovian.commit()
```

```
<IPython.core.display.Javascript object>
```

```
[jovian] Updating notebook "shubhammeena712/python-pandas-data-analysis" on  
https://jovian.ai  
[jovian] Committed successfully! https://jovian.ai/shubhammeena712/python-  
pandas-data-analysis  
[38]: 'https://jovian.ai/shubhammeena712/python-pandas-data-analysis'
```

1.3 Analyzing data from data frames

Let's try to answer some questions about our data.

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

Similar to Numpy arrays, a Pandas series supports the `sum` method to answer these questions.

```
[39]: total_cases = covid_df.new_cases.sum()  
total_deaths = covid_df.new_deaths.sum()
```

```
[40]: print('The number of reported cases is {} and the number of reported deaths is  
→{}.'.format(int(total_cases), int(total_deaths)))
```

The number of reported cases is 271515 and the number of reported deaths is 35497.

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

```
[41]: death_rate = covid_df.new_deaths.sum() / covid_df.new_cases.sum()
```

```
[42]: print("The overall reported death rate in Italy is {:.2f} %."  
→format(death_rate*100))
```

The overall reported death rate in Italy is 13.07 %.

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

```
[43]: initial_tests = 935310  
total_tests = initial_tests + covid_df.new_tests.sum()
```

```
[44]: total_tests
```

```
[44]: 5214766.0
```

Q: What fraction of tests returned a positive result?

```
[45]: positive_rate = total_cases / total_tests
```

```
[50]: print('{:.3f}% of tests in Italy led to a positive diagnosis.'  
→format(positive_rate*100))
```

5.207% of tests in Italy led to a positive diagnosis.

Try asking and answering some more questions about the data using the empty cells below.

[]:

[]:

Let's save and commit our work before continuing.

[51]: `import jovian`

[52]: `jovian.commit()`

```
<IPython.core.display.Javascript object>  
[jovian] Updating notebook "shubhammeena712/python-pandas-data-analysis" on  
https://jovian.ai  
[jovian] Committed successfully! https://jovian.ai/shubhammeena712/python-  
pandas-data-analysis
```

[52]: `'https://jovian.ai/shubhammeena712/python-pandas-data-analysis'`

1.4 Querying and sorting rows

Let's say we want only want to look at the days which had more than 1000 reported cases. We can use a boolean expression to check which rows satisfy this criterion.

[53]: `high_new_cases = covid_df.new_cases > 1000`

[54]: `high_new_cases`

```
[54]: 0      False  
1      False  
2      False  
3      False  
4      False  
...  
243     True  
244     True  
245     False  
246     False  
247     True  
Name: new_cases, Length: 248, dtype: bool
```

The boolean expression returns a series containing True and False boolean values. You can use this series to select a subset of rows from the original dataframe, corresponding to the True values in the series.

[55]: `covid_df[high_new_cases]`

```
[55]:      date  new_cases  new_deaths  new_tests
 68  2020-03-08    1247.0       36.0      NaN
 69  2020-03-09    1492.0      133.0      NaN
 70  2020-03-10    1797.0       98.0      NaN
 72  2020-03-12   2313.0      196.0      NaN
 73  2020-03-13   2651.0      189.0      NaN
 ...
241 2020-08-28   1409.0        5.0    65135.0
242 2020-08-29   1460.0        9.0    64294.0
243 2020-08-30   1444.0       1.0    53541.0
244 2020-08-31   1365.0        4.0    42583.0
247 2020-09-03   1326.0        6.0      NaN
```

[72 rows x 4 columns]

We can write this succinctly on a single line by passing the boolean expression as an index to the data frame.

```
[56]: high_cases_df = covid_df[covid_df.new_cases > 1000]
```

```
[57]: high_cases_df
```

```
[57]:      date  new_cases  new_deaths  new_tests
 68  2020-03-08    1247.0       36.0      NaN
 69  2020-03-09    1492.0      133.0      NaN
 70  2020-03-10    1797.0       98.0      NaN
 72  2020-03-12   2313.0      196.0      NaN
 73  2020-03-13   2651.0      189.0      NaN
 ...
241 2020-08-28   1409.0        5.0    65135.0
242 2020-08-29   1460.0        9.0    64294.0
243 2020-08-30   1444.0       1.0    53541.0
244 2020-08-31   1365.0        4.0    42583.0
247 2020-09-03   1326.0        6.0      NaN
```

[72 rows x 4 columns]

The data frame contains 72 rows, but only the first & last five rows are displayed by default with Jupyter for brevity. We can change some display options to view all the rows.

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

```
      date  new_cases  new_deaths  new_tests
 68  2020-03-08    1247.0       36.0      NaN
 69  2020-03-09    1492.0      133.0      NaN
 70  2020-03-10    1797.0       98.0      NaN
 72  2020-03-12   2313.0      196.0      NaN
```

73	2020-03-13	2651.0	189.0	NaN
74	2020-03-14	2547.0	252.0	NaN
75	2020-03-15	3497.0	173.0	NaN
76	2020-03-16	2823.0	370.0	NaN
77	2020-03-17	4000.0	347.0	NaN
78	2020-03-18	3526.0	347.0	NaN
79	2020-03-19	4207.0	473.0	NaN
80	2020-03-20	5322.0	429.0	NaN
81	2020-03-21	5986.0	625.0	NaN
82	2020-03-22	6557.0	795.0	NaN
83	2020-03-23	5560.0	649.0	NaN
84	2020-03-24	4789.0	601.0	NaN
85	2020-03-25	5249.0	743.0	NaN
86	2020-03-26	5210.0	685.0	NaN
87	2020-03-27	6153.0	660.0	NaN
88	2020-03-28	5959.0	971.0	NaN
89	2020-03-29	5974.0	887.0	NaN
90	2020-03-30	5217.0	758.0	NaN
91	2020-03-31	4050.0	810.0	NaN
92	2020-04-01	4053.0	839.0	NaN
93	2020-04-02	4782.0	727.0	NaN
94	2020-04-03	4668.0	760.0	NaN
95	2020-04-04	4585.0	764.0	NaN
96	2020-04-05	4805.0	681.0	NaN
97	2020-04-06	4316.0	527.0	NaN
98	2020-04-07	3599.0	636.0	NaN
99	2020-04-08	3039.0	604.0	NaN
100	2020-04-09	3836.0	540.0	NaN
101	2020-04-10	4204.0	612.0	NaN
102	2020-04-11	3951.0	570.0	NaN
103	2020-04-12	4694.0	619.0	NaN
104	2020-04-13	4092.0	431.0	NaN
105	2020-04-14	3153.0	564.0	NaN
106	2020-04-15	2972.0	604.0	NaN
107	2020-04-16	2667.0	578.0	NaN
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
114	2020-04-23	3370.0	437.0	37083.0
115	2020-04-24	2646.0	464.0	95273.0
116	2020-04-25	3021.0	420.0	38676.0
117	2020-04-26	2357.0	415.0	24113.0
118	2020-04-27	2324.0	260.0	26678.0
119	2020-04-28	1739.0	333.0	37554.0
120	2020-04-29	2091.0	382.0	38589.0

121	2020-04-30	2086.0	323.0	41441.0
122	2020-05-01	1872.0	285.0	43732.0
123	2020-05-02	1965.0	269.0	31231.0
124	2020-05-03	1900.0	474.0	27047.0
125	2020-05-04	1389.0	174.0	22999.0
126	2020-05-05	1221.0	195.0	32211.0
127	2020-05-06	1075.0	236.0	37771.0
128	2020-05-07	1444.0	369.0	13665.0
129	2020-05-08	1401.0	274.0	45428.0
130	2020-05-09	1327.0	243.0	36091.0
131	2020-05-10	1083.0	194.0	31384.0
134	2020-05-13	1402.0	172.0	37049.0
236	2020-08-23	1071.0	3.0	47463.0
237	2020-08-24	1209.0	7.0	33358.0
240	2020-08-27	1366.0	13.0	57640.0
241	2020-08-28	1409.0	5.0	65135.0
242	2020-08-29	1460.0	9.0	64294.0
243	2020-08-30	1444.0	1.0	53541.0
244	2020-08-31	1365.0	4.0	42583.0
247	2020-09-03	1326.0	6.0	NaN

We can also formulate more complex queries that involve multiple columns. As an example, let's try to determine the days when the ratio of cases reported to tests conducted is higher than the overall `positive_rate`.

```
[59]: positive_rate
```

```
[59]: 0.05206657403227681
```

```
[60]: high_ratio_df = covid_df[covid_df.new_cases / covid_df.new_tests > positive_rate]
```

```
[61]: high_ratio_df
```

	date	new_cases	new_deaths	new_tests
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
114	2020-04-23	3370.0	437.0	37083.0
116	2020-04-25	3021.0	420.0	38676.0
117	2020-04-26	2357.0	415.0	24113.0
118	2020-04-27	2324.0	260.0	26678.0
120	2020-04-29	2091.0	382.0	38589.0
123	2020-05-02	1965.0	269.0	31231.0
124	2020-05-03	1900.0	474.0	27047.0
125	2020-05-04	1389.0	174.0	22999.0
128	2020-05-07	1444.0	369.0	13665.0

The result of performing an operation on two columns is a new series.

```
[62]: covid_df.new_cases / covid_df.new_tests
```

```
[62]: 0      NaN  
1      NaN  
2      NaN  
3      NaN  
4      NaN  
...  
243    0.026970  
244    0.032055  
245    0.018311  
246      NaN  
247      NaN  
Length: 248, dtype: float64
```

We can use this series to add a new column to the data frame.

```
[74]: covid_df['positive_rate'] = covid_df.new_cases / covid_df.new_tests
```

```
[75]: covid_df
```

```
[75]:      date  new_cases  new_deaths  new_tests  positive_rate  
0   2019-12-31      0.0       0.0      NaN        NaN  
1   2020-01-01      0.0       0.0      NaN        NaN  
2   2020-01-02      0.0       0.0      NaN        NaN  
3   2020-01-03      0.0       0.0      NaN        NaN  
4   2020-01-04      0.0       0.0      NaN        NaN  
...     ...     ...     ...     ...     ...  
243  2020-08-30    1444.0      1.0    53541.0    0.026970  
244  2020-08-31    1365.0      4.0    42583.0    0.032055  
245  2020-09-01    996.0      6.0    54395.0    0.018311  
246  2020-09-02    975.0      8.0      NaN        NaN  
247  2020-09-03   1326.0      6.0      NaN        NaN
```

[248 rows x 5 columns]

However, keep in mind that sometimes it takes a few days to get the results for a test, so we can't compare the number of new cases with the number of tests conducted on the same day. Any inference based on this `positive_rate` column is likely to be incorrect. It's essential to watch out for such subtle relationships that are often not conveyed within the CSV file and require some external context. It's always a good idea to read through the documentation provided with the dataset or ask for more information.

For now, let's remove the `positive_rate` column using the `drop` method.

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

Can you figure the purpose of the `inplace` argument?

1.4.1 Sorting rows using column values

The rows can also be sorted by a specific column using `.sort_values`. Let's sort to identify the days with the highest number of cases, then chain it with the `head` method to list just the first ten results.

```
[77]: covid_df.sort_values('new_cases', ascending=False).head(10)
```

```
[77]:      date  new_cases  new_deaths  new_tests
 82  2020-03-22    6557.0     795.0      NaN
 87  2020-03-27    6153.0     660.0      NaN
 81  2020-03-21    5986.0     625.0      NaN
 89  2020-03-29    5974.0     887.0      NaN
 88  2020-03-28    5959.0     971.0      NaN
 83  2020-03-23    5560.0     649.0      NaN
 80  2020-03-20    5322.0     429.0      NaN
 85  2020-03-25    5249.0     743.0      NaN
 90  2020-03-30    5217.0     758.0      NaN
 86  2020-03-26    5210.0     685.0      NaN
```

It looks like the last two weeks of March had the highest number of daily cases. Let's compare this to the days where the highest number of deaths were recorded.

```
[78]: covid_df.sort_values('new_deaths', ascending=False).head(10)
```

```
[78]:      date  new_cases  new_deaths  new_tests
 88  2020-03-28    5959.0     971.0      NaN
 89  2020-03-29    5974.0     887.0      NaN
 92  2020-04-01    4053.0     839.0      NaN
 91  2020-03-31    4050.0     810.0      NaN
 82  2020-03-22    6557.0     795.0      NaN
 95  2020-04-04    4585.0     764.0      NaN
 94  2020-04-03    4668.0     760.0      NaN
 90  2020-03-30    5217.0     758.0      NaN
 85  2020-03-25    5249.0     743.0      NaN
 93  2020-04-02    4782.0     727.0      NaN
```

It appears that daily deaths hit a peak just about a week after the peak in daily new cases.

Let's also look at the days with the least number of cases. We might expect to see the first few days of the year on this list.

```
[79]: covid_df.sort_values('new_cases').head(10)
```

```
[79]:      date  new_cases  new_deaths  new_tests
172  2020-06-20    -148.0      47.0   29875.0
  0   2019-12-31      0.0       0.0      NaN
 29   2020-01-29      0.0       0.0      NaN
 30   2020-01-30      0.0       0.0      NaN
 32   2020-02-01      0.0       0.0      NaN
```

```
33    2020-02-02        0.0        0.0      NaN
34    2020-02-03        0.0        0.0      NaN
36    2020-02-05        0.0        0.0      NaN
37    2020-02-06        0.0        0.0      NaN
38    2020-02-07        0.0        0.0      NaN
```

It seems like the count of new cases on Jun 20, 2020, was -148, a negative number! Not something we might have expected, but that's the nature of real-world data. It could be a data entry error, or the government may have issued a correction to account for miscounting in the past. Can you dig through news articles online and figure out why the number was negative?

Let's look at some days before and after Jun 20, 2020.

```
[80]: covid_df.loc[169:175]
```

```
[80]:      date  new_cases  new_deaths  new_tests
169  2020-06-17     210.0      34.0   33957.0
170  2020-06-18     328.0      43.0   32921.0
171  2020-06-19     331.0      66.0   28570.0
172  2020-06-20    -148.0      47.0   29875.0
173  2020-06-21     264.0      49.0   24581.0
174  2020-06-22     224.0      24.0   16152.0
175  2020-06-23     221.0      23.0   23225.0
```

For now, let's assume this was indeed a data entry error. We can use one of the following approaches for dealing with the missing or faulty value: 1. Replace it with 0. 2. Replace it with the average of the entire column 3. Replace it with the average of the values on the previous & next date 4. Discard the row entirely

Which approach you pick requires some context about the data and the problem. In this case, since we are dealing with data ordered by date, we can go ahead with the third approach.

You can use the `.at` method to modify a specific value within the dataframe.

```
[81]: covid_df.at[172, 'new_cases'] = (covid_df.at[171, 'new_cases'] + covid_df.
                                         →at[173, 'new_cases'])/2
```

Here's a summary of the functions & methods we looked at in this section:

- `covid_df.new_cases.sum()` - Computing the sum of values in a column or series
- `covid_df[covid_df.new_cases > 1000]` - Querying a subset of rows satisfying the chosen criteria using boolean expressions
- `df['pos_rate'] = df.new_cases / df.new_tests` - Adding new columns by combining data from existing columns
- `covid_df.drop('positive_rate')` - Removing one or more columns from the data frame
- `sort_values` - Sorting the rows of a data frame using column values
- `covid_df.at[172, 'new_cases'] = ...` - Replacing a value within the data frame

Let's save and commit our work before continuing.

```
[82]: import jovian
```

```
[83]: jovian.commit()
```

```
<IPython.core.display.Javascript object>  
[jovian] Updating notebook "shubhammeena712/python-pandas-data-analysis" on  
https://jovian.ai  
[jovian] Committed successfully! https://jovian.ai/shubhammeena712/python-  
pandas-data-analysis
```

```
[83]: 'https://jovian.ai/shubhammeena712/python-pandas-data-analysis'
```

1.5 Working with dates

While we've looked at overall numbers for the cases, tests, positive rate, etc., it would also be useful to study these numbers on a month-by-month basis. The date column might come in handy here, as Pandas provides many utilities for working with dates.

```
[84]: covid_df.date
```

```
[84]: 0      2019-12-31  
1      2020-01-01  
2      2020-01-02  
3      2020-01-03  
4      2020-01-04  
...  
243    2020-08-30  
244    2020-08-31  
245    2020-09-01  
246    2020-09-02  
247    2020-09-03  
Name: date, Length: 248, dtype: object
```

The data type of date is currently `object`, so Pandas does not know that this column is a date. We can convert it into a `datetime` column using the `pd.to_datetime` method.

```
[85]: covid_df['date'] = pd.to_datetime(covid_df.date)
```

```
[86]: covid_df['date']
```

```
[86]: 0      2019-12-31  
1      2020-01-01  
2      2020-01-02  
3      2020-01-03  
4      2020-01-04  
...  
243    2020-08-30  
244    2020-08-31  
245    2020-09-01  
246    2020-09-02
```

```
247    2020-09-03  
Name: date, Length: 248, dtype: datetime64[ns]
```

You can see that it now has the datatype `datetime64`. We can now extract different parts of the data into separate columns, using the `DatetimeIndex` class ([view docs](#)).

```
[87]: 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  
covid_df['weekday'] = pd.DatetimeIndex(covid_df.date).weekday
```

```
[88]: covid_df
```

```
[88]:      date  new_cases  new_deaths  new_tests  year  month  day  weekday  
0    2019-12-31      0.0       0.0      NaN  2019     12    31       1  
1    2020-01-01      0.0       0.0      NaN  2020      1     1       2  
2    2020-01-02      0.0       0.0      NaN  2020      1     2       3  
3    2020-01-03      0.0       0.0      NaN  2020      1     3       4  
4    2020-01-04      0.0       0.0      NaN  2020      1     4       5  
..  
243  2020-08-30   1444.0       1.0   53541.0  2020      8    30       6  
244  2020-08-31   1365.0       4.0   42583.0  2020      8    31       0  
245  2020-09-01   996.0       6.0   54395.0  2020      9     1       1  
246  2020-09-02   975.0       8.0      NaN  2020      9     2       2  
247  2020-09-03   1326.0       6.0      NaN  2020      9     3       3
```

[248 rows x 8 columns]

Let's check the overall metrics for May. We can query the rows for May, choose a subset of columns, and use the `sum` method to aggregate each selected column's values.

```
[91]: # Query the rows for May  
covid_df_may = covid_df[covid_df.month == 5]
```

```
[92]: covid_df_may
```

```
[92]:      date  new_cases  new_deaths  new_tests  year  month  day  weekday  
122  2020-05-01   1872.0     285.0   43732.0  2020      5     1       4  
123  2020-05-02   1965.0     269.0   31231.0  2020      5     2       5  
124  2020-05-03   1900.0     474.0   27047.0  2020      5     3       6  
125  2020-05-04   1389.0     174.0   22999.0  2020      5     4       0  
126  2020-05-05   1221.0     195.0   32211.0  2020      5     5       1  
127  2020-05-06   1075.0     236.0   37771.0  2020      5     6       2  
128  2020-05-07   1444.0     369.0   13665.0  2020      5     7       3  
129  2020-05-08   1401.0     274.0   45428.0  2020      5     8       4  
130  2020-05-09   1327.0     243.0   36091.0  2020      5     9       5  
131  2020-05-10   1083.0     194.0   31384.0  2020      5    10       6  
132  2020-05-11   802.0      165.0   25823.0  2020      5    11       0
```

133	2020-05-12	744.0	179.0	39620.0	2020	5	12	1
134	2020-05-13	1402.0	172.0	37049.0	2020	5	13	2
135	2020-05-14	888.0	195.0	41131.0	2020	5	14	3
136	2020-05-15	992.0	262.0	39027.0	2020	5	15	4
137	2020-05-16	789.0	242.0	40657.0	2020	5	16	5
138	2020-05-17	875.0	153.0	33505.0	2020	5	17	6
139	2020-05-18	675.0	145.0	26101.0	2020	5	18	0
140	2020-05-19	451.0	99.0	40226.0	2020	5	19	1
141	2020-05-20	813.0	162.0	38617.0	2020	5	20	2
142	2020-05-21	665.0	161.0	40644.0	2020	5	21	3
143	2020-05-22	642.0	156.0	42987.0	2020	5	22	4
144	2020-05-23	652.0	130.0	42579.0	2020	5	23	5
145	2020-05-24	669.0	119.0	34206.0	2020	5	24	6
146	2020-05-25	531.0	50.0	20676.0	2020	5	25	0
147	2020-05-26	300.0	92.0	33944.0	2020	5	26	1
148	2020-05-27	397.0	78.0	37299.0	2020	5	27	2
149	2020-05-28	584.0	117.0	39838.0	2020	5	28	3
150	2020-05-29	593.0	70.0	38233.0	2020	5	29	4
151	2020-05-30	516.0	87.0	36051.0	2020	5	30	5
152	2020-05-31	416.0	111.0	28948.0	2020	5	31	6

[93]: *# Extract the subset of columns to be aggregated*
covid_df_may_metrics = covid_df_may[['new_cases', 'new_deaths', 'new_tests']]

[94]: covid_df_may_metrics

	new_cases	new_deaths	new_tests
122	1872.0	285.0	43732.0
123	1965.0	269.0	31231.0
124	1900.0	474.0	27047.0
125	1389.0	174.0	22999.0
126	1221.0	195.0	32211.0
127	1075.0	236.0	37771.0
128	1444.0	369.0	13665.0
129	1401.0	274.0	45428.0
130	1327.0	243.0	36091.0
131	1083.0	194.0	31384.0
132	802.0	165.0	25823.0
133	744.0	179.0	39620.0
134	1402.0	172.0	37049.0
135	888.0	195.0	41131.0
136	992.0	262.0	39027.0
137	789.0	242.0	40657.0
138	875.0	153.0	33505.0
139	675.0	145.0	26101.0
140	451.0	99.0	40226.0
141	813.0	162.0	38617.0

```
142      665.0      161.0    40644.0
143      642.0      156.0    42987.0
144      652.0      130.0    42579.0
145      669.0      119.0    34206.0
146      531.0       50.0    20676.0
147      300.0       92.0    33944.0
148      397.0       78.0    37299.0
149      584.0      117.0    39838.0
150      593.0       70.0    38233.0
151      516.0       87.0    36051.0
152      416.0      111.0    28948.0
```

```
[95]: # Get the column-wise sum
covid_may_totals = covid_df_may_metrics.sum()
```

```
[96]: covid_may_totals
```

```
[96]: new_cases      29073.0
new_deaths       5658.0
new_tests        1078720.0
dtype: float64
```

```
[97]: type(covid_may_totals)
```

```
[97]: pandas.core.series.Series
```

We can also combine the above operations into a single statement.

```
[98]: covid_df[covid_df.month == 5][['new_cases', 'new_deaths', 'new_tests']].sum()
```

```
[98]: new_cases      29073.0
new_deaths       5658.0
new_tests        1078720.0
dtype: float64
```

As another example, let's check if the number of cases reported on Sundays is higher than the average number of cases reported every day. This time, we might want to aggregate columns using the `.mean` method.

```
[99]: # Overall average
covid_df.new_cases.mean()
```

```
[99]: 1096.6149193548388
```

```
[100]: # Average for Sundays
covid_df[covid_df.weekday == 6].new_cases.mean()
```

```
[100]: 1247.2571428571428
```

It seems like more cases were reported on Sundays compared to other days.

Try asking and answering some more date-related questions about the data using the cells below.

[]:

[]:

Let's save and commit our work before continuing.

[101]: `import jovian`

[102]: `jovian.commit()`

```
<IPython.core.display.Javascript object>

[jovian] Updating notebook "shubhammeena712/python-pandas-data-analysis" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/shubhammeena712/python-
pandas-data-analysis
```

[102]: `'https://jovian.ai/shubhammeena712/python-pandas-data-analysis'`

1.6 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.

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

[104]: `covid_month_df`

month	new_cases	new_deaths	new_tests
1	3.0	0.0	0.0
2	885.0	21.0	0.0
3	100851.0	11570.0	0.0
4	101852.0	16091.0	419591.0
5	29073.0	5658.0	1078720.0
6	8217.5	1404.0	830354.0
7	6722.0	388.0	797692.0
8	21060.0	345.0	1098704.0
9	3297.0	20.0	54395.0
12	0.0	0.0	0.0

The result is a new data frame that uses unique values from the column passed to groupby as the index. Grouping and aggregation is a powerful method for progressively summarizing data into smaller data frames.

Instead of aggregating by sum, you can also aggregate by other measures like mean. Let's compute the average number of daily new cases, deaths, and tests for each month.

```
[105]: covid_month_mean_df = covid_df.groupby('month')[['new_cases', 'new_deaths',  
    →'new_tests']].mean()
```

```
[106]: covid_month_mean_df
```

```
[106]:      new_cases  new_deaths  new_tests  
month  
1          0.096774   0.000000       NaN  
2         30.517241   0.724138       NaN  
3        3253.258065  373.225806       NaN  
4       3395.066667  536.366667  38144.636364  
5        937.838710 182.516129  34797.419355  
6       273.916667  46.800000  27678.466667  
7       216.838710 12.516129  25732.000000  
8       679.354839 11.129032  35442.064516  
9      1099.000000  6.666667  54395.000000  
12      0.000000   0.000000       NaN
```

Apart from grouping, another form of aggregation is the running or cumulative sum of cases, tests, or death up to each row's date. We can use the `cumsum` method to compute the cumulative sum of a column as a new series. Let's add three new columns: `total_cases`, `total_deaths`, and `total_tests`.

```
[107]: covid_df['total_cases'] = covid_df.new_cases.cumsum()
```

```
[108]: covid_df['total_deaths'] = covid_df.new_deaths.cumsum()
```

```
[109]: covid_df['total_tests'] = covid_df.new_tests.cumsum() + initial_tests
```

We've also included the initial test count in `total_test` to account for tests conducted before daily reporting was started.

```
[110]: covid_df
```

```
[110]:      date  new_cases  new_deaths  new_tests  year  month  day  weekday  \\\n0  2019-12-31      0.0      0.0       NaN  2019     12    31       1  
1  2020-01-01      0.0      0.0       NaN  2020      1    1       2  
2  2020-01-02      0.0      0.0       NaN  2020      1    2       3  
3  2020-01-03      0.0      0.0       NaN  2020      1    3       4  
4  2020-01-04      0.0      0.0       NaN  2020      1    4       5  
..    ...     ...     ...     ...     ...     ...     ...     ...  
243 2020-08-30  1444.0      1.0  53541.0  2020      8    30       6  
244 2020-08-31  1365.0      4.0  42583.0  2020      8    31       0  
245 2020-09-01   996.0      6.0  54395.0  2020      9    1       1  
246 2020-09-02   975.0      8.0       NaN  2020      9    2       2  
247 2020-09-03  1326.0      6.0       NaN  2020      9    3       3  
  
total_cases  total_deaths  total_tests
```

```

0          0.0        0.0      NaN
1          0.0        0.0      NaN
2          0.0        0.0      NaN
3          0.0        0.0      NaN
4          0.0        0.0      NaN
...
243     267298.5    35473.0   5117788.0
244     268663.5    35477.0   5160371.0
245     269659.5    35483.0   5214766.0
246     270634.5    35491.0      NaN
247     271960.5    35497.0      NaN

```

[248 rows x 11 columns]

Notice how the NaN values in the total_tests column remain unaffected.

1.7 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.

```
[111]: urlretrieve('https://gist.githubusercontent.com/aakashns/  
→8684589ef4f266116cdce023377fc9c8/raw/99ce3826b2a9d1e6d0bde7e9e559fc8b6e9ac88b/  
→locations.csv',  
'locations.csv')
```

```
[111]: ('locations.csv', <http.client.HTTPMessage at 0x7effe14c0a0>)
```

```
[112]: locations_df = pd.read_csv('locations.csv')
```

```
[113]: locations_df
```

```

[113]:      location continent  population  life_expectancy \
0      Afghanistan      Asia  3.892834e+07       64.83
1          Albania     Europe  2.877800e+06       78.57
2          Algeria     Africa  4.385104e+07       76.88
3         Andorra     Europe  7.726500e+04       83.73
4          Angola     Africa  3.286627e+07       61.15
...
207         Yemen      Asia  2.982597e+07       66.12
208        Zambia     Africa  1.838396e+07       63.89
209        Zimbabwe     Africa  1.486293e+07       61.49
210        World        NaN  7.794799e+09       72.58
211 International      NaN        NaN        NaN

      hospital_beds_per_thousand  gdp_per_capita
0                         0.500           1803.987

```

```

1           2.890    11803.431
2           1.900    13913.839
3             NaN        NaN
4             NaN    5819.495
..            ...
207          0.700    1479.147
208          2.000    3689.251
209          1.700    1899.775
210          2.705   15469.207
211           NaN        NaN

```

[212 rows x 6 columns]

```
[114]: locations_df[locations_df.location == "Italy"]
```

```

[114]:   location continent population life_expectancy \
97      Italy      Europe   60461828.0               83.51

           hospital_beds_per_thousand gdp_per_capita
97                  3.18            35220.084

```

We can merge this data into our existing data frame by adding more columns. However, to merge two data frames, we need at least one common column. Let's insert a `location` column in the `covid_df` dataframe with all values set to "Italy".

```
[115]: covid_df['location'] = "Italy"
```

```
[116]: covid_df
```

```

[116]:   date new_cases new_deaths new_tests year month day weekday \
0  2019-12-31      0.0      0.0      NaN  2019    12   31       1
1  2020-01-01      0.0      0.0      NaN  2020     1    1       2
2  2020-01-02      0.0      0.0      NaN  2020     1    2       3
3  2020-01-03      0.0      0.0      NaN  2020     1    3       4
4  2020-01-04      0.0      0.0      NaN  2020     1    4       5
..      ...
243 2020-08-30    1444.0      1.0    53541.0  2020     8   30       6
244 2020-08-31    1365.0      4.0    42583.0  2020     8   31       0
245 2020-09-01    996.0      6.0    54395.0  2020     9    1       1
246 2020-09-02    975.0      8.0      NaN  2020     9    2       2
247 2020-09-03   1326.0      6.0      NaN  2020     9    3       3

           total_cases total_deaths total_tests location
0            0.0        0.0      NaN    Italy
1            0.0        0.0      NaN    Italy
2            0.0        0.0      NaN    Italy
3            0.0        0.0      NaN    Italy
4            0.0        0.0      NaN    Italy

```

```

...
243    267298.5      35473.0    5117788.0    Italy
244    268663.5      35477.0    5160371.0    Italy
245    269659.5      35483.0    5214766.0    Italy
246    270634.5      35491.0        NaN    Italy
247    271960.5      35497.0        NaN    Italy

```

[248 rows x 12 columns]

We can now add the columns from locations_df into covid_df using the .merge method.

[117]: merged_df = covid_df.merge(locations_df, on="location")

[118]: merged_df

	date	new_cases	new_deaths	new_tests	year	month	day	weekday	\
0	2019-12-31	0.0	0.0	NaN	2019	12	31	1	
1	2020-01-01	0.0	0.0	NaN	2020	1	1	2	
2	2020-01-02	0.0	0.0	NaN	2020	1	2	3	
3	2020-01-03	0.0	0.0	NaN	2020	1	3	4	
4	2020-01-04	0.0	0.0	NaN	2020	1	4	5	
..
243	2020-08-30	1444.0	1.0	53541.0	2020	8	30	6	
244	2020-08-31	1365.0	4.0	42583.0	2020	8	31	0	
245	2020-09-01	996.0	6.0	54395.0	2020	9	1	1	
246	2020-09-02	975.0	8.0	NaN	2020	9	2	2	
247	2020-09-03	1326.0	6.0	NaN	2020	9	3	3	
		total_cases	total_deaths	total_tests	location	continent	population	\	
0		0.0	0.0	NaN	Italy	Europe	60461828.0		
1		0.0	0.0	NaN	Italy	Europe	60461828.0		
2		0.0	0.0	NaN	Italy	Europe	60461828.0		
3		0.0	0.0	NaN	Italy	Europe	60461828.0		
4		0.0	0.0	NaN	Italy	Europe	60461828.0		
..
243	267298.5	35473.0	5117788.0	Italy	Europe	60461828.0			
244	268663.5	35477.0	5160371.0	Italy	Europe	60461828.0			
245	269659.5	35483.0	5214766.0	Italy	Europe	60461828.0			
246	270634.5	35491.0	NaN	Italy	Europe	60461828.0			
247	271960.5	35497.0	NaN	Italy	Europe	60461828.0			
		life_expectancy	hospital_beds_per_thousand	gdp_per_capita					
0		83.51		3.18			35220.084		
1		83.51		3.18			35220.084		
2		83.51		3.18			35220.084		
3		83.51		3.18			35220.084		
4		83.51		3.18			35220.084		
..

```

243      83.51          3.18    35220.084
244      83.51          3.18    35220.084
245      83.51          3.18    35220.084
246      83.51          3.18    35220.084
247      83.51          3.18    35220.084

```

[248 rows x 17 columns]

The location data for Italy is appended to each row within covid_df. If the covid_df data frame contained data for multiple locations, then the respective country's location data would be appended for each row.

We can now calculate metrics like cases per million, deaths per million, and tests per million.

```
[119]: merged_df['cases_per_million'] = merged_df.total_cases * 1e6 / merged_df.
       →population
```

```
[120]: merged_df['deaths_per_million'] = merged_df.total_deaths * 1e6 / merged_df.
       →population
```

```
[121]: merged_df['tests_per_million'] = merged_df.total_tests * 1e6 / merged_df.
       →population
```

```
[122]: merged_df
```

	date	new_cases	new_deaths	new_tests	year	month	day	weekday	\
0	2019-12-31	0.0	0.0	NaN	2019	12	31	1	
1	2020-01-01	0.0	0.0	NaN	2020	1	1	2	
2	2020-01-02	0.0	0.0	NaN	2020	1	2	3	
3	2020-01-03	0.0	0.0	NaN	2020	1	3	4	
4	2020-01-04	0.0	0.0	NaN	2020	1	4	5	
..
243	2020-08-30	1444.0	1.0	53541.0	2020	8	30	6	
244	2020-08-31	1365.0	4.0	42583.0	2020	8	31	0	
245	2020-09-01	996.0	6.0	54395.0	2020	9	1	1	
246	2020-09-02	975.0	8.0	NaN	2020	9	2	2	
247	2020-09-03	1326.0	6.0	NaN	2020	9	3	3	
									\
	total_cases	total_deaths	total_tests	location	continent	population			\
0	0.0	0.0	NaN	Italy	Europe	60461828.0			
1	0.0	0.0	NaN	Italy	Europe	60461828.0			
2	0.0	0.0	NaN	Italy	Europe	60461828.0			
3	0.0	0.0	NaN	Italy	Europe	60461828.0			
4	0.0	0.0	NaN	Italy	Europe	60461828.0			
..
243	267298.5	35473.0	5117788.0	Italy	Europe	60461828.0			
244	268663.5	35477.0	5160371.0	Italy	Europe	60461828.0			
245	269659.5	35483.0	5214766.0	Italy	Europe	60461828.0			

```

246    270634.5      35491.0        NaN     Italy    Europe  60461828.0
247    271960.5      35497.0        NaN     Italy    Europe  60461828.0

    life_expectancy  hospital_beds_per_thousand  gdp_per_capita  \
0                  83.51                      3.18      35220.084
1                  83.51                      3.18      35220.084
2                  83.51                      3.18      35220.084
3                  83.51                      3.18      35220.084
4                  83.51                      3.18      35220.084
..
243                 ...
244                 ...
245                 ...
246                 ...
247                 ...

    cases_per_million  deaths_per_million  tests_per_million
0            0.000000          0.000000        NaN
1            0.000000          0.000000        NaN
2            0.000000          0.000000        NaN
3            0.000000          0.000000        NaN
4            0.000000          0.000000        NaN
..
243           4420.946386        586.700753      84644.943252
244           4443.522614        586.766910      85349.238862
245           4459.995818        586.866146      86248.897403
246           4476.121695        586.998461        NaN
247           4498.052887        587.097697        NaN

[248 rows x 20 columns]

```

Let's save and commit our work before continuing.

```
[123]: import jovian
```

```
[124]: jovian.commit()
```

```
<IPython.core.display.Javascript object>

[jovian] Updating notebook "shubhammeena712/python-pandas-data-analysis" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/shubhammeena712/python-
pandas-data-analysis
```

```
[124]: 'https://jovian.ai/shubhammeena712/python-pandas-data-analysis'
```

1.8 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.

```
[125]: 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']]
```

```
[126]: result_df
```

```
[126]:      date  new_cases  total_cases  new_deaths  total_deaths  new_tests  \
0    2019-12-31        0.0        0.0        0.0        0.0        NaN
1    2020-01-01        0.0        0.0        0.0        0.0        NaN
2    2020-01-02        0.0        0.0        0.0        0.0        NaN
3    2020-01-03        0.0        0.0        0.0        0.0        NaN
4    2020-01-04        0.0        0.0        0.0        0.0        NaN
..     ...
243   2020-08-30    1444.0    267298.5       1.0      35473.0    53541.0
244   2020-08-31    1365.0    268663.5       4.0      35477.0    42583.0
245   2020-09-01    996.0    269659.5       6.0      35483.0    54395.0
246   2020-09-02    975.0    270634.5       8.0      35491.0        NaN
247   2020-09-03   1326.0    271960.5       6.0      35497.0        NaN

      total_tests  cases_per_million  deaths_per_million  tests_per_million
0            NaN           0.000000           0.000000           NaN
1            NaN           0.000000           0.000000           NaN
2            NaN           0.000000           0.000000           NaN
3            NaN           0.000000           0.000000           NaN
4            NaN           0.000000           0.000000           NaN
..             ...
243    5117788.0      4420.946386      586.700753    84644.943252
244    5160371.0      4443.522614      586.766910    85349.238862
245    5214766.0      4459.995818      586.866146    86248.897403
246            NaN      4476.121695      586.998461        NaN
247            NaN      4498.052887      587.097697        NaN

[248 rows x 10 columns]
```

To write the data from the data frame into a file, we can use the `to_csv` function.

```
[127]: result_df.to_csv('results.csv', index=None)
```

The `to_csv` function also includes an additional column for storing the index of the dataframe by default. We pass `index=None` to turn off this behavior. You can now verify that the `results.csv` is created and contains data from the data frame in CSV format:

```
date,new_cases,total_cases,new_deaths,total_deaths,new_tests,total_tests,cases_per_million,death
2020-02-27,78.0,400.0,1.0,12.0,,,6.61574439992122,0.1984723319976366,
2020-02-28,250.0,650.0,5.0,17.0,,,10.750584649871982,0.28116913699665186,
2020-02-29,238.0,888.0,4.0,21.0,,,14.686952567825108,0.34732658099586405,
2020-03-01,240.0,1128.0,8.0,29.0,,,18.656399207777838,0.47964146899428844,
2020-03-02,561.0,1689.0,6.0,35.0,,,27.93498072866735,0.5788776349931067,
2020-03-03,347.0,2036.0,17.0,52.0,,,33.67413899559901,0.8600467719897585,
...
...
```

You can attach the `results.csv` file to our notebook while uploading it to [Jovian](#) using the `outputs` argument to `jovian.commit`.

```
[128]: import jovian
```

```
[129]: jovian.commit(outputs=['results.csv'])
```

```
<IPython.core.display.Javascript object>
[jovian] Updating notebook "shubhammeena712/python-pandas-data-analysis" on
https://jovian.ai
[jovian] Uploading additional outputs...
[jovian] Committed successfully! https://jovian.ai/shubhammeena712/python-
pandas-data-analysis
```

```
[129]: 'https://jovian.ai/shubhammeena712/python-pandas-data-analysis'
```

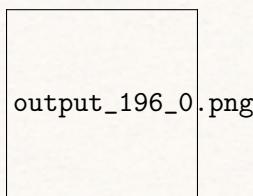
You can find the CSV file in the “Files” tab on the project page.

1.9 Bonus: 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.

```
[130]: result_df.new_cases.plot();
```



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.

```
[131]: result_df.set_index('date', inplace=True)
```

```
[132]: result_df
```

```
[132]:
```

date	new_cases	total_cases	new_deaths	total_deaths	new_tests	\
2019-12-31	0.0	0.0	0.0	0.0	NaN	
2020-01-01	0.0	0.0	0.0	0.0	NaN	
2020-01-02	0.0	0.0	0.0	0.0	NaN	
2020-01-03	0.0	0.0	0.0	0.0	NaN	
2020-01-04	0.0	0.0	0.0	0.0	NaN	
...	
2020-08-30	1444.0	267298.5	1.0	35473.0	53541.0	
2020-08-31	1365.0	268663.5	4.0	35477.0	42583.0	
2020-09-01	996.0	269659.5	6.0	35483.0	54395.0	
2020-09-02	975.0	270634.5	8.0	35491.0	NaN	
2020-09-03	1326.0	271960.5	6.0	35497.0	NaN	
date	total_tests	cases_per_million	deaths_per_million	\		
2019-12-31	NaN	0.000000	0.000000			
2020-01-01	NaN	0.000000	0.000000			
2020-01-02	NaN	0.000000	0.000000			
2020-01-03	NaN	0.000000	0.000000			
2020-01-04	NaN	0.000000	0.000000			
...			
2020-08-30	5117788.0	4420.946386	586.700753			
2020-08-31	5160371.0	4443.522614	586.766910			
2020-09-01	5214766.0	4459.995818	586.866146			
2020-09-02	NaN	4476.121695	586.998461			
2020-09-03	NaN	4498.052887	587.097697			
date	tests_per_million					
2019-12-31	NaN					
2020-01-01	NaN					
2020-01-02	NaN					
2020-01-03	NaN					
2020-01-04	NaN					
...	...					
2020-08-30	84644.943252					
2020-08-31	85349.238862					
2020-09-01	86248.897403					

```
2020-09-02          NaN  
2020-09-03          NaN
```

[248 rows x 9 columns]

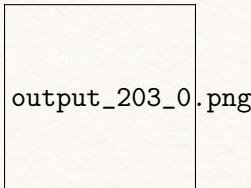
Notice that the index of a data frame doesn't have to be numeric. Using the date as the index also allows us to get the data for a specific date using .loc.

```
[133]: result_df.loc['2020-09-01']
```

```
[133]: new_cases          9.960000e+02  
total_cases          2.696595e+05  
new_deaths           6.000000e+00  
total_deaths         3.548300e+04  
new_tests            5.439500e+04  
total_tests          5.214766e+06  
cases_per_million   4.459996e+03  
deaths_per_million  5.868661e+02  
tests_per_million   8.624890e+04  
Name: 2020-09-01 00:00:00, dtype: float64
```

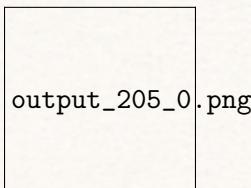
Let's plot the new cases & new deaths per day as line graphs.

```
[134]: result_df.new_cases.plot()  
result_df.new_deaths.plot();
```



We can also compare the total cases vs. total deaths.

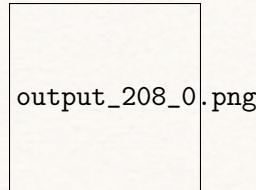
```
[135]: result_df.total_cases.plot()  
result_df.total_deaths.plot();
```



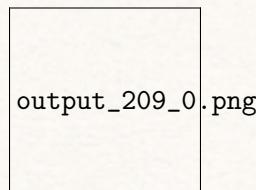
Let's see how the death rate and positive testing rates vary over time.

```
[136]: death_rate = result_df.total_deaths / result_df.total_cases
```

```
[137]: death_rate.plot(title='Death Rate');
```

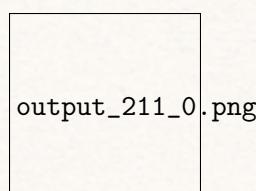


```
[138]: positive_rates = result_df.total_cases / result_df.total_tests  
positive_rates.plot(title='Positive Rate');
```



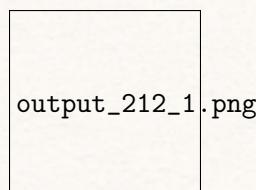
Finally, let's plot some month-wise data using a bar chart to visualize the trend at a higher level.

```
[139]: covid_month_df.new_cases.plot(kind='bar');
```



```
[140]: covid_month_df.new_tests.plot(kind='bar')
```

```
[140]: <AxesSubplot:xlabel='month'>
```



Let's save and commit our work to Jovian.

```
[141]: import jovian
```

```
[142]: jovian.commit()
```

```
<IPython.core.display.Javascript object>  
[jovian] Updating notebook "shubhammeena712/python-pandas-data-analysis" on  
https://jovian.ai  
[jovian] Committed successfully! https://jovian.ai/shubhammeena712/python-  
pandas-data-analysis
```

```
[142]: 'https://jovian.ai/shubhammeena712/python-pandas-data-analysis'
```

1.10 Exercises

Try the following exercises to become familiar with Pandas dataframe and practice your skills:

- Assignment on Pandas dataframes: <https://jovian.ml/aakashns/pandas-practice-assignment>
- Additional exercises on Pandas: https://github.com/guipsamora/pandas_exercises
- Try downloading and analyzing some data from Kaggle: <https://www.kaggle.com/datasets>

1.11 Summary and Further Reading

We've covered the following topics in this tutorial:

- 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

Check out the following resources to learn more about Pandas:

- User guide for Pandas: https://pandas.pydata.org/docs/user_guide/index.html
- Python for Data Analysis (book by Wes McKinney - creator of Pandas): <https://www.oreilly.com/library/view/python-for-data/9781491957653/>

You are ready to move on to the next tutorial: [Data Visualization using Matplotlib & Seaborn](#).

1.12 Questions for Revision

Try answering the following questions to test your understanding of the topics covered in this notebook:

1. What is Pandas? What makes it useful?
2. How do you install the Pandas library?
3. How do you import the pandas module?
4. What is the common alias used while importing the pandas module?
5. How do you read a CSV file using Pandas? Give an example?
6. What are some other file formats you can read using Pandas? Illustrate with examples.
7. What are Pandas dataframes?
8. How are Pandas dataframes different from Numpy arrays?
9. How do you find the number of rows and columns in a dataframe?
10. How do you get the list of columns in a dataframe?
11. What is the purpose of the describe method of a dataframe?
12. How are the info and describe dataframe methods different?
13. Is a Pandas dataframe conceptually similar to a list of dictionaries or a dictionary of lists?
Explain with an example.
14. What is a Pandas Series? How is it different from a Numpy array?
15. How do you access a column from a dataframe?
16. How do you access a row from a dataframe?
17. How do you access an element at a specific row & column of a dataframe?
18. How do you create a subset of a dataframe with a specific set of columns?
19. How do you create a subset of a dataframe with a specific range of rows?
20. Does changing a value within a dataframe affect other dataframes created using a subset of the rows or columns? Why is it so?
21. How do you create a copy of a dataframe?
22. Why should you avoid creating too many copies of a dataframe?
23. How do you view the first few rows of a dataframe?
24. How do you view the last few rows of a dataframe?
25. How do you view a random selection of rows of a dataframe?
26. What is the “index” in a dataframe? How is it useful?
27. What does a NaN value in a Pandas dataframe represent?
28. How is Nan different from 0?
29. How do you identify the first non-empty row in a Pandas series or column?
30. What is the difference between df.loc and df.at?
31. Where can you find a full list of methods supported by Pandas DataFrame and Series objects?
32. How do you find the sum of numbers in a column of dataframe?
33. How do you find the mean of numbers in a column of a dataframe?
34. How do you find the number of non-empty numbers in a column of a dataframe?
35. What is the result obtained by using a Pandas column in a boolean expression? Illustrate with an example.
36. How do you select a subset of rows where a specific column’s value meets a given condition?
Illustrate with an example.
37. What is the result of the expression df[df.new_cases > 100] ?
38. How do you display all the rows of a pandas dataframe in a Jupyter cell output?
39. What is the result obtained when you perform an arithmetic operation between two columns of a dataframe? Illustrate with an example.
40. How do you add a new column to a dataframe by combining values from two existing columns? Illustrate with an example.
41. How do you remove a column from a dataframe? Illustrate with an example.

42. What is the purpose of the `inplace` argument in dataframe methods?
43. How do you sort the rows of a dataframe based on the values in a particular column?
44. How do you sort a pandas dataframe using values from multiple columns?
45. How do you specify whether to sort by ascending or descending order while sorting a Pandas dataframe?
46. How do you change a specific value within a dataframe?
47. How do you convert a dataframe column to the `datetime` data type?
48. What are the benefits of using the `datetime` data type instead of `object`?
49. How do you extract different parts of a date column like the month, year, month, weekday, etc., into separate columns? Illustrate with an example.
50. How do you aggregate multiple columns of a dataframe together?
51. What is the purpose of the `groupby` method of a dataframe? Illustrate with an example.
52. What are the different ways in which you can aggregate the groups created by `groupby`?
53. What do you mean by a running or cumulative sum?
54. How do you create a new column containing the running or cumulative sum of another column?
55. What are other cumulative measures supported by Pandas dataframes?
56. What does it mean to merge two dataframes? Give an example.
57. How do you specify the columns that should be used for merging two dataframes?
58. How do you write data from a Pandas dataframe into a CSV file? Give an example.
59. What are some other file formats you can write to from a Pandas dataframe? Illustrate with examples.
60. How do you create a line plot showing the values within a column of dataframe?
61. How do you convert a column of a dataframe into its index?
62. Can the index of a dataframe be non-numeric?
63. What are the benefits of using a non-numeric dataframe? Illustrate with an example.
64. How you create a bar plot showing the values within a column of a dataframe?
65. What are some other types of plots supported by Pandas dataframes and series?

[]:

- ✓ 1. What is Pandas? What makes it useful?
- ✓ 2. How do you install the Pandas library?
- ✓ 3. How do you import the pandas module?
- ✓ 4. What is the common alias used while importing the pandas module?
- ✓ 5. How do you read a CSV file using Pandas? Give an example?
- ✓ 6. What are some other file formats you can read using Pandas? Illustrate with examples.
- ✓ 7. What are Pandas dataframes?
- ✓ 8. How are Pandas dataframes different from Numpy arrays?
- ✓ 9. How do you find the number of rows and columns in a dataframe?
- ✓ 10. How do you get the list of columns in a dataframe?

1. Pandas is a python library. It provides functions to easily deal with tabular data (csv, excel, spreadsheets, JSON, SQL, tables)

2. ! pip install pandas

3. import pandas as pd

4. pd

5. dataframe = pd.read_csv(file name)

6. excel, spreadsheets, JSON

7. Pandas Dataframe is one of the core data structures in pandas used for A 2D array storing & working with tabular data.

B. Numpy arrays can be more than 2D, arrays have similar type of objects
Pandas DF can only be 2D, it can have objects of multiple or similar data type.

9. .shape method

10. • columns

11. What is the purpose of the describe method of a dataframe?
12. How are the info and describe dataframe methods different?
13. Is a Pandas Dataframe conceptually similar to a list of dictionaries or a dictionary of lists?
Explain with an example.
14. What is a Pandas Series? How is it different from a Numpy array?
15. How do you access a column from a dataframe?
16. How do you access a row from a dataframe?
17. How do you access an element at a specific row & column of a dataframe?
18. How do you create a subset of a dataframe with a specific set of columns?
19. How do you create a subset of a dataframe with a specific range of rows?
20. Does changing a value within a dataframe affect other dataframes created using a subset of the rows or columns? Why is it so?

11. • describe gives statistical info for numerical columns (mean, std dev, min, max, no. of non-empty values).

12. Info method allows us to learn the shape of object types of our data.

13. dictionary of lists

Key → Column Header
Value → Values in columns

14. series is a column in a table. It is 1D array holding data of any type

In Numpy Array an implicitly defined integer index is used to access the values,
the Pandas has an explicitly defined index associated with the values.

15. do index notation

df ['column-name']

OR df . column - name

16. use .loc

double Index df [Column] [Row]

18. using indexing notation

cases = covid_df [['date' , 'new - cases']]

19. population_500 = housing [housing ['population'] > 500]
population_500

20. Yes, b/w they are sharing the data b/w them

21. How do you create a copy of a dataframe?

22. Why should you avoid creating too many copies of a dataframe?

23. How do you view the first few rows of a dataframe?

24. How do you view the last few rows of a dataframe?

25. How do you view a random selection of rows of a dataframe?

26. What is the "index" in a dataframe? How is it useful?

27. What does a NaN value in a Pandas dataframe represent?

28. How is Nan different from 0?

29. How do you identify the first non-empty row in a Pandas series or column?

30. What is the difference between df . loc and df . at?

Q1. • copy ()

Q2. It creates new dataframes

Q3. • head (n)

24

◦ tail(n)

25

◦ Sample

26.

Index is the address of row & column,

27.

Not a Number, no entry

28.

only parameter is
↓

30.

iloc is for rows

◦ at is [↓ ↓]
r + c (2 parameters)

29.

use ◦ first_valid_index()

31. Where can you find a full list of methods supported by Pandas DataFrame and Series objects?

32. How do you find the sum of numbers in a column of dataframe?

33. How do you find the mean of numbers in a column of a dataframe?

34. How do you find the number of non-empty numbers in a column of a dataframe?

35. What is the result obtained by using a Pandas column in a boolean expression? Illustrate with an example.

36. How do you select a subset of rows where a specific column's value meets a given condition? Illustrate with an example.

37. What is the result of the expression df[df.new_cases > 100] ?

is a
new
at true

38. How do you display all the rows of a pandas dataframe in a Jupyter cell output?

39.

What is the result obtained when you perform an arithmetic operation between two columns of a dataframe? Illustrate with an example.

40. How do you add a new column to a dataframe by combining values from two existing columns? Illustrate with an example.

41. How do you remove a column from a dataframe? Illustrate with an example.

31. Python.org/doc

32. df[column].sum

33. " . mean

34. Use a boolean expression

non-empty-value = covid_df.new_deaths != None

35. A new df

36. Boolean expression df.column-name ≥ N

37. A

38. syntax from stack overflow

```
from IPython.display import display
```

```
with pd.option_context('display.max_rows', 100)
```

```
display(covid_df[covid_df['new_cases'] > 1000])
```

39. we get a series showing result of the operation

40. so create a new

existing df ['new_name_column'] = operation of 2 columns

41. drop

- 42. What is the purpose of the inplace argument in dataframe methods?
- 43. How do you sort the rows of a dataframe based on the values in a particular column?
- 44. How do you sort a pandas dataframe using values from multiple columns?
- 45. How do you specify whether to sort by ascending or descending order while sorting a Pandas dataframe?
- 46. How do you change a specific value within a dataframe?
- 47. How do you convert a dataframe column to the datetime data type?
- 48. What are the benefits of using the datetime data type instead of object?
- 49. How do you extract different parts of a date column like the month, year, month, weekday, etc., into separate columns? Illustrate with an example.
- 50. How do you aggregate multiple columns of a dataframe together?

42. `inplace = True` updates the df and returns nothing

= `False` (by default) performs the operation and returns a copy of the object.

43. `df.sort_values('columns', ascending/Descending)`

44. uses a list of columns and orders corresponding to them

C1	Asans
C2	Des

45. give in the parameters, by default is `asc=T`

46. `df.use_at[Row, column] = new value`

47. `covid_df['date'] = pd.to_datetime(covid_df.date)`

48. You can use many methods using • Date time Index

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

• day

50. `new_df_name = existing_df[[column names]]`

51. What is the purpose of the groupby method of a dataframe? Illustrate with an example.
52. What are the different ways in which you can aggregate the groups created by groupby?
53. What do you mean by a running or cumulative sum?
54. How do you create a new column containing the running or cumulative sum of another column?
55. What are other cumulative measures supported by Pandas dataframes?
56. What does it mean to merge two dataframes? Give an example.
57. How do you specify the columns that should be used for merging two dataframes?
58. How do you write data from a Pandas dataframe into a CSV file? Give an example.
59. What are some other file formats you can write to from a Pandas dataframe? Illustrate with examples.
60. How do you create a line plot showing the values within a column of dataframe?

51. groupby helps to group and create a group of desired thing , select the columns we wish to aggregate ,

52. aggregate functions count()
Max , Min , SUM , Avg , CUMSUM

53. Sum upto the present entry

54. Apart from grouping, another form of aggregation is the running or cumulative sum of cases, tests, or death up to each row's date. We can use the cumsum method to compute the cumulative sum of a column as a new series. Let's add three new columns: total_cases, total_deaths, and total_tests.

```
] covid_df['total_cases'] = covid_df.new_cases.cumsum()
] covid_df['total_deaths'] = covid_df.new_deaths.cumsum()
] covid_df['total_tests'] = covid_df.new_tests.cumsum() + initial_tests
```

55. . Series.sum

- cummax
- cum min
- cum sum
- cum prod

56. Merging 2 df adding their columns , they need to have atleast 1 column in common

57. merged-df = df-1.merge(df-2, on = "column1", "column2")

58. result-df.to_csv('results.csv', index = None)

59. JSON
HTML
Spreadsheet

60. .plot()

61. How do you convert a column of a dataframe into its index?
62. Can the index of a dataframe be non-numeric?
63. What are the benefits of using a non-numeric dataframe? Illustrate with an example.
64. How you create a bar plot showing the values within a column of a dataframe?
65. What are some other types of plots supported by Pandas dataframes and series?

61. `df.set_index('column name', inplace = True)`

62. Yes

63. We can use other things like date as index for a specific data

64. `df.plot(kind = 'bar')`

65 Bar, Box, Hist, Area,
Scatter, pie, hexbin, kde