Analyzing Tabular Data using Python and Pandas

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](https://jovian.ai/aakashns/first-steps-with-python),notebook
2. [A Quick Tour of Variables and Data Types](https://jovian.ai/aakashns/python-variables-and-data-types)
3. [Branching using Conditional Statements and Loops](https://jovian.ai/aakashns/python-branching-and-loops)
4. [Writing Reusable Code Using Functions](https://jovian.ai/aakashns/python-functions-and-scope)
5. [Reading from and Writing to Files](https://jovian.ai/aakashns/python-os-and-filesystem)
6. [Numerical Computing with Python and Numpy](https://jovian.ai/aakashns/python-numerical-computing-with-numpy)
7. [Analyzing Tabular Data using Pandas](https://jovian.ai/aakashns/python-pandas-data-analysis)
8. [Data Visualization using Matplotlib & Seaborn](https://jovian.ai/aakashns/python-matplotlib-data-visualization)
9. [Exploratory Data Analysis - A Case Study](https://jovian.ai/aakashns/python-eda-stackoverflow-survey)

This tutorial covers the following topics:

* Reading a CSV file into a Pandas data frame
* Retrieving data from Pandas data frames
* Querying, soring, 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

How to run the code

This tutorial is an executable [Jupyter notebook](https://jovian.ai/outlink?url=https%3A%2F%2Fjupyter.org" \t "_blank) hosted on [Jovian](https://www.jovian.ai/). 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](https://jovian.ai/outlink?url=https%3A%2F%2Fcolab.research.google.com) or [Kaggle](https://jovian.ai/outlink?url=https%3A%2F%2Fkaggle.com" \t "_blank) 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](https://jovian.ai/outlink?url=https%3A%2F%2Fwww.python.org), download the notebook and install the required libraries. We recommend using the [Conda](https://jovian.ai/outlink?url=https%3A%2F%2Fdocs.conda.io%2Fprojects%2Fconda%2Fen%2Flatest%2Fuser-guide%2Finstall%2F" \t "_blank) 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](https://jovian.ai/outlink?url=https%3A%2F%2Fjupyter.org" \t "_blank) - 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.

Reading a CSV file using Pandas

[Pandas](https://jovian.ai/outlink?url=https%3A%2F%2Fpandas.pydata.org%2F) 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.

In [2]:

from urllib.request import urlretrieve

In [3]:

italy\_covid\_url = 'https://gist.githubusercontent.com/aakashns/f6a004fa20c84fec53262f9a8bfee775/raw/f309558b1cf5103424cef58e2ecb8704dcd4d74c/italy-covid-daywise.csv'

urlretrieve(italy\_covid\_url, 'italy-covid-daywise.csv')

Out[3]:

('italy-covid-daywise.csv', <http.client.HTTPMessage at 0x7f87ec447b80>)

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

In [4]:

!pip install pandas --upgrade --quiet

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

In [5]:

import pandas as pd

In [6]:

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.

In [7]:

type(covid\_df)

Out[7]:

pandas.core.frame.DataFrame

In [8]:

covid\_df

Out[8]:

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.

In [9]:

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.

In [10]:

covid\_df.describe()

Out[10]:

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

In [11]:

covid\_df.columns

Out[11]:

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 method.

In [12]:

covid\_df.shape

Out[12]:

(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 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

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](https://www.jovian.ai/) offers an easy way of saving and sharing your Jupyter notebooks online.

In [13]:

*# Install the library*

!pip install jovian --upgrade --quiet

In [14]:

import jovian

In [14]:

jovian.commit(project='python-pandas-data-analysis')

[jovian] Attempting to save notebook.. [jovian] Updating notebook "aakashns/python-pandas-data-analysis" on <https://jovian.ai/> [jovian] Uploading notebook.. [jovian] Capturing environment.. [jovian] Committed successfully! <https://jovian.ai/aakashns/python-pandas-data-analysis>

Out[14]:

'<https://jovian.ai/aakashns/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](https://jovian.ai/) 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.

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.

In [15]:

*# Pandas format is simliar 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).

In [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.

In [17]:

covid\_data\_dict['new\_cases']

Out[17]:

[1444, 1365, 996, 975, 1326]

In [18]:

covid\_df['new\_cases']

Out[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.

In [19]:

type(covid\_df['new\_cases'])

Out[19]:

pandas.core.series.Series

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

In [20]:

covid\_df['new cases'][246]

Out[20]:

975.0

In [21]:

covid\_df['new\_tests'][240]

Out[21]:

57640.0

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

In [22]:

covid\_df.at[246, 'new\_cases']

Out[22]:

975.0

In [23]:

covid\_df.at[240, 'new\_tests']

Out[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.

In [24]:

covid\_df.new\_cases

Out[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.

In [25]:

cases\_df = covid\_df[['date', 'new\_cases']]

cases\_df

Out[25]:

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.

In [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.

In [27]:

covid\_df

Out[27]:

In [28]:

covid\_df.loc[243]

Out[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.

In [29]:

type(covid\_df.loc[243])

Out[29]:

pandas.core.series.Series

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

In [30]:

covid\_df.head(5)

Out[30]:

In [31]:

covid\_df.tail(4)

Out[31]:

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.

In [32]:

covid\_df.at[0, 'new\_tests']

Out[32]:

nan

In [33]:

type(covid\_df.at[0, 'new\_tests'])

Out[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. 93,5310 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.

In [34]:

covid\_df.new\_tests.first\_valid\_index()

Out[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.

In [35]:

covid\_df.loc[108:113]

Out[35]:

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

In [36]:

covid\_df.sample(10)

Out[36]:

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.

In [37]:

import jovian

In [38]:

jovian.commit()

[jovian] Attempting to save notebook.. [jovian] Updating notebook "aakashns/python-pandas-data-analysis" on [https://jovian.ai](https://jovian.ai/) [jovian] Uploading notebook.. [jovian] Uploading additional files... [jovian] Committed successfully! <https://jovian.ai/aakashns/python-pandas-data-analysis>

Out[38]:

'<https://jovian.ai/aakashns/python-pandas-data-analysis>'

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.

In [39]:

total\_cases = covid\_df.new\_cases.sum()

total\_deaths = covid\_df.new\_deaths.sum()

In [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)?**

In [41]:

death\_rate = covid\_df.new\_deaths.sum() / covid\_df.new\_cases.sum()

In [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.**

In [43]:

initial\_tests = 935310

total\_tests = initial\_tests + covid\_df.new\_tests.sum()

In [44]:

total\_tests

Out[44]:

5214766.0

**Q: What fraction of tests returned a positive result?**

In [45]:

positive\_rate = total\_cases / total\_tests

In [46]:

print('{:.2f}% of tests in Italy led to a positive diagnosis.'.format(positive\_rate\*100))

5.21% of tests in Italy led to a positive diagnosis.

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

In [ ]:

In [ ]:

Let's save and commit our work before continuing.

In [47]:

import jovian

In [48]:

jovian.commit()

[jovian] Attempting to save notebook.. [jovian] Updating notebook "aakashns/python-pandas-data-analysis" on [https://jovian.ai](https://jovian.ai/) [jovian] Uploading notebook.. [jovian] Uploading additional files... [jovian] Committed successfully! <https://jovian.ai/aakashns/python-pandas-data-analysis>

Out[48]:

'<https://jovian.ai/aakashns/python-pandas-data-analysis>'

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.

In [49]:

high\_new\_cases = covid\_df.new\_cases > 1000

In [50]:

high\_new\_cases

Out[50]:

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.

In [51]:

covid\_df[high\_new\_cases]

Out[51]:

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

In [52]:

high\_cases\_df = covid\_df[covid\_df.new\_cases > 1000]

In [53]:

high\_cases\_df

Out[53]:

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.

In [54]:

from IPython.display import display

with pd.option\_context('display.max\_rows', 100):

display(covid\_df[covid\_df.new\_cases > 1000])

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.

In [55]:

positive\_rate

Out[55]:

0.05206657403227681

In [56]:

high\_ratio\_df = covid\_df[covid\_df.new\_cases / covid\_df.new\_tests > positive\_rate]

In [57]:

high\_ratio\_df

Out[57]:

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

In [58]:

covid\_df.new\_cases / covid\_df.new\_tests

Out[58]:

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.

In [59]:

covid\_df['positive\_rate'] = covid\_df.new\_cases / covid\_df.new\_tests

In [60]:

covid\_df

Out[60]:

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.

In [61]:

covid\_df.drop(columns=['positive\_rate'], inplace=True)

Can you figure the purpose of the inplace argument?

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.

In [62]:

covid\_df.sort\_values('new\_cases', ascending=False).head(10)

Out[62]:

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.

In [63]:

covid\_df.sort\_values('new\_deaths', ascending=False).head(10)

Out[63]:

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.

In [64]:

covid\_df.sort\_values('new\_cases').head(10)

Out[64]:

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.

In [65]:

covid\_df.loc[169:175]

Out[65]:

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.

In [66]:

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.

In [67]:

import jovian

In [68]:

jovian.commit()

[jovian] Attempting to save notebook.. [jovian] Updating notebook "aakashns/python-pandas-data-analysis" on [https://jovian.ai](https://jovian.ai/) [jovian] Uploading notebook.. [jovian] Uploading additional files... [jovian] Committed successfully! <https://jovian.ai/aakashns/python-pandas-data-analysis>

Out[68]:

'<https://jovian.ai/aakashns/python-pandas-data-analysis>'

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.

In [69]:

covid\_df.date

Out[69]:

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.

In [70]:

covid\_df['date'] = pd.to\_datetime(covid\_df.date)

In [71]:

covid\_df['date']

Out[71]:

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](https://jovian.ai/outlink?url=https%3A%2F%2Fpandas.pydata.org%2Fpandas-docs%2Fversion%2F0.23.4%2Fgenerated%2Fpandas.DatetimeIndex.html)).

In [72]:

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

In [73]:

covid\_df

Out[73]:

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.

In [74]:

*# Query the rows for May*

covid\_df\_may = covid\_df[covid\_df.month == 5]

*# Extract the subset of columns to be aggregated*

covid\_df\_may\_metrics = covid\_df\_may[['new\_cases', 'new\_deaths', 'new\_tests']]

*# Get the column-wise sum*

covid\_may\_totals = covid\_df\_may\_metrics.sum()

In [75]:

covid\_may\_totals

Out[75]:

new\_cases 29073.0

new\_deaths 5658.0

new\_tests 1078720.0

dtype: float64

In [76]:

type(covid\_may\_totals)

Out[76]:

pandas.core.series.Series

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

In [77]:

covid\_df[covid\_df.month == 5][['new\_cases', 'new\_deaths', 'new\_tests']].sum()

Out[77]:

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.

In [78]:

*# Overall average*

covid\_df.new\_cases.mean()

Out[78]:

1096.6149193548388

In [79]:

*# Average for Sundays*

covid\_df[covid\_df.weekday == 6].new\_cases.mean()

Out[79]:

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.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Let's save and commit our work before continuing.

In [80]:

import jovian

In [81]:

jovian.commit()

[jovian] Attempting to save notebook.. [jovian] Updating notebook "aakashns/python-pandas-data-analysis" on [https://jovian.ai](https://jovian.ai/) [jovian] Uploading notebook.. [jovian] Uploading additional files... [jovian] Committed successfully! <https://jovian.ai/aakashns/python-pandas-data-analysis>

Out[81]:

'<https://jovian.ai/aakashns/python-pandas-data-analysis>'

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.

In [82]:

covid\_month\_df = covid\_df.groupby('month')[['new\_cases', 'new\_deaths', 'new\_tests']].sum()

In [83]:

covid\_month\_df

Out[83]:

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.

In [84]:

covid\_month\_mean\_df = covid\_df.groupby('month')[['new\_cases', 'new\_deaths', 'new\_tests']].mean()

In [85]:

covid\_month\_mean\_df

Out[85]:

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.

In [86]:

covid\_df['total\_cases'] = covid\_df.new\_cases.cumsum()

In [87]:

covid\_df['total\_deaths'] = covid\_df.new\_deaths.cumsum()

In [88]:

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.

In [89]:

covid\_df

Out[89]:

Notice how the NaN values in the total\_tests column remain unaffected.

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.

In [90]:

urlretrieve('https://gist.githubusercontent.com/aakashns/8684589ef4f266116cdce023377fc9c8/raw/99ce3826b2a9d1e6d0bde7e9e559fc8b6e9ac88b/locations.csv',

'locations.csv')

Out[90]:

('locations.csv', <http.client.HTTPMessage at 0x7f87c57a41f0>)

In [91]:

locations\_df = pd.read\_csv('locations.csv')

In [92]:

locations\_df

Out[92]:

In [93]:

locations\_df[locations\_df.location == "Italy"]

Out[93]:

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".

In [94]:

covid\_df['location'] = "Italy"

In [95]:

covid\_df

Out[95]:

We can now add the columns from locations\_df into covid\_df using the .merge method.

In [96]:

merged\_df = covid\_df.merge(locations\_df, on="location")

In [97]:

merged\_df

Out[97]:

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.

In [98]:

merged\_df['cases\_per\_million'] = merged\_df.total\_cases \* 1e6 / merged\_df.population

In [99]:

merged\_df['deaths\_per\_million'] = merged\_df.total\_deaths \* 1e6 / merged\_df.population

In [100]:

merged\_df['tests\_per\_million'] = merged\_df.total\_tests \* 1e6 / merged\_df.population

In [101]:

merged\_df

Out[101]:

Let's save and commit our work before continuing.

In [102]:

import jovian

In [103]:

jovian.commit()

[jovian] Attempting to save notebook.. [jovian] Updating notebook "aakashns/python-pandas-data-analysis" on [https://jovian.ai](https://jovian.ai/) [jovian] Uploading notebook.. [jovian] Uploading additional files... [jovian] Committed successfully! <https://jovian.ai/aakashns/python-pandas-data-analysis>

Out[103]:

'<https://jovian.ai/aakashns/python-pandas-data-analysis>'

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.

In [104]:

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']]

In [105]:

result\_df

Out[105]:

To write the data from the data frame into a file, we can use the to\_csv function.

In [106]:

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,deaths\_per\_million,tests\_per\_million

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](https://jovian.ai/) using the outputs argument to jovian.commit.

In [107]:

import jovian

In [ ]:

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

[jovian] Attempting to save notebook..

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

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.

In [109]:

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.

In [110]:

result\_df.set\_index('date', inplace=True)

In [111]:

result\_df

Out[111]:

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 data using .loc.

In [112]:

result\_df.loc['2020-09-01']

Out[112]:

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.

In [113]:

result\_df.new\_cases.plot()

result\_df.new\_deaths.plot();

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

In [114]:

result\_df.total\_cases.plot()

result\_df.total\_deaths.plot();

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

In [115]:

death\_rate = result\_df.total\_deaths / result\_df.total\_cases

In [116]:

death\_rate.plot(title='Death Rate');

In [117]:

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.

In [118]:

covid\_month\_df.new\_cases.plot(kind='bar');

In [119]:

covid\_month\_df.new\_tests.plot(kind='bar')

Out[119]:

<AxesSubplot:xlabel='month'>

Let's save and commit our work to Jovian.