Pandas Tutorial

**Data Input and Output (I/O)**

Pandas offers a variety of functions to read data from and write data to different file formats as given below:

* [Read CSV Files with Pandas](https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/)

**Pandas Read CSV in Python**

CSV files are the Comma Separated Files. It allows users to load tabular data into a **DataFrame**, which is a powerful structure for data manipulation and analysis. To access data from the CSV file, we require a function read\_csv() from Pandas that retrieves data in the form of the data frame. Here’s a quick example to get you started.

Suppose you have a file named [people.csv](https://media.geeksforgeeks.org/wp-content/uploads/20241121154629307916/people_data.csv). First, we must import the [Pandas](https://www.geeksforgeeks.org/python-pandas-dataframe/)library. then using Pandas load this data into a DataFrame as follows:

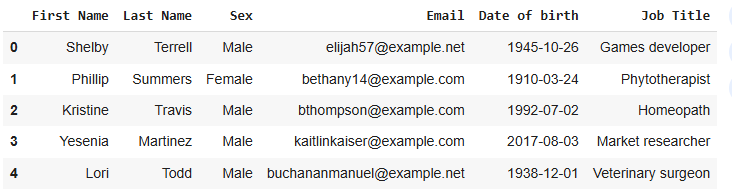
**import** **pandas** **as** **pd**

*# reading csv file*

df = pd.read\_csv("people.csv")

df

**Output:**

Pandas Read CSV in Python

**read\_csv() function - Syntax & Parameters**

**read\_csv() function** in Pandas is used to read data from CSV files into a Pandas **DataFrame**. A DataFrame is a powerful data structure that allows you to manipulate and analyze tabular data efficiently. CSV files are plain-text files where each row represents a record, and columns are separated by commas (or other delimiters).

Here is the **Pandas read CSV** syntax with its parameters.

***Syntax: pd.read\_csv****(filepath\_or\_buffer, sep=' ,' , header='infer',  index\_col=None, usecols=None, engine=None, skiprows=None, nrows=None)*

***Parameters:***

* ***filepath\_or\_buffer****: Location of the csv file. It accepts any string path or URL of the file.*
* ***sep****: It stands for separator, default is ', '.*
* ***header****: It accepts int, a list of int, row numbers to use as the column names, and the start of the data. If no names are passed, i.e., header=None, then, it will display the first column as 0, the second as 1, and so on.*
* ***usecols****: Retrieves only selected columns from the CSV file.*
* ***nrows****: Number of rows to be displayed from the dataset.*
* ***index\_col****: If None, there are no index numbers displayed along with records.*
* ***skiprows****: Skips passed rows in the new data frame.*

**Features in Pandas read\_csv**

**1. Read specific columns using read\_csv**

The **usecols parameter** allows to load only specific columns from a CSV file. This reduces memory usage and processing time by importing only the required data.

df = pd.read\_csv("people.csv", usecols=["First Name", "Email"])

print(df)

**Output:**

First Name Email

0 Shelby elijah57@example.net

1 Phillip bethany14@example.com

2 Kristine bthompson@example.com

3 Yesenia kaitlinkaiser@example.com

4 Lori buchananmanuel@example.net

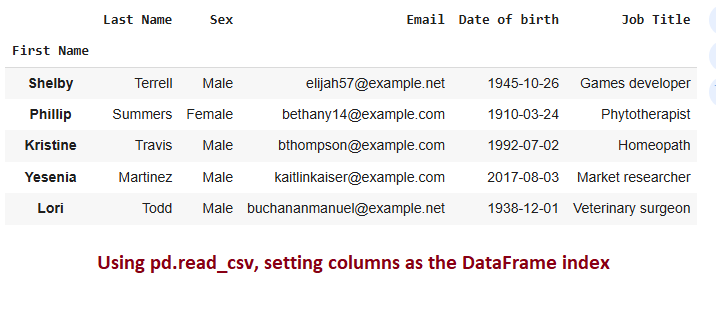
**2. Setting an Index Column (index\_col)**

The index\_col parameter **sets one or more columns as the DataFrame index**, making the specified column(s) act as row labels for easier data referencing.

df = pd.read\_csv("people.csv", index\_col="First Name")

print(df)

**Output:**

Read CSV in Python

**3. Handling Missing Values Using read\_csv**

The na\_values parameter replaces specified strings (e.g., "N/A", "Unknown") with NaN, enabling consistent handling of missing or incomplete data during analysis.\

df = pd.read\_csv("people.csv", na\_values=["N/A", "Unknown"])

We won't got nan values as there is no missing value in our dataset.

**4. Reading CSV Files with Different Delimiters**

In this example, we will take a CSV file and then add some special characters to see how the **sep**parameter works.

**import** **pandas** **as** **pd**

*# Sample data stored in a multi-line string*

data = """totalbill\_tip, sex:smoker, day\_time, size

16.99, 1.01:Female|No, Sun, Dinner, 2

10.34, 1.66, Male, No|Sun:Dinner, 3

21.01:3.5\_Male, No:Sun, Dinner, 3

23.68, 3.31, Male|No, Sun\_Dinner, 2

24.59:3.61, Female\_No, Sun, Dinner, 4

25.29, 4.71|Male, No:Sun, Dinner, 4"""

*# Save the data to a CSV file*

**with** open("sample.csv", "w") **as** file:

file.write(data)

print(data)

**Output:**

totalbill\_tip, sex:smoker, day\_time, size

16.99, 1.01:Female|No, Sun, Dinner, 2

10.34, 1.66, Male, No|Sun:Dinner, 3

21.01:3.5\_Male, No:Sun, Dinner, 3

23.68, 3.31, Male|No, Sun\_Dinner, 2

24.59:3.61, Female\_No, Sun, Dinner, 4

25.29, 4.71|Male, No:Sun, Dinner, 4

The sample data is stored in a multi-line string for demonstration purposes.

* **Separator (sep)**: The sep='[:, |\_]' argument allows Pandas to handle multiple delimiters (:, |, \_, ,) using a regular expression.
* **Engine**: The engine='python' argument is used because the default C engine does not support regular expressions for delimiters.

1

# Load the CSV file using pandas with multiple delimiters

2

df = pd.read\_csv('sample.csv',

3

sep='[:, |\_]', # Define the delimiters

4

engine='python') # Use Python engine for regex separators

5

df

**Output:**

totalbill tip Unnamed: 2 sex smoker Unnamed: 5 day time Unnamed: 8 size

16.99 NaN 1.01 Female No NaN Sun NaN Dinner NaN 2.0

10.34 NaN 1.66 NaN Male NaN No Sun Dinner NaN 3.0

21.01 3.50 Male NaN No Sun NaN Dinner NaN 3.0 NaN

23.68 NaN 3.31 NaN Male No NaN Sun Dinner NaN 2.0

24.59 3.61 NaN Female No NaN Sun NaN Dinner NaN 4.0

25.29 NaN 4.71 Male NaN No Sun NaN Dinner NaN 4.0

**5. Using nrows in read\_csv()**

The nrows parameter limits the number of rows read from a file, enabling quick previews or partial data loading for large datasets. Here, we just display only 5 rows using **nrows parameter**.

df = pd.read\_csv('people.csv', nrows=3)

df

**Output:**

First Name Last Name Sex Email Date of birth Job Title

0 Shelby Terrell Male elijah57@example.net 1945-10-26 Games developer

1 Phillip Summers Female bethany14@example.com 1910-03-24 Phytotherapist

2 Kristine Travis Male bthompson@example.com 1992-07-02 Homeopath

**6. Using skiprows in read\_csv()**

The skiprows parameter skips unnecessary rows at the start of a file, which is useful for ignoring metadata or extra headers that are not part of the dataset.

df= pd.read\_csv("people.csv")

print("Previous Dataset: ")

print(df)

*# using skiprows*

df = pd.read\_csv("people.csv", skiprows = [4,5])

print("Dataset After skipping rows: ")

print(df)

**Output:**

**Previous Dataset:**

First Name Last Name Sex Email Date of birth Job Title

0 Shelby Terrell Male elijah57@example.net 1945-10-26 Games developer

1 Phillip Summers Female bethany14@example.com 1910-03-24 Phytotherapist

2 Kristine Travis Male bthompson@example.com 1992-07-02 Homeopath

3 Yesenia Martinez Male kaitlinkaiser@example.com 2017-08-03 Market researcher

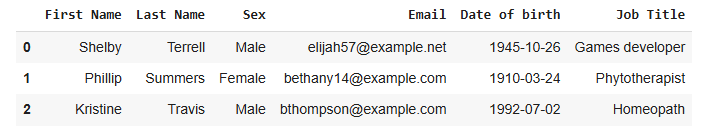
4 Lori Todd Male buchananmanuel@example.net 1938-12-01 Veterinary surgeon

5 Erin Day Male tconner@example.org 2015-10-28 Management officer

6 Katherine Buck Female conniecowan@example.com 1989-01-22 Analyst

7 Ricardo Hinton Male wyattbishop@example.com 1924-03-26 Hydrogeologist

**Dataset After skipping rows:**

Pandas Read CSV

**7. Parsing Dates (parse\_dates)**

The parse\_dates parameter converts date columns into datetime objects, simplifying operations like filtering, sorting, or time-based analysis.

df = pd.read\_csv("people.csv", parse\_dates=["Date of birth"])

print(df.info())

**Output:**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 5 entries, 0 to 4

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 First Name 5 non-null object

1 Last Name 5 non-null object

2 Sex 5 non-null object

3 Email 5 non-null object

**4 Date of birth 5 non-null datetime64[ns]**

5 Job Title 5 non-null object

dtypes: datetime64[ns](1), object(5)

memory usage: 368.0+ bytes

**Loading a CSV Data from a URL**

Pandas allows you to directly read a CSV file hosted on the internet using the file's URL. This can be incredibly useful when working with datasets shared on websites, cloud storage, or public repositories like GitHub.

1

url = "https://media.geeksforgeeks.org/wp-content/uploads/20241121154629307916/people\_data.csv"

2

df = pd.read\_csv(url)

3

df

**Output:**

First Name Last Name Sex Email Date of birth Job Title

0 Shelby Terrell Male elijah57@example.net 1945-10-26 Games developer

1 Phillip Summers Female bethany14@example.com 1910-03-24 Phytotherapist

2 Kristine Travis Male bthompson@example.com 1992-07-02 Homeopath

3 Yesenia Martinez Male kaitlinkaiser@example.com 2017-08-03 Market researcher

4 Lori Todd Male buchananmanuel@example.net 1938-12-01 Veterinary surgeon

* [Writing data to CSV Files](https://www.geeksforgeeks.org/saving-a-pandas-dataframe-as-a-csv/)

**Saving a Pandas Dataframe as a CSV**

In this article, we will learn how we can export a [Pandas DataFrame](https://www.geeksforgeeks.org/python-pandas-dataframe/) to a CSV file by using the [Pandas](https://www.geeksforgeeks.org/python-pandas-dataframe/) to\_csv() method. By default, the to csv() method exports DataFrame to a CSV file with row index as the first column and comma as the delimiter.

**Table of Content**

* [Export CSV to a Working Directory](https://www.geeksforgeeks.org/saving-a-pandas-dataframe-as-a-csv/#export-csv-to-a-working-directory)
* [Saving CSV Without Headers and Index](https://www.geeksforgeeks.org/saving-a-pandas-dataframe-as-a-csv/#saving-csv-without-headers-and-index)
* [Save the CSV file to a Specified Location](https://www.geeksforgeeks.org/saving-a-pandas-dataframe-as-a-csv/#save-the-csv-file-to-a-specified-location)
* [Write a DataFrame to CSV file using Tab Separator](https://www.geeksforgeeks.org/saving-a-pandas-dataframe-as-a-csv/#write-a-dataframe-to-csv-file-using-tab-separator)

Here, we are taking sample data to convert DataFrame to CSV.

*# importing pandas as pd*

**import** **pandas** **as** **pd**

*# list of name, degree, score*

nme = ["aparna", "pankaj", "sudhir", "Geeku"]

deg = ["MBA", "BCA", "M.Tech", "MBA"]

scr = [90, 40, 80, 98]

*# dictionary of lists*

dict = {'name': nme, 'degree': deg, 'score': scr}

df = pd.DataFrame(dict)

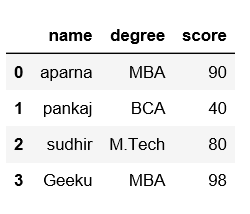
**Export CSV to a Working Directory**

Here, we simply export a Dataframe to a CSV file using df.to\_csv().

*# saving the dataframe*

df.to\_csv('file1.csv')

**Output:**



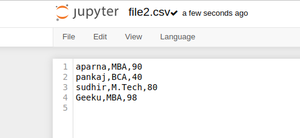
**Saving CSV Without Headers and Index**

Here, we are saving the file with no header and no index number.

*# saving the dataframe*

df.to\_csv('file2.csv', header=**False**, index=**False**)

**Output:**



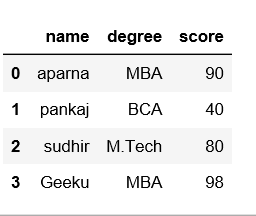
**Save the CSV file to a Specified Location**

We can also, save our file at some specific location.

*# saving the dataframe*

df.to\_csv(r'C:\Users\Admin\Desktop\file3.csv')

**Output:**



**Write a DataFrame to CSV file using Tab Separator**

We can also save our file with some specific separate as we want. i.e, "\t" .

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

users = {'Name': ['Amit', 'Cody', 'Drew'],

'Age': [20,21,25]}

*#create DataFrame*

df = pd.DataFrame(users, columns=['Name','Age'])

print("Original DataFrame:")

print(df)

print('Data from Users.csv:')

df.to\_csv('Users.csv', sep='**\t**', index=**False**,header=**True**)

new\_df = pd.read\_csv('Users.csv')

print(new\_df)

**Output:**

Original DataFrame:  
 Name Age  
0 Amit 20  
1 Cody 21  
2 Drew 25  
Data from Users.csv:  
 Name\tAge  
0 Amit\t20  
1 Cody\t21  
2 Drew\t25

* [Export Pandas dataframe to a CSV file](https://www.geeksforgeeks.org/export-pandas-dataframe-to-a-csv-file/)

When working on a Data Science project one of the key tasks is **data management** which includes data collection, cleaning and storage. Once our data is cleaned and processed it’s essential to save it in a structured format for further analysis or sharing.

A [**CSV (Comma-Separated Values) file**](https://www.geeksforgeeks.org/csv-file-format/) is a widely used format for storing tabular data. In Python **Pandas** provides an easy-to-use function **to\_csv()**to export a DataFrame into a CSV file. This article will walk we through the process with step-by-step examples and customizations.

**Creating a Sample DataFrame**

Before exporting let's first [create a sample DataFrame](https://www.geeksforgeeks.org/different-ways-to-create-pandas-dataframe/)using Pandas.

**import** **pandas** **as** **pd**

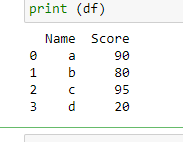
scores = {'Name': ['a', 'b', 'c', 'd'],

'Score': [90, 80, 95, 20]}

df = pd.DataFrame(scores)

print(df)

**Output :**



Now that we have a sample DataFrame, let's export it to a CSV file.

**Exporting DataFrame to CSV**

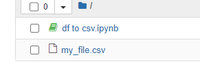
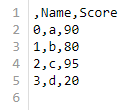
**1. Basic Export**

The simplest way to export a DataFrame to a CSV file is by using the [to\_csv()](https://www.geeksforgeeks.org/saving-a-pandas-dataframe-as-a-csv/" \t "_blank) function without any additional parameters. This method creates a CSV file where the DataFrame's contents are written as-is.

1

df.to\_csv("your\_name.csv")

**Output**

File Successfully saved

**Customizing the CSV Export**

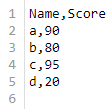
**2. Remove Index Column**

The **to\_csv()** exports the **index** column which represents the row numbers of the DataFrame. If we do not want this extra column in our CSV file we can remove it by setting index=False.

1

df.to\_csv('your\_name.csv', index = False)

**Output :** 



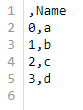
**3. Export only selected columns**

In some cases we may not want to export all columns from our DataFrame. The columns parameter in to\_csv() allows us to specify which columns should be included in the output file.

1

df.to\_csv("your\_name.csv", columns = ['Name'])

**Output :** 



**4. Exclude Header Row**

By default the**to\_csv()** function includes column names as the first row of the CSV file. However if we need a headerless file e.g., for certain machine learning models or integration with other systems we can set header=False.

1

df.to\_csv('your\_name.csv', header = False)

**Output :** 



**5. Handling Missing Values**

DataFrames often contain missing values (NaN) which can cause issues in downstream analysis. By default Pandas writes NaN as an empty field but we can customize this behavior using the na\_rep parameter.

1

df.to\_csv("your\_name.csv", na\_rep = 'nothing')

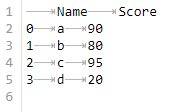
**6. Change Column Separator**

CSV files use **commas (,)** by default as delimiters to separate values. However in some cases other delimiters may be required such as **tabs ()**, semicolons (;), or pipes (|). Using a different delimiter can make the file more readable or compatible with specific systems.

1

df.to\_csv("your\_name.csv", sep ='\t')

**Output :** 



* [Read JSON Files with Pandas](https://www.geeksforgeeks.org/how-to-read-json-files-with-pandas/)

[JSON (JavaScript Object Notation)](https://www.geeksforgeeks.org/json/) store data using key-value pairs. Reading JSON files using Pandas is simple and helpful when you're working with data in .json format. There are mainly three methods to read Json file using Pandas Some of them are:

* Using pd.read\_json() Method
* Using JSON Module and pd.json\_normalize() Method
* Using pd.Dataframe() Methods

**1. Using pd.read\_json() to Read JSON Files in Pandas**

The [pd.read\_json() function](https://www.geeksforgeeks.org/how-to-read-json-files-with-pandas/" \t "_blank) helps to read JSON data directly into a DataFrame. This method is used when we working with standard JSON structures. If the file is located on a remote server we can also pass the URL instead of a local file path. Let’s say you have a file named data.json with the following content:

*[*

*{"id": 1, "name": "Alice", "age": 25},*

*{"id": 2, "name": "Bob", "age": 30},*

*{"id": 3, "name": "Charlie", "age": 22}*

*]*

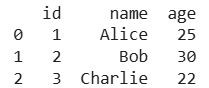
You can read this JSON file using the code below:

**import** **pandas** **as** **pd**

df = pd.read\_json('data.json')

print(df.head())

**Output:**



**2. Using json Module and pd.json\_normalize() method**

The[json\_normalize()](https://www.geeksforgeeks.org/python-pandas-flatten-nested-json/) is used when we are working with nested JSON structues. JSON from APIs often comes in nested form and this method helps to **flatten** it into a tabular format that’s easier to work with in Pandas. This method is helpful when working with real-world JSON responses from APIs.

import pandas as pd

import json

data = {"One": {"0": 60, "1": 60, "2": 60, "3": 45, "4": 45, "5": 60},

"Two": {"0": 110, "1": 117, "2": 103, "3": 109, "4": 117, "5": 102}}

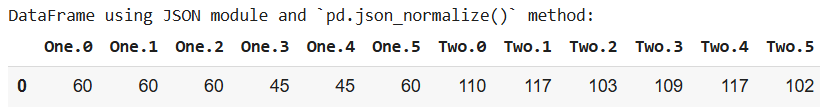
json\_data = json.dumps(data)

df\_normalize = pd.json\_normalize(json.loads(json\_data))

print("\nDataFrame using JSON module and `pd.json\_normalize()` method:")

df\_normalize

**Output:**



**3. Using pd.DataFrame with a Dictionary**

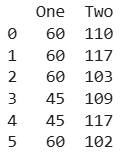
If JSON data is stored as a dictionary we can directly use [pd.DataFrame()](https://www.geeksforgeeks.org/python-pandas-dataframe/" \t "_blank)convert it into a structured DataFrame. This is helpful when you are working with pre-loaded or manually created JSON data in memory.

**import** **pandas** **as** **pd**

df = pd.DataFrame(data)

print(df)

**Output:**

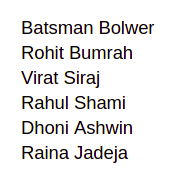


These methods help you to use JSON data into Pandas for analysis and visualization. With just a few lines of code you can turn raw JSON into a clean and usable DataFrame.

* [Parsing JSON Dataset](https://www.geeksforgeeks.org/pandas-parsing-json-dataset/)
* [Exporting Pandas DataFrame to JSON File](https://www.geeksforgeeks.org/exporting-pandas-dataframe-to-json-file/)
* [Working with Excel Files in Pandas](https://www.geeksforgeeks.org/working-with-excel-files-using-pandas/)
* [Read Text Files with Pandas](https://www.geeksforgeeks.org/how-to-read-text-files-with-pandas/)

In this article, we will discuss how to read text files with pandas in Python. In [Python](https://www.geeksforgeeks.org/python-programming-language/), the Pandas module allows us to load DataFrames from external files and work on them. The dataset can be in different types of files.

**Text File Used**



**Read Text Files with Pandas**

Below are the methods by which we can read text files with Pandas:

* Using read\_csv()
* Using read\_table()
* Using read\_fwf()

**Read Text Files with Pandas Using read\_csv()**

We will read the text file with pandas using the [read\_csv() function](https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/). Along with the text file, we also pass separator as a single space (‘ ’) for the space character because, for text files, the space character will separate each field. There are three parameters we can pass to the read\_csv() function.

**Syntax:**

***Syntax****: data=pandas.read\_csv('filename.txt', sep=' ', header=None, names=["Column1", "Column2"])*

***Parameters:***

* ***filename.txt:*** *As the name suggests it is the name of the text file from which we want to read data.*
* ***sep****: It is a separator field. In the text file, we use the space character(' ') as the separator.*
* ***header:*** *This is an optional field. By default, it will take the first line of the text file as a header. If we use header=None then it will create the header.*
* ***names:*** *We can assign column names while importing the text file by using the names argument.*

**Example 1**

In this example, we are using read\_csv() function to read the csv file.

*# Read Text Files with Pandas using read\_csv()*

*# importing pandas*

**import** **pandas** **as** **pd**

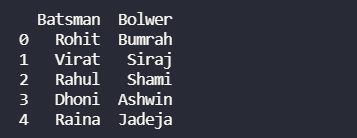
*# read text file into pandas DataFrame*

df = pd.read\_csv("gfg.txt", sep=" ")

*# display DataFrame*

print(df)

**Output:**



**Example 2**

In this example, we will make the header filed equal to None. This will create a default header in the output. And take the first line of the text file as data entry. The created header name will be a number starting from 0.

*# Read Text Files with Pandas using read\_csv()*

*# importing pandas*

**import** **pandas** **as** **pd**

*# read text file into pandas DataFrame and*

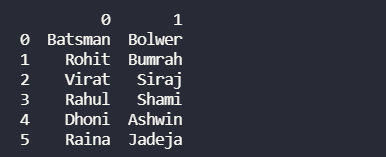
*# create header*

df = pd.read\_csv("gfg.txt", sep=" ", header=**None**)

*# display DataFrame*

print(df)

**Output:**



**Example 3:**

In the above output, we can see it creates a header starting from number 0. But we can also give names to the header. In this example, we will see how to create a header with a name using pandas.

*# Read Text Files with Pandas using read\_csv()*

*# importing pandas*

**import** **pandas** **as** **pd**

*# read text file into pandas DataFrame and create*

*# header with names*

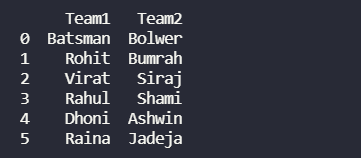
df = pd.read\_csv("gfg.txt", sep=" ", header=**None**,

names=["Team1", "Team2"])

*# display DataFrame*

print(df)

**Output:**



**Read Text Files with Pandas Using read\_table()**

We can read data from a text file using [read\_table()](https://www.geeksforgeeks.org/pandas-read_table-function/) in pandas. This function reads a general delimited file to a DataFrame object. This function is essentially the same as the read\_csv() function but with the delimiter = '\t', instead of a comma by default. We will read data with the read\_table function making separator equal to a single space(' ').

***Syntax:****data=pandas.read\_table('filename.txt', delimiter = ' ')*

***Parameters:***

* ***filename.txt:*** *As the name suggests it is the name of the text file from which we want to read data.*

**Example:** In this example, we are using read\_table() function to read the table.

*# Read Text Files with Pandas using read\_table()*

*# importing pandas*

**import** **pandas** **as** **pd**

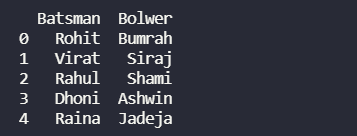
*# read text file into pandas DataFrame*

df = pd.read\_table("gfg.txt", delimiter=" ")

*# display DataFrame*

print(df)

**Output:**



**Read Text Files with Pandas Using read\_fwf()**

The fwf in the read\_fwf() function stands for fixed-width lines. We can use this function to load DataFrames from files. This function also supports text files. We will read data from the text files using the read\_fwf() function with pandas. It also supports optionally iterating or breaking the file into chunks. Since the columns in the text file were separated with a fixed width, this read\_fwf() read the contents effectively into separate columns.

***Syntax:****data=pandas.read\_fwf('filename.txt')*

***Parameters:***

* ***filename.txt:*** *As the name suggests it is the name of the text file from which we want to read data.*

**Example:** In this example, we are using read\_fwf to read the data.

*# Read Text Files with Pandas using read\_fwf()*

*# importing pandas*

**import** **pandas** **as** **pd**

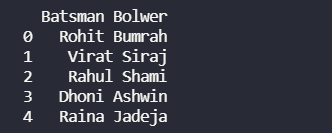
*# read text file into pandas DataFrame*

df = pd.read\_fwf("gfg.txt")

*# display DataFrame*

print(df)

**Output:**



* [Text File to CSV using Python Pandas](https://www.geeksforgeeks.org/convert-text-file-to-csv-using-python-pandas/)