import pandas as pd df = pd.read_csv("people_data.csv") df

read_csv synatx:

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.

In [8]:
```
df = pd.read_csv("people_data.csv", usecols=["First Name", "Email"])
print(df)
```

```
  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
```

In [10]:
```
df = pd.read_csv("people_data.csv", index_col="First Name")
print(df)
```

```
           Last Name     Sex                       Email Date of birth  \
First Name
Shelby       Terrell    Male        elijah57@example.net    1945-10-26
Phillip      Summers  Female       bethany14@example.com    1910-03-24
Kristine      Travis    Male       bthompson@example.com    1992-07-02
Yesenia     Martinez    Male   kaitlinkaiser@example.com    2017-08-03
Lori            Todd    Male  buchananmanuel@example.net    1938-12-01

                    Job Title
First Name
Shelby         Games developer
Phillip         Phytotherapist
Kristine              Homeopath
Yesenia      Market researcher
Lori        Veterinary surgeon
```

In [18]:
```
# 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_data.csv", na_values=["N/A", "Unknown"])
```

In [20]:
```
# Reading CSV Files with Different Delimiters
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)
```

```
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
```

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.

In [25]:
```
df = pd.read_csv('sample.csv',sep='[:, |_]',engine='python')
df
```

Out[25]:

	totalbill	tip	Unnamed: 2	sex	smoker	Unnamed: 5	day	time	Unnamed: 8	si
16.99	NaN	1.01	Female	No	NaN	Sun	NaN	Dinner	NaN	2
10.34	NaN	1.66	NaN	Male	NaN	No	Sun	Dinner	NaN	3
21.01	3.50	Male	NaN	No	Sun	NaN	Dinner	NaN	3.0	Na
23.68	NaN	3.31	NaN	Male	No	NaN	Sun	Dinner	NaN	2
24.59	3.61	NaN	Female	No	NaN	Sun	NaN	Dinner	NaN	4
25.29	NaN	4.71	Male	NaN	No	Sun	NaN	Dinner	NaN	4

In [27]:
```
df = pd.read_csv("people_data.csv",nrows=3)
df
```

Out[27]:

	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

In [29]:
```
# skiprows in readcsv
df = pd.read_csv("people_data.csv")
print("prev data")
print(df)
# skiprows
df = pd.read_csv("people_data.csv",skiprows=[4,5])
print("now data")
print(df)
```

```
prev data
  First Name Last Name     Sex                       Email Date of birth  \
0     Shelby   Terrell    Male        elijah57@example.net    1945-10-26
1    Phillip   Summers  Female       bethany14@example.com    1910-03-24
2   Kristine    Travis    Male       bthompson@example.com    1992-07-02
3    Yesenia  Martinez    Male   kaitlinkaiser@example.com    2017-08-03
4       Lori      Todd    Male  buchananmanuel@example.net    1938-12-01

            Job Title
0     Games developer
1      Phytotherapist
2           Homeopath
3   Market researcher
4  Veterinary surgeon
now data
  First Name Last Name     Sex                   Email Date of birth  \
0     Shelby   Terrell    Male   elijah57@example.net    1945-10-26
1    Phillip   Summers  Female  bethany14@example.com    1910-03-24
2   Kristine    Travis    Male  bthompson@example.com    1992-07-02

         Job Title
0  Games developer
1   Phytotherapist
2        Homeopath
```

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

In [35]:
```
# parse data
df = pd.read_csv("people_data.csv", parse_dates=["Date of birth"])
print(df.info())
```

```
<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: 372.0+ bytes
None
```

In [37]:
```
url = "https://media.geeksforgeeks.org/wp-content/uploads/20241121154629307916/peop
df = pd.read_csv(url)
df
```

Out[37]:

	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

In [39]:
```
# 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)
```

In [43]:
```
# saving the dataframe
df.to_csv('file1.csv')
df
```

Out[43]:

	name	degree	score
0	aparna	MBA	90
1	pankaj	BCA	40
2	sudhir	M.Tech	80
3	Geeku	MBA	98

In [55]:
```
# Saving CSV Without Headers and Index
# saving the dataframe
df.to_csv('file2.csv', header=False, index=False)
df
```

Out[55]:

	name	degree	score
0	aparna	MBA	90
1	pankaj	BCA	40
2	sudhir	M.Tech	80
3	Geeku	MBA	98

In [ ]:
```
# saving the dataframe
df.to_csv(r'C:\Users\Admin\Desktop\file3.csv')
```

In [ ]:
```
#Write a DataFrame to CSV file using Tab Separator
import pandas as pd
import numpy as np

users = {'Name': ['Amit', 'ratna', 'Smruti'],
    'Age': [25,21,20]}

#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)
```

Export Pandas dataframe to a CSV file

A CSV (Comma-Separated Values) file 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.

In [64]:
```
import pandas as pd

scores = {'Name': ['a', 'b', 'c', 'd'],
          'Score': [90, 80, 95, 20]}

df = pd.DataFrame(scores)
print(df)
```

```
  Name  Score
0    a     90
1    b     80
2    c     95
3    d     20
```

In [66]:
```
# Exporting DataFrame to CSV
df.to_csv("your_name.csv")
```

In [72]:
```
#Customizing the CSV Export
#2. Remove Index Column
df.to_csv('your_name.csv', index = False)
```

In [74]:
```
df
```

Out[74]:

	Name	Score
0	a	90
1	b	80
2	c	95
3	d	20

In [76]:
```
#Export only selected column
df.to_csv("your_name.csv", columns = ['Name'])
```

In [78]:
```
df
```

Out[78]:

	Name	Score
0	a	90
1	b	80
2	c	95
3	d	20

In [ ]:
```
# Exclude Header Row
#By default theto_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.
df.to_csv('your_name.csv', header = False)
```

In [ ]:
```
# 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.
df.to_csv("your_name.csv", na_rep = 'nothing')
```

In [88]:
```
#Change Column Separator
df.to_csv("your_name.csv", sep ='\t')
```

How to Read JSON Files with Pandas?

JSON (JavaScript Object Notation) 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

In [100…
```
[

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

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

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

]
```

Out[100…
```
[{'id': 1, 'name': 'Alice', 'age': 25},
 {'id': 2, 'name': 'Bob', 'age': 30},
 {'id': 3, 'name': 'Charlie', 'age': 22}]
```

In [ ]:
```
# reading json
df = pd.read_json('data.json')
print(df.head())
```

In [102…
```
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
```

```
DataFrame using JSON module and `pd.json_normalize()` method:
```

Out[102…

	One.0	One.1	One.2	One.3	One.4	One.5	Two.0	Two.1	Two.2	Two.3	Two.4	Two.5
0	60	60	60	45	45	60	110	117	103	109	117	102

In [104…
```
#  Using pd.DataFrame with a Dictionary
df = pd.DataFrame(data)

print(df)
```

```
   One  Two
0   60  110
1   60  117
2   60  103
3   45  109
4   45  117
5   60  102
```

JSON (JavaScript Object Notation) is a popular way to store and exchange data especially used in web APIs and configuration files. Pandas provides tools to parse JSON data and convert it into structured DataFrames for analysis. In this guide we will explore various ways to read, manipulate and normalize JSON datasets in Pandas.

In [107…
```
import pandas as pd
import requests
```

Reading JSON Files To read a JSON file or URL in pandas we use the read_json function. In the below code path_or_buf is the file or web URL to the JSON file.

pd.read_json(path_or_buf)

Create a DataFrame and Convert It to JSON if you don't have JSON file then create a small DataFrame and see how to convert it to JSON using different orientations.

orient='split': separates columns, index and data clearly. orient='index': shows each row as a key-value pair with its index.

In [110…
```
df = pd.DataFrame([['a', 'b'], ['c', 'd']],
                  index=['row 1', 'row 2'],
                  columns=['col 1', 'col 2'])

print(df.to_json(orient='split'))
print(df.to_json(orient='index'))
```

```
{"columns":["col 1","col 2"],"index":["row 1","row 2"],"data":[["a","b"],["c","d"]]}
{"row 1":{"col 1":"a","col 2":"b"},"row 2":{"col 1":"c","col 2":"d"}}
```

Read the JSON File directly from Web Data

You can fetch JSON data from online sources using the requests library and then convert it to a DataFrame. In the below example it reads and prints JSON data from the specified API endpoint using the pandas library in Python.

requests.get(url) fetches data from the URL.

response.json() converts response to a Python dictionary/list.

json_normalize() converts nested JSON into a flat table.

In [113…

```
import pandas as pd
import requests

url = 'https://jsonplaceholder.typicode.com/posts'
response = requests.get(url)

data = pd.json_normalize(response.json())
data.head()
```

Out[113…

	userId	id	title	body
0	1	1	sunt aut facere repellat provident occaecati e...	quia et suscipit\nsuscipit recusandae consequu...
1	1	2	qui est esse	est rerum tempore vitae\nsequi sint nihil repr...
2	1	3	ea molestias quasi exercitationem repellat qui...	et iusto sed quo iure\nvoluptatem occaecati om...
3	1	4	eum et est occaecati	ullam et saepe reiciendis voluptatem adipisci\...
4	1	5	nesciunt quas odio	repudiandae veniam quaerat sunt sed\nalias aut...

Handling Nested JSON in Pandas

Sometimes JSON data has layers like lists or dictionaries inside other dictionaries then it is called as Nested JSON. To turn deeply nested JSON into a table use json_normalize() from pandas making it easier to analyze or manipulate in a table format.

json.load(f): Loads the raw JSON into a Python dictionary.

json_normalize(d['programs']): Extracts the list under the programs key and flattens it into columns.

In [ ]:
```
import json
import pandas as pd
from pandas import json_normalize

with open('/content/raw_nyc_phil.json') as f:
    d = json.load(f)

nycphil = json_normalize(d['programs'])
nycphil.head(3)
```

In [119…
```
import pandas as pd

df = pd.DataFrame([['a', 'b', 'c'], ['d', 'e', 'f'], ['g', 'h', 'i']],
                  index=['row 1', 'row 2', 'row 3'],
                  columns=['col 1', 'col 2', 'col 3'])

df.to_json('file.json', orient='split', compression='infer', index=True)

df = pd.read_json('file.json', orient='split', compression='infer')
print(df)
```

```
      col 1 col 2 col 3
row 1     a     b     c
row 2     d     e     f
row 3     g     h     i
```

In [121…
```
#  Exporting a More Detailed DataFrame
import pandas as pd

df = pd.DataFrame(data=[
    ['15135', 'Alex', '25/4/2014'],
    ['23515', 'Bob', '26/8/2018'],
    ['31313', 'Smruti', '18/1/2019'],
    ['55665', 'Ratna', '5/5/2026'],
    ['63513', 'Maria', '9/12/2025']],
    columns=['ID', 'NAME', 'DATE OF JOINING'])

df.to_json('file1.json', orient='split', compression='infer')

df = pd.read_json('file1.json', orient='split', compression='infer')
print(df)
```

```
      ID    NAME DATE OF JOINING
0  15135    Alex       25/4/2014
1  23515     Bob       26/8/2018
2  31313  Smruti       18/1/2019
3  55665   Ratna        5/5/2026
4  63513   Maria       9/12/2025
```

JSON Orientations in Pandas

Pandas supports multiple orient options for JSON format allowing different ways to structure the data. Choosing the right orientation depends on the use case.

records: List of dictionaries

columns: Dictionary with column labels

index: Dictionary with row indices

split: Dictionary with index, columns, and data

table: JSON table schema

In [124…
```
df.to_json('file.json', orient='records')
```

In [ ]:
```

```