

## Day-10

### Count values in Pandas Data Frame:

**Syntax:** DataFrame.count(axis=0, level=None, numeric\_only=False)

**Returns:** It returns count of non-null values and if level is used it returns dataframe.

#### Steps:

- 1) Importing libraries
- 2) Create dataframe with missing values
- 3) use the **.count()** function to count all the values of different columns.
- 4) By default axis is 0 or index where counts are generated for each column.
- 5) If we want to count all the values with respect to row then we have to pass axis=1 or 'columns'.
- 6) count null values in our dataframe.
- 7) Count based on some condition

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists several Python files: Lambda function.py, Lists and slicing.py, Map & Map methods.py, Mapping function.py, Method overriding.py, Module Calculator.py, nested.loop.py, nested.json.py, nestedJSONtoCSV.py, Number function.py, Operators.py, pandas\_count.py, pandas\_simpleprgm.py, pass.py, Polymorphism.py, Processing python lists.py, Read data from CSV file.py, ReadWriteintoCSVfile.py, Set & Set methods.py, and set.add.clear.discard.py. The current file, pandas\_count.py, is open in the editor. The code creates a DataFrame with missing values and prints it. The output window shows the DataFrame with columns Name, Physics, Chemistry, and Math, and rows numbered 0 to 5. The terminal tab shows the command run and the resulting output.

```
# Count values in Pandas dataframe
import numpy as np
import pandas as pd

# create dataframe with missing values
NaN = np.nan
dataframe = pd.DataFrame({'Name': ['Shobhit', 'Vaibhav', 'Vimal', 'Sourabh', 'Rahul', 'Shobhit'],
                           'Physics': [11, 12, 13, 14, NaN, 11],
                           'Chemistry': [10, 14, NaN, 18, 20, 10],
                           'Math': [13, 10, 15, NaN, NaN, 13]})

print(dataframe)
```

	Name	Physics	Chemistry	Math
0	Shobhit	11.0	10.0	13.0
1	Vaibhav	12.0	14.0	10.0
2	Vimal	13.0	NaN	15.0
3	Sourabh	14.0	18.0	NaN
4	Rahul	NaN	20.0	NaN
5	Shobhit	11.0	10.0	13.0

Process finished with exit code 0

## Count with respect to rows and columns

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help. The title bar says "JSON load.dumps fnsp.py - pandas\_count.py". The left sidebar has a "Project" view with files like Lambda function.py, Lists and slicing.py, Map & Map methods.py, etc., and a "Run" view showing the output of "pandas\_count". The main code editor contains the following Python script:

```
1 # Count values in Pandas dataframe
2 import numpy as np
3 import pandas as pd
4
5 # create dataframe with missing values
6 NaN = np.nan
7
8 dataframe = pd.DataFrame({'Name': ['Shobhit', 'Vaibhav',
9                                'Vimal', 'Sourabh',
10                               'Rahul', 'Shobhit'],
11                             'Physics': [11, 12, 13, 14, NaN, 11],
12                             'Chemistry': [10, 14, NaN, 18, 20, 10],
13                             'Math': [13, 10, 15, NaN, NaN, 13]})
14
15 print(dataframe)
16 print("Count with respect to column:")
17 print(dataframe.count())
18
19 #count all the values with respect to row give axis=0 or axis='columns'
20 print("Count with respect to row:")
21 print(dataframe.count(axis=1))
```

The "Run" tab shows the output:

```
4 2
5 4
dtype: int64

Process finished with exit code 0
```

The screenshot shows another instance of the PyCharm IDE. The title bar says "JSON load.dumps fnsp.py - pandas\_count.py". The left sidebar has a "Project" view with files like Lambda function.py, Lists and slicing.py, Map & Map methods.py, etc., and a "Run" view showing the output of "pandas\_count". The main code editor contains the same Python script as above. The "Run" tab shows the output:

```
'Physics': [11, 12, 13, 14, NaN, 11],
'Chemistry': [10, 14, NaN, 18, 20, 10],
'Math': [13, 10, 15, NaN, NaN, 13]}

print(dataframe)
print("Count with respect to column:")
print(dataframe.count())

#count all the values with respect to row give axis=0 or axis='columns'
print("Count with respect to row:")
print(dataframe.count(axis=1))

4  Rahul   NaN    20.0  NaN
5  Shobhit 11.0   10.0  13.0
Count with respect to column:
Name      6
Physics   5
Chemistry 5
Math      4
dtype: int64
Count with respect to row:
0    4
1    4
2    3
3    3
4    2
5    4
dtype: int64

Process finished with exit code 0
```

## Count of null values in individual columns and total count of null values

The screenshot shows a PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and a tab for 'JSON load.loads.dumps fn.py - pandas\_count.py'. Below the navigation bar is a toolbar with icons for file operations like New, Open, Save, and Run. The main area is divided into two panes: a Project pane on the left containing a list of Python files and a Run pane on the right displaying the output of the executed code.

```
print(dataframe)
print("Count with respect to column:")
print(dataframe.count())

#count all the values with respect to row give axis=0 or axis='columns'
print("Count with respect to row:")
print(dataframe.count(axis=1))

#count null values in our dataframe.
#count of individual columns count of null values
print("Count of null values of individual columns")
print(dataframe.isnull().sum())

# count of total null values present in our dataframe
print("Count of total null values")
print(dataframe.isnull().sum().sum())

Process finished with exit code 0
```

The Run pane shows the output of the script:

```
Count of null values of individual columns
Name          0
Physics        1
Chemistry      1
Math           2
dtype: int64
Count of total null values
4

Process finished with exit code 0
```

count no of students whose physics marks are greater than 11.

The screenshot shows a PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and a tab for 'JSON load.loads.dumps fn.py - pandas\_count.py'. Below the navigation bar is a toolbar with icons for file operations like New, Open, Save, and Run. The main area is divided into two panes: a Project pane on the left containing a list of Python files and a Run pane on the right displaying the output of the executed code.

```
# count of total null values present in our dataframe
print("Count of total null values")
print(dataframe.isnull().sum().sum())

#count no of students whose physics marks are greater than 11.
print("Count of students with physics marks greater than 11 is->",
      dataframe[dataframe['Physics'] > 11]['Name'].count())
print(dataframe[dataframe['Physics']>11])
```

The Run pane shows the output of the script:

```
Count of total null values
4

Count of students with physics marks greater than 11 is->
      Name Physics Chemistry Math
1  Vaibhav    12.0     14.0   10.0
2   Vimal     13.0      NaN   15.0
3 Sourabh    14.0     18.0      NaN

Process finished with exit code 0
```

## Count of students whose physics marks are greater than 10,chemistry marks are greater than 11 and math marks are greater than 9.

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and JSON load, loads, dumps fn.py - pandas\_count.py. The left sidebar shows a project structure with files like Lambda function.py, Lists and slicing.py, Map & Map methods.py, Mapping function.py, Method overriding.py, Module Calculator.py, nested loop.py, nested\_json.json, nestedJSONtoCSV.py, Number function.py, Operators.py, pandas\_count.py, pandas\_simpleprgm.py, pass.py, Polymorphism.py, Processing python lists.py, Read data from CSV file.py, ReadWriteintoCSVFile.py, Set & Set methods.py. The main code editor window contains the following Python script:

```
print(dataframe['Physics']>=10)
#Count of students whose physics marks are greater than 10.
print("Count of students is ->",dataframe[(dataframe["Physics"]>=10)&(dataframe["Chemistry"]>=11)&(dataframe["Math"]>9)]['Name'].count())

print(dataframe[(dataframe['Physics'] > 10) &
               (dataframe['Chemistry'] > 11) &
               (dataframe['Math'] > 9)])
```

The Run tab shows the output of the script:

```
4
Count of students with physics marks greater than 11 is-> 3
   Name Physics Chemistry Math
1  Vaibhav    12.0     14.0  10.0
2   Vimal    13.0      NaN  15.0
3 Sourabh    14.0     18.0   NaN
Count of students is -> 1
   Name Physics Chemistry Math
1  Vaibhav    12.0     14.0  10.0

Process finished with exit code 0
```

The bottom status bar shows PEP 8: E303 too many blank lines (4), 41:1 CRLF UTF-8 4 spaces Python 3.9, and a system tray with various icons.

## Get Count by Status using Pandas Dataframe APIs

1. Create a DataFrame .
2. Use the `groupby()` function to group the data by the 'status' column.
3. Use the `size()` function to get the count of each group.

## Example-1:

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help. The title bar says "JSON load.loads.dumps fn.py - pandas\_count.py". The code editor contains Python code to count rows where 'Math' is greater than 9. The run tab shows the output of the script.

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.loads.dumps fn.py - pandas_count.py
DEPython Project
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.loads.dumps fn.py - pandas_count.py x Sorting_JSON.py x pandas_count.py x pandas_joins.py x
Run: pandas_count
1  Vaibhav  12.0  14.0  10.0
...
8 Customer A  102   450  Active  City A
1 Customer B  101   300  Inactive City B
2 Customer C  103   600  Active  City A
3 Customer D  104   550  Inactive City C
4 Customer E  105   400  Active  City B
status
Active      3
Inactive    2
dtype: int64
```

Indexing completed in 21 sec. Shared indexes were applied to 5,742 of 5,756 files (99%) (today 09:57) 55:38 Python 3.9

## Example-2

By passing null values

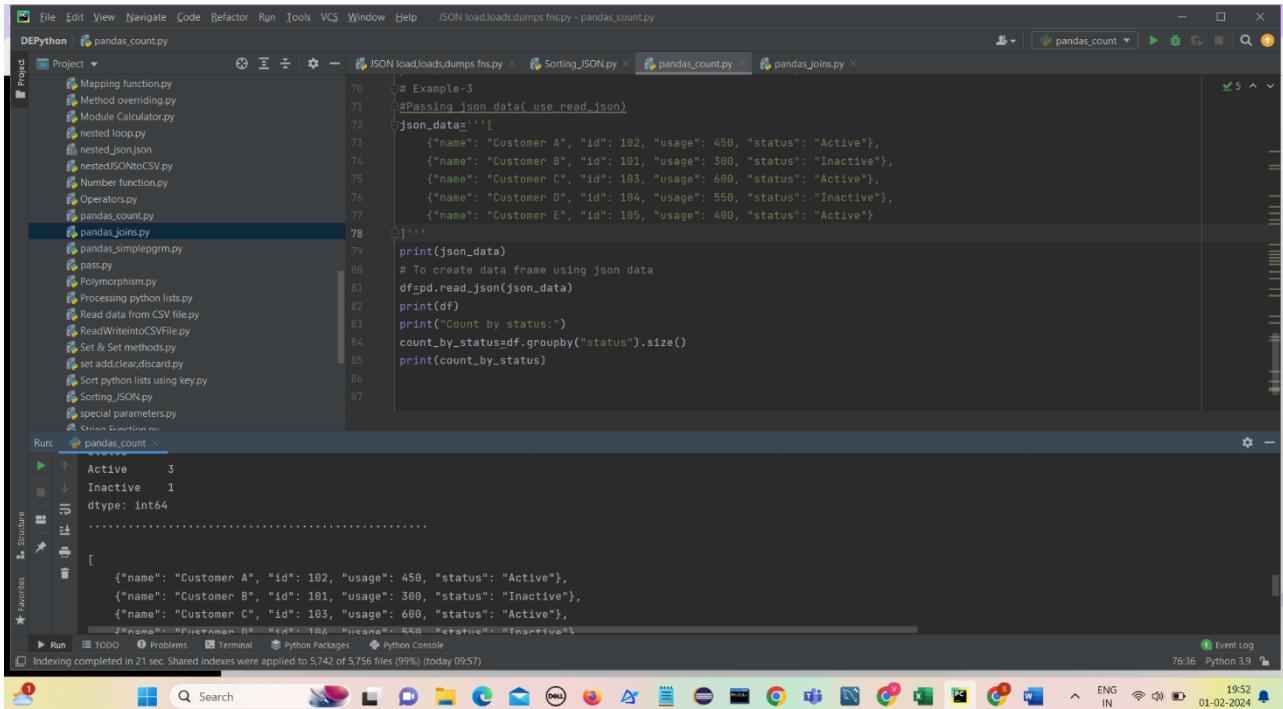
The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help. The title bar says "JSON load.loads.dumps fn.py - pandas\_count.py". The code editor contains Python code to count rows by status, including a row with a null value in the 'usage' column. The run tab shows the output of the script.

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.loads.dumps fn.py - pandas_count.py x Sorting_JSON.py x pandas_count.py x pandas_joins.py x
Run: pandas_count
...
8 Customer A  102   450  Active
1 Customer B  101   300  None
2 Customer C  103   600  Active
3 Customer D  104   550  Inactive
4 Customer E  105   400  Active
status
Active      3
Inactive    1
dtype: int64
```

63:31 CRF UTF-8 4 spaces Python 3.9 19:45 01-02-2024

### Example-3:

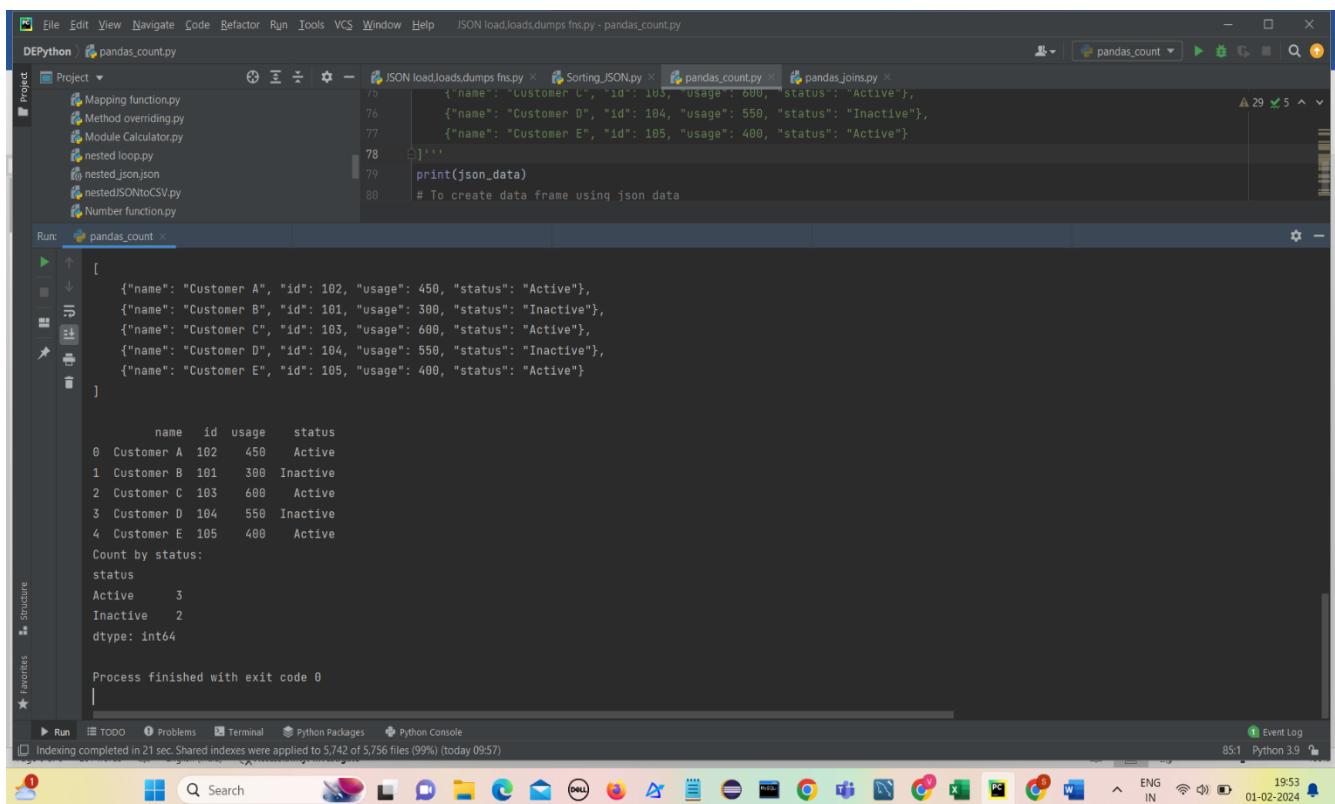
#### Passing json data( use read\_json)



```
# Example-3
#Passing json data( use read_json)
json_data='''[
    {"name": "Customer A", "id": 102, "usage": 450, "status": "Active"},
    {"name": "Customer B", "id": 101, "usage": 300, "status": "Inactive"},
    {"name": "Customer C", "id": 103, "usage": 600, "status": "Active"},
    {"name": "Customer D", "id": 104, "usage": 550, "status": "Inactive"},
    {"name": "Customer E", "id": 105, "usage": 400, "status": "Active"}
]'''
print(json_data)
# To create data frame using json data
df=pd.read_json(json_data)
print(df)
print("Count by status:")
count_by_status=df.groupby("status").size()
print(count_by_status)

Active      3
Inactive    2
dtype: int64

[{"name": "Customer A", "id": 102, "usage": 450, "status": "Active"}, {"name": "Customer B", "id": 101, "usage": 300, "status": "Inactive"}, {"name": "Customer C", "id": 103, "usage": 600, "status": "Active"}, {"name": "Customer D", "id": 104, "usage": 550, "status": "Inactive"}, {"name": "Customer E", "id": 105, "usage": 400, "status": "Active"}]
```



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.loads.dumps fns.py - pandas_count.py
DEPython pandas_count.py
Project
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.loads.dumps fns.py x Sorting_JSON.py x pandas_count.py x pandas.joins.py x
Run: pandas_count x
[{"name": "Customer A", "id": 102, "usage": 450, "status": "Active"}, {"name": "Customer B", "id": 101, "usage": 300, "status": "Inactive"}, {"name": "Customer C", "id": 103, "usage": 600, "status": "Active"}, {"name": "Customer D", "id": 104, "usage": 550, "status": "Inactive"}, {"name": "Customer E", "id": 105, "usage": 400, "status": "Active"}]

      name   id  usage  status
0 Customer A  102    450  Active
1 Customer B  101    300  Inactive
2 Customer C  103    600  Active
3 Customer D  104    550  Inactive
4 Customer E  105    400  Active
Count by status:
status
Active      3
Inactive    2
dtype: int64

Process finished with exit code 0
```

## Get Count by Month and Status using Pandas Dataframe APIs

1. Create a DataFrame
2. Convert the relevant date column to a datetime type (if applicable).
3. Extract the month from the date column.
4. Group the data by month and status columns.
5. Use the **size()** function to get the count of each group

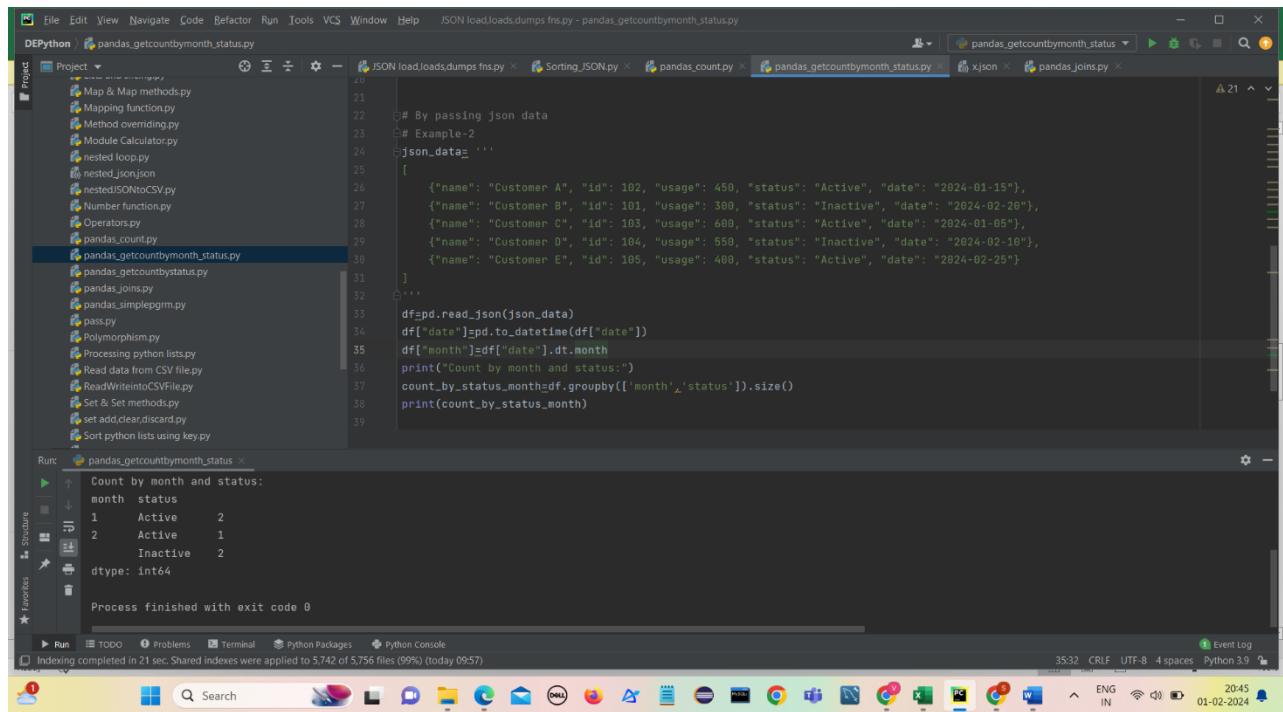
### Example-1

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.loads.dumps fns.py - pandas_count.py
DEPython > pandas_count.py
Project > Set & Set methods.py
          Set add,clear,discard.py
          Sort python lists using key.py
          Sorting_JSON.py
          special parameters.py
          String Function.py
          Student_details.csv
          Students_Data.csv
          Students_Data.csv
          teacher.json
          tuple.py
          typing.txt
          Unique elements from list.py
          Variables.py
          while loop.py
          writelines.txt
          x.json
          x.py
          xyz.txt
> External Libraries
Scratches and Consoles
Run: pandas_count >
      dtype: int64
      .....
      month  status
      1      Active   2
      2      Active   1
              Inactive  2
      9      Inactive  1
      dtype: int64
Process finished with exit code 0
Run TODO Problems Terminal Python Packages Python Console Event Log
Indexing completed in 21 sec. Shared indexes were applied to 5,742 of 5,756 files (99%) (today 09:57) 90:1 Python 3.9
01-02-2024 20:07 ENG IN
```

The screenshot shows the PyCharm IDE interface with a Python script named 'pandas\_count.py' open. The code demonstrates how to use Pandas Dataframe APIs to get the count of data by month and status. The code includes comments explaining steps 1 through 5. The run tab shows the output of the script, which is a Pandas DataFrame with columns 'month' and 'status', and row counts 1, 2, 9, and 1 respectively. The status values are Active (2), Active (1), Inactive (2), and Inactive (1). The bottom status bar indicates indexing completed at 09:57 and the current date and time as 01-02-2024 20:07.

## Example-2

### By passing Json data

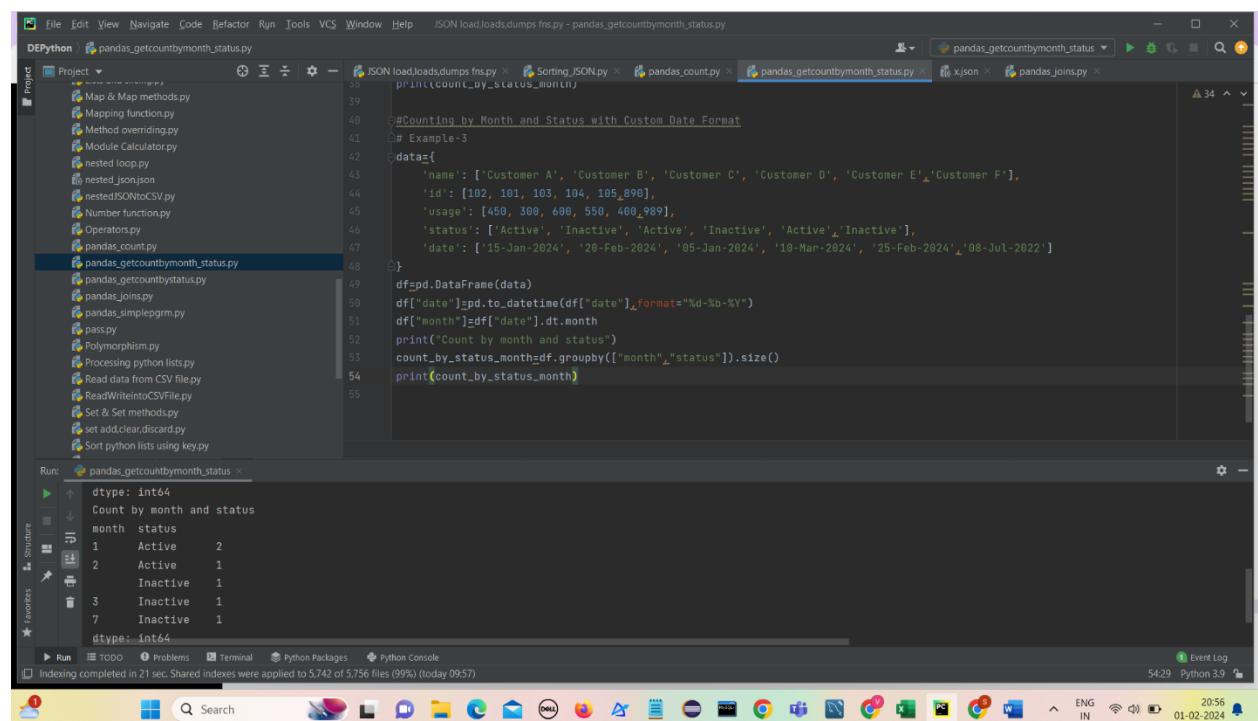


```
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.loads.dumps fns.py - pandas_getcountbymonth_status.py
DEPython > pandas_getcountbymonth_status.py
Project
  Map & Map methods.py
  Mapping function.py
  Method overriding.py
  Module Calculator.py
  nested_loop.py
  nested_json.json
  nestedJSONtoCSV.py
  Number function.py
  Operators.py
  pandas.count.py
  pandas_getcountbymonth_status.py
  pandas_getcountbystatus.py
  pandas.joins.py
  pandas_simplepgm.py
  pass.py
  Polymorphism.py
  Processing python lists.py
  Read data from CSV file.py
  ReadWriteintoCSVfile.py
  Set & Set methods.py
  set.add.clear.discard.py
  Sort python lists using key.py
  pandas_getcountbymonth_status.py
  Count by month and status:
    month status
    1 Active 2
    2 Active 1
    Inactive 2
  dtype: int64
  Process finished with exit code 0
Run: pandas_getcountbymonth_status
Indexing completed in 21 sec. Shared indexes were applied to 5,742 of 5,756 files (99%) (today 09:57)
  Search  Python Packages  Python Console
  Event Log 3532 CRLF UTF-8 4 spaces Python 3.9
  01-02-2024 20:45 ENG IN
  Icons: File Explorer, Project Explorer, Task List, Problems, Run, Terminal, Python Packages, Python Console, Event Log, Indexing Progress, Status Bar
```

The code reads a JSON string containing customer data and groups it by month and status. The output shows two Active customers in month 1 and one Active customer in month 2.

## Example-3

### Counting by Month and Status with Custom Date Format



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.loads.dumps fns.py - pandas_getcountbymonth_status.py
DEPython > pandas_getcountbymonth_status.py
Project
  Map & Map methods.py
  Mapping function.py
  Method overriding.py
  Module Calculator.py
  nested_loop.py
  nested_json.json
  nestedJSONtoCSV.py
  Number function.py
  Operators.py
  pandas.count.py
  pandas_getcountbymonth_status.py
  pandas_getcountbystatus.py
  pandas.joins.py
  pandas_simplepgm.py
  pass.py
  Polymorphism.py
  Processing python lists.py
  Read data from CSV file.py
  ReadWriteintoCSVfile.py
  Set & Set methods.py
  set.add.clear.discard.py
  Sort python lists using key.py
  pandas_getcountbymonth_status.py
  print(count_by_status_month)
  #Counting by Month and Status with Custom Date Format
  # Example-3
  data={
    'name': ['Customer A', 'Customer B', 'Customer C', 'Customer D', 'Customer E', 'Customer F'],
    'id': [102, 101, 103, 104, 105, 890],
    'usage': [450, 300, 600, 550, 400, 989],
    'status': ['Active', 'Inactive', 'Active', 'Inactive', 'Active', 'Inactive'],
    'date': ['15-Jan-2024', '20-Feb-2024', '05-Jan-2024', '10-Mar-2024', '25-Feb-2024', '08-Jul-2022']
  }
  df=pd.DataFrame(data)
  df["date"] = pd.to_datetime(df["date"], format="%d-%b-%Y")
  df["month"] = df["date"].dt.month
  print("Count by month and status:")
  count_by_status_month=df.groupby(["month","status"]).size()
  print(count_by_status_month)
Run: pandas_getcountbymonth_status
  dtype: int64
  Count by month and status:
    month status
    1 Active 2
    2 Active 1
    3 Inactive 1
    7 Inactive 1
  dtype: int64
  Process finished with exit code 0
Indexing completed in 21 sec. Shared indexes were applied to 5,742 of 5,756 files (99%) (today 09:57)
  Search  Python Packages  Python Console
  Event Log 5429 CRLF UTF-8 4 spaces Python 3.9
  01-02-2024 20:56 ENG IN
```

The code reads a JSON string with a custom date format ('%d-%b-%Y') and groups the data by month and status. The output shows counts for Active and Inactive customers across different months.

# Joins between Pandas dataframes

Different types of join operations can be performed on Pandas Dataframe.

There are five types of Joins in Pandas.

- Inner Join
- Left Outer Join
- Right Outer Join
- Full Outer Join or simply Outer Join
- Index Join

Create two dataframes a and b

The image shows two side-by-side screenshots of the DEPython IDE. Both windows have the title bar 'DEPython' and the project name 'pandas\_joins.py'. The left window displays the following Python code:

```
#># joins between Pandas dataframes
#># create two dataframes a and b
#>import pandas as pd
#>pd.DataFrame()
#>print(a)
#># Create dictionary
#>d={ 'id':[1,2,10,12],
#>     'val1':[ 'a','b','c','d']}
#>a=pd.DataFrame(d)
#>print(a)
#>b=pd.DataFrame()
#>print(b)
#>d={ 'id':[1,2,9,8],
#>     'val1':[ 'p','q','r','s']}
#>b=pd.DataFrame(d)
#>print(b)
```

The right window shows the output of running this code in the 'Run' tab. It displays two dataframes, 'a' and 'b', both of which are currently empty DataFrames.

Both windows also show a 'Project' sidebar with various Python files listed, and a 'Run' tab at the bottom with the command 'C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/pandas\_joins.py"'.

## 1) Inner Join

- It returns a Dataframe with only those rows that have common characteristics.
- This is similar to the intersection of two sets.

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists various Python files. The main editor window contains the following code:

```
# Create dictionary
d=[{'id':1,'val1':1},{'id':2,'val1':2},{'id':3,'val1':3},{'id':4,'val1':4},{'id':5,'val1':5},{'id':6,'val1':6},{'id':7,'val1':7},{'id':8,'val1':8},{'id':9,'val1':9},{'id':10,'val1':10}]
df=pd.DataFrame(d)
print(df)

b=pd.DataFrame()
print(b)
d=[{'id':1,'val1':1},{'id':2,'val1':2},{'id':3,'val1':3},{'id':4,'val1':4},{'id':5,'val1':5},{'id':6,'val1':6},{'id':7,'val1':7},{'id':8,'val1':8},{'id':9,'val1':9},{'id':10,'val1':10}]
df=pd.DataFrame(d)
print(df)

# Inner join
print("Inner Join")
df=df.merge(a,b,on='id',how='inner')
print(df)
```

The 'Run' tab shows the output of the code:

```
Inner Join
      id val1_x val1_y
0    1     a      p
1    2     b      q
2    10    c    Null
3    12    d    Null
```

The status bar at the bottom right indicates the date and time as 01-02-2024.

## 2) Left Outer Join

- All the records from the first Dataframe will be displayed, irrespective of whether the keys in the first Dataframe can be found in the second Dataframe.
- For the second Dataframe, only the records with the keys in the second Dataframe that can be found in the first Dataframe will be displayed.

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists various Python files. The main editor window contains the following code:

```
b=pd.DataFrame()
print(b)
d=[{'id':1,'val1':1},{'id':2,'val1':2},{'id':3,'val1':3},{'id':4,'val1':4},{'id':5,'val1':5},{'id':6,'val1':6},{'id':7,'val1':7},{'id':8,'val1':8},{'id':9,'val1':9},{'id':10,'val1':10}]
df=pd.DataFrame(d)
print(df)

# Inner join
print("Inner Join")
df=df.merge(a,b,on='id',how='inner')
print(df)

# Left outer join
print("Left Outer Join")
df=df.merge(a,b,on='id',how='left')
print(df)
```

The 'Run' tab shows the output of the code:

```
Left Outer Join
      id val1_x val1_y
0    1     a      p
1    2     b      q
2    10    c    Null
3    12    d    Null
```

The status bar at the bottom right indicates the date and time as 01-02-2024.

### 3) Right outer join

- All the records from the second Dataframe will be displayed.
- Only the records with the keys in the first Dataframe that can be found in the second Dataframe will be displayed.

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and JSON load.loads.dumps fn.py - pandas\_joins.py. The left sidebar has a Project view with various Python files listed. The main editor window contains the following code:

```
18
19 # Inner join
20 print("Inner Join")
21 dfpd.merge(a,b,on='id',how='inner')
22 print(df)
23
24 # Left outer join
25 print("Left Outer Join")
26 dfpd.merge(a,b,on='id',how='left')
27 print(df)
28
29 # Right outer join
30 print("Right Outer Join")
31 dfpd.merge(a,b,on='id',how='right')
32 print(df)
```

The Run tab shows the output of the code:

```
Right Outer Join
id val1_x val1_y
0 1 a p
1 2 b q
2 9 NaN r
3 8 NaN s
```

The status bar at the bottom right indicates the process finished with exit code 0, and the system tray shows the date and time as 01-02-2024.

### 4) Full Outer Join

returns all the rows from the left Dataframe, and all the rows from the right Dataframe, and matches up rows where possible, with NaNs elsewhere.

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and JSON load.loads.dumps fn.py - pandas\_joins.py. The left sidebar has a Project view with various Python files listed. The main editor window contains the following code:

```
23
24 # Left outer join
25 print("Left Outer Join")
26 dfpd.merge(a,b,on='id',how='left')
27 print(df)
28
29 # Right outer join
30 print("Right Outer Join")
31 dfpd.merge(a,b,on='id',how='right')
32 print(df)
33
34 # Full outer join
35 print("Full outer join")
36 df = pd.merge(a, b, on='id', how='outer')
37 print(df)
```

The Run tab shows the output of the code:

```
Full outer join
id val1_x val1_y
0 1 a p
1 2 b q
2 8 NaN s
3 9 NaN r
4 10 c NaN
5 12 d NaN
```

The status bar at the bottom right indicates the process finished with exit code 0, and the system tray shows the date and time as 01-02-2024.

## 5) Index Join

- To merge the Dataframe on indices.
- pass the *left\_index* and *right\_index* arguments as True i.e. both the Dataframes are merged on an index using default Inner Join.

The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and a tab for 'pandas\_joins.py'. Below the navigation bar is a project tree with files like Lambda function.py, Lists and slicing.py, etc. The main code editor window contains the following Python code:

```
print(df)
# Right outer join
print("Right Outer Join")
df=pd.merge(a,b, on='id', how='right')
print(df)

# Full outer join
print("Full outer join")
df = pd.merge(a, b, on='id', how='outer')
print(df)

# Index Join
print("Index Join")
df=pd.merge(a,b, left_index=True, right_index=True)
print(df)
```

Below the code editor is a 'Run' section with a dropdown set to 'pandas\_joins'. The output pane shows the results of the execution:

```
4 10    c   NaN
5 12    d   NaN
Index Join
  id_x val1_x  id_y val1_y
0     1      a     1      p
1     2      b     2      q
2    10      c     9      r
3    12      d     8      s
```

At the bottom of the PyCharm interface, there's a toolbar with icons for file operations, and a status bar showing the date and time.

## Read data from CSV files to Pandas Data Frames

The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and a tab for 'pandas\_ReadCSVdata\_to\_DataFrame.py'. Below the navigation bar is a project tree with files like abc.txt, Access Specifiers.py, app.txt, etc. The main code editor window contains the following Python code:

```
# Reading CSV data using pandas
# 1) read_csv method
import pandas as pd
data=pd.read_csv("Example-1.csv")
print(data)
print(type(data))
print("Names:")
print(data.Name)
print("Salary")
print(data.Salary)
# displays first 5 rows
print(data.head())
print('.....')
# 2) read_table method
print("Read table method")
data=pd.read_table("Example-1.csv", delimiter=',')
print(data)
print(type(data))
print('.....')
# 3) Using the csv module
import csv
with open('Example-1.csv') as csv_file:
    csv_reader=csv.reader(csv_file)
    df=pd.DataFrame(csv_reader)
print(df)
print(type(df))
```

Below the code editor is a 'Run' section with a dropdown set to 'pandas\_ReadCSVdata\_to\_DataFrame (1)'. The output pane shows the results of the execution.

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.dumps fn.py - pandas_ReadCSVdata_to_Dataframe.py
DEPython: pandas_ReadCSVdata_to_Dataframe(1) ×
Project Run File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.dumps fn.py - pandas_ReadCSVdata_to_Dataframe.py
Run: pandas_ReadCSVdata_to_Dataframe(1) ×
C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/pandas_ReadCSVdata_to_DataFrame.py"
   0   1   Sona  12  12000
   1   2   Mona  21  43000
   2   3   Bannu 13  25000
   3   4   Chinnu 23  10000
   4   5   sunny 14  23000
   5   6   raju 24  78000
   6   7   ram 15  45000
   7   8   lucky 25  50000
   8   9   Appy 17  60000
   9   10  Lekha 38  21890
  10  11  Vinnni 11  78654
  11  12  pavan 19  34567
  12  13  Sita 32  78901
  13  14  latha 33  56455
  14  15  Gita 28  25678
<class 'pandas.core.frame.DataFrame'>
Names:
  0   Sona
  1   Mona
  2   Bannu
  3   Chinnu
  4   sunny
  5   raju
  6   ram
  7   lucky
  8   Appy
  9   Lekha
  10  Vinnni
  11  pavan
  12  Sita
  13  latha
  14  Gita
Name: Name, dtype: object
Salary
  0   12000
  1   43000
  2   25000
  3   10000
  4   23000
  5   78000
  6   45000
  7   50000
  8   60000
  9   21890
  10  78654
  11  34567
  12  78901
  13  56455
  14  25678
Name: Salary, dtype: int64
  0   ID   Name  Age  Salary
  1   1    Sona  12  12000
  1   2    Mona  21  43000
  2   3    Bannu 13  25000
  3   4    Chinnu 23  10000
  4   5    sunny 14  23000
  5   6    raju 24  78000
  6   7    ram 15  45000
  7   8    lucky 25  50000
  8   9    Appy 17  60000
  9   10   Lekha 38  21890
  10  11   Vinnni 11  78654
  11  12   pavan 19  34567
  12  13   Sita 32  78901
  13  14   latha 33  56455
  14  15   Gita 28  25678
<class 'pandas.core.frame.DataFrame'>
Indexing completed in 21 sec. Shared indexes were applied to 5,742 of 5,756 files (99%) (today 09:57)
Event Log 86:1 Python 3.9
Run TODO Problems Terminal Python Packages Python Console
22:15 01-02-2024
```

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.dumps fn.py - pandas_ReadCSVdata_to_Dataframe.py
DEPython: pandas_ReadCSVdata_to_Dataframe(1) ×
Project Run File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.dumps fn.py - pandas_ReadCSVdata_to_Dataframe.py
Run: pandas_ReadCSVdata_to_Dataframe(1) ×
  8   9   Appy
  9   10  Lekha
  10  11  Vinnni
  11  12  pavan
  12  13  Sita
  13  14  latha
  14  15  Gita
Name: Name, dtype: object
Salary
  0   12000
  1   43000
  2   25000
  3   10000
  4   23000
  5   78000
  6   45000
  7   50000
  8   60000
  9   21890
  10  78654
  11  34567
  12  78901
  13  56455
  14  25678
Name: Salary, dtype: int64
  0   ID   Name  Age  Salary
  1   1    Sona  12  12000
  1   2    Mona  21  43000
  2   3    Bannu 13  25000
  3   4    Chinnu 23  10000
  4   5    sunny 14  23000
  5   6    raju 24  78000
  6   7    ram 15  45000
  7   8    lucky 25  50000
  8   9    Appy 17  60000
  9   10   Lekha 38  21890
  10  11   Vinnni 11  78654
  11  12   pavan 19  34567
  12  13   Sita 32  78901
  13  14   latha 33  56455
  14  15   Gita 28  25678
<class 'pandas.core.frame.DataFrame'>
Indexing completed in 21 sec. Shared indexes were applied to 5,742 of 5,756 files (99%) (today 09:57)
Event Log 86:1 Python 3.9
Run TODO Problems Terminal Python Packages Python Console
22:16 01-02-2024
```

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.dumps fn.py - pandas_ReadCSVdata_to_Dataframe.py
DEPython: pandas_ReadCSVdata_to_Dataframe(1) ×
Project Run File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.dumps fn.py - pandas_ReadCSVdata_to_Dataframe.py
Run: pandas_ReadCSVdata_to_Dataframe(1) ×
   0   1   Sona  12  12000
   1   2   Mona  21  43000
   2   3   Bannu 13  25000
   3   4   Chinnu 23  10000
   4   5   sunny 14  23000
   5   6   raju 24  78000
   6   7   ram 15  45000
   7   8   lucky 25  50000
   8   9   Appy 17  60000
   9   10  Lekha 38  21890
  10  11  Vinnni 11  78654
  11  12  pavan 19  34567
  12  13  Sita 32  78901
  13  14  latha 33  56455
  14  15  Gita 28  25678
<class 'pandas.core.frame.DataFrame'>
.....
      0   ...
      0   [ID, Name, Age, Salary] ... [15, Gita, 28, 25678]
[1 rows x 16 columns]
<class 'pandas.core.frame.DataFrame'>
Process finished with exit code 0
Run TODO Problems Terminal Python Packages Python Console
22:16 01-02-2024
```

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load.dumps fn.py - pandas_ReadCSVdata_to_DataFrame.py
DEPython C:\Users\DELL\OneDrive\Desktop\Data Engineering
Project
  DEPython
    abc.txt
    Access Specifiers.py
    app.txt
    Arbitrary args.py
    Book1.xlsx
    break.py
    Calculator.py
    Class and Obj.py
    Constructor.py
    continue.py
    data.json
    data1.json
    Datatypes.py
    Date and time function.py
    Default arguments.py
    Dictionary and dict methods.py
Run: pandas_ReadCSVdata_to_DataFrame () ×
  ID  Name  Age  Salary
0   1  Sona  12  12000
1   2  Mona  21  43000
2   3  Banni 13  25000
3   4  Chinnu 23  10000
4   5  sunny  14  23000
5   6  raju  24  78000
6   7  ram   15  45000
7   8  Lucky  25  50000
8   9  Appy  17  60000
9  10  Lekha 38  21890
10 11  Vinnni 11  78654
11 12  pavan 19  34567
12 13  Sita  32  78901
  Run TODO Problems Terminal Python Packages Python Console
Indexing completed in 21 sec. Shared indexes were applied to 5,742 of 5,756 files (99%) (today 09:57)
  2844 CRLF UTF-8 4 spaces Python 3.9 Event Log
  01-02-2024 22:24 ENG IN

```

## Filter data in pandas Dataframe using query

It is used to filter dataframe

**Syntax:** DataFrame.query(expr, inplace=False, \*\*kwargs)

**Parameters:**

- **expr:** Expression in string form to filter data.
- **inplace:** Make changes in the original data frame if True
- **kwargs:** Other keyword arguments.

**Return type:** Filtered Data frame

Dataframe.query() method only works if the column name doesn't have any empty spaces.

If any empty spaces are there, before using query method replace all empty spaces with “\_”

```

# replacing blank spaces with '_'
data.columns =
    [column.replace(" ", "_") for column in data.columns]

```

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists files like Number function.py, Operators.py, pandas\_count.py, etc., and the current file is pandas\_filter\_using\_query.py. The code editor displays the following Python script:

```
import pandas as pd
data=pd.read_csv("Example-1.csv")
print(type(data))
# replacing blank spaces with '_' ( only if necessary)
data.columns =[column.replace(" ", "_") for column in data.columns]
a=data.query("Age==14")
print(a)
a=data.query("Name=='Sona'")
print(a)
print(data.query('Age>15'))
```

The Run tab below shows the command: C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/pandas\_filter\_using\_query.py". The output pane displays the filtered DataFrame:

ID	Name	Age	Salary
4	sunny	14	23000
0	Sona	12	12000
1	Mena	21	43000
3	Chinnu	23	10000
5	raju	24	78000
7	lucky	25	50000
8	Appy	17	60000
9	Lekha	38	21890
11	pavan	19	34567
12	Sita	32	78901
13	Lathe	33	56455
14	Gita	28	25678

The system tray at the bottom right shows the date as 01-02-2024.

## Filtering with multiple conditions

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists files like Number function.py, Operators.py, pandas\_count.py, etc., and the current file is pandas\_filter\_using\_query.py. The code editor displays the following Python script:

```
a=data.query('Name=="Sona"')
print(a)
print('.....')
# Filter using multiple conditions
a=data.query('ID==4' and 'Age==23')
print(a)

a=data.query('ID==6' and 'Name=="raju" and 'Salary==78000')
print(a)
```

The Run tab below shows the command: C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/pandas\_filter\_using\_query.py". The output pane displays the filtered DataFrame:

ID	Name	Age	Salary
4	sunny	14	23000
0	Sona	12	12000
3	Chinnu	23	10000
5	raju	24	78000

The system tray at the bottom right shows the date as 01-02-2024.

## Records with salary>25000

```
# Filter using multiple conditions
a=data.query('ID==4 and Age==23')
print(a)

a=data.query('ID==6 and Name=="raju" and Salary==78000')
print(a)

# More examples
#Records with salary>25000
salary_limit=25000
print("Records with salary>25000")
print(data.query('Salary>'+str(salary_limit)))
```

ID	Name	Age	Salary
5	raju	24	78000
1	Mona	21	43000
5	raju	24	78000
6	ram	15	45000
7	lucky	25	50000
8	Appy	17	60000
10	Vinni	11	78654
11	pavan	19	34567
12	Sita	32	78901
13	letha	33	56455
14	Gita	28	25678

## Aggregations in Pandas

Aggregation can be used to get a summary of columns in our dataset like getting sum, minimum, maximum, etc. from a particular column of our dataset.

- **sum()** :Compute sum of column values
- **min()** :Compute min of column values
- **max()** :Compute max of column values
- **mean()** :Compute mean of column
- **size()** :Compute column sizes
- **describe()** :Generates descriptive statistics
- **first()** :Compute first of group values
- **last()** :Compute last of group values
- **count()** :Compute count of column values
- **std()** :Standard deviation of column
- **var()** :Compute variance of column
- **sem()** :Standard error of the mean of column

```

import pandas as pd
# Create DataFrame
data=[[9, 4, 8, 9],
      [8, 10, 7, 6],
      [7, 6, 8, 5]]
df=pd.DataFrame(data,columns=['Maths', 'English',
                               'Science', 'History'])
print(df)

```

Process finished with exit code 0

## 1) sum()

```

# 1) sum()
print("sum")
Sum=df.sum()
print(Sum)

```

Sum

	Maths	English	Science	History
Maths	24			
English	20			
Science	23			
History	20			
<b>dtype:</b>	<b>int64</b>			
<b>None</b>				

Process finished with exit code 0

## 2) Minimum and

## 3) Maximum

```

#2) min()
print("minimum")
Min=df.min()
print(Min)

#3) max()
print("maximum")
Max=df.max()
print(Max)

```

	Maths	English	Science	History
Maths	7	4	8	9
English	10	9	8	7
Science	9	7	6	8
History	20	10	8	7
<b>dtype:</b>	<b>int64</b>			
<b>None</b>				

Process finished with exit code 0

## 4) Mean()

## 5) Size()

The screenshot shows the PyCharm IDE interface. The code editor displays a script named `pandas_aggregation.py` with the following content:

```
print(Max)
# 4) mean()
print("Mean")
Mean=df.mean()
print(Mean)

#5) size
print("Size")
Size=df.size() # int object is not callable
print(Size)
```

The run output window shows the results of the executed code:

```
dtype: int64
Maximum
Maths    9
English   10
Science   8
History   9
dtype: int64
Mean
Maths     8.00000
English   6.666667
Science   7.666667
History   6.666667
dtype: float64
Size

Process finished with exit code 1
```

The status bar at the bottom right indicates the date as 01-02-2024.

## 6) describe()

## 7) agg()

The screenshot shows the PyCharm IDE interface. The code editor displays a script named `pandas_aggregation.py` with the following content:

```
print(Size)

# 6) describe
print('describe')
print(df.describe())
print('Agg function')
# 7) agg()
print(df.agg(['sum','min','max','mean']))
```

The run output window shows the results of the executed code:

```
describe
   Maths   English   Science   History
count    3.0    3.000000  3.000000  3.000000
mean     8.0    6.666667  7.666667  6.666667
std      1.0    3.055050  0.577350  2.081666
min      7.0    4.000000  7.000000  5.000000
25%     7.5    5.000000  7.500000  5.500000
50%     8.0    6.000000  8.000000  6.000000
75%     8.5    8.000000  8.000000  7.500000
max     9.0   10.000000  8.000000  9.000000
Agg function
   Maths   English   Science   History
sum    24.0   20.000000  23.000000  20.000000
min    7.0    4.000000  7.000000  5.000000
max   9.0   10.000000  8.000000  9.000000
mean   8.0    6.666667  7.666667  6.666667

Process finished with exit code 0
```

The status bar at the bottom right indicates the date as 02-02-2024.

# Grouping in Pandas

Grouping is used to group data using some criteria from our dataset. It is used as split-apply-combine strategy.

- Splitting the data into groups based on some criteria.
- Applying a function to each group independently.
- Combining the results

```
# 8) group by
a=df.groupby("Maths")
print(a.first())

# group by multiple columns
# First group by Maths and later by english
a=df.groupby(['Maths','English'])
print(a.first())
```

	Maths	English	Science	History
max	9.0	10.000000	8.000000	9.000000
mean	8.0	6.666667	7.666667	6.666667
	Maths	English	Science	History
7	6	8	5	
8	10	7	6	
9	4	8	9	

Process finished with exit code 0

## Example-2

```
# creating an another dataframe
data=[[{"Sona":1,95, 42, 88, 92},
       ["Mona",8, 100, 72, 60, 89],
       ["Chinu",7, 67, 83, 52, 77],
       ["Ganu",1,34,19,56,45],
       ["Sunny",7,33,66,77,76]]]

df=pd.DataFrame(data,columns=['Name','ID','Maths','Physics','Chemistry','Biology'])

print(df)

# group by add and find sum
print(df.groupby('ID').sum())

# group by two columns( first by ID and then by Name)
a=df.groupby(['ID','Name'])
print(a.first())

# group by two columns(first by Name and then ID)
a=df.groupby(['Name','ID'])
print(a.last())

# group by Name
print(df.groupby('Name').mean())
print(df.groupby('ID').sum())
print(df.groupby('ID').min())
print(df.groupby('Name').min())
print(df.groupby(['ID']).max())
print(df.groupby(['Name']).max())
```

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load/loads/dumps/fn.py - pandas_aggregation.py
DEPython | pandas_aggregation.py | pandas_joins.py | pandas_aggregation.py | pandas_getcountbystatus.py | Run: pandas_aggregation | Event Log | Search
Run: pandas_aggregation
+-----+
| Name ID Maths Physics Chemistry Biology
| 0 Sona 1 95 42 88 92
| 1 Mona 8 100 72 60 89
| 2 Chinnu 7 67 83 52 77
| 3 Bannu 1 34 19 56 45
| 4 Sunny 7 33 66 77 76
|       Name Maths Physics Chemistry Biology
| ID
| 1 SonaBannu 129 61 136 137
| 7 ChinnuSunny 100 149 129 153
| 8 Mona 100 72 60 89
|       ID Maths Physics Chemistry Biology
| Name
| Bannu 1.0 34.0 19.0 56.0 45.0
| Chinnu 7.0 67.0 83.0 52.0 77.0
| Mona 8.0 100.0 72.0 60.0 89.0
| Sona 1.0 95.0 42.0 88.0 92.0
| Sunny 7.0 33.0 66.0 77.0 76.0
|       Name Maths Physics Chemistry Biology
| ID
| 1 SonaBannu 129 61 136 137
| 7 ChinnuSunny 100 149 129 153
| 8 Mona 100 72 60 89
|       Name Maths Physics Chemistry Biology
| ID
| 1 Bannu 34 19 56 45
| 7 Chinnu 33 66 52 76
| 8 Mona 100 72 60 89
|       ID Maths Physics Chemistry Biology
| Name
| Bannu 1 34 19 56 45
| Chinnu 7 67 83 52 77
| Mona 8 100 72 60 89
| Sona 1 95 42 88 92
| Sunny 7 33 66 77 76
|       Name Maths Physics Chemistry Biology
| ID
| 1 Sona 95 42 88 92
| 7 Sunny 67 83 77 77
| 8 Mona 100 72 60 89
| Process finished with exit code 0
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load/loads/dumps/fn.py - pandas_aggregation.py
DEPython | pandas_aggregation.py | pandas_joins.py | pandas_aggregation.py | pandas_getcountbystatus.py | Run: pandas_aggregation | Event Log | Search
Run: pandas_aggregation
+-----+
| Name ID Maths Physics Chemistry Biology
| 0 Sona 1 95 42 88 92
| 1 Mona 8 100 72 60 89
| 2 Chinnu 7 67 83 52 77
| 3 Bannu 1 34 19 56 45
| 4 Sunny 7 33 66 77 76
|       Name Maths Physics Chemistry Biology
| ID
| 1 SonaBannu 129 61 136 137
| 7 ChinnuSunny 100 149 129 153
| 8 Mona 100 72 60 89
|       ID Maths Physics Chemistry Biology
| Name
| Bannu 1.0 34.0 19.0 56.0 45.0
| Chinnu 7.0 67.0 83.0 52.0 77.0
| Mona 8.0 100.0 72.0 60.0 89.0
| Sona 1.0 95.0 42.0 88.0 92.0
| Sunny 7.0 33.0 66.0 77.0 76.0
|       Name Maths Physics Chemistry Biology
| ID
| 1 SonaBannu 129 61 136 137
| 7 ChinnuSunny 100 149 129 153
| 8 Mona 100 72 60 89
|       Name Maths Physics Chemistry Biology
| ID
| 1 Bannu 34 19 56 45
| 7 Chinnu 33 66 52 76
| 8 Mona 100 72 60 89
|       ID Maths Physics Chemistry Biology
| Name
| Bannu 1 34 19 56 45
| Chinnu 7 67 83 52 77
| Mona 8 100 72 60 89
| Sona 1 95 42 88 92
| Sunny 7 33 66 77 76
|       Name Maths Physics Chemistry Biology
| ID
| 1 Sona 95 42 88 92
| 7 Sunny 67 83 77 77
| 8 Mona 100 72 60 89
| Process finished with exit code 0
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load/loads/dumps/fn.py - pandas_aggregation.py
DEPython | pandas_aggregation.py | pandas_joins.py | pandas_aggregation.py | pandas_getcountbystatus.py | Run: pandas_aggregation | Event Log | Search
Run: pandas_aggregation
+-----+
| nested_selection.py
| nested_join.py
| advanced_indexing.py
| Number_functions.py
| Operators.py
| pandas_groupby.py
| pandas_query.py
| pandas_filtering.py
| pandas_getcountbymonth_status.py
| pandas_getcountbystatus.py
| pandas_pivot.py
| pandas_melt.py
| pandas_wide_to_long.py
| pandas_simplegfm.py
| pass.py
| Polymorphism.py
| Grouping python lists.py
| Read data from CSV file.py
| ReadWriteintoCSVfile.py
| Set & Del methods.py
| Append clear discard.py
| Sort python list using key.py
| Sorting .ISOR.py
| print(df.agg(['min','max','sum']))
| Process finished with exit code 0
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load/loads/dumps/fn.py - pandas_aggregation.py
DEPython | pandas_aggregation.py | pandas_joins.py | pandas_aggregation.py | pandas_getcountbystatus.py | Run: pandas_aggregation | Event Log | Search
Run: pandas_aggregation
+-----+
|      Name ID Maths Physics Chemistry Biology
| 0  Sunny 67 83 77 77 99
| 1  Mona 100 72 60 60 89
| 2 Chinnu 7 67 83 52 77
| 3  Bannu 1 33 19 52 45
| 4  Sona 1 95 42 88 92
|      Name ID Maths Physics Chemistry Biology
| min
| 0  Bannu 1 33 19 52 45
| 1  Sunny 7 33 66 77 76
| 2 Chinnu 7 67 83 52 77
| 3  Sona 1 95 42 88 92
| 4  Mona 8 100 72 60 89
|      Name ID Maths Physics Chemistry Biology
| max
| 0  Sona 1 95 42 88 92
| 1  Mona 8 100 72 60 89
| 2 Chinnu 7 67 83 52 77
| 3  Bannu 1 33 19 52 45
| 4  Sunny 7 33 66 77 76
|      Name ID Maths Physics Chemistry Biology
| sum
| 0  SonaMonaChinnuBannuSunny 24 329 282 325 379
| Process finished with exit code 0
File Edit View Navigate Code Refactor Run Tools VCS Window Help JSON load/loads/dumps/fn.py - pandas_aggregation.py
DEPython | pandas_aggregation.py | pandas_joins.py | pandas_aggregation.py | pandas_getcountbystatus.py | Run: pandas_aggregation | Event Log | Search
Run: pandas_aggregation
+-----+
| print(df.agg(['sum','min','max','mean','median','prod']))
```

## Writing pandas data frames to JSON files

`to_json()` function is used to convert a DataFrame to a JSON string. This function provides several parameters to customize the output JSON format.

1. **path\_or\_buf**: Specifies the file path or a writable buffer where the JSON string will be written. If not provided, the JSON string will be returned as a string.
2. **orient**: Specifies the orientation of the JSON string. Possible values are:
  - `'split'`: Dict like {index -> [index], columns -> [columns], data -> [values]} The JSON output is split into three parts: index, columns, and data
  - `'records'`: List like [{column -> value}, ..., {column -> value}]
  - `'index'`: Dict like {index -> {column -> value}}
  - `'columns'`: Dict like {column -> {index -> value}}
  - `'values'`: Just the values array
3. **date\_format**: Specifies the date format to use for datetime-like objects. The default is 'iso'.
4. **double\_precision**: Specifies the precision for floating point numbers. The default is 10.
5. **force\_ascii**: If True, all non-ASCII characters in the JSON string will be escaped. The default is True.
6. **date\_unit**: Specifies the time unit to encode dates. The default is 'ms' (milliseconds since epoch).
7. **lines**: If True, the output will be a JSON string per line. The default is False.
8. **compression**: Specifies the compression mode. Possible values are {'infer', 'gzip', 'bz2', 'xz', None}. The default is 'infer'( compresses based on extension of output file)

```

import pandas as pd
# Example-1
data=[[{"a":1,"b":2,"c":3}, {"d":4,"e":5,"f":6}, {"g":7,"h":8,"i":9}]]
df=pd.DataFrame(data,columns=['col1','col2','col3'],index=['row1','row2','row3'])
print(df)
print(type(df))
# converting to json
df.to_json('x.json',orient='split',compression='infer',index=True)
# Reading from that json file
df=pd.read_json('x.json')
print(df)
print(df)
print(df)
print(".....")

```

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists several Python files. The main editor window displays the code for reading and writing JSON files using the pandas library. Below the editor is a terminal window showing the output of the run command, which prints the DataFrame and its type. A status bar at the bottom provides system information like battery level (0.67%), signal strength, and date/time.

## Sort data in Pandas Data Frames

### Sort all rows of DataFrame by a column

In Pandas, the DataFrame provides a method `sort_values()`, and it sorts the DataFrame by values along the given axis.

```

df=pd.DataFrame(employees,columns=['ID','Name','Age','City','Experience'],index=[1,2,3,4,5,6,7,8,9])
print(df)
#sort all rows in a data frame
print("Sort by experience")
df=df.sort_values(by=['Experience'])
print(df)
print("Sort by City:")
print(df.sort_values(by=['City']))
print("Sort by Age and City")
print(df.sort_values(by=['Age','City']))

```

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists several Python files. The main editor window displays the code for sorting DataFrame rows by a specific column. Below the editor is a terminal window showing the output of the run command, which prints the sorted DataFrame. A status bar at the bottom provides system information like battery level (0.67%), signal strength, and date/time.

```
E 15 Veena 43 Delhi 14
F 16 Shanak 42 Mumbai 10
G 17 Shaun 40 Colombo 12
Sort by experience
ID Name Age City Experience
G 17 Shauna 42 Mumbai 10
C 15 Aadi 46 New York 11
G 17 Shaun 40 Colombo 12
E 15 Veena 43 Delhi 14
D 14 Mohit 45 Delhi 15
B 12 Riti 41 Delhi 17
A 11 Jack 44 Sydney 19
Sort by City:
ID Name Age City Experience
G 17 Shauna 40 Colombo 12
E 15 Veena 43 Delhi 14
D 14 Mohit 45 Delhi 15
B 12 Riti 41 Delhi 17
F 16 Shanak 42 Mumbai 10
C 15 Aadi 46 New York 11
A 11 Jack 44 Sydney 19
Sort by Age and City:
ID Name Age City Experience
G 17 Shauna 40 Colombo 12
B 12 Riti 41 Delhi 17
F 16 Shanak 42 Mumbai 10
E 15 Veena 43 Delhi 14
A 11 Jack 44 Sydney 19
D 14 Mohit 45 Delhi 15
C 15 Aadi 46 New York 11
```

It sorted the DataFrame along the ‘index’ axis, i.e., sorted all the rows along the column ‘Experience’.

## Sort all rows of DataFrame by a column in Descending Order

To sort the DataFrame in descending order, pass the argument ascending=False in the sort\_values() function.

```
#sort all rows in a data frame
df=df.sort_values(by=['Experience'])
print(df)
print("Sort by City:")
print(df.sort_values(by=['City']))
print("Sort by Age and City")
print(df.sort_values(by=['Age','City']))
# Sort by rows in descending order
print("Descending order")
print(df.sort_values(by=['Name'],ascending=False))
```

## Sort DataFrame by row index labels

In Pandas, the DataFrame provides a method `sort_index()`, and it sorts the DataFrame by index labels along the given axis. By default, it sorts the rows of DataFrame based on row index labels.

### Sort DataFrame by column names

Pass the `axis=1` argument in the `sort_index()` method of DataFrame. It will sort the DataFrame by the column names.

So Age comes first followed by City followed by Experience so on.

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code to sort a DataFrame. The code includes comments explaining the sorting logic. The output cell displays the sorted DataFrame with columns: Age, City, Experience, ID, and Name. The data rows are sorted by ID in ascending order, then by Name in ascending order, then by City in ascending order, then by Experience in ascending order, and finally by Age in descending order.

```
# Sort DataFrame by column names
    Age   City  Experience  ID   Name
F  42   Mumbai        10  16 Shanak
C  46  New York      11  13 Aadi
G  40   Colombo       12  17 Shaun
E  43   Delhi         14  15 Veeva
D  45   Delhi         15  14 Mohit
B  41   Delhi         17  12 Riti
A  44   Sydney        19  11 Jack

#Sort DataFrame by row index labels
print("Sort by Row Index")
print(df.sort_index())
#Sort DataFrame by column names
print("Sort DataFrame by column names")
print(df.sort_index(axis=1))
```

Process finished with exit code 0

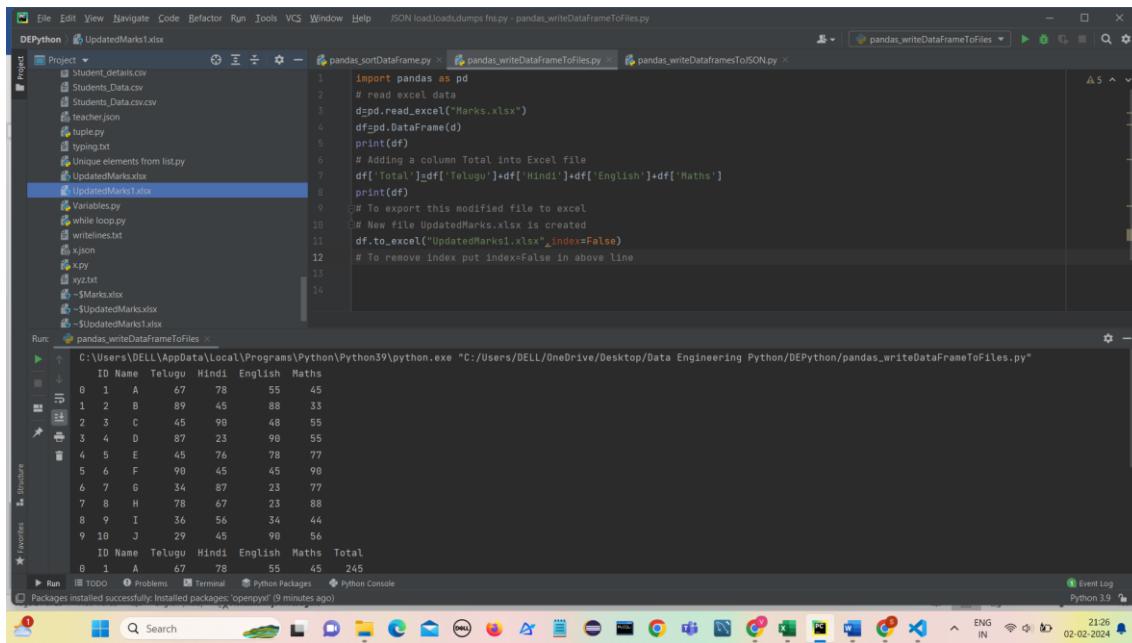
The screenshot shows a Jupyter Notebook interface with a code cell containing Python code to sort a DataFrame. The code includes comments explaining the sorting logic. The output cell displays two sorted DataFrames. The first is sorted by Row Index (ID), showing rows A through G. The second is sorted by column names, showing columns Age, City, Experience, ID, and Name.

```
#Sort DataFrame by row index labels
print("Sort by Row Index")
print(df.sort_index())
#Sort DataFrame by column names
print("Sort DataFrame by column names")
print(df.sort_index(axis=1))
```

Process finished with exit code 0

# Writing Pandas DataFrames to files

## 1) Excel file



The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists several files: student\_details.csv, Students\_Data.csv, Students\_Data.csv.csv, teacher.json, tuple.py, typing.txt, Unique elements from list.py, UpdatedMarks.xlsx, UpdatedMarks1.xlsx, Variables.py, while loop.py, writelines.txt, k.json, k.py, xyz.txt, \$Marks.xlsx, \$UpdatedMarks.xlsx, and \$UpdatedMarks1.xlsx. Below the navigation bar, there are two tabs open: 'pandas\_sortDataFrame.py' and 'pandas\_writeDataFrameToFiles.py'. The 'pandas\_writeDataFrameToFiles.py' tab contains the following Python code:

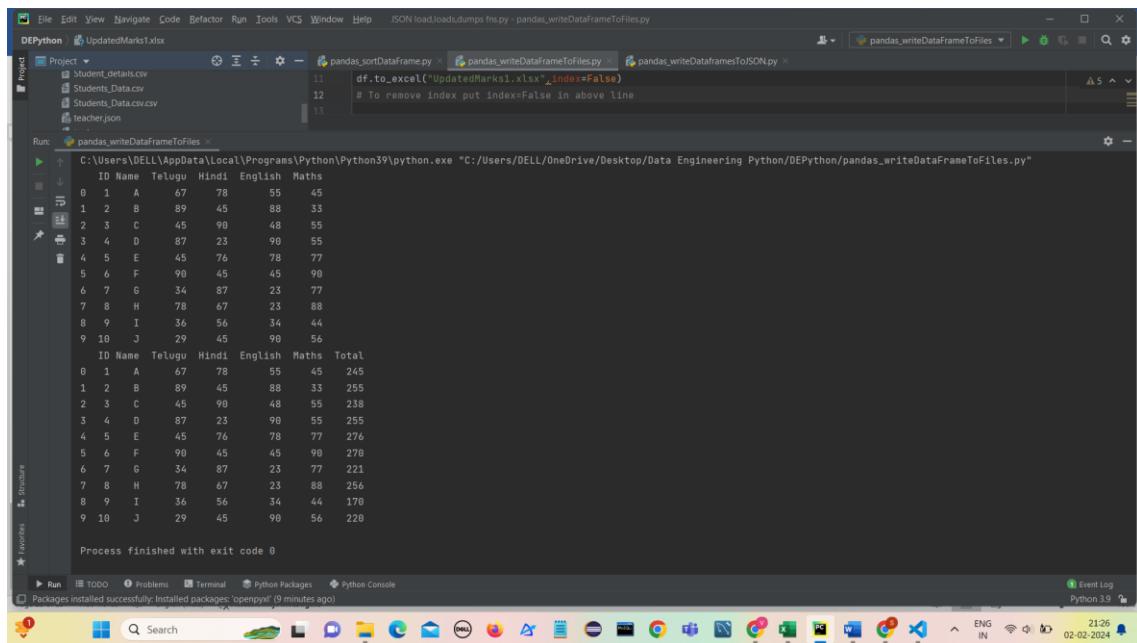
```
import pandas as pd
# read excel data
dfpd.read_excel("Marks.xlsx")
df=pd.DataFrame(d)
print(df)

# Adding a column Total into Excel file
df['Total']=df['Telugu']+df['Hindi']+df['English']+df['Maths']
print(df)

# To export this modified file to excel
# New file UpdatedMarks.xlsx is created
df.to_excel("UpdatedMarks1.xlsx",index=False)

# To remove index put index=False in above line
```

The 'Run' tab at the bottom shows the command: C:/Users/DELL/AppData/Local/Programs/Python/Python39/python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/pandas\_writeDataFrameToFiles.py". The output window displays the DataFrame data and the command prompt.



The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists several files: Student\_details.csv, Students\_Data.csv, Students\_Data.csv.csv, teacher.json, tuple.py, typing.txt, Unique elements from list.py, UpdatedMarks.xlsx, UpdatedMarks1.xlsx, Variables.py, while loop.py, writelines.txt, k.json, k.py, xyz.txt, \$Marks.xlsx, \$UpdatedMarks.xlsx, and \$UpdatedMarks1.xlsx. Below the navigation bar, there are two tabs open: 'pandas\_sortDataFrame.py' and 'pandas\_writeDataFrameToFiles.py'. The 'pandas\_writeDataFrameToFiles.py' tab contains the following Python code:

```
df.to_excel("UpdatedMarks1.xlsx",index=False)

# To remove index put index=False in above line
```

The 'Run' tab at the bottom shows the command: C:/Users/DELL/AppData/Local/Programs/Python/Python39/python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/pandas\_writeDataFrameToFiles.py". The output window displays the DataFrame data and the command prompt.

## 2) CSV file

The screenshot shows the PyCharm IDE interface. The project navigation bar at the top lists several Python files. In the center, a code editor window displays a script named `pandas_writeDataFrameToFile.py`. The code reads a CSV file named `UpdatedMarks.csv` and prints its contents. It then creates a new DataFrame, adds some data, and writes it back to a file named `UpdatedMarks.csv`. Below the code editor, a terminal window shows the output of the script's execution. The system tray at the bottom right indicates the date as 02-02-2024.

```
# Write to csv File
df.to_csv('UpdatedMarks.csv', index=False)
# reading from CSV file
print("Data from CSV file")
x=pd.read_csv("UpdatedMarks.csv")
print(x)
```

## 3) Text file

This screenshot shows the PyCharm IDE again. The project navigation bar includes the same set of Python files. The code editor now contains a script named `pandas_writeDataFrameToFile.py`. The code reads a CSV file, converts it to a text file using commas as separators, and then prints the data. The terminal window shows the successful execution of the script. The system tray at the bottom right shows the date as 02-02-2024.

```
# Write to txt file
print("Writing to txt file")
df.to_csv('UpdatedMarks.txt', sep=',', index=False)
print("Data from txt file")
x=pd.read_csv('UpdatedMarks.txt')
print(x)
```

## 4) Html file

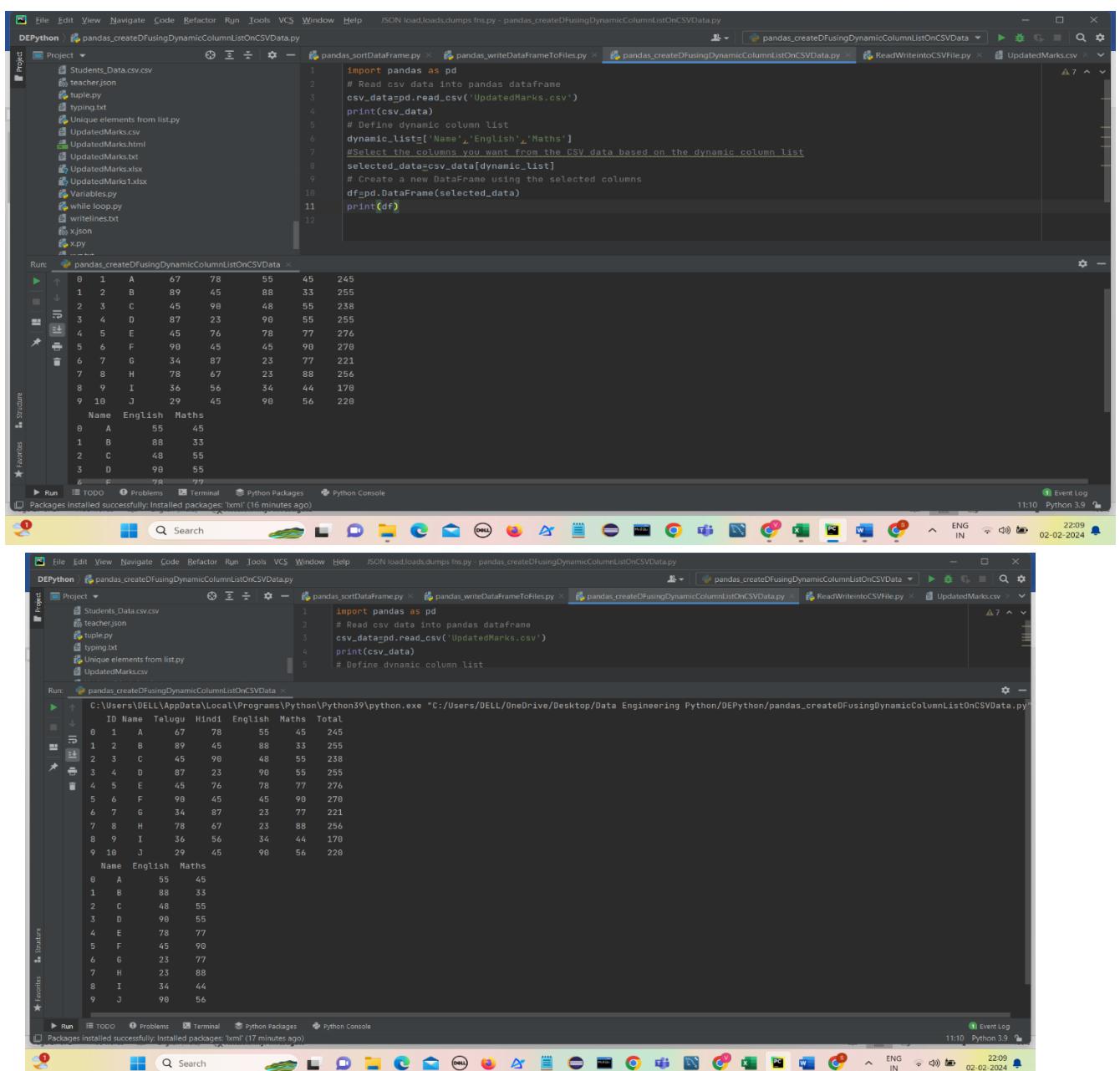
The final screenshot shows the PyCharm IDE once more. The project navigation bar remains consistent. The code editor contains a script named `pandas_writeDataFrameToFile.py`. This script reads a CSV file and converts it into an HTML table. The terminal window shows the completion of the script with an exit code of 0. The system tray at the bottom right shows the date as 02-02-2024.

```
# Write to html file
print("Writing to html file")
df.to_html('UpdatedMarks.html', index=False)
print("Data from HTML file")
y=pd.read_html('UpdatedMarks.html')
print(y)
```

# Create Data Frames using dynamic column list on csv data

- Read the CSV data into a pandas DataFrame
- Define your dynamic column list
- Select the columns you want from the CSV data based on the dynamic column list
- Create a new DataFrame using the selected columns
- Display the new DataFrame

## Example-1



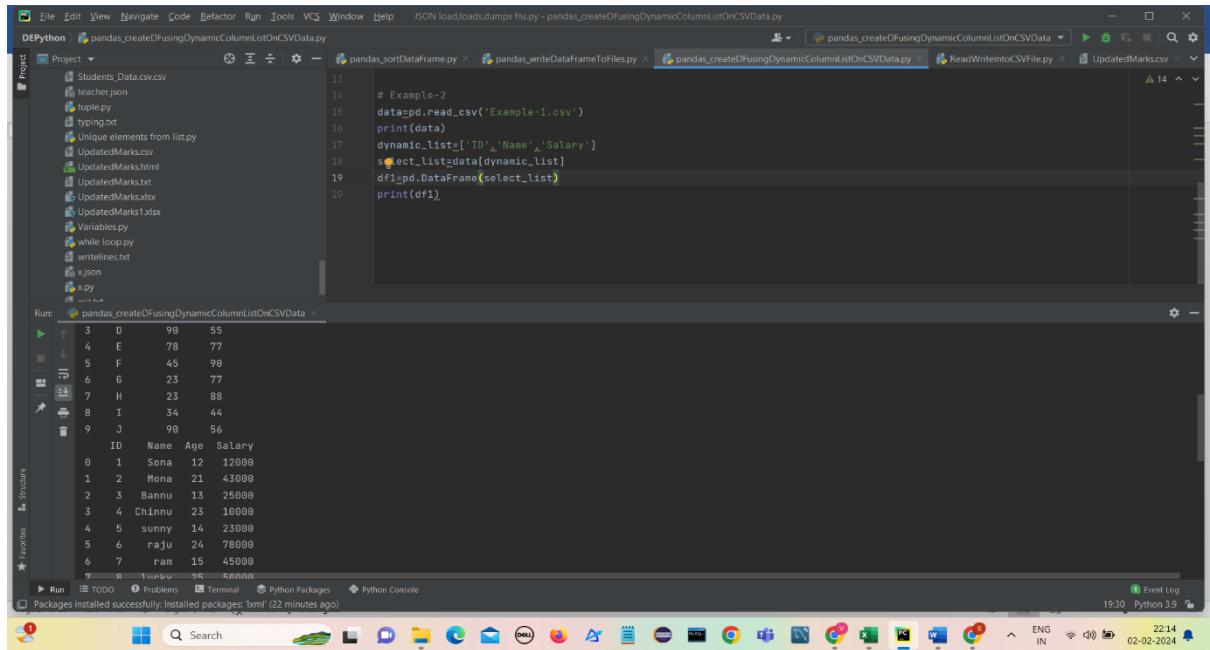
```
DEPython > pandas_createDFusingDynamicColumnListOnCSVData.py
```

```
1 import pandas as pd
2 # Read csv data into pandas dataframe
3 csv_data=pd.read_csv('UpdatedMarks.csv')
4 print(csv_data)
5 # Define dynamic column list
6 dynamic_list=['Name','English','Maths']
7 #Select the columns you want from the CSV data based on the dynamic column list
8 selected_data=csv_data[dynamic_list]
9 # Create a new DataFrame using the selected columns
10 df=pd.DataFrame(selected_data)
11 print(df)
```

ID	Name	Telugu	Hindi	English	Maths	Total
0	A	67	78	55	45	245
1	B	89	45	88	33	255
2	C	45	98	48	55	238
3	D	87	23	90	55	255
4	E	45	76	78	77	276
5	F	90	45	45	90	270
6	G	34	87	23	77	221
7	H	78	67	23	88	256
8	I	36	56	34	44	170
9	J	29	45	90	56	220

Name	English	Maths
A	55	45
B	88	33
C	48	55
D	90	55
E	78	77
F	45	90
G	34	87
H	23	88
I	36	44
J	29	56

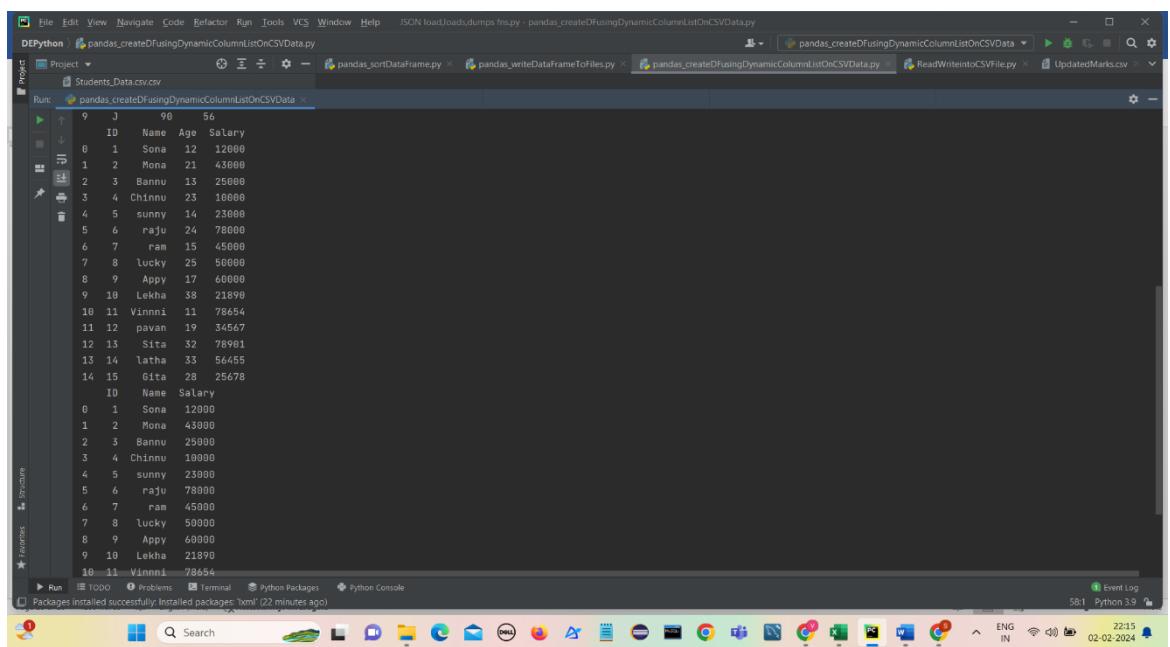
## Example-2



The screenshot shows the PyCharm IDE interface. The code editor displays a Python script named `pandas_createDFusingDynamicColumnListOnCSVData.py`. The script reads a CSV file, prints its contents, and then creates a DataFrame using a dynamic list of columns. The run output window shows the original CSV data and the resulting DataFrame.

```
# Example-2
data=pd.read_csv('Example-1.csv')
print(data)
dynamic_list=['ID','Name','Salary']
select_list=data[dynamic_list]
df=pd.DataFrame(select_list)
print(df)
```

ID	Name	Age	Salary
0	Sona	12	12000
1	Mona	21	43000
2	Bannu	13	25000
3	Chinnu	23	10000
4	sunny	14	23000
5	raju	24	78000
6	ram	15	45000
7	lucky	25	50000
8	Appy	17	60000
9	Lekha	38	21890
10	Name	Salary	
0	Sona	12000	
1	Mona	43000	
2	Bannu	25000	
3	Chinnu	10000	
4	sunny	23000	
5	raju	78000	
6	ram	45000	
7	lucky	50000	
8	Appy	60000	
9	Lekha	21890	
10-11	Vinnni	78654	



The screenshot shows the PyCharm IDE interface. The code editor displays a Python script named `pandas_createDFusingDynamicColumnListOnCSVData.py`. The script reads a CSV file, prints its contents, and then creates a DataFrame using a dynamic list of columns. The run output window shows the original CSV data and the resulting DataFrame.

```
# Example-2
data=pd.read_csv('Example-1.csv')
print(data)
dynamic_list=['ID','Name','Salary']
select_list=data[dynamic_list]
df=pd.DataFrame(select_list)
print(df)
```

ID	Name	Age	Salary
0	Sona	12	12000
1	Mona	21	43000
2	Bannu	13	25000
3	Chinnu	23	10000
4	sunny	14	23000
5	raju	24	78000
6	ram	15	45000
7	lucky	25	50000
8	Appy	17	60000
9	Lekha	38	21890
10	Name	Salary	
0	Sona	12000	
1	Mona	43000	
2	Bannu	25000	
3	Chinnu	10000	
4	sunny	23000	
5	raju	78000	
6	ram	45000	
7	lucky	50000	
8	Appy	60000	
9	Lekha	21890	
10-11	Vinnni	78654	