Q1. How do you load a CSV file into a Pandas DataFrame?

Ans:- import pandas as pd

df = pd.read\_csv ('file\_name.csv')  
print(df)

Example:-

*#reading data in pandas*

*Import pandas as pd*

df = pd.read\_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

df

Q2. How do you check the data type of a column in a Pandas DataFrame?

Ans:- To check the data type in pandas DataFrame we can use the “dtype” attribute. The attribute returns a series with the data type of each column. And the column names of the DataFrame are represented as the index of the resultant series object and the corresponding data types are returned as values of the series object. If any column has mixed data types are stored then the data type of the entire column is indicated as object dtype.

Example 1

Apply the pandas dtype property and verify the data type of each in the DataFrame object.

# importing pandas package

import pandas as pd

# create a Pandas DataFrame

df = pd.DataFrame({'Col1':[4.1, 23.43], 'Col2':['a', 'w'], 'Col3':[1, 8]})

print("DataFrame:")

print(df)

# apply the dtype attribute

result = df.dtypes

print("Output:")

print(result)

## Output

The output is mentioned below −

DataFrame:

      Col1 Col2 Col3

0     4.10    a    1

1    23.43    w    8

Output:

Col1    float64

Col2     object

Col3      int64

dtype: object

In this output block, we can notice that Col1 has float64 type data, Col2 has objective data and column “Col3” has stored the integer type data.

Example2

import pandas as pd

df = pd.read\_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

df

Output:-

PassengerId Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked

0 1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S

1 2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C

2 3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S

3 4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S

4 5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

... ... ... ... ... ... ... ... ... ... ... ... ...

886 887 0 2 Montvila, Rev. Juozas male 27.0 0 0 211536 13.0000 NaN S

887 888 1 1 Graham, Miss. Margaret Edith female 19.0 0 0 112053 30.0000 B42 S

888 889 0 3 Johnston, Miss. Catherine Helen "Carrie" female NaN 1 2 W./C. 6607 23.4500 NaN S

889 890 1 1 Behr, Mr. Karl Howell male 26.0 0 0 111369 30.0000 C148 C

890 891 0 3 Dooley, Mr. Patrick male 32.0 0 0 370376 7.7500 NaN Q

*# checking the datatypes of the columns*

df.dtypes

Output:-

PassengerId int64

Survived int64

Pclass int64

Name object

Sex object

Age float64

SibSp int64

Parch int64

Ticket object

Fare float64

Cabin object

Embarked object

dtype: object

Q3. How do you select rows from a Pandas DataFrame based on a condition?

Ans:- import pandas as pd

df = pd.read\_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

df

Output:-

PassengerId Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked

0 1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S

1 2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C

2 3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S

3 4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S

4 5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

... ... ... ... ... ... ... ... ... ... ... ... ...

886 887 0 2 Montvila, Rev. Juozas male 27.0 0 0 211536 13.0000 NaN S

887 888 1 1 Graham, Miss. Margaret Edith female 19.0 0 0 112053 30.0000 B42 S

888 889 0 3 Johnston, Miss. Catherine Helen "Carrie" female NaN 1 2 W./C. 6607 23.4500 NaN S

889 890 1 1 Behr, Mr. Karl Howell male 26.0 0 0 111369 30.0000 C148 C

890 891 0 3 Dooley, Mr. Patrick male 32.0 0 0 370376 7.7500 NaN Q

*#Selecting a row using index*

df.loc[100]

Out[21]:

PassengerId 101

Survived 0

Pclass 3

Name Petranec, Miss. Matilda

Sex female

Age 28.0

SibSp 0

Parch 0

Ticket 349245

Fare 7.8958

Cabin NaN

Embarked S

Name: 100, dtype: object

*#Selecting a row using default index*

df.iloc[100]

Out[22]:

PassengerId 101

Survived 0

Pclass 3

Name Petranec, Miss. Matilda

Sex female

Age 28.0

SibSp 0

Parch 0

Ticket 349245

Fare 7.8958

Cabin NaN

Embarked S

Name: 100, dtype: object

# Selecting rows in pandas DataFrame based on conditions

Let’s see how to Select rows based on some conditions in Pandas DataFrame.

### Selecting rows based on particular column value using '>', '=', '=', '<=', '!=' operator.

**Example:-**

**Code #1 :** Selecting all the rows from the given dataframe in which ‘Percentage’ is greater than 80 using basic method.

# importing pandas

import pandas as pd

record = {

 'Name': ['Ankit', 'Amit', 'Aishwarya', 'Priyanka', 'Priya', 'Shaurya' ],

 'Age': [21, 19, 20, 18, 17, 21],

 'Stream': ['Math', 'Commerce', 'Science', 'Math', 'Math', 'Science'],

 'Percentage': [88, 92, 95, 70, 65, 78] }

# create a dataframe

dataframe = pd.DataFrame(record, columns = ['Name', 'Age', 'Stream', 'Percentage'])

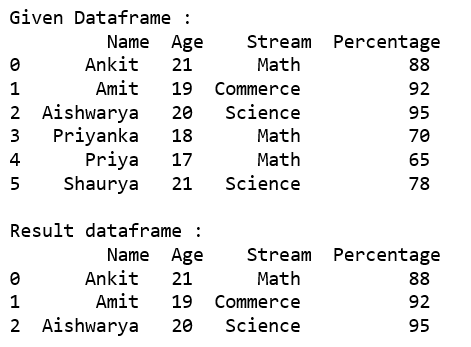
print("Given Dataframe :\n", dataframe)

# selecting rows based on condition

rslt\_df = dataframe[dataframe['Percentage'] > 80]

print('\nResult dataframe :\n', rslt\_df)

**Output :**



Q4. How do you rename columns in a Pandas DataFrame?

## Ans:- ****Method 1:****Using rename() function

One way of renaming the columns in a Pandas Dataframe is by using the rename() function. This method is quite useful when we need to rename some selected columns because we need to specify information only for the columns which are to be renamed.

**Example 1:**Rename a **single column**.

# Import pandas package

import pandas as pd

# Define a dictionary containing ICC rankings

rankings = {'test': ['India', 'South Africa', 'England',

'New Zealand', 'Australia'],

'odi': ['England', 'India', 'New Zealand',

'South Africa', 'Pakistan'],

't20': ['Pakistan', 'India', 'Australia',

'England', 'New Zealand']}

# Convert the dictionary into DataFrame

rankings\_pd = pd.DataFrame(rankings)

# Before renaming the columns

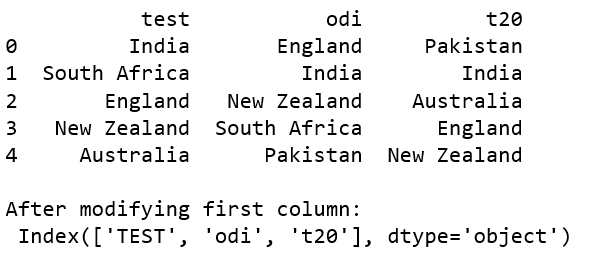
print(rankings\_pd)

rankings\_pd.rename(columns = {'test':'TEST'}, inplace = True)

# After renaming the columns

print("\nAfter modifying first column:\n", rankings\_pd.columns)

**Output:**



**Example 2:**Rename **multiple columns**.

# Import pandas package

import pandas as pd

# Define a dictionary containing ICC rankings

rankings = {'test': ['India', 'South Africa', 'England',

'New Zealand', 'Australia'],

'odi': ['England', 'India', 'New Zealand',

'South Africa', 'Pakistan'],

't20': ['Pakistan', 'India', 'Australia',

'England', 'New Zealand']}

# Convert the dictionary into DataFrame

rankings\_pd = pd.DataFrame(rankings)

# Before renaming the columns

print(rankings\_pd.columns)

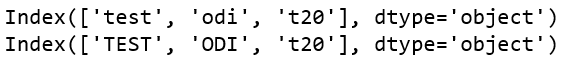
rankings\_pd.rename(columns = {'test':'TEST', 'odi':'ODI',

't20':'T20'}, inplace = True)

# After renaming the columns

print(rankings\_pd.columns)

**Output:**



### ****Method 2:****By assigning a list of new column names

The columns can also be renamed by directly assigning a list containing the new names to the columns attribute of the Dataframe object for which we want to rename the columns. The disadvantage of this method is that we need to provide new names for all the columns even if want to rename only some of the columns

Example-1.

# Import pandas package

import pandas as pd

# Define a dictionary containing ICC rankings

rankings = {'test': ['India', 'South Africa', 'England',

'New Zealand', 'Australia'],

'odi': ['England', 'India', 'New Zealand',

'South Africa', 'Pakistan'],

't20': ['Pakistan', 'India', 'Australia',

'England', 'New Zealand']}

# Convert the dictionary into DataFrame

rankings\_pd = pd.DataFrame(rankings)

# Before renaming the columns

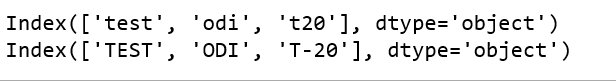
print(rankings\_pd.columns)

rankings\_pd.columns = ['TEST', 'ODI', 'T-20']

# After renaming the columns

print(rankings\_pd.columns)

**Output:**



### Method 3: Rename column names using DataFrame set\_axis() function

In this example, we will rename the column name using the set\_axis function, we will pass the new column name and axis that should be replaced with a new name in the column as a parameter.

Example:-

# Import pandas package

import pandas as pd

# Define a dictionary containing ICC rankings

rankings = {'test': ['India', 'South Africa', 'England',

'New Zealand', 'Australia'],

'odi': ['England', 'India', 'New Zealand',

'South Africa', 'Pakistan'],

't20': ['Pakistan', 'India', 'Australia',

'England', 'New Zealand']}

# Convert the dictionary into DataFrame

rankings\_pd = pd.DataFrame(rankings)

# Before renaming the columns

print(rankings\_pd.columns)

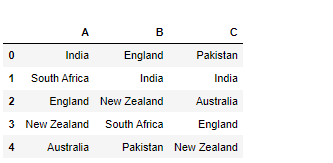
rankings\_pd.set\_axis(['A', 'B', 'C'], axis='columns', inplace=True)

# After renaming the columns

print(rankings\_pd.columns)

rankings\_pd.head()

**Output:**



### Method 4: Rename column names using DataFrame add\_prefix() and add\_suffix() functions

In this example, we will rename the column name using the add\_Sufix and add\_Prefix function, we will pass the prefix and suffix that should be added to the first and last name of the column name.

Example:-

# Import pandas package

import pandas as pd

# Define a dictionary containing ICC rankings

rankings = {'test': ['India', 'South Africa', 'England',

'New Zealand', 'Australia'],

'odi': ['England', 'India', 'New Zealand',

'South Africa', 'Pakistan'],

't20': ['Pakistan', 'India', 'Australia',

'England', 'New Zealand']}

# Convert the dictionary into DataFrame

rankings\_pd = pd.DataFrame(rankings)

# Before renaming the columns

print(rankings\_pd.columns)

rankings\_pd = rankings\_pd.add\_prefix('col\_')

rankings\_pd = rankings\_pd.add\_suffix('\_1')

# After renaming the columns

rankings\_pd.head()

**Output:**

col\_test\_1 col\_odi\_1 col\_t20\_1

0 India England Pakistan

1 South Africa India India

2 England New Zealand Australia

3 New Zealand South Africa England

4 Australia Pakistan New Zealand

### Method 5: Replace specific texts of column names using Dataframe.columns.str.replace function

In this example, we will rename the column name using the replace function, we will pass the old name with the new name as a parameter for the column.

# Import pandas package

import pandas as pd

# Define a dictionary containing ICC rankings

rankings = {'test': ['India', 'South Africa', 'England',

'New Zealand', 'Australia'],

'odi': ['England', 'India', 'New Zealand',

'South Africa', 'Pakistan'],

't20': ['Pakistan', 'India', 'Australia',

'England', 'New Zealand']}

# Convert the dictionary into DataFrame

rankings\_pd = pd.DataFrame(rankings)

# Before renaming the columns

print(rankings\_pd.columns)

# df = rankings\_pd

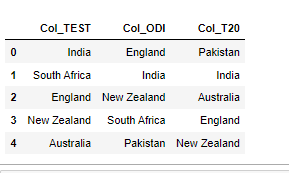
rankings\_pd.columns = rankings\_pd.columns.str.replace('test', 'Col\_TEST')

rankings\_pd.columns = rankings\_pd.columns.str.replace('odi', 'Col\_ODI')

rankings\_pd.columns = rankings\_pd.columns.str.replace('t20', 'Col\_T20')

rankings\_pd.head()

**Output:**



Q5. How do you drop columns in a Pandas DataFrame?

Ans:- # creating new column

df['new\_col'] = df['Survived'] + df['Pclass']

df.head()

Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked new\_col

PassengerId

1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S 3

2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C 2

3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S 4

4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S 2

5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S 3

#deleting the column

df.drop('new\_col', axis=1)

Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked

PassengerId

1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S

2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C

3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S

4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S

5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

... ... ... ... ... ... ... ... ... ... ... ...

887 0 2 Montvila, Rev. Juozas male 27.0 0 0 211536 13.0000 NaN S

888 1 1 Graham, Miss. Margaret Edith female 19.0 0 0 112053 30.0000 B42 S

889 0 3 Johnston, Miss. Catherine Helen "Carrie" female NaN 1 2 W./C. 6607 23.4500 NaN S

890 1 1 Behr, Mr. Karl Howell male 26.0 0 0 111369 30.0000 C148 C

891 0 3 Dooley, Mr. Patrick male 32.0 0 0 370376 7.7500 NaN Q

df

Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked new\_col

PassengerId

1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S 3

2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C 2

3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S 4

4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S 2

5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S 3

... ... ... ... ... ... ... ... ... ... ... ... ...

887 0 2 Montvila, Rev. Juozas male 27.0 0 0 211536 13.0000 NaN S 2

888 1 1 Graham, Miss. Margaret Edith female 19.0 0 0 112053 30.0000 B42 S 2

889 0 3 Johnston, Miss. Catherine Helen "Carrie" female NaN 1 2 W./C. 6607 23.4500 NaN S 3

890 1 1 Behr, Mr. Karl Howell male 26.0 0 0 111369 30.0000 C148 C 2

891 0 3 Dooley, Mr. Patrick male 32.0 0 0 370376 7.7500 NaN Q 3

#deleting the column

df.drop('new\_col', axis=1, inplace=True)

df

Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked

PassengerId

1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S

2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C

3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S

4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S

5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

... ... ... ... ... ... ... ... ... ... ... ...

887 0 2 Montvila, Rev. Juozas male 27.0 0 0 211536 13.0000 NaN S

888 1 1 Graham, Miss. Margaret Edith female 19.0 0 0 112053 30.0000 B42 S

889 0 3 Johnston, Miss. Catherine Helen "Carrie" female NaN 1 2 W./C. 6607 23.4500 NaN S

890 1 1 Behr, Mr. Karl Howell male 26.0 0 0 111369 30.0000 C148 C

891 0 3 Dooley, Mr. Patrick male 32.0 0 0 370376 7.7500 NaN Q

Q6. How do you find the unique values in a column of a Pandas DataFrame?

Ans:- import pandas as pd

df = pd.read\_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

df

Output:-

PassengerId Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked

0 1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S

1 2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C

2 3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S

3 4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S

4 5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

... ... ... ... ... ... ... ... ... ... ... ... ...

886 887 0 2 Montvila, Rev. Juozas male 27.0 0 0 211536 13.0000 NaN S

887 888 1 1 Graham, Miss. Margaret Edith female 19.0 0 0 112053 30.0000 B42 S

888 889 0 3 Johnston, Miss. Catherine Helen "Carrie" female NaN 1 2 W./C. 6607 23.4500 NaN S

889 890 1 1 Behr, Mr. Karl Howell male 26.0 0 0 111369 30.0000 C148 C

890 891 0 3 Dooley, Mr. Patrick male 32.0 0 0 370376 7.7500 NaN Q

df['Pclass'].unique()

array([3, 1, 2])

df['Embarked'].unique()

array(['S', 'C', 'Q', nan], dtype=object)

Q7. How do you find the number of missing values in each column of a Pandas DataFrame?

Ans:- import pandas as pd

df = pd.read\_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

df

Output:-

PassengerId Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked

0 1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S

1 2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C

2 3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S

3 4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S

4 5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

... ... ... ... ... ... ... ... ... ... ... ... ...

886 887 0 2 Montvila, Rev. Juozas male 27.0 0 0 211536 13.0000 NaN S

887 888 1 1 Graham, Miss. Margaret Edith female 19.0 0 0 112053 30.0000 B42 S

888 889 0 3 Johnston, Miss. Catherine Helen "Carrie" female NaN 1 2 W./C. 6607 23.4500 NaN S

889 890 1 1 Behr, Mr. Karl Howell male 26.0 0 0 111369 30.0000 C148 C

890 891 0 3 Dooley, Mr. Patrick male 32.0 0 0 370376 7.7500 NaN Q

*#checking missing values*

df.isnull()

Out[27]:

|  | **Survived** | **Pclass** | **Name** | **Sex** | **Age** | **SibSp** | **Parch** | **Ticket** | **Fare** | **Cabin** | **Embarked** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PassengerId** |  |  |  |  |  |  |  |  |  |  |  |
| **1** | False | False | False | False | False | False | False | False | False | True | False |
| **2** | False | False | False | False | False | False | False | False | False | False | False |
| **3** | False | False | False | False | False | False | False | False | False | True | False |
| **4** | False | False | False | False | False | False | False | False | False | False | False |
| **5** | False | False | False | False | False | False | False | False | False | True | False |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **887** | False | False | False | False | False | False | False | False | False | True | False |
| **888** | False | False | False | False | False | False | False | False | False | False | False |
| **889** | False | False | False | False | True | False | False | False | False | True | False |
| **890** | False | False | False | False | False | False | False | False | False | False | False |
| **891** | False | False | False | False | False | False | False | False | False | True | False |

*# checking the total number of missing values*

df.isnull().sum()

Survived 0

Pclass 0

Name 0

Sex 0

Age 177

SibSp 0

Parch 0

Ticket 0

Fare 0

Cabin 687

Embarked 2

dtype: int64

Q8. How do you fill missing values in a Pandas DataFrame with a specific value?

Ans:- Pandas DataFrame fillna() Method

The fillna() method replaces the NULL values with a specified value. The fillna() method returns a new DataFrame object unless the inplace parameter is set to True , in that case the fillna() method does the replacing in the original DataFrame instead.

Syntax

*dataframe*.fillna(value, method, axis, inplace, limit, downcast)

Example

Duration,Pulse,Maxpulse,Calories

60,110,130,409.1

60,117,145,479.0

60,103,135,340.0

45,109,175,282.4

45,117,148,406.0

60,102,127,300.5

60,110,136,374.0

45,104,134,253.3

30,109,133,195.1

60,98,124,269.0

60,103,147,329.3

60,100,120,250.7

60,106,128,345.3

60,104,132,379.3

60,98,123,275.0

60,98,120,215.2

60,100,120,300.0

45,90,112,

60,103,123,323.0

45,97,125,243.0

60,108,131,364.2

45,100,119,282.0

60,130,101,300.0

45,105,132,246.0

60,102,126,334.5

60,100,120,250.0

60,92,118,241.0

60,103,132

60,100,132,280.0

60,102,129,380.3

60,92,115,243.0

45,90,112,180.1

60,101,124,299.0

60,93,113,223.0

60,107,136,361.0

60,114,140,415.0

60,102,127,300.5

60,100,120,300.1

60,100,120,300.0

45,104,129,266.0

45,90,112,180.1

60,98,126,286.0

60,100,122,329.4

60,111,138,400.0

60,111,131,397.0

60,99,119,273.0

60,109,153,387.6

45,111,136,300.0

45,108,129,298.0

60,111,139,397.6

60,107,136,380.2

80,123,146,643.1

60,106,130,263.0

60,118,151,486.0

30,136,175,238.0

60,121,146,450.7

60,118,121,413.0

45,115,144,305.0

20,153,172,226.4

45,123,152,321.0

210,108,160,1376.0

160,110,137,1034.4

160,109,135,853.0

45,118,141,341.0

20,110,130,131.4

180,90,130,800.4

150,105,135,873.4

150,107,130,816.0

20,106,136,110.4

300,108,143,1500.2

150,97,129,1115.0

60,109,153,387.6

90,100,127,700.0

150,97,127,953.2

45,114,146,304.0

90,98,125,563.2

45,105,134,251.0

45,110,141,300.0

120,100,130,500.4

270,100,131,1729.0

30,159,182,319.2

45,149,169,344.0

30,103,139,151.1

120,100,130,500.0

45,100,120,225.3

30,151,170,300.1

45,102,136,234.0

120,100,157,1000.1

45,129,103,242.0

20,83,107,50.3

180,101,127,600.1

45,107,137,

30,90,107,105.3

15,80,100,50.5

20,150,171,127.4

20,151,168,229.4

30,95,128,128.2

25,152,168,244.2

30,109,131,188.2

90,93,124,604.1

20,95,112,77.7

90,90,110,500.0

90,90,100,500.0

90,90,100,500.4

30,92,108,92.7

30,93,128,124.0

180,90,120,800.3

30,90,120,86.2

90,90,120,500.3

210,137,184,1860.4

60,102,124,325.2

45,107,124,275.0

15,124,139,124.2

45,100,120,225.3

60,108,131,367.6

60,108,151,351.7

60,116,141,443.0

60,97,122,277.4

60,105,125,

60,103,124,332.7

30,112,137,193.9

45,100,120,100.7

60,119,169,336.7

60,107,127,344.9

60,111,151,368.5

60,98,122,271.0

60,97,124,275.3

60,109,127,382.0

90,99,125,466.4

60,114,151,384.0

60,104,134,342.5

60,107,138,357.5

60,103,133,335.0

60,106,132,327.5

60,103,136,339.0

20,136,156,189.0

45,117,143,317.7

45,115,137,318.0

45,113,138,308.0

20,141,162,222.4

60,108,135,390.0

60,97,127,

45,100,120,250.4

45,122,149,335.4

60,136,170,470.2

45,106,126,270.8

60,107,136,400.0

60,112,146,361.9

30,103,127,185.0

60,110,150,409.4

60,106,134,343.0

60,109,129,353.2

60,109,138,374.0

30,150,167,275.8

60,105,128,328.0

60,111,151,368.5

60,97,131,270.4

60,100,120,270.4

60,114,150,382.8

30,80,120,240.9

30,85,120,250.4

45,90,130,260.4

45,95,130,270.0

45,100,140,280.9

60,105,140,290.8

60,110,145,300.4

60,115,145,310.2

75,120,150,320.4

75,125,150,330.4

Example

import pandas as pd

df = pd.read\_csv('data.csv')

newdf = df.fillna(222222)

print(newdf.to\_string())

#Note that we use the to\_string() method to return the entire DataFrame.

Output:-

Duration Pulse Maxpulse Calories

0 60 110 130 409.1

1 60 117 145 479.0

2 60 103 135 340.0

3 45 109 175 282.4

4 45 117 148 406.0

5 60 102 127 300.5

6 60 110 136 374.0

7 45 104 134 253.3

8 30 109 133 195.1

9 60 98 124 269.0

10 60 103 147 329.3

11 60 100 120 250.7

12 60 106 128 345.3

13 60 104 132 379.3

14 60 98 123 275.0

15 60 98 120 215.2

16 60 100 120 300.0

17 45 90 112 222222.0

18 60 103 123 323.0

19 45 97 125 243.0

20 60 108 131 364.2

21 45 100 119 282.0

22 60 130 101 300.0

23 45 105 132 246.0

24 60 102 126 334.5

25 60 100 120 250.0

26 60 92 118 241.0

27 60 103 132 222222.0

28 60 100 132 280.0

29 60 102 129 380.3

30 60 92 115 243.0

31 45 90 112 180.1

32 60 101 124 299.0

33 60 93 113 223.0

34 60 107 136 361.0

35 60 114 140 415.0

36 60 102 127 300.5

37 60 100 120 300.1

38 60 100 120 300.0

39 45 104 129 266.0

40 45 90 112 180.1

41 60 98 126 286.0

42 60 100 122 329.4

43 60 111 138 400.0

44 60 111 131 397.0

45 60 99 119 273.0

46 60 109 153 387.6

47 45 111 136 300.0

48 45 108 129 298.0

49 60 111 139 397.6

50 60 107 136 380.2

51 80 123 146 643.1

52 60 106 130 263.0

53 60 118 151 486.0

54 30 136 175 238.0

55 60 121 146 450.7

56 60 118 121 413.0

57 45 115 144 305.0

58 20 153 172 226.4

59 45 123 152 321.0

60 210 108 160 1376.0

61 160 110 137 1034.4

62 160 109 135 853.0

63 45 118 141 341.0

64 20 110 130 131.4

65 180 90 130 800.4

66 150 105 135 873.4

67 150 107 130 816.0

68 20 106 136 110.4

69 300 108 143 1500.2

70 150 97 129 1115.0

71 60 109 153 387.6

72 90 100 127 700.0

73 150 97 127 953.2

74 45 114 146 304.0

75 90 98 125 563.2

76 45 105 134 251.0

77 45 110 141 300.0

78 120 100 130 500.4

79 270 100 131 1729.0

80 30 159 182 319.2

81 45 149 169 344.0

82 30 103 139 151.1

83 120 100 130 500.0

84 45 100 120 225.3

85 30 151 170 300.1

86 45 102 136 234.0

87 120 100 157 1000.1

88 45 129 103 242.0

89 20 83 107 50.3

90 180 101 127 600.1

91 45 107 137 222222.0

92 30 90 107 105.3

93 15 80 100 50.5

94 20 150 171 127.4

95 20 151 168 229.4

96 30 95 128 128.2

97 25 152 168 244.2

98 30 109 131 188.2

99 90 93 124 604.1

100 20 95 112 77.7

101 90 90 110 500.0

102 90 90 100 500.0

103 90 90 100 500.4

104 30 92 108 92.7

105 30 93 128 124.0

106 180 90 120 800.3

107 30 90 120 86.2

108 90 90 120 500.3

109 210 137 184 1860.4

110 60 102 124 325.2

111 45 107 124 275.0

112 15 124 139 124.2

113 45 100 120 225.3

114 60 108 131 367.6

115 60 108 151 351.7

116 60 116 141 443.0

117 60 97 122 277.4

118 60 105 125 222222.0

119 60 103 124 332.7

120 30 112 137 193.9

121 45 100 120 100.7

122 60 119 169 336.7

123 60 107 127 344.9

124 60 111 151 368.5

125 60 98 122 271.0

126 60 97 124 275.3

127 60 109 127 382.0

128 90 99 125 466.4

129 60 114 151 384.0

130 60 104 134 342.5

131 60 107 138 357.5

132 60 103 133 335.0

133 60 106 132 327.5

134 60 103 136 339.0

135 20 136 156 189.0

136 45 117 143 317.7

137 45 115 137 318.0

138 45 113 138 308.0

139 20 141 162 222.4

140 60 108 135 390.0

141 60 97 127 222222.0

142 45 100 120 250.4

143 45 122 149 335.4

144 60 136 170 470.2

145 45 106 126 270.8

146 60 107 136 400.0

147 60 112 146 361.9

148 30 103 127 185.0

149 60 110 150 409.4

150 60 106 134 343.0

151 60 109 129 353.2

152 60 109 138 374.0

153 30 150 167 275.8

154 60 105 128 328.0

155 60 111 151 368.5

156 60 97 131 270.4

157 60 100 120 270.4

158 60 114 150 382.8

159 30 80 120 240.9

160 30 85 120 250.4

161 45 90 130 260.4

162 45 95 130 270.0

163 45 100 140 280.9

164 60 105 140 290.8

165 60 110 145 300.4

166 60 115 145 310.2

167 75 120 150 320.4

168 75 125 150 330.4

Q9. How do you concatenate two Pandas DataFrames?

Ans:- import pandas as pd

df = pd.read\_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

df

Output:-

PassengerId Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked

0 1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S

1 2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C

2 3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S

3 4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S

4 5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

... ... ... ... ... ... ... ... ... ... ... ... ...

886 887 0 2 Montvila, Rev. Juozas male 27.0 0 0 211536 13.0000 NaN S

887 888 1 1 Graham, Miss. Margaret Edith female 19.0 0 0 112053 30.0000 B42 S

888 889 0 3 Johnston, Miss. Catherine Helen "Carrie" female NaN 1 2 W./C. 6607 23.4500 NaN S

889 890 1 1 Behr, Mr. Karl Howell male 26.0 0 0 111369 30.0000 C148 C

890 891 0 3 Dooley, Mr. Patrick male 32.0 0 0 370376 7.7500 NaN Q

data1 = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],

'Age':[27, 24, 22, 32],

'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],

'Qualification':['Msc', 'MA', 'MCA', 'Phd'],

'Salary':[4567, 5678, 6788, 78899]}

data2 = {'Name':['Abhi', 'Ayushi', 'Dhiraj', 'Hitesh','Gaurav'],

'Age':[17, 14, 12, 52, 89],

'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj', 'Kolkata'],

'Qualification':['Btech', 'B.A', 'Bcom', 'B.hons','BCA'],

'Salary':[5678, 6789, 7890, 8900, 46578]}

df1=pd.DataFrame(data1)

df2=pd.DataFrame(data2)

df1

Output:-

Name Age Address Qualification Salary

0 Jai 27 Nagpur Msc 4567

1 Princi 24 Kanpur MA 5678

2 Gaurav 22 Allahabad MCA 6788

3 Anuj 32 Kannuaj Phd 78899

df2

Name Age Address Qualification Salary

0 Abhi 17 Nagpur Btech 5678

1 Ayushi 14 Kanpur B.A 6789

2 Dhiraj 12 Allahabad Bcom 7890

3 Hitesh 52 Kannuaj B.hons 8900

4 Gaurav 89 Kolkata BCA 46578

# Convert the dictionary into DataFrame

df1 = pd.DataFrame(data1,index=[0, 1, 2, 3])

# Convert the dictionary into DataFrame

df2 = pd.DataFrame(data2, index=[4, 5, 6, 7,8])

df1

Name Age Address Qualification Salary

0 Jai 27 Nagpur Msc 4567

1 Princi 24 Kanpur MA 5678

2 Gaurav 22 Allahabad MCA 6788

3 Anuj 32 Kannuaj Phd 78899

df2

Name Age Address Qualification Salary

4 Abhi 17 Nagpur Btech 5678

5 Ayushi 14 Kanpur B.A 6789

6 Dhiraj 12 Allahabad Bcom 7890

7 Hitesh 52 Kannuaj B.hons 8900

8 Gaurav 89 Kolkata BCA 46578

df\_concat=[df1, df2]

res=pd.concat(df\_concat)

res

Output:-

Name Age Address Qualification Salary

0 Jai 27 Nagpur Msc 4567

1 Princi 24 Kanpur MA 5678

2 Gaurav 22 Allahabad MCA 6788

3 Anuj 32 Kannuaj Phd 78899

4 Abhi 17 Nagpur Btech 5678

5 Ayushi 14 Kanpur B.A 6789

6 Dhiraj 12 Allahabad Bcom 7890

7 Hitesh 52 Kannuaj B.hons 8900

8 Gaurav 89 Kolkata BCA 46578

Q10. How do you merge two Pandas DataFrames on a specific column?

Ans:- # importing modules

import pandas as pd

# creating a dataframe

df1 = pd.DataFrame({'Name':['Raju', 'Rani', 'Geeta', 'Sita', 'Sohit'],

'Marks':[80, 90, 75, 88, 59]})

# creating another dataframe with different data

df2 = pd.DataFrame({'Name':['Raju', 'Divya', 'Geeta', 'Sita'],

'Grade':['A', 'A', 'B', 'A'],

'Rank':[3, 1, 4, 2 ],

'Gender':['Male', 'Female', 'Female', 'Female']})

# display df1

display(df1)

# display df2

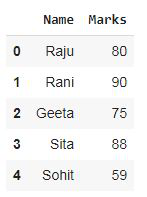
display(df2)

# applying merge with more parameters

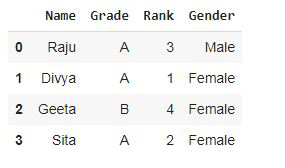
df2.merge(df1[['Marks', 'Name']])

**Output:**

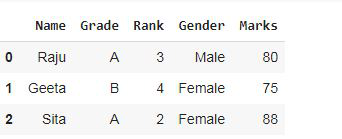
**df1**



df2



*Merged Dataframe*



Q11. How do you group data in a Pandas DataFrame by a specific column and apply an aggregation function?

Ans:- data1 = {'Name':['Jai', 'Anuj', 'Jai', 'Princi',

'Gaurav', 'Anuj', 'Princi', 'Abhi','Gaurav'],

'Age':[27, 24, 22, 32,

33, 36, 27, 32,56],

'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj',

'Jaunpur', 'Kanpur', 'Allahabad', 'Aligarh','Varanasi'],

'Qualification':['Msc', 'MA', 'MCA', 'Phd',

'B.Tech', 'B.com', 'Msc','MA', 'B.Tech'],

'Salary':[4789, 4989, 8990, 58959, 494940, 89900, 89900, 67789, 8899] }

df\_new = pd.DataFrame(data1)

df\_new

Name Age Address Qualification Salary

0 Jai 27 Nagpur Msc 4789

1 Anuj 24 Kanpur MA 4989

2 Jai 22 Allahabad MCA 8990

3 Princi 32 Kannuaj Phd 58959

4 Gaurav 33 Jaunpur B.Tech 494940

5 Anuj 36 Kanpur B.com 89900

6 Princi 27 Allahabad Msc 89900

7 Abhi 32 Aligarh MA 67789

8 Gaurav 56 Varanasi B.Tech 8899

df\_new.groupby('Name')

print(df\_new.groupby('Name').groups)

{'Abhi': [7], 'Anuj': [1, 5], 'Gaurav': [4, 8], 'Jai': [0, 2], 'Princi': [3, 6]}

df\_new.groupby(['Name', 'Qualification'])

print(df\_new.groupby(['Name', 'Qualification']).groups)

{('Abhi', 'MA'): [7], ('Anuj', 'B.com'): [5], ('Anuj', 'MA'): [1], ('Gaurav', 'B.Tech'): [4, 8], ('Jai', 'MCA'): [2], ('Jai', 'Msc'): [0], ('Princi', 'Msc'): [6], ('Princi', 'Phd'): [3]}

df.aggregate(['sum', 'min'])

PassengerId Survived Pclass Name Sex Age SibSp Parch Ticket Fare

sum 397386 342 2057 Braund, Mr. Owen HarrisCumings, Mrs. John Brad... malefemalefemalefemalemalemalemalemalefemalefe... 21205.17 466 340 A/5 21171PC 17599STON/O2. 31012821138033734503... 28693.9493

min 1 0 1 Abbing, Mr. Anthony female 0.42 0 0 110152 0.0000

Q12. How do you pivot a Pandas DataFrame?

Ans:- Pandas DataFrame: pivot() function

The pivot() function is used to reshaped a given DataFrame organized by given index / column values. This function does not support data aggregation, multiple values will result in a MultiIndex in the columns. Column to use to make new frame's index. If None, uses existing index.

# pandas.DataFrame.pivot

**DataFrame.pivot(*\**, *index=None*, *columns=None*, *values=None*)**

Return reshaped DataFrame organized by given index / column values.

Reshape data (produce a “pivot” table) based on column values. Uses unique values from specified index / columns to form axes of the resulting DataFrame. This function does not support data aggregation, multiple values will result in a MultiIndex in the columns. See the [User Guide](https://pandas.pydata.org/docs/user_guide/reshaping.html#reshaping) for more on reshaping.

**Parameters**

**index*str or object or a list of str, optional***

DataFrame.pivot(self, index=None, columns=None, values=None)

**pandas.pivot(index, columns, values)** function produces pivot table based on 3 columns of the DataFrame. Uses unique values from index / columns and fills with values.

***Parameters:******index[ndarray] :****Labels to use to make new frame’s index****columns[ndarray] :****Labels to use to make new frame’s columns****values[ndarray] :****Values to use for populating new frame’s values*

***Returns:****Reshaped DataFrame****Exception:****ValueError raised if there are any duplicates.*

# Create a simple dataframe

# importing pandas as pd

import pandas as pd

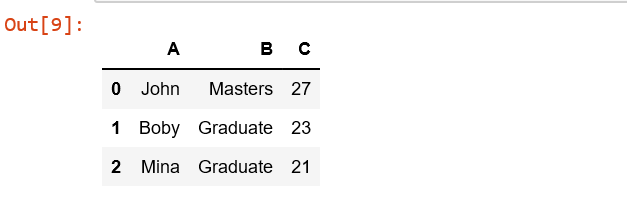
# creating a dataframe

df = pd.DataFrame({'A': ['John', 'Boby', 'Mina'],

      'B': ['Masters', 'Graduate', 'Graduate'],

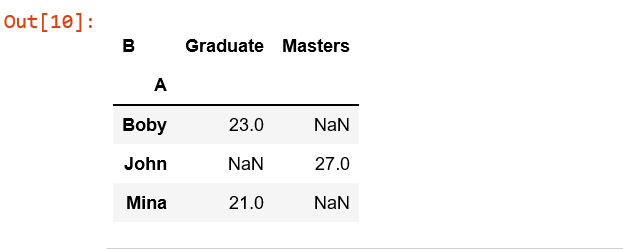
      'C': [27, 23, 21]})

df



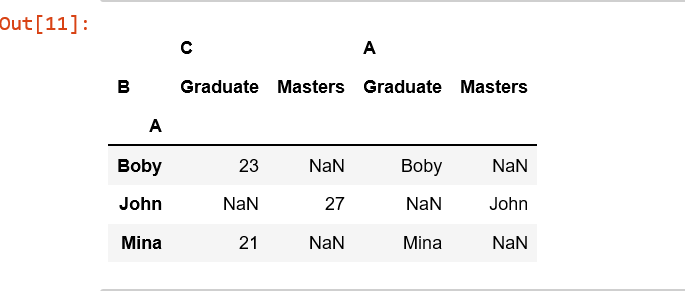
# values can be an object or a list

df.pivot('A', 'B', 'C')



# value is a list

df.pivot(index ='A', columns ='B', values =['C', 'A'])



Raise ValueError when there are any index, columns combinations with multiple values.

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # creating a dataframe  df = pd.DataFrame({'A': ['John', 'John', 'Mina'],        'B': ['Masters', 'Masters', 'Graduate'],        'C': [27, 23, 21]})      df.pivot('A', 'B', 'C') |

ValueError: Index contains duplicate entries, cannot reshape

Q13. How do you change the data type of a column in a Pandas DataFrame?

Ans:- To change the datatype of DataFrame columns, use DataFrame.astype() method, DataFrame.infer\_objects() method, or pd.to\_numeric.

### Method 1 – Using DataFrame.astype()

DataFrame.astype() casts this DataFrame to a specified datatype. Following is the syntax of astype() method.

astype(dtype, copy=True, errors='raise', \*\*kwargs)

we are interested only in the first argument **dtype.** dtype is data type, or dict of column name -> data type.

So, let us use astype() method with dtype argument to change datatype of one or more columns of DataFrame.

#### Change Datatype of One Column

Let us first start with changing datatype of just one column.

In the following program, we shall change the datatype of column **a** to **float**.

**Python Program**

import pandas as pd

import numpy as np

#initialize a dataframe

df = pd.DataFrame(

[[21, 72, 67],

[23, 78, 62],

[32, 74, 54],

[52, 54, 76]],

columns=['a', 'b', 'c'])

print('Previous Datatypes\n', df.dtypes, sep='')

#change datatype of column

df = df.astype({'a': np.float})

#print results

print('\nNew Datatypes\n', df.dtypes, sep='')

print('\nDataFrame\n', df, sep='')

**Output**

Previous Datatypes

a int64

b int64

c int64

dtype: object

New Datatypes

a float64

b int64

c int64

dtype: object

DataFrame

a b c

0 21.0 72 67

1 23.0 78 62

2 32.0 74 54

3 52.0 54 76

#### Change Datatype of Multiple Columns

Now, let us change datatype of more than one column. All, we have to do is provide more column\_name:datatype key:value pairs in the argument to astype() method.

In the following program, we shall change the datatype of column **a** to **float**, and **b** to **int8**.

**Python Program**

import pandas as pd

import numpy as np

#initialize a dataframe

df = pd.DataFrame(

[[21, 72, 67],

[23, 78, 62],

[32, 74, 54],

[52, 54, 76]],

columns=['a', 'b', 'c'])

print('Previous Datatypes\n', df.dtypes, sep='')

#change datatype of column

df = df.astype({'a': np.float, 'b': np.int8})

#print results

print('\nNew Datatypes\n', df.dtypes, sep='')

print('\nDataFrame\n', df, sep='')

Previous Datatypes

a int64

b int64

c int64

dtype: object

New Datatypes

a float64

b int8

c int64

dtype: object

DataFrame

a b c

0 21.0 72 67

1 23.0 78 62

2 32.0 74 54

3 52.0 54 76

#### Change Datatype of All Columns

If you would like to change the datatype of all columns of DataFrame, you can just pass this datatype as argument to astype() method, without the need of dictionary.

In the following program, we shall change the datatype of all column to **float**.

**Python Program**

import pandas as pd

import numpy as np

#initialize a dataframe

df = pd.DataFrame(

[[21, 72, 67],

[23, 78, 62],

[32, 74, 54],

[52, 54, 76]],

columns=['a', 'b', 'c'])

print('Previous Datatypes\n', df.dtypes, sep='')

#change datatype of column

df = df.astype(np.float)

#print results

print('\nNew Datatypes\n', df.dtypes, sep='')

print('\nDataFrame\n', df, sep='')

**Output**

Previous Datatypes

a int64

b int64

c int64

dtype: object

New Datatypes

a float64

b float64

c float64

dtype: object

DataFrame

a b c

0 21.0 72.0 67.0

1 23.0 78.0 62.0

2 32.0 74.0 54.0

3 52.0 54.0 76.0

### Method 2 – pd.to\_numeric

Consider that you have imported a DataFrame from Excel, CSV, or some other source, and you got all string values for DataFrame Elements. The datatype of these columns could be object. And you would like to convert the datatype of all these columns to fitting numeric datatypes.

Use the following syntax to convert datatype of DataFrame columns to numeric.

df = df.apply(pd.to\_numeric)

**Python Program**

import pandas as pd

import numpy as np

#initialize a dataframe

df = pd.DataFrame(

[['21', '72', '67'],

['23', '78', '62'],

['32', '74', '54'],

['52', '54', '76']],

columns=['a', 'b', 'c'])

print('Previous Datatypes\n', df.dtypes, sep='')

#change datatype of all columns

df = df.apply(pd.to\_numeric)

#print results

print('\nNew Datatypes\n', df.dtypes, sep='')

print('\nDataFrame\n', df, sep='')

**Output**

Previous Datatypes

a object

b object

c object

dtype: object

New Datatypes

a int64

b int64

c int64

dtype: object

DataFrame

a b c

0 21 72 67

1 23 78 62

2 32 74 54

3 52 54 76

Q14. How do you sort a Pandas DataFrame by a specific column?

Ans:- To sort the DataFrame based on the values in a single column, you'll use . sort\_values() . By default, this will return a new DataFrame sorted in ascending order. It does not modify the original DataFrame.

### Sorting by a Column in Ascending Order

To use .sort\_values(), you pass a single argument to the method containing the name of the column you want to sort by. In this example, you sort the DataFrame by the city08 column, which represents city MPG for fuel-only cars:

>>>

>>> df.sort\_values("city08")

city08 cylinders fuelType ... mpgData trany year

99 9 8 Premium ... N Automatic 4-spd 1993

1 9 12 Regular ... N Manual 5-spd 1985

80 9 8 Regular ... N Automatic 3-spd 1985

47 9 8 Regular ... N Automatic 3-spd 1985

3 10 8 Regular ... N Automatic 3-spd 1985

.. ... ... ... ... ... ... ...

9 23 4 Regular ... Y Automatic 4-spd 1993

8 23 4 Regular ... Y Manual 5-spd 1993

7 23 4 Regular ... Y Automatic 3-spd 1993

76 23 4 Regular ... Y Manual 5-spd 1993

2 23 4 Regular ... Y Manual 5-spd 1985

[100 rows x 10 columns]

This sorts your DataFrame using the column values from city08, showing the vehicles with the lowest MPG first. By default, .sort\_values() sorts your data in **ascending order**. Although you didn’t specify a name for the argument you passed to .sort\_values(), you actually used the by parameter, which you’ll see in the next example.

Q15. How do you create a copy of a Pandas DataFrame?

Ans:- Pandas DataFrame copy() Method

The copy() method returns a copy of the DataFrame. By default, the copy is a "deep copy" meaning that any changes made in the original DataFrame will NOT be reflected in the copy.

# Pandas DataFrame copy() Method

import pandas as pd

data = {

"name": ["Sally", "Mary", "John"],

"qualified": [True, False, False]

}

df = pd.DataFrame(data)

print(df)

#Make a copy:

newdf = df.copy()

print(newdf)

Output:-

name qualified

0 Sally True

1 Mary False

2 John False

name qualified

0 Sally True

1 Mary False

2 John False

# pandas.DataFrame.copy

**DataFrame.copy(*deep=True*)**

Make a copy of this object’s indices and data.

When deep=True (default), a new object will be created with a copy of the calling object’s data and indices. Modifications to the data or indices of the copy will not be reflected in the original object (see notes below).

When deep=False, a new object will be created without copying the calling object’s data or index (only references to the data and index are copied). Any changes to the data of the original will be reflected in the shallow copy (and vice versa).

**Parameters**

**deep*bool, default True***

Make a deep copy, including a copy of the data and the indices. With deep=False neither the indices nor the data are copied.

# Python Pandas – DataFrame.copy() function

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python packages. Pandas is one of those packages and makes importing and analyzing data much easier.

There are many ways to copy DataFrame in pandas. The first way is a simple way of assigning a dataframe object to a variable, but this has some drawbacks.

***Syntax:****DataFrame.copy(deep=True)*

*When deep=True (default), a new object will be created with a copy of the calling object’s data and indices. Modifications to the data or indices of the copy will not be reflected in the original object (see notes below).*

*When deep=False, a new object will be created without copying the calling object’s data or index (only references to the data and index are copied). Any changes to the data of the original will be reflected in the shallow copy (and vice versa).*

**Step** **1)**Let us first make a dummy data frame, which we will use for our illustration

**Step** **2)**Assign that dataframe object to a variable

**Step 3)**Make changes in the original dataframe to see if there is any difference in copied variable

import pandas as pd

#Create Series

s = pd.Series([3,4,5],['earth','mars','jupiter'])

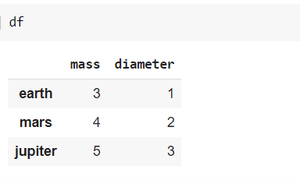
k = pd.Series([1,2,3],['earth','mars','jupiter'])

#Create DataFrame df from two series

df = pd.DataFrame({'mass':s,'diameter':k})

df

**Output:**



*Dummy DataFrame df*

Now, let’s assign the dataframe df to a variable and perform changes:

#Assign df to variable\_copy

variable\_copy = df

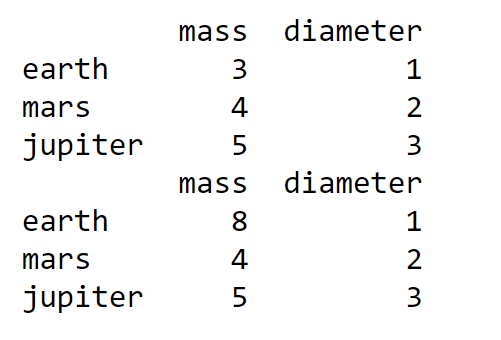
print(variable\_copy)

#Update the value of mass of earth in original dataframe

df['mass']['earth']=8

print(variable\_copy)

**Output:**



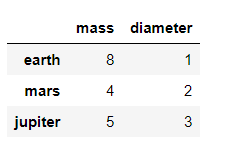
Here, we can see that if we change the values in the original dataframe, then the data in the copied variable also changes. To overcome this, we use DataFrame.copy()

**Let us see this, with examples when deep=True(default ):**

* Python3

|  |
| --- |
| res = df.copy(deep=True)  print(res) |

**Output:**



Q16. How do you filter rows of a Pandas DataFrame by multiple conditions?

Ans:-

# [Pandas dataframe filter with Multiple conditions](https://kanoki.org/2020/01/21/pandas-dataframe-filter-with-multiple-conditions/)

Selecting or filtering rows from a dataframe can be sometime tedious if you don’t know the exact methods and how to filter rows with multiple conditions

In this post we are going to see the different ways to select rows from a dataframe using multiple conditions

Let’s create a dataframe with 5 rows and 4 columns i.e. Name, Age, Salary\_in\_1000 and FT\_Team(Football Team)

import pandas as pd

df=pd.DataFrame({'Name':['JOHN','ALLEN','BOB','NIKI','CHARLIE','CHANG'],

'Age':[35,42,63,29,47,51],

'Salary\_in\_1000':[100,93,78,120,64,115],

'FT\_Team':['STEELERS','SEAHAWKS','FALCONS','FALCONS','PATRIOTS','STEELERS']})

df

**Output**:

| **-** | **Name** | **Age** | **Salary\_in\_1000** | **FT\_Team** |
| --- | --- | --- | --- | --- |
| 0 | JOHN | 35 | 100 | STEELERS |
| 1 | ALLEN | 42 | 93 | SEAHAWKS |
| 2 | BOB | 63 | 78 | FALCONS |
| 3 | NIKI | 29 | 120 | FALCONS |
| 4 | CHARLIE | 47 | 64 | PATRIOTS |
| 5 | CHANG | 51 | 115 | STEELERS |

## ****Selecting Dataframe rows on multiple conditions using these 5 functions****

In this section we are going to see how to filter the rows of a dataframe with multiple conditions using these five methods

a) loc b) numpy where c) Query d) Boolean Indexing e) eval

**What’s the Condition or Filter Criteria ?**

Get all rows having salary greater or equal to 100K and Age < 60 and Favourite Football Team Name starts with ‘S’

## ****Using loc with multiple conditions****

loc is used to Access a group of rows and columns by label(s) or a boolean array

As an input to label you can give a single label or it’s index or a list of array of labels

Enter all the conditions and with & as a logical operator between them

df.loc[(df['Salary\_in\_1000']>=100) & (df['Age']< 60) & (df['FT\_Team'].str.startswith('S')),['Name','FT\_Team']]

**Output:**

|  | **Name** | **FT\_Team** |
| --- | --- | --- |
| 0 | JOHN | STEELERS |
| 5 | CHANG | STEELERS |

## ****Using np.where with multiple conditions****

numpy where can be used to filter the array or get the index or elements in the array where conditions are met.

Numpy where with multiple conditions and & as logical operators outputs the index of the matching rows

import numpy as np

idx = np.where((df['Salary\_in\_1000']>=100) & (df['Age']< 60) & (df['FT\_Team'].str.startswith('S')))

**Output:**

(array([0, 5], dtype=int64),)

The output from the np.where, which is a list of row index matching the multiple conditions is fed to dataframe loc function

df.loc[idx]

**Output:**

|  | **Name** | **Age** | **Salary\_in\_1000** | **FT\_Team** |
| --- | --- | --- | --- | --- |
| 0 | JOHN | 35 | 100 | STEELERS |
| 5 | CHANG | 51 | 115 | STEELERS |

## ****Using Query with multiple Conditions****

It is used to Query the columns of a DataFrame with a boolean expression

df.query('Salary\_in\_1000 >= 100 & Age < 60 & FT\_Team.str.startswith("S").values')

**Output:**

|  | **Name** | **Age** | **Salary\_in\_1000** |
| --- | --- | --- | --- |
| 0 | JOHN | 35 | 100 |
| 5 | CHANG | 51 | 115 |

## ****pandas boolean indexing multiple conditions****

It is a standrad way to select the subset of data using the values in the dataframe and applying conditions on it

We are using the same multiple conditions here also to filter the rows from pur original dataframe with salary >= 100 and Football team starts with alphabet ‘S’ and Age is less than 60

df[(df['Salary\_in\_1000']>=100) & (df['Age']<60) & df['FT\_Team'].str.startswith('S')][['Name','Age','Salary\_in\_1000']]

**Output:**

|  | **Name** | **Age** | **Salary\_in\_1000** |
| --- | --- | --- | --- |
| 0 | JOHN | 35 | 100 |
| 5 | CHANG | 51 | 115 |

## ****Pandas Eval multiple conditions****

Evaluate a string describing operations on DataFrame column. It Operates on columns only, not specific rows or elements

df[df.eval("Salary\_in\_1000>=100 & (Age <60) & FT\_Team.str.startswith('S').values")]

**Output:**

|  | **Name** | **Age** | **Salary\_in\_1000** |
| --- | --- | --- | --- |
| 0 | JOHN | 35 | 100 |
| 5 | CHANG | 51 | 115 |

Q17. How do you calculate the mean of a column in a Pandas DataFrame?

Ans:- To calculate the mean of whole columns in the DataFrame, use pandas.Series.mean() with a list of DataFrame columns. You can also get the mean for all numeric columns using [DataFrame.mean()](https://sparkbyexamples.com/pandas/pandas-dataframe-mean-examples/), use axis=0 argument to calculate the column-wise mean of the DataFrame.

# Using DataFrame.mean() to get entire column mean

df2 = df.mean()

print(df2)

# Using multiple columns mean using DataFrame.mean()

df2 = df[["Fee","Discount"]].mean()

print(df2)

# Average of each column using DataFrame.mean()

df2 = df.mean(axis=0)

print(df2)

Above all examples yields the same below output.

Fee 24250.0

Discount 1625.0

dtype: float64

Pandas**dataframe.mean()** function return the mean of the values for the requested axis. If the method is applied on a pandas series object, then the method returns a scalar value which is the mean value of all the observations in the dataframe. If the method is applied on a pandas dataframe object, then the method returns a pandas series object which contains the mean of the values over the specified axis.

***Syntax:****DataFrame.mean(axis=None, skipna=None, level=None, numeric\_only=None, \*\*kwargs)*

***Parameters :******axis :****{index (0), columns (1)}****skipna :****Exclude NA/null values when computing the result*

***level :****If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a Series*

***numeric\_only :****Include only float, int, boolean columns. If None, will attempt to use everything, then use only numeric data. Not implemented for Series.*

***Returns :****mean : Series or DataFrame (if level specified)*

# importing pandas as pd

import pandas as pd

# Creating the dataframe

df = pd.DataFrame({"A":[12, 4, 5, 44, 1],

"B":[5, 2, 54, 3, 2],

"C":[20, 16, 7, 3, 8],

"D":[14, 3, 17, 2, 6]})

# Print the dataframe

df

Output:-



Let’s use the dataframe.mean() function to find the mean over the index axis.

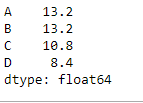
# Even if we do not specify axis = 0,

# the method will return the mean over

# the index axis by default

df.mean(axis = 0)

Output:-



Q18. How do you calculate the standard deviation of a column in a Pandas DataFrame?

Ans:- **DataFrame.std(*axis=None*, *skipna=True*, *level=None*, *ddof=1*, *numeric\_only=None*, *\*\*kwargs*)**

Return sample standard deviation over requested axis.

Normalized by N-1 by default. This can be changed using the ddof argument.

**Parameters**

**axis*{index (0), columns (1)}***

For *Series* this parameter is unused and defaults to 0.

**skipna*bool, default True***

Exclude NA/null values. If an entire row/column is NA, the result will be NA.

**level*int or level name, default None***

If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a Series.

**ddof*int, default 1***

Delta Degrees of Freedom. The divisor used in calculations is N - ddof, where N represents the number of elements.

**numeric\_only*bool, default None***

Include only float, int, boolean columns. If None, will attempt to use everything, then use only numeric data. Not implemented for Series.

**Returns**

**Series or DataFrame (if level specified)**

**Notes**

To have the same behaviour as *numpy.std*, use *ddof=0* (instead of the default *ddof=1*)

**Examples**

>>>

>>> df = pd.DataFrame({'person\_id': [0, 1, 2, 3],

... 'age': [21, 25, 62, 43],

... 'height': [1.61, 1.87, 1.49, 2.01]}

... ).set\_index('person\_id')

>>> df

age height

person\_id

0 21 1.61

1 25 1.87

2 62 1.49

3 43 2.01

The standard deviation of the columns can be found as follows:

>>>

>>> df.std()

age 18.786076

height 0.237417

Alternatively, *ddof=0* can be set to normalize by N instead of N-1:

>>>

>>> df.std(ddof=0)

age 16.269219

height 0.205609

Q19. How do you calculate the correlation between two columns in a Pandas DataFrame?

# Ans:- pandas.DataFrame.corr

**DataFrame.corr(*method='pearson'*, *min\_periods=1*, *numeric\_only=\_NoDefault.no\_default*)**

Compute pairwise correlation of columns, excluding NA/null values.

**Parameters**

**method*{‘pearson’, ‘kendall’, ‘spearman’} or callable***

Method of correlation:

* pearson : standard correlation coefficient
* kendall : Kendall Tau correlation coefficient
* spearman : Spearman rank correlation
* **callable: callable with input two 1d ndarrays**

and returning a float. Note that the returned matrix from corr will have 1 along the diagonals and will be symmetric regardless of the callable’s behavior.

**min\_periods*int, optional***

Minimum number of observations required per pair of columns to have a valid result. Currently only available for Pearson and Spearman correlation.

**numeric\_only*bool, default True***

Include only float, int or boolean data.

***New in version 1.5.0.***

***Deprecated since version 1.5.0:***The default value of numeric\_only will be False in a future version of pandas.

**Returns**

**DataFrame**

Correlation matrix.

See also

[**DataFrame.corrwith**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corrwith.html#pandas.DataFrame.corrwith)

Compute pairwise correlation with another DataFrame or Series.

[**Series.corr**](https://pandas.pydata.org/docs/reference/api/pandas.Series.corr.html#pandas.Series.corr)

Compute the correlation between two Series.

**Notes**

Pearson, Kendall and Spearman correlation are currently computed using pairwise complete observations.

* [Pearson correlation coefficient](https://en.wikipedia.org/wiki/Pearson_correlation_coefficient)
* [Kendall rank correlation coefficient](https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient)
* [Spearman’s rank correlation coefficient](https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient)

**Examples**

>>>

>>> **def** histogram\_intersection(a, b):

... v = np.minimum(a, b).sum().round(decimals=1)

... **return** v

>>> df = pd.DataFrame([(.2, .3), (.0, .6), (.6, .0), (.2, .1)],

... columns=['dogs', 'cats'])

>>> df.corr(method=histogram\_intersection)

dogs cats

dogs 1.0 0.3

cats 0.3 1.0

>>>

>>> df = pd.DataFrame([(1, 1), (2, np.nan), (np.nan, 3), (4, 4)],

... columns=['dogs', 'cats'])

>>> df.corr(min\_periods=3)

dogs cats

dogs 1.0 NaN

cats NaN 1.0

Q20. How do you select specific columns in a DataFrame using their labels?

# Ans:- Accessing columns of a DataFrame using column labels in Pandas

To access specific columns of a DataFrame with their columns labels, directly use DataFrame[~] or use the [DataFrame.loc](https://www.skytowner.com/explore/pandas_dataframe_loc_property) property.

# Example

Consider the following DataFrame:

df = pd.[DataFrame](https://www.skytowner.com/explore/pandas_dataframe_constructor)({"A":[3,4],"B":[5,6]}, index=["a","b"])

df

   A  B

a  3  5

b  4  6

## Accessing a single column

To access a single column:

df["A"]

a 3

b 4

Name: A, dtype: int64

We could also use the [loc](https://www.skytowner.com/explore/pandas_dataframe_loc_property) property, which is slightly more verbose:

df.[loc](https://www.skytowner.com/explore/pandas_dataframe_loc_property)[:,"A"]

a 3

b 4

Name: A, dtype: int64

Here, the : before the comma indicates that we want to retrieve all rows. The "A" after the comma then indicates that we just want to fetch column A.

## Accessing multiple columns

Consider the same df as above:

df = pd.[DataFrame](https://www.skytowner.com/explore/pandas_dataframe_constructor)({"A":[3,4],"B":[5,6]}, index=["a","b"])

df

   A  B

a  3  5

b  4  6

To access multiple columns, pass in a list of column labels:

df[["A","B"]]

   A  B

a  3  5

b  4  6

Using [loc](https://www.skytowner.com/explore/pandas_dataframe_loc_property) property:

df.[loc](https://www.skytowner.com/explore/pandas_dataframe_loc_property)[:,["A","B"]]

A B

a 3 5

b 4 6

The only difference with the single-case is that here we pass in a list of column labels as opposed to a single string.

Q21. How do you select specific rows in a DataFrame using their indexes?

Ans:- Pandas iloc is a method for integer-based indexing, which is used for selecting specific rows and subsetting pandas DataFrames and Series. The command to use this method is ***pandas.DataFrame.iloc()***  
  
The iloc method **accepts only integer-value arguments**. However, these arguments can be passed in different ways.

# Load and read the dataset as a DataFrame df = pd.read\_csv('D:/PERSONAL/DATASETS/vgsales.csv')

df



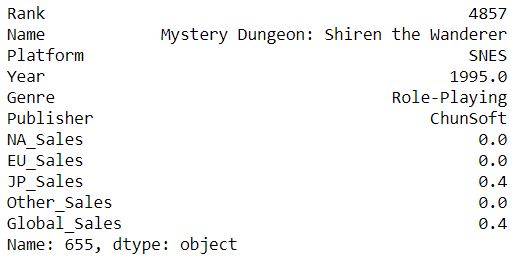
### Using a single integer value in Pandas iloc

You can pass a single integer value as the row index **to select a single row across all the columns**from the dataframe.

#### Example 1

# Subset a single row of the DataFrame

print(df.iloc[655])



Q22. How do you sort a DataFrame by a specific column?

# Ans:- pandas.DataFrame.sort

DataFrame.**sort**(columns=None, axis=0, ascending=True, inplace=False, kind='quicksort', na\_position='last', \*\*kwargs)

DEPRECATED: use [**DataFrame.sort\_values()**](https://pandas.pydata.org/pandas-docs/version/0.18/generated/pandas.DataFrame.sort_values.html#pandas.DataFrame.sort_values)

Sort DataFrame either by labels (along either axis) or by the values in column(s)

|  |  |
| --- | --- |
| **Parameters:** | **columns***: object*  Column name(s) in frame. Accepts a column name or a list for a nested sort. A tuple will be interpreted as the levels of a multi-index.  **ascending***: boolean or list, default True*  Sort ascending vs. descending. Specify list for multiple sort orders  **axis***: {0 or ‘index’, 1 or ‘columns’}, default 0*  Sort index/rows versus columns  **inplace***: boolean, default False*  Sort the DataFrame without creating a new instance  **kind***: {‘quicksort’, ‘mergesort’, ‘heapsort’}, optional*  This option is only applied when sorting on a single column or label.  **na\_position***: {‘first’, ‘last’} (optional, default=’last’)*  ‘first’ puts NaNs at the beginning ‘last’ puts NaNs at the end |
| **Returns:** | **sorted***: DataFrame* |

**Examples**

**>>>** result = df.sort(['A', 'B'], ascending=[1, 0])

# pandas.DataFrame.sort\_values

**DataFrame.sort\_values(*by*, *\**, *axis=0*, *ascending=True*, *inplace=False*, *kind='quicksort'*, *na\_position='last'*, *ignore\_index=False*, *key=None*)**

Sort by the values along either axis.

**Parameters**

**by*str or list of str***

Name or list of names to sort by.

* if axis is 0 or ‘index’ then by may contain index levels and/or column labels.
* if axis is 1 or ‘columns’ then by may contain column levels and/or index labels.

**axis*{0 or ‘index’, 1 or ‘columns’}, default 0***

Axis to be sorted.

**ascending*bool or list of bool, default True***

Sort ascending vs. descending. Specify list for multiple sort orders. If this is a list of bools, must match the length of the by.

**inplace*bool, default False***

If True, perform operation in-place.

**kind*{‘quicksort’, ‘mergesort’, ‘heapsort’, ‘stable’}, default ‘quicksort’***

Choice of sorting algorithm. See also [**numpy.sort()**](https://numpy.org/doc/stable/reference/generated/numpy.sort.html#numpy.sort) for more information. mergesort and stable are the only stable algorithms. For DataFrames, this option is only applied when sorting on a single column or label.

**na\_position*{‘first’, ‘last’}, default ‘last’***

Puts NaNs at the beginning if first; last puts NaNs at the end.

**ignore\_index*bool, default False***

If True, the resulting axis will be labeled 0, 1, …, n - 1.

***New in version 1.0.0.***

**key*callable, optional***

Apply the key function to the values before sorting. This is similar to the key argument in the builtin **sorted()** function, with the notable difference that this key function should be vectorized. It should expect a Series and return a Series with the same shape as the input. It will be applied to each column in by independently.

***New in version 1.1.0.***

**Returns**

**DataFrame or None**

DataFrame with sorted values or None if inplace=True.

See also

[**DataFrame.sort\_index**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_index.html#pandas.DataFrame.sort_index)

Sort a DataFrame by the index.

[**Series.sort\_values**](https://pandas.pydata.org/docs/reference/api/pandas.Series.sort_values.html#pandas.Series.sort_values)

Similar method for a Series.

**Examples**

>>>

>>> df = pd.DataFrame({

... 'col1': ['A', 'A', 'B', np.nan, 'D', 'C'],

... 'col2': [2, 1, 9, 8, 7, 4],

... 'col3': [0, 1, 9, 4, 2, 3],

... 'col4': ['a', 'B', 'c', 'D', 'e', 'F']

... })

>>> df

col1 col2 col3 col4

0 A 2 0 a

1 A 1 1 B

2 B 9 9 c

3 NaN 8 4 D

4 D 7 2 e

5 C 4 3 F

Sort by col1

>>>

>>> df.sort\_values(by=['col1'])

col1 col2 col3 col4

0 A 2 0 a

1 A 1 1 B

2 B 9 9 c

5 C 4 3 F

4 D 7 2 e

3 NaN 8 4 D

Sort by multiple columns

>>>

>>> df.sort\_values(by=['col1', 'col2'])

col1 col2 col3 col4

1 A 1 1 B

0 A 2 0 a

2 B 9 9 c

5 C 4 3 F

4 D 7 2 e

3 NaN 8 4 D

Sort Descending

>>>

>>> df.sort\_values(by='col1', ascending=**False**)

col1 col2 col3 col4

4 D 7 2 e

5 C 4 3 F

2 B 9 9 c

0 A 2 0 a

1 A 1 1 B

3 NaN 8 4 D

Putting NAs first

>>>

>>> df.sort\_values(by='col1', ascending=**False**, na\_position='first')

col1 col2 col3 col4

3 NaN 8 4 D

4 D 7 2 e

5 C 4 3 F

2 B 9 9 c

0 A 2 0 a

1 A 1 1 B

Sorting with a key function

>>>

>>> df.sort\_values(by='col4', key=**lambda** col: col.str.lower())

col1 col2 col3 col4

0 A 2 0 a

1 A 1 1 B

2 B 9 9 c

3 NaN 8 4 D

4 D 7 2 e

5 C 4 3 F

Natural sort with the key argument, using the natsort <https://github.com/SethMMorton/natsort> package.

>>>

>>> df = pd.DataFrame({

... "time": ['0hr', '128hr', '72hr', '48hr', '96hr'],

... "value": [10, 20, 30, 40, 50]

... })

>>> df

time value

0 0hr 10

1 128hr 20

2 72hr 30

3 48hr 40

4 96hr 50

>>> **from** natsort **import** index\_natsorted

>>> df.sort\_values(

... by="time",

... key=**lambda** x: np.argsort(index\_natsorted(df["time"]))

... )

time value

0 0hr 10

3 48hr 40

2 72hr 30

4 96hr 50

1 128hr 20

Q23. How do you create a new column in a DataFrame based on the values of another column?

# Ans:- Creating a new column based on other columns in Pandas DataFrame

To create a new column based on other columns, either:

* use column-arithmetics for fastest performance.
* use NumPy's [where(~)](https://www.skytowner.com/explore/numpy_where_method) method for creating binary columns
* use the [apply(~)](https://www.skytowner.com/explore/pandas_dataframe_apply_method) method, which is the slowest but offers the most flexibility
* use the Series' replace(~) method for mapping new values from existing columns.

# Creating new columns using arithmetics

Consider the following DataFrame:

df = pd.[DataFrame](https://www.skytowner.com/explore/pandas_dataframe_constructor)({"A":[3,4],"B":[5,6]}, index=["a","b"])

df

A B

a 3 5

b 4 6

The fastest and simplest way of creating a new column is to use simple column-arithmetics:

df["C"] = df["A"] + df["B"]

df

A B C

a 3 5 8

b 4 6 10

For slightly more complicated operations, use the DataFrame's native methods:

df["C"] = df.[max](https://www.skytowner.com/explore/pandas_dataframe_max_method)(axis=1)

df

A B C

a 3 5 5

b 4 6 6

Note the following:

* we are populating the new column C with the maximum of each row (axis=1).
* the return type of [df.max(axis=1)](https://www.skytowner.com/explore/pandas_dataframe_max_method) is Series.

# Creating binary column values

Consider the following Pandas DataFrame:

df = pd.[DataFrame](https://www.skytowner.com/explore/pandas_dataframe_constructor)({'name':['Alex','Bob','Cathy'],'age':[20,30,40]})

df.[head](https://www.skytowner.com/explore/pandas_dataframe_head_method)()

name age

0 Alex 20

1 Bob 30

2 Cathy 40

To create a new column of binary values that are based on the age column, use NumPy's [where(~)](https://www.skytowner.com/explore/numpy_where_method) method:

df['status'] = np.[where](https://www.skytowner.com/explore/numpy_where_method)(df['age'] < 25, 'JUNIOR', 'SENIOR')

df.[head](https://www.skytowner.com/explore/pandas_dataframe_head_method)()

name age status

0 Alex 20 JUNIOR

1 Bob 30 SENIOR

2 Cathy 40 SENIOR

Here, the first argument of the [where(~)](https://www.skytowner.com/explore/numpy_where_method) method is a boolean mask. If the boolean value is True, then resulting value will be 'JUNIOR', otherwise the value will be 'SENIOR'.

# Creating column with multiple values

Once again, consider the following Pandas DataFrame:

df = pd.[DataFrame](https://www.skytowner.com/explore/pandas_dataframe_constructor)({'name':['Alex','Bob','Cathy'],'age':[20,30,40]})

df.[head](https://www.skytowner.com/explore/pandas_dataframe_head_method)()

name age

0 Alex 20

1 Bob 30

2 Cathy 40

To create a new column with multiple values based on the age column, use the [apply(~)](https://www.skytowner.com/explore/pandas_dataframe_apply_method) function:

def my\_func(row):

if row['age'] < 25:

val = 'JUNIOR'

elif row['age'] < 35:

val = 'MID-LEVEL'

else:

val = 'SENIOR'

return val

df['status'] = df.[apply](https://www.skytowner.com/explore/pandas_dataframe_apply_method)(my\_func, axis=1)

df.[head](https://www.skytowner.com/explore/pandas_dataframe_head_method)()

name age status

0 Alex 20 JUNIOR

1 Bob 30 MID-LEVEL

2 Cathy 40 SENIOR

Here, the [apply(~)](https://www.skytowner.com/explore/pandas_dataframe_apply_method) function is iteratively called for each row, and takes in as argument a Series representing a row.

# Creating column via mapping

Consider the same Pandas DataFrame as before:

df = pd.[DataFrame](https://www.skytowner.com/explore/pandas_dataframe_constructor)({'name':['Alex','Bob','Cathy'],'age':[20,30,40]})

df.[head](https://www.skytowner.com/explore/pandas_dataframe_head_method)()

name age

0 Alex 20

1 Bob 30

2 Cathy 40

To create a new column that is based on some mapping of an existing column:

mapping = {

'Alex': 'ALEX',

'Bob': 'BOB',

'Cathy': 'CATHY'

}

df['upper\_name'] = df['name'].replace(mapping)

df.[head](https://www.skytowner.com/explore/pandas_dataframe_head_method)()

name age upper\_name

0 Alex 20 ALEX

1 Bob 30 BOB

2 Cathy 40 CATHY

# Creating column using the assign method

Consider the following Pandas DataFrame:

df = pd.[DataFrame](https://www.skytowner.com/explore/pandas_dataframe_constructor)({"A":[3,4],"B":[5,6]}, index=["a","b"])

df

A B

a 3 5

b 4 6

We could also use the DataFrame's [assign(~)](https://www.skytowner.com/explore/pandas_dataframe_assign_method) method, which takes in as argument a function with the DataFrame as the input and returns the new column values:

def foo(df):

if df["A"].sum() > df["B"].sum():

return [-1,-1]

else:

return [0,0]

df.[assign](https://www.skytowner.com/explore/pandas_dataframe_assign_method)(C=foo)

A B C

0 3 5 0

1 4 6 0

Note the following:

* if the sum of column A is larger than that of column B, then [-1,-1] will be used as the new column, otherwise [0,0] will be used.
* the keyword argument (C) became the new column label.

Q24. How do you remove duplicates from a DataFrame?

# Ans:- pandas.DataFrame.drop\_duplicates

**DataFrame.drop\_duplicates(*subset=None*, *\**, *keep='first'*, *inplace=False*, *ignore\_index=False*)**

Return DataFrame with duplicate rows removed.

Considering certain columns is optional. Indexes, including time indexes are ignored.

**Parameters**

**subset*column label or sequence of labels, optional***

Only consider certain columns for identifying duplicates, by default use all of the columns.

**keep*{‘first’, ‘last’, False}, default ‘first’***

Determines which duplicates (if any) to keep. - first : Drop duplicates except for the first occurrence. - last : Drop duplicates except for the last occurrence. - False : Drop all duplicates.

**inplace*bool, default False***

Whether to modify the DataFrame rather than creating a new one.

**ignore\_index*bool, default False***

If True, the resulting axis will be labeled 0, 1, …, n - 1.

***New in version 1.0.0.***

**Returns**

**DataFrame or None**

DataFrame with duplicates removed or None if inplace=True.

See also

[**DataFrame.value\_counts**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.value_counts.html#pandas.DataFrame.value_counts)

Count unique combinations of columns.

**Examples**

Consider dataset containing ramen rating.

>>>

>>> df = pd.DataFrame({

... 'brand': ['Yum Yum', 'Yum Yum', 'Indomie', 'Indomie', 'Indomie'],

... 'style': ['cup', 'cup', 'cup', 'pack', 'pack'],

... 'rating': [4, 4, 3.5, 15, 5]

... })

>>> df

brand style rating

0 Yum Yum cup 4.0

1 Yum Yum cup 4.0

2 Indomie cup 3.5

3 Indomie pack 15.0

4 Indomie pack 5.0

By default, it removes duplicate rows based on all columns.

>>>

>>> df.drop\_duplicates()

brand style rating

0 Yum Yum cup 4.0

2 Indomie cup 3.5

3 Indomie pack 15.0

4 Indomie pack 5.0

To remove duplicates on specific column(s), use subset.

>>>

>>> df.drop\_duplicates(subset=['brand'])

brand style rating

0 Yum Yum cup 4.0

2 Indomie cup 3.5

To remove duplicates and keep last occurrences, use keep.

>>>

>>> df.drop\_duplicates(subset=['brand', 'style'], keep='last')

brand style rating

1 Yum Yum cup 4.0

2 Indomie cup 3.5

4 Indomie pack 5.0

Q25. What is the difference between .loc and .iloc in Pandas?

Ans:- Indexing in pandas means simply selecting particular rows and columns of data from a DataFrame. **.loc()** and **.iloc()** are two such most commonly used indexing technique in pandas.

**.loc()** is a **label-based** data selection technique, which means we have to pass the name of the row or column to be selected. Input you can use for **.loc()** are:

1. A string
2. A list of strings
3. Slice notation using strings as the start and stop values

**.iloc()** is an **indexed-based** selection technique which means that we have to pass integer index in the method to select a specific row/column. Input you can use for **.iloc** are:

1. An integer
2. A list of integers
3. Slice notation using integers as the start and stop values

**Important to note**: iloc() does not include the last element of the range passed in it, unlike .loc()

import pandas as pd

*#reading data in pandas*

df = pd.read\_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

df

PassengerId Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked

0 1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S

1 2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C

2 3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S

3 4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S

4 5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

... ... ... ... ... ... ... ... ... ... ... ... ...

886 887 0 2 Montvila, Rev. Juozas male 27.0 0 0 211536 13.0000 NaN S

887 888 1 1 Graham, Miss. Margaret Edith female 19.0 0 0 112053 30.0000 B42 S

888 889 0 3 Johnston, Miss. Catherine Helen "Carrie" female NaN 1 2 W./C. 6607 23.4500 NaN S

889 890 1 1 Behr, Mr. Karl Howell male 26.0 0 0 111369 30.0000 C148 C

890 891 0 3 Dooley, Mr. Patrick male 32.0 0 0 370376 7.7500 NaN Q

*#Selecting a row using index*

df.loc[100]

Out[21]:

PassengerId 101

Survived 0

Pclass 3

Name Petranec, Miss. Matilda

Sex female

Age 28.0

SibSp 0

Parch 0

Ticket 349245

Fare 7.8958

Cabin NaN

Embarked S

Name: 100, dtype: object

*#Selecting a row using default index*

df.iloc[100]

Out[22]:

PassengerId 101

Survived 0

Pclass 3

Name Petranec, Miss. Matilda

Sex female

Age 28.0

SibSp 0

Parch 0

Ticket 349245

Fare 7.8958

Cabin NaN

Embarked S

Name: 100, dtype: object