Contents

Dataframe

Dataframe is most commonly used object in pandas. It is a table like datastructure containing rows and columns similar to excel spreadsheet

Creating a Dataframe using pandas

pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)

To save created dataframe into a csv file use:

df.to_csv('weather_data.csv', index=False)

Operations

- df.describe() → generates descriptive statistics about the DataFrame such as count, mean, median, min, max…etc
- df.info() →print a concise summary of a DataFrame such as Non-Null Count and Dtype
- df.duplicated() → returns boolean Series denoting duplicate rows, if I want to return sum of duplicated I can use df.duplicated().sum()
- df.isnull().sum() → returns the sum of each null values for each column
- df.nunique() → counts number of distinct elements in specified axis
- df['col'].std() → returns the STD value of a specific column
- max(), min() → returns the max/min value in a column
- df.shape → returns a tuple representing the dimensionality of the DataFrame.
- df.T → The transpose of the DataFrame.
- df.empty → checks if DataFrame is empty
- df.head(n) → returns the first n rows and if n is not specified its 5 by default
- df.tail(n) → returns the last n rows.
- df.columns → returns a list of column labels in the DataFrame

- df.set_index("col") → Set the DataFrame index using existing column
- df.reset_index(inplace=True) → reset the index, or a level of it.
- df.loc[col_name] → access a group of rows and columns by label(s) or a boolean array
- df.values → return a Numpy representation of the DataFrame.
- df.size → return an int representing the number of elements in this object.
- df.abs() → absolute numeric value of each element in the DataFrame
- apply(func[, axis, raw, result_type, args, ...]) → Apply a function along an axis of the DataFrame.
- boxplot([column, by, ax, fontsize, rot, ...]) → Make a box plot from DataFrame columns.
- clip([lower, upper, axis, inplace]) → Trim values at input threshold(s).
- corr([method, min_periods, numeric_only]) → Compute pairwise correlation of columns, excluding NA/null values.
- drop([labels, axis, index, columns, level, ...]) → Drop specified labels from rows or columns.
- drop_duplicates([subset, keep, inplace, ...]) → Return DataFrame with duplicate rows removed.
- dropna() → Remove missing values.
- fillna([value, method, axis, inplace, ...]) → Fill NA/NaN values using the specified method.
- hist([column, by, grid, xlabelsize, xrot, ...]) → Make a histogram of the DataFrame's columns.
- insert(loc, column, value[, allow_duplicates]) → Insert column into DataFrame at specified location.

- interpolate([method, axis, limit, inplace, ...]) → Fill NaN values using an interpolation method.
- items() → Iterate over (column name, Series) pairs.
- iterrows() → Iterate over DataFrame rows as (index, Series) pairs.
- mean() → Returns the mean of the values over the requested axis.
- median() → Returns the median of the values over the requested axis.
- mode() → Get the mode(s) of each element along the selected axis.
- I can use add/div/sub/mul methods on numerical data
- aggregate([func, axis]) → Aggregate using one or more operations over the specified axis.
- convert_dtypes([infer_objects, ...]) → Convert columns to the best possible dtypes using dtypes supporting pd.NA.
-

Numpy

Import numpy as np

Create an array

Arr = np.array([1,2,3,4,5]) → 1D array

Arr2 = np.array([[1,2,3],[4,5,6]]) → 2D array

I can even specify the dimension using ndim :

arr = np.array([1, 2, 3, 4], ndmin=5)

Check for dimension

Arr.ndim

Array Shape

Shape of an array is the number of elements in each dimension.

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape) → (2,4)

I can also reshape an array using reshape(x,y)

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3) → 4 arrays, each with 3 elements

2 arrays that contains 3 arrays, each with 2 elements: arr.reshape(2, 3, 2)

Flattening array: means converting a multidimensional array into a 1D

array. Use arr.reshape(-1)

Access an element(s)

print(arr2[0])

print(arr2[0,1])

Slicing

print(arr[1:5])

print(arr[4:])

arr3 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4]) → second element, slice elements from index 1 to index 4

Data Types

Use dtype to specify the data type

Available data types:

i - integer

b - boolean

u - unsigned integer

f - float

c - complex float

m - timedelta

M - datetime

O - object

S - string

U - unicode string

V - fixed chunk of memory for other type

To change the data type of an existing array, is to make a copy of the array with the astype() method.

Array Copy vs View

The copy is a new array, and the view is just a view of the original array

x = arr.copy()

y = arr.view()

copies owns the data, and views does not own the data we can use base()that returns None if the array owns the data

Array Iterating

1D array

```
arr = np.array([1, 2, 3])

for x in arr:

  print(x)
```

2D array

Iterate on the elements

```
arr2 = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr2:

  print(x)
```

Iterate on each scalar element

```
for x in arr:

  for y in x:

    print(y)
```

Or we use The function nditer() it's a helping function that can be used from very basic to very advanced iterations. It is used with high dimensionality arrays since its harder to use basic for loops with them.

We can use different step sizes too

```
for x in np.nditer(arr[:, ::2]):

  print(x)
```

Joining Array

in NumPy we join arrays by axes

We pass a sequence of arrays that we want to join to the concatenate() function, along with the axis. If axis is not explicitly passed default is 0.

Stacking is same as concatenation, the only difference is that stacking is done along a new axis.

hstack() to stack along rows.

vstack() to stack along columns.

Splitting Array

We use array_split() for splitting arrays, we pass it the array we want to split and the number of splits.

hsplit() opposite of hstack()

Searching Arrays

We can use where() method it returns the indexes that get a match of the value specified

searchsorted() method is assumed to be used on sorted arrays (performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.)

Sorting Arrays

Use np.sort(arr), it returns a copy of the array, leaving the original array unchanged and can be used to sort numeric and alphabetic array

Operations

Operations on numpy array

 # Mean

np.mean(arr)

 # Median

np.median(arr)

 # Variance

np.var(arr)

 # STD

np.std(arr)

 # Sum

np.sum(arr)

 # cummulative sum

np.cumsum(arr)

 # Cumulative product

np.cumprod(arr)

 # MIN and MAX

np.min(arr) and np.max(arr)))

 # Addition

np.add(x, y)

 # Subtraction

Subtracts elements of the second array from the first element-wise.

np.subtract(x, y)

 # Multiplication

Multiplies elements of two arrays element-wise.

np.multiply(x, y)

 # Division

np.divide(x, y)

 # Power

Element-wise exponentiation, raising elements of the first array to the powers of the second array.

np.power(x, y)

 # Absolute

Returns the absolute value of each element.

np.abs(x) or np.absolute(x)

 # Exponentiation

Computes the exponential (e^x) of each element.

np.exp(x)

 # Square Root

Returns the square root of each element.

np.sqrt(x)

 # Square

Squares each element of the array.

np.square(x)