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)

data → is the main input for the DataFrame and should be specified can be a numpy array, lists, dictionaries and series

index → Labels for the rows if not provided, it defaults to a RangeIndex (0, 1, 2, …). This allows you to define custom row labels.

Columns → Labels for the columns. default to a RangeIndex. custom labels for the columns can be set.

Dtype → The data type for the entire DataFrame. If not specified, pandas will infer the data type of each column from the provided data.

To save created dataframe into a csv file use:

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

Operations

1. df.describe() → generates descriptive statistics about the DataFrame such as count, mean, median, min, max…etc
2. df.info() →print a concise summary of a DataFrame such as Non-Null Count and Dtype
3. df.duplicated() → returns boolean Series denoting duplicate rows, if I want to return sum of duplicated I can use df.duplicated().sum()
4. df.isnull().sum() → returns the sum of each null values for each column
5. df.nunique() → counts number of distinct elements in specified axis
6. df['col'].std() → returns the STD value of a specific column
7. max(), min() → returns the max/min value in a column

8. df.shape → returns a tuple representing the dimensionality of the DataFrame.
9. df.T → The transpose of the DataFrame.
10. df.empty → checks if DataFrame is empty
11. df.head(n) → returns the first n rows and if n is not specified its 5 by default
12. df.tail(n) → returns the last n rows.
13. df.columns → returns a list of column labels in the DataFrame
14. df.set_index(“col”) → Set the DataFrame index using existing column
15. df.reset_index(inplace=True) → reset the index, or a level of it.
16. df.loc[col_name] → access a group of rows and columns by label(s) or a boolean array
17. df.values → return a Numpy representation of the DataFrame.
18. df.size → return an int representing the number of elements in this object.
19. df.abs() → absolute numeric value of each element in the DataFrame
20. apply(func[, axis, result_type, args]) → Apply a function along an axis of the DataFrame.

    func: The function to apply to each row or column. Can be a Python function, lambda function, or even a NumPy function

    result_type: Specifies the output type

    a. 'expand': Expands the result into separate columns.
    b. 'reduce': Tries to reduce the result to the lowest dimension.
    c. 'broadcast': Broadcasts the result to match the original shape.

    axis: Determines whether to apply the function to rows or columns (0 (default) applies on cols and 1 for rows)

    d. args: Additional positional arguments to pass to the function.

21. boxplot([column, by, ax, fontsize]) → Make a box plot from DataFrame columns.
    a. column: Specifies the column(s) to be plotted. default all numeric columns are plotted.
    b. by: Group data based on a column and create a boxplot for each group.
    c. ax: The matplotlib axes object to draw the box plot on.
    d. fontsize: Font size of labels and title.
22. clip([lower, upper, axis, inplace]) → Trim values at input threshold(s)
    a. lower: Minimum threshold value, values below this are replaced by this threshold.
    b. upper: Maximum threshold value, values above this are replaced by this threshold.
    c. axis: Determines whether to apply clipping across rows (axis=0) or columns (axis=1).
    d. inplace: If True, performs the operation in-place, modifying the original DataFrame.
23. corr([method, min_periods, numeric_only]) → Compute pairwise correlation of columns, excluding NA/null values.
    a. method: The correlation method to use: 'pearson' (default), 'kendall', or 'spearman'.
    b. min_periods: Minimum number of observations required per pair of columns to compute the correlation.
    c. numeric_only: Whether to include only float, int, or boolean data types.

24. drop([labels, axis, index, columns, level, inplace]) → Drop specified labels from rows or columns.
    a. labels: The labels to drop, can be row or column names.
    b. axis: Axis along which to drop labels (0 for rows, 1 for columns).
    c. index/columns: Alternative way to specify labels from rows (index) or columns (columns).
    d. level: For multi-level (hierarchical) indexing, specifies which level to drop.
    e. inplace: If True, performs the operation in-place, modifying the original DataFrame.
25. drop_duplicates([subset, keep, inplace]) → Return DataFrame with duplicate rows removed.
    a. subset: Columns to consider when identifying duplicates. If not specified, all columns are used.
    b. keep: Decides which duplicate to keep ('first', 'last', or False to drop all duplicates).
    c. inplace: If True, performs the operation in-place, modifying the original DataFrame.
26. dropna() → Remove missing values.
27. fillna([value, method, axis, inplace]) → Fill NA/NaN values using the specified method.
    a. value: Scalar, dictionary, or Series value to replace missing data.
    b. method: Method to use for filling NA values ('ffill' for forward fill, 'bfill' for backward fill).
    c. axis: Axis along which to fill the values (0 for rows, 1 for columns).

   d. inplace: If True, performs the operation in-place.

28. hist([column, by, grid, xlabelsize]) → Make a histogram of the DataFrame's columns.
    a. column: Specifies which column(s) to plot the histogram for. If not specified, all numeric columns are plotted.
    b. by: Groups the data by a specific column and creates histograms for each group.
    c. grid: Whether to display a grid on the histogram (True by default).
    d. xlabelsize: Size of the x-axis labels.
29. insert(loc, column, value[, allow_duplicates]) → Insert column into DataFrame at specified location.
    a. loc: The position (index) to insert the column.
    b. column: Name of the new column to be inserted.
    c. value: The values for the new column.
    d. allow_duplicates: If True, allows duplicate column names.
30. interpolate([method, axis, limit, inplace]) → Fill NaN values using an interpolation method.
    a. method: The interpolation method to use, such as 'linear' (default), 'polynomial', or 'spline'.
    b. axis: Axis along which to interpolate (0 for rows, 1 for columns).
    c. limit: Maximum number of consecutive NaN values to fill.
    d. inplace: If True, performs the operation in-place.
31. items() → Iterate over (column name, Series) pairs.
32. iterrows() → Iterate over DataFrame rows as (index, Series) pairs.
33. mean() → Returns the mean of the values over the requested axis.

34. median() → Returns the median of the values over the requested axis.
35. mode() → Get the mode(s) of each element along the selected axis.
36. I can use add/div/sub/mul methods on numerical data
37. aggregate([func, axis]) → Aggregate using one or more operations over the specified axis.
38. convert_dtypes([infer_objects]) → Convert columns to the best possible dtypes using dtypes supporting pd.NA.
39. groupby([by, axis, level, as_index, sort, ...]) → Group DataFrame using a mapper or by a Series of columns.
    a) by: Specifies the column(s), Series, or function used to group the data. Can be a single column, list of columns, or a function that maps rows.
    b) axis: Specifies whether to group rows (0, default) or columns (1).
    c) level: Groups by levels of a MultiIndex DataFrame. Can be a single level or a list of levels.
    d) as_index: If True (default), the grouping labels become the index in the result. If False, the result retains the original index.
    e) sort: If True (default), sorts the group labels. If False, groups are kept in their original order.
    f) group_keys: If True (default), the group labels are included when applying a function.
    g) dropna: If True (default), drops groups with all NA values. If False, includes NA as a valid group.

Numpy

Import numpy as np

Create an array

numpy.array(object, dtype=None, ndmin=0, like=None)

dtype → to specify the data type

like → Reference object to allow the creation of arrays which are not NumPy arrays.

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

Arr[start:end:step]

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 (i4

b - boolean

u - unsigned integer

f - float

c - complex float

O - object

S - string

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)
```