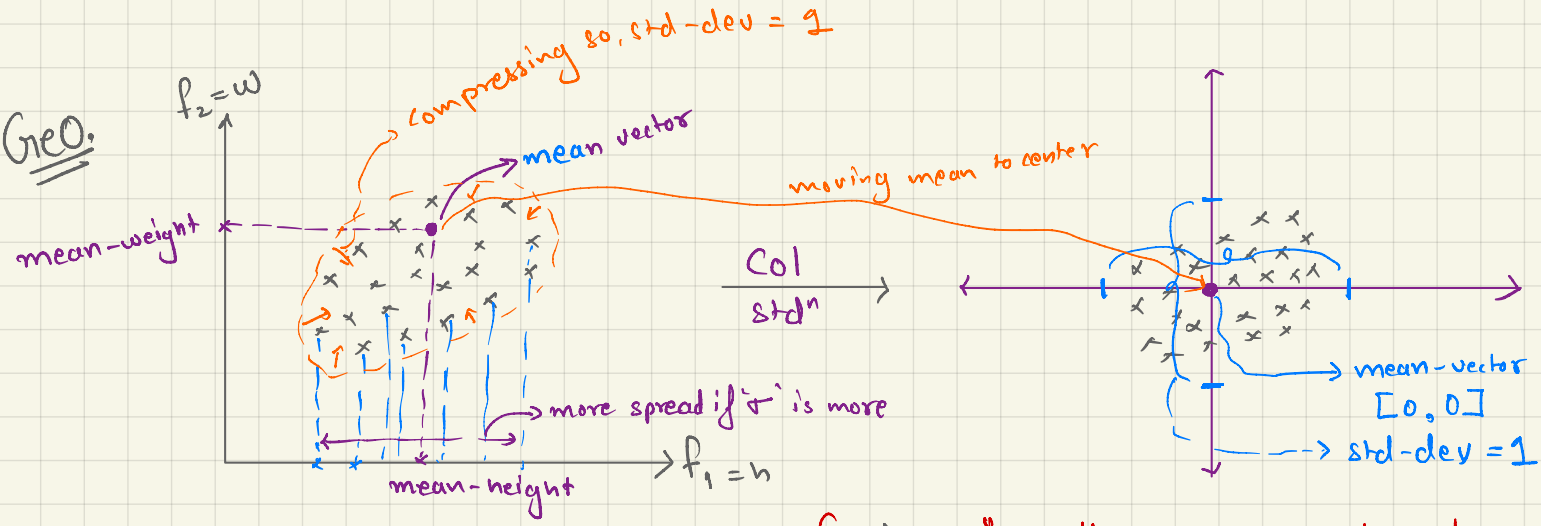
Geo.

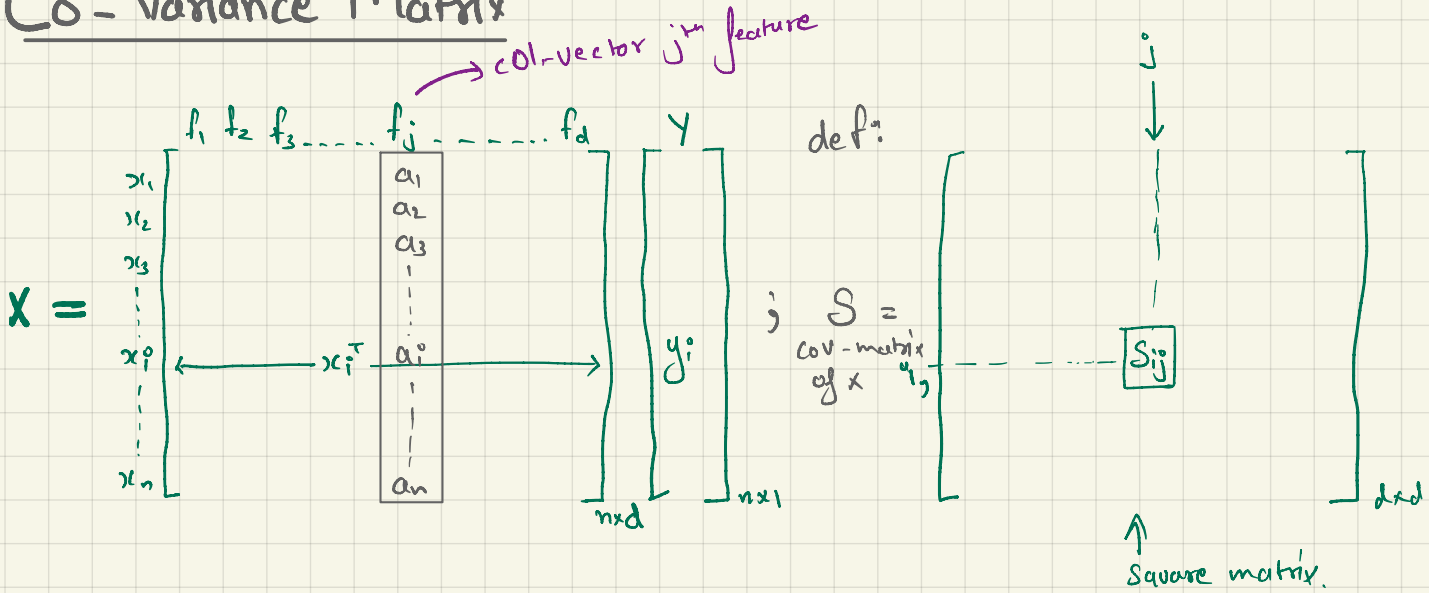


col. std. →
- → moving the mean-vector to origin.
- → squishing/expanding s.t std-dev for any feature is 1.

col. stand. :- mean-centering → origin

+ scaling → std-dev for all feature } = 1

Co-Variance Matrix



S_{ij} = i^{th} row j^{th} col ele in S.

x_{ij} = j^{th} feature for i^{th} data-point

$$S_{ij} = cov(f_i, f_j)$$

i : 1 → d

j : 1 → d

$$cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

$$cov(x,x) = var.(x) \quad -①$$

$$cov(f_i, f_j) = cov(f_j, f_i) \quad -② \rightarrow s_{ij} = s_{ji}$$

Variance of features

Square → Symmetric matrix

$$S = \begin{bmatrix} s_{11} & s_{12} & s_{13} & \cdots & s_{1d} \\ s_{21} & s_{22} & s_{23} & \cdots & s_{2d} \\ \vdots & & & & \\ s_{d1} & & & & s_{dd} \end{bmatrix}_{d\times d}$$

(s_{ij} and s_{ji} are marked)

$$X = \begin{bmatrix} f_1 & f_2 & \cdots & f_i & f_j & \cdots & f_d \\ x_{1i} & & & & x_{1j} & & \\ x_{2i} & & & & x_{2j} & & \\ \vdots & & & & & & \\ x_{ni} & & & & & & \end{bmatrix}_{n\times d}$$

Let, X: col. standardization $\Rightarrow$ mean $\{f_i\} = 0$

$\Rightarrow$ std-dev $\{f_i\} = 1$

$$cov(f_1, f_2) = \frac{1}{n}\sum_{i=1}^{n}(x_{i_1} - \mu_1)(x_{i_2} - \mu_2)$$

(μ_1 → mean(f_1), μ_2 → mean(f_2))

$$= \frac{1}{n}\sum_{i=1}^{n} x_{i_1} * x_{i_2}$$

$$cov(f_1, f_2) = \underbrace{(f_1^T f_2)}_{f_1 \cdot f_2} * \frac{1}{n}$$

$$X = \begin{bmatrix} f_1 & f_2 & f_3 & \cdots & f_i & f_j & \cdots & f_d \\ \bullet & \bullet & & & & & & \\ \checkmark & \checkmark & & & & & & \\ * & * & & & & & & \\ x_{i_1} & x_{i_2} & & & & & & \\ & & & & & & & \\ & & & & & & & \end{bmatrix}_{n\times d}$$

If f_1 & f_2 have been standardize,

$$cov(f_1, f_2) = \frac{f_1^T f_2}{n}$$

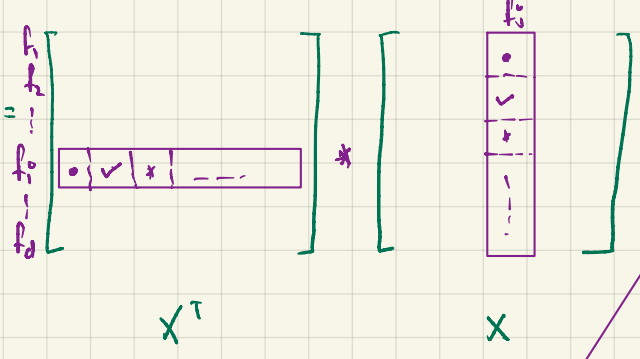
argue

$$S_{d\times d} = \frac{1}{n}\underbrace{(X^T)}_{d\times n}\underbrace{(X)}_{n\times d}{}_{n\times d} = ()_{d\times d}$$

(X → data-matrix)

→ here I am assuming that 'X' has been col. stdardize.

$S_{ij} = cov(f_i, f_j) = \frac{f_i^T f_i}{n}$

$(i,j) =$ 

$S_{ij} = \frac{f_i^T f_j}{n}$

Biased Estimator

A biased estimator systematically overestimates or underestimates the parameter it is trying to estimate. In the context of covariance:

- **Covariance Matrix (Biased Estimator):**

$$\text{Cov}(X) = \frac{1}{n} X^T X$$

where n is the number of samples.

- The biased estimator uses n in the denominator. This makes the calculation slightly more efficient but introduces bias, meaning the estimate might not be centered around the true population parameter, especially for small sample sizes.

Unbiased Estimator

An unbiased estimator, on average, hits the true parameter it estimates, meaning the expected value of the estimator is equal to the true value of the parameter.

- **Covariance Matrix (Unbiased Estimator):**

$$\text{Cov}(X) = \frac{1}{n-1} X^T X \longrightarrow \text{numpy}$$

where n is the number of samples.

- The unbiased estimator uses $n - 1$ in the denominator. This adjustment corrects the bias that occurs when estimating the population parameter from a sample. It ensures that the average of the estimates across many samples equals the true population parameter.

Why the Difference Matters

1. **Small Sample Sizes:**
   - With small sample sizes, the difference between dividing by n and $n - 1$ can be significant. The $n - 1$ adjustment helps counteract the tendency to underestimate the variance and covariance.

by default uses this.

$S_{dxd} = X^T_{dxn} X_{nxd}$ if x has been col. std.

MNIST dataset (784 features)

Let, → 60k training data points

10k Test datapoints.

data points ← $D = \{x_i, y_i\}_{i=1}^{60k}$

x_i: 28 × 28

$y_i \in \{0, 1, 2, 3, 4, \ldots 9\}$ → classes/label

Obj:- Given a new image, determine whether a user has written {0, 1, 2 ... 9}.

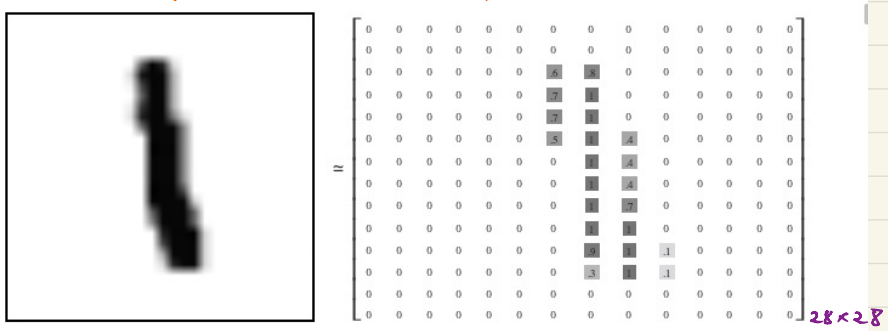
OR

classify the written character into one of the '10' numeric classes.

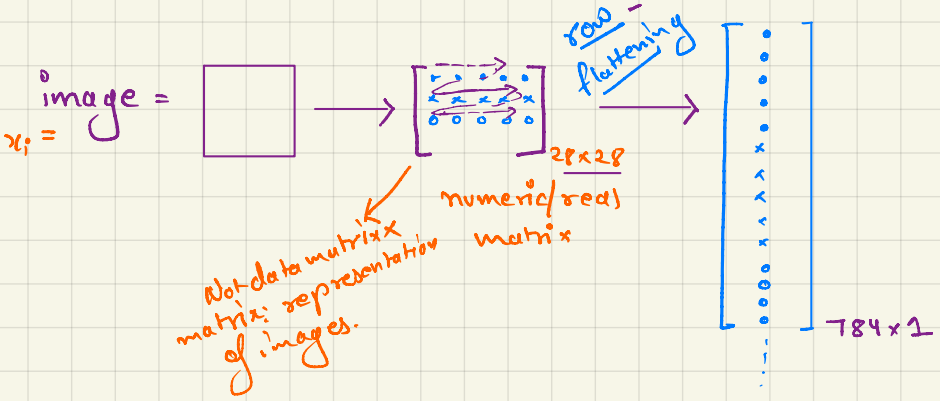
$x_i = [\quad]_{28 \times 28}$ → How do, I convert it into a image.

Every MNIST data point, every image, can be thought of as an array of numbers describing how dark each pixel is. For example, we might think of 1 as something like:

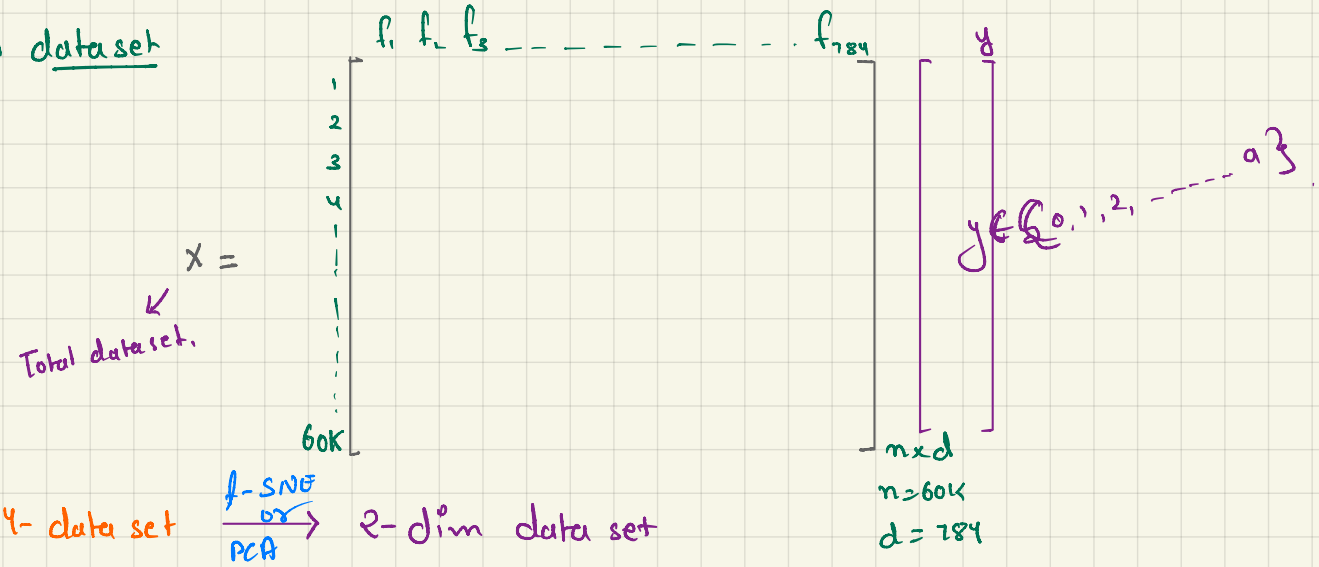
image to matrix



Since each image has 28 by 28 pixels, we get a 28x28 array. We can flatten each array into a $28 * 28 = 784$ dimensional vector. Each component of the vector is a value between zero and one describing the intensity of the pixel. Thus, we generally think of MNIST as being a collection of 784-dimensional vectors.



Overall dataset



Code to Load Mnist

CSV
comma separate value

f_1 f_2 f_3 f_4 → header
x_1: 2 , 2.1 , 2.3 , 2.4
x_2:
x_3:

```
# MNIST dataset downloaded from Kaggle :
#https://www.kaggle.com/c/digit-recognizer/data

# Functions to read and show images.

from google.colab import files
uploaded = files.upload()
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

d0 = pd.read_csv('mnist_train.csv')

print(d0.head(5)) # print first five rows of d0.

# save the labels into a variable l.
l = d0['label']  → separating class/label from csv file.

# Drop the label feature and store the pixel data in d.
d = d0.drop('label',axis=1)
```

Choose Files No file chosen

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	→ header
0	1	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	

	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	\
0	0	...	0	0	0	0	0	0	
1	0	...	0	0	0	0	0	0	
2	0	...	0	0	0	0	0	0	
3	0	...	0	0	0	0	0	0	
4	0	...	0	0	0	0	0	0	

	pixel780	pixel781	pixel782	pixel783
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

[5 rows x 785 columns]

printing first five rows.

index

```
print(d.shape)
print(l.shape)

# display or plot a number.
plt.figure(figsize=(7,7))
idx = 1

grid_data = d.iloc[idx].to_numpy().reshape(28,28)  # reshape from 1d to 2d pixel array
plt.imshow(grid_data, interpolation = "none", cmap = "gray")
plt.show()

print(l[idx])
```

print(d.shape) → 42000, 784
print(l.shape) → 42000,

→ image show

→ colour map

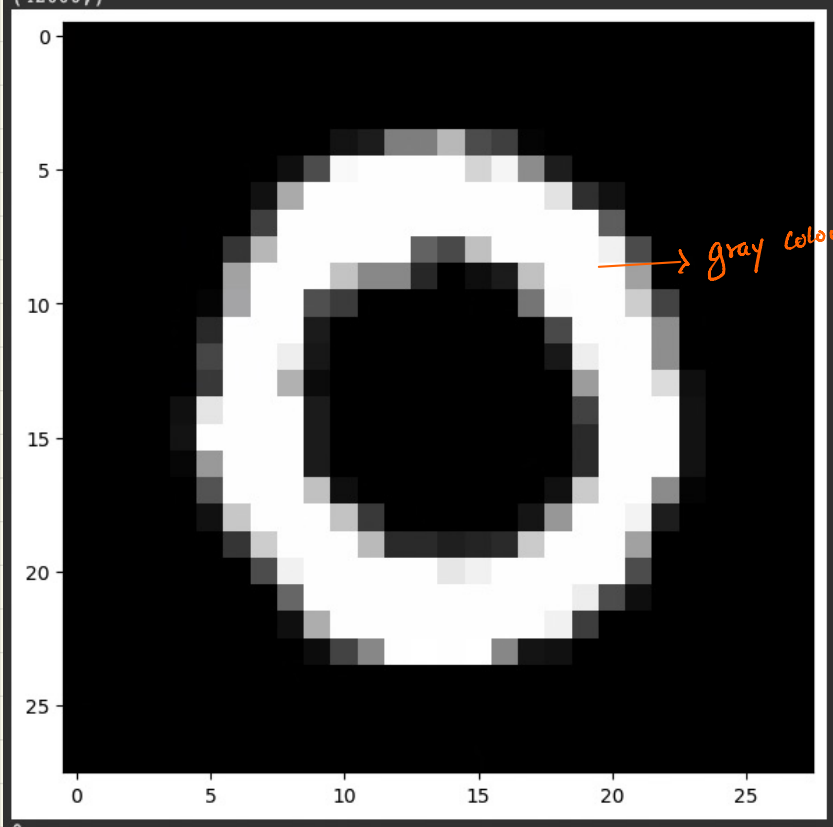
→ get the data of '1' row & convert it into '28 x 28' matrix.

→ Series.to_numpy()

↓

it is used to convert 'series' to numpy array. or index

```
(42000, 784)
(42000,)
```



→ gray colour

```
0
```

head() in pandas

DataFrame.head(n)

DataFrame.head(*n=5*) [source]

Return the first *n* rows.

This function returns the first *n* rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

For negative values of *n*, this function returns all rows except the last *|n|* rows, equivalent to `df[:n]`.

If n is larger than the number of rows, this function returns all rows.

Parameters: **n** : *int, default 5*
Number of rows to select.

Returns: **same type as caller**
The first *n* rows of the caller object.

See also

`DataFrame.tail`
Returns the last *n* rows.

Examples

```
>>> df = pd.DataFrame({'animal': ['alligator', 'bee', 'falcon', 'lion',
...                    'monkey', 'parrot', 'shark', 'whale', 'zebra']})
>>> df
      animal
0  alligator
1        bee
2     falcon
3       lion
4     monkey
5     parrot
6      shark
7      whale
8      zebra
```

Viewing the first 5 lines

```
>>> df.head()
      animal
0  alligator
1        bee
2     falcon
3       lion
4     monkey
```

Viewing the first *n* lines (three in this case)

```
>>> df.head(3)
      animal
0  alligator
1        bee
2     falcon
```

For negative values of *n*

```
>>> df.head(-3)
      animal
0  alligator
1        bee
2     falcon
3       lion
4     monkey
5     parrot
```

DataFrame.drop( )

pandas.DataFrame.drop

DataFrame.drop(*labels=None, *, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise'*) [source]

Drop specified labels from rows or columns.

Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. When using a multi-index, labels on different levels can be removed by specifying the level. See the user guide for more information about the now unused levels.

Parameters:

labels : *single label or list-like*
Index or column labels to drop. A tuple will be used as a single label and not treated as a list-like.

axis : *{0 or 'index', 1 or 'columns'}, default 0*
Whether to drop labels from the index (0 or 'index') or columns (1 or 'columns').

index : *single label or list-like*
Alternative to specifying axis (`labels, axis=0` is equivalent to `index=labels`).

columns : *single label or list-like*
Alternative to specifying axis (`labels, axis=1` is equivalent to `columns=labels`).

level : *int or level name, optional*
For MultiIndex, level from which the labels will be removed.

inplace : *bool, default False*
If False, return a copy. Otherwise, do operation inplace and return None.

errors : *{'ignore', 'raise'}, default 'raise'*
If 'ignore', suppress error and only existing labels are dropped.

Returns:

DataFrame or None
DataFrame without the removed index or column labels or None if `inplace=True`.

Examples

```
>>> df = pd.DataFrame(np.arange(12).reshape(3, 4),
...                   columns=['A', 'B', 'C', 'D'])
>>> df
   A  B   C   D
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
```

Drop columns

```
>>> df.drop(['B', 'C'], axis=1)
   A   D
0  0   3
1  4   7
2  8  11
```

```
>>> df.drop(columns=['B', 'C'])
   A   D
0  0   3
1  4   7
2  8  11
```

Drop a row by index

```
>>> df.drop([0, 1])
   A  B   C   D
2  8  9  10  11
```

Drop columns and/or rows of MultiIndex DataFrame

```
>>> midx = pd.MultiIndex(levels=[['lama', 'cow', 'falcon'],
...                              ['speed', 'weight', 'length']],
...                      codes=[[0, 0, 0, 1, 1, 1, 2, 2, 2],
...                             [0, 1, 2, 0, 1, 2, 0, 1, 2]])
>>> df = pd.DataFrame(index=midx, columns=['big', 'small'],
...                   data=[[45, 30], [200, 100], [1.5, 1], [30, 20],
...                         [250, 150], [1.5, 0.8], [320, 250],
...                         [1, 0.8], [0.3, 0.2]])
>>> df
                big     small
lama    speed   45.0    30.0
        weight  200.0   100.0
        length  1.5     1.0
cow     speed   30.0    20.0
        weight  250.0   150.0
        length  1.5     0.8
falcon  speed   320.0   250.0
        weight  1.0     0.8
        length  0.3     0.2
```

Drop a specific index combination from the MultiIndex DataFrame, i.e., drop the combination `'falcon'` and `'weight'`, which deletes only the corresponding row

```
>>> df.drop(index=('falcon', 'weight'))
                big     small
lama    speed   45.0    30.0
        weight  200.0   100.0
        length  1.5     1.0
cow     speed   30.0    20.0
        weight  250.0   150.0
        length  1.5     0.8
falcon  speed   320.0   250.0
        length  0.3     0.2
```

```
>>> df.drop(index='cow', columns='small')
                big
lama    speed   45.0
        weight  200.0
        length  1.5
falcon  speed   320.0
        weight  1.0
        length  0.3
```