**Installation**

```
pip install d2l==1.0.3
```

```
Collecting d2l==1.0.3
  Downloading d2l-1.0.3-py3-none-any.whl.metadata (556 bytes)
Collecting jupyter==1.0.0 (from d2l==1.0.3)
  Downloading jupyter-1.0.0-py2.py3-none-any.whl.metadata (995 bytes)
Collecting numpy==1.23.5 (from d2l==1.0.3)
  Downloading numpy-1.23.5-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.3 kB)
Collecting matplotlib==3.7.2 (from d2l==1.0.3)
  Downloading matplotlib-3.7.2-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.6 kB)
Collecting matplotlib-inline==0.1.6 (from d2l==1.0.3)
  Downloading matplotlib_inline-0.1.6-py3-none-any.whl.metadata (2.8
kB)
Collecting requests==2.31.0 (from d2l==1.0.3)
  Downloading requests-2.31.0-py3-none-any.whl.metadata (4.6 kB)
Collecting pandas==2.0.3 (from d2l==1.0.3)
  Downloading pandas-2.0.3-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Collecting scipy==1.10.1 (from d2l==1.0.3)
  Downloading scipy-1.10.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (58 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.9/58.9 kB 3.6 MB/s eta
0:00:00
ent already satisfied: notebook in /usr/local/lib/python3.10/dist-
packages (from jupyter==1.0.0->d2l==1.0.3) (6.5.5)
Collecting qtconsole (from jupyter==1.0.0->d2l==1.0.3)
  Downloading qtconsole-5.6.0-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: jupyter-console in
/usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0-
>d2l==1.0.3) (6.1.0)
Requirement already satisfied: nbconvert in
/usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0-
>d2l==1.0.3) (6.5.4)
Requirement already satisfied: ipykernel in
/usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0-
>d2l==1.0.3) (5.5.6)
Requirement already satisfied: ipywidgets in
/usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0-
>d2l==1.0.3) (7.7.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2-
>d2l==1.0.3) (1.3.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2-
>d2l==1.0.3) (0.12.1)
```

```
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2-
>d2l==1.0.3) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2-
>d2l==1.0.3) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2-
>d2l==1.0.3) (24.1)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2-
>d2l==1.0.3) (10.4.0)
Collecting pyparsing<3.1,>=2.3.1 (from matplotlib==3.7.2->d2l==1.0.3)
  Downloading pyparsing-3.0.9-py3-none-any.whl.metadata (4.2 kB)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2-
>d2l==1.0.3) (2.8.2)
Requirement already satisfied: traitlets in
/usr/local/lib/python3.10/dist-packages (from matplotlib-
inline==0.1.6->d2l==1.0.3) (5.7.1)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas==2.0.3-
>d2l==1.0.3) (2024.2)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas==2.0.3-
>d2l==1.0.3) (2024.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests==2.31.0-
>d2l==1.0.3) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests==2.31.0-
>d2l==1.0.3) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests==2.31.0-
>d2l==1.0.3) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests==2.31.0-
>d2l==1.0.3) (2024.8.30)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib==3.7.2->d2l==1.0.3) (1.16.0)
Requirement already satisfied: ipython-genutils in
/usr/local/lib/python3.10/dist-packages (from ipykernel-
>jupyter==1.0.0->d2l==1.0.3) (0.2.0)
Requirement already satisfied: ipython>=5.0.0 in
/usr/local/lib/python3.10/dist-packages (from ipykernel-
>jupyter==1.0.0->d2l==1.0.3) (7.34.0)
Requirement already satisfied: jupyter-client in
/usr/local/lib/python3.10/dist-packages (from ipykernel-
```

```
>jupyter==1.0.0->d2l==1.0.3) (6.1.12)
Requirement already satisfied: tornado>=4.2 in
/usr/local/lib/python3.10/dist-packages (from ipykernel-
>jupyter==1.0.0->d2l==1.0.3) (6.3.3)
Requirement already satisfied: widgetsnbextension~=3.6.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets-
>jupyter==1.0.0->d2l==1.0.3) (3.6.9)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets-
>jupyter==1.0.0->d2l==1.0.3) (3.0.13)
Requirement already satisfied: prompt-toolkit!=3.0.0,!
=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from
jupyter-console->jupyter==1.0.0->d2l==1.0.3) (3.0.47)
Requirement already satisfied: pygments in
/usr/local/lib/python3.10/dist-packages (from jupyter-console-
>jupyter==1.0.0->d2l==1.0.3) (2.18.0)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-
packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (4.9.4)
Requirement already satisfied: beautifulsoup4 in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (4.12.3)
Requirement already satisfied: bleach in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (6.1.0)
Requirement already satisfied: defusedxml in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (0.4)
Requirement already satisfied: jinja2>=3.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (3.1.4)
Requirement already satisfied: jupyter-core>=4.7 in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (0.10.0)
Requirement already satisfied: nbformat>=5.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (5.10.4)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (1.5.1)
Requirement already satisfied: tinycss2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (1.3.0)
Requirement already satisfied: pyzmq<25,>=17 in
/usr/local/lib/python3.10/dist-packages (from notebook-
>jupyter==1.0.0->d2l==1.0.3) (24.0.1)
Requirement already satisfied: argon2-cffi in
/usr/local/lib/python3.10/dist-packages (from notebook-
>jupyter==1.0.0->d2l==1.0.3) (23.1.0)
Requirement already satisfied: nest-asyncio>=1.5 in
/usr/local/lib/python3.10/dist-packages (from notebook-
>jupyter==1.0.0->d2l==1.0.3) (1.6.0)
Requirement already satisfied: Send2Trash>=1.8.0 in
/usr/local/lib/python3.10/dist-packages (from notebook-
>jupyter==1.0.0->d2l==1.0.3) (1.8.3)
Requirement already satisfied: terminado>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from notebook-
>jupyter==1.0.0->d2l==1.0.3) (0.18.1)
Requirement already satisfied: prometheus-client in
/usr/local/lib/python3.10/dist-packages (from notebook-
>jupyter==1.0.0->d2l==1.0.3) (0.20.0)
Requirement already satisfied: nbclassic>=0.4.7 in
/usr/local/lib/python3.10/dist-packages (from notebook-
>jupyter==1.0.0->d2l==1.0.3) (1.1.0)
Collecting qtpy>=2.4.0 (from qtconsole->jupyter==1.0.0->d2l==1.0.3)
  Downloading QtPy-2.4.1-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0-
>ipykernel->jupyter==1.0.0->d2l==1.0.3) (71.0.4)
Collecting jedi>=0.16 (from ipython>=5.0.0->ipykernel->jupyter==1.0.0-
>d2l==1.0.3)
  Using cached jedi-0.19.1-py2.py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: decorator in
/usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0-
>ipykernel->jupyter==1.0.0->d2l==1.0.3) (4.4.2)
Requirement already satisfied: pickleshare in
/usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0-
>ipykernel->jupyter==1.0.0->d2l==1.0.3) (0.7.5)
Requirement already satisfied: backcall in
/usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0-
>ipykernel->jupyter==1.0.0->d2l==1.0.3) (0.2.0)
Requirement already satisfied: pexpect>4.3 in
/usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0-
>ipykernel->jupyter==1.0.0->d2l==1.0.3) (4.9.0)
```

```
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.7-
>nbconvert->jupyter==1.0.0->d2l==1.0.3) (4.3.6)
Requirement already satisfied: notebook-shim>=0.2.3 in
/usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7-
>notebook->jupyter==1.0.0->d2l==1.0.3) (0.2.4)
Requirement already satisfied: fastjsonschema>=2.15 in
/usr/local/lib/python3.10/dist-packages (from nbformat>=5.1-
>nbconvert->jupyter==1.0.0->d2l==1.0.3) (2.20.0)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from nbformat>=5.1-
>nbconvert->jupyter==1.0.0->d2l==1.0.3) (4.23.0)
Requirement already satisfied: wcwidth in
/usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!
=3.0.1,<3.1.0,>=2.0.0->jupyter-console->jupyter==1.0.0->d2l==1.0.3)
(0.2.13)
Requirement already satisfied: ptyprocess in
/usr/local/lib/python3.10/dist-packages (from terminado>=0.8.3-
>notebook->jupyter==1.0.0->d2l==1.0.3) (0.7.0)
Requirement already satisfied: argon2-cffi-bindings in
/usr/local/lib/python3.10/dist-packages (from argon2-cffi->notebook-
>jupyter==1.0.0->d2l==1.0.3) (21.2.0)
Requirement already satisfied: soupsieve>1.2 in
/usr/local/lib/python3.10/dist-packages (from beautifulsoup4-
>nbconvert->jupyter==1.0.0->d2l==1.0.3) (2.6)
Requirement already satisfied: webencodings in
/usr/local/lib/python3.10/dist-packages (from bleach->nbconvert-
>jupyter==1.0.0->d2l==1.0.3) (0.5.1)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from jedi>=0.16-
>ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3) (0.8.4)
Requirement already satisfied: attrs>=22.2.0 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3) (24.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.20.0)
Requirement already satisfied: jupyter-server<3,>=1.8 in
/usr/local/lib/python3.10/dist-packages (from notebook-shim>=0.2.3-
>nbclassic>=0.4.7->notebook->jupyter==1.0.0->d2l==1.0.3) (1.24.0)
Requirement already satisfied: cffi>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from argon2-cffi-bindings-
>argon2-cffi->notebook->jupyter==1.0.0->d2l==1.0.3) (1.17.1)
```

```
Requirement already satisfied: pycparser in
/usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-
cffi-bindings->argon2-cffi->notebook->jupyter==1.0.0->d2l==1.0.3)
(2.22)
Requirement already satisfied: anyio<4,>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8-
>notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook->jupyter==1.0.0-
>d2l==1.0.3) (3.7.1)
Requirement already satisfied: websocket-client in
/usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8-
>notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook->jupyter==1.0.0-
>d2l==1.0.3) (1.8.0)
Requirement already satisfied: sniffio>=1.1 in
/usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0-
>jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7-
>notebook->jupyter==1.0.0->d2l==1.0.3) (1.3.1)
Requirement already satisfied: exceptiongroup in
/usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0-
>jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7-
>notebook->jupyter==1.0.0->d2l==1.0.3) (1.2.2)
Downloading d2l-1.0.3-py3-none-any.whl (111 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 111.7/111.7 kB 6.9 MB/s eta
0:00:00
atplotlib-3.7.2-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.6/11.6 MB 47.8 MB/s eta
0:00:00
atplotlib_inline-0.1.6-py3-none-any.whl (9.4 kB)
Downloading numpy-1.23.5-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.1/17.1 MB 50.5 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.3/12.3 MB 62.0 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.6/62.6 kB 5.2 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (34.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34.4/34.4 MB 17.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.3/98.3 kB 7.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 124.7/124.7 kB 10.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 93.5/93.5 kB 6.9 MB/s eta
0:00:00
py, matplotlib-inline, jedi, scipy, pandas, matplotlib, qtconsole,
jupyter, d2l
  Attempting uninstall: requests
```

```
    Found existing installation: requests 2.32.3
    Uninstalling requests-2.32.3:
      Successfully uninstalled requests-2.32.3
  Attempting uninstall: pyparsing
    Found existing installation: pyparsing 3.1.4
    Uninstalling pyparsing-3.1.4:
      Successfully uninstalled pyparsing-3.1.4
  Attempting uninstall: numpy
    Found existing installation: numpy 1.26.4
    Uninstalling numpy-1.26.4:
      Successfully uninstalled numpy-1.26.4
  Attempting uninstall: matplotlib-inline
    Found existing installation: matplotlib-inline 0.1.7
    Uninstalling matplotlib-inline-0.1.7:
      Successfully uninstalled matplotlib-inline-0.1.7
  Attempting uninstall: scipy
    Found existing installation: scipy 1.13.1
    Uninstalling scipy-1.13.1:
      Successfully uninstalled scipy-1.13.1
  Attempting uninstall: pandas
    Found existing installation: pandas 2.1.4
    Uninstalling pandas-2.1.4:
      Successfully uninstalled pandas-2.1.4
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.7.1
    Uninstalling matplotlib-3.7.1:
      Successfully uninstalled matplotlib-3.7.1
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
albucore 0.0.16 requires numpy>=1.24, but you have numpy 1.23.5 which
is incompatible.
albumentations 1.4.15 requires numpy>=1.24.4, but you have numpy
1.23.5 which is incompatible.
bigframes 1.17.0 requires numpy>=1.24.0, but you have numpy 1.23.5
which is incompatible.
chex 0.1.86 requires numpy>=1.24.1, but you have numpy 1.23.5 which is
incompatible.
google-colab 1.0.0 requires pandas==2.1.4, but you have pandas 2.0.3
which is incompatible.
google-colab 1.0.0 requires requests==2.32.3, but you have requests
2.31.0 which is incompatible.
mizani 0.11.4 requires pandas>=2.1.0, but you have pandas 2.0.3 which
is incompatible.
pandas-stubs 2.1.4.231227 requires numpy>=1.26.0; python_version <
"3.13", but you have numpy 1.23.5 which is incompatible.
plotnine 0.13.6 requires pandas<3.0.0,>=2.1.0, but you have pandas
2.0.3 which is incompatible.
xarray 2024.9.0 requires numpy>=1.24, but you have numpy 1.23.5 which
is incompatible.
```

```
xarray 2024.9.0 requires pandas>=2.1, but you have pandas 2.0.3 which
is incompatible.
Successfully installed d2l-1.0.3 jedi-0.19.1 jupyter-1.0.0 matplotlib-
3.7.2 matplotlib-inline-0.1.6 numpy-1.23.5 pandas-2.0.3 pyparsing-
3.0.9 qtconsole-5.6.0 qtpy-2.4.1 requests-2.31.0 scipy-1.10.1
```

{"id":"2ce8df78af11461a8648f546b63ee39f","pip_warning":{"packages":
["matplotlib","matplotlib_inline","mpl_toolkits","numpy"]}}

### 2.1 Data manipulation

```
import torch

x = torch.arange(12, dtype=torch.float32)
x
```

```
tensor([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.])
```

```
x.numel()
```

```
12
```

```
x.shape
```

```
torch.Size([12])
```

```
X = x.reshape(3, 4)
X
```

```
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.]])
```

```
torch.zeros((2, 3, 4))
```

```
tensor([[[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]]])
```

```
torch.ones((2, 3, 4))
```

```
tensor([[[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]],

        [[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]]])
```

```
torch.randn(3, 4)

tensor([[ 1.4329,  1.1178, -0.6164,  1.7440],
        [ 0.9816, -0.1129, -0.5310, -1.5249],
        [ 1.9685, -0.7598, -0.4390, -0.5409]])

torch.tensor([[2, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])

tensor([[2, 1, 4, 3],
        [1, 2, 3, 4],
        [4, 3, 2, 1]])

X[-1]

tensor([ 8.,  9., 10., 11.])

 X[1:3]

tensor([[ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.]])

X[1, 2] = 17
X

tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5., 17.,  7.],
        [ 8.,  9., 10., 11.]])

X[:2, :] = 12
X

tensor([[12., 12., 12., 12.],
        [12., 12., 12., 12.],
        [ 8.,  9., 10., 11.]])

torch.exp(x)

tensor([162754.7969, 162754.7969, 162754.7969, 162754.7969,
162754.7969,
        162754.7969, 162754.7969, 162754.7969,   2980.9580,
8103.0840,
         22026.4648,  59874.1406])

x = torch.tensor([1.0, 2, 4, 8])
y = torch.tensor([2, 2, 2, 2])
x + y

tensor([ 3.,  4.,  6., 10.])

x - y

tensor([-1.,  0.,  2.,  6.])
```

```python
x * y
```

```
tensor([ 2.,  4.,  8., 16.])
```

```python
x / y
```

```
tensor([0.5000, 1.0000, 2.0000, 4.0000])
```

```python
x ** y
```

```
tensor([ 1.,  4., 16., 64.])
```

```python
X = torch.arange(12, dtype=torch.float32).reshape((3,4))
Y = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
torch.cat((X, Y), dim=0)
```

```
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.],
        [ 2.,  1.,  4.,  3.],
        [ 1.,  2.,  3.,  4.],
        [ 4.,  3.,  2.,  1.]])
```

```python
torch.cat((X, Y), dim=1)
```

```
tensor([[ 0.,  1.,  2.,  3.,  2.,  1.,  4.,  3.],
        [ 4.,  5.,  6.,  7.,  1.,  2.,  3.,  4.],
        [ 8.,  9., 10., 11.,  4.,  3.,  2.,  1.]])
```

```python
X == Y
```

```
tensor([[False,  True, False,  True],
        [False, False, False, False],
        [False, False, False, False]])
```

```python
X.sum()
```

```
tensor(66.)
```

```python
a = torch.arange(3).reshape((3, 1))
b = torch.arange(2).reshape((1, 2))
a, b
```

```
(tensor([[0],
         [1],
         [2]]),
 tensor([[0, 1]]))
```

```python
a + b
```

```
tensor([[0, 1],
        [1, 2],
        [2, 3]])
```

```python
before = id(Y)
Y = Y + X
id(Y) == before
```

```
False
```

```python
Z = torch.zeros_like(Y)
print('id(Z):', id(Z))
Z[:] = X + Y
print('id(Z):', id(Z))
```

```
id(Z): 134105290139312
id(Z): 134105290139312
```

```python
before = id(X)
X += Y
id(X) == before
```

```
True
```

```python
A = X.numpy()
B = torch.from_numpy(A)
type(A), type(B)
```

```
(numpy.ndarray, torch.Tensor)
```

```python
a = torch.tensor([3.5])
a, a.item(), float(a), int(a)
```

```
(tensor([3.5000]), 3.5, 3.5, 3)
```

```python
type(a.item()), type(a)
```

```
(float, torch.Tensor)
```

```python
x = torch.arange(12, dtype=torch.float32).reshape((3,4))
y = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
x,y
```

```
(tensor([[ 0.,  1.,  2.,  3.],
         [ 4.,  5.,  6.,  7.],
         [ 8.,  9., 10., 11.]]),
 tensor([[2., 1., 4., 3.],
         [1., 2., 3., 4.],
         [4., 3., 2., 1.]]))
```

```python
x == y
```

```
tensor([[False,  True, False,  True],
        [False, False, False, False],
        [False, False, False, False]])
```

```python
x < y
```

```
tensor([[ True, False,  True, False],
        [False, False, False, False],
        [False, False, False, False]])
```

```
x > y
```

```
tensor([[False, False, False, False],
        [ True,  True,  True,  True],
        [ True,  True,  True,  True]])
```

**2.2 Data preprocessing**

```
import os

os.makedirs(os.path.join('..', 'data'), exist_ok=True)
data_file = os.path.join('..', 'data', 'house_tiny.csv')
with open(data_file, 'w') as f:
    f.write('''NumRooms,RoofType,Price
NA,NA,127500
2,NA,106000
4,Slate,178100
NA,NA,140000''')

import pandas as pd

data = pd.read_csv(data_file)
print(data)
```

```
   NumRooms RoofType   Price
0       NaN      NaN  127500
1       2.0      NaN  106000
2       4.0    Slate  178100
3       NaN      NaN  140000
```

Note that here we one-hot encode only the RoofType column by specifying the columns parameter in get_dummies function.

```
inputs, targets = data.iloc[:, 0:2], data.iloc[:, 2]
inputs = pd.get_dummies(inputs, columns= ["RoofType"], dummy_na=True)
print(inputs)
#print(inputs),print(targets)
#print(pd.get_dummies(inputs, dummy_na=True))
#print(inputs.iloc[:, 4:6])
```

```
   NumRooms  RoofType_Slate  RoofType_nan
0       NaN           False          True
1       2.0           False          True
2       4.0            True         False
3       NaN           False          True
```

Executing the original code leads to "TypeError: Could not convert [' NA 2 4 NA'] to numeric". We cannot calculate the mean of column containing non-numeric values such as string with 'NA'. We convert 'NumRooms' column to numeric type.The errors='coerce' argument handles invalid parsing such as the whitespace before NA by setting them to NaN,

```
inputs['NumRooms'] = pd.to_numeric(inputs['NumRooms'],errors='coerce')

inputs = inputs.fillna(inputs.mean(skipna=True))
print(inputs)
```

```
   NumRooms  RoofType_Slate  RoofType_nan
0       3.0           False          True
1       2.0           False          True
2       4.0            True         False
3       3.0           False          True
```

```
import torch

X = torch.tensor(inputs.to_numpy(dtype=float))
y = torch.tensor(targets.to_numpy(dtype=float))
X, y
```

```
(tensor([[3., 0., 1.],
         [2., 0., 1.],
         [4., 1., 0.],
         [3., 0., 1.]], dtype=torch.float64),
 tensor([127500., 106000., 178100., 140000.], dtype=torch.float64))
```

**Pandas tutorial**

```
import numpy as np

import pandas as pd

s = pd.Series([1, 3, 5, np.nan, 6, 8])
s
```

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

```
dates = pd.date_range("20130101", periods=6)
dates
```

```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

```
df = pd.DataFrame(np.random.randn(6, 4), index=dates,
columns=list("ABCD"))
df
```


{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.5855083983853894,\n
\"min\": -0.16616359127929095,\n        \"max\": 1.4521329600781396,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
1.1602775319225755,\n          0.28417744419985536,\n
0.72718473544741\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.3959563277654459,\n        \"min\": -1.2849716232500534,\n
\"max\": 2.618101891928222,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          -0.2705845324801159,\n          -
0.48233376443626313,\n          -1.2849716232500534\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.8345439571317859,\n
\"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
0.7083114693504143,\n          0.9296359944796136,\n          -
0.24932038458473663\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.6214507888973646,\n        \"min\": -2.6053105851958795,\n
\"max\": 1.8296929996762517,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1.3408881089843163,\n
0.8260543348634037,\n          -0.00974908882497 91\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}


```
df2 = pd.DataFrame(
    {
        "A": 1.0,
        "B": pd.Timestamp("20130102"),
        "C": pd.Series(1, index=list(range(4)), dtype="float32"),
        "D": np.array([3] * 4, dtype="int32"),
        "E": pd.Categorical(["test", "train", "test", "train"]),
        "F": "foo",
    }
)
df2
```


{"summary":"{\n  \"name\": \"df2\",\n  \"rows\": 4,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\": 1.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          1.0\n        ],\n        \"semantic_type\": \"\",\n

\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": \"2013-01-02 00:00:00\",\n        \"max\": \"2013-01-02
00:00:00\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"2013-01-02 00:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"float32\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          1.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"D\",\n      \"properties\": {\n        \"dtype\":
\"int32\",\n        \"num_unique_values\": 1,\n        \"samples\": [\
n          3\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"E\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"train\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"F\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          \"foo\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df2"}

```
df.dtypes
```

```
A    float64
B    float64
C    float64
D    float64
dtype: object
```

```
df2.dtypes
```

```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

```
df.head()
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.5855083983853894,\n
\"min\": -0.16616359127929095,\n        \"max\": 1.4521329600781396,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
1.1602775319225755,\n          0.28417744419985536,\n
0.72718473544741\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":

```
\"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.3959563277654459,\n        \"min\": -1.2849716232500534,\n
\"max\": 2.618101891928222,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          -0.2705845324801159,\n          -
0.48233376443626313,\n          -1.2849716232500534\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.8345439571317859,\n
\"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
0.7083114693504143,\n          0.9296359944796136,\n          -
0.24932038458473663\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.6214507888973646,\n        \"min\": -2.6053105851958795,\n
\"max\": 1.8296929996762517,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1.3408881089843163,\n
0.8260543348634037,\n          -0.0097490888249791\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```



```
df.tail(3)
```



```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 3,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.47180459312183537,\n
\"min\": 0.5665562995218522,\n        \"max\": 1.4521329600781396,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.5665562995218522,\n          1.4521329600781396,\n
0.72718473544741\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.5979837394566896,\n        \"min\": -1.2849716232500534,\n
\"max\": -0.13119792011918546,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -0.13119792011918546,\n          -
0.9807720274128267,\n          -1.2849716232500534\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.3950021938281266,\n
\"min\": -0.24932038458473663,\n        \"max\": 0.5259242173191606,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.006687421148937388,\n          0.5259242173191606,\n          -
0.24932038458473663\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.9500028932730334,\n        \"min\": -0.0097490888249791,\n
\"max\": 1.8296929996762517,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.3221079693446776,\n
1.8296929996762517,\n          -0.0097490888249791\n        ],\n
```

\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

```
df.index
```

```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

```
df.columns
```

```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

```
df.to_numpy()
```

```
array([[ 1.16027753, -0.27058453,  0.70831147,  1.34088811],
       [ 0.28417744, -0.48233376,  0.92963599,  0.82605433],
       [-0.16616359,  2.61810189, -1.35327458, -2.60531059],
       [ 0.5665563 , -0.13119792,  0.00668742,  1.32210797],
       [ 1.45213296, -0.98077203,  0.52592422,  1.829693  ],
       [ 0.72718474, -1.28497162, -0.24932038, -0.00974909]])
```

```
df2.dtypes
```

```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

```
df2.to_numpy()
```

```
array([[1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train',
'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train',
'foo']],
      dtype=object)
```

```
df.describe()
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.947380901351536,\n
\"min\": -0.16616359127929095,\n        \"max\": 6.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
0.6706942299817569,\n          0.6468705174846311,\n          6.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n

\"dtype\": \"number\",\n        \"std\": 2.414510214806282,\n        \"min\": -1.2849716232500534,\n        \"max\": 6.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          -0.08862632929503707,\n          -0.3764591484581895,\n          6.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 2.1835902887016347,\n        \"min\": -1.3532745772468233,\n        \"max\": 6.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          0.09466069007776101,\n          0.266305819234049,\n          6.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 2.3808883313361435,\n        \"min\": -2.6053105851958795,\n        \"max\": 6.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n          0.4506139564746317,\n          1.0740811521040405,\n          6.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

```
df.T
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 4,\n  \"fields\": [\n    {\n      \"column\": \"2013-01-01 00:00:00\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.7210810409383586,\n        \"min\": -0.2705845324801159,\n        \"max\": 1.3408881089843163,\n        \"num_unique_values\": 4,\n        \"samples\": [\n          -0.2705845324801159,\n          1.3408881089843163,\n          1.1602775319225755\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"2013-01-02 00:00:00\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.6464037641972217,\n        \"min\": -0.48233376443626313,\n        \"max\": 0.9296359944796136,\n        \"num_unique_values\": 4,\n        \"samples\": [\n          -0.48233376443626313,\n          0.8260543348634037,\n          0.28417744419985536\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"2013-01-03 00:00:00\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 2.231110858653373,\n        \"min\": -2.6053105851958795,\n        \"max\": 2.618101891928222,\n        \"num_unique_values\": 4,\n        \"samples\": [\n          2.618101891928222,\n          -2.6053105851958795,\n          -0.16616359127929095\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"2013-01-04 00:00:00\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.6603417064969745,\n        \"min\": -0.13119792011918546,\n        \"max\": 1.3221079693446776,\n        \"num_unique_values\": 4,\n        \"samples\": [\n          -0.13119792011918546,\n          1.3221079693446776,\n          0.5665562995218522\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\

n    },\n    {\n      \"column\": \"2013-01-05 00:00:00\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.2512693946899183,\n        \"min\": -0.9807720274128267,\n
\"max\": 1.8296929996762517,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          -0.9807720274128267,\n
1.8296929996762517,\n          1.4521329600781396\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"2013-01-06 00:00:00\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.8317435165992622,\n        \"min\": -1.2849716232500534,\n
\"max\": 0.72718473544741,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          -1.2849716232500534,\n          -
0.0097490888249791,\n          0.72718473544741\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

```
df.sort_index(axis=1, ascending=False)
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"D\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.6214507888973646,\n
\"min\": -2.6053105851958795,\n        \"max\": 1.8296929996762517,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
1.3408881089843163,\n          0.8260543348634037,\n          -
0.0097490888249791\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.8345439571317859,\n        \"min\": -1.3532745772468233,\n
\"max\": 0.9296359944796136,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          0.7083114693504143,\n
0.9296359944796136,\n          -0.24932038458473663\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.3959563277654459,\n
\"min\": -1.2849716232500534,\n        \"max\": 2.618101891928222,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          -
0.2705845324801159,\n          -0.48233376443626313,\n          -
1.2849716232500534\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"A\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.5855083983853894,\n        \"min\": -0.16616359127929095,\n
\"max\": 1.4521329600781396,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1.1602775319225755,\n
0.28417744419985536,\n          0.72718473544741\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

```
df.sort_values(by="B")
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.5855083983853894,\n
\"min\": -0.16616359127929095,\n        \"max\": 1.4521329600781396,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
0.72718473544741,\n          1.4521329600781396,\n          -
0.16616359127929095\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.3959563277654459,\n        \"min\": -1.2849716232500534,\n
\"max\": 2.618101891928222,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          -1.2849716232500534,\n          -
0.9807720274128267,\n          2.618101891928222\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.8345439571317859,\n
\"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          -
0.24932038458473663,\n          0.5259242173191606,\n          -
1.3532745772468233\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.6214507888973644,\n        \"min\": -2.6053105851958795,\n
\"max\": 1.8296929996762517,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          -0.0097490888249791,\n
1.8296929996762517,\n          -2.6053105851958795\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

```
df["A"]
```

```
2013-01-01    1.160278
2013-01-02    0.284177
2013-01-03   -0.166164
2013-01-04    0.566556
2013-01-05    1.452133
2013-01-06    0.727185
Freq: D, Name: A, dtype: float64
```

```
df[0:3]
```

{"summary":"{\n  \"name\": \"df[0:3]\",\n  \"rows\": 3,\n  \"fields\":
[\n    {\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.6745127210642777,\n
\"min\": -0.16616359127929095,\n        \"max\": 1.1602775319225755,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
1.1602775319225755,\n          0.28417744419985536,\n          -
0.16616359127929095\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n

\"std\": 1.7321493514826853,\n        \"min\": -0.48233376443626313,\n
\"max\": 2.618101891928222,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -0.2705845324801159,\n          -
0.48233376443626313,\n          2.618101891928222\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.2590209325403359,\n
\"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.7083114693504143,\n          0.9296359944796136,\n          -
1.3532745772468233\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2.145219614380562,\n        \"min\": -2.6053105851958795,\n
\"max\": 1.3408881089843163,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.3408881089843163,\n
0.8260543348634037,\n          -2.6053105851958795\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

```
df["20130102":"20130104"]
```

{"summary":"{\n  \"name\": \"df[\\\"20130102\\\":\\\"20130104\\\"]\",\
n  \"rows\": 3,\n  \"fields\": [\n    {\n      \"column\": \"A\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.36955453015421824,\n        \"min\": -0.16616359127929095,\n
\"max\": 0.5665562995218522,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.28417744419985536,\n          -
0.16616359127929095,\n          0.5665562995218522\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.6977753628864085,\n
\"min\": -0.48233376443626313,\n        \"max\": 2.618101891928222,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          -
0.48233376443626313,\n          2.618101891928222,\n          -
0.13119792011918546\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.148405516690718,\n        \"min\": -1.3532745772468233,\n
\"max\": 0.9296359944796136,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.9296359944796136,\n          -
1.3532745772468233,\n          0.006687421148937388\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2.1387282407015067,\n
\"min\": -2.6053105851958795,\n        \"max\": 1.3221079693446776,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.8260543348634037,\n          -2.6053105851958795,\n
1.3221079693446776\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

```
df.loc[dates[0]]
```

```
A    1.160278
B   -0.270585
C    0.708311
D    1.340888
Name: 2013-01-01 00:00:00, dtype: float64
```

```
df.loc[:, ["A", "B"]]
```


{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n    {\n      \"column\": \"A\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.5855083983853894,\n        \"min\": -0.16616359127929095,\n        \"max\": 1.4521329600781396,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          1.1602775319225755,\n          0.28417744419985536,\n          0.72718473544741\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.3959563277654459,\n        \"min\": -1.2849716232500534,\n        \"max\": 2.618101891928222,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          -0.2705845324801159,\n          -0.48233376443626313,\n          -1.2849716232500534\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}


```
df.loc["20130102":"20130104", ["A", "B"]]
```


{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 3,\n  \"fields\": [\n    {\n      \"column\": \"A\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.36955453015421824,\n        \"min\": -0.16616359127929095,\n        \"max\": 0.5665562995218522,\n        \"num_unique_values\": 3,\n        \"samples\": [\n          0.28417744419985536,\n          -0.16616359127929095,\n          0.5665562995218522\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.6977753628864085,\n        \"min\": -0.48233376443626313,\n        \"max\": 2.618101891928222,\n        \"num_unique_values\": 3,\n        \"samples\": [\n          -0.48233376443626313,\n          2.618101891928222,\n          -0.13119792011918546\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}


```
df.loc[dates[0], "A"]
df.at[dates[0], "A"] #same but faster
```

```
1.1602775319225755
```

```
df.iloc[3]
```

```
A    0.566556
B   -0.131198
C    0.006687
D    1.322108
Name: 2013-01-04 00:00:00, dtype: float64
```

```
df.iloc[3:5, 0:2]
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 2,\n  \"fields\": [\n    {\n      \"column\": \"A\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.6261972619398882,\n        \"min\": 0.5665562995218522,\n        \"max\": 1.4521329600781396,\n        \"num_unique_values\": 2,\n        \"samples\": [\n          1.4521329600781396,\n          0.5665562995218522\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.6007396123878412,\n        \"min\": -0.9807720274128267,\n        \"max\": -0.13119792011918546,\n        \"num_unique_values\": 2,\n        \"samples\": [\n          -0.9807720274128267,\n          -0.13119792011918546\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

```
df.iloc[[1, 2, 4], [0, 2]]
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 3,\n  \"fields\": [\n    {\n      \"column\": \"A\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.8352455590465888,\n        \"min\": -0.16616359127929095,\n        \"max\": 1.4521329600781396,\n        \"num_unique_values\": 3,\n        \"samples\": [\n          0.28417744419985536,\n          -0.16616359127929095,\n          1.4521329600781396\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.218335749223548,\n        \"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n        \"num_unique_values\": 3,\n        \"samples\": [\n          0.9296359944796136,\n          -1.3532745772468233,\n          0.5259242173191606\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

```
df.iloc[1:3, :]
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 2,\n  \"fields\": [\n    {\n      \"column\": \"A\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.31843920003387594,\n        \"min\": -0.16616359127929095,\n        \"max\": 0.28417744419985536,\n        \"num_unique_values\": 2,\n        \"samples\": [\n          -0.16616359127929095,\n          0.28417744419985536\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n

\"dtype\": \"number\",\n        \"std\": 2.192339077247892,\n
\"min\": -0.48233376443626313,\n        \"max\": 2.618101891928222,\n
\"num_unique_values\": 2,\n        \"samples\": [\n
2.618101891928222,\n          -0.48233376443626313\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.6142615461102219,\n
\"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          -
1.3532745772468233,\n          0.9296359944796136\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2.426341403699555,\n
\"min\": -2.6053105851958795,\n        \"max\": 0.8260543348634037,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          -
2.6053105851958795,\n          0.8260543348634037\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}


```
df.iloc[:, 1:3]
```


{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.3959563277654459,\n
\"min\": -1.2849716232500534,\n        \"max\": 2.618101891928222,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          -
0.2705845324801159,\n          -0.48233376443626313,\n          -
1.2849716232500534\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.8345439571317859,\n        \"min\": -1.3532745772468233,\n
\"max\": 0.9296359944796136,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          0.7083114693504143,\n
0.9296359944796136,\n          -0.24932038458473663\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}


```
df.iloc[1, 1]
df.iat[1, 1] #same but faster
```

```
-0.48233376443626313
```

```
df[df["A"] > 0]
```


{"summary":"{\n  \"name\": \"df[df[\\\"A\\\"] > 0]\",\n  \"rows\": 5,\
n  \"fields\": [\n    {\n      \"column\": \"A\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.46736046240888135,\n        \"min\": 0.28417744419985536,\n
\"max\": 1.4521329600781396,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.28417744419985536,\n
0.72718473544741,\n          0.5665562995218522\n        ],\n

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.4878055834198269,\n
\"min\": -1.2849716232500534,\n        \"max\": -0.13119792011918546,\
n        \"num_unique_values\": 5,\n        \"samples\": [\n
-0.48233376443626313,\n          -1.2849716232500534,\n          -
0.13119792011918546\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.49155321072370217,\n        \"min\": -0.24932038458473663,\
n        \"max\": 0.9296359944796136,\n        \"num_unique_values\":
5,\n        \"samples\": [\n          0.9296359944796136,\n          -
0.24932038458473663,\n          0.006687421148937388\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.6962584898617896,\n
\"min\": -0.0097490888249791,\n        \"max\": 1.8296929996762517,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.8260543348634037,\n          -0.0097490888249791,\n
1.3221079693446776\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
df[df > 0]
```


```
{"summary":"{\n  \"name\": \"df[df > 0]\",\n  \"rows\": 6,\n
\"fields\": [\n    {\n      \"column\": \"A\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.46736046240888135,\n        \"min\": 0.28417744419985536,\n
\"max\": 1.4521329600781396,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.28417744419985536,\n
0.72718473544741,\n          0.5665562995218522\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
2.618101891928222,\n        \"max\": 2.618101891928222,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
2.618101891928222\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.3935892612303366,\n        \"min\": 0.006687421148937388,\n
\"max\": 0.9296359944796136,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.9296359944796136\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.4098108375204878,\n
\"min\": 0.8260543348634037,\n        \"max\": 1.8296929996762517,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.8260543348634037\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
df2 = df.copy()
df2["E"] = ["one", "one", "two", "three", "four", "three"]
df2[df2["E"].isin(["two", "four"])]
```

{"summary":"{\n  \"name\": \"df2[df2[\\\"E\\\"]\",\n  \"rows\": 2,\n
\"fields\": [\n    {\n      \"column\": \"A\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.144308465435643,\n        \"min\": -0.16616359127929095,\n
\"max\": 1.4521329600781396,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1.4521329600781396,\n          -
0.16616359127929095\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2.5447881530014635,\n        \"min\": -0.9807720274128267,\n
\"max\": 2.618101891928222,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          -0.9807720274128267,\n
2.618101891928222\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.328794210835193,\n        \"min\": -1.3532745772468233,\n
\"max\": 0.5259242173191606,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.5259242173191606,\n          -
1.3532745772468233\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3.136021109449732,\n        \"min\": -2.6053105851958795,\n
\"max\": 1.8296929996762517,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1.8296929996762517,\n          -
2.6053105851958795\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"E\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"four\",\n          \"two\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}

```
s1 = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date_range("20130102",
periods=6))
s1
```

```
2013-01-02    1
2013-01-03    2
2013-01-04    3
2013-01-05    4
2013-01-06    5
2013-01-07    6
Freq: D, dtype: int64
```

```
df["F"] = s1
df
```

```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.5855083983853894,\n
\"min\": -0.16616359127929095,\n        \"max\": 1.4521329600781396,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
1.1602775319225755,\n          0.28417744419985536,\n
0.72718473544741\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.3959563277654459,\n        \"min\": -1.2849716232500534,\n
\"max\": 2.618101891928222,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          -0.2705845324801159,\n          -
0.48233376443626313,\n          -1.2849716232500534\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.8345439571317859,\n
\"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
0.7083114693504143,\n          0.9296359944796136,\n          -
0.24932038458473663\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.6214507888973646,\n        \"min\": -2.6053105851958795,\n
\"max\": 1.8296929996762517,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1.3408881089843163,\n
0.8260543348634037,\n          -0.00974908882497 91\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"F\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.5811388300841898,\n
\"min\": 1.0,\n        \"max\": 5.0,\n        \"num_unique_values\":
5,\n        \"samples\": [\n          2.0,\n          5.0,\n
3.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


```
df.at[dates[0], "A"] = 0
df.iat[0, 1] = 0
df.loc[:, "D"] = np.array([5] * len(df))
df
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.5830721004546995,\n
\"min\": -0.16616359127929095,\n        \"max\": 1.4521329600781396,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          0.0,\n
0.28417744419985536,\n          0.72718473544741\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.3932705063131376,\n
\"min\": -1.2849716232500534,\n        \"max\": 2.618101891928222,\n
```

\"num_unique_values\": 6,\n        \"samples\": [\n          0.0,\n          -0.48233376443626313,\n          -1.2849716232500534\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.8345439571317859,\n        \"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          0.7083114693504143,\n          0.9296359944796136,\n          -0.24932038458473663\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\": 5.0,\n        \"max\": 5.0,\n        \"num_unique_values\": 1,\n        \"samples\": [\n          5.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"F\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.5811388300841898,\n        \"min\": 1.0,\n        \"max\": 5.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\n          2.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}

```
df2 = df.copy()
df2[df2 > 0] = -df2
df2
```

{"summary":"{\n  \"name\": \"df2\",\n  \"rows\": 6,\n  \"fields\": [\n    {\n      \"column\": \"A\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.5223426498073981,\n        \"min\": -1.4521329600781396,\n        \"max\": 0.0,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          0.0,\n          -0.28417744419985536,\n          -0.72718473544741\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.9675249575212089,\n        \"min\": -2.618101891928222,\n        \"max\": 0.0,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          0.0,\n          -0.48233376443626313,\n          -1.2849716232500534\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.4823484274793122,\n        \"min\": -1.3532745772468233,\n        \"max\": -0.006687421148937388,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          -0.7083114693504143,\n          -0.9296359944796136,\n          -0.24932038458473663\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\": -5.0,\n        \"max\": -5.0,\n        \"num_unique_values\": 1,\n        \"samples\": [\n          -5.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n

\"column\": \"F\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.5811388300841898,\n        \"min\": -
5.0,\n        \"max\": -1.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          -2.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df2"}

```
df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ["E"])

df1.loc[dates[0] : dates[1], "E"] = 1

df1
```

{"summary":"{\n  \"name\": \"df1\",\n  \"rows\": 4,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.32259062085091145,\n
\"min\": -0.16616359127929095,\n        \"max\": 0.5665562995218522,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.28417744419985536,\n          0.5665562995218522,\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.425919684742139,\n
\"min\": -0.48233376443626313,\n        \"max\": 2.618101891928222,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          -
0.48233376443626313,\n          -0.13119792011918546,\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0289318596873802,\n
\"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.9296359944796136,\n          0.006687421148937388,\n
0.7083114693504143\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0,\n        \"min\": 5.0,\n        \"max\": 5.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          5.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"F\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0,\n        \"min\": 1.0,\n
\"max\": 3.0,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"E\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0,\n        \"min\": 1.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          1.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df1"}

```
df1.dropna(how="any")
```

```
{"summary":"{\n  \"name\": \"df1\",\n  \"rows\": 1,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
0.28417744419985536,\n        \"max\": 0.28417744419985536,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
0.28417744419985536\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": null,\n        \"min\": -0.48233376443626313,\n
\"max\": -0.48233376443626313,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          -0.48233376443626313\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
0.9296359944796136,\n        \"max\": 0.9296359944796136,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
0.9296359944796136\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": null,\n        \"min\": 5.0,\n        \"max\": 5.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          5.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"F\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\": 1.0,\
n        \"max\": 1.0,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          1.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"E\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 1.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
df1.fillna(value=5)
```


```
{"summary":"{\n  \"name\": \"df1\",\n  \"rows\": 4,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.32259062085091145,\n
\"min\": -0.16616359127929095,\n        \"max\": 0.5665562995218522,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.28417744419985536,\n          0.5665562995218522,\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.425919684742139,\n
\"min\": -0.48233376443626313,\n        \"max\": 2.618101891928222,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          -
0.48233376443626313,\n          -0.13119792011918546,\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0289318596873802,\n
```

\"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.9296359944796136,\n          0.006687421148937388,\n
0.7083114693504143\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0,\n        \"min\": 5.0,\n        \"max\": 5.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          5.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"F\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.707825127659933,\n
\"min\": 1.0,\n        \"max\": 5.0,\n        \"num_unique_values\":
4,\n        \"samples\": [\n          1.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"E\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2.309401076758503,\n
\"min\": 1.0,\n        \"max\": 5.0,\n        \"num_unique_values\":
2,\n        \"samples\": [\n          5.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

```
pd.isna(df1)
```

{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 4,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"boolean\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          false\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"boolean\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          false\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"boolean\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          false\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n
\"dtype\": \"boolean\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          false\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"F\",\n      \"properties\": {\n
\"dtype\": \"boolean\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          false\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"E\",\n      \"properties\": {\n
\"dtype\": \"boolean\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          true\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}

```
df
```


{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n    {\n      \"column\": \"A\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.5830721004546995,\n        \"min\": -0.16616359127929095,\n        \"max\": 1.4521329600781396,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          0.0,\n          0.28417744419985536,\n          0.72718473544741\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.3932705063131376,\n        \"min\": -1.2849716232500534,\n        \"max\": 2.618101891928222,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          0.0,\n          -0.48233376443626313,\n          -1.2849716232500534\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.8345439571317859,\n        \"min\": -1.3532745772468233,\n        \"max\": 0.9296359944796136,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          0.7083114693504143,\n          0.9296359944796136,\n          -0.24932038458473663\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\": 5.0,\n        \"max\": 5.0,\n        \"num_unique_values\": 1,\n        \"samples\": [\n          5.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"F\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.5811388300841898,\n        \"min\": 1.0,\n        \"max\": 5.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\n          2.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}


```
df.mean()
```

```
A    0.477315
B   -0.043529
C    0.094661
D    5.000000
F    3.000000
dtype: float64
```

```
df.mean(axis=1)
```

```
2013-01-01    1.427078
2013-01-02    1.346296
2013-01-03    1.619733
2013-01-04    1.688409
2013-01-05    1.999457
```

```
2013-01-06    1.838579
Freq: D, dtype: float64
```

```
s = pd.Series([1, 3, 5, np.nan, 6, 8], index=dates).shift(2)
s,df.sub(s, axis="index")
```

```
(2013-01-01    NaN
 2013-01-02    NaN
 2013-01-03    1.0
 2013-01-04    3.0
 2013-01-05    5.0
 2013-01-06    NaN
 Freq: D, dtype: float64,
                    A         B         C    D    F
 2013-01-01       NaN       NaN       NaN  NaN  NaN
 2013-01-02       NaN       NaN       NaN  NaN  NaN
 2013-01-03 -1.166164  1.618102 -2.353275  4.0  1.0
 2013-01-04 -2.433444 -3.131198 -2.993313  2.0  0.0
 2013-01-05 -3.547867 -5.980772 -4.474076  0.0 -1.0
 2013-01-06       NaN       NaN       NaN  NaN  NaN)
```

```
df.agg(lambda x: np.mean(x) * 5.6)
```

```
A     2.672962
B    -0.243762
C     0.530100
D    28.000000
F    16.800000
dtype: float64
```

```
df.transform(lambda x: x * 101.2)
```


{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 59.00689656601559,\n
\"min\": -16.815755437464244,\n        \"max\": 146.95585555990775,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          0.0,\n
28.758757353025363,\n          73.59109522727789\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 140.99897523888953,\n
\"min\": -130.0391282729054,\n        \"max\": 264.95191146313607,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          0.0,\n
-48.81217696094983,\n          -130.0391282729054\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 84.45584846173672,\n
\"min\": -136.9513872173785,\n        \"max\": 94.07916264133691,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
71.68112069826194,\n          94.07916264133691,\n          -
25.231222919975348\n        ],\n        \"semantic_type\": \"\",\n

\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"D\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0,\n        \"min\": 506.0,\n        \"max\": 506.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          506.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"F\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 160.01124960452,\n
\"min\": 101.2,\n        \"max\": 506.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          202.4\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}

```
s = pd.Series(np.random.randint(0, 7, size=10))
s
```

```
0    6
1    0
2    1
3    1
4    3
5    3
6    2
7    2
8    3
9    3
dtype: int64
```

```
s.value_counts()
```

```
3    4
1    2
2    2
6    1
0    1
Name: count, dtype: int64
```

```
s = pd.Series(["A", "B", "C", "Aaba", "Baca", np.nan, "CABA", "dog",
"cat"])
```

```
s.str.lower()
```

```
0       a
1       b
2       c
3    aaba
4    baca
5     NaN
6    caba
7     dog
8     cat
dtype: object
```

```
df = pd.DataFrame(np.random.randn(10, 4))
df
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 10,\n  \"fields\": [\n    {\n      \"column\": 0,\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.5822256925034867,\n        \"min\": -0.5907961327849429,\n        \"max\": 1.0379427563902164,\n        \"num_unique_values\": 10,\n        \"samples\": [\n          0.41179164009168495,\n          -0.4520799984521189,\n          0.30673927299704595\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": 1,\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.1671084339718834,\n        \"min\": -2.224535512022536,\n        \"max\": 1.0881954627028063,\n        \"num_unique_values\": 10,\n        \"samples\": [\n          0.6785636122288032,\n          -0.06314545439653191,\n          -1.9284264211734132\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": 2,\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.9732183777459199,\n        \"min\": -2.4346282566538817,\n        \"max\": 0.5775656122866767,\n        \"num_unique_values\": 10,\n        \"samples\": [\n          0.19156683631159743,\n          -0.19526114303350073,\n          -0.9030024194491025\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": 3,\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.2979584394674801,\n        \"min\": -1.4961741605301446,\n        \"max\": 2.976484999543247,\n        \"num_unique_values\": 10,\n        \"samples\": [\n          0.5600892937048139,\n          -0.9207961199293849,\n          -0.39760986122096925\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}

```
pieces = [df[:3], df[3:7], df[7:]]
```

```
pd.concat(pieces)
```

{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 10,\n  \"fields\": [\n    {\n      \"column\": 0,\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.5822256925034867,\n        \"min\": -0.5907961327849429,\n        \"max\": 1.0379427563902164,\n        \"num_unique_values\": 10,\n        \"samples\": [\n          0.41179164009168495,\n          -0.4520799984521189,\n          0.30673927299704595\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": 1,\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.1671084339718834,\n        \"min\": -2.224535512022536,\n        \"max\": 1.0881954627028063,\n        \"num_unique_values\": 10,\n        \"samples\": [\n          0.6785636122288032,\n          -0.06314545439653191,\n          -1.9284264211734132\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\

n    },\n    {\n      \"column\": 2,\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.9732183777459199,\n        \"min\": -2.4346282566538817,\n        \"max\": 0.5775656122866767,\n        \"num_unique_values\": 10,\n        \"samples\": [\n          0.19156683631159743,\n          -0.19526114303350073,\n          -0.9030024194491025\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": 3,\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1.2979584394674801,\n        \"min\": -1.4961741605301446,\n        \"max\": 2.976484999543247,\n        \"num_unique_values\": 10,\n        \"samples\": [\n          0.5600892937048139,\n          -0.9207961199293849,\n          -0.39760986122096925\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

```
left = pd.DataFrame({"key": ["foo", "foo"], "lval": [1, 2]})

right = pd.DataFrame({"key": ["foo", "foo"], "rval": [4, 5]})

left, right
```

```
(   key  lval
 0  foo     1
 1  foo     2,
    key  rval
 0  foo     4
 1  foo     5)
```

```
pd.merge(left, right, on="key")
```

{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 4,\n  \"fields\": [\n    {\n      \"column\": \"key\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          \"foo\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"lval\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 1,\n        \"max\": 2,\n        \"num_unique_values\": 2,\n        \"samples\": [\n          2\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"rval\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 4,\n        \"max\": 5,\n        \"num_unique_values\": 2,\n        \"samples\": [\n          5\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

```
left = pd.DataFrame({"key": ["foo", "bar"], "lval": [1, 2]})

right = pd.DataFrame({"key": ["foo", "bar"], "rval": [4, 5]})

left, right
```

```
(   key  lval
 0  foo     1
 1  bar     2,
    key  rval
 0  foo     4
 1  bar     5)
```

```python
pd.merge(left, right, on="key")
```

{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 2,\n  \"fields\": [\n
{\n      \"column\": \"key\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"bar\",\n          \"foo\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lval\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 2,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          2,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"rval\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 4,\n
\"max\": 5,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          5,\n          4\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}

```python
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one",
"three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)


df
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"bar\",\n          \"foo\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"B\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"one\",\n          \"two\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0160526142342754,\n

```
\"min\": -0.5980665113447898,\n        \"max\": 2.4729600637802305,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
0.9730363943489663,\n          0.24836380466173108\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.8016659669885627,\n
\"min\": -0.24832676903223064,\n        \"max\": 2.2306401812413417,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          -
0.16555631057446324,\n          1.104006609116566\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
df.groupby("A")[["C", "D"]].sum()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 2,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"foo\",\n          \"bar\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"C\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.43272196599579327,\n
\"min\": 0.8534618330956943,\n        \"max\": 1.4654231061436944,\n
\"num_unique_values\": 2,\n        \"samples\": [\n
1.4654231061436944,\n          0.8534618330956943\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3.0159732201131417,\n
\"min\": 0.690123529509872,\n        \"max\": 4.955353761147933,\n
\"num_unique_values\": 2,\n        \"samples\": [\n
4.955353761147933,\n          0.690123529509872\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
arrays = [
   ["bar", "bar", "baz", "baz", "foo", "foo", "qux", "qux"],
   ["one", "two", "one", "two", "one", "two", "one", "two"],
]

index = pd.MultiIndex.from_arrays(arrays, names=["first", "second"])

df = pd.DataFrame(np.random.randn(8, 2), index=index, columns=["A",
"B"])

df2 = df[:4]
df2
```


```
{"summary":"{\n  \"name\": \"df2\",\n  \"rows\": 4,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.9973077940874391,\n
\"min\": -1.341493411102551,\n        \"max\": 0.9072995955710308,\n
```

\"num_unique_values\": 4,\n        \"samples\": [\n
0.13051654325424566,\n          -1.341493411102551,\n
0.9072995955710308\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.6515776739828759,\n        \"min\": -0.3113984441245247,\n
\"max\": 1.1024277048311908,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          -0.3113984441245247,\n
0.4313709156112516,\n          1.1024277048311908\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df2"}

```
stacked = df2.stack()
stacked
```

```
first  second
bar    one     A    0.907300
               B    1.102428
       two     A    0.130517
               B   -0.311398
baz    one     A   -0.808122
               B   -0.196245
       two     A   -1.341493
               B    0.431371
dtype: float64
```

```
stacked.unstack()
```

{"summary":"{\n  \"name\": \"stacked\",\n  \"rows\": 4,\n  \"fields\":
[\n    {\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.9973077940874391,\n
\"min\": -1.341493411102551,\n        \"max\": 0.9072995955710308,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.13051654325424566,\n          -1.341493411102551,\n
0.9072995955710308\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.6515776739828759,\n        \"min\": -0.3113984441245247,\n
\"max\": 1.1024277048311908,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          -0.3113984441245247,\n
0.4313709156112516,\n          1.1024277048311908\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

```
df = pd.DataFrame(
    {
        "A": ["one", "one", "two", "three"] * 3,
        "B": ["A", "B", "C"] * 4,
        "C": ["foo", "foo", "foo", "bar", "bar", "bar"] * 2,
        "D": np.random.randn(12),
```

```
        "E": np.random.randn(12),
    }
)

df
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 12,\n  \"fields\": [\n
{\n      \"column\": \"A\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"one\",\n          \"two\",\n
\"three\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"B\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"A\",\n
\"B\",\n          \"C\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"C\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"bar\",\n          \"foo\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"D\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0117156986595175,\n
\"min\": -1.2408488082277833,\n        \"max\": 1.8121795298877301,\n
\"num_unique_values\": 12,\n        \"samples\": [\n
1.8121795298877301,\n          -0.4814139181407084\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"E\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0277434337737457,\n
\"min\": -1.2112809329208898,\n        \"max\": 2.0799323280622923,\n
\"num_unique_values\": 12,\n        \"samples\": [\n
2.0032969791468966,\n          2.0799323280622923\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
pd.pivot_table(df, values="D", index=["A", "B"], columns=["C"])
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 9,\n  \"fields\": [\n
{\n      \"column\": \"bar\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0854216125595606,\n
\"min\": -1.2098104596133443,\n        \"max\": 1.8121795298877301,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          -
0.4814139181407084,\n          -0.7569381228387065,\n
1.8121795298877301\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"foo\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.0077806315024422,\n        \"min\": -1.2408488082277833,\n
\"max\": 1.2995812435532499,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1.2995812435532499,\n
1.104813841016105,\n          -0.6604775084798318\n        ],\n
```

\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

```
rng = pd.date_range("1/1/2012", periods=100, freq="s")

ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)
ts
```

```
2012-01-01 00:00:00    366
2012-01-01 00:00:01    309
2012-01-01 00:00:02     33
2012-01-01 00:00:03    333
2012-01-01 00:00:04    247
                      ...
2012-01-01 00:01:35    156
2012-01-01 00:01:36    227
2012-01-01 00:01:37    190
2012-01-01 00:01:38     28
2012-01-01 00:01:39    439
Freq: S, Length: 100, dtype: int64
```

```
ts.resample("5Min").sum()
```

```
2012-01-01    21714
Freq: 5T, dtype: int64
```

```
rng = pd.date_range("3/6/2012 00:00", periods=5, freq="D")

ts = pd.Series(np.random.randn(len(rng)), rng)

ts
```

```
2012-03-06    0.575040
2012-03-07    0.038225
2012-03-08   -0.068503
2012-03-09   -0.230276
2012-03-10    0.499263
Freq: D, dtype: float64
```

```
ts_utc = ts.tz_localize("UTC")

ts_utc
```

```
2012-03-06 00:00:00+00:00    0.575040
2012-03-07 00:00:00+00:00    0.038225
2012-03-08 00:00:00+00:00   -0.068503
2012-03-09 00:00:00+00:00   -0.230276
2012-03-10 00:00:00+00:00    0.499263
Freq: D, dtype: float64
```

```
ts_utc.tz_convert("US/Eastern")
```

```
2012-03-05 19:00:00-05:00    0.575040
2012-03-06 19:00:00-05:00    0.038225
2012-03-07 19:00:00-05:00   -0.068503
2012-03-08 19:00:00-05:00   -0.230276
2012-03-09 19:00:00-05:00    0.499263
Freq: D, dtype: float64
```

```
rng
```

```
DatetimeIndex(['2012-03-06', '2012-03-07', '2012-03-08', '2012-03-09',
               '2012-03-10'],
              dtype='datetime64[ns]', freq='D')
```

```
rng + pd.offsets.BusinessDay(5)
```

```
DatetimeIndex(['2012-03-13', '2012-03-14', '2012-03-15', '2012-03-16',
               '2012-03-16'],
              dtype='datetime64[ns]', freq=None)
```

```
df = pd.DataFrame(
    {"id": [1, 2, 3, 4, 5, 6], "raw_grade": ["a", "b", "b", "a", "a",
"e"]}
)
df["grade"] = df["raw_grade"].astype("category")
df["grade"]
```

```
0    a
1    b
2    b
3    a
4    a
5    e
Name: grade, dtype: category
Categories (3, object): ['a', 'b', 'e']
```

```
new_categories = ["very good", "good", "very bad"]
```

```
df["grade"] = df["grade"].cat.rename_categories(new_categories)
```

```
df["grade"]
```

```
0    very good
1         good
2         good
3    very good
4    very good
5     very bad
Name: grade, dtype: category
Categories (3, object): ['very good', 'good', 'very bad']
```

```
df["grade"] = df["grade"].cat.set_categories(
    ["very bad", "bad", "medium", "good", "very good"]
)


df["grade"]
```

```
0    very good
1         good
2         good
3    very good
4    very good
5     very bad
Name: grade, dtype: category
Categories (5, object): ['very bad', 'bad', 'medium', 'good', 'very
good']
```

```
df.sort_values(by="grade")
```

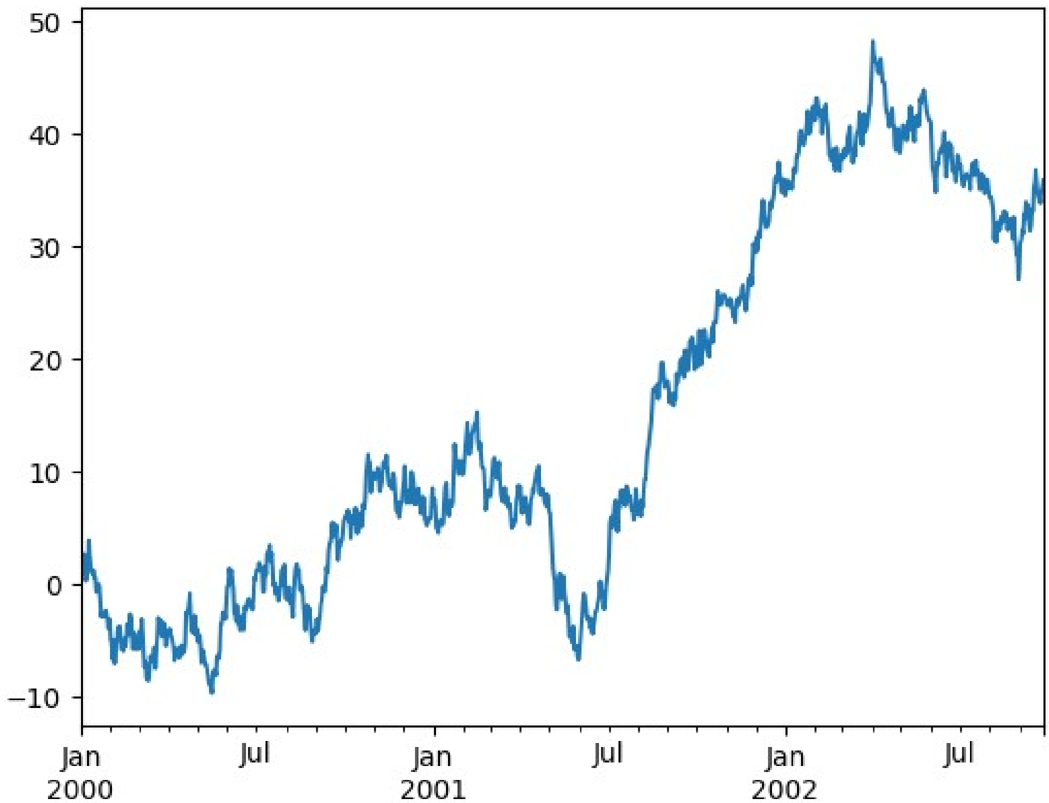
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6,\n  \"fields\": [\n    {\n      \"column\": \"id\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 1,\n        \"max\": 6,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          6,\n          2,\n          5\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"raw_grade\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 3,\n        \"samples\": [\n          \"e\",\n          \"b\",\n          \"a\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"grade\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 3,\n        \"samples\": [\n          \"very bad\",\n          \"good\",\n          \"very good\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

```
df.groupby("grade", observed=False).size()
```

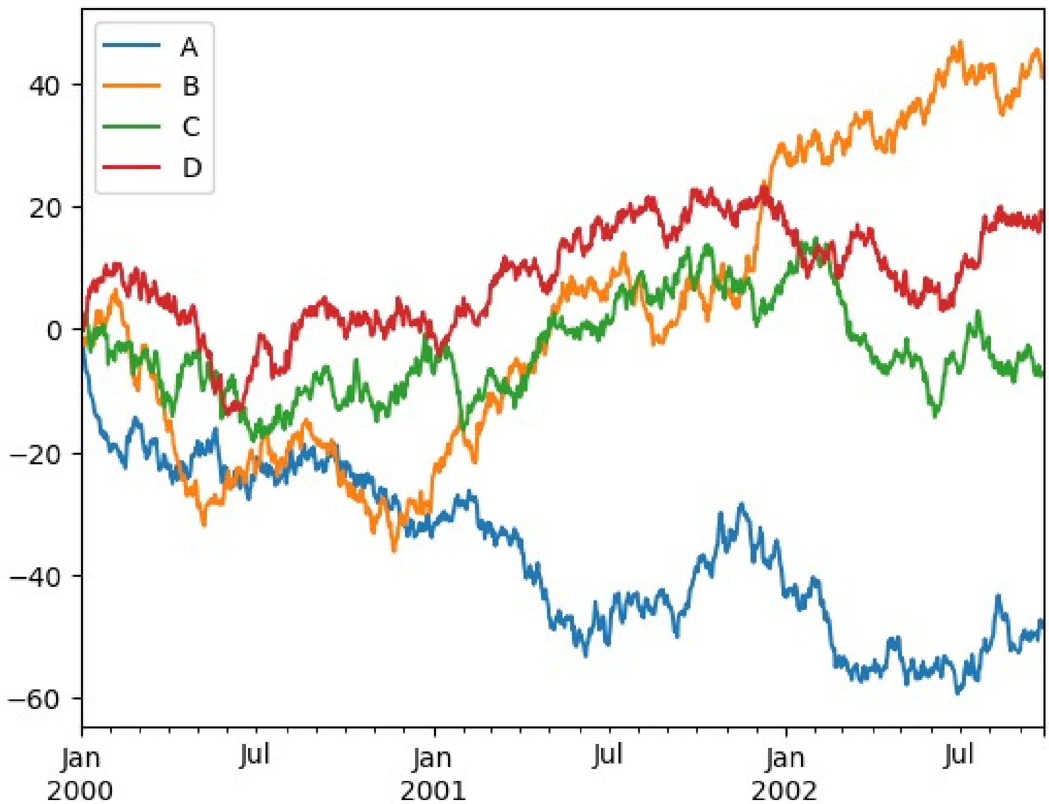
```
grade
very bad     1
bad          0
medium       0
good         2
very good    3
dtype: int64
```

```
import matplotlib.pyplot as plt

plt.close("all")
```

```
ts = pd.Series(np.random.randn(1000), index=pd.date_range("1/1/2000",
periods=1000))

ts = ts.cumsum()

ts.plot();
```




```
df = pd.DataFrame(
    np.random.randn(1000, 4), index=ts.index, columns=["A", "B", "C",
"D"]
)


df = df.cumsum()

plt.figure();

df.plot();

plt.legend(loc='best');

<Figure size 640x480 with 0 Axes>
```




```
df = pd.DataFrame(np.random.randint(0, 5, (10, 5)))

df.to_csv("foo.csv")

pd.read_csv("foo.csv")
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"Unnamed: 0\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3,\n        \"min\": 0,\n
\"max\": 9,\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          8,\n          1,\n          5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"0\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 4,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          0,\n          3,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"1\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 4,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          3,\n          2,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"2\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 3,\n        \"num_unique_values\": 4,\n        \"samples\":
```

```
[\n          3,\n          0,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"3\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 4,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          2,\n          3,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"4\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 4,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          4,\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
df.to_parquet("foo.parquet")
```

```
pd.read_parquet("foo.parquet")
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": 0,\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 4,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          0,\n          3,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 1,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 4,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          3,\n          2,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 2,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 3,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          3,\n          0,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 3,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 4,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          2,\n          3,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 4,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 4,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          4,\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
df.to_excel("foo.xlsx", sheet_name="Sheet1")
```

```
pd.read_excel("foo.xlsx", "Sheet1", index_col=None, na_values=["NA"])
```

```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"Unnamed: 0\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3,\n        \"min\": 0,\n
\"max\": 9,\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          8,\n          1,\n          5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 0,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 4,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          0,\n          3,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 1,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 4,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          3,\n          2,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 2,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 3,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          3,\n          0,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 3,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 4,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          2,\n          3,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 4,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 4,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          4,\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Discussion and Exercise 2.2.5

```
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/
abalone.data"
names = ['Sex', 'Length', 'Diameter', 'Height', 'Whole weight',
'Shucked weight', 'Viscera weight', 'Shell weight', 'Rings']
data = pd.read_csv(url, names=names)

print(data.head())
print(data.info())
#print(data.describe())
```

```
  Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera
weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245
```

```
0.1010
1   M   0.350     0.265   0.090         0.2255           0.0995
0.0485
2   F   0.530     0.420   0.135         0.6770           0.2565
0.1415
3   M   0.440     0.365   0.125         0.5160           0.2155
0.1140
4   I   0.330     0.255   0.080         0.2050           0.0895
0.0395

   Shell weight  Rings
0         0.150     15
1         0.070      7
2         0.210      9
3         0.155     10
4         0.055      7
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
None
```

```python
missing_values = data.isnull().sum().sum()
/(data.shape[0]*data.shape[1])
print(missing_values)
```

```
0.0
```

```python
num_vars = data.select_dtypes(include='number').shape[1]
cat_vars = data.select_dtypes(include='object').shape[1]
text_vars = data.select_dtypes(include='string').shape[1]
print("Numerical variables:", num_vars)
print("Categorical variables:", cat_vars)
print("Text variables:", text_vars)
```

```
Numerical variables: 8
Categorical variables: 1
Text variables: 0
```

```python
selected_data = data[['Length', 'Height', 'Rings']]
selected_data
```

{"summary":"{\n  \"name\": \"selected_data\",\n  \"rows\": 4177,\n  \"fields\": [\n    {\n      \"column\": \"Length\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.12009291256479956,\n        \"min\": 0.075,\n        \"max\": 0.815,\n        \"num_unique_values\": 134,\n        \"samples\": [\n          0.815,\n          0.65,\n          0.29\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Height\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.041827056607257274,\n        \"min\": 0.0,\n        \"max\": 1.13,\n        \"num_unique_values\": 51,\n        \"samples\": [\n          0.235,\n          0.035,\n          0.015\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Rings\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 3,\n        \"min\": 1,\n        \"max\": 29,\n        \"num_unique_values\": 28,\n        \"samples\": [\n          11,\n          27,\n          14\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"selected_data"}

```python
data.memory_usage()
```

```
Index               128
Sex               33416
Length            33416
Diameter          33416
Height            33416
Whole weight      33416
Shucked weight    33416
Viscera weight    33416
Shell weight      33416
Rings             33416
dtype: int64
```

```python
data.memory_usage(deep=True).sum()
```

```
509722
```

Pandas can struggle with datasets that exceed available system memory. This is due to its in-memory analytics and the creation of intermediate copies during operations suach as filtering, sorting, and modifying data. The problem can be solved by loading only the necessary columns or a subset of the data, to reduce the memory footprint.

When dealing with data that has a very large number of categories, if categories are similar or have low frequency, we can combine them into a single category. One-hot encoding and label encoding are also other common approaches. One-hot encoding can create a high-dimensional sparse matrix, especially with many categories. If teh data has many categories with low frequency specialized libraries like SciPy's sparse matrices can help to efficiently store and manipulate it. By label encoding we assign a unique integer to each category. This is more memory-efficient but might not be suitable for all machine learning algorithms.

**2.3 Linear Algebra**

```
x = torch.tensor(3.0)
y = torch.tensor(2.0)

x + y, x * y, x / y, x**y
```

```
(tensor(5.), tensor(6.), tensor(1.5000), tensor(9.))
```

```
x = torch.arange(3)
x
```

```
tensor([0, 1, 2])
```

```
x[2]
```

```
tensor(2)
```

```
len(x)
```

```
3
```

```
x.shape
```

```
torch.Size([3])
```

```
A = torch.arange(6).reshape(3, 2)
A
```

```
tensor([[0, 1],
        [2, 3],
        [4, 5]])
```

```
A.T
```

```
tensor([[0, 2, 4],
        [1, 3, 5]])
```

```
A = torch.tensor([[1, 2, 3], [2, 0, 4], [3, 4, 5]])
A == A.T
```

```
tensor([[True, True, True],
        [True, True, True],
        [True, True, True]])
```

```
torch.arange(24).reshape(2, 3, 4)
```

```
tensor([[[ 0,  1,  2,  3],
         [ 4,  5,  6,  7],
         [ 8,  9, 10, 11]],
```

```
        [[12, 13, 14, 15],
         [16, 17, 18, 19],
         [20, 21, 22, 23]]]])
```

```
A = torch.arange(6, dtype=torch.float32).reshape(2, 3)
B = A.clone()
A, A + B
```

```
(tensor([[0., 1., 2.],
         [3., 4., 5.]]),
 tensor([[ 0.,  2.,  4.],
         [ 6.,  8., 10.]]))
```

```
A * B
```

```
tensor([[ 0.,  1.,  4.],
        [ 9., 16., 25.]])
```

```
a = 2
X = torch.arange(24).reshape(2, 3, 4)
a + X, (a * X).shape
```

```
(tensor([[[ 2,  3,  4,  5],
          [ 6,  7,  8,  9],
          [10, 11, 12, 13]],

         [[14, 15, 16, 17],
          [18, 19, 20, 21],
          [22, 23, 24, 25]]]),
 torch.Size([2, 3, 4]))
```

```
x = torch.arange(3, dtype=torch.float32)
x, x.sum()
```

```
(tensor([0., 1., 2.]), tensor(3.))
```

```
A.shape, A.sum()
```

```
(torch.Size([2, 3]), tensor(15.))
```

```
A.shape, A.sum(axis=0).shape
```

```
(torch.Size([2, 3]), torch.Size([3]))
```

```
A.shape, A.sum(axis=1).shape
```

```
(torch.Size([2, 3]), torch.Size([2]))
```

```
A.sum(axis=[0, 1]) == A.sum()
```

```
tensor(True)
```

```
A.mean(), A.sum() / A.numel()
```

```
(tensor(2.5000), tensor(2.5000))
```

```
A.mean(axis=0), A.sum(axis=0) / A.shape[0]
```

```
(tensor([1.5000, 2.5000, 3.5000]), tensor([1.5000, 2.5000, 3.5000]))
```

```
sum_A = A.sum(axis=1, keepdims=True)
sum_A, sum_A.shape
```

```
(tensor([[ 3.],
         [12.]]),
 torch.Size([2, 1]))
```

```
A / sum_A
```

```
tensor([[0.0000, 0.3333, 0.6667],
        [0.2500, 0.3333, 0.4167]])
```

```
A.cumsum(axis=0)
```

```
tensor([[0., 1., 2.],
        [3., 5., 7.]])
```

```
y = torch.ones(3, dtype = torch.float32)
x, y, torch.dot(x, y)
```

```
(tensor([0., 1., 2.]), tensor([1., 1., 1.]), tensor(3.))
```

```
torch.sum(x * y)
```

```
tensor(3.)
```

```
A.shape, x.shape, torch.mv(A, x), A@x
```

```
(torch.Size([2, 3]), torch.Size([3]), tensor([ 5., 14.]), tensor([ 5.,
14.]))
```

```
B = torch.ones(3, 4)
torch.mm(A, B), A@B
```

```
(tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]),
 tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]))
```

```
u = torch.tensor([3.0, -4.0])
torch.norm(u)
```

```
tensor(5.)
```

```
torch.abs(u).sum()
```

```
tensor(7.)
```

```
torch.norm(torch.ones((4, 9)))
```

```
tensor(6.)
```

### 2.5 Automatic Differentiation

```
x = torch.arange(4.0, requires_grad=True)
x.grad
```

```
y = 2 * torch.dot(x, x)
y
```

```
tensor(28., grad_fn=<MulBackward0>)
```

```
y.backward()
x.grad
```

```
tensor([ 0.,  4.,  8., 12.])
```

```
x.grad == 4 * x
```

```
tensor([True, True, True, True])
```

```
x.grad.zero_()  # Reset the gradient
y = x.sum()
y.backward()
x.grad
```

```
tensor([1., 1., 1., 1.])
```

```
x.grad.zero_()
y = x * x
y.backward(gradient=torch.ones(len(y)))
x.grad
```

```
tensor([0., 2., 4., 6.])
```

```
x.grad.zero_()
y = x * x
u = y.detach()
z = u * x

z.sum().backward()
x.grad == u
```

```
tensor([True, True, True, True])
```

```
x.grad.zero_()
y.sum().backward()
x.grad == 2 * x
```

```
tensor([True, True, True, True])
```

```
def f(a):
    b = a * 2
    while b.norm() < 1000:
        b = b * 2
    if b.sum() > 0:
        c = b
    else:
        c = 100 * b
    return c

a = torch.randn(size=(), requires_grad=True)
d = f(a)
d.backward()

a.grad == d / a

tensor(True)
```

# Linear Neural Networks for Regression

**3.1 Linear Regression**

```
import math
import time
import numpy as np
import torch
from torch import nn
from d2l import torch as d2l

n = 10000
a = torch.ones(n)
b = torch.ones(n)

c = torch.zeros(n)
t = time.time()
for i in range(n):
    c[i] = a[i] + b[i]
f'{time.time() - t:.5f} sec'

{"type":"string"}

t = time.time()
d = a + b
f'{time.time() - t:.5f} sec'

{"type":"string"}
```

```
def normal(x, mu, sigma):
    p = 1 / math.sqrt(2 * math.pi * sigma**2)
    return p * np.exp(-0.5 * (x - mu)**2 / sigma**2)

x = np.arange(-7, 7, 0.01)

params = [(0, 1), (0, 2), (3, 1)]
d2l.plot(x, [normal(x, mu, sigma) for mu, sigma in params],
xlabel='x',
         ylabel='p(x)',
         figsize=(4.5, 2.5),
         legend=[f'mean {mu}, std {sigma}' for mu, sigma in params])
```

**3.2. Object-Oriented Design for Implementation**

```
def add_to_class(Class):
    #Register functions as methods in created class
    def wrapper(obj):
        setattr(Class, obj.__name__, obj)
    return wrapper

class A:
    def __init__(self):
        self.b = 1

a = A()

@add_to_class(A)
def do(self):
    print('Class attribute "b" is', self.b)

a.do()
```

```
Class attribute "b" is 1
```

```python
class HyperParameters:
    #The base class of hyperparameters
    def save_hyperparameters(self, ignore=[]):
        raise NotImplemented

# Call the fully implemented HyperParameters class saved in d2l
class B(d2l.HyperParameters):
    def __init__(self, a, b, c):
        self.save_hyperparameters(ignore=['c'])
        print('self.a =', self.a, 'self.b =', self.b)
        print('There is no self.c =', not hasattr(self, 'c'))

b = B(a=1, b=2, c=3)
```

```
self.a = 1 self.b = 2
There is no self.c = True
```

```python
class ProgressBoard(d2l.HyperParameters):

    #The board that plots data points in animation
    def __init__(self, xlabel=None, ylabel=None, xlim=None,
                 ylim=None, xscale='linear', yscale='linear',
                 ls=['-', '--', '-.', ':'], colors=['C0', 'C1', 'C2',
'C3'],
                 fig=None, axes=None, figsize=(3.5, 2.5),
display=True):
        self.save_hyperparameters()

    def draw(self, x, y, label, every_n=1):
        raise NotImplemented

board = d2l.ProgressBoard('x')
for x in np.arange(0, 10, 0.1):
    board.draw(x, np.sin(x), 'sin', every_n=2)
    board.draw(x, np.cos(x), 'cos', every_n=10)
```

```
class Module(nn.Module, d2l.HyperParameters):
    #The base class of models
    def __init__(self, plot_train_per_epoch=2,
plot_valid_per_epoch=1):
        super().__init__()
        self.save_hyperparameters()
        self.board = ProgressBoard()

    def loss(self, y_hat, y):
        raise NotImplementedError

    def forward(self, X):
        assert hasattr(self, 'net'), 'Neural network is defined'
        return self.net(X)

    def plot(self, key, value, train):
        #Plot a point in animation
        assert hasattr(self, 'trainer'), 'Trainer is not inited'
        self.board.xlabel = 'epoch'
        if train:
            x = self.trainer.train_batch_idx / \
                self.trainer.num_train_batches
            n = self.trainer.num_train_batches / \
                self.plot_train_per_epoch
        else:
            x = self.trainer.epoch + 1
            n = self.trainer.num_val_batches / \
                self.plot_valid_per_epoch
        self.board.draw(x, value.to(d2l.cpu()).detach().numpy(),
('train_' if train else 'val_') + key, every_n=int(n))

    def training_step(self, batch):
        l = self.loss(self(*batch[:-1]), batch[-1])
        self.plot('loss', l, train=True)
        return l

    def validation_step(self, batch):
        l = self.loss(self(*batch[:-1]), batch[-1])
        self.plot('loss', l, train=False)

    def configure_optimizers(self):
        raise NotImplementedError

class DataModule(d2l.HyperParameters):
    def __init__(self, root='../data', num_workers=4):
        self.save_hyperparameters()

    def get_dataloader(self, train):
        raise NotImplementedError
```

```
    def train_dataloader(self):
        return self.get_dataloader(train=True)

    def val_dataloader(self):
        return self.get_dataloader(train=False)

class Trainer(d2l.HyperParameters):
    #The base class for training models with data
    def __init__(self, max_epochs, num_gpus=0, gradient_clip_val=0):
        self.save_hyperparameters()
        assert num_gpus == 0, 'No GPU support yet'

    def prepare_data(self, data):
        self.train_dataloader = data.train_dataloader()
        self.val_dataloader = data.val_dataloader()
        self.num_train_batches = len(self.train_dataloader)
        self.num_val_batches = (len(self.val_dataloader)
                                if self.val_dataloader is not None
else 0)

    def prepare_model(self, model):
        model.trainer = self
        model.board.xlim = [0, self.max_epochs]
        self.model = model

    def fit(self, model, data):
        self.prepare_data(data)
        self.prepare_model(model)
        self.optim = model.configure_optimizers()
        self.epoch = 0
        self.train_batch_idx = 0
        self.val_batch_idx = 0
        for self.epoch in range(self.max_epochs):
            self.fit_epoch()

    def fit_epoch(self):
        raise NotImplementedError
```

**3.4. Linear Regression Implementation from Scratch**

```
class LinearRegressionScratch(d2l.Module):
    #The linear regression model implemented from scratch
    def __init__(self, num_inputs, lr, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.w = torch.normal(0, sigma, (num_inputs, 1),
requires_grad=True)
        self.b = torch.zeros(1, requires_grad=True)
```

```python
@d2l.add_to_class(LinearRegressionScratch)
def forward(self, X):
    return torch.matmul(X, self.w) + self.b

@d2l.add_to_class(LinearRegressionScratch)
def loss(self, y_hat, y):
    l = (y_hat - y) ** 2 / 2
    return l.mean()

class SGD(d2l.HyperParameters):
    # Minibatch stochastic gradient descent
    def __init__(self, params, lr):
        self.save_hyperparameters()

    def step(self):
        for param in self.params:
            param -= self.lr * param.grad

    def zero_grad(self):
        for param in self.params:
            if param.grad is not None:
                param.grad.zero_()

@d2l.add_to_class(LinearRegressionScratch)
def configure_optimizers(self):
    return SGD([self.w, self.b], self.lr)

@d2l.add_to_class(d2l.Trainer)
def prepare_batch(self, batch):
    return batch

@d2l.add_to_class(d2l.Trainer)
def fit_epoch(self):
    self.model.train()
    for batch in self.train_dataloader:
        loss = self.model.training_step(self.prepare_batch(batch))
        self.optim.zero_grad()
        with torch.no_grad():
            loss.backward()
            if self.gradient_clip_val > 0:  # To be discussed later
                self.clip_gradients(self.gradient_clip_val,
self.model)
            self.optim.step()
        self.train_batch_idx += 1
    if self.val_dataloader is None:
        return
    self.model.eval()
    for batch in self.val_dataloader:
        with torch.no_grad():
```

```
            self.model.validation_step(self.prepare_batch(batch))
        self.val_batch_idx += 1

model = LinearRegressionScratch(2, lr=0.03)
data = d2l.SyntheticRegressionData(w=torch.tensor([2, -3.4]), b=4.2)
trainer = d2l.Trainer(max_epochs=3)
trainer.fit(model, data)
```

```
with torch.no_grad():
    print(f'error in estimating w: {data.w -
model.w.reshape(data.w.shape)}')
    print(f'error in estimating b: {data.b - model.b}')

error in estimating w: tensor([ 0.0981, -0.1842])
error in estimating b: tensor([0.2252])
```

**3. Key takeaways**

To summerize this section, Linear Regression is used to predict numerical values. It assumes a linear relationship between features and the target variable, and aims to find the optimal weights and bias that minimize the squared error loss.

Directly composing two linear layers is equivalent to a single linear layer.To create a more complex model, you need to introduce non-linear activations between the layers.

# Linear Neural Networks for Classification

**4.1 Softmax Regression**

Softmax regression is used in classification problems, where the goal is to predict which category a new data point belongs to. The softmax function maps a vector of real numbers to a probability distribution over the same set of real numbers. It's used to normalize the output of neural network so that it can be interpreted as a probability distribution. In more detail, the softmax function exponentiates each output and then divides it by the sum of all exponentiated outputs. This ensures that the resulting values are non-negative and sum to 1.

**4.2 Image classification Dataset**

```
import time
import torch
import torchvision
from torchvision import transforms
from d2l import torch as d2l

d2l.use_svg_display()

class FashionMNIST(d2l.DataModule):
    # The Fashion-MNIST dataset
    def __init__(self, batch_size=64, resize=(28, 28)):
        super().__init__()
        self.save_hyperparameters()
        trans =
transforms.Compose([transforms.Resize(resize),transforms.ToTensor()])

        self.train = torchvision.datasets.FashionMNIST(root=self.root,
train=True, transform=trans, download=True)
        self.val = torchvision.datasets.FashionMNIST(root=self.root,
train=False, transform=trans, download=True)

data = FashionMNIST(resize=(32, 32))
len(data.train), len(data.val)
```

```
(60000, 10000)
```

```
data.train[0][0].shape
```

```
torch.Size([1, 32, 32])
```

```
@d2l.add_to_class(FashionMNIST)
def text_labels(self, indices):
    #Return text labels
    labels = ['t-shirt', 'trouser', 'pullover', 'dress',
'coat','sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [labels[int(i)] for i in indices]

@d2l.add_to_class(FashionMNIST)
def get_dataloader(self, train):
    data = self.train if train else self.val
    return torch.utils.data.DataLoader(data, self.batch_size,
shuffle=train, num_workers=self.num_workers)

X, y = next(iter(data.train_dataloader()))
print(X.shape, X.dtype, y.shape, y.dtype)
```

```
torch.Size([64, 1, 32, 32]) torch.float32 torch.Size([64]) torch.int64
```

```python
tic = time.time()
for X, y in data.train_dataloader():
    continue
f'{time.time() - tic:.2f} sec'
```

{"type":"string"}

```python
def show_images(imgs, num_rows, num_cols, titles=None, scale=1.5):
    # Plot a list of images
    raise NotImplementedError

@d2l.add_to_class(FashionMNIST)
def visualize(self, batch, nrows=1, ncols=8, labels=[]):
    X, y = batch
    if not labels:
        labels = self.text_labels(y)
    d2l.show_images(X.squeeze(1), nrows, ncols, titles=labels)
batch = next(iter(data.val_dataloader()))
data.visualize(batch)
```

**4.3 The Base classification Model**

```python
class Classifier(d2l.Module):
    #The base class of classification models
    def validation_step(self, batch):
        Y_hat = self(*batch[:-1])
        self.plot('loss', self.loss(Y_hat, batch[-1]), train=False)
        self.plot('acc', self.accuracy(Y_hat, batch[-1]), train=False)

@d2l.add_to_class(d2l.Module)
def configure_optimizers(self):
    return torch.optim.SGD(self.parameters(), lr=self.lr)

@d2l.add_to_class(Classifier)
def accuracy(self, Y_hat, Y, averaged=True):
    # Compute the number of correct predictions
    Y_hat = Y_hat.reshape((-1, Y_hat.shape[-1]))
    preds = Y_hat.argmax(axis=1).type(Y.dtype)
    compare = (preds == Y.reshape(-1)).type(torch.float32)
    return compare.mean() if averaged else compare
```

**4.4. Softmax Regression Implementation from Scratch**

```python
X = torch.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
X.sum(0, keepdims=True), X.sum(1, keepdims=True)
```

```
(tensor([[5., 7., 9.]]),
 tensor([[ 6.],
         [15.]]))
```

```python
def softmax(X):
    X_exp = torch.exp(X)
    partition = X_exp.sum(1, keepdims=True)
    return X_exp / partition  # The broadcasting mechanism is applied
here
```

```python
X = torch.rand((2, 5))
X_prob = softmax(X)
X_prob, X_prob.sum(1)
```

```
(tensor([[0.2124, 0.2463, 0.2284, 0.2156, 0.0973],
         [0.1182, 0.2165, 0.1333, 0.2957, 0.2363]]),
 tensor([1., 1.]))
```

```python
class SoftmaxRegressionScratch(d2l.Classifier):
    def __init__(self, num_inputs, num_outputs, lr, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.W = torch.normal(0, sigma, size=(num_inputs,
num_outputs), requires_grad=True)
        self.b = torch.zeros(num_outputs, requires_grad=True)

    def parameters(self):
        return [self.W, self.b]
```

```python
@d2l.add_to_class(SoftmaxRegressionScratch)
def forward(self, X):
    X = X.reshape((-1, self.W.shape[0]))
    return softmax(torch.matmul(X, self.W) + self.b)
```

```python
y = torch.tensor([0, 2])
y_hat = torch.tensor([[0.1, 0.3, 0.6], [0.3, 0.2, 0.5]])
y_hat[[0, 1], y]
```

```
tensor([0.1000, 0.5000])
```

```python
def cross_entropy(y_hat, y):
    return -torch.log(y_hat[list(range(len(y_hat))), y]).mean()
```

```python
cross_entropy(y_hat, y)
```

```
tensor(1.4979)
```

```python
@d2l.add_to_class(SoftmaxRegressionScratch)
def loss(self, y_hat, y):
    return cross_entropy(y_hat, y)

data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10,
lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```

```python
X, y = next(iter(data.val_dataloader()))
preds = model(X).argmax(axis=1)
preds.shape
```

```
torch.Size([256])
```

```python
wrong = preds.type(y.dtype) != y
X, y, preds = X[wrong], y[wrong], preds[wrong]
labels = [a+'\n'+b for a, b in zip(data.text_labels(y),
data.text_labels(preds))]
data.visualize([X, y], labels=labels)
```

**4. Key takeaways**

- Softmax regression ensures that the predicted probabilities sum to 1, making them interpretable as probabilities which is used for multi-class classification. Regression models don't have this constraint.

- After reading this section one question pops up, what is the role of the cross-entropy loss in softmax regression? Cross-entropy measures the difference between the predicted and actual probability distributions. Minimizing it aims to make the model's predictions as close as possible to the true labels.

# Multilayer Perceptrons

**5.1 Multilayer Perceptrons**

```
import torch
from d2l import torch as d2l

x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = torch.relu(x)
d2l.plot(x.detach(), y.detach(), 'x', 'relu(x)', figsize=(5, 2.5))
```

```
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of relu', figsize=(5, 2.5))
```

```
y = torch.sigmoid(x)
d2l.plot(x.detach(), y.detach(), 'x', 'sigmoid(x)', figsize=(5, 2.5))
```

```
# Clear out previous gradients
x.grad.data.zero_()
y.backward(torch.ones_like(x),retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of sigmoid', figsize=(5, 2.5))
```

```
y = torch.tanh(x)
d2l.plot(x.detach(), y.detach(), 'x', 'tanh(x)', figsize=(5, 2.5))
```

```
# Clear out previous gradients
x.grad.data.zero_()
y.backward(torch.ones_like(x),retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of tanh', figsize=(5, 2.5))
```

### 5.2 Implementation of Multilayer Perceptrons

```
import torch
from torch import nn
from d2l import torch as d2l

class MLPScratch(d2l.Classifier):
    def __init__(self, num_inputs, num_outputs, num_hiddens, lr,
sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens) *
sigma)
        self.b1 = nn.Parameter(torch.zeros(num_hiddens))
        self.W2 = nn.Parameter(torch.randn(num_hiddens, num_outputs) *
sigma)
        self.b2 = nn.Parameter(torch.zeros(num_outputs))

def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)

@d2l.add_to_class(MLPScratch)
def forward(self, X):
    X = X.reshape((-1, self.num_inputs))
    H = relu(torch.matmul(X, self.W1) + self.b1)
    return torch.matmul(H, self.W2) + self.b2

model = MLPScratch(num_inputs=784, num_outputs=10, num_hiddens=256,
lr=0.1)
data = d2l.FashionMNIST(batch_size=256)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```

```python
class MLP(d2l.Classifier):
    def __init__(self, num_outputs, num_hiddens, lr):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(nn.Flatten(),
nn.LazyLinear(num_hiddens),
                                 nn.ReLU(),
nn.LazyLinear(num_outputs))

model = MLP(num_outputs=10, num_hiddens=256, lr=0.1)
trainer.fit(model, data)
```

**5.3 Forward Propagation, Backward Propagation, and Computational Graphs**

In this subsection we learned that forward propagation involves calculating intermediate variables and outputs for a neural network, layer by layer, starting from the input and moving towards the output layer. In a single hidden layer MLP without bias, the intermediate variable z is calculated as the dot product of the input example x and the weight parameter W of the hidden layer. The hidden activation vector h is then obtained by applying an activation function ϕ to z.Finally, the output layer calculates the predicted value y_hat by taking the dot product of the hidden activation vector h and the weight parameter W of the output layer, followed by applying the activation function σ.

The backpropagation is the process of calculating gradients for the weights and biases in the neural network. It's essentially the reverse of forward propagation, where we propagate the error backward through the network to update the parameters. The steps were as follows:

1. Calculate the gradient of the loss function with respect to the output: ∂L/∂y_hat.
2. Calculate the gradient of the output with respect to the output layer weights: ∂y_hat/∂W'.
3. Calculate the gradient of the output with respect to the hidden layer activations: ∂y_hat/∂h.
4. Calculate the gradient of the hidden layer activations with respect to the hidden layer weights: ∂h/∂W.
5. Use the chain rule to combine these gradients to obtain the gradients for the weights and biases.

**5. key takeawyas**

- Activation functions introduce non-linearity into MLPs, allowing them to learn complex patterns that linear models cannot. Without non-linear activations, MLPs would essentially be equivalent to linear regression models. Popular activation functions include ReLU, sigmoid, and tanh.
- We learned that the number of hidden layers and neurons in an MLP directly influences its capacity, or ability to learn complex patterns. More layers and neurons generally increase the model's capacity but can also lead to overfitting if the model becomes too complex for the given dataset. Finding the right balance is crucial.
- Furthermore we can't insert a hidden layer with only a single neuron becuase essentially it becomes a linear transformation, similar to the input layer and the network loses its ability to learn non-linear relationships between features.
- Common optimization algorithms for MLPs include gradient descent, stochastic gradient descent (SGD). These algorithms iteratively adjust the model's parameters based on the gradient of the loss function with respect to those parameters. The goal is to minimize the loss, which typically measures the difference between the model's predictions and the true values.
- As for lr a good learning rate will lead to stable convergence and good accuracy. A very low rate might be too slow, while a high rate might cause oscillations or divergence.
- From what I've learned in my parallell programming courses misaligned matrices lead to cache missed and require additional memory access, leading to slower execution. And GPUs are significantly faster for matrix operations compared to CPUs due to their parallel processing architecture.