定义一个字典

```python
# 定义一个 DataFrame 对象
# 数据使用字典
# 也要列出列的名称以便于一一匹对
df = pd.DataFrame(
    {
        "id": [1001, 1002, 1003, 1004, 1005, 1006],
        "date": pd.date_range("20130102", periods=6),
        "city": ["Beijing", "SH", "guangzhou", "Shenzhen", "shanghai", "BEIJING"],
        "age": [23, 44, 54, 32, 34, 32],
        "category": ["100-A", "100-B", "110-A", "110-C", "210-A", "130-F"],
        "price": [1200, np.nan, 2133, 5433, np.nan, 4432],
    },
    columns=["id", "date", "city", "category", "age", "price"],
)
df
```

	id	date	city	category	age	price
0	1001	2013-01-02	Beijing	100-A	23	1200.0
1	1002	2013-01-03	SH	100-B	44	NaN
2	1003	2013-01-04	guangzhou	110-A	54	2133.0
3	1004	2013-01-05	Shenzhen	110-C	32	5433.0
4	1005	2013-01-06	shanghai	210-A	34	NaN
5	1006	2013-01-07	BEIJING	130-F	32	4432.0

值得注意的是，`date` 数据可以按照时间戳的方式输出。

输出形状

```
# 输出形状
df.shape
```

```
(6, 6)
```

输出信息

```
# 输出信息
df.info
```

```
<bound method DataFrame.info of      id date       city
  category  age   price
0  1001  NaN    Beijing    100-A   23  1200.0
1  1002  NaN         SH    100-B   44     NaN
2  1003  NaN  guangzhou    110-A   54  2133.0
3  1004  NaN   Shenzhen    110-C   32  5433.0
4  1005  NaN   shanghai    210-A   34     NaN
5  1006  NaN    BEIJING    130-F   32  4432.0>
```

返回每行的下标和相应下标的值

```
# 行索引的下标以及相应下标的值
row_index_name_1 = df.index
row_index_name_2 = df.index.values
print(row_index_name_1)
print(row_index_name_2)
```

```
RangeIndex(start=0, stop=6, step=1)
[0 1 2 3 4 5]
```

输出相应变量的值

```
# 输出相应变量的值
df["city"].values
```

```
array(['Beijing', 'SH', 'guangzhou', 'Shenzhen',
'shanghai', 'BEIJING'],
      dtype=object)
```

输出变量的数据类型

```
df.dtypes
```

```
id            int64
date         object
city         object
category     object
age           int64
price       float64
dtype: object
```

判断是否为空

```
# 判断是否为空
df["price"].isnull()
```

```
0    False
1     True
2    False
3    False
4     True
5    False
Name: price, dtype: bool
```

去除重复值

```
# 去除重复值
df["age"].unique()
```

```
array([23, 44, 54, 32, 34], dtype=int64)
```

筛选

```
dataframe = pd.DataFrame({"a": [1, 2, 3], "b": ["aaa", "bbb", "ccc"]})

# 选取属性 "a" 中大于 1 的行:
dataframe[dataframe.a > 1]
```

	a	b
1	2	bbb
2	3	ccc

```
# 选取属性 "a" 中大于 1 的 "b" 列:
dataframe["b"][dataframe.a > 1]
```

```
1    bbb
2    ccc
Name: b, dtype: object
```

```
# 选取属性 "b" 在 list 中的行
array = np.array(["aaa", "ccc"])
dataframe[dataframe.b.isin(array)]
```

	a	b
0	1	aaa
2	3	ccc

```
dataframe_1 = df.loc[df["city"] == "Beijing", ["age", "id"]]
dataframe_1
```

	age	id
0	23	1001

```
dataframe_2 = df.loc[df["age"] < 34, ["city", "id"]]
dataframe_2
```

	city	id
0	Beijing	1001
3	Shenzhen	1004
5	BEIJING	1006

```
dataframe_3 = df.loc[(df["age"] < 50) & (df["price"] < 6000), ["id", "city"]]
dataframe_3
```

	id	city
0	1001	Beijing
3	1004	Shenzhen
5	1006	BEIJING

填充空缺值

```
# 填充空缺值，如果在原对象修改，则添加 inplace 参数，并置为 True，即 fillna(value=10, inplace=True)
new_df = df.fillna(value=10)
new_df
```

	id	date	city	category	age	price
0	1001	2013-01-02	Beijing	100-A	23	1200.0

	id	date	city	category	age	price
1	1002	2013-01-02	SH	110-B	45	1000
2	1003	2013-01-04	guangzhou	110-A	54	2133.0
3	1004	2013-01-05	Shenzhen	110-C	32	5433.0
4	1005	2013-01-06	shanghai	210-A	34	10.0
5	1006	2013-01-07	BEIJING	130-F	32	4432.0

```
# 均值填充
df["price"] =
df["price"].fillna(value=df["price"].mean())
# 线性填充
df["price"] = df["price"].interpolate()
# 三次样条填充
df["price"] = df["price"].interpolate()
```

字符处理方法

```
df["city"] = df["city"].str.upper()
df
```

数据处理方法

改变类型

```
df["price"] = df["price"].astype(int)
df
```

去重

```
df["city"] = df["city"].drop_duplicates()
df
```

合并

```
df_inner = pd.merge(df, df1, how="inner")
df_left = pd.merge(df, df1, how="left")
df_right = pd.merge(df, df1, how="right")
df_outer = pd.merge(df, df1, how="outer")
```

`left` 和 `right` 代表着左合并与右合并，一个以左边数据集为基础，一个以右边数据集为基础。

`inner` 只对两个所共用的进行保存，`outer` 是并集。

设置下标

```
df_inner.set_index("id")
```

将数据中的某一列设置为下标。

	date	city	category	age	price	gender	pay	m-point
id								
1001	2013-01-02	Beijing	100-A	23	1200	male	Y	10
1002	2013-01-03	Sh	100-B	44	3299	female	N	12
1003	2013-01-04	Guangzhou	110-A	54	2133	male	Y	20
1004	2013-01-05	Shenzhen	110-C	32	5433	female	Y	40
1005	2013-01-06	Shanghai	210-A	34	3299	male	N	40
1006	2013-01-07	NaN	130-F	32	4432	female	Y	40

排序

根据某一列进行排序

```
df_inner.sort_values(by=["price"])
```

根据下标进行排序

```
df_inner = df_inner.sort_index()
df_inner
```

分组

```
data = {
    'Product': ['A', 'B', 'A', 'B', 'A', 'B'],
    'Sales': [100, 200, 150, 300, 120, 250],
    'Quantity': [10, 15, 12, 20, 8, 18]
}

df = pd.DataFrame(data)

grouped = df.groupby("Product")
total_quantities = grouped["Quantity"].sum()
total_quantities
```

```
Product
A    30
B    53
Name: Quantity, dtype: int64
```

抽样

在 *Pandas* 中，`sample()` 方法用于从 `DataFrame` 中随机抽取指定数量或比例的样本。下面是该方法的基本语法和一些示例用法：

1. **基本语法**：

```
df.sample(n=None, frac=None, replace=False,
random_state=None)
```

- `n`：要抽取的样本数量。
- `frac`：要抽取的样本比例，取值范围为 [0, 1]。
- `replace`：是否允许重复抽样，默认为 False。
- `random_state`：随机种子，用于复现随机抽样结果。

参数 `n` 和 `frac` 二选一，如果同时指定了，`frac` 会被忽略。

2. **示例**：

假设有一个包含学生信息的 `DataFrame`：

```
import pandas as pd

data = {
    'StudentID': [1, 2, 3, 4, 5],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [20, 21, 22, 23, 24]
}

df = pd.DataFrame(data)
print(df)
```

输出：

```
   StudentID     Name  Age
0          1    Alice   20
1          2      Bob   21
2          3  Charlie   22
3          4    David   23
4          5      Eva   24
```

现在，我们可以使用 `sample()` 方法随机抽取一定数量的样本：

```
# 抽取 2 个样本
sample1 = df.sample(n=2)
print(sample1)
```

输出：

```
   StudentID     Name  Age
3          4    David   23
2          3  Charlie   22
```

你也可以指定抽样的比例：

```python
# 抽取 50% 的样本
sample2 = df.sample(frac=0.5)
print(sample2)
```

输出：

```
   StudentID   Name  Age
0          1  Alice   20
4          5    Eva   24
```

如果你希望允许重复抽样，可以设置 `replace=True`：

```python
# 允许重复抽样，抽取 3 个样本
sample3 = df.sample(n=3, replace=True)
print(sample3)
```

输出：

```
   StudentID     Name  Age
2          3  Charlie   22
3          4    David   23
2          3  Charlie   22
```

生成统计摘要

```python
# 生成统计摘要
df.describe()
```

	Sales	Quantity
count	6.000000	6.000000
mean	186.666667	13.833333
std	77.888810	4.665476

	Sales	Quantity
min	100.000000	8.000000
25%	127.500000	10.500000
50%	175.000000	13.500000
75%	237.500000	17.250000
max	300.000000	20.000000