Pandas读取文件

当使用 Pandas 做数据分析的时，需要读取事先准备好的数据集，这是做数据分析的第一步。Panda 提供了多种读取数据的方法：

```
read_csv() 用于读取文本文件
read_excel() 用于读取文本文件
read_json() 用于读取 json 文件
read_sql_query() 读取 sql 语句的,
```

通用流程

```
1. 导入库 import pandas as pd
2. 找到文件所在位置（绝对路径 = 全称）（相对路径 = 和程序在同一个文件夹中的路径的简称）
3. 变量名 = pd.读写操作方法（文件路径，具体的筛选条件，……）
```

CSV文件读取

CSV 又称逗号分隔值文件，是一种简单的文件格式，以特定的结构来排列表格数据。 CSV 文件能够以纯文本形式存储表格数据，比如电子表格、数据库文件，并具有数据交换的通用格式。CSV 文件会在 Excel 文件中被打开，其行和列都定义了标准的数据格式。

将 CSV 中的数据转换为 DataFrame 对象是非常便捷的。和一般文件读写不一样，它不需要你做打开文件、读取文件、关闭文件等操作。相反，您只需要一行代码就可以完成上述所有步骤，并将数据存储在 DataFrame 中。

下面进行实例演示，源数据如下：

	A	B	C	D	E
1	col1	col2	col3	col4	col5
2	2	a	1.4	apple	2022/1/1
3	3	b	3.4	banana	2022/1/2
4	6	c	2.5	orange	2020/1/5
5	5	d	3.2	grape	2020/1/7

读取文件:

```
import pandas as pd
# 读取csv文件, 相对路径
#df = pd.read_csv("data/my_csv.csv")
df = pd.read_csv("./data/my_csv.csv")
# os动态取得绝对路径 os.getcwd() os.path.json
#df = pd.read_csv(r"D:\2021年财贸数据\呼和浩特\课件\数据分析\pandas\第16天\data\my_csv.csv")
print(df,type(df))
```

```
   col1 col2  col3    col4      col5
0     2    a   1.4   apple  2022/1/1
1     3    b   3.4  banana  2022/1/2
2     6    c   2.5  orange  2022/1/5
3     5    d   3.2   grape  2022/1/7 <class 'pandas.core.frame.DataFrame'>
```

```
import os
os.getcwd()
```

```
'D:\\桌面\\数据分析-班级\\3-4班\\第16天'
```

方法详细说明:

```
read_csv(filepath_or_buffer, sep=',', header='infer', names=None, index_col=None, usecols=None, squeeze=None, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False,
```

```
skiprows=None, skipfooter=0, nrows=None, na_values=None,
keep_default_na=True, na_filter=True, verbose=False,
skip_blank_lines=True, parse_dates=None,
infer_datetime_format=False, keep_date_col=False, date_parser=None,
dayfirst=False, cache_dates=True, iterator=False, chunksize=None,
compression='infer', thousands=None, decimal='.',
lineterminator=None, quotechar='"', quoting=0, doublequote=True,
escapechar=None, comment=None, encoding=None,
encoding_errors='strict', dialect=None, error_bad_lines=None,
warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False,
low_memory=True, memory_map=False, float_precision=None,
storage_options=None)
```

一、基本参数

```
1、filepath_or_buffer：数据输入的路径：可以是文件路径、可以是URL，也可以是实现read方法的任意对象。这个参数，就是我们输入的第一个参数。
```

```
import pandas as pd
pd.read_csv(r"data\students.csv")
```

```
# 还可以是一个URL，如果访问该URL会返回一个文件的话，那么pandas的read_csv函数会自动将
# 该文件进行读取。比如：我们服务器上放的数据，将刚才的文件返回。
pd.read_csv("http://my-teaching.top/static/data/students.csv") # 需要网络请求,因此读取文件比较慢
```

```
# 里面还可以是一个 _io.TextIOWrapper，比如：
f = open(r"data\students.csv", encoding="utf-8")
pd.read_csv(f)
# pandas默认使用utf-8读取文件
```

```
2、sep：读取csv文件时指定的分隔符，默认为逗号。注意："csv文件的分隔符" 和 "我们读取csv文件时指定的分隔符" 一定要一致。
```

```
import pandas as pd
pd.read_csv(r"data\students_step.csv")
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id|name|address|gender|birthday
0	1|朱梦雪|地球村|女|2004/11/2
1	2|许文博|月亮星|女|2003/8/7
2	3|张兆媛|艾尔星|女|2004/11/2
3	4|付延旭|克哈星|男|2003/10/11
4	5|王杰|查尔星|男|2002/6/12
5	6|董泽宇|塔桑尼斯|男|2002/2/12

由于指定的分隔符 和 csv文件采用的分隔符 不一致，因此多个列之间没有分开，而是连在一起了。 所以，我们需要将分隔符设置成"\t"才可以。

```
df = pd.read_csv(r"data\students_step.csv", sep="|")
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12
5	6	董泽宇	塔桑尼斯	男	2002/2/12

```
3.delim_whitespace ：默认为 False，设置为 True 时，表示分割符为空白字符，可以是空格、"\t"等等。不管分隔符是什么，只要是空白字符，那么可以通过delim_whitespace=True进行读取。
```

```
df = pd.read_csv(r"data\students_whitespace.txt", sep=" ")
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博\t月亮星	女	2003/8/7	NaN
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰\t查尔星	男	2002/6/12	NaN
5	6	董泽宇\t塔桑尼斯	男	2002/2/12	NaN

```
df = pd.read_csv(r"data\students_whitespace.txt",
delim_whitespace=True)
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12
5	6	董泽宇	塔桑尼斯	男	2002/2/12

- header:用作列名的行号，以及数据的开头。默认行为是推断列名：如果没有传递任何名称，则该行为与header=0相同，并且从文件的第一行推断列名，如果显式传递列名，则该行为与header=None相同。显式传递header=0以替换现有名称。标题可以是整数列表，指定列上多索引的行位置，例如[0,1,3]。未指定的中间行将被跳过（例如，本例中跳过2行）。请注意，如果skip_blank_lines=True，此参数将忽略注释行和空行，因此header=0表示数据的第一行，而不是文件的第一行。
- names:当names没被赋值时，header会变成0，即选取数据文件的第一行作为列名；当names 被赋值，header 没被赋值时，那么header会变成None。如果都赋值，就会实现两个参数的组合功能。

1) names 没有被赋值，header 也没赋值：

```
# 这种情况下，header为0，即选取文件的第一行作为表头
pd.read_csv(r"data\students.csv")
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12
5	6	董泽宇	塔桑尼斯	男	2002/2/12

2) names 没有被赋值，header 被赋值：

```
# 不指定names，指定header为1，则选取第二行当做表头，第二行下面为数据
pd.read_csv(r"data\students.csv", header=1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	1	朱梦雪	地球村	女	2004/11/2
0	2	许文博	月亮星	女	2003/8/7
1	3	张兆媛	艾尔星	女	2004/11/2
2	4	付延旭	克哈星	男	2003/10/11
3	5	王杰	查尔星	男	2002/6/12
4	6	董泽宇	塔桑尼斯	男	2002/2/12

3) names 被赋值，header 没有被赋值：

```
pd.read_csv(r"data\students.csv", names=["编号", "姓名", "地址", "性别", "出生日期"])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	编号	姓名	地址	性别	出生日期
0	id	name	address	gender	birthday
1	1	朱梦雪	地球村	女	2004/11/2
2	2	许文博	月亮星	女	2003/8/7
3	3	张兆媛	艾尔星	女	2004/11/2
4	4	付延旭	克哈星	男	2003/10/11
5	5	王杰	查尔星	男	2002/6/12
6	6	董泽宇	塔桑尼斯	男	2002/2/12

可以看到，names适用于没有表头的情况，指定names没有指定header，那么header相当于None。
一般来说，读取文件的时候会有一个表头，一般默认是第一行，但是有的文件中是没有表头的，那么这个时候就可以通过names手动指定、或者生成表头，而文件里面的数据则全部是内容。所以这里id、name、address、date也当成是一条记录了，本来它是表头的，但是我们指定了names，所以它就变成数据了，表头是我们在names里面指定的。

4) names和header都被赋值：

```
pd.read_csv(r"data\students.csv",
            names=["编号", "姓名", "地址", "性别", "出生日期"],
            header=0)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	编号	姓名	地址	性别	出生日期
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12
5	6	董泽宇	塔桑尼斯	男	2002/2/12

这个时候，相当于先不看names，只看header，header为0代表先把第一行当做表头，下面的当成数据；然后再把表头用names给替换掉。

所以names和header的使用场景主要如下：

```
①. csv文件有表头并且是第一行，那么names和header都无需指定；
②. csv文件有表头、但表头不是第一行，可能从下面几行开始才是真正的表头和数据，这个时候指定header即可；
③. csv文件没有表头，全部是纯数据，那么我们可以通过names手动生成表头；
④. csv文件有表头、但是这个表头你不想用，这个时候同时指定names和header。先用header选出表头和数据，然后再用names将表头替换掉，就等价于将数据读取进来之后再对列名进行rename；

4 index_col：我们在读取文件之后所得到的DataFrame的索引默认是0、1、2……，我们可以通过set_index设定索引，但是也可以在读取的时候就指定某列为索引。
```

```

df = pd.read_csv(r"data\students.csv", index_col="birthday")
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender
birthday				
2004/11/2	1	朱梦雪	地球村	女
2003/8/7	2	许文博	月亮星	女
2004/11/2	3	张兆媛	艾尔星	女
2003/10/11	4	付延旭	克哈星	男
2002/6/12	5	王杰	查尔星	男
2002/2/12	6	董泽宇	塔桑尼斯	男

```
df = pd.read_csv(r"data\students.csv")
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12
5	6	董泽宇	塔桑尼斯	男	2002/2/12

```
df.index=df['birthday']
del df['birthday']
df
```


```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender
birthday				
2004/11/2	1	朱梦雪	地球村	女
2003/8/7	2	许文博	月亮星	女
2004/11/2	3	张兆媛	艾尔星	女
2003/10/11	4	付延旭	克哈星	男
2002/6/12	5	王杰	查尔星	男
2002/2/12	6	董泽宇	塔桑尼斯	男

```
df.loc['2004/11/2']
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender
birthday				
2004/11/2	1	朱梦雪	地球村	女
2004/11/2	3	张兆媛	艾尔星	女

```
df.index
```

```
Index(['2004/11/2', '2003/8/7', '2004/11/2', '2003/10/11',
'2002/6/12',
       '2002/2/12'],
      dtype='object', name='birthday')
```

```
# to_datetime
pd.to_datetime(df.index)
```

```
DatetimeIndex(['2004-11-02', '2003-08-07', '2004-11-02', '2003-10-11',
               '2002-06-12', '2002-02-12'],
              dtype='datetime64[ns]', name='birthday', freq=None)
```

```
df.index = pd.to_datetime(df.index)
```

```
df.index
```

```
DatetimeIndex(['2004-11-02', '2003-08-07', '2004-11-02', '2003-10-11',
               '2002-06-12', '2002-02-12'],
              dtype='datetime64[ns]', name='birthday', freq=None)
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender
birthday				
2004-11-02	1	朱梦雪	地球村	女
2003-08-07	2	许文博	月亮星	女
2004-11-02	3	张兆媛	艾尔星	女
2003-10-11	4	付延旭	克哈星	男
2002-06-12	5	王杰	查尔星	男
2002-02-12	6	董泽宇	塔桑尼斯	男

```
df['2004']
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender
birthday				
2004-11-02	1	朱梦雪	地球村	女
2004-11-02	3	张兆媛	艾尔星	女

```
df['2002-02']
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender
birthday				
2002-02-12	6	董泽宇	塔桑尼斯	男

这里，我们在读取的时候指定了name列作为索引；
此外，除了指定单个列，还可以指定多列作为索引，比如["id", "name"]。同时，我们除了可以输入列名外，还可以输入列对应的索引。比如："id"、"name"、"address"、"date"对应的索引就分别是0、1、2、3。

```
df2 = pd.read_csv(r"data\students.csv", index_col=
["gender","birthday"])
df2
```


```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

		id	name	address
gender	**birthday**			
女	**2004/11/2**	1	朱梦雪	地球村
	2003/8/7	2	许文博	月亮星
	2004/11/2	3	张兆媛	艾尔星
男	**2003/10/11**	4	付延旭	克哈星
	2002/6/12	5	王杰	查尔星
	2002/2/12	6	董泽宇	塔桑尼斯

```
df2.loc["女"]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address
birthday			
2004/11/2	1	朱梦雪	地球村
2003/8/7	2	许文博	月亮星
2004/11/2	3	张兆媛	艾尔星

```
df2.sort_index(inplace=True)
df2
```

```
type(df2.loc["男"])
```

```
5. usecols：返回列的子集。如果是类似列表的，则所有元素都必须是位置性的（即文档列中的整数索引），或者是与用户在名称中提供的列名或从文档标题行推断的列名相对应的字符串。如果给出了名称，则不考虑文档标题行
```

```
pd.read_csv(r"data\students.csv", usecols=["name","birthday"])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name
0	朱梦雪
1	许文博
2	张兆媛
3	付延旭
4	王杰
5	董泽宇

二、通用解析参数

```
1. encoding:这只编码格式  utf-8  gbk
```

```
pd.read_csv(r"data\students_gbk.csv") # UnicodeDecodeError
```

```
# 如果提示错误喂UnicodeDecodeError --->需要想到编码问题.
# 默认pandas使用utf-8格式读取.
pd.read_csv(r"data\students_gbk.csv", encoding="gbk")
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12
5	6	董泽宇	塔桑尼斯	男	2002/2/12

2. dtype：在读取数据的时候，设定字段的类型。比如，公司员工的id一般是：00001234，如果默认读取的时候，会显示为1234，所以这个时候要把他转为字符串类型，才能正常显示为00001234：

```
df = pd.read_csv(r"data\students_step_001.csv", sep="|")
df
```


```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12
5	6	董泽宇	塔桑尼斯	男	2002/2/12

```

df = pd.read_csv(r"data\students_step_001.csv", sep="|", dtype = {"id":str})
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	001	朱梦雪	地球村	女	2004/11/2
1	002	许文博	月亮星	女	2003/8/7
2	003	张兆媛	艾尔星	女	2004/11/2
3	004	付延旭	克哈星	男	2003/10/11
4	005	王杰	查尔星	男	2002/6/12
5	006	董泽宇	塔桑尼斯	男	2002/2/12

3. converters：在读取数据的时候对列数据进行变换，例如将id增加10，但是注意int(x)，在使用converters参数时，解析器默认所有列的类型为 str，所以需要进行类型转换。

```
pd.read_csv('data\students.csv', converters={"id": lambda x: int(x) + 10})
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	11	朱梦雪	地球村	女	2004/11/2
1	12	许文博	月亮星	女	2003/8/7
2	13	张兆媛	艾尔星	女	2004/11/2
3	14	付延旭	克哈星	男	2003/10/11
4	15	王杰	查尔星	男	2002/6/12
5	16	董泽宇	塔桑尼斯	男	2002/2/12

4.true_values和false_values：指定哪些值应该被清洗为True，哪些值被清洗为False。

```
pd.read_csv('data\students.csv', true_values=['男'], false_values=['女'])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	False	2004/11/2
1	2	许文博	月亮星	False	2003/8/7
2	3	张兆媛	艾尔星	False	2004/11/2
3	4	付延旭	克哈星	True	2003/10/11
4	5	王杰	查尔星	True	2002/6/12
5	6	董泽宇	塔桑尼斯	True	2002/2/12

```
pd.read_csv('data\students.csv', true_values=['女'])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12
5	6	董泽宇	塔桑尼斯	男	2002/2/12

这里的替换规则为，只有当某一列的数据类别全部出现在true_values + false_values里面，才会被替换。

我们看到"错"并没有被替换成False，原因就是result字段中只有"错"这个类别的值在true_values + false_values中，而"对"并没有出现，所以不会替换。

而最后的对、错都出现在了true_values + false_values中，所以全部被替换。

```
5、skiprows：表示过滤行，想过滤掉哪些行，就写在一个列表里面传递给skiprows即可。注意的是：这里是先过滤，然后再确定表头，比如：
```

```
pd.read_csv('data\students.csv', skiprows=[0,3])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	1	朱梦雪	地球村	女	2004/11/2
0	2	许文博	月亮星	女	2003/8/7
1	4	付延旭	克哈星	男	2003/10/11
2	5	王杰	查尔星	男	2002/6/12
3	6	董泽宇	塔桑尼斯	男	2002/2/12

这里把第一行过滤掉了，因为第一行是表头，所以在过滤掉之后第二行就变成表头了。

当然里面除了传入具体的数值，来表明要过滤掉哪些行，还可以传入一个函数。

```
pd.read_csv('data\students.csv', skiprows=lambda x: x > 0 and x % 2 == 0)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	3	张兆媛	艾尔星	女	2004/11/2
2	5	王杰	查尔星	男	2002/6/12

由于索引从0开始，所以凡是索引大于0、并且%2等于0的记录都过滤掉。索引大于0，是为了保证表头不被过滤掉。

```
6、skipfooter：从文件末尾过滤行
```

```
pd.read_csv('data\students.csv', skipfooter=1)
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support skipfooter; you can avoid this warning by specifying engine='python'.
  """Entry point for launching an IPython kernel.
```


```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	鏈辨vi闆€	鏃扮悊鏃€	濂€	2004/11/2
1	2	璁告构鍗€	鏈堜寒鎸€	濂€	2003/8/7
2	3	寮犲厜濯€	鑼惧皵鎸€	濂€	2004/11/2
3	4	浠樺欢鏃€	鍏嬪搸鎸€	鑹€	2003/10/11
4	5	鏊嬪寮	鏌ュ皵鎸€	鑹€	2002/6/12

```
pd.read_csv('data\students.csv', skipfooter=1, engine="python", encoding="utf-8")
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12

pandas解析数据时用的引擎，目前解析引擎有两种：c、python。默认为 c，因为 c 引擎解析速度更快，但是特性没有 python 引擎全

skipfooter接收整型，表示从结尾往上过滤掉指定数量的行，因为引擎退化为python，那么要手动指定engine="python"，不然会警告。另外需要指定encoding="utf-8"，因为csv存在编码问题，当引擎退化为python的时候，在Windows上读取会乱码。

```
7、nrows：设置一次性读入的文件行数，在读入大文件时很有用，比如 16G 内存的PC无法容纳几百 G 的大文件。
```

```
pd.read_csv('data\students.csv', nrows=3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2

三、空值处理相关参数

```
na_values：该参数可以配置哪些值需要处理成 NaN：
```

```
pd.read_csv('data\students.csv', na_values=["女", "朱梦雪"])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	NaN	地球村	NaN	2004/11/2
1	2	许文博	月亮星	NaN	2003/8/7
2	3	张兆媛	艾尔星	NaN	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12
5	6	董泽宇	塔桑尼斯	男	2002/2/12

可以看到将"女"和"朱梦雪"设置成了NaN，这里的情况是不同的列中包含了不同的值。

四、时间处理相关参数

```
1、parse_dates：指定某些列为时间类型，这个参数一般搭配date_parser使用。

2、date_parser：是用来配合parse_dates参数的，因为有的列虽然是日期，但没办法直接转化，需要我们指定一个解析格式：
```

```
df = pd.read_csv('data\students.csv')
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004/11/2
1	2	许文博	月亮星	女	2003/8/7
2	3	张兆媛	艾尔星	女	2004/11/2
3	4	付延旭	克哈星	男	2003/10/11
4	5	王杰	查尔星	男	2002/6/12
5	6	董泽宇	塔桑尼斯	男	2002/2/12

```
df.dtypes
```

```
id           int64
name        object
address     object
gender      object
birthday    object
dtype: object
```

```
df = pd.read_csv('data\students.csv', parse_dates=["birthday"])
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	name	address	gender	birthday
0	1	朱梦雪	地球村	女	2004-11-02
1	2	许文博	月亮星	女	2003-08-07
2	3	张兆媛	艾尔星	女	2004-11-02
3	4	付延旭	克哈星	男	2003-10-11
4	5	王杰	查尔星	男	2002-06-12
5	6	董泽宇	塔桑尼斯	男	2002-02-12

```
df.dtypes
```

```
df2 = pd.read_csv('data\students_年月日.csv', parse_dates=["birthday"])
print(df2)
df2.dtypes
```

```
   id name address gender     birthday
0   1  朱梦雪      地球村      女   2004年11月2日
1   2  许文博      月亮星      女    2003年8月7日
2   3  张兆媛      艾尔星      女   2004年11月2日
3   4  付延旭      克哈星      男  2003年10月11日
4   5   王杰       查尔星      男   2002年6月12日
5   6  董泽宇     塔桑尼斯      男   2002年2月12日
```

```
id            int64
name         object
address      object
gender       object
birthday     object
dtype: object
```

```
import pandas as pd
from datetime import datetime

df2 = pd.read_csv('data\students_年月日.csv',
                  parse_dates=["birthday"],
                  date_parser=lambda x: datetime.strptime(x, "%Y年%m月%d日"))
print(df2)
df2.dtypes
```

```
   id name address gender   birthday
0   1  朱梦雪      地球村       女 2004-11-02
1   2  许文博      月亮星       女 2003-08-07
2   3  张兆媛      艾尔星       女 2004-11-02
3   4  付延旭      克哈星       男 2003-10-11
4   5   王杰       查尔星       男 2002-06-12
5   6  董泽宇     塔桑尼斯       男 2002-02-12
```

```
id                   int64
name                object
address             object
gender              object
birthday    datetime64[ns]
dtype: object
```

五、分块读入相关参数

```
1、iterator：迭代器，iterator 为 bool类型，默认为False。如果为True，那么返回一个 TextFileReader 对象，以便逐块处理文件。这个在文件很大、内存无法容纳所有数据文件时，可以分批读入，依次处理。
```

```
chunk = pd.read_csv('data\students.csv', iterator=True)
chunk
```

```
<pandas.io.parsers.TextFileReader at 0x1b27f00ef88>
```

```
print(chunk.get_chunk(2))
```

```
   id name address gender   birthday
0   1  朱梦雪      地球村       女  2004/11/2
1   2  许文博      月亮星       女   2003/8/7
```

```
# 文件还剩下三行，但是我们指定读取100，那么也不会报错，不够指定的行数，那么有多少返回多少
print(chunk.get_chunk(100))
```

```
   id name address gender    birthday
2   3  张兆媛      艾尔星       女   2004/11/2
3   4  付延旭      克哈星       男  2003/10/11
4   5   王杰       查尔星       男   2002/6/12
5   6  董泽宇     塔桑尼斯       男   2002/2/12
```

```
chunk.get_chunk(5)
```

```
try:
    # 但是在读取完毕之后，再读的话就会报错了
    chunk.get_chunk(5)
except StopIteration as e:
    print("读取完毕")
# 读取完毕
```

```
读取完毕
```

```
2、chunksize：整型，默认为 None，设置文件块的大小。
```

```
chunk = pd.read_csv('data\students.csv', chunksize=2)
# 还是返回一个类似于迭代器的对象
print(chunk)
# <pandas.io.parsers.TextFileReader object at 0x0000025501143AF0>

# 调用get_chunk，如果不指定行数，那么就是默认的chunksize
print(chunk.get_chunk())
```

```
<pandas.io.parsers.TextFileReader object at 0x000001B27F05C5C8>
   id name address gender   birthday
0   1  朱梦雪      地球村       女  2004/11/2
1   2  许文博      月亮星       女   2003/8/7
```

```
print(chunk.get_chunk())
```

```
   id name address gender    birthday
2   3  张兆媛      艾尔星       女   2004/11/2
3   4  付延旭      克哈星       男  2003/10/11
```

```
print(chunk.get_chunk())
```

```
   id name address gender   birthday
4   5   王杰      查尔星       男  2002/6/12
5   6  董泽宇     塔桑尼斯       男  2002/2/12
```

```
# 也可以指定
print(chunk.get_chunk(100))
```

```
try:
    chunk.get_chunk(5)
except StopIteration as e:
    print("读取完毕")
# 读取完毕
```

```
读取完毕
```

以上便是pandas的read_csv函数中绝大部分参数了，同时其中的部分参数也适用于读取其它类型的文件。

其实在读取csv文件时所使用的参数不多，很多参数平常我们都不会用到的，不过不妨碍我们了解一下，因为在某些特定的场景下它们是可以很方便地帮我们解决一些问题的。个人感觉分块读取这个参数最近在工作中提高了很大的效率。

上面列到的read_csv函数中的参数并不是全部，有几个还没有介绍到

```

```