



大数据，成就未来



# pandas统计分析基础

2018/9/26

# pandas统计分析基础

---

## 概述

- 统计分析是数据分析的重要组成部分，它几乎贯穿了整个数据分析的流程。
- 运用统计方法，将定量与定性结合，进行的研究活动叫作统计分析。
- 统计分析除了包含单一数值型特征的数据集中趋势、分散趋势和峰度与偏度等统计知识外，还包含了多个特征比较计算等知识。
- 本章将介绍使用pandas库进行统计分析所需要掌握的基本知识。

# pandas统计分析基础

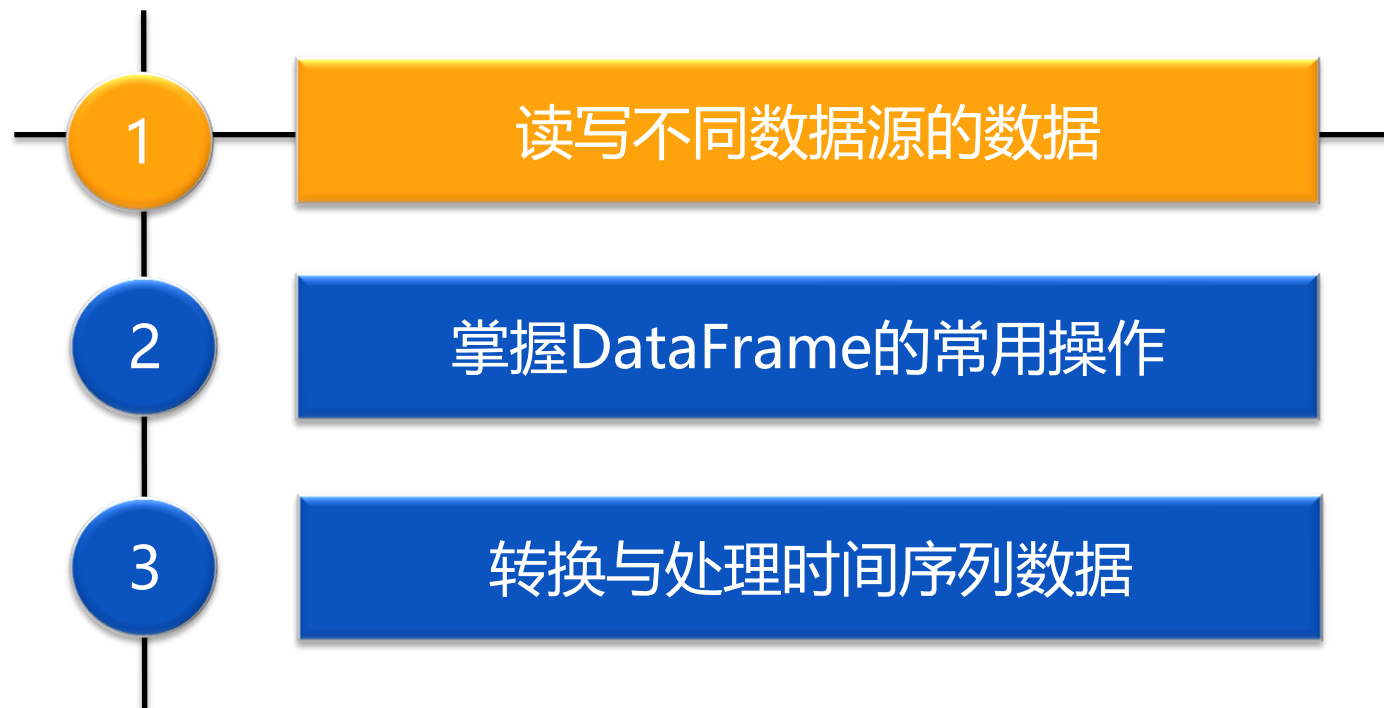
---

## 学习目标

- (1)掌握常见的数据读取方式;
- (2)掌握DataFrame常用属性与方法;
- (3)掌握基础时间数据处理方法;
- (4)掌握分组聚合的原理与方法;
- (5)掌握透视表与交叉表的制作。

# 目录

---



# 读写不同数据源的数据

---

- 数据读取是进行数据预处理、建模与分析的前提。不同的数据源，需要使用不同的函数读取。
- pandas 内置了10余种数据源读取函数和对应的数据写入函数。
- 常见的数据源有3种，分别是数据库数据、文本文件(包括一般文本文件和CSV文件)和Excel文件。
- 掌握这3种数据源读取方法，便能够完成80%左右的数据读取工作。
- 以餐饮销售数据为例，该数据存在于3个系统中。3个系统中的数据源并不相同，所以需要多种数据读取方式读取相应数据。

# 读写不同数据源的数据


## 任务分析


➤ 读取订单详情数据库数据


➤ 读取订单信息表CSV数据


➤ 读取客户信息Excel数据

➤ .sql文件是MySQL数据库的备份文件

 meal\_order\_detail.xlsx

 meal\_order\_detail1.sql

 meal\_order\_detail2.sql

 meal\_order\_detail3.sql

 meal\_order\_info.csv

 users.xlsx

# 读写数据库数据

---

## 什么是数据库

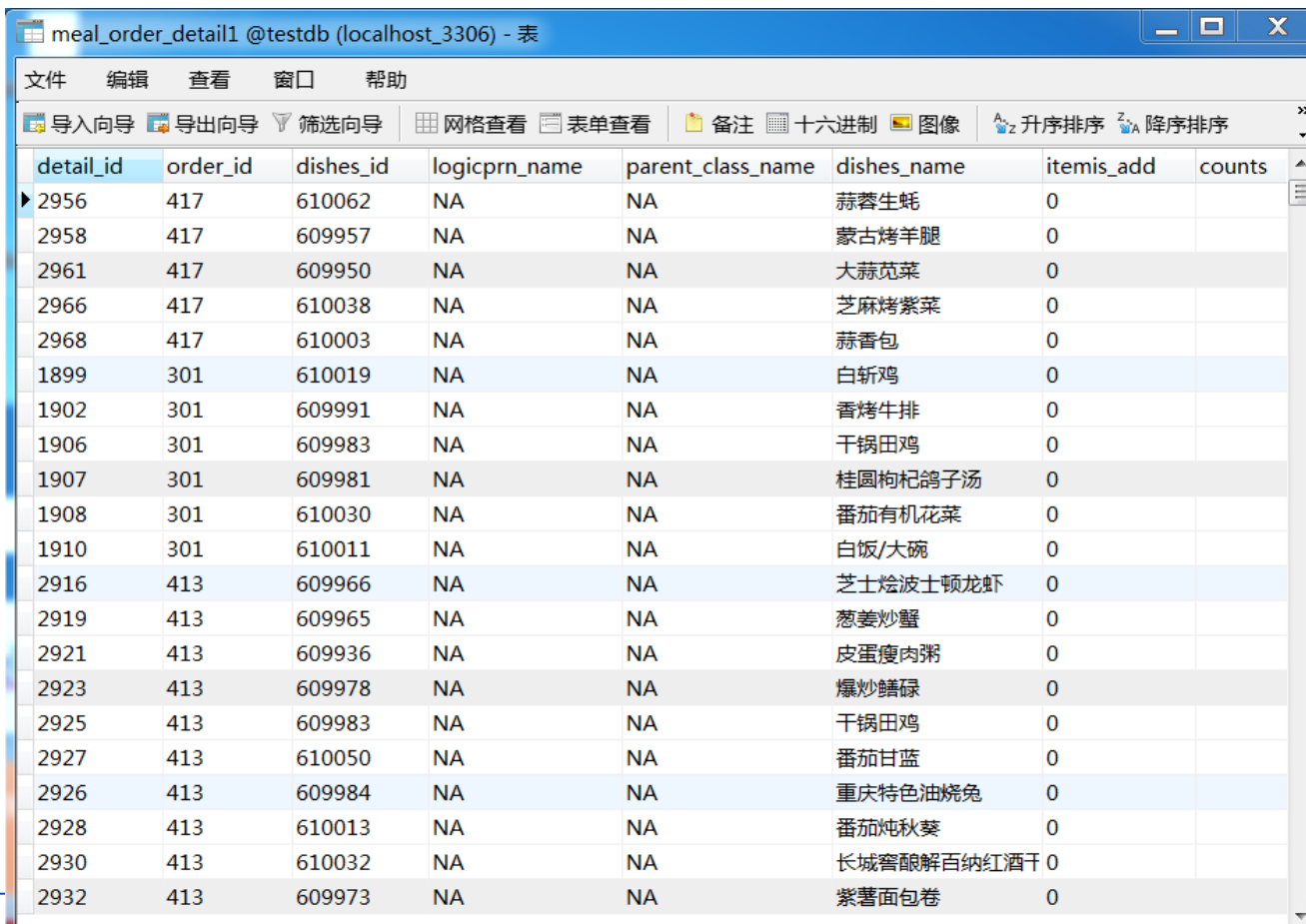
- 数据库 (Database) 是按照数据结构来组织、存储和管理数据的仓库;
- 每个数据库都有一个或多个不同的API用于创建, 访问, 管理, 搜索和复制所保存的数据。
- 我们也可以将数据存储于文件中, 但是在文件中读写数据速度相对较慢。
- 所以, 现在我们使用关系型数据库管理系统 (RDBMS) 来存储和管理的大数据量。
- 所谓的关系型数据库, 是建立在关系模型基础上的数据库, 借助于集合代数等数学概念和方法来处理数据库中的数据。

# 读写数据库数据

## 关系型数据库管理系统 (RDBMS)

➤ RDBMS即关系数据库管理系统(Relational Database Management System)的特点:

- 1.数据以表格的形式出现
- 2.每行为各种记录名称
- 3.每列为记录名称所对应的数据域
- 4.许多的行和列组成一张表单
- 5.若干的表单组成database



detail_id	order_id	dishes_id	logicprn_name	parent_class_name	dishes_name	itemis_add	counts
2956	417	610062	NA	NA	蒜蓉生蚝	0	
2958	417	609957	NA	NA	蒙古烤羊腿	0	
2961	417	609950	NA	NA	大蒜苋菜	0	
2966	417	610038	NA	NA	芝麻烤紫菜	0	
2968	417	610003	NA	NA	蒜香包	0	
1899	301	610019	NA	NA	白斩鸡	0	
1902	301	609991	NA	NA	香烤牛排	0	
1906	301	609983	NA	NA	干锅田鸡	0	
1907	301	609981	NA	NA	桂圆枸杞鸽子汤	0	
1908	301	610030	NA	NA	番茄有机花菜	0	
1910	301	610011	NA	NA	白饭/大碗	0	
2916	413	609966	NA	NA	芝士焗波士顿龙虾	0	
2919	413	609965	NA	NA	葱姜炒蟹	0	
2921	413	609936	NA	NA	皮蛋瘦肉粥	0	
2923	413	609978	NA	NA	爆炒鳝碌	0	
2925	413	609983	NA	NA	干锅田鸡	0	
2927	413	610050	NA	NA	番茄甘蓝	0	
2926	413	609984	NA	NA	重庆特色油烧兔	0	
2928	413	610013	NA	NA	番茄炖秋葵	0	
2930	413	610032	NA	NA	长城窖酿解百纳红酒干	0	
2932	413	609973	NA	NA	紫薯面包卷	0	



# 读写数据库数据

---

## RDBMS 术语

- 在我们开始学习MySQL 数据库前, 让我们先了解下RDBMS的一些术语:
  - 数据库: 数据库是一些关联表的集合。
  - 数据表: 表是数据的矩阵。在一个数据库中的表看起来像一个简单的电子表格。
  - 列: 一列(数据元素) 包含了相同的数据, 例如邮政编码的数据。
  - 行: 一行 (=元组, 或记录) 是一组相关的数据, 例如一条用户订阅的数据。
  - 冗余: 存储两倍数据, 冗余降低了性能, 但提高了数据的安全性。

# 读写数据库数据

---

## RDBMS 术语

- 在我们开始学习MySQL 数据库前，让我们先了解下RDBMS的一些术语：
  - 主键：主键是唯一的。一个数据表中只能包含一个主键。你可以使用主键来查询数据。
  - 外键：外键用于关联两个表。
  - 复合键：复合键（组合键）将多个列作为一个索引键，一般用于复合索引。
  - 索引：使用索引可快速访问数据库表中的特定信息。索引是对数据库表中一列或多列的值进行排序的一种结构。类似于书籍的目录。
  - 参照完整性：参照的完整性要求关系中不允许引用不存在的实体。与实体完整性是关系模型必须满足的完整性约束条件，目的是保证数据的一致性。

# 读写数据库数据

---

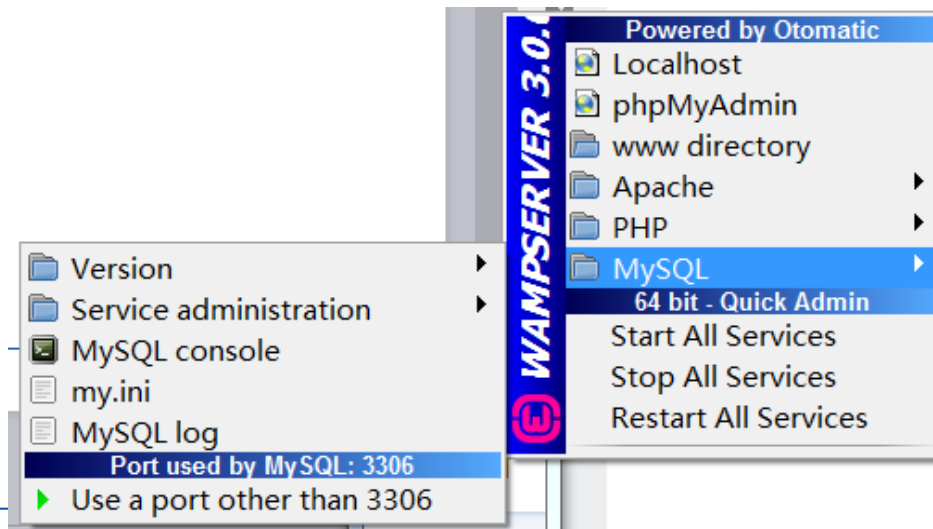
## MySQL数据库

- MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 公司。
- MySQL 是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。
  - MySQL 是开源的，所以你不需支付额外的费用。
  - MySQL 使用标准的SQL数据语言形式。
  - MySQL 可以运行于多个系统上，并且支持多种语言。如C、C++、Python、Java、Perl、PHP、Ruby等。
  - MySQL 支持大型数据库，支持5000万条记录的数据仓库，32位系统表文件最大可支持4GB，64位系统支持最大的表文件为8TB。
  - MySQL 是可以定制的，采用了GPL协议，你可以修改源码来开发自己的 MySQL 系统。

# 读写数据库数据

## MySQL数据库安装

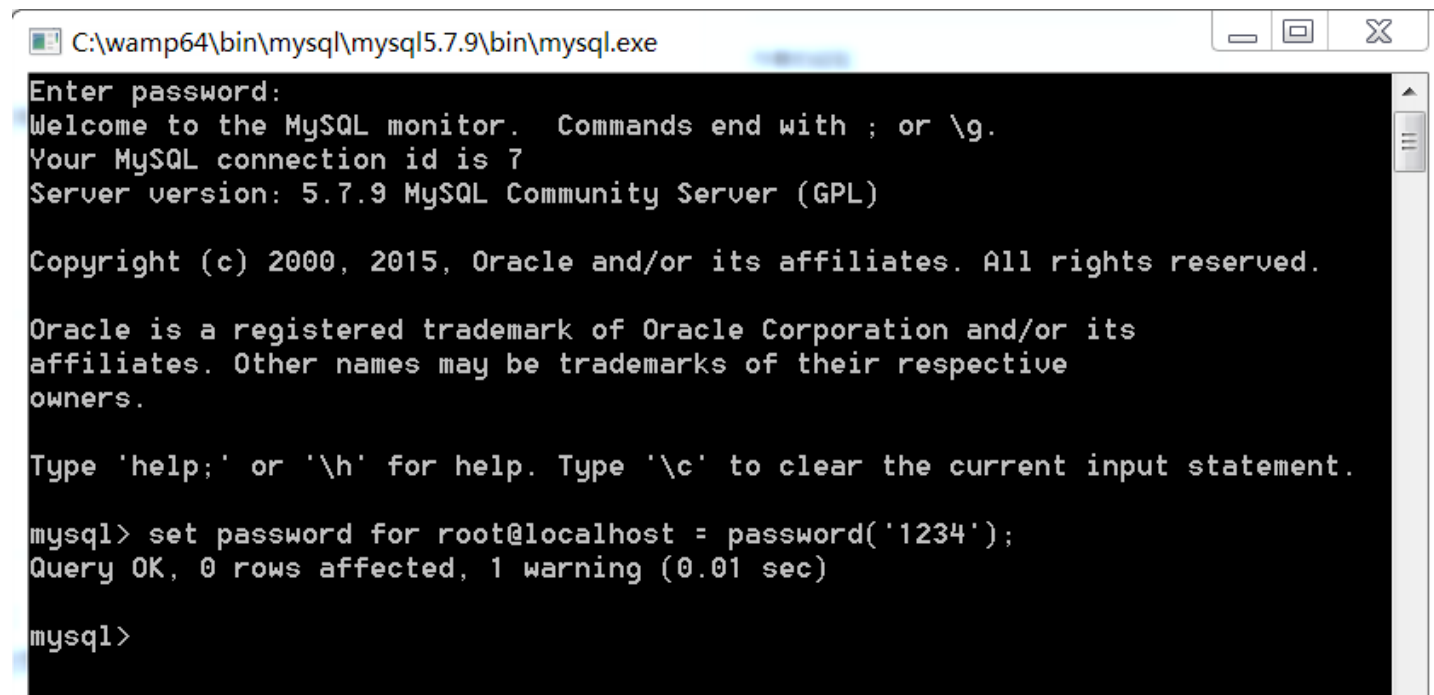
- `wampserver3_x64_apache2.4.17_mysql5.7.9_php5.6.16_php7.0.0.exe`
  - WampServer是一款由法国人开发的Apache Web服务器、PHP解释器以及MySQL数据库的整合软件包。免去了开发人员将时间花费在繁琐的配置环境过程，从而腾出更多精力去做开发。
  - 在windows下Apache+PHP+Mysql 集成环境，拥有简单的图形和菜单安装和配置环境。PHP扩展、Apache模块，开启/关闭鼠标点点就搞定，再也不用亲自去修改配置文件了，WAMP它会去做



# 读写数据库数据

## MySQL数据库修改用户密码

- 启动MySQL console: 缺省用户名root, 密码 “空”
- Windows环境下如何修改MySQL用户root密码:
  - `mysql> set password for 用户名@localhost = password('新密码');`
  - 例如: `mysql> set password for root@localhost = password('1234');`



```
C:\wamp64\bin\mysql\mysql5.7.9\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.7.9 MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> set password for root@localhost = password('1234');
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql>
```

# 读写数据库数据

---

## 常用数据库命令

- 1、显示当前数据库服务器中的数据库列表：

```
mysql> SHOW DATABASES;
```

- 2、建立数据库：

```
mysql> CREATE DATABASE 库名;
```

- 3、建立数据表：

```
mysql> USE 库名;
```

```
mysql> CREATE TABLE 表名 (字段名 VARCHAR(20), 字段名 CHAR(1));
```

- 4、删除数据库：

```
mysql> DROP DATABASE 库名;
```

- 5、删除数据表：

```
mysql> DROP TABLE 表名;
```

# 读写数据库数据

---

## 常用数据库命令

6、将表中记录清空:

```
mysql> DELETE FROM 表名;
```

7、往表中插入记录:

```
mysql> INSERT INTO 表名 VALUES ("hyq","M");
```

8、更新表中数据:

```
mysql-> UPDATE 表名 SET 字段名1='a',字段名2='b' WHERE 字段名3='c';
```

9、用文本方式将数据装入数据表中:

```
mysql> LOAD DATA LOCAL INFILE "D:/mysql.txt" INTO TABLE 表名;
```

10、导入.sql文件命令:

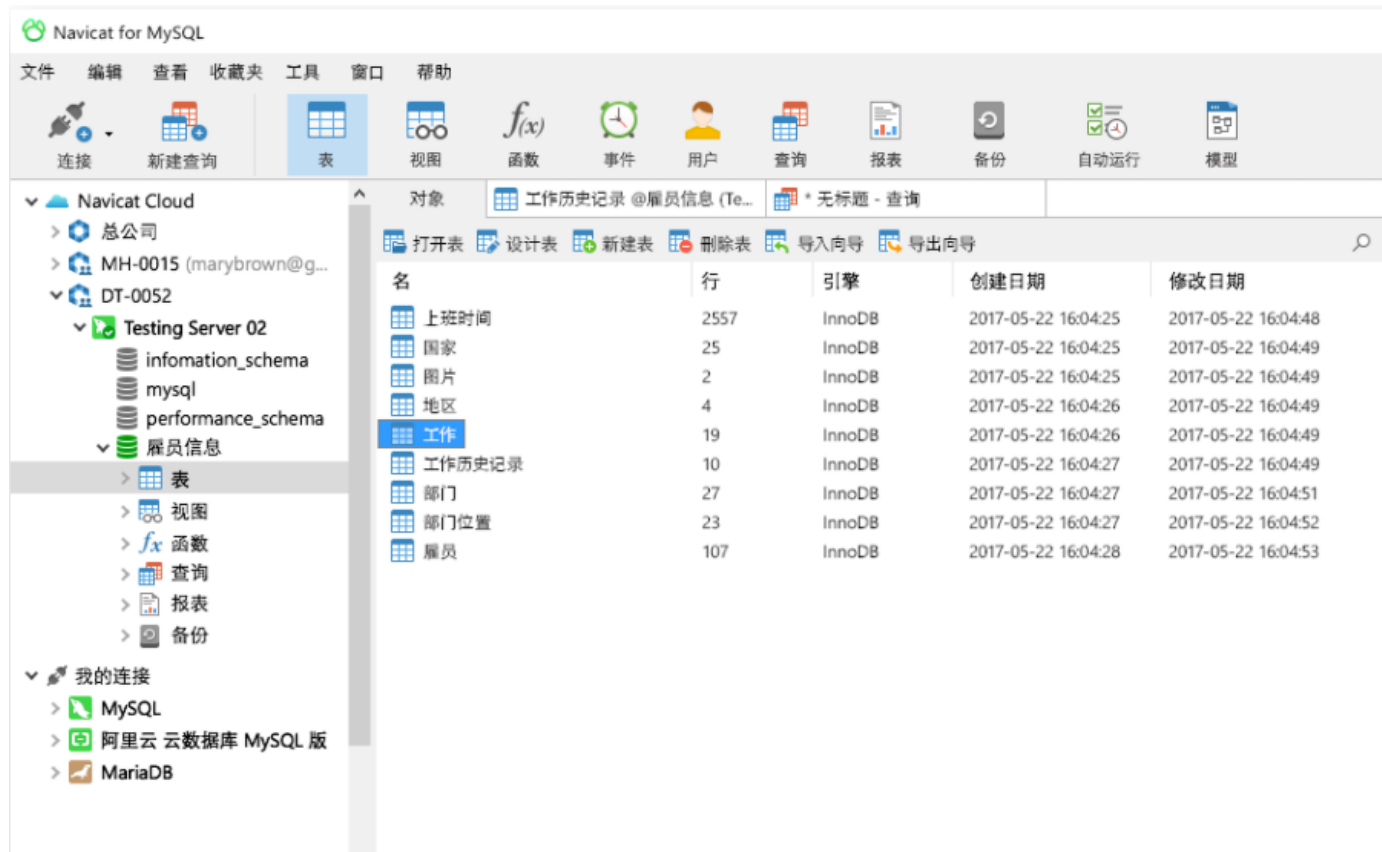
```
mysql> USE 数据库名;
```

```
mysql> SOURCE d:/mysql.sql;
```

# 读写数据库数据

## 使用Navicat for MySQL构建数据库

- Navicat for MySQL 是管理和开发 MySQL 或 MariaDB 的理想解决方案。它是一套单一的应用程序，能同时连接 MySQL 和 MariaDB 数据库，并与 Amazon RDS、Amazon Aurora、Oracle Cloud、阿里云、腾讯云和华为云等云数据库兼容。
- 这套全面的前端工具为数据库管理、开发和维护提供了一款直观而强大的图形界面。

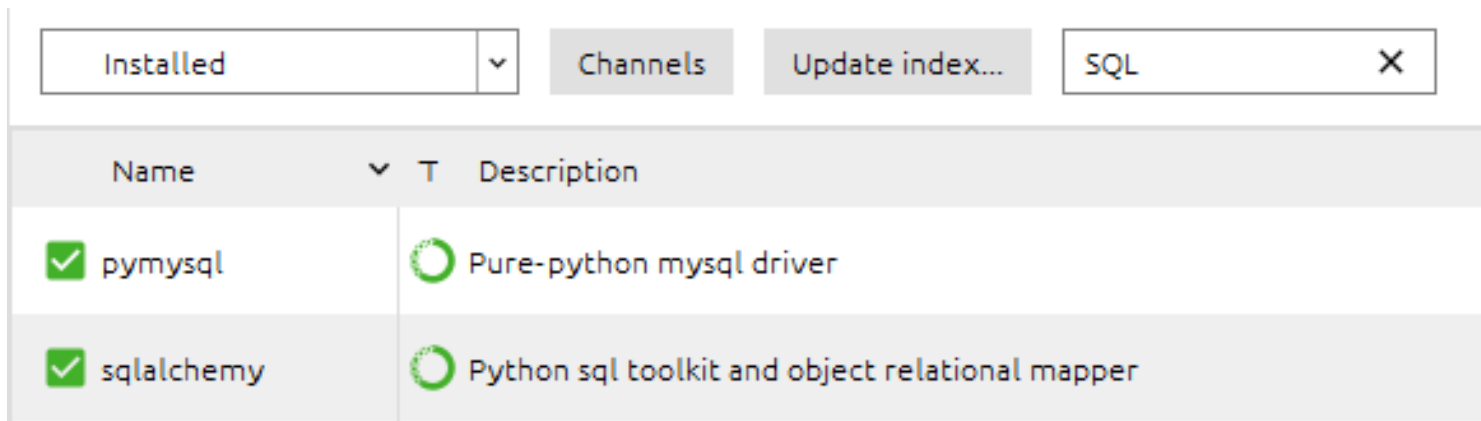




# 读写数据库数据

## 1.数据库数据读取

- pandas提供了读取与存储关系型数据库数据的函数与方法。
- 除了pandas库外，还需要使用SQLAlchemy库建立对应的数据库连接。
  - SQLAlchemy是python中最著名的ORM(Object Relationship Mapping对象关系映射器)框架
- SQLAlchemy配合相应数据库的Python连接工具（例如MySQL数据库需要安装mysqlclient或者pymysql库），使用create\_engine函数，建立一个数据库连接。
- 所以，需要在anaconda中安装这两个库：SQLAlchemy库和pymysql库



# 读写数据库数据

## 1.数据库数据读取

- 使用create\_engine函数，建立一个数据库连接
- create\_engine中填入的是一个连接字符串。
- 在使用Python的SQLAlchemy时，MySQL数据库连接字符串的格式如下：

*数据库产品名+连接工具名：//用户名:密码@数据库IP地址:数据库端口号/数据库名称? charset = 数据库数据编码*

例如：'mysql+pymysql://root:1234@127.0.0.1:3306/testdb?charset=utf8'

# 读写数据库数据

---

## 1.数据库数据读取

- `read_sql_table`只能够读取数据库的某一个表格，不能实现查询的操作。

*`pandas.read_sql_table(table_name, con, schema=None, index_col=None, coerce_float=True, columns=None)`*

- `read_sql_query`则只能实现查询操作，不能直接读取数据库中的某个表。

*`pandas.read_sql_query(sql, con, index_col=None, coerce_float=True)`*

- `read_sql`是两者的综合，既能够读取数据库中的某一个表，也能够实现查询操作。

*`pandas.read_sql(sql, con, index_col=None, coerce_float=True, columns=None)`*

# 读写数据库数据

## 1.数据库数据读取

- read\_sql\_table只能够读取数据库的某一个表格，不能实现查询的操作。

```
In [12]: detail1 = pd.read_sql_table('meal_order_detail1',con = engine)
```

```
In [13]: print('使用read_sql_table读取订单详情表的长度为:',len(detail1))
```

```
使用read_sql_table读取订单详情表的长度为： 2779
```

```
In [14]: detail1
```

```
Out[14]:
```

	detail_id	order_id	dishes_id	...	bar_code	picture_file	emp_id
0	2956	417	610062	...	NA	caipu/104001.jpg	1442
1	2958	417	609957	...	NA	caipu/202003.jpg	1442
2	2961	417	609950	...	NA	caipu/303001.jpg	1442
3	2966	417	610038	...	NA	caipu/105002.jpg	1442
4	2968	417	610003	...	NA	caipu/503002.jpg	1442
5	1899	301	610019	...	NA	caipu/204002.jpg	1095
6	1902	301	609991	...	NA	caipu/201001.jpg	1095

# 读写数据库数据

## 1.数据库数据读取

- read\_sql\_query则只能实现查询操作，不能直接读取数据库中的某个表。

```
In [15]: formlist = pd.read_sql_query('show tables', con = engine)
...: print('testdb数据库数据表清单为:', '\n', formlist)
```

testdb数据库数据表清单为:

```
      Tables_in_testdb
0  meal_order_detail1
1  meal_order_detail2
2  meal_order_detail3
```

# 读写数据库数据

## 1.数据库数据读取

- read\_sql是两者的综合，既能够读取数据库中的某一个表，也能够实现查询操作。

```
In [20]: detail3 = pd.read_sql('meal_order_detail3',con = engine)
....: print('使用read_sql函数+表格名称读取的订单详情表长度为:',
....:       len(detail3))
使用read_sql函数+表格名称读取的订单详情表长度为: 3611
```

```
In [21]: detail3
```

```
Out[21]:
```

	detail_id	order_id	dishes_id	...	bar_code	picture_file	emp_id
0	3085	431	609966	...	NA	caipu/101001.jpg	1579
1	3086	431	609942	...	NA	caipu/103003.jpg	1579
2	3088	431	609970	...	NA	caipu/101006.jpg	1579
3	3091	431	609956	...	NA	caipu/202002.jpg	1579
4	3094	431	609957	...	NA	caipu/202003.jpg	1579
5	3099	431	610003	...	NA	caipu/503002.jpg	1579
6	3102	431	610027	...	NA	caipu/304001.jpg	1579
7	3106	431	610072	...	NA	caipu/401005.jpg	1579

# 读写数据库数据

## 1.数据库数据读取

- read\_sql是两者的综合，既能够读取数据库中的某一个表，也能够实现查询操作。

```
In [18]: detail2 = pd.read_sql('select * from meal_order_detail2',
...:                           con = engine)
...: print('使用read_sql函数+sql语句读取的订单详情表长度为:', len(detail2))
```

使用read\_sql函数+sql语句读取的订单详情表长度为: 3647

```
In [19]: detail2
```

```
Out[19]:
```

	detail_id	order_id	...	picture_file	emp_id
0	2352	366	...	caipu/101002.jpg	1159
1	2354	366	...	caipu/102001.jpg	1159
2	2356	366	...	caipu/201001.jpg	1159
3	2358	366	...	/jsp/pc/images/606106.jpg	1159
4	2361	366	...	caipu/503002.jpg	1159
5	2360	366	...	caipu/305002.jpg	1159
6	2362	366	...	caipu/601005.jpg	1159
7	2901	412	...	caipu/101004.jpg	1169
8	2905	412	...	caipu/602001.jpg	1169
9	2904	412	...	caipu/102003.jpg	1169

# 读写数据库数据

## 1.数据库数据读取

- read\_sql是两者的综合，既能够读取数据库中的某一个表，也能够实现查询操作。

```
In [24]: detail2 = pd.read_sql('select * from meal_order_detail2 where order_id=366',  
....:                          con = engine)  
....: print('使用read_sql函数+sql语句读取的订单详情表长度为:',len(detail2))  
....: print(detail2)
```

使用read\_sql函数+sql语句读取的订单详情表长度为： 7

	detail_id	order_id	dishes_id	...	bar_code	picture_file	emp_id
0	2352	366	609967	...	NA	caipu/101002.jpg	1159
1	2354	366	609961	...	NA	caipu/102001.jpg	1159
2	2356	366	606000	...	NA	caipu/201001.jpg	1159
3	2358	366	606106	...	NA	/jsp/pc/images/606106.jpg	1159
4	2361	366	610003	...	NA	caipu/503002.jpg	1159
5	2360	366	610043	...	NA	caipu/305002.jpg	1159
6	2362	366	610011	...	NA	caipu/601005.jpg	1159

[7 rows x 19 columns]



# 读写数据库数据

## 1.数据库数据读取

pandas三个数据库数据读取函数的参数几乎完全一致，唯一的区别在于传入的是语句还是表名。

参数名称	说明
sql or table_name	接收string。表示读取的数据的表名或者sql语句。无默认。
con	接收数据库连接。表示数据库连接信息。无默认
index_col	接收int， sequence或者False。表示设定哪一列作为index， 如果设置多个索引（多重索引）这个参数就是一个数列。默认为None。
coerce_float	接收boolean。将数据库中的decimal类型的数据转换为pandas中的float64类型的数据。默认为True。
columns	接收list。表示读取数据的列名。默认为None。

# 读写数据库数据

## 2.数据库数据存储

数据库数据读取有三个函数，但数据存储则只有一个to\_sql方法。

*DataFrame.to\_sql(name, con, schema=None, if\_exists='fail', index=True, index\_label=None, dtype=None)*

参数名称	说明
name	接收string。代表数据库表名。无默认。
con	接收数据库连接。无默认。
if_exists	接收fail, replace, append。fail表示如果表名存在则不执行写入操作；replace表示如果存在，将原数据库表删除，再重新创建；append则表示在原数据库表的基础上追加数据。默认为fail。
index	接收boolean。表示是否将行索引作为数据传入数据库。默认True。
index_label	接收string或者sequence。代表是否引用索引名称，如果index参数为True此参数为None则使用默认名称。如果为多重索引必须使用sequence形式。默认为None。
dtype	接收dict。代表写入的数据类型（列名为key，数据格式为values）。默认为None。

# 读写数据库数据

## 2.数据库数据存储

```
detail1.to_sql('test1',con = engine,index = False,if_exists = 'replace')
```

```
formlist1 = pd.read_sql_query('show tables',con = engine)
```

```
print('新增一个表格后testdb数据库数据表清单为: ', '\n',formlist1)
```

```
In [5]: detail1.to_sql('test1',con = engine,index = False,
....:         if_exists = 'replace')
....: ## 使用read_sql读取test表
....: formlist1 = pd.read_sql_query('show tables',con = engine)
....: print('新增一个表格后testdb数据库数据表清单为: ', '\n',formlist1)
```

新增一个表格后testdb数据库数据表清单为：

```
Tables_in_testdb
0 meal_order_detail1
1 meal_order_detail2
2 meal_order_detail3
3 test1
```



# 读写文本文件

---

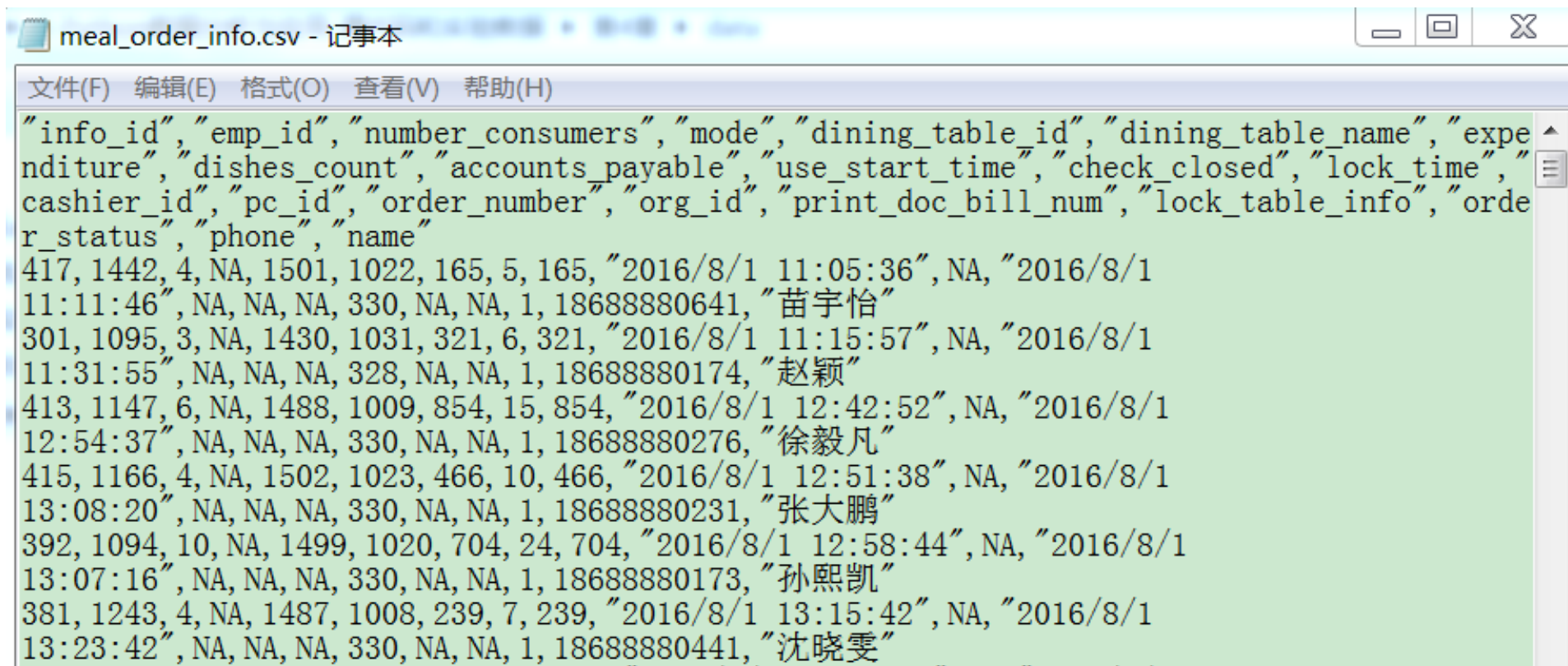
## 1.文本文件读取

- 文本文件是一种由若干行字符构成的计算机文件，它是一种典型的顺序文件。
- csv是一种逗号分隔的文件格式，因为其分隔符不一定是逗号，又被称为字符分隔文件，文件以纯文本形式存储表格数据（数字和文本）。
- csv文件是一种通用、相对简单的文件格式，最广泛的应用是在程序之间转移表格数据，而这些程序本身是在不兼容的格式上进行操作的（往往是私有的、无规范的格式）。
- 因为大量程序都支持csv或者其变体，因此可以作为大多数程序的输入和输出格式。

# 读写文本文件

## 1. 文本文件读取

- csv是一种文本文件，也是一种字符分隔文件，所以可以用读取文本文件的read\_table函数读取，也可以用读取csv字符分隔文件的read\_csv函数来读取。



```
meal_order_info.csv - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
"info_id","emp_id","number_consumers","mode","dining_table_id","dining_table_name","expe
nditure","dishes_count","accounts_payable","use_start_time","check_closed","lock_time","
cashier_id","pc_id","order_number","org_id","print_doc_bill_num","lock_table_info","orde
r_status","phone","name"
417,1442,4,NA,1501,1022,165,5,165,"2016/8/1 11:05:36",NA,"2016/8/1
11:11:46",NA,NA,NA,330,NA,NA,1,18688880641,"苗宇怡"
301,1095,3,NA,1430,1031,321,6,321,"2016/8/1 11:15:57",NA,"2016/8/1
11:31:55",NA,NA,NA,328,NA,NA,1,18688880174,"赵颖"
413,1147,6,NA,1488,1009,854,15,854,"2016/8/1 12:42:52",NA,"2016/8/1
12:54:37",NA,NA,NA,330,NA,NA,1,18688880276,"徐毅凡"
415,1166,4,NA,1502,1023,466,10,466,"2016/8/1 12:51:38",NA,"2016/8/1
13:08:20",NA,NA,NA,330,NA,NA,1,18688880231,"张大鹏"
392,1094,10,NA,1499,1020,704,24,704,"2016/8/1 12:58:44",NA,"2016/8/1
13:07:16",NA,NA,NA,330,NA,NA,1,18688880173,"孙熙凯"
381,1243,4,NA,1487,1008,239,7,239,"2016/8/1 13:15:42",NA,"2016/8/1
13:23:42",NA,NA,NA,330,NA,NA,1,18688880441,"沈晓雯"
```

# 读写文本文件

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
info_id	emp_id	number_consume	mode	dining_table_id	dining_table_name	expenditure	dishes_count	accounts_payable	use_start_time	check_clock	lock_time	cashier_id	pc_id	order_number	org_id	print_department	lock_table_id	order_status	phone	name
417	1442	4	NA	1501	1022	165	5	165	2016/8/1 11:05	NA	2016/8/1 11:11	NA	NA	NA	330	NA	NA	1	18688880641	苗宇怡
301	1095	3	NA	1430	1031	321	6	321	2016/8/1 11:15	NA	2016/8/1 11:31	NA	NA	NA	328	NA	NA	1	18688880174	赵颖
413	1147	6	NA	1488	1009	854	15	854	2016/8/1 12:42	NA	2016/8/1 12:54	NA	NA	NA	330	NA	NA	1	18688880276	徐毅凡
415	1166	4	NA	1502	1023	466	10	466	2016/8/1 12:51	NA	2016/8/1 13:08	NA	NA	NA	330	NA	NA	1	18688880231	张大鹏
392	1094	10	NA	1499	1020	704	24	704	2016/8/1 12:58	NA	2016/8/1 13:07	NA	NA	NA	330	NA	NA	1	18688880173	孙熙凯
381	1243	4	NA	1487	1008	239	7	239	2016/8/1 13:15	NA	2016/8/1 13:23	NA	NA	NA	330	NA	NA	1	18688880441	沈晓雯
429	1452	4	NA	1501	1022	699	15	699	2016/8/1 13:17	NA	2016/8/1 13:34	NA	NA	NA	330	NA	NA	1	18688880651	苗泽坤
433	1109	8	NA	1490	1011	511	14	511	2016/8/1 13:38	NA	2016/8/1 13:50	NA	NA	NA	330	NA	NA	1	18688880212	李达明
569	1143	6	NA	1488	1009	326	9	326	2016/8/1 17:06	NA	2016/8/1 17:18	NA	NA	NA	330	NA	NA	1	18688880272	陈有浩
655	1268	8	NA	1492	1013	263	10	263	2016/8/1 17:32	NA	2016/8/1 17:44	NA	NA	NA	330	NA	NA	1	18688880466	沈丹丹
577	1150	7	NA	1492	1013	380	7	380	2016/8/1 17:37	NA	2016/8/1 17:50	NA	NA	NA	330	NA	NA	1	18688880279	胡煜
622	1220	4	NA	1483	1004	164	7	164	2016/8/1 17:40	NA	2016/8/1 17:47	NA	NA	NA	330	NA	NA	1	18688880419	徐骏太
651	1593	3	NA	1485	1006	137	5	137	2016/8/1 18:12	NA	2016/8/1 18:20	NA	NA	NA	330	NA	NA	1	18688880792	高僮桐
694	1122	8	NA	1492	1013	819	10	819	2016/8/1 18:26	NA	2016/8/1 18:37	NA	NA	NA	330	NA	NA	1	18688880316	朱钰
462	1187	7	NA	1490	1011	431	13	431	2016/8/1 18:45	NA	2016/8/1 18:49	NA	NA	NA	330	NA	NA	1	18688880366	孙新潇
458	1455	2	NA	1480	1001	700	14	700	2016/8/1 19:27	NA	2016/8/1 19:31	NA	NA	NA	330	NA	NA	1	18688880654	牛长金
467	1213	10	NA	1495	1016	615	15	615	2016/8/1 19:40	NA		NA	NA	NA	330	NA	NA	0	18688880412	赵英
562	1552	8	NA	1508	1029	366	7	366	2016/8/1 19:44	NA	2016/8/1 19:57	NA	NA	NA	330	NA	NA	1	18688880751	王嘉淇
486	1156	7	NA	1492	1013	443	12	443	2016/8/1 20:31	NA	2016/8/1 20:36	NA	NA	NA	330	NA	NA	1	18688880285	张芳语
517	1008	2	NA	1480	1001	294	7	294	2016/8/1 21:11	NA	2016/8/1 21:21	NA	NA	NA	330	NA	NA	1	18688880027	许和怡
452	1114	8	NA	1492	1013	167	7	167	2016/8/1 21:19	NA	2016/8/1 21:29	NA	NA	NA	330	NA	NA	1	18688880193	邵昱笑
448	1449	6	NA	1505	1026	609	14	609	2016/8/1 21:37	NA	2016/8/1 21:52	NA	NA	NA	330	NA	NA	1	18688880648	苗秋兰
193	1084	3	NA	1401	1005	238	8	238	2016/8/2 11:20	NA	2016/8/2 11:33	NA	NA	NA	328	NA	NA	1	18688880163	张靖雯
166	986	2	NA	1402	1003	260	7	260	2016/8/2 11:22	NA	2016/8/2 11:30	NA	NA	NA	328	NA	NA	1	18688880005	莫子建
342	1450	2	NA	1402	1003	109	5	109	2016/8/2 11:58	NA	2016/8/2 12:10	NA	NA	NA	328	NA	NA	1	18688880649	苗家畅
260	990	4	NA	1421	1022	302	9	302	2016/8/2 12:35	NA	2016/8/2 12:53	NA	NA	NA	328	NA	NA	1	18688880009	张馥雨

# 读写文本文件

---

## 1. 文本文件读取

- 使用read\_table来读取文本文件。

*pandas.read\_table(filepath\_or\_buffer, sep='\t', header='infer', names=None, index\_col=None, dtype=None, engine=None, nrows=None)*

- 使用read\_csv函数来读取csv文件。

*pandas.read\_csv(filepath\_or\_buffer, sep=',', header='infer', names=None, index\_col=None, dtype=None, engine=None, nrows=None)*



# 读写文本文件

## 1.文本文件读取

read\_table和read\_csv常用参数及其说明。

参数名称	说明
filepath	接收string。代表文件路径。无默认。
sep	接收string。代表分隔符。read_csv默认为“,”，read_table默认为制表符“[Tab]”。
header	接收int或sequence。表示将某行数据作为列名。默认为infer，表示自动识别。
names	接收array。表示列名。默认为None。
index_col	接收int、sequence或False。表示索引列的位置，取值为sequence则代表多重索引。默认为None。
dtype	接收dict。代表写入的数据类型（列名为key，数据格式为values）。默认为None。
engine	接收c或者python。代表数据解析引擎。默认为c。
nrows	接收int。表示读取前n行。默认为None。



# 读写文本文件

## 1.文本文件读取

```
order = pd.read_table('data/meal_order_info.csv',sep = ',',encoding = 'gbk')
```

```
print('使用read_table读取的订单信息表的长度为: ',len(order))
```

```
In [8]: order = pd.read_table('data/meal_order_info.csv',
....:         sep = ',',encoding = 'gbk')
....: print('使用read_table读取的订单信息表的长度为: ',len(order))
```

使用read\_table读取的订单信息表的长度为: 945

```
In [9]: order
```

```
Out[9]:
```

	info_id	emp_id	number_consumers	...	order_status	phone	name
0	417	1442	4	...	1	18688880641	苗宇怡
1	301	1095	3	...	1	18688880174	赵颖
2	413	1147	6	...	1	18688880276	徐毅凡
3	415	1166	4	...	1	18688880231	张大鹏
4	392	1094	10	...	1	18688880173	孙熙凯
5	381	1243	4	...	1	18688880441	沈晓雯
6	429	1452	4	...	1	18688880651	苗泽坤

# 读写文本文件

---

## 1.文本文件读取

- `read_table`和`read_csv`函数中的`sep`参数是指定文本的分隔符的，如果分隔符指定错误，在读取数据的时候，每一行数据将连成一片。
- `header`参数是用来指定列名的，如果是`None`则会添加一个默认的列名。
- `encoding`代表文件的编码格式，常用的编码有`utf-8`、`utf-16`、`gbk`、`gb2312`、`gb18030`等。如果编码指定错误数据将无法读取，IPython解释器会报解析错误。

# 读写文本文件

## 1.文本文件读取（更改参数读取菜品订单信息表）

```
order2 = pd.read_table('data/meal_order_info.csv',sep = ';',encoding = 'gbk')
```

```
print('分隔符为;时订单信息表为：\n',order2)
```

```
In [11]: order2 = pd.read_table('data/meal_order_info.csv',
...:         sep = ';',encoding = 'gbk')
...: print('分隔符为;时订单信息表为：\n',order2)
```

分隔符为;时订单信息表为：

```
info_id,"emp_id","number_consumers","mode","dining_table_id","dining_table_name","expenditure","dishes_count","accounts_payable","use_start_time","check_closed","lock_time","cashier_id","pc_id","order_number","org_id","print_doc_bill_num","lock_table_info","order_status","phone","name"
```

```
0    417,1442,4,NA,1501,1022,165,5,165,"2016/8/1 11...
```

```
1    301,1095,3,NA,1430,1031,321,6,321,"2016/8/1 11...
```

```
2    413,1147,6,NA,1488,1009,854,15,854,"2016/8/1 1...
```

```
3    415,1166,4,NA,1502,1023,466,10,466,"2016/8/1 1...
```

```
942  692,1155,8,NA,1492,1013,735,10,735,"2016/8/31 ...
```

```
943  647,1094,4,NA,1485,1006,262,9,262,"2016/8/31 2...
```

```
944  570,1113,8,NA,1517,1038,589,13,589,"2016/8/31 ...
```

```
[945 rows x 1 columns]
```

# 读写文本文件

## 1.文本文件读取

```
order3 = pd.read_csv('data/meal_order_info.csv',sep = ',',header = None,encoding = 'gbk')
```

```
print('订单信息表为: ', '\n', order3)
```

```
In [14]: order3 = pd.read_csv('data/meal_order_info.csv',
....:         sep = ',', header = None, encoding = 'gbk')
....: print('订单信息表为: ', '\n', order3)
```

订单信息表为:

	0	1	2	...	18	19	20
	info_id	emp_id	number_consumers	...	order_status	phone	name
0	417	1442	4	...	1	18688880641	苗宇怡
1	301	1095	3	...	1	18688880174	赵颖
2	413	1147	6	...	1	18688880276	徐毅凡
3	415	1166	4	...	1	18688880231	张大鹏
4	392	1094	10	...	1	18688880173	孙熙凯
5	381	1243	4	...	1	18688880441	沈晓雯
6	692	1155	8	...	1	18688880327	习一冰
943	647	1094	4	...	1	18688880207	章春华
944	570	1113	8	...	1	18688880313	唐雅嘉

[946 rows x 21 columns]

# 读写文本文件

## 1. 文本文件读取

```
order4 = pd.read_csv('data/meal_order_info.csv', sep = ',', encoding = 'utf-8')
```

```
File "pandas\_libs\parsers.pyx", line 1240, in pandas._libs.parsers.TextReader._convert_with_dtype
```

```
File "pandas\_libs\parsers.pyx", line 1256, in pandas._libs.parsers.TextReader._string_convert
```

```
File "pandas\_libs\parsers.pyx", line 1494, in pandas._libs.parsers._string_box_utf8
```

```
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xc3 in position 0: invalid continuation byte
```

- 以上列举了read\_table和read\_csv函数的主要参数，能够满足多数情况读取文本文件或者csv文件的需求。其实，这两个函数还有50多个参数，如果感兴趣，可以阅读pandas官方的API文档。

# 读写文本文件

## 2.文本文件储存

文本文件的存储和读取类似，结构化数据可以通过pandas中的to\_csv函数实现以csv文件格式存储文件。

```
DataFrame.to_csv(path_or_buf=None, sep=',', na_rep='', columns=None, header=True, index=True, index_label=None, mode='w', encoding=None)
```

参数名称	说明	参数名称	说明
path_or_buf	接收string。代表文件路径。无默认。	index	接收boolean，代表是否将行名（索引）写出。默认为True。
sep	接收string。代表分隔符。默认为“,”。	index_labels	接收sequence。表示索引名。默认为None。
na_rep	接收string。代表缺失值。默认为“”。	mode	接收特定string。代表数据写入模式。默认为w。
columns	接收list。代表写出的列名。默认为None。	encoding	接收特定string。代表存储文件的编码格式。默认为None。
header	接收boolean，代表是否将列名写出。默认为True。		

# 读写文本文件

## 2.文本文件储存

```
import os
```

```
print('订单信息表写入文本文件前目录内文件列表为：\n', os.listdir('tmp'))
```

```
order.to_csv('../tmp/orderInfo.csv',sep = ';',index = False)      ## 将order以csv格式存储
```

```
print('订单信息表写入文本文件后目录内文件列表为：\n', os.listdir('tmp'))
```

```
In [21]: import os
....: print('订单信息表写入文本文件前目录内文件列表为：\n',
....:       os.listdir('tmp'))
....: ## 将order以csv格式存储
....: order.to_csv('tmp/orderInfo.csv',sep = ';',index = False)
....: print('订单信息表写入文本文件后目录内文件列表为：\n',
....:       os.listdir('tmp'))
```

订单信息表写入文本文件前目录内文件列表为：

```
[]
```

订单信息表写入文本文件后目录内文件列表为：

```
['orderInfo.csv']
```

# 读写Excel文件

---

## 1.Excel文件读取

- Excel是微软公司的办公软件Microsof Office 的组件之一，它可以对数据进行处理、统计分析等操作，广泛地应用于管理、财经和金融等众多领域。其文件保存依照程序版本的不同分为两种。

(1) Microsoft Oflice Excel 2007之前的版本(不包括2007 )默认保存的文件扩展名为.xls。

(2 ) Microsoft Office Excel 2007之后的版本默认保存的文件扩展名为.xlsx。

- pandas提供了read\_excel函数来读取 “xls” “xlsx” 两种Excel文件。

*pandas.read\_excel(io, sheetname=0, header=0, index\_col=None, names=None, dtype=None)*



# 读写Excel文件

## 1.Excel文件读取

```
pandas.read_excel(io, sheetname=0, header=0, index_col=None, names=None, dtype=None)
```

参数名称	说明
io	接收string。表示文件路径。无默认。
sheetname	接收string、int。代表excel表内数据的分表位置。默认为0。
header	接收int或sequence。表示将某行数据作为列名。默认为infer，表示自动识别。
names	接收int、sequence或者False。表示索引列的位置，取值为sequence则代表多重索引。默认为None。
index_col	接收int、sequence或者False。表示索引列的位置，取值为sequence则代表多重索引。默认为None。
dtype	接收dict。代表写入的数据类型（列名为key，数据格式为values）。默认为None。

# 读写Excel文件

## 1.Excel文件读取

```
user = pd.read_excel('data/users.xlsx')          ## 读取user.xlsx文件
```

```
print('客户信息表长度为: ',len(user))
```

```
In [22]: user = pd.read_excel('data/users.xlsx')## 读取user.xlsx文件
...: print('客户信息表长度为: ',len(user))
客户信息表长度为: 734
```

```
In [23]: user
```

```
Out[23]:
```

	USER_ID	MYID	ACCOUNT	NAME	...	sex	poo	address	age
0	1	admin	超级管理员	admin	...	男	广东广州	泰迪科技	23.0
1	981	NaN	老师	teacher	...	NaN	NaN	NaN	NaN
2	982	NaN	叶亦凯	sx	...	男	广东广州	广州	21.0
3	983	NaN	邓彬彬	lyy	...	女	广东广州	广州	21.0
4	984	NaN	张建涛	zad	...	男	广东广州	广州	22.0
5	985	NaN	莫诗怡	mf	...	女	广西	广州	23.0
6	986	NaN	莫子建	mjc	...	男	广西	广州	22.0
7	987	NaN	易子歆	yll	...	女	广西	广州	21.0

# 读写Excel文件

---

## 2.Excel文件储存

- 将文件存储为Excel文件，可以使用to\_excel方法。其语法格式如下。

*DataFrame.to\_excel(excel\_writer=None, sheetname=None, na\_rep='', header=True, index=True, index\_label=None, mode='w', encoding=None)*

- 与to\_csv方法的常用参数基本一致，区别之处在于指定存储文件的文件路径参数名称为excel\_writer，并且没有sep参数，增加了一个sheetnames参数用来指定存储的Excel sheet的名称，默认为sheet1。

# 读写Excel文件

## 2.Excel文件储存

```
print('客户信息表写入excel文件前目录内文件列表为：\n', os.listdir('tmp'))
```

```
user.to_excel('tmp/userInfo.xlsx')
```

```
print('客户信息表写入excel文件后目录内文件列表为：\n', os.listdir('tmp'))
```

```
In [25]: print('客户信息表写入excel文件前目录内文件列表为：\n',  
....:         os.listdir('tmp'))  
....: user.to_excel('tmp/userInfo.xlsx')  
....: print('客户信息表写入excel文件后目录内文件列表为：\n',  
....:         os.listdir('tmp'))
```

客户信息表写入excel文件前目录内文件列表为：

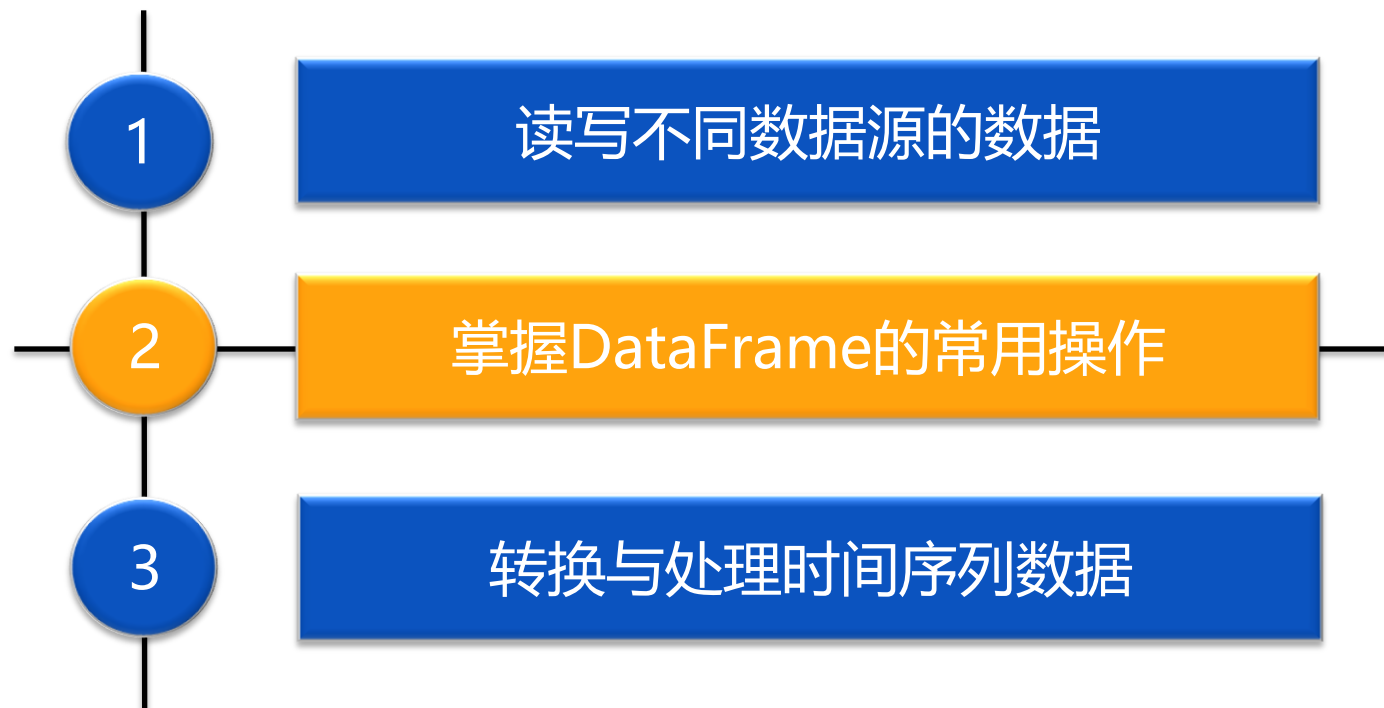
```
['orderInfo.csv']
```

客户信息表写入excel文件后目录内文件列表为：

```
['orderInfo.csv', 'userInfo.xlsx']
```

# 目录

---



# 掌握DataFrame的常用操作

---

## 任务描述

- DataFrame是最常用的pandas对象，类似于Microsoft Office Excel表格。完成数据读取后，数据就以DataFrame数据结构存储在内存中。
- 但此时并不能直接开始统计分析工作，需要使用DataFrame的属性与方法对数据的分布、大小等基本的数据状况有一个了解。
- 只有对数据基本状况有了一个深度的了解，才能够依据数据的状况，进行量身定制的统计分析。

# 掌握DataFrame的常用操作

---

## 任务分析

- (1) 查看数据的大小、维度等基本信息。
- (2) 查看销售菜品数据的基本统计信息。
- (3) 剔除全为空值或者所有元素取值相同的列。

# 查看DataFrame的常用属性

## 基础属性

函数	返回值
values	元素
index	索引
columns	列名
dtypes	类型
size	元素个数
ndim	维度数
shape	数据形状（行列数目）



# 查看DataFrame的常用属性

---

```
from sqlalchemy import create_engine

import pandas as pd

## 创建数据库连接

engine = create_engine('mysql+pymysql://root:1234@127.0.0.1:3306/testdb?charset=utf8')

detail= pd.read_sql_table('meal_order_detail1',con = engine)

print('订单详情表的索引为： ', detail.index)

print('订单详情表的所有值为： ', '\n', detail.values)

print('订单详情表的列名为： ', '\n', detail.columns)

print('订单详情表的数据类型为： ', '\n', detail.dtypes)
```

## 查看DataFrame的常用属性

`print('订单详情表的索引为: ', detail.index)`

```
In [36]: print('订单详情表的索引为: ', detail.index)
订单详情表的索引为:  RangeIndex(start=0, stop=2779, step=1)
```

`print('订单详情表的所有值为: ', '\n', detail.values)`

```
In [37]: print('订单详情表的所有值为: ', '\n', detail.values)
订单详情表的所有值为:
[['2956' '417' '610062' ... 'NA' 'caipu/104001.jpg' '1442']
 ['2958' '417' '609957' ... 'NA' 'caipu/202003.jpg' '1442']
 ['2961' '417' '609950' ... 'NA' 'caipu/303001.jpg' '1442']
 ...
 ['6756' '774' '609949' ... 'NA' 'caipu/404005.jpg' '1138']
 ['6763' '774' '610014' ... 'NA' 'caipu/302003.jpg' '1138']
 ['6764' '774' '610017' ... 'NA' 'caipu/302006.jpg' '1138']]
```

## 查看DataFrame的常用属性

```
print('订单详情表的列名为: ', '\n', detail.columns)
```

```
In [38]: print('订单详情表的列名为: ', '\n', detail.columns)
```

订单详情表的列名为:

```
Index(['detail_id', 'order_id', 'dishes_id', 'logicprn_name',  
      'parent_class_name', 'dishes_name', 'itemis_add', 'counts', 'amounts',  
      'cost', 'place_order_time', 'discount_amt', 'discount_reason',  
      'kick_back', 'add_inprice', 'add_info', 'bar_code', 'picture_file',  
      'emp_id'],  
      dtype='object')
```

## 查看DataFrame的常用属性

`print('订单详情表的数据类型为: ', '\n', detail.dtypes)`

```
In [39]: print('订单详情表的数据类型为: ', '\n', detail.dtypes)
```

订单详情表的数据类型为:

detail_id	object
order_id	object
dishes_id	object
logicprn_name	object
parent_class_name	object
dishes_name	object
itemis_add	object
counts	float64
amounts	float64
cost	object
place_order_time	datetime64[ns]
discount_amt	object
discount_reason	object
kick_back	object
add_inprice	object
add_info	object
bar_code	object
picture_file	object
emp_id	object
dtype:	object

## 查看DataFrame的常用属性

---

```
print('订单详情表的元素个数为：', detail.size)
```

```
print('订单详情表的维度数为：', detail.ndim) ## 查看DataFrame的维度数
```

```
print('订单详情表的形状为：', detail.shape) ## 查看DataFrame的形状
```

```
In [40]: print('订单详情表的元素个数为：', detail.size)
...: print('订单详情表的维度数为：', detail.ndim) ## 查看DataFrame的维度数
...: print('订单详情表的形状为：', detail.shape) ## 查看DataFrame的形状
```

```
订单详情表的元素个数为： 52801
```

```
订单详情表的维度数为： 2
```

```
订单详情表的形状为： (2779, 19)
```

# 查看DataFrame的常用属性

```
print('订单详情表转置前形状为: ',detail.shape)
```

```
print('订单详情表转置后形状为为: ',detail.T.shape)
```

```
In [41]: print('订单详情表转置前形状为: ',detail.shape)
...: print('订单详情表转置后形状为为: ',detail.T.shape)
订单详情表转置前形状为: (2779, 19)
订单详情表转置后形状为为: (19, 2779)
```

```
In [42]: detail.T
```

```
Out[42]:
```

	0	...	2778
detail_id	2956	...	6764
order_id	417	...	774
dishes_id	610062	...	610017
logicprn_name	NA	...	NA
parent_class_name	NA	...	NA
dishes_name	蒜蓉生蚝	...	酸辣藕丁
itemis_add	0	...	0
counts	1	...	1
amounts	49	...	33
cost	NA	...	NA
place_order_time	2016-08-01 11:05:00	...	2016-08-10 22:04:00

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——数据基本查看方式

- **对单列数据的访问**：DataFrame的单列数据为一个**Series**。根据DataFrame的定义可以知晓DataFrame是一个带有标签的二维数组，每个标签相当每一列的列名。有以下两种方式来实现对单列数据的访问。
  - 以字典访问某一个key的值的方式使用对应的列名，实现单列数据的访问。
  - 以属性的方式访问，实现单列数据的访问。（不建议使用，易引起混淆）

```
order_id = detail['order_id']
```

```
print('订单详情表中的order_id的形状为:', '\n', order_id.shape)
```

```
In [4]: order_id = detail['order_id']
...: print('订单详情表中的order_id的形状为:', '\n', order_id.shape)
订单详情表中的order_id的形状为:
(2779,)
```

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——数据基本查看方式

- 对单列数据的访问：以属性的方式访问，实现单列数据的访问。（不建议使用，易引起混淆）

```
dishes_name = detail.dishes_name
```

```
print('订单详情表中的dishes_name的形状为:',dishes_name.shape)
```

```
In [6]: dishes_name = detail.dishes_name
....: print('订单详情表中的dishes_name的形状为:',dishes_name.shape)
订单详情表中的dishes_name的形状为: (2779,)
```

```
In [7]: dishes_name
```

```
Out[7]:
```

0	蒜蓉生蚝
1	蒙古烤羊腿
2	大蒜苋菜
3	芝麻烤紫菜
4	蒜香包
5	白斩鸡



# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——数据基本查看方式

- **对某一列的某几行访问**：访问DataFrame中某一列的某几行时，单独一列的DataFrame可以视为一个 **Series**（另一种pandas提供的类，可以看作是只有一列的DataFrame），而访问一个Series基本和访问一个一维的ndarray相同。

```
dishes_name5 = detail['dishes_name'][:5]
```

```
print('订单详情表中的dishes_name前5个元素为：', '\n', dishes_name5)
```

```
In [8]: dishes_name5 = detail['dishes_name'][:5]
...: print('订单详情表中的dishes_name前5个元素为：', '\n', dishes_name5)
```

订单详情表中的dishes\_name前5个元素为：

```
0    蒜蓉生蚝
1    蒙古烤羊腿
2    大蒜苋菜
3    芝麻烤紫菜
4    蒜香包
```

```
Name: dishes_name, dtype: object
```

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——数据基本查看方式

- **对多列数据访问：**访问DataFrame多列数据可以将多个列索引名称视为一个列表，同时访问DataFrame多列数据中的多行数据和访问单列数据的多行数据方法基本相同。

- **访问DataFrame多列的多行数据**

```
orderDish = detail[['order_id','dishes_name']][:5]
```

```
print('订单详情表中的order_id和dishes_name前5个元素为：', '\n', orderDish)
```

```
In [9]: orderDish = detail[['order_id','dishes_name']][:5]
....: print('订单详情表中的order_id和dishes_name前5个元素为：',
....:       '\n', orderDish)
```

订单详情表中的order\_id和dishes\_name前5个元素为：

	order_id	dishes_name
0	417	蒜蓉生蚝
1	417	蒙古烤羊腿
2	417	大蒜苋菜
3	417	芝麻烤紫菜
4	417	蒜香包

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——数据基本查看方式

### ➤ 对某几行访问:

- 如果只是需要访问DataFrame某几行数据的实现方式则和上述的访问多列多行相似，选择所有列，使用 “:” 代替即可。

```
order5 = detail[:][1:6]
```

```
print('订单详情表的1-6行元素为: ', '\n', order5)
```

```
In [10]: order5 = detail[:][1:6]
...: print('订单详情表的1-6行元素为: ', '\n', order5)
```

订单详情表的1-6行元素为:

	detail_id	order_id	dishes_id	...	bar_code	picture_file	emp_id
1	2958	417	609957	...	NA	caipu/202003.jpg	1442
2	2961	417	609950	...	NA	caipu/303001.jpg	1442
3	2966	417	610038	...	NA	caipu/105002.jpg	1442
4	2968	417	610003	...	NA	caipu/503002.jpg	1442
5	1899	301	610019	...	NA	caipu/204002.jpg	1095

```
[5 rows x 19 columns]
```

# 查改增删DataFrame数据

---

## 1.查看访问DataFrame中的数据——数据基本查看方式

### ➤ 对某几行访问:

- head和tail也可以得到多行数据，但是用这两种方法得到的数据都是从开始或者末尾获取的连续数据。默认参数为访问5行，只要在方法后方的 “()” 中填入访问行数即可实现目标行数的查看。

```
print('订单详情表中前五行为数据为','\n',detail.head())
```

```
print('订单详情表中后五个元素为：','\n',detail.tail())
```

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——数据基本查看方式

```
In [11]: print('订单详情表中前五行为数据为','\n',detail.head())
...: print('订单详情表中后五个元素为: ','\n',detail.tail())
```

订单详情表中前五行为数据为

	detail_id	order_id	dishes_id	...	bar_code	picture_file	emp_id
0	2956	417	610062	...	NA	caipu/104001.jpg	1442
1	2958	417	609957	...	NA	caipu/202003.jpg	1442
2	2961	417	609950	...	NA	caipu/303001.jpg	1442
3	2966	417	610038	...	NA	caipu/105002.jpg	1442
4	2968	417	610003	...	NA	caipu/503002.jpg	1442

[5 rows x 19 columns]

订单详情表中后五个元素为:

	detail_id	order_id	dishes_id	...	bar_code	picture_file	emp_id
2774	6750	774	610011	...	NA	caipu/601005.jpg	1138
2775	6742	774	609996	...	NA	caipu/201006.jpg	1138
2776	6756	774	609949	...	NA	caipu/404005.jpg	1138
2777	6763	774	610014	...	NA	caipu/302003.jpg	1138
2778	6764	774	610017	...	NA	caipu/302006.jpg	1138

[5 rows x 19 columns]

是获取的连续数据  
的查看。

# 查改增删DataFrame数据

---

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- loc方法是针对DataFrame索引名称的切片方法，如果传入的不是索引名称，那么切片操作将无法执行。利用loc方法，能够实现所有单层索引切片操作。loc方法使用方法如下。

*DataFrame.loc[行索引/名称或条件, 列索引/名称]*

- iloc和loc区别是iloc接收的必须是行索引和列索引的位置。iloc方法的使用方法如下。

*DataFrame.iloc[行索引/位置, 列索引/位置]*

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- 使用loc和iloc实现单列切片

*DataFrame.loc[行索引/名称或条件, 列索引/名称]*

*DataFrame.iloc[行索引/位置, 列索引/位置]*

```
In [12]: dishes_name1 = detail.loc[:, 'dishes_name']  
....: print('使用loc提取dishes_name列的size为: ', dishes_name1.size)  
使用loc提取dishes_name列的size为: 2779
```

```
In [13]: dishes_name2 = detail.iloc[:, 3]  
....: print('使用iloc提取第3列的size为: ', dishes_name2.size)  
使用iloc提取第3列的size为: 2779
```

# 查改增删DataFrame数据

---

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- 使用loc方法和iloc实现多列切片，其原理的通俗解释就是将多列的列名或者位置作为一个列表或者数据传入。
- 使用loc, iloc方法可以取出DataFrame中的任意数据。
- 在loc使用的时候内部传入的行索引名称如果为一个区间，则前后均为闭区间；iloc方法使用时内部传入的行索引位置或列索引位置为区间时，则为前闭后开区间。
- loc内部还可以传入表达式，结果会返回满足表达式的所有值。



# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- 使用loc方法和iloc实现多列切片

```
In [15]: orderDish1 = detail.loc[:,['order_id','dishes_name']]
...: print('使用loc提取order_id和dishes_name列的size为: ', orderDish1.size)
使用loc提取order_id和dishes_name列的size为: 5558
```

```
In [16]: orderDish2 = detail.iloc[:,[1,3]]
...: print('使用iloc提取第1和第3列的size为: ', orderDish1.size)
使用iloc提取第1和第3列的size为: 5558
```

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- 使用loc方法和iloc实现花式切片

```
In [46]: print('列名为order_id和dishes_name的行名为3的数据为: \n',  
....:         detail.loc[3,['order_id','dishes_name']])  
列名为order_id和dishes_name的行名为3的数据为:  
order_id      417  
dishes_name    芝麻烤紫菜  
Name: 3, dtype: object
```

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- 使用loc方法和iloc实现花式切片

```
In [67]: print('列名为order_id和dishes_name行名为2,3,4,5,6的数据为: \n',  
....:         detail.loc[2:6,['order_id','dishes_name']])
```

列名为order\_id和dishes\_name行名为2,3,4,5,6的数据为:

	order_id	dishes_name
2	417	大蒜苋菜
3	417	芝麻烤紫菜
4	417	蒜香包
5	301	白斩鸡
6	301	香烤牛排

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- 使用loc方法和iloc实现花式切片

```
In [68]: print('列位置为1和3行位置为3的数据为: \n',detail.iloc[3,[1,3]])  
列位置为1和3行位置为3的数据为:  
  order_id      417  
logicprn_name    NA  
Name: 3, dtype: object
```

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- 使用loc方法和iloc实现花式切片

```
In [69]: print('列位置为1和3行位置为2,3,4,5,6的数据为: \n',  
          ....: detail.iloc[2:7,[1,3]])  
列位置为1和3行位置为2,3,4,5,6的数据为:
```

	order_id	logicprn_name
2	417	NA
3	417	NA
4	417	NA
5	301	NA
6	301	NA

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

### ➤ 使用loc函数实现条件切片

```
In [73]: print('detail中order_id为458的dishes_name为: \n',  
....:         detail.loc[detail['order_id']=='458',['order_id','dishes_name']])  
detail中order_id为458的dishes_name为:
```

	order_id	dishes_name
145	458	蒜香辣花甲
146	458	剁椒鱼头
147	458	凉拌蒜蓉西兰花
148	458	木须豌豆
149	458	辣炒鱿鱼
150	458	酸辣藕丁
151	458	炆炒大白菜
152	458	香菇鸡肉粥
153	458	干锅田鸡
154	458	桂圆枸杞鸽子汤

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- 使用iloc函数实现条件切片

```
In [74]: print('detail中order_id为458的第1,5列数据为: \n',  
...:         detail.iloc[detail['order_id']=='458',[1,5]])  
Traceback (most recent call last):
```

```
File "C:\Users\Jerry\Anaconda3\lib\site-packages\pandas\core\indexing.py", line 1947, in  
    raise NotImplementedError("iLocation based boolean "
```

```
NotImplementedError: iLocation based boolean indexing on an integer type is not available
```

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- 若使用`detail.iloc[detail['order_id']=='458',[1,5]]`读取数据, 则会报错,
- 原因在于此处条件返回的为一个布尔值Series, 而iloc可以接收的数据类型并不包括Series。根据Series的构成只要取出该Series的values就可以了。需改为`detail.iloc[(detail[ 'order_id' ]== '458' ).values,[1,5]]`。

```
In [75]: print('detail中order_id为458的第1,5列数据为: \n',
      ....:      detail.iloc[(detail['order_id']=='458').values,[1,5]])
detail中order_id为458的第1,5列数据为:
   order_id  dishes_name
145      458      蒜香辣花甲
146      458      剁椒鱼头
147      458  凉拌蒜蓉西兰花
148      458      木须豌豆
149      458      辣炒鱿鱼
150      458      酸辣藕丁
151      458      炆炒大白菜
```



# 查改增删DataFrame数据

---

## 1.查看访问DataFrame中的数据——loc, iloc访问方式

- loc更加灵活多变，代码的可读性更高，
- iloc的代码简洁，但可读性不高。
- 具体在数据分析工作中使用哪一种方法，根据情况而定，
- 大多数时候建议使用loc方法。

# 查改增删DataFrame数据

---

## 1.查看访问DataFrame中的数据——切片方法之ix

- ix方法更像是loc和iloc两种切片方法的融合。ix方法在使用时既可以接收索引名称也可以接收索引位置。其使用方法如下。

*DataFrame.ix[行索引的名称或位置或者条件, 列索引名称或位置]*

- 使用ix方法时有个注意事项，当索引名称和位置存在部分重叠时，ix默认优先识别名称。

# 查改增删DataFrame数据

## 1.查看访问DataFrame中的数据——切片方法之ix

```
print('列名为dishes_name, 行名为2,3,4,5,6的数据为: \n', detail.loc[2:6,'dishes_name'])
```

```
print('列位置为5, 行位置为2至6的数据为: \n',detail.iloc[2:6,5])
```

```
print('列位置为5, 行名为2至6的数据为: ', '\n',detail.ix[2:6,5])
```

列位置为5行名为2至6的数据为:

```
2    大蒜苋菜
3    芝麻烤紫菜
4    蒜香包
5    白斩鸡
6    香烤牛排
```

```
Name: dishes_name, dtype: object
__main__:4: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing
```

# 查改增删DataFrame数据

---

## 1.查看访问DataFrame中的数据——切片方法之ix

控制ix方法需要注意以下几点。

- 使用ix参数时，尽量保持行索引名称和行索引位置重叠，使用时就无须考虑取值时区间的问题。一律为闭区间。
- 使用列索引名称，而非列索引位置。主要用来保证代码可读性。
- 使用列索引位置时，需要注解。同样保证代码可读性。
- 除此之外ix方法还有一个缺点，就是在面对数据量巨大的任务的时候，其效率会低于loc和iloc方法，所以在日常的数据分析工作中建议使用loc和iloc方法来执行切片操作。

# 查改增删DataFrame数据

---

## 2.更新修改DataFrame中的数据

- 更改DataFrame中的数据，原理是将这部分数据提取出来，重新赋值为新的数据。

**##将order\_id为458的，变换为45800**

```
detail.loc[detail['order_id']=='458','order_id'] = '45800'
```

- 需要注意的是，数据更改直接针对DataFrame原数据更改，操作无法撤销，如果做出更改，需要对更改条件做确认或对数据进行备份。

# 查改增删DataFrame数据

## 2.更新修改DataFrame中的数据

```
In [83]: detail.loc[detail['order_id']=='45800','order_id']
```

```
Out[83]:
```

```
145    45800
146    45800
147    45800
148    45800
149    45800
150    45800
151    45800
152    45800
153    45800
154    45800
155    45800
156    45800
157    45800
158    45800
```

```
Name: order_id, dtype: object
```

```
In [84]: detail.loc[detail['order_id']=='458','order_id']
```

```
Out[84]: Series([], Name: order_id, dtype: object)
```

# 查改增删DataFrame数据

## 3.为DataFrame增添数据

- DataFrame添加一列的方法非常简单，只需要新建一个列索引。并对该索引下的数据进行赋值操作即可。

```
detail['payment'] = detail['counts']*detail['amounts']
```

```
print('detail新增列payment的前五行为: ', '\n', detail['payment'].head())
```

```
In [85]: detail['payment'] = detail['counts']*detail['amounts']
...: print('detail新增列payment的前五行为: ', '\n', detail['payment'].head())
detail新增列payment的前五行为:
 0    49.0
1    48.0
2    30.0
3    25.0
4    13.0
Name: payment, dtype: float64
```

# 查改增删DataFrame数据

## 3.为DataFrame增添数据

- 新增的一列值是相同的则直接赋值一个常量即可。

```
detail['pay_way'] = '现金支付'
```

```
print('detail新增列pay_way的前五行为: ','\n',detail['pay_way'].head())
```

```
In [94]: detail['pay_way'] = '现金支付'
....: print('detail新增列pay_way的前五行为: ','\n',detail['pay_way'].head())
detail新增列pay_way的前五行为:
0    现金支付
1    现金支付
2    现金支付
3    现金支付
4    现金支付
Name: pay_way, dtype: object
```



# 查改增删DataFrame数据

## 4.删除某列或某行数据

删除某列或某行数据需要用到pandas提供的方法drop，drop方法的使用如下。

- axis为0时表示删除行，axis为1时表示删除列。

*drop(labels, axis=0, level=None, inplace=False, errors='raise')*

- 常用参数如下所示。

参数名称	说明
labels	接收string或array。代表删除的行或列的标签。无默认。
axis	接收0或1。代表操作的轴向。默认为0。
levels	接收int或者索引名。代表标签所在级别。默认为None。
inplace	接收boolean。代表操作是否对原数据生效。默认为False。

# 查改增删DataFrame数据

---

## 4.删除某列或某行数据

```
print('删除pay_way前deatil的列索引为: ', '\n', detail.columns)
```

```
detail.drop(labels = 'pay_way', axis = 1, inplace = True)
```

```
print('删除pay_way后detail的列索引为: ', '\n', detail.columns)
```

## 查改增删DataFrame数据

### 4.删除某列或某行数据

```
In [95]: print('删除pay_way前deatil的列索引为: ', '\n', detail.columns)
....: detail.drop(labels = 'pay_way', axis = 1, inplace = True)
....: print('删除pay_way后detail的列索引为: ', '\n', detail.columns)
```

删除pay\_way前deatil的列索引为:

```
Index(['detail_id', 'order_id', 'dishes_id', 'logicprn_name',
       'parent_class_name', 'dishes_name', 'itemis_add', 'counts', 'amounts',
       'cost', 'place_order_time', 'discount_amt', 'discount_reason',
       'kick_back', 'add_inprice', 'add_info', 'bar_code', 'picture_file',
       'emp_id', 'payment', 'pay_way'],
      dtype='object')
```

删除pay\_way后detail的列索引为:

```
Index(['detail_id', 'order_id', 'dishes_id', 'logicprn_name',
       'parent_class_name', 'dishes_name', 'itemis_add', 'counts', 'amounts',
       'cost', 'place_order_time', 'discount_amt', 'discount_reason',
       'kick_back', 'add_inprice', 'add_info', 'bar_code', 'picture_file',
       'emp_id', 'payment'],
      dtype='object')
```

# 查改增删DataFrame数据

## 4.删除某列或某行数据

- 要删除某行的数据，只需要将drop方法参数中的 'label' 参数换成相对应的行索引，将 'axis' 参数设置成 '0' 即可。

```
print('删除1-10行前detail的长度为: ',len(detail))
```

```
detail.drop(labels = range(1,11),axis = 0,inplace = True)
```

```
print('删除1-10行后detail的列索引为: ',len(detail))
```

```
In [96]: print('删除1-10行前detail的长度为: ',len(detail))
...: detail.drop(labels = range(1,11),axis = 0,inplace = True)
...: print('删除1-10行后detail的列索引为: ',len(detail))
```

```
删除1-10行前detail的长度为: 2779
```

```
删除1-10行后detail的列索引为: 2769
```

# 描述分析DataFrame数据

## 1.数值型特征的描述性统计——NumPy中的描述性统计函数

- 数值型数据的描述性统计主要包括了计算数值型数据的完整情况、最小值、均值、中位数、最大值、四分位数、极差、标准差、方差、协方差和变异系数等。
- 在NumPy库中已经提到了很多统计函数，一些常用的统计学函数如下表所示。

函数名称	说明	函数名称	说明
np.min	最小值	np.max	最大值
np.mean	均值	np.ptp	极差
np.median	中位数	np.std	标准差
np.var	方差	np.cov	协方差

- **pandas库基于NumPy，自然也可以用这些函数对DataFrame进行描述性统计。**

# 描述分析DataFrame数据

## 1.数值型特征的描述性统计——NumPy中的描述性统计函数

```
import numpy as np
```

```
print('订单详情表中amount（价格）的平均值为: ', np.mean(detail['amounts']))
```

```
In [97]: import numpy as np
...: print('订单详情表中amount（价格）的平均值为: ', np.mean(detail['amounts']))
订单详情表中amount（价格）的平均值为: 45.343084145901045
```

# 描述分析DataFrame数据

---

## 1.数值型特征的描述性统计——pandas描述性统计方法

```
print('订单详情表中amount（价格）的平均值为：', detail['amounts'].mean())
```

```
In [100]: print('订单详情表中amount（价格）的平均值为：', detail['amounts'].mean())  
订单详情表中amount（价格）的平均值为： 45.343084145901045
```

# 描述分析DataFrame数据

## 1.数值型特征的描述性统计—— pandas描述性统计方法

- pandas还提供了一个方法叫作describe，能够一次性得出DataFrame所有数值型特征的非空值数目、均值、四分位数、标准差。

print('订单详情表counts和amounts两列的描述性统计为：\n ', detail[['counts','amounts']].describe())

```
In [101]: print('订单详情表counts和amounts两列的描述性统计为：\n',  
....:         detail[['counts','amounts']].describe())
```

订单详情表counts和amounts两列的描述性统计为：

	counts	amounts
count	2769.0	2769.000000
mean	2.0	45.343084
std	0.0	36.841316
min	2.0	1.000000
25%	2.0	25.000000
50%	2.0	35.000000
75%	2.0	56.000000
max	2.0	178.000000



# 描述分析DataFrame数据

## 1.数值型特征的描述性统计—— pandas描述性统计方法

➤ pandas还提供了与统计相关的主要方法，如下表。

方法名称	说明	方法名称	说明
min	最小值	max	最大值
mean	均值	ptp	极差
median	中位数	std	标准差
var	方差	cov	协方差
sem	标准误差	mode	众数
skew	样本偏度	kurt	样本峰度
quantile	四分位数	count	非空值数目
describe	描述统计	mad	平均绝对离差

# 描述分析DataFrame数据

## 2.类别型特征的描述性统计

- 描述类别型特征的分布状况，可以使用频数统计表。pandas库中实现频数统计的方法为value\_counts。

`print('订单详情表dishes_name频数统计结果前10为: \n ',detail['dishes_name'].value_counts()[0:10])`

```
In [102]: print('订单详情表dishes_name频数统计结果前10为: \n',
....:         detail['dishes_name'].value_counts()[0:10])
订单详情表dishes_name频数统计结果前10为:
  白饭/大碗          91
  凉拌菠菜          77
  谷稻小庄          72
  麻辣小龙虾        65
  白饭/小碗          60
  五色糯米饭(七色)   58
  焖猪手            55
  芝士烩波士顿龙虾   55
  辣炒鱿鱼          53
  水煮鱼            47
Name: dishes_name, dtype: int64
```

# 描述分析DataFrame数据

---

## 2.类别型特征的描述性统计

- pandas提供了categories类，可以使用astype方法将目标特征的数据类型转换为category类别。
- describe方法除了支持传统数值型以外，还能够支持对category类型的数据进行描述性统计，四个统计量分别为**列非空元素的数目，类别的数目，数目最多的类别，数目最多类别的数目**。
- 例如，订单表中的菜品不是数值型数据，那按照之前介绍的describe函数，就不能进行“非空值数目、均值、四分位数、标准差。”的统计，但是如果将菜品数据类型转换成category类型，就可以使用describe进行“非空元素的数目，类别的数目，数目最多的类别，数目最多类别的数目”的统计。

# 描述分析DataFrame数据

## 2.类别型特征的描述性统计

- (1) 将object数据强制转换为category类型

```
detail['dishes_name'] = detail['dishes_name'].astype('category')
```

```
print('订单信息表dishes_name列转变数据类型后为: ',detail['dishes_name'].dtypes)
```

```
In [103]: detail['dishes_name'] = detail['dishes_name'].astype('category')
...: print('订单信息表dishes_name列转变数据类型后为: ',detail['dishes_name'].dtypes)
订单信息表dishes_name列转变数据类型后为:  category
```

# 描述分析DataFrame数据

## 2.类别型特征的描述性统计

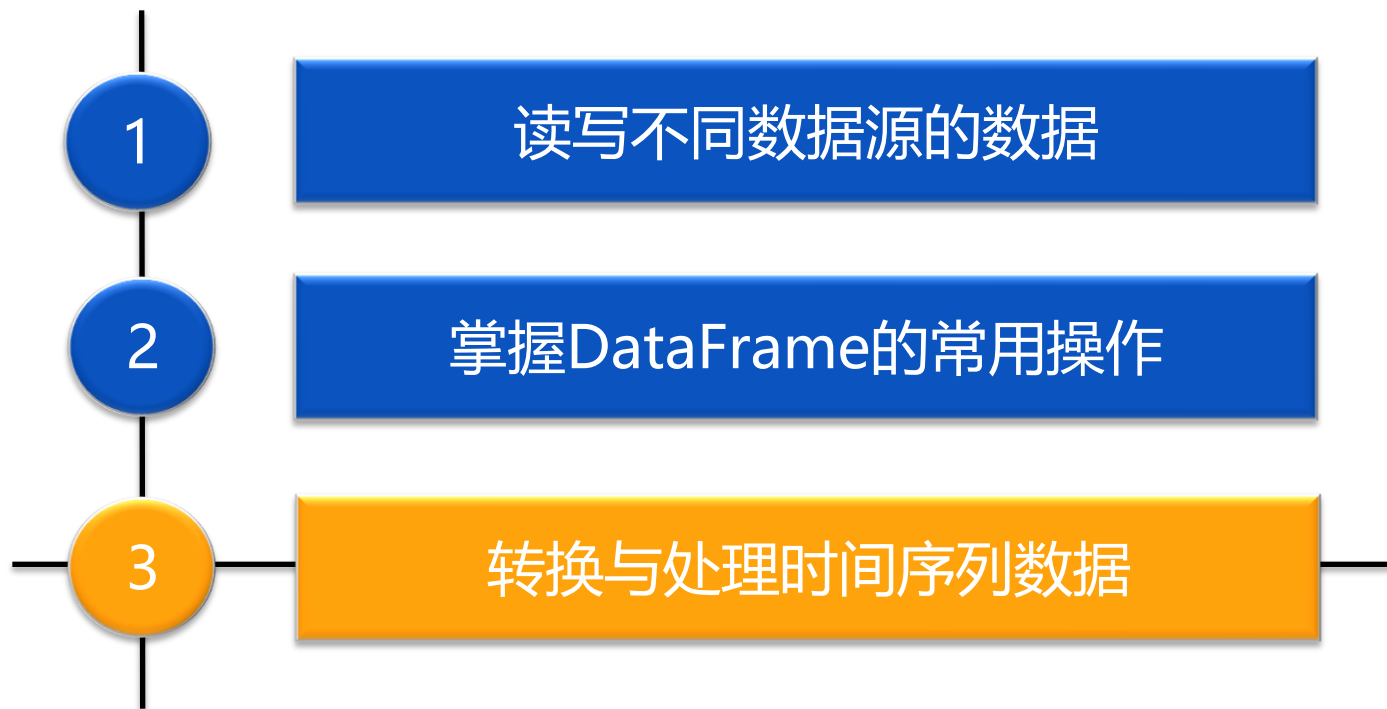
- (2) 使用describe对category类型进行特征的描述性统计

`print('订单信息表dishes_name的描述统计结果为：\n ',detail['dishes_name'].describe())`

```
In [104]: print('订单信息表dishes_name的描述统计结果为：\n',
....:         detail['dishes_name'].describe())
订单信息表dishes_name的描述统计结果为：
count      2769
unique      145
top         白饭/大碗
freq         91
Name: dishes_name, dtype: object
```

# 目录

---



# 转换与处理时间序列数据

---

## 概述

- 数据分析的分析对象不仅局限于数值型和类别型两种，常用的数据类型还包括了时间类型。
- 通过时间类型数据能够获取到对应的年月日和星期等信息。
- 但时间类型数据在读入Python后常常以字符串形式出现，无法实现大部分与时间相关的分析。
- pandas 库继承了NumPy库的datetime64以及timedelta64模块，能够快速实现时间字符串的转换、信息提取和时间运算。

# 转换与处理时间序列数据

---

## 概述

- 本小节主要完成以下任务：
  - 将订单信息表数据中的时间转换为标准的时间格式；
  - 提取订单信息表数据中的年月日和星期信息；
  - 查看餐饮的数据时间分布



# 转换字符串时间为标准时间

## pandas时间相关的类

- 在多数情况下，对时间类型数据进行分析的前提就是将原本为字符串的时间转换为标准时间类型。  
pandas继承了NumPy库和datetime库的时间相关模块，提供了6种时间相关的类。

类名称	说明
Timestamp	最基础的时间类。表示某个时间点。在绝大多数的场景中的时间数据都是Timestamp形式的时间。
Period	表示单个时间跨度，或者某个时间段，例如某一天，某一小时等。
Timedelta	表示不同单位的时间，例如1天，1.5小时，3分钟，4秒等，而非具体的某个时间段。
DatetimeIndex	一组Timestamp构成的Index，可以用来作为Series或者DataFrame的索引。
PeriodtimeIndex	一组Period构成的Index，可以用来作为Series或者DataFrame的索引。
TimedeltaIndex	一组Timedelta构成的Index，可以用来作为Series或者DataFrame的索引。

# 转换字符串时间为标准时间

## Timestamp类型

- 其中Timestamp作为时间类中最基础的，也是最为常用的。在多数情况下，时间相关的字符串都会转换成为Timestamp。pandas提供了to\_datetime函数，能够实现这一目标。

```
import pandas as pd
```

```
order = pd.read_table('data/meal_order_info.csv',sep = ',',encoding = 'gbk')
```

```
print('进行转换前订单信息表lock_time的类型为: ',order['lock_time'].dtypes)
```

```
order['lock_time'] = pd.to_datetime(order['lock_time'])
```

```
print('进行转换后订单信息表lock_time的类型为: ',order['lock_time'].dtypes)
```

```
进行转换前订单信息表lock_time的类型为:  object
```

```
进行转换后订单信息表lock_time的类型为:  datetime64[ns]
```

# 转换字符串时间为标准时间

## Timestamp类型

- 值得注意的是，Timestamp类型时间是有限制的。

```
In [107]: print('最小时间为: ', pd.Timestamp.min)
         ....: print('最大时间为: ', pd.Timestamp.max)
最小时间为:  1677-09-21 00:12:43.145225
最大时间为:  2262-04-11 23:47:16.854775807
```

# 转换字符串时间为标准时间

## DatetimeIndex与PeriodIndex函数

- 除了将数据字原始DataFrame中直接转换为Timestamp格式外，还可以将数据单独提取出来将其转换为DatetimeIndex或者PeriodIndex。
- 转换为PeriodIndex的时候需要注意，需要通过freq参数指定时间间隔，常用的时间间隔有Y为年，M为月，D为日，H为小时，T为分钟，S为秒。

```
dateIndex = pd.DatetimeIndex(order['lock_time'])
```

```
print('转换为DatetimeIndex后数据的类型为: \n',type(dateIndex))
```

```
periodIndex = pd.PeriodIndex(order['lock_time'],freq = 'S')
```

```
print('转换为DatetimeIndex后数据的类型为: \n',type(periodIndex))
```

# 转换字符串时间为标准时间

## DatetimeIndex与PeriodIndex函数

- 两个函数可以用来转换数据还可以用来创建时间序列数据，其参数非常类似。
- 调用pd.date\_range()创建DatetimeIndex序列：

```
index=pd.date_range("2018-09-02","2018-09-30",freq="2H")
```

```
loc=np.random.choice(np.arange(len(index)),size=4,replace=False) #随机选取4个互不相同的数
```

```
loc.sort()
```

```
ts_index=index[loc]
```

```
ts_index
```

运行结果：

```
DatetimeIndex(['2018-09-09 20:00:00', '2018-09-13 22:00:00',  
               '2018-09-16 20:00:00', '2018-09-23 22:00:00'],  
              dtype='datetime64[ns]', freq=None)
```

# 转换字符串时间为标准时间

## DatetimeIndex与PeriodIndex函数及其参数说明

- DatetimeIndex和PeriodIndex两者区别在日常使用的过程中相对较小，其中DatetimeIndex是用来指代一系列时间点的一种数据结构，而PeriodIndex则是用来指代一系列时间段的数据结构。

参数名称	说明
data	接收array。表示DatetimeIndex的值。无默认。
freq	接收string。表示时间的间隔频率。无默认。
start	接收string。表示生成规则时间数据的起始点。无默认。
periods	表示需要生成的周期数目。无默认。
end	接收string。表示生成规则时间数据的终结点。无默认。
tz	接收timezone。表示数据的时区。默认为None。
name	接收int, string。默认为空。指定DatetimeIndex的名字。

# 提取时间序列数据信息

## Timestamp类常用属性

- 在多数涉及时间相关的数据处理，统计分析的过程中，需要提取时间中的年份，月份等数据。使用对应的Timestamp类属性就能够实现这一目的。
- 结合Python列表推导式，可以实现对DataFrame某一系列时间信息数据的提取。

属性名称	说明	属性名称	说明
year	年	week	一年中第几周
month	月	quarter	季节
day	日	weekofyear	一年中第几周
hour	小时	dayofyear	一年中的第几天
minute	分钟	dayofweek	一周第几天
second	秒	weekday	一周第几天
date	日期	weekday_name	星期名称
time	时间	is_leap_year	是否闰年

# 提取时间序列数据信息

## 提取datetime数据中的时间序列数据

```
year1 = [i.year for i in order['lock_time']]
print('lock_time中的年份数据前5个为: ',year1[:5])
month1 = [i.month for i in order['lock_time']]
print('lock_time中的月份数据前5个为: ',month1[:5])
day1 = [i.day for i in order['lock_time']]
print('lock_time中的日期数据前5个为: ',day1[:5])
weekday1 = [i.weekday_name for i in order['lock_time']]
print('lock_time中的星期名称数据前5个为: ',weekday1[:5])
```

```
lock_time中的年份数据前5个为: [2016, 2016, 2016, 2016, 2016]
lock_time中的月份数据前5个为: [8, 8, 8, 8, 8]
lock_time中的日期数据前5个为: [1, 1, 1, 1, 1]
lock_time中的星期名称数据前5个为: ['Monday', 'Monday', 'Monday', 'Monday', 'Monday']
```



# 提取时间序列数据信息

## 在DatetimeIndex和PeriodIndex中提取信息

- 在DatetimeIndex和PeriodIndex中提取对应信息的方法更加简单，可以以类属性方式实现。
- 值得注意的是PeriodIndex相比于DatetimeIndex少了weekday\_name属性，所以不能够用该属性提取星期名称数据。若想要提取信息名称可以通过提取weekday属性，而后将0-6四个标签分别赋值为Monday至Sunday。

```
In [7]: print('dateIndex中的星期名称数据前5个为: \n',dateIndex.weekday_name[:5])
...: print('periodIndex中的星期标号数据前5个为: ',periodIndex.weekday[:5])
dateIndex中的星期名称数据前5个为:
Index(['Monday', 'Monday', 'Monday', 'Monday', 'Monday'], dtype='object', name='lock_time')
periodIndex中的星期标号数据前5个为: Int64Index([0, 0, 0, 0, 0], dtype='int64', name='lock_time')
```

# 加减时间数据

## Timedelta类

- Timedelta是时间相关的类中的一个异类，不仅能够使用正数，还能够使用负数表示单位时间，例如1秒，2分钟，3小时等。使用Timedelta类，配合常规的时间相关类能够轻松实现时间的算术运算。目前Timedelta函数中时间周期中没有年和月。所有周期名称，对应单位及其说明如下表所示。

周期名称	单位	说明	周期名称	单位	说明
weeks	无	星期	seconds	s	秒
days	D	天	milliseconds	ms	毫秒
hours	h	小时	microseconds	us	微妙
minutes	m	分	nanoseconds	ns	纳秒

# 加减时间数据

## Timedelta类

- 使用Timedelta，可以很轻松地实现在某个时间上加减一段时间。

```
In [9]: time1 = order['lock_time']+pd.Timedelta(days = 1)
...: print('lock_time在加上一天前前5行数据为: \n',order['lock_time'][:5])
...: print('lock_time在加上一天后前5行数据为: \n',time1[:5])
lock_time在加上一天前前5行数据为:
0    2016-08-01 11:11:46
1    2016-08-01 11:31:55
2    2016-08-01 12:54:37
3    2016-08-01 13:08:20
4    2016-08-01 13:07:16
Name: lock_time, dtype: datetime64[ns]
lock_time在加上一天后前5行数据为:
0    2016-08-02 11:11:46
1    2016-08-02 11:31:55
2    2016-08-02 12:54:37
3    2016-08-02 13:08:20
4    2016-08-02 13:07:16
Name: lock_time, dtype: datetime64[ns]
```

# 加减时间数据

## Timedelta类

- 除了使用Timedelta实现时间的平移外，还能够直接对两个时间序列进行相减，从而得出一个Timedelta。

```
In [11]: timeDelta = pd.to_datetime('2018-09-26')-order['lock_time']
....: print('lock_time距今天的时间间隔数据: \n',timeDelta[:5])
....: print('时间间隔数据的数据类型为: ',timeDelta.dtypes)
lock_time距今天的时间间隔数据:
0    785 days 12:48:14
1    785 days 12:28:05
2    785 days 11:05:23
3    785 days 10:51:40
4    785 days 10:52:44
Name: lock_time, dtype: timedelta64[ns]
时间间隔数据的数据类型为:  timedelta64[ns]
```



大数据，成就未来



# Thank you!