目录：

▼ 01、复杂查询

- 1、逻辑运算
- 2、逻辑筛选数据
- 3、函数筛选
- 4、比较函数
- 5、查询df.query()
- 6、筛选df.filter()
- 7、按数据类型查询

▼ 02、数据类型转换

- 1、推断类型
- 2、指定类型
- 3、类型转换astype()
- 4、转为时间类型

▼ 03、数据排序

- 1、索引排序df.sort_index()
- 2、数值排序sort_values()
- 3、混合排序
- 4、按值大小排序nsmallest()和nlargest()

▼ 04、添加修改

- 1、修改数值
- 2、替换数据
- 3、填充空值

- 4、修改索引名
- 5、增加列
- 6、插入列df.insert()
- 7、指定列df.assign()
- 8、执行表达式df.eval()
- 9、增加行
- 10、追加合并
- 11、删除
- 12、删除空值

▼ 05、高级过滤

- 1、df.where()
- 2、np.where()
- 3、df.mask()
- 4、df.lookup()

▼ 06、数据迭代

- 1、迭代Series
- 2、df.iterrows()
- 3、df.itertuples()
- 4、df.items()
- 5、按列迭代

▼ 07、函数应用

- 1、pipe()
- 2、apply()
- 3、applymap()

- 4、map()
- 5、agg()
- 6、transform()
- 7、copy()

01、复杂查询

1、逻辑运算

```
# Q1成绩大于36
df.Q1> 36
# Q1成绩不小于60分，并且是C组成员
~(df.Q1< 60) & (df['team'] == 'C')
```

2、逻辑筛选数据

切片（[ ]）、.loc[ ]和.iloc[ ]均支持上文所介绍的逻辑表达式。

以下是切片（[ ]）的逻辑筛选示例：

```
# 表达式与切片一致
df.loc[df['Q1']> 90, 'Q1':] # Q1大于90
df.loc[(df.Q1> 80) & (df.Q2 < 15)] # a
df.loc[(df.Q1> 90) | (df.Q2 < 90)] # o
df.loc[df['Q1']== 8] # 等于8
df.loc[df.Q1== 8] # 等于8
df.loc[df['Q1']> 90, 'Q1':] # Q1大于90
```

以下是.loc[ ]和.lic[ ]示例：

```
# 表达式与切片一致
df.loc[df['Q1']> 90, 'Q1':] # Q1大于90
df.loc[(df.Q1> 80) & (df.Q2 < 15)] # a
df.loc[(df.Q1> 90) | (df.Q2 < 90)] # o
df.loc[df['Q1']== 8] # 等于8
df.loc[df.Q1== 8] # 等于8
df.loc[df['Q1']> 90, 'Q1':] # Q1大于90
```

3、函数筛选

```
# 查询最大索引的值
df.Q1[lambdas: max(s.index)] # 值为21
# 计算最大值
max(df.Q1.index)
# 99
df.Q1[df.index==99]
```

4、比较函数

```
# 以下相当于 df[df.Q1 == 60]
df[df.Q1.eq(60)]
df.ne() # 不等于 !=
df.le() # 小于等于 <=
df.lt() # 小于 <
df.ge() # 大于等于 >=
df.gt() # 大于 >
```

5、查询df.query()

`df.query('Q1 > Q2 > 90') # 直接写类型SQL where语句`

还支持使用@符引入变量

```
# 支持传入变量，如大于平均分40分的
a = df.Q1.mean()
df.query('Q1 > @a+40')
df.query('Q1 > `Q2`+@a')
```

df.eval()与df.query()类似，也可以用于表达式筛选。

```
# df.eval()用法与df.query类似
df[df.eval("Q1 > 90 > Q3 >10")]
df[df.eval("Q1 > `Q2`+@a")]
```

6、筛选df.filter()

```
df.filter(items=['Q1', 'Q2']) # 选择两
df.filter(regex='Q', axis=1) # 列名包含
df.filter(regex='e$', axis=1) # 以e结尾
df.filter(regex='1$', axis=0) # 正则，
df.filter(like='2', axis=0) # 索引中有2
# 索引中以2开头、列名有Q的
df.filter(regex='^2',axis=0).filter(li
```

7、按数据类型查询

```
df.select_dtypes(include=['float64'])
df.select_dtypes(include='bool')
df.select_dtypes(include=['number']) #
df.select_dtypes(exclude=['int']) # 排
df.select_dtypes(exclude=['datetime64'
```

02、数据类型转换

在开始数据分析前，我们需要为数据分配好合适的类型，这样才能够高效地处理数据。不同的数据类型适用于不同的处理方法。

```
# 对所有字段指定统一类型
df = pd.DataFrame(data, dtype='float32
# 对每个字段分别指定
df = pd.read_excel(data, dtype={'team'
```

1、推断类型

```
# 自动转换合适的数据类型
df.infer_objects() # 推断后的DataFrame
df.infer_objects().dtypes
```

2、指定类型

```
# 按大体类型推定
m = ['1', 2, 3]
s = pd.to_numeric(s) # 转成数字
pd.to_datetime(m) # 转成时间
pd.to_timedelta(m) # 转成时间差
pd.to_datetime(m, errors='coerce') # 转
pd.to_numeric(m, errors='ignore')
pd.to_numeric(m errors='coerce').filln
pd.to_datetime(df[['year', 'month', 'd
# 组合成日期
```

3、类型转换astype()

```
df.Q1.astype('int32').dtypes
# dtype('int32')
df.astype({'Q1': 'int32','Q2':'int32'}
```

4、转为时间类型

```
t = pd.Series(['20200801', '20200802']
```

03、数据排序

数据排序是指按一定的顺序将数据重新排列，帮助使用者发现数据的变化趋势，同时

提供一定的业务线索，还具有对数据纠错、分类等作用。

1、索引排序df.sort_index()

```
s.sort_index() # 升序排列
df.sort_index() # df也是按索引进行排序
df.team.sort_index()s.sort_index(ascen
s.sort_index(inplace=True) # 排序后生效
# 索引重新0-(n-1)排，很有用，可以得到它的
s.sort_index(ignore_index=True)
s.sort_index(na_position='first') # 空
s.sort_index(level=1) # 如果多层，排一级
s.sort_index(level=1, sort_remaining=F
# 行索引排序，表头排序
df.sort_index(axis=1) # 会把列按列名顺序
```

2、数值排序sort_values()

```
df.Q1.sort_values()
df.sort_values('Q4')
df.sort_values(by=['team', 'name'],asc
```

其他方法：

```
s.sort_values(ascending=False) # 降序
s.sort_values(inplace=True) # 修改生效
s.sort_values(na_position='first') # 空
# df按指定字段排列
df.sort_values(by=['team'])
```

```
df.sort_values('Q1')
# 按多个字段，先排team，在同team内再看Q1
df.sort_values(by=['team', 'Q1'])
# 全降序

df.sort_values(by=['team', 'Q1'], asce
# 对应指定team升Q1降
df.sort_values(by=['team', 'Q1'],ascen
# 索引重新0-(n-1)排
df.sort_values('team', ignore_index=Tr
```

3、混合排序

```
df.set_index('name', inplace=True) # 设
df.index.names = ['s_name'] # 给索引起名
df.sort_values(by=['s_name', 'team'])
```

4、按值大小排序nsmallest()和nlargest()

```
s.nsmallest(3) # 最小的3个
s.nlargest(3) # 最大的3个
# 指定列
df.nlargest(3, 'Q1')
df.nlargest(5, ['Q1', 'Q2'])
df.nsmallest(5, ['Q1', 'Q2'])
```

04、添加修改

数据的修改、增加和删除在数据整理过程中时常发生。修改的情况一般是修改错误、格式转换，数据的类型修改等。

1、修改数值

```
df.iloc[0,0] # 查询值
# 'Liver'
df.iloc[0,0] = 'Lily' # 修改值
df.iloc[0,0] # 查看结果
# 'Lily'

# 将小于60分的成绩修改为60
df[df.Q1 < 60] = 60
# 查看
df.Q1

# 生成一个长度为100的列表
v = [1, 3, 5, 7, 9] * 20
```

2、替换数据

```
s.replace(0, 5) # 将列数据中的0换为5
df.replace(0, 5) # 将数据中的所有0换为5
df.replace([0, 1, 2, 3], 4) # 将0～3全
df.replace([0, 1, 2, 3], [4, 3, 2, 1])
s.replace([1, 2], method='bfill') # 向
df.replace({0: 10, 1: 100}) # 字典对应
df.replace({'Q1': 0, 'Q2': 5}, 100) #
df.replace({'Q1': {0: 100, 4: 400}}) #
```

3、填充空值

```
df.fillna(0) # 将空值全修改为0
# {'backfill', 'bfill', 'pad', 'ffill'
df.fillna(method='ffill') # 将空值都修改
values = {'A': 0, 'B': 1, 'C': 2, 'D':
df.fillna(value=values) # 为各列填充不同
df.fillna(value=values, limit=1) # 只替
```

4、修改索引名

```
df.rename(columns={'team':'class'})
```

常用方法如下：

```
df.rename(columns={"Q1":"a", "Q2": "b"
df.rename(index={0: "x", 1:"y", 2: "z"
df.rename(index=str) # 对类型进行修改
df.rename(str.lower, axis='columns') #
df.rename({1: 2, 2: 4}, axis='index')

# 对索引名进行修改
s.rename_axis("animal")
df.rename_axis("animal") # 默认是列索引
df.rename_axis("limbs",axis="columns")

# 索引为多层索引时可以将type修改为class
df.rename_axis(index={'type': 'class'}

# 可以用set_axis进行设置修改
s.set_axis(['a', 'b', 'c'], axis=0)
df.set_axis(['I', 'II'], axis='columns
```

```
df.set_axis(['i', 'ii'], axis='columns
```

5、增加列

```
df['foo'] = 100 # 增加一列foo，所有值都是
df['foo'] = df.Q1 + df.Q2 # 新列为两列相
df['foo'] = df['Q1'] + df['Q2'] # 同上
# 把所有为数字的值加起来
df['total'] =df.select_dtypes(include=
df.loc[:,'Q1':'Q4'].apply(lambda x: su
df.loc[:, 'Q10'] = '我是新来的' # 也可以
# 增加一列并赋值，不满足条件的为NaN
df.loc[df.num >= 60, '成绩'] = '合格'
df.loc[df.num < 60, '成绩'] = '不合格'
```

6、插入列df.insert()

```
# 在第三列的位置上插入新列total列，值为每
df.insert(2, 'total', df.sum(1))
```

7、指定列df.assign()

```
# 增加total列
df.assign(total=df.sum(1))
# 增加两列
df.assign(total=df.sum(1), Q=100)
df.assign(total=df.sum(1)).assign(Q=10
```

其他使用示例：

```
df.assign(Q5=[100]*100) # 新增加一列Q5
```

```
df.assign(Q5=[100]*100) # 新增加一列Q5
df = df.assign(Q5=[100]*100) # 赋值生效
df.assign(Q6=df.Q2/df.Q1) # 计算并增加Q
df.assign(Q7=lambda d: d.Q1 * 9 / 5 +
df.assign(tag=df.Q1>df.Q2)
# 比较计算，True为1，False为0
df.assign(tag=(df.Q1>df.Q2).astype(int
# 映射文案
df.assign(tag=(df.Q1>60).map({True:'及
# 增加多个
df.assign(Q8=lambda d: d.Q1*5,
          Q9=lambda d: d.Q8+1) # Q8没有
```

8、执行表达式df.eval()

```
# 传入求总分表达式
df.eval('total = Q1+Q3+Q3+Q4')
```

其他方法：

```
df['C1'] = df.eval('Q2 + Q3')
df.eval('C2 = Q2 + Q3') # 计算
a = df.Q1.mean()df.eval("C3 =`Q3`+@a")
df.eval("C3 = Q2 > (`Q3`+@a)") #加一个
df.eval('C4 = name + team', inplace=Tr
```

9、增加行

```
# 新增索引为100的数据
df.loc[100] = ['tom', 'A', 88, 88, 88,
```

其他方法：

```
df.loc[101]={'Q1':88,'Q2':99} # 指定列
df.loc[df.shape[0]+1] = {'Q1':88,'Q2':
df.loc[len(df)+1] = {'Q1':88,'Q2':99}
# 批量操作，可以使用迭代
rows = [[1,2],[3,4],[5,6]]
for row in rows:
    df.loc[len(df)] = row
```

10、追加合并

```
df = pd.DataFrame([[1, 2], [3, 4]],col
df2 = pd.DataFrame([[5, 6], [7, 8]],co
df.append(df2)
```

11、删除

```
# 删除索引为3的数据
s.pop(3)
# 93s
s
```

12、删除空值

```
df.dropna() # 一行中有一个缺失值就删除
df.dropna(axis='columns') # 只保留全有值
df.dropna(how='all') # 行或列全没值才删除
df.dropna(thresh=2) # 至少有两个空值时才
df.dropna(inplace=True) # 删除并使替换生
```

05、高级过滤

介绍几个非常好用的复杂数据处理的数据过滤输出方法。

1、df.where()

```
# 数值大于70
df.where(df > 70)
```

2、np.where()

```
# 小于60分为不及格
np.where(df>=60, '合格', '不合格')
```

3、df.mask()

```
# 符合条件的为NaN
df.mask(s > 80)
```

4、df.lookup()

```
# 行列相同数量，返回一个array
df.lookup([1,3,4], ['Q1','Q2','Q3']) #
df.lookup([1], ['Q1']) # array([36])
```

06、数据迭代

1、迭代Series

```
# 迭代指定的列
for i in df.name:
      print(i)
# 迭代索引和指定的两列
for i,n,q in zip(df.index, df.name,df.
    print(i, n, q)
```

2、df.iterrows()

```
# 迭代，使用name、Q1数据
for index, row in df.iterrows():
    print(index, row['name'], row.Q1)
```

3、df.itertuples()

```
for row in df.itertuples():
    print(row)
```

4、df.items()

```
# Series取前三个
for label, ser in df.items():
    print(label)
    print(ser[:3], end='\n\n')
```

5、按列迭代

```
# 直接对DataFrame迭代
for column in df:
    print(column)
```

07、函数应用

1、pipe()

应用在整个DataFrame或Series上。

```
# 对df多重应用多个函数
f(g(h(df), arg1=a), arg2=b, arg3=c)
# 用pipe可以把它们连接起来
(df.pipe(h)
    .pipe(g, arg1=a)
    .pipe(f, arg2=b, arg3=c)
)
```

2、apply()

应用在DataFrame的行或列中，默认为列。

```
# 将name全部变为小写
df.name.apply(lambda x: x.lower())
```

3、applymap()

应用在DataFrame的每个元素中。

```
# 计算数据的长度
def mylen(x):
    return len(str(x))
df.applymap(lambda x:mylen(x)) # 应用函
df.applymap(mylen) # 效果同上
```

4、map()

应用在Series或DataFrame的一列的每个元素中。

```
df.team.map({'A':'一班', 'B':'二班','C'
df['name'].map(f)
```

5、agg()

```
# 每列的最大值
df.agg('max')
# 将所有列聚合产生sum和min两行
df.agg(['sum', 'min'])
# 序列多个聚合
df.agg({'Q1' : ['sum', 'min'], 'Q2' :
# 分组后聚合
df.groupby('team').agg('max')
df.Q1.agg(['sum', 'mean'])
```

6、transform()

```
df.transform(lambda x: x*2) # 应用匿名函
df.transform([np.sqrt, np.exp]) # 调用
```

7、copy()

```
s = pd.Series([1, 2], index=["a","b"])
s_1 = s
s_copy = s.copy()
s_1 is s # True
s_copy is s # False
```