前置package：

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

from statsmodels.tsa.stattools import adfuller

import math

import logging

目录

[class DataClean(object):代码注释 1](#_Toc144429030)

[类说明 1](#_Toc144429031)

[方法说明 2](#_Toc144429032)

[class AutoCorr (object):代码注释 4](#_Toc144429033)

[类说明 4](#_Toc144429034)

[方法说明 4](#_Toc144429035)

[class DataNormalizer (object):代码注释 6](#_Toc144429036)

[类说明 6](#_Toc144429037)

[方法说明 7](#_Toc144429038)

[实操实例 8](#_Toc144429039)

### class DataClean(object):代码注释

## 类说明

这个类用于处理输入的dataframe、dataframe的某一列或是series（这里统称为数据集）当中所出现的na值，0值和波动过大的异常值（例如价格变动大于涨跌停板幅度）。三种异常值的处理分别对应三个方法，其中包含：

1. def clean\_na(self, input\_na):将数据集当中的na值用na值之前的值替代（如果na值是第一位，则用后一位替代），同时打印出现na值的位置
2. def clean\_zero(self, input\_zero): 将数据集当中的0值用0值之前的值替代，同时打印出现0值的位置​
3. def clean\_abnormal(self, input\_abnormal, val=0.2): 通过变量val设定数据的波动上下限（这里默认为20%）将数据集当中的异常值（波动超过）用异常值之前的值替代，同时打印出现异常值的位置​

## 方法说明

1. 初始化变量

--init—当中的变量仅包含创建日志用的变量，若无需创建日志则可以#此部分

**def \_\_init\_\_(self, log\_filename='app.log'):**

**# Configure logging for the instance**

**logging.basicConfig(filename=log\_filename, filemode='w', format='%(name)s - %(levelname)s - %(message)s', level=logging.INFO)**

**self.logger = logging.getLogger(\_\_name\_\_)**

1. **def clean\_na(self, input\_na):** 图形用户界面, 文本

   描述已自动生成

首先判断输入变量是否为dataframe或series。若为series则进入if将其转变为dataframe；若都不是，则输出提示然后结束运行

检测NA值：

文本

描述已自动生成

检测input\_na这个输入变量中的每个值是否为Na。这会返回一个与input\_na形状相同的DataFrame，其中的值为True（如果相应的值是Na）或False。

之后遍历na\_positions的每一列。遍历每一行的值。这里使用enumerate函数同时返回行的索引（index）和该行是否为Na用于后面的if判定。

if is\_na:使用日志记录器记录Na值的位置并且打印。并且如果当前Na值不是该列的第一个值，则将Na值替换为其前一个值；如果当前Na值是该列的第一个值，则将Na值替换为其后一个值。（注：如果头两行都是Na值则需要另外处理）

1. **def clean\_zero(self, input\_zero):**

该方法逻辑同上。但该方法在当第一行为0时不会处理0值，即不会去第二行值覆盖0值。另一方面也需要留意数据集头几行连续出现0值得情况。

1. **def clean\_abnormal(self, input\_abnormal, val=0.2):**

首先判断输入变量是否为dataframe或series的部分同上

修改波动过大的异常值：

文本

描述已自动生成

因为输入的dataframe可能会携带日期列，这里为了跳过日期格式的列，这里使用pandas的 infer\_objects()方法来检查每列是否为日期格式。

if pd.api.types.is\_datetime64\_any\_dtype(input\_abnormal[column]):

continue

如果是，使用continue语句跳过该列。如果不是，则将该列的每行前一行的值转换为float的格式赋予变量prev\_value，将该列的每一行的值转换为float的格式赋予变量current\_value。这里还添加了一个try，except模块来处理可能无法转换为浮点数的值，这样可以安全地跳过这些值而不会引发错误。同时添加一个if判断逻辑用来跳过在之后的运算中出现分母为0的情况。

for index in range(1, len(input\_abnormal)):

try:

prev\_value = float(input\_abnormal.at[index - 1, column])

current\_value = float(input\_abnormal.at[index, column])

except ValueError: # If transform to float fails, skip

continue

if prev\_value == 0: # Avoid 0 becomes demomenator

continue

计算每一行的值，相对于前一行的波动。然后如果波动过大（fluctuation > val），则使用前一行的值替代这一行的值

文本

描述已自动生成

### class AutoCorr (object):代码注释

## 类说明

此类用于自动化检测dataframe的所有列或一个series的时间序列（以下统称为序列）是否为stationary。其中包含：

1. ADF\_test(self)：绘制输入变量的pacf 和 acf图,并对序列做ADF检验。

将是stationary的序列放入data\_stationary\_original这个局部dataframe中；将是non-stationary的序列放入data\_non\_stationary\_original这个局部dataframe中

1. Differencing(self）:对序列进行一阶差分 ∇Yt​=Yt​−Yt−1​

绘制差分后的变量的pacf 和 acf图,并对序列做ADF检验。

将是stationary的序列放入data\_stationary\_diff这个局部dataframe中,并删除data\_non\_stationary\_diff当中的相同序列；将是non-stationary的序列放入data\_non\_stationary\_diff这个局部dataframe中。

1. 最终局部变量data\_stationary\_diff会储存差分后stationary的序列

## 方法说明

**1.def \_\_init\_\_(self,data\_set):**初始化变量

self.DATA\_SET\_all=data\_set

self.DATA\_SET=None

self.data\_stationary\_diff = pd.DataFrame()

self.data\_non\_stationary\_diff = pd.DataFrame()

self.data\_stationary\_original = pd.DataFrame()

self.data\_non\_stationary\_original = pd.DataFrame()

**2.def ADF\_test(self):**

文本

描述已自动生成

首先判断输入变量是否为dataframe或series。若为dataframe则进入if；若为series则进入elif；若都不是，则输出提示

If嵌套当中：

文本

描述已自动生成逐一对输入的dataframe的每一列绘制1.ACF图，2.PACF图，3.做ADF检验，4.打印检验结果，5.判断检验是否显著，并依此将序列放入stationary的集合里或是non-stationry的集合里

Elif内逻辑同上。

**3.def differencing(self):**

文本

低可信度描述已自动生成

同上一方法一样，首先判断输入的变量的格式

If内嵌套：

文本

描述已自动生成

对序列做一阶差分，并将0赋予na值（即第一位数）

文本

描述已自动生成

之后同def ADF\_test(self)一样，画出图像并做自相关性检验。在将序列放入stationary的局部dataframe时会将non-stationary里的同名列删去（例如，对非平稳序列进行第二次差分并使序列变为stationary后，也会将该序列从标识non-stationary的局部dataframe中剔除）

### class DataNormalizer (object):代码注释

## 类说明

此类用于标准化dataframe的所有列或一个series其中包含：

1. def \_\_init\_\_(self, input\_normalize)：判断输入变量类型是否为series或dataframe
2. def \_normalize\_method(self, column, method)：

包含各类标准化的方法，目前包括z-score，min-max, robust 三种。可按需求添加其他方法

1. def normalize(self, method='z-score', columns=None):

通过method来调用\_normalize\_method当中的一种计算方法来标准化数据集

## 方法说明

**1.def \_\_init\_\_(self, input\_normalize):**

文本

描述已自动生成

判断数据格式是否正确

**2. def \_normalize\_method(self, column, method):**

通过输入的method变量来选定标准化的方法

文本

描述已自动生成

**3.** def normalize(self, method='z-score', columns=None):

对所有dataframe的所有列使用相同的标准化方法。

电视萤幕的截图

描述已自动生成

## 实操实例

1. **读取数据，更改格式**

文本

描述已自动生成若数据板式固定，亦可将此部分封装进类当中

1. **数据清理**

文本

描述已自动生成

**发现数据中存在na和异常值（波动上限设定为0.3），打印位置并将na值替换**



1. **平稳性测试**

文本

描述已自动生成

此处用dataframe做例子，用obj将类实例化。最后将差分好的序列集打印出来

屏幕上有字

描述已自动生成

使用series的例子：



输出结果为：

文本

描述已自动生成

1. **标准化**

手机屏幕的截图

描述已自动生成

**输出结果**

文本

中度可信度描述已自动生成