Pyhton数据分析

- conda常用操作

```
#查看虚拟环境
conda info -e
#创建虚拟环境
conda create -n your_env_name python=your_python_version
#删除虚拟环境
conda remove -n your_env_name --all
#进入指定环境
conda activate your_env_name
#退出指定环境
conda deactivate your_env_name

#镜像源安装
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple +包名
```

- numpy库

```
#导入 import numpy as np
#方法:
    np.array([],dtype='int8') dtype 用于指定数组的数据类型
    np.round(array,num) 保留小数
    np.count_nonzero(t1):计算数组中不为0的元素个数
    np.isnan(t1):获取t1中数据类型为nande
    np.loadtxt(frame,dtype,delimiter=None,skiprows=0,usecols=None,unpack=False)
 读取本地数据
            参数：frame:文件，字符串或产生器，可以是.gz或bz2压缩文件
                  dtype:数据类型，可选，csv的字符串以什么数据类型读入数组中，默认np.float
                  delimiter:分割字符串，默认时任何空格，改为逗号
                  skiprows: 跳过前x行，一般跳过第一行表头
                  usecols:读取指定的列，索引，元组类型
                  unpack:如果为True,读入属性分别写入不同数组变量，False读入数据只写入一个
数组变量，默认False
#numpy 中的nan:

    1.两个nan是不相等的     nan(NAN,Nan):not a number表示不是一个数字，当读取本地文件为
float的时候，如果有数据缺失，就会出现nan,当做了一个       不合适的计算时也会出现
    2.np.nan!=np.nan
    3.np.count_nonzero(t!=t):利用nan不相等的特性判断nan的个数
    4.由于np.nan!=np.nan 通过np.isnan(a)l来判断，返回bool类型，比如将nan替换为0----》
t[np.isnan(t)]=0
    5.nan和任何值计算都为nan

#创建数组:
          a=np.array([1,2,3,4,5])
          b=np.array(range(1,6))
          c=np.arange(1,6)
          np.arange的用法: arange([start,] stop[,step],dtype=None)
          np.zeros((3,4)):创建一个全为0的数组
          np.ones((3,4)):创建一个全是1的数组
          np.eye(3):创建一个对角线为1的正方形数组（方阵）
```

#numpy生成随机数数组：
.rand(d0,d1,..dn) ：创建d0-dn维度的均匀分布的随机数组，浮点数，范围从0-1
.randn(d0,d1,..dn)：创建d0-dn维度的标准正态分布随机数，浮点数，平均数0，标准差1
.randint(low,high,(shape))：从给定上下限范围选取随机整数，范围是low,high,形状是shape
.uniform(low,high,(size))：产生具有均匀分布的数组，low起始值，high结束值，size形状
.normal(loc,scale,(size))：从正太分布中随机抽取样本，分布中心是loc(概率分布的均值)，标准差是scale，形状是size
.seed(s)：随机数种子，s是给定的种子值，因为计算机生成的是伪随机数，所以通过设定相同的随机数种子，可以每次生成相同的随机数

#拼接数组：
np.vstack((t1,t2))：竖直拼接
np.hstack(t1,t2)：水平拼接
np.argmax(t,axis=0)：获取最大值的位置
np.argmin(t,axis=1)：获取最小值的位置
np.arange的用法：arange([start,] stop[,step],dtype=None)

#数组值修改：
np.where(t<10,5,6)：三元运算符，t<10 ? 5:6
clip(10,18)：小于10的替换成10，大于18的替换成18 （nan浮点型）
transpose() / .T / swapaxes(1,0) 转置
astype() 用于修改数组的数据类型
shape()查看数组的形状
reshape 修改数组的维度
#数组的计算：
求和：np.sum(t1,[axis=0|1])：当axis=0时求得是数组中每一列的和，当axis=1时，计算数组中每一行的和
均值：np.mean(a,axis=None) 受离群点的影响较大
中值：np.median(t,axis=None)
最大值：t.max(axis=None)
最小值：t.min(axis=None)
极值：np.ptp(t,axis=None)
标准差：t.std(axis=None)：标准差是一组数据平均值分散程度的一种度量，一个较大的标准差，带表大部分数值和其平均值之间差异较大;一个较小的 标准差，代表这些数值较接近平均值反映出的数据波动稳定情况，越大表示波动越大，越不稳定。
默认返回多维数组的全部统计结果,如果指定axis则返回一个当前轴上的结果

```
import numpy as np
t1=np.array([1.0235,2.3456,3.5678])
print(t1.dtype) #打印数组的数据类型
t1=t1.astype('float') #为数组指定数据类型
print(t1.dtype)
t1=np.round(t1,2)#保留小数点后两位
s=t1.shape#数组形状
print(s)


#数组切片和索引
import numpy as np
path='./test.csv'
t1=np.loadtxt(path,delimiter=",",dtype="int")
print("本地文件：text.csv\n",t1)
print("*"*30)
print("t1[行，列]")
```

```
print('****行操作：')
print("取第四行t1[3]：\n",t1[3])
print("取不连续的多行t1[[0,3]]：\n",t1[[0,3]])
print("取连续的多行t1[::2]：\n",t1[::2])
print('\n****列操作：')
print("取第n列:t1[:,0]\n",t1[:,0])
print("取某行某列:4行4列t1[3,3]\n",t1[3,3])
print("取连续的列t1[:,2:]\n",t1[:,2:])
print("取不连续的多列0列和2列t1[:,[0,2]]：\n",t1[:,[0,2]])
print("取多行多列：\n  取第三行到第五行，第2列到第4列\n",t1[2:5,1:4])
print("取多个不相邻的点\n",t1[[0,2,2],[0,1,3]])
print("转置之后:\n",t1.T)
print("t1小于10",t1<10)
print("数组的替换：\n")
t1[t1<10]=3
print("t1中t1小于10的令其等于3，t1[t1<10]=3\n",t1)
t2=np.where(t1<10,2,8)
print("t1中小于10的令其等于2，大于10的令其等于8\n",t2)
t1=t1.clip(10,18)
print("将t1中小于等于10的替换成10，大于等于18的替换成18\n",t1)
print("数组的拼接:\n")
print("数组的行列交换：\n行交换：t[[1,2],:]=t[[2,1],:]\n列交换：t[:,[0,2]]=t[:,
[2,0]]")

import numpy as np
t1=np.zeros((3,4)) #生成一个3行4列数值为0的数组
t2=np.ones((3,4))  #生成一个3行4列数值为1的数组
t3=np.eye(5)       #生成一个对角线为1的正方形方阵
print(t1,'\n',t2)
print(t3)

import numpy as np
np.random.seed(3)
t=np.random.randint(0,20,(3,4))
num=np.count_nonzero(t)
t1=np.isnan(t)
print(t)
print("数组中是否含有nan:\n",t1)
print("数组中不为0的元素个数",num)
print("数组t行的总和：",np.sum(t,axis=1))
print("数组t列的总和：",np.sum(t,axis=0))

#给nan赋值
import numpy as np
t1=np.arange(12).reshape(3,4).astype("float")
t1[1,2:]=np.nan
def fill_ndarray(t1):
     for i in range(t1.shape[1]):  #遍历每一列
            temp_col=t1[:,i]  #当前的一列
            nan_num=np.count_nonzero(temp_col!=temp_col)
            if nan_num!=0:  #不为0，说明当前这一列中有nan
                temp_not_nan_col=temp_col[temp_col==temp_col]  #当前一列不为nan的
array
                #选中当前为nan的位置，把值赋值为不为nan的均值
                temp_col[np.isnan(temp_col)]=temp_not_nan_col.mean()
     return t1
if __name__=='__main__':
    t1=np.arange(12).reshape((3,4)).astype("float")
```

```
    t1[1,2:]=np.nan
    print(t1)
    t1=fill_ndarray(t1)
    print(t1)
```

- pandas库

```
# 导入：
    import pandas as pd
# 拓展：
    import string
    string.ascill_uppercase[i] 0<i<26 可取到对应的26个大写字母
# 方法：
    pd.Series(list,index=""):列表创建
    t.index: 获取Series中的索引（可变量，可迭代,可用list强制转换成列表，进行切片操作）
    t.values:获取Series中值，与index具有同样的属性
    where(t<10,n): t<10的替换成n
    clip(10,18):小于10的替换成10，大于18的替换成18 （nan浮点型）
    argmax(t,axis=0):获取最大值的位置
    argmin(t,axis=1):获取最小值的位置
    pd.read_csv(path):读取path路径下的文件
    pd .read_sql(sql,connect):sql语句 connect链接
    pd.DataFrame(数组对象，index,columns): 当axis(index)=0时，行索引，当axis(index)=1
时，列索引
# Series的创建：带标签的数组（键，值）
# 字典推导式创建：
    a={string.ascii_uppercase[i]:i for i in range(10)}
# 列表创建：
    pd.Series(list,[index=""])  可为list指定等长的索引index，默认时数字
# Series的切片：

# dataFrame的操作：
     df.shape:行数,列数
     df.dtypes:列数据类型
     df.ndim:数据维度
     df.columns:列索引
     df.values:对象,二维ndarray数组
     df.head(3) #显示头部几行,默认5行
     df.tail(3) #显示末尾几行,默认5行
     df.info()  #相关信息概览
     df.describe()  #快速综合统计接过:计数,均值,标准差,最大值,四分位数,最小值
     df.sort_values(by="字段",ascending):按照某字段排序 ascending=True升序排列,false
降序排列
     loc("行字段", "列字段"):  可获取数据和赋值改变数据
     iloc(行控制，列控制):  可获取数据和赋值改变数据   ：在前不包含后面的   ：在后包含前面的
eg:[:2,1:] 第2行之前，第1列之后，前不包，后包

# pandas(str) 字符串操作：
     cat:实现元素级的字符串操作，可指定分割符
     contains:返回表示各字符串是否含有指定模式的布尔数组
     count:模式出现的次数
     endswith,startswith:相当于对各个元素执行x.endswith(pattern)或
x.startswith(pattern)
     findall:计算各字符串的模式列表
     get:获取各元素的第i个字符
     join:根据指定的分隔符将Series中各元素的字符串连接起来
     len:计算各字符串的长度
```

```
      lower,upper: 转换大小写，相当于各元素执行x.lower()或x.upper()
      match: 根据指定的正则表达式对各元素执行re.match
      pad: 在字符串的左边，右边或左右两边添加空白符
      center: 相当于pad(side='both')
      repeat: 重复值，例如，s.str.repeat(3) 相当于对各个字符串执行x*3
      replace:用于指定字符串替换找到的模式
      slice: 对Series中的个各个字符串进行子串截取
      split: 根据分隔符或正则表达式对字符串进行拆分
      strip,rstrip,lstrip: 去除空白符,包括换行符,相当于对各个元素执行
x.strip(),x.strip(),x.lstrip()
# 缺失数据的处理：
      pd.isnull() / pd.notnull()  判断是否为NaN
      方法1：dropna(axis=0,how='any',inplace=False)  删除NaN所在的行列
      方法2：t.fillna(t.mean(),t.fillna(t.median()),t.fillna(0))  填充数据
# 处理为0的数据：
    t[[t==0]=np.nan]  注意：并不是每次为0的数据都需要处理，计算平均值等情况，nan不参与计算，
但0会
-------------------------------------------------------------------------------
-----------------------------------------
import pandas as pd
import string
#直接创建Series
t=pd.Series([1,2,3,4,5,6],index=list("abcdef"))
print(t)
#字典推导式创建Series
a={string.ascii_uppercase[i]:i for i in range(26)}
a=pd.Series(a)
print(a)
#列表创建Series
list1=[1,2,3,5,6,6,6,8,9,545,5,8]
b=pd.Series(list1,index=[string.ascii_uppercase[i] for i in range(len(list1))])
print(b)
df=pd.read_csv('./dog.csv')
print(df)

#DataFrame的应用
import pandas as pd
import numpy as np
##通过numpy创建：
t1=pd.DataFrame(np.arange(12).reshape(3,4))
print("默认索引：\n",t1)
t2=pd.DataFrame(np.arange(12).reshape(3,4),index=list("abc"),columns=list("wxyz"
))
print("指定索引：\n",t2)
##通过字典创建：
d1={"name":["张三","李四","王五"],"age":[20,36,18],"tel":
["13991426365","18456792012","13991426629"]}
t3=pd.DataFrame(d1)
print(t3)
print(type(t3))

#pandas之取行或者取列操作
import pandas as pd
df=pd.read_csv("./dog.csv")
df=df.sort_values(by="Count_AnimalName",ascending=False)#按照Count_AnimalName降序
排列  默认ascending=True为升序排列
print('输出文件所有信息:\n',df)
print("*"*35)
```

```
print('输出文件前20条信息(取行操作):\n',df[:20])
print('输出文件某字段的全部信息（取列操作）:\n',df["Row_Labels"])

#pandas之loc操作
import pandas as pd
import numpy as np
t3=pd.DataFrame(np.arange(12).reshape(3,4),index=list("abc"),columns=list("WXYZ"
))
print("全部信息：\n",t3)
print("loc某行某列：\n",t3.loc["b","X"])#某行某列
print("loc某行多列：\n",t3.loc["b",["Y","Z","W"]])#某行多列
print("loc某列多行：\n",t3.loc[["a","b","c"],"W"])#某列多行
print("当前列：\n",t3.loc[:,"Y"])#当前列
print("当前行：\n",t3.loc['c',:])#当前行
print("多行多列：\n",t3.loc["a":"c","X":"Z"])#多行多列


#pandas之iloc
import pandas as pd
import numpy as np
t4=pd.DataFrame(np.arange(12).reshape(3,4),index=list("abc"),columns=list("WXYZ"
))
print("全部信息:\n",t4)
print("当前行：\n",t4.iloc[1,:])
print("当前列：\n",t4.iloc[:,1])
print("当前任意列：\n",t4.iloc[:,[1,2]])
print("当前任意行：\n",t4.iloc[[1,2],:])
print("某行某列：\n",t4.iloc[2,1])
print("多行多列（不连续）：\n",t4.iloc[[1,2],[1,2]])
print("某行之后的多行(不包含)，某列之后的多列（不包含）：\n",t4.iloc[:1,2:]) #第1行之前，第
2列之后
print("行后列前：\n",t4.iloc[1:,:2]) #第1行之后，第2列之前（不包含第2列）（前不包，后包）
print("行前列后：\n",t4.iloc[:2,1:]) #第2行之前，第1列之后（不包含第2行）
t3.iloc[:2,1:]=30  #赋值操作
print("赋值：\n",t3.iloc[:2,1:])
# print("",t3.iloc[[2,1]])

#pandas之布尔索引
import pandas as pd
import numpy as np
df=pd.read_csv("./dog.csv")  # &表示且，|表示或
print(df[(80<df["Count_AnimalName"])&(df["Count_AnimalName"]<1000)])

#panda之字符串操作
import pandas as pd
import numpy as np
pd=pd.read_csv("./dog.csv")
get=pd.loc[11,["Row_Labels"]]#获取第11行,键值为"Row_Labels"的信息
print(get.str.split("/").tolist())
print(pd)# 输出文件内所有信息
```

- matplotlib库

```
# 拓展导入字体:
     方法1：windows和linux设置字体的方式(用不了)
               font={'family':'MicroSoft YaHei',
                     'weight':'bold',
                     'size':'larger'}
            matplotlib.rc("font",**font)
            matplotlib.rc("font",family='MicriSoft Yahei',weight="bold")

     方法2：from matplotlib import font_manager导入字体设置模块
           my_font=font_manager.FontProperties(fname="path") path:字体存在的路径

     方法3：plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
           plt.rcParams['axes.unicode_minus'] = False #用来正常显示负号
           plt.rcParams['font.size']=12


from matplotlib import pyplot as plt 导入matplotlib库
# 方法：  plt.figure(figsize=(n1,n2),dpi) 设置图片大小
# 散点图：plt.scatter(x,y,[label="name",color=" ",linestyle=":/--",linewidth=""])
# 条形图：plt.bar(x,y,width)竖着 plt.barh(x,y,height)
# 折线图：plt.plot(x,y,[label="name",color=" ",linestyle=":/--",linewidth=""]) 绘
制图片  label：线名  linewidth:线条粗细
# 直方图：plt.hist(list,分组数):
         plt.xticks(x,[str],rotation=90) 设置x轴坐标 可通过列表的步长(间隔取
值),rotation:旋转角度,
                                          str:字符串用于替换变量x,作为轴的横坐标
         plt.yticks() 设置y轴坐标

# 字符串字体设置使用# font_manager模块
         plt.savefig(path) 保存图片
         plt.show() 输出展示图片
         plt.title() 图表标题
         plt.xlabel() x轴描述信息
         plt.ylabel() y轴描述信息
         plt.grid([alpha=0.4,linestyle=":/--"],linewidth="")  绘制网格 alpha:网格透
明度,0-1
         plt.legend([prop=fonttype],[loc="upper left"]) 图例  prop:字体 loc: 图例出
现的位置
-------------------------------------------------------------------------------
-----------------------------------------
from matplotlib import pyplot as plt# 导入
x=range(2,26,2)# x点
y=[15,13,14.5,17,20,25,26,26,27,22,18,15]# y点

plt.figure(figsize=(20,8),dpi=80)#设置图片大小
plt.plot(x,y)#绘制图片
# plt.xticks(range(2,26))# 设置x轴的刻度
# plt.yticks(range(2,26))# 设置y轴的刻度
plt.savefig("./t1.png")#保存图片
plt.show()

实例(折线图) :列表a表示10点到11点的每一分钟的气温,如何绘制折线图观察每一分钟气温的变化情况?
      a=[random.randint(20,35) for i in range(120)]

from matplotlib import pyplot as plt
```

```
import random
from matplotlib import font_manager

my_font=font_manager.FontProperties(fname="./font/Dengl.ttf")#设置字体
y=[random.randint(20,35) for i in range(60)]
x=range(0,60)
y_1=[random.randint(2,15) for i in range(60)]
x_1=range(0,60)
plt.figure(figsize=(20,8),dpi=80)
plt.plot(x,y,label="白天",linestyle=":")
plt.plot(x_1,y_1,label="晚上")
_x=['10点{}分'.format(i) for i in range(60)]#生成字符串替换列表
plt.xticks(list(x)[::3],_x[::3],rotation=45,fontproperties=my_font)
plt.title("10点到12点的每一分钟的气温",fontproperties=my_font)
plt.ylabel("温度",fontproperties=my_font)
plt.xlabel("时间",fontproperties=my_font)
plt.grid(alpha=0.4)#绘制网格
plt.legend(prop=my_font,loc="upper left")
plt.show()
print(y)#现实每一分钟的温度
print(list(x)[::3])

# 实例(散点图):
from matplotlib import pyplot as plt
import random
from matplotlib import font_manager

my_font=font_manager.FontProperties(fname="./font/Dengl.ttf")
y=[random.randint(16,19) for i in range(15)]
x=range(0,15)
_y=[random .randint(16,33) for i in range(15)]
_x=range(32,47)
x_1=list(x)+list(_x)
_x_label=['3月{}号'.format(i) for i in x]0
_x_label+=['10月{}号'.format(i) for i in _x]
plt.figure(figsize=(20,8),dpi=80)
plt.xticks(x_1,_x_label,rotation=45,fontproperties=my_font)
plt.scatter(x,y,label="3月份",linestyle=":")
plt.scatter(_x,_y,label="10月份",linestyle=":")
plt.title("10点到12点的每一分钟的气温",fontproperties=my_font)
plt.ylabel("温度",fontproperties=my_font)
plt.xlabel("时间",fontproperties=my_font)
plt.grid(alpha=0.4)#绘制网格
plt.legend(prop=my_font,loc="upper left")
plt.show()

#实例(直方图):
from matplotlib import pyplot as plt
from matplotlib import font_manager

my_font=font_manager.FontProperties(fname="./font/Dengl.ttf")
a=["战狼2","速度与激情","功夫瑜伽","西游降魔片","最后的勇士"]
b=[65,74,32,66,52]
plt.figure(figsize=(20,8),dpi=80)
plt.barh(range(len(a)),b,height=0.3,color="blue")
plt.yticks(range(5),a,fontproperties=my_font)
# plt.grid(alpha=0.4)
plt.show()
```

```
# 3列表a中视频上映三天的播放次数
from matplotlib import pyplot as plt
from matplotlib import font_manager
my_font=font_manager.FontProperties(fname="./font/Dengl.ttf")
a=["战狼2","速度与激情","功夫瑜伽","西游降魔片","最后的勇士"]
b_16=[15476,312,556,778,664]
b_15=[13456,421,562,456,987]
b_14=[14565,522,456,122,333]

bar_width=0.2

x_14=list(range(len(a)))
x_15=[i+bar_width for i in x_14]
x_16=[i+bar_width*2 for i in x_14]

plt.bar(range(len(a)),b_14,width=bar_width,label="9月14")
plt.bar(x_15,b_15,width=bar_width,label="9月15号")
plt.bar(x_16,b_16,width=bar_width,label="9月16号")
plt.xticks(x_15,a,fontproperties=my_font)
plt.legend(prop=my_font)
plt.show()
```

- PyQt5

- ui文件转py文件

```
python -m PyQt5.uic.pyuic demo.ui -o demo.py
```

- 按钮控件的使用方法

```
from base_control.action_class import Action
from PySide2.QtWidgets import QApplication
from PySide2.QtUiTools import QUiLoader
from PySide2.QtCore import QFile,QSize
from PySide2.QtGui import QIcon
from PySide2.QtWidgets import QMessageBox

class Control1(Action):

    def __init__(self):

        ui_file = QFile("./uiDic/clickButton.ui")
        ui_file.open(QFile.ReadOnly)
        ui_file.close()

        # 从文件中加载UI定义
        # 从 UI 定义中动态 创建一个相应的窗口对象
        # 注意：里面的控件对象也成为窗口对象的属性了
        # 比如 self.ui.button , self.ui.textEdit
        self.ui = QUiLoader().load(ui_file)


    def buttonControl(self):
        '''
        普通按钮
        :return:
```

```
        '''
        self.ui.clickButton.setIcon(QIcon('./uiDic/lol.jpg'))   # 使用图片设置按钮图标
        self.ui.clickButton.setIconSize(QSize(30, 30))   # 设置图标大小
        self.ui.clickButton.setText("小跳蛙")   # 设置按钮文本
        self.ui.clickButton.setEnabled(True)    # 设置按钮是否被禁用，False为禁用，True为启用，应用场景：某些操作后按钮不允许点击
        self.ui.clickButton.clicked.connect(self.display)  # 将点击的信号连接到clickButton方法上，即点击后执行连接的方法

    def lineEditControl(self):
        '''
        单行文输入框
        :return:
        '''
        # .textChanged()函数会获取文本框值改变的信号，当获取这个信号时则执行后面的方法
        self.ui.lineEdit.textChanged.connect(self.handleTextChange)   #将单行文本框中输入的值作为参数传递给handleTextChange方法
        self.ui.lineEdit.returnPressed.connect(self.printText)  # 在光标在单行文本框内时，点击回车就调用连接的方法
        self.ui.lineEdit.setPlaceholderText('请在这里输入URL')  # 设置单行输入框内的暗纹提示语
        self.ui.lineEdit.setText("预设的内容")  # 在输入框内预设内容
        text = self.ui.lineEdit.text() # 获取单行文本框的输入的值
        QMessageBox.about(self.ui, '点击结果', text)    # 弹窗展示文本框输入的结果

    def plainTextEditControl(self):
        '''
        多行文本输入框
        :return:
        '''
        self.ui.plainTextEdit.textChanged.connect(self.handleTextChange1)  # Qt在调用这个信号处理函数时，不会传入文本框目前的内容字符串，作为参数,这个行为 和 单行文本框不同。

 self.ui.plainTextEdit.cursorPositionChanged.connect(self.handleTextChange1) # 当文本框中的光标位置变动，就会发出 cursorPositionChanged 信号，可以这样指定处理该信号的函数
        text = self.ui.plainTextEdit.toPlainText() # 获取多行文本框的内容
        # 获取 QTextCursor 对象
        textCursor =self.ui.plainTextEdit.textCursor()
        selection = textCursor.selectedText()
        self.ui.plainTextEdit.setPlaceholderText('测试多行输入文本框的暗纹提示') # 设置多行输入框内的暗纹提示语
        self.ui.plainTextEdit.setPlainText("设置多行输入文本框内的默认值")  # 设置多行文本框的内默认值
        self.ui.plainTextEdit.appendPlainText('末尾添加内容并换行')  # 在编辑框末尾添加文本内容并自动换行
        self.ui.plainTextEdit.insertPlainText('小工具')  # 光标处插入文本，不会自动换行
        self.ui.plainTextEdit.document().setMaximumBlockCount(1000)  # 设置多行文本框的最大行数

    def textBrowserControl(self):
        '''
        通常用来显示一些操作日志信息、或者不需要用户编辑的大段文本内容，内容无法编辑
        获取文本、设置文本、清除文本、光标出添加文本 等等， 都和上面介绍的 多行纯文本框是一样的
        :return:
        '''
```

```
        self.ui.textBrowser.append('多行展示文本框末尾添加文本')   # 多行展示文本框末尾添加文本，不会自动换行
        self.ui.textBrowser.ensureCursorVisible() #在末尾添加了内容，自动翻滚到当前添加的这行，自动换行

    def labelControl(self):
        '''
        常见的标签，可以用来显示文字（包括纯文本和富文本）、图片 甚至动画。
        怎么用QLabel 显示图片呢？
        可以在 Qt Designer上 属性编辑器 QLabel 栏 的 pixmap 属性设置中选择图片文件指定。
        :return:
        '''
        self.ui.label.setText("自定义标签框")

    def pushButton(self):
        self.ui.pushButton.clicked.connect(self.clearLine)
    def clearLine(self):
        '''
        文本清除
        :return:
        '''
        return self.ui.lineEdit.clear()   # .clear()清除输入框内的内容

    def handleTextChange(self,inputText):       # 当单行文本框调用该函数时，会将文本框内输入的字符串当做参数传入该方法，即：inputText=输入值
        '''
        捕捉文本框内容变化执行对应的方法
        :param inputText:
        :return:
        '''
        print(inputText)

    def handleTextChange1(self):
        print(1)

    def display(self):
        '''
        提示文本弹窗
        :return:
        '''
        QMessageBox.about(self.ui, '信息', '信息提示文本')   # 弹窗展示文本框输入的结果
        QMessageBox.information(self.ui, '信息', '信息提示文本')
        QMessageBox.critical(self.ui, '错误', '错误提示文本！')
        QMessageBox.warning(self.ui, '警告', '警告提示文本')

    def printText(self):
        print("hahah")


app = QApplication([])
stats = Control1()
stats.pushButton()
# stats.buttonControl()
stats.ui.show()
app.exec_()
```