技术文档

20172131138 曾德明 4班

————统计图生成软件

一、 开发过程

1、设计过程

前期准备:

下载MySQL5.7并进行安装，使用Navicat for MySQL连接数据库并进行操作，同时在网上寻找相关数据生成数据表（项目中使用了网上的一个关于出行方式调查的问卷的一部分数据进行生成），安装anaconda3和pycharm搭建好相应的python开发平台，即可进行项目开发

1.1、设计软件的主界面

由于软件的主要是用于生成并显示，所以主界面以简单为主，有四个下拉选择框，第一个是选择数据库中存在的数据表，第二个和第三个下拉框则是用于选择充当x轴和充当y轴的字段，第四格则是用于选择统计图表的类型；界面还有两个按钮，一个文本显示框，一个按钮用于分析数据，并生成二维交叉表，显示在文本框中，第二个按钮则是用于生成统计图。

1.2、设计数据库连接类和全局变量类

软件需要跟数据库进行连接，因此需要相应的数据库处理类来实现链接数据库，读取数据库信息等基本操作。同时，软件需要存储相关的全局变量，所以要设计一个全局变量类保存运行过程中需要存储的全局变量，以提供给所有模块使用。

1.3、实现处理数据库数据并生成二维表功能

二维交叉表则是软件的核心，没有二维交叉表，则没法进行画图，因为统计图是选择了两个字段分别充当x轴变量和y轴变量，因此它们之间有关联关系，不能直接进行画图。因此，当把数据库的数据读入后，要进行相应的处理，生成二维交叉表。然后根据图像类型，来选择行或列进行绘制统计图表。

1.4、根据不同的选择进行统计图表的绘制

统计图的类型有多种，在软件中实现了其中的三种，柱状图，饼状图，折线图。通过下拉框选择可以生成不同的统计图表，同时可以保存为svg格式。

1.5、关闭界面

直接关闭软件即可

2、实现过程

2.1、设计软件的主界面

首先新建一个 python 项目，然后引用 python 的第三方库 tkinter 进行 UI 的设计。主窗口则直接使用 rootWindow=TK（）进行生成，后边的控件则直接添加在主窗口中。

根据需求，需要放置四个下拉框，两个按钮，一个文本框因此在全局变量中添加相关控件。

```
# 表名下拉框
tablename = StringVar()     # StringVar 是 Tk 库内部定义的字符串变量类型，在这里用于管理部件上面的字符
com1 = ttk.Combobox(rootWindow, textvariable=tablename)
# 字段 1 下拉框
fieldname1 = StringVar()
fieldselect1 = ttk.Combobox(rootWindow, textvariable=fieldname1)
# 字段 2 下拉框
fieldname2 = StringVar()
fieldselect2 = ttk.Combobox(rootWindow, textvariable=fieldname2)
# 统计图类型下拉框
graphname = StringVar()
graphselect = ttk.Combobox(rootWindow, textvariable=graphname)
# 分析按钮和画图按钮
btn_analy = Button(rootWindow, text="分析数据", anchor="center", relief="raised") # 样式：居中凸起
btn_draw = Button(rootWindow, text="画图", anchor="center", relief="raised")
# 样式：居中凸起
# 滚动文本框
scr = scrolledtext.ScrolledText(rootWindow, width=90, height=20,font=("隶书",10))
```

然后通过 window_init()函数进行界面的初始化，如空间布局，窗口大小等，在每次运行软件之前，都先执行一次这个函数，然后再调用 rootWindow.mainloop()运行软件。

其中，下拉框控件和按钮控件需要绑定相关的函数以执行功能。

表名下拉框：下拉框的内容是数据库的数据表，

```
com1.bind("<<ComboboxSelected>>", get_com1_valus)  # 绑定事件,(下拉列表框被选中时，绑定函数)
```

```
def get_com1_valus(*args):
    """处理事件，即用户选定数据表事件，*args 表示可变参数 """
    usingtable = com1.get()
    model = dbmodel.DBModel(usingtable)     # 当选定了数据表则根据数据表名来
```

```
生成模型
    """
    result = model.findAll()
    datatext.delete('1.0', 'end')
    for v in result:
        datatext.insert('end', result)
        datatext.index('end', '\n')
    """
    if gl.get_value('model') is None:       # 如果还没保存模型，则直接添加全
局变量
        gl.set_value('model', model)
    else:                                   # 如果已经添加了，则先消除原本的实
例，再添加
        tmp = gl.get_value('model')
        del tmp
        gl.set_value('model', model)
    # 更改字段下拉框的可选项
    global fieldselect1
    global fieldselect2
    fieldselect1['value'] = tuple(model.get_fields())
    fieldselect2['value'] = tuple(model.get_fields())
```

字段名下拉框：下拉框的内容是数据表字段

```
label_fieldx = Label(rootWindow, text="x 轴的字段:")
label_fieldx.place(relx=0.4, rely=0.1)
global fieldselect1
fieldselect1.place(relx=0.4, rely=0.15)
fieldvalue1 = ("(还没选择数据库)")
fieldselect1['value'] = fieldvalue1

label_fieldy = Label(rootWindow, text="y 轴的字段:")
label_fieldy.place(relx=0.7, rely=0.1)
global fieldselect2
fieldselect2.place(relx=0.7, rely=0.15)
fieldvalue2 = ("(还没选择数据库)")
fieldselect2['value'] = fieldvalue2
```

分析数据按钮：

```
global btn_analy
btn_analy.place(relx=0.1, rely=0.2)
btn_analy.bind("<Button-1>", btn_analy_action) # 绑定动作函数<Button-1>为
鼠标左击事件
```

```
def btn_analy_action(event):
    field1 = fieldselect1.get()
    field2 = fieldselect2.get()
    gl.set_value('field1', field1)
    gl.set_value('field2', field2)
    if field1 == '' or field2 == '' or field1 == '(还没选择数据库)' or field2
== '(还没选择数据库)':
        # 弹出对话框
        result = tkinter.messagebox.askquestion(title='warming', message='
警告：你还没选择变量')
        # 返回值为：yes/no
        print(result)
    elif field1 == field2:
        # 弹出对话框
        result = tkinter.messagebox.askquestion(title='warming', message='
警告：不能选择相同的变量')
        # 返回值为：yes/no
        print(result)
    else:
        model = gl.get_value('model')
        if mf.analy_data(field1, field2, model):
            # 弹出对话框
            result = tkinter.messagebox.askquestion(title='warming',
message='提示：分析成功')
            # 显示二维表
            pcrosstab = gl.get_value('crosstab')
            pgroups = gl.get_value('groups')
            pfactors = gl.get_value('factors')
            #print(pgroups)
            #print(pfactors)
            #print(pcrosstab)
            xname = []
            for key in pgroups.keys():
                xname.append(key)
            scr.delete('1.0','end')
            scr.insert('end', '交叉表：' + '\n')
            for i in pfactors:
                scr.insert('end', i)
                scr.insert('end', '\t')
            scr.insert('end', '\n')
            x = 0
            for i in pcrosstab:
                for j in i:
                    scr.insert('end', j)
```

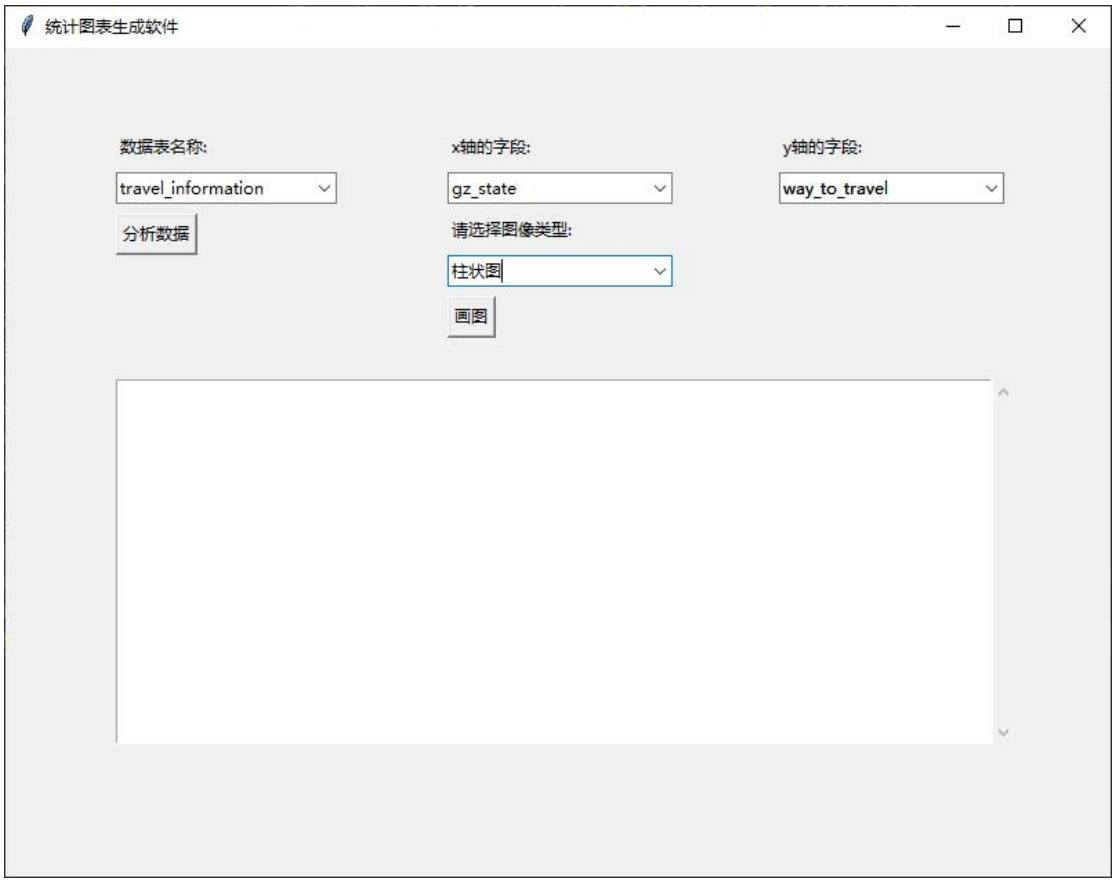
```
                scr.insert('end', '\t')
            #scr.insert('end', i)
            scr.insert('end', xname[x])
            x += 1
            scr.insert('end', '\n')

        else:
            # 弹出对话框
            result = tkinter.messagebox.askquestion(title='warming',
message='警告：分析失败')
```

生成统计图按钮：

```
global btn_draw
btn_draw.place(relx=0.4, rely=0.3)
btn_draw.bind("<Button-1>", btn_draw_action)  # 绑定动
```

```
def btn_draw_action(event):
    field1 = fieldselect1.get()
    field2 = fieldselect2.get()
    if field1 != gl.get_value('field1') or field2 != gl.get_value('field2'):
        result = tkinter.messagebox.askquestion(title='warming', message='
警告：还没进行数据分析')
    else:
        type = graphselect.get()
        if type == '':
            # 弹出对话框
            result = tkinter.messagebox.askquestion(title='warming',
message='警告：你还没选择类型')
            # 返回值为：yes/no
            print(result)
        else:
            mf.draw_graph(type)
```

最终样式：



2.2、设计数据库连接类和全局变量类

数据库连接类：db 模块

Config.py

```
# 建立数据库的连接信息
host = "localhost"      # 数据库的 ip 地址
user = "root"           # 数据库的账号
password = "hs123456"   # 数据库的密码
port = 3306             # mysql 数据库通用端口号
dbname = "test"         # 软件使用的数据库名
```

Dbmodel.py

```
import pymysql
from db import config

class DBModel:
    """单表信息操作类"""
    table_name = None   # 数据表名
    link = None         # 数据库连接对象
    cursor = None       # 游标对象
```

```
    pk = 'id'           # 主键字段名
    fields = []         # 表的所有字段名

    def __init__(self,tablename,config=config):
        """构造类方法，通过表名初始化，通过 config 获取基本信息"""
        try:
            # 表名
            self.table_name = tablename
            # 初始化字段名列表
            self.fields.clear()
            # 连接信息
            self.link = pymysql.connect(host = config.host, user =
config.user, password = config.password, db = config.dbname, charset =
"utf8")
            # 使用 cursor()方法创建一个游标对象 cursor，参数
pymysql.cursors.DictCursor 让查询结果元素以字典数据格式返回，默认是列表
            self.cursor = self.link.cursor(pymysql.cursors.DictCursor)
            # 调用内部方法，加载当前表的字段信息
            self.__loadFields()
        except Exception as err:
            print("Model 初始化报错，原因：%s" % err)

    def __loadFields(self):
        """加载当前表的字段信息,内部私有方法"""
        # sql
        sql = "SHOW COLUMNS FROM %s" % (self.table_name)
        # 执行 sql
        self.cursor.execute(sql)
        # 获取所有记录，返回的是字典列表，每个元素类似这样：{'Field': 'id',
'Type': 'int(11)', 'Null': 'NO', 'Key': 'PRI', 'Default': None, 'Extra':
'auto_increment'}
        dlist = self.cursor.fetchall()
        # print(dlist)
        # 遍历所有字段信息
        for v in dlist:
            # 收集每个字段名称
            self.fields.append(v['Field'])
            # 判断并收集表的主键名称
            if v['Key'] == 'PRI':
                self.pk = v['Field']

    def get_table_name(self):
        return self.table_name
```

```
    def get_fields(self):
        return self.fields

    def findAll(self):
        """获取当前表的所有信息，没有信息返回空列表[]"""
        try:
            sql = "select * from %s" % (self.table_name)
            self.cursor.execute(sql)
            return self.cursor.fetchall()
        except Exception as err:
            print("SQL 查询执行错误，原因：%s" % err)
            return []

    def find(self, id):
        """获取指定主键值的当条信息，没有返回None"""
        try:
            sql = "select * from %s where %s='%s'" % (self.table_name, self.pk,
id)
            self.cursor.execute(sql)
            return self.cursor.fetchone()
        except Exception as err:
            print("SQL 查询执行错误，原因：%s" % err)
            return None

    def select(self, where=[], order=None, limit=None):
        '''获取当前表的所有信息，没有信息返回空列表[]'''
        try:
            sql = "select * from %s " % (self.table_name)
            # 判断并封装 where 搜索条件
            if isinstance(where, list) and len(where) > 0:
                sql += " where " + " and ".join(where)
            # 判断并封装 order 排序条件
            if order is not None:
                sql += " order by " + order
            # 判断并封装 limit 条件
            if limit is not None:
                sql += " limit " + str(limit)
            print(sql)
            self.cursor.execute(sql)
            return self.cursor.fetchall()
        except Exception as err:
            print("SQL 查询执行错误，原因：%+s" % err)
            return []
```

```
    def save(self, data={}):
        ''' 添加数据方法，通过字典参数 data 传递要添加的信息，并实现添加操作'''
        try:
            # 组装 sql 语句
            keys = []
            values = []
            for k, v in data.items():
                if k in self.fields:
                    keys.append(k)
                    values.append(v)
            sql = "insert into %s(%s) values(%s)" % (self.table_name,
','.join(keys), ','.join(['%s'] * len(values)))
            # print(sql)
            # 指定参数，并执行 sql 添加
            self.cursor.execute(sql, tuple(values))
            # 事务提交
            self.link.commit()
            # 获取并返回最新自增 ID
            return self.cursor.lastrowid
        except Exception as err:
            print("SQL 添加执行错误，原因：%s" % err)
            return 0

    def update(self, data={}):
        '''修改方法，通过字典参数 data，传递要修改信息，并完成修改操作'''
        try:
            # 组装 sql 语句
            values = []
            for k, v in data.items():
                if (k in self.fields) and (k != self.pk):
                    values.append("%s='%s'" % (k, v))
            sql = "update %s set %s where %s='%s'" % (self.table_name,
','.join(values), self.pk, data.get(self.pk))
            print(sql)
            # 指定参数，并执行修改 sql
            self.cursor.execute(sql)
            # 事务提交
            self.link.commit()
            # 返回数据条数或影响行数
            return self.cursor.rowcount
        except Exception as err:
            print("SQL 修改执行错误，原因：%s" % err)
            return 0
```

```
    def delete(self, id=0):
        '''接收参数值 id 并执行删除对应的数据信息'''
        try:
            # 组装 sql 语句
            sql = "delete from %s where %s='%s'" % (self.table_name, self.pk,
id)
            print(sql)
            # 指定参数，并执行修改 sql
            self.cursor.execute(sql)
            # 事务提交
            self.link.commit()
            # 返回数据条数或影响行数
            return self.cursor.rowcount
        except Exception as err:
            print("SQL 修改执行错误，原因：%s" % err)
            return 0

    def __del__(self):
        # 关闭游标对象
        if self.cursor != None:
            self.cursor.close()
        # 关闭数据库连接
        if self.link != None:
            self.link.close()
```

全局变量类：

全局变量存储在字典中，键值对为 变量名-变量值。通过两个函数设置全局变量，get_value()通过变量名获取变量值，set_value()通过变量值和变量名来设置变量。

```
# 全局变量管理模块(global 关键字仅限于在一个 py 文件中调用全局变量)
from db import dbmodel

def _init():
    """使用键值对存储全局变量"""
    global _global_dict
    _global_dict = {}

def set_value(name, value):
    """设置全局变量"""
    _global_dict[name] = value

def get_value(name, defValue=None):
    """获取全局变量，如果没有则返回 None"""
```

```
    try:
        return _global_dict[name]
    except KeyError:
        return defValue
```

2.3、实现处理数据库数据并生成二维表功能

读取数据后，通过字段名，统计该字段有多少个不同的值，生成相应的键值对，然后通过键值对，在二维表中统计频数。最后生成二维交叉表

```
def analy_data(field1, field2, model):
    """进行数据表的分析"""
    len1 = 0
    len2 = 0
    groups = {}
    factors = {}
    crosstab = []
    percrosstab = []
    result = model.findAll()
    if field1 == 'age':
        i = 0
        groups = cut(field1)
        ages_valus = gl.get_value('ages_valus')
        for v in result:
            if v[field2] not in factors:
                factors[v[field2]] = i
                i += 1
        # 建立二维交叉表
        len1 = len(groups)
        len2 = len(factors)
        for i in range(len1):
            crosstab.append([0]*len2)
        for v in result:
            # 先获取年龄的值，再根据年龄的值获取年龄的下标
            crosstab[ages_valus[v[field1]]][factors[v[field2]]] += 1
        # 建立频率二维表
        for i in range(len1):
            percrosstab.append([0]*len2)
        sum = []
        sum.clear()
        for i in range(len1):
            n = 0
            for j in range(len2):
                n += crosstab[i][j]
```

```
            sum.append(n)
        # print('sum')
        # print(sum)
        for i in range(len1):
            for j in range(len2):
                percrosstab[i][j] = crosstab[i][j] / sum[i]
    elif field2 == 'age':
        i = 0
        factors = cut(field2)
        ages_valus = gl.get_value('ages_valus')
        for v in result:
            if v[field1] not in groups:
                groups[v[field1]] = i
                i += 1
        # 建立二维交叉表
        len1 = len(groups)
        len2 = len(factors)
        for i in range(len1):
            crosstab.append([0] * len2)
        for v in result:
            # 先获取年龄的值，再根据年龄的值获取下标值
            crosstab[groups[v[field1]]][ages_valus[v[field2]]] += 1
        # 建立频率交叉表
        for i in range(len1):
            percrosstab.append([0]*len2)
        sum = []
        sum.clear()
        for i in range(len1):
            n = 0
            for j in range(len2):
                n += crosstab[i][j]
            sum.append(n)
        # print('sum')
        # print(sum)
        for i in range(len1):
            for j in range(len2):
                percrosstab[i][j] = crosstab[i][j] / sum[i]
    else:
        i = 0
        j = 0
        for v in result:
            # 对 x 轴元素进行分组编号
            if v[field1] not in groups:
                groups[v[field1]] = i
```

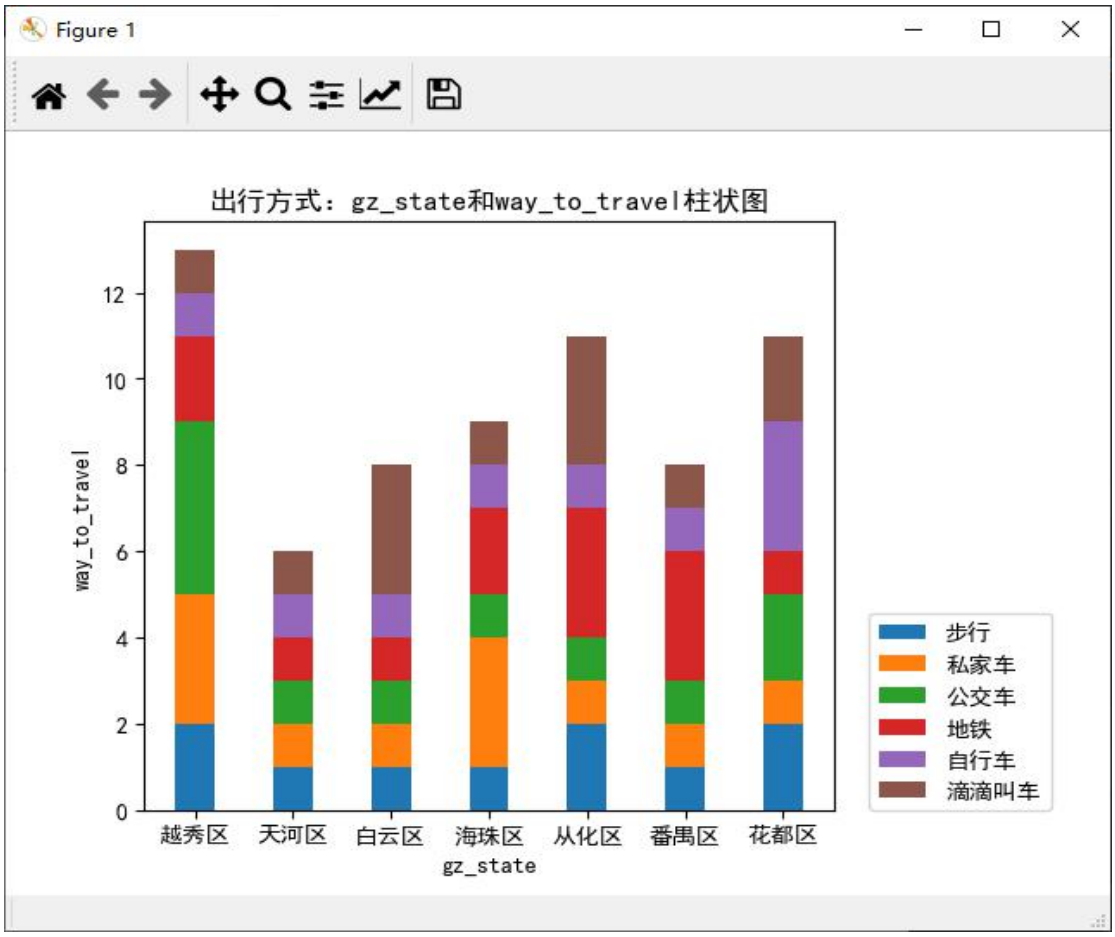
```
            i = i+1
        # 对 y 轴元素进行分组编号
        if v[field2] not in factors:
            factors[v[field2]] = j
            j = j+1
    # 建立二维交叉表
    len1 = len(groups)
    len2 = len(factors)
    for i in range(len1):
        crosstab.append([0]*len2)
    # 根据数据生成二维交叉表
    for v in result:
        crosstab[groups[v[field1]]][factors[v[field2]]] += 1
    # 生成概率二维表
    for i in range(len1):
        percrosstab.append([0]*len2)
    sum = []
    sum.clear()
    for i in range(len1):
        n = 0
        for j in range(len2):
            n += crosstab[i][j]
        sum.append(n)
    # print('sum')
    # print(sum)
    for i in range(len1):
        for j in range(len2):
            percrosstab[i][j] = crosstab[i][j] / sum[i]
# 存储全局变量
gl.set_value('groups', groups)
gl.set_value('factors', factors)
gl.set_value('len1', len1)
gl.set_value('len2', len2)
gl.set_value('crosstab', crosstab)
gl.set_value('percrosstab', percrosstab)
if len(crosstab) == len1:
    return True
else:
    return False
```

生成结果：

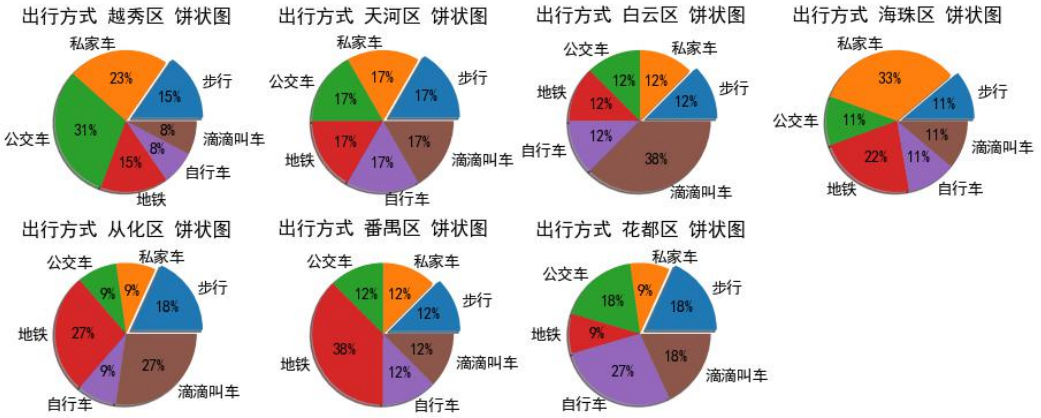
2.4、根据不同的选择进行统计图表的绘制

一共有三种图片可以绘制，柱状图，饼状图，折线图

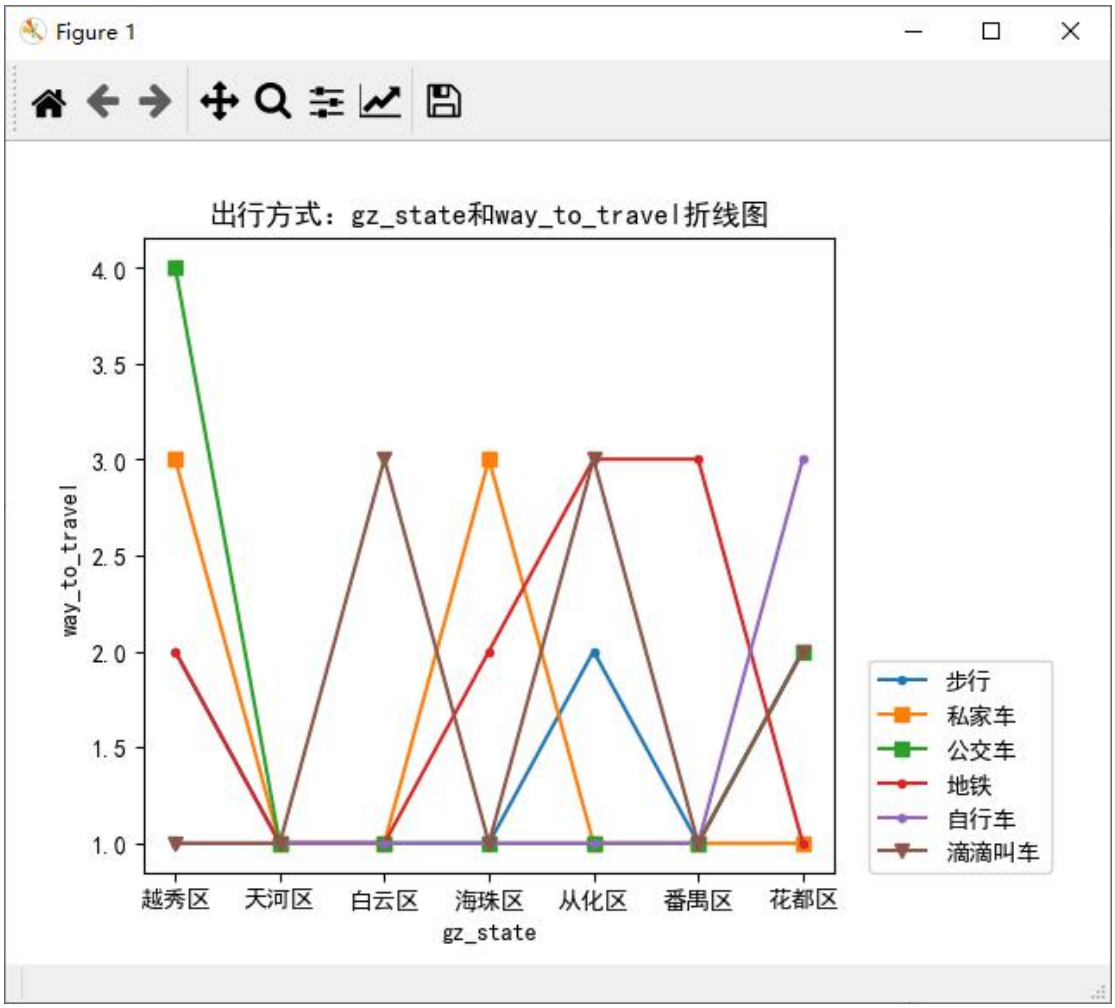
柱状图：



饼状图:



折线图:



2.5、关闭界面

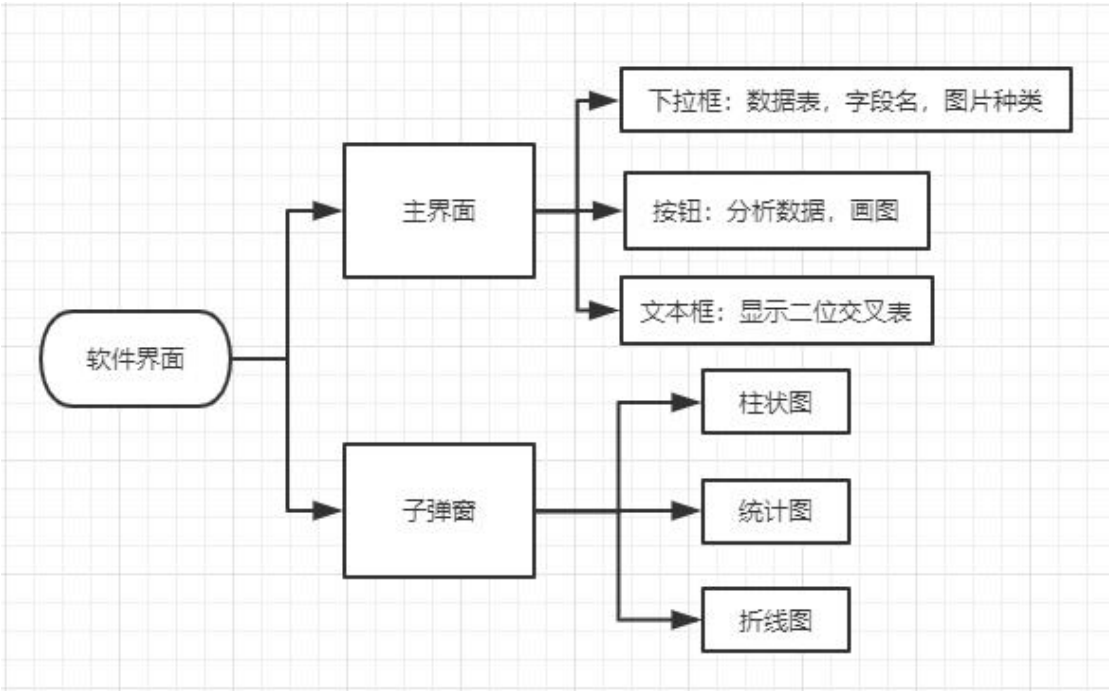
直接关闭软件即可。

二、 技术方案

1、功能组成

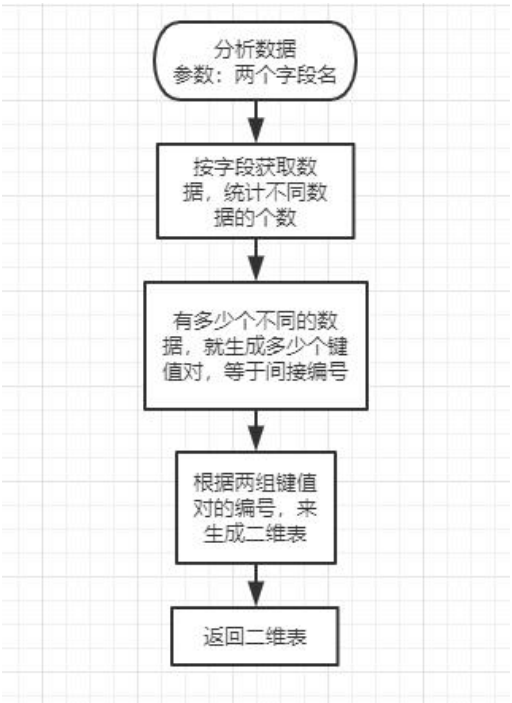
1.1、界面

界面包含一个主界面，三个子弹窗。

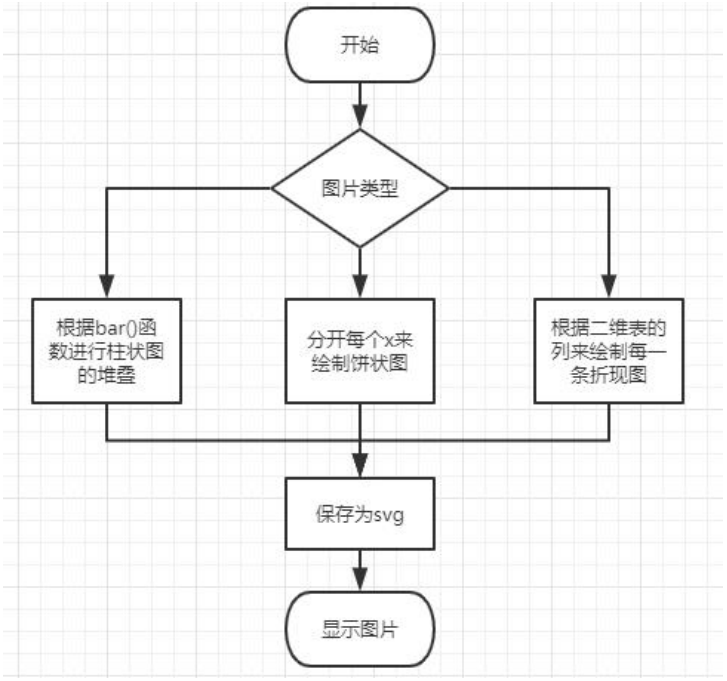


1.2、功能

分析数据

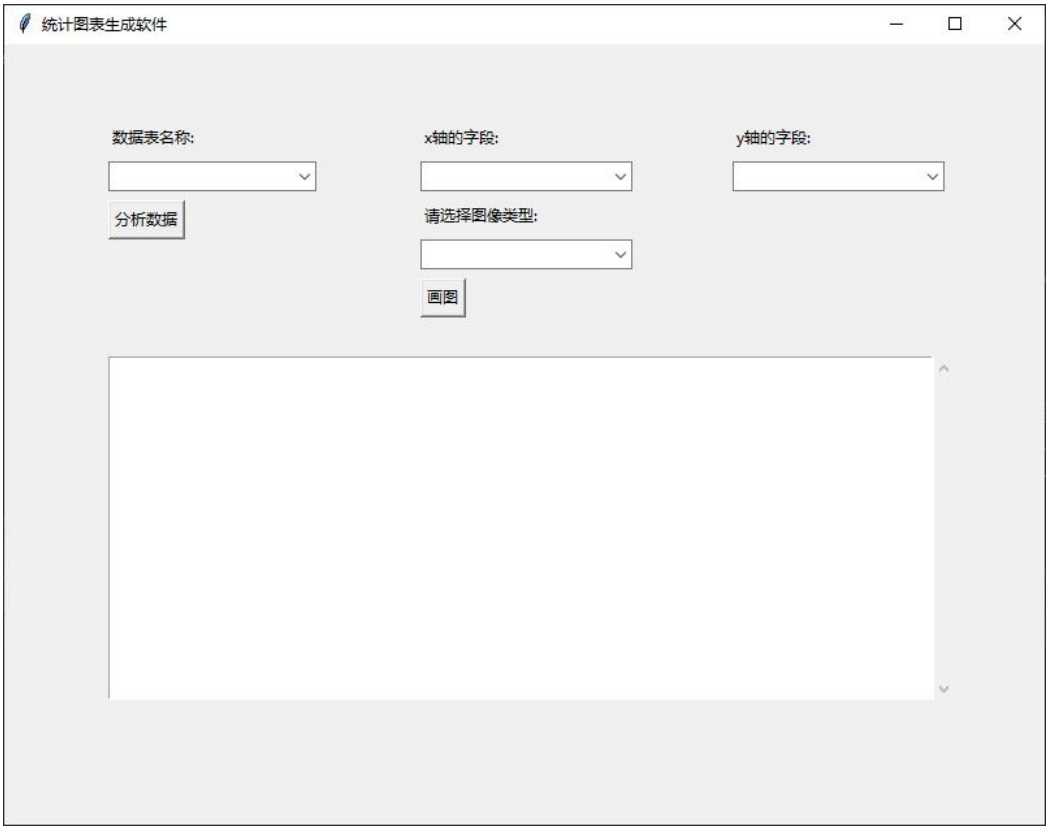


绘画统计图



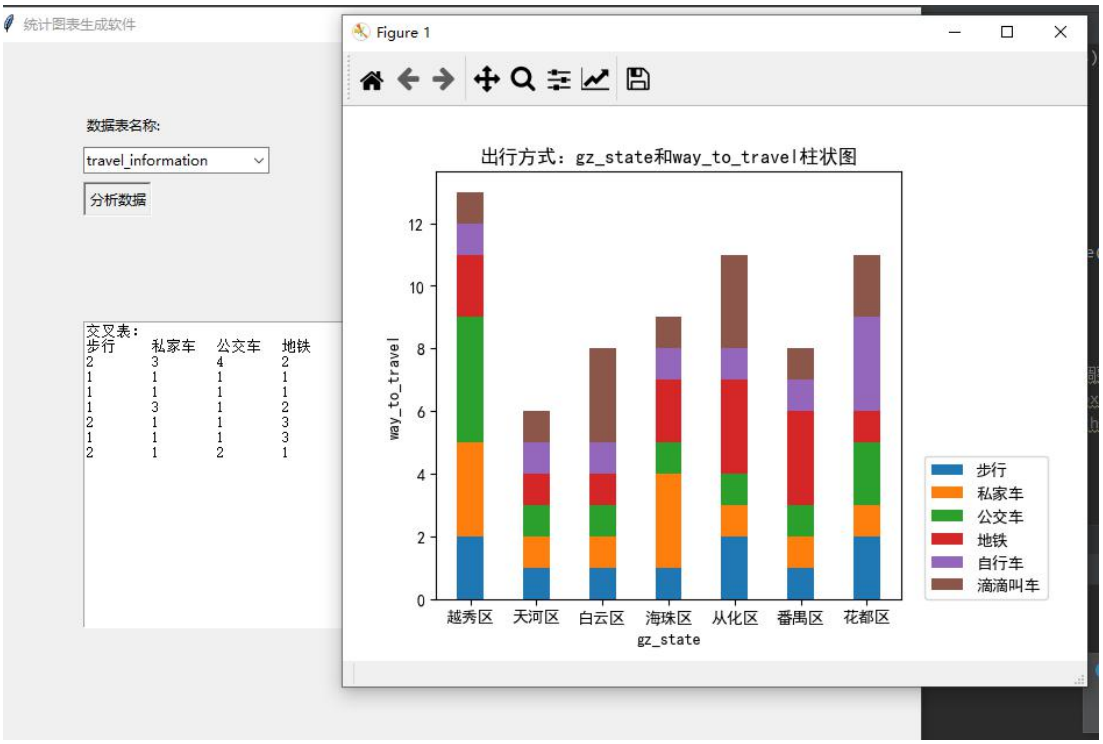
2、界面组成

2.1、主界面



主界面

2.2、绘图界面



绘图界面

3、 数据结构

1.1、

生成二维表的时候，使用了列表来进行二维表的存储，简单来说结构有点类似[[……],[……],[……]……,[……]]。

1.2、

在读入数据库和进行数据分析的过程中，都使用了键值对来存储数据并进行运算。例如选择了 gz_state 和 way_to_travel 后，生成存储 x 轴和 y 轴的两个字典，键对应的值则是二维数组的下标

```
x轴变量:
{'越秀区': 0, '天河区': 1, '白云区': 2, '海珠区': 3, '从化区': 4, '番禺区': 5, '花都区': 6}
y轴变量:
{'步行': 0, '私家车': 1, '公交车': 2, '地铁': 3, '自行车': 4, '滴滴叫车': 5}
```

```
交叉表:
步行    私家车    公交车    地铁    自行车    滴滴叫车
2       3         4         2       1         1         越秀区
1       1         1         1       1         1         天河区
1       1         1         1       1         3         白云区
1       3         1         2       1         1         海珠区
2       1         1         3       1         3         从化区
1       1         1         3       1         1         番禺区
2       1         2         1       3         2         花都区
```

1.3、

全局变量均使用了键值对来存储，set_value()函数有两个参数，一个接受变量名(字符串)，另一个接受变量值。

三、 技术讨论

1、存在问题

1.1、

软件仅仅完成了初步的功能，分析生成最简单的二维交叉表，对于更复杂的数据，比如年龄等比较散且变化大的数据，要进行额外的特判进行分组，无法做到适应全部类似的数据种类。

1.2、

实现了基本的三种图片，但是图片的样式很简单。

1.3、

这个软件是根据数据表来进行分析操作的，功能明确，但暂时来说还没能做到全部适用，如果换一个数据表，换一种字段名命名方式，可能会有不同的结果。

1.4、

代码中部分函数存在代码冗余稍高的情况，不同的功能的实现有若干部分的代码是一样或者相似的。

2、改进方向

1.1、

改进分组算法，尝试识别出分布不规律的数值，改进分组的操作。

1.2、

美化图片，尝试添加数值标签，并且尝试绘画更多样的数据统计图。

1.3、

自己建立或者在网上寻找数据进行多数据表的测试，找出其中的规律，尝试抽象统一出数据模型，由此来适配更多不同的数据表。

1.4、

可以尝试合并有冗余代码的函数，如果的确需要分开成两个功能进行操作，可以考虑添加一个新的参数来进行函数代码运行的选择（即如果是 0 则运行此段代码，为 1 则运行另一段……）。