

本科毕业设计（论文）

**Python数据分析包在电池数据分析中的应用**

**Application of Python Data Analysis Package in Battery Data Analysis**

学 院： 电气工程学院

专 业： 电气工程及其自动化

学生姓名： 蒋俊成

学 号： 12291180

指导教师： 张彩萍

**北京交通大学**

2016年5月

学士论文版权使用授权书

本学士论文作者完全了解北京交通大学有关保留、使用学士论文的规定。特授权北京交通大学可以将学士论文的全部或部分内容编入有关数据库进行检索，提供阅览服务，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名： 指导教师签名：

签字日期： 年 月 日 签字日期： 年 月 日

题 目： Python数据分析包在电池数据分析中的应用

学 院： 电气工程学院 专 业： 电气工程及其自动化

学生姓名： 蒋俊成 学 号： 12291180

指导教师建议成绩：

评阅教师建议成绩：

答辩小组建议成绩：

成绩评定委员会意见：

最终成绩：

主管教学副院长或成绩评定委员会主席签字：

年 月 日

题目： Python数据分析包在电池数据分析中的应用

学院： 电气 指导教师姓名： 张彩萍 职称： 副教授

|  |  |  |  |
| --- | --- | --- | --- |
| 题目类型 | □工程设计 □理论研究 □软件开发 ■其他 | | |
| 题目来源 | ■导师科研项目 □工程实际 □导师自拟 □其他 | | |
| 完成形式 | □设计 ■论文（含设计） | | |
| 校企双导师 | □是 ■否 | 校外导师姓名 |  |
| 校外导师单位 |  | 校外导师职称 |  |
| 课题面向专业 | 电气工程及其自动化  电气信息工程 | | |
| 课题简介 | 理解并熟练使用Python数据分析语言，利用其中的三种主流数据分析处理库，Pandas、Numpy、Scipy来完成电池分析处理工作。  依托实际的电池数据，对大量数据进行批量自动化处理。  课题研究可实现电池数据的自动化和批量处理，显著提高海量数据的处理速度，为电池安全、高效管理提供技术支撑。 | | |
| 研究重难点 | （1）使用数据分析包建立一次大规模批量处理电池数据的流程；  （2）完成基于电池数据的特征参数自动化提取。 | | |

指导教师（签字）： 填写日期： 年 月 日

题 目： Python数据分析包在电池数据分析中的应用

学院： 电气工程学院 专业：电气工程及其自动化 学生姓名：蒋俊成 学号：12291180

|  |  |  |  |
| --- | --- | --- | --- |
| 文献综述：  能源短缺与环境污染已经成为全世界共同面对，亟待解决的热点问题，因此发展新能源是本世纪的重要课题。电能作为一种清洁无污染的二次能源，在发展新能源的探索中广受关注，而电能的储存一直是制约新能源发展的瓶颈之一，为了打破这一瓶颈对于发展新能源的限制，电池技术的开发具有重要的研究价值。在对电池性能的研究中，对电池所产生的数据的处理是十分关键的一环。由于实际应用中如电动汽车电池每天都会产生大量的数据，若由人工处理其工作重复而繁琐，十分不便，将会给研究人员进行后续数据处理分析带来很大的困难。  在国内对电池数据处理该课题上，哈尔滨工业大学的纪淼淼提出选用LabVIEW编程，完成对不同电动汽车电池数据类型的读取，实现对整车、单体和故障信号分析及显示等功能，以减少研究人员的工作量[1]。北京交通大学的王玮提出利用时间序列模型和人工神经网络的混合模型做数据处理，以及基于高斯过程模型的数据处理的研究方法[2]。四川理工学院的胡阳波等提出采用MINITAB软件对采集到的不同项目的电池数据进行合理的数据分析，从而得出不同方面的性能，针对数据中比较特殊的数据，提出合理的解决方案，从而保证电芯的品质要求[3]。  为满足对大量电池数据处理的需求，需要一种实用可靠的计算机工具的支持。  Python是一种面向对象、解释型计算机程序设计语言，由Guido van Rossum于1989年发明，其设计哲学是“优雅”、“明确”、“简单”。Python作为一种有强大功能的面向对象编程，其语言的突出特点为：使用优雅的语法以易于编程者书写与阅读，支持各种平台包括Windows，Linux，Macos，Unix等，可以轻松地使用C或者C++语言拓展其功能，支持多种数据类型如整型、浮点型、复数型，字符串类支持ASCII编码的字符串以及Uncode字符串以方便处理不同国家的语言文字，Python支持面向对线的编程，支持生成类以及子类，支持多重继承，Python为动态语言，即其各个变量并不绑定特定的类型，使得在编程时可以更加灵活地使用变量，不必先定义再使用；其自有内存回收装置，不必担心内存的泄露。拥有多种高级语言的特点，极大地方便了程序的开发与调试，一般情况下使用Python的开发程序原型速度是C++、Java语言的数十倍，高速的软件迭代速度也使Python成为了当下流行的编程语言[4]。  本课题采用Python数据分析语言，利用其中的三种主流数据分析处理库，Pandas、Numpy、Scipy来完成电池分析处理工作。依托实际的电池数据，使用数据分析包建立一次大规模批量处理电池数据的流程，完成基于电池数据的特征参数自动化提取，对大量数据进行批量自动化处理。课题研究可实现电池数据的自动化和批量处理，显著提高海量数据的处理速度，为电池安全、高效管理提供技术支持。 | | | |
| 研究方案：  Python语言作为一种解释型的编程语言，十分适合作为数据处理程序的编程语言。本课题采用Python数据分析语言，利用其中的三种主流数据分析处理库，Pandas、Numpy、Scipy编写软件，来完成电池自动分析处理工作。  首先学习使用Python语言的基本功能和高级特性，及其各类数据分析处理库的使用方法。并在前期用Python语言编写一个简易的数据处理软件。然后进入本课题的主要内容，编写程序处理电池数据：  1、分析数据文件组织形式。将分析文件、文件名称在文件系统中的性质得到所有欲分析文件的路径列表，为下一步数据分析提供前提条件。  2、对数据格式进行转化与清洗。在此步骤中，将会对所有列表中的文件进行读取，最终得到分析所需的全部基础数据。并根据数据内部其他逻辑对各行数据进行清洗。  3、对数据进行加工与可视化。该步骤中将对已经表格化且已剔除不规则值的数据进行需要的运算处理，并利用整理完毕的数据进行简单的统计分析。使用DataFrame的describe方法可以获得单个变量的各种统计信息，如最小值，最大值，中位数，平均值等信息，根据这些信息可以简单的了解各个变量自身的性质。整理出的数据进行配合，可以根据需要绘制出时序图，散点图，柱状分布图等，从而更加直观的分析各个变量的变化趋势，各个变量间的相关关系。以便研究人员对结果的阅读与分析。  预期达到对实际电池数据的特征参数自动化提取，使用数据分析包一次大规模批量处理电池数据，对大量数据进行批量自动化处理，并得到可视化的输出结果。 | | | |
| 主要参考文献：  [1]纪淼淼.纯电动汽车电池管理数据分析的研究[D].哈尔滨:哈尔滨工业大学，2014.6-27.  [2]王纬.基于数据驱动的锂电池数据处理[D].北京:北京交通大学，2013:7-1.  [3]胡阳波.数据分析在锂电池品质中的应用[J].电子设计工程，2014年第22卷第18期：32-35 .  [4]龚敏明. Pandas在电池数据分析中的应用.北京:北京交通大学，2015  [5]邱慧.基于电动汽车的电池数据记录仪的设计[D].北京:北京交通大学，2010.4-9.  [6]熊瑞.基于数据模型融合的电动车辆动力电池组状态估计研究[D].北京:北京理工大学，2014.6  [7]陈峭岩.电动汽车电池状态估计及均衡管理研究[D].天津:天津大学，2013.11  [8]刘小诗.动力电池测试平台数据采集系统设计[D].北京:北京交通大学，2014.3-28  [9] Wes McKinney. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Ipython[M]. California:O'Reilly Media,2012 | | | |
| 毕业设计（论文）进度安排： | | | |
| 序号 | 毕业设计（论文）各阶段内容 | 时间安排 | 备注 |
| 1 | 收集文献资料，学习Python及其数据库的使用 | 开题至2016.3.25 |  |
| 2 | 编写简易的Python数据处理软件 | 至2016.4.15 |  |
| 3 | 编写Python软件进行电池数据处理 | 至2016.5.10 |  |
| 4 | 做后期调整改进，撰写论文 | 至2016.5.20 |  |
| 5 | 答辩 |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| 指导教师意见： | | | |

指导教师（审核签名）： 审核日期： 年 月 日

题 目： Python数据分析包在电池数据分析中的应用

学院：电气工程学院 专业：电气工程及其自动化 学生姓名：蒋俊成 学号：12291180

|  |
| --- |
| 毕业设计（论文）完成情况和成果质量（工作量、任务难度、专业理论的运用、综合运用能力、资料的充足与可信情况、成果水平）评价意见（80分）： |
| 答辩表现评价意见（20分）： |

答辩总成绩： 答辩委员会组长： 日期： 年 月 日

中文摘要

**摘要：**在新能源电动汽车的研究中，对于电动汽车的电池数据进行处理，分析其运行状态及出现的问题具有重要的意义。由于电动汽车每天产生的电池数据量十分庞大，给研究人员的人工处理带来困难。为满足对大量电池数据处理的需求，需要一种实用可靠的计算机工具的支持。本文致力于使用Python语言自动化批量处理电池数据，建立对电池数据处理分析的一般性方法，具有高效率，易修改，兼容性好，可跨平台等优点，为电动汽车研究人员的后续研究提供技术支持。本文的主要工作有：

**建立Python数据平台。**Python作为一种解释型的语言，需要搭建相应的编程环境进行开发。安装Python众多的扩展包，诸如科学计算包Numpy、Pandas、Scipy，数据绘图包Matplotlib等，可以使程序进行数据读写，科学计算，绘图等工作，同时优化程序提高处理效率。安装使用Anaconda解释可一次性包含所有Python开发常用扩展包，并避免兼容性问题的出现。使用Eclipse+PyDev的组合作为开发时所用的IDE可有效提高开发效率，增加程序的可读性及可修改性。安装Eric+PyQt的组合作为后期GUI编程工具。安装PyInstaller作为程序打包工具。

**使用Python语言对电池数据进行分析处理。**利用Python语言对单个或多个数据文件进行读取，对数据进行提取，格式转化，排除坏值等处理。提取出指定的特征参量，如总电流、总电压、单个电池电压、温度等数据的最大值、最小值、均方差、极差等，并输出特定量如最大温度值所对应的电池序号、出现时间等情况。利用绘制散点图的方法进行相关性分析，对电池组进行一致性分析，找出与其他电池表现不一致的电池序号。分析电动汽车的驾驶行为。分析电池温场等。

**搭建交互界面，并将程序打包。**使用Eric及PyQt工具组合可以开发出基于Python语言的GUI用户交互界面，使用户面对一个简洁直观的软件界面而不是一堆代码进行数据处理。Python作为一种解释型的脚本语言，本身依赖于系统的命令控制台，不能单独作为一个程序运行；并且由于解释型语言的特点，执行代码要求解释器的存在，依赖于编程环境，不具有可移植性。利用PyInstaller工具可以对Python程序进行打包，将Python语言生成的py文件编译为一个Windows系统可执行exe的文件，脱离对编程环境的依赖，使其可跨平台使用。

**关键词：**Python；电池；数据分析；科学计算；电动汽车

ABSTRACT

**ABSTRACT:** In the research of new energy electric vehicle, it has important significance to analyze the running status and the problems of the battery in the method of analyzing battery data of the electric vehicle. Because of the huge amount of data generated by the electric vehicle every day, the artificial treatment of the researchers is difficult. In order to meet the needs of a large number of battery data processing, a practical and reliable support for computer tools is needed. This paper aims to use the Python programing language to automate the batch processing battery data, a general method for analyzing and processing the data battery, with high efficiency, easy to modify, good compatibility, cross platform etc. advantage, provide technical support for the researcher in study of electric vehicle research. The main work of this paper is:

**Establishes Python-Based data analysis platform.** Because Python is an Explanatory language, programers need to build the corresponding programming environment for development. Install Python numerous expansion packs, such as scientific computing package numpy, SciPy and pandas, data drawing package Matplotlib can make the program to read and write data, scientific computing, and drawing, and optimize the program to improve processing efficiency. Installation and use of Anaconda to explain the use of a one-time extension of all Python development package, and to avoid the emergence of compatibility issues. Using the combination of Eclipse+PyDev as the development of the IDE can effectively improve the development efficiency, increase the readability of the program and can be modified. Installation of Eric+PyQt combination as a late GUI programming tool. Install PyInstaller as a program packaging tool.

**Using Python programing language to analyze the battery data.** Using Python language to read a single or a number of data files, data extraction, format conversion, rule out the bad values, etc.. Extract specified characteristic parameters, such as the total current, total voltage, single cell battery voltage and temperature data of the maximum value, minimum value, mean square deviation, and poor and output a specific amount such as maximum temperature value corresponding to the cell number and time. By using the method of drawing scatter plot, the correlation analysis is carried out to analyze the consistency of the battery pack. Analysis of driving behavior of electric vehicle. Analysis of battery temperature field, etc.

**Set up the GUI, and package the program.** Python is an explanatory scripting language, itself relies on the system console commands, not as a separate a program to run; and because of the characteristics of language interpretation, the implementation of the code interpreter, dependent on the programming environment, does not have the portability. Using PyInstaller tools can be of a python program packaged Python language generation py files are compiled into a Windows executable EXE file, away from dependence on the programming environment, the use of cross platform.

**KEYWORDS：**Python; Battery; Data Analysis; Scientific Computing; Electric Vehicle

目 录

[中文摘要 i](#_Toc451802558)

[ABSTRACT ii](#_Toc451802559)

[目 录 iv](#_Toc451802560)

[1 引言 1](#_Toc451802561)

[1.1 研究背景 1](#_Toc451802562)

[1.2 研究意义 1](#_Toc451802563)

[1.3 研究现状 2](#_Toc451802564)

[1.4 研究方法 2](#_Toc451802565)

[2 Python语言 5](#_Toc451802566)

[2.1 Python语言简介 5](#_Toc451802567)

[2.2 为何使用Python进行数据分析 5](#_Toc451802568)

[2.2.1 Python相比于Matlab等软件的优势 6](#_Toc451802569)

[2.2.2 Python相比于Java、C语言等编程语言的优势 6](#_Toc451802570)

[2.2.3 Python语言的不足 7](#_Toc451802571)

[2.3 Python数据平台及开发环境搭建 7](#_Toc451802572)

[2.3.1 Python版本选择 7](#_Toc451802573)

[2.3.2 Python数据分析包 8](#_Toc451802574)

[2.3.3 Python科学计算环境Anaconda 9](#_Toc451802575)

[2.3.4 Eclipse+PyDev搭建Python开发环境 9](#_Toc451802576)

[2.3.5 Eric+PyQt搭建GUI开发环境 9](#_Toc451802577)

[3 电池数据分析 11](#_Toc451802578)

[3.1 数据的提取、清洗与转化 11](#_Toc451802579)

[3.2 数据去坏值、去零值 13](#_Toc451802580)

[3.3 特征参量提取 14](#_Toc451802581)

[3.3.1 最值（以温度为例） 14](#_Toc451802582)

[3.3.2 均方差、极差（以总电流为例） 16](#_Toc451802583)

[3.3.3 数据压缩（以能量消耗为例） 16](#_Toc451802584)

[3.4 相关性分析 17](#_Toc451802585)

[3.4.1 单个文件数据相关性分析 19](#_Toc451802586)

[3.4.2 结合特征参量提取的多文件数据相关性分析 20](#_Toc451802587)

[3.5 电池组电压一致性分析 22](#_Toc451802588)

[3.6 驾驶行为分析 26](#_Toc451802589)

[3.6.1 停车时间 26](#_Toc451802590)

[3.6.2 驾驶路径 27](#_Toc451802591)

[3.7 温场分析 28](#_Toc451802592)

[4 GUI设计 29](#_Toc451802593)

[4.1 交互界面布置 29](#_Toc451802594)

[4.2 程序打包 32](#_Toc451802595)

[5 经济性分析 33](#_Toc451802596)

[6 结论 35](#_Toc451802597)

[参考文献 36](#_Toc451802598)

[致 谢 37](#_Toc451802599)

[附 录 38](#_Toc451802600)

[附录1 文献翻译 38](#_Toc451802601)

[原文 38](#_Toc451802602)

[译文 47](#_Toc451802603)

[附录2 程序代码 55](#_Toc451802604)

[自定义函数集 55](#_Toc451802605)

[特征参量提取 58](#_Toc451802606)

[相关性分析 61](#_Toc451802607)

[一致性分析 65](#_Toc451802608)

[温场分析 67](#_Toc451802609)

[驾驶行为分析 69](#_Toc451802610)

1. 引言
   1. 研究背景

能源危机与环境污染已经是当今世界面临的首要问题。根据世界能源统计年鉴2015，世界探明石油可采储量1万亿桶，共2358亿吨，探明天然气可采储量187.3万亿立方米，煤炭探明可采储量8609亿吨[1]。根据化石能源当前的储采比，即年尾剩余已探明资源余量除以当年的资源采集速度折算，石油约可采集50年，天然气可采集约55年，煤炭可采集约109年，而且这还是按照当前的采集率折算的，随着开采技术的发展和能源需求的进一步增加，化石能源真正枯竭的那一天可能会比预期提前到来，更重要的是，在化石能源枯竭时，其价格会攀升到令人无法接受程度。另一方面，化石燃料的燃烧是温室效应的罪魁祸首，也是大气污染的主要污染源。传统机动车耗油约占全世界石油消耗总量的75%，作为化石燃料的主要消费者，在能源危机与环境污染的问题中负重要责任。因此，开发以电能为主要能源的新能源汽车已经成为了世界各国的共识，新能源汽车也被我国确定为7大战略性新兴产业之一。

电动汽车的三大关键技术为电机、电控和电池，其中电池技术是制约电动汽车发展的最大瓶颈[2]。电动汽车于1837年由Davidon发明，比内燃汽车还早十余年，但与随后内燃汽车的蓬勃发展形成鲜明对比的是，电动汽车在一百余年间几乎无人问津，其最重要的原因就是电能无法大量储存这一技术瓶颈。随着近年来大容量锂电池的发展与能源、环境问题所带来的需求，电动汽车再一次进入人们的视野。美国、日本、德国等传统汽车制造业大国都启动了对大容量动力电池的研究攻关，意图开发出满足电动汽车常规使用需求的动力电池。我国在该项研究上亦投入了大量的精力与资源。

* 1. 研究意义

在对电动汽车电池研究的过程中，对每天产生的电池数据进行处理具有重要的意义。其可以帮助研究人员了解电池的运行状态，分析电气特性，评估使用效果，发现潜在问题。然而由于在数据采集过程中，传感器每隔数秒便会采集一组数据传入系统，一天便是几千组数据。而一组数据则是包含了电池组全体以及单个电池电流电压温度等几十个参量的数值。也就是说，电动汽车运行一天下来所产生的数据量十分庞大，使人工处理数据成为不可能，给研究人员的进一步研究带来困难。因此，建立一种利用计算机技术对电池数据进行自动化批处理的方法具有十分重要的意义。

* 1. 研究现状

在早期的电池数据中，常用Excel表格自带的统计功能以及宏命令对数据进行分析，例如最大值、最小值、均值、方差的提取，数据的重排及分类等，然而在面对复杂的数据类型以及对多变量数据的处理时Excel的功能便显得捉襟见肘。

在国内的该课题上，哈尔滨工业大学的纪淼淼提出选用LabVIEW编程，完成对不同电动汽车电池数据类型的读取，实现对整车、单体和故障信号分析及显示等功能，以减少研究人员的工作量[3]。北京交通大学的王玮提出利用时间序列模型和人工神经网络的混合模型做数据处理，以及基于高斯过程模型的数据处理的研究方法[4]。四川理工学院的胡阳波等提出采用MINITAB软件对采集到的不同项目的电池数据进行合理的数据分析，从而得出不同方面的性能，针对数据中比较特殊的数据，提出合理的解决方案，从而保证电芯的品质要求[5]。

* 1. 研究方法

为满足对大量电池数据处理的需求，需要一种实用可靠的计算机工具的支持。

本课题采用Python作为数据分析语言，利用其中的三种主流的数据分析处理库，Pandas、Numpy、Scipy来完成电池分析处理工作。利用绘图扩展包Matplotlib根据数据进行绘图可视化处理。依托实际的电池数据，使用数据分析包建立一次大规模批量处理电池数据的流程，完成基于电池数据的特征参数自动化提取，对大量数据进行批量自动化处理。课题研究可实现电池数据的自动化和批量处理，显著提高海量数据的处理速度，为电池安全、高效管理提供技术支持。

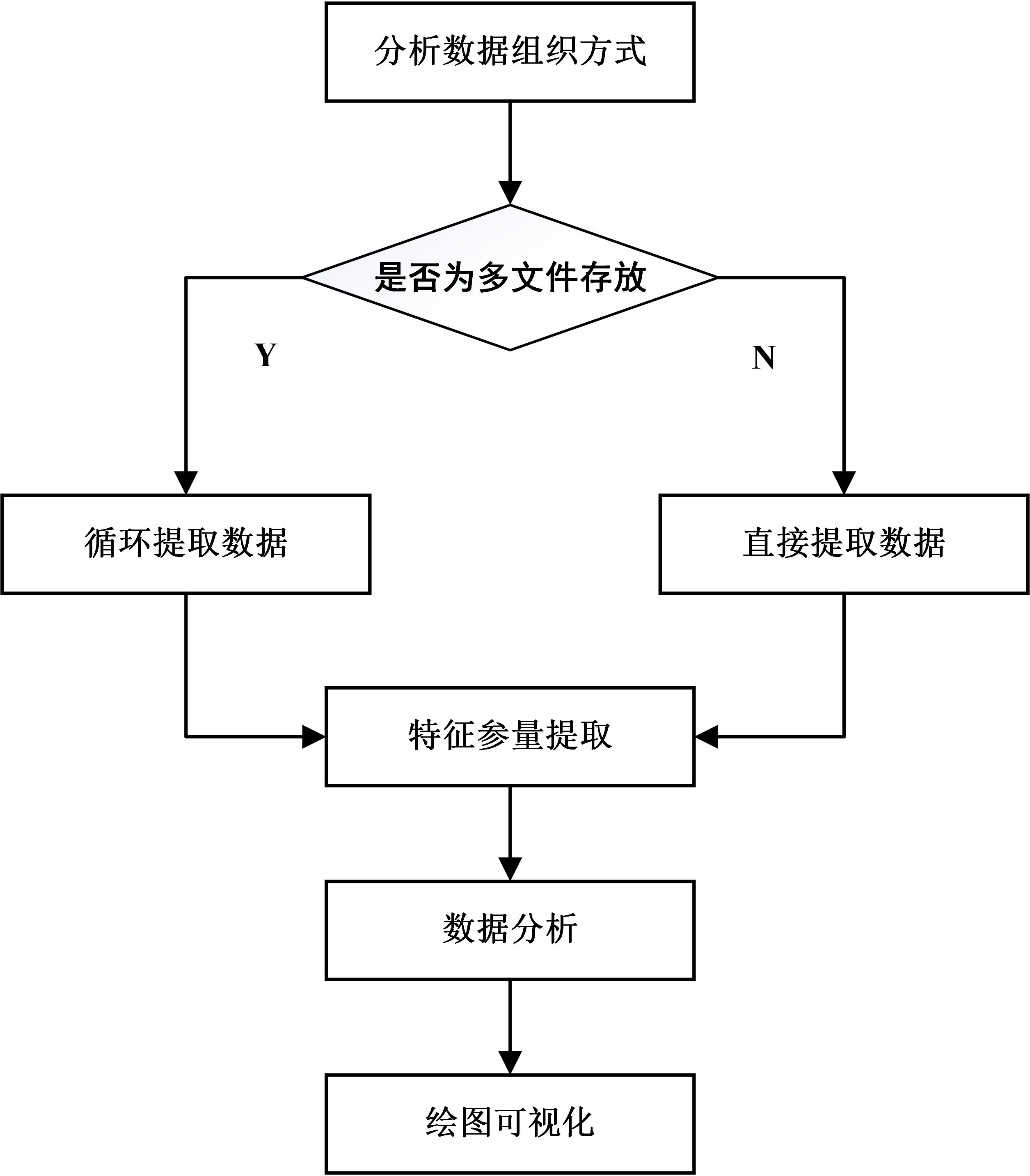


图 1 - 1

流程如图1 – 1，具体操作是，首先分析数据文件的组织形式，掌握其文件格式，存放路径，命名特点，重要的是掌握数据在文件内部的组织框架，以便有针对性地制定数据提取方案。

接着从文件中提取数据，要了解分析数据处理任务是对单个文件的数据进行分析还是处理全部文件的数据，若是处理单个文件，则直接提取其中的数据；若是要求处理全部文件中的数据，则调用OS模块中文件遍历的方法打开每个文件，并且制定好数据存放的框架，如Pandas模块中的DataFrame对象，将提取出的数据存放在其中。根据要求判断是否需要进行排除与其他数值偏差过大的去坏值操作，以及排除数值全部为0的去零值操作。

然后根据要求进行相关数据的特征参量提取，注意分析其格式，组织形式，数据类型，将数据提取压缩，使其能反应对应变量的参数特点。在此基础上可以进行更进一步的数据处理操作，比如变量相关性分析和数据一致性分析等。

最后对数据处理结果进行绘图可视化操作，使其能够更简洁明了地反应数据的特点，让研究人员能一目了然地了解电池的运行情况。

在进行完数据分析过后，可以根据以往的经验判断此类分析方法是否常用。如果是常用的数据分析流程，可以考虑将程序做GUI设计并封装成一个可脱离Python解释器使用的Windows exe文件，方便以后反复使用。

1. Python语言
   1. Python语言简介

Python是一种解释型的、面向对象的计算机编程脚本语言，由Guido van Rossum于1989年发明，其设计哲学是“优雅”、“明确”、“简单”[6]。Python作为一种拥有强大功能的面向对象的程序设计语言，其语言的特点有：语法简明优雅，使程序编写者的代码编写与源码阅读更加方便容易，支持包括Windows，Linux，Unix，Mac在内的各种系统平台，可以利用C语言或者C++实现功能的拓展，支持包括整型、浮点型、复数型在内的多种数据类型， 使用ASCII编码的字符串以及Uncode字符串支持字符串类，以方便处理不同国家的语言文字，使用缩进作为代码的分隔符来定义语句，使程序代码清晰可读，支持面向对象的编程，支持生成类以及子类，支持多重继承，支持异常处理，极大地降低了检查错误时的工作量和复杂度[7]。

Python是一种动态语言，也就是它的各个变量并不与特定的数据类型相绑定，使得在编程时不必先定义变量再使用，使变量的使用更加灵活，以应对不同情况。Python语言自带内存回收功能，避免了内存泄露的发生。同时具有多种高级编程语言的特点，极大地方便了程序的开发与调试，在一般情况下，使用Python语言程序开发所需要的工作时间仅为C++、Java语言开发时间的几十分之一，软件迭代速度快，版本更新与扩展包发布速度显示出Python的旺盛的发展活力与极高的受欢迎程度。Python具有丰富和强大的库。它常被人称为胶水语言，能够把用其他语言制作的各种模块（尤其是C/C++）很轻松地联结在一起[8]。

由于Python语言的简明、易用以及高可扩展性，在国外用Python做科学计算的研究机构日益增多，一些知名大学已经采用Python教授程序设计课程[9]。oogle的三大编程语言之一，集成于Linux以及MacOs系统之中。Python已经成为最受欢迎的程序设计语言之一，它被TIOBE编程语言排行榜评为2010年度语言，最新的编程语言排行榜TIOBE\_2016显示Python已经成为全世界使用率排第五的编程语言，仅次于JAVA、C、C++和C#。

* 1. 为何使用Python进行数据分析

在通过编程进行自动化数据处理这一领域，Python语言有着其他工具及编程语言所不具有的优势，是进行多类型大量数据处理的最优选择。

* + 1. Python相比于Matlab等软件的优势

Matlab是美国MathWorks公司出品的商业数学软件，主要用于数值计算、算法开发、数据分析及数据可视化的高级技术计算语言和交互式环境[10]。是计算机数学工具中的佼佼者。

然而在科学计算与数据分析领域，Matlab相比于Python有不少的不足之处。首先，Matlab作为一款商业软件，获得正版授权，价格不菲。最便宜的学生版，核心组件单个授权要99美元，想使用额外工具箱，则是每个工具箱29美元。而Matlab中国大陆的标准版售价为一万五千元人民币，给实验室带来额外的研究开支。其次是文件操作的能力，Matlab对于单个文件科学计算的表现无可置疑，然而实际数据处理中还有文件操作，界面设计等任务，Matlab对于多文件读写与提取的功能很弱，难以胜任大量文件的读取与数据分析工作。还有就是版权问题，虽然Mathworks论坛活跃着很多用户，也有很多有价值的代码，但是，版权归Mathworks公司，要想使用必须获得它的授权。

相比之下Python具有不输于Matlab的科学计算能力，但其使用完全免费，开源代码不存在版权问题，而且有强大的多文件读写提取操作的能力，这些优势让我们有理由放弃Matlab。

* + 1. Python相比于Java、C语言等编程语言的优势

Java、C语言、是当今最流行的编程语言，功能强大，使用广泛，几乎可以实现所有的用户需求的功能。

首先，Python是一种相当高级的语言，也就是高度封装了的编程语言，更加接近人的日常文字语言，易读易写。高级语言的另一个特性就是代码量极少，例如完成同样一段代码，用C语言写需要1000行代码，用 Java写需要100行代码，用Python写可能只需要20行代码。也就是说使用Python语言进行程序开发的效率远远高于Java、C语言。

其次，常见的Java、C等编程语言都是静态语言，而Python是动态语言。所谓静态语言就是在程序编写过程中需要先给变量确定类型，一旦确定不可改变，同时给变量分配一个确定的存储位置；而动态语言的变量类型不需要事先规定，由解释器在运行过程中自动获取，使用过程中可以改变类型，而且变量空间用完即释放，不用考虑存储问题。

因此Python语言具有远高于其他语言的开发编写效率，更高的可读性，更好的可移植性，通过改写已有代码来实现类似功能也更加容易，是最适合做数据处理的编程语言。

* + 1. Python语言的不足

Python语言最大的缺点就是运行速度慢，因为Python是解释型语言，代码在执行时会一行一行地翻译成CPU机器码，这个翻译过程耗时较长，拖慢程序运行速度。而编译型语言是运行前直接编译成CPU能执行的机器码，所以运行速度快。对此可以直观地解释为程序的运行速度与开发速度成反比，编写代码最耗时的C程序运行速度最快，而编写代码最快的Python语言运行速度最慢，同样功能的一段代码，C语言运行时间也许只需要0.001秒，而Python语言需要1秒才能运行完毕。

然而在做数据处理时并不需要程序运行速度有多快，由于每次数据处理都需要修改或重写代码，一段已写成的代码并不会反复使用许多次，并且程序需要消耗几秒的时间对于研究人员来说并不重要，因此在做数据处理时程序编写速度直接影响工作效率，而程序运行速度则影响不大，而此需求正符合Python语言的特点。

* 1. Python数据平台及开发环境搭建

在当前计算机最常用的Windows系统中并没有集成Python开发环境，因此需要手动配置Python的编程环境，由于近年Python语言的快速发展，其开发环境的语言版本、编译器各不相同，IDE、GUI工具众多，扩展包的数量也十分庞大，为了遵循程序开发的开发环境简单化原则，即Easy模式，我们需要搭建一套最优化的数据平台及开发环境。

* + 1. Python版本选择

Python语言分为Python 2 . x 和Python 3 . x 两个版本。由于Python语言的发展是由社区支持的，在发展的过程中出现了断层的现象，造成了Python 3 . x并不向下兼容Python 2 . x的问题。由于Python 2 . x发展较早，版本比较稳定，大部分的工具在2 . x版本下运行都比较稳定，支持也比较好。虽然现在有很多工具已经转向3 . x支持，许多工具现在是2 . x也可以运行3. x也可以运行，但是现在还有相当一部分工具还没有转到3 . x上来，并且2 . x的版本稳定性也有待时间的考验。鉴于此原因，本课题主要使用Python 2 . x版本，具体版本号是Python 2 . 7 . 11。

* + 1. Python数据分析包

由于Python语言的开源性特点，其社区发展过程中开发出了许多使用而强大的扩展包，其中一些数据分析包对于完成数据分析，提高工作效率有很大帮助，在电池数据处理中使用到的扩展包如下：

Numpy

Numpy是Python中的基础性科学计算扩展包，它提供了多维数组Narry这一重要的对象类型，各种复杂的矩阵操作、数值计算都基于对多维数组的各种操作，且对于各种数组的操作采取了相应的并行计算加速策略，包括数学计算、逻辑计算、数组维度改变、排序、IO操作、离散傅里叶变换等。与MATLAB相比，其具有更高的可拓展性、更大的灵活性、良好的多平台应用开发能力。由于其相当一部分多维数组操作/数值运算操作底层使用C语言实现，所以在计算速度上表现良好。

Scipy

Scipy是用于高级科学与工程计算的扩展包。SciPy依赖于NumPy，提供方便、快捷的n维数组操作。SciPy库与Numpy一同提供了很多人性化的，如优化数值积分程序的高效数值计算程序，易于使用，功能强大

Pandas

Pandas提供了相当多的基于结构化数据的操作，表格型数据是其主要的数据处理对象。Pandas集合了Numpy中的多维数组表现能力以及关系型数据库语言中的数据表达能力，同时提供了复杂的索引操作以方便对数据的维度改变、切割、合并以及子数据的选择。其中分组数据的操作对电池分析十分帮助，如对各个子数据组应用函数算法时，其分组功能可以方便的为每个分组生成相应输出值。

Ipython

Ipython为一个基于Python的交互命令行程序，支持变量自动补全，自动缩进，支持bash shell 命令，内置了许多很有用的功能和函数。比如其中的Notebook模块可以在运行Ipython的主机上启动一个http服务，将本机的数据处理环境开放给拥有相应权限的与该计算机联网的所有计算机。

利用Python自带的pip命令可以容易地安装Python扩展包。在Windows系统下在任何位置打开命令提示符，键入pip install package，package为扩展名名称，如Numpy，系统便会自动从Python扩展包官方网站PyPI（https:// pypi. python. org/ pypi）下载相应扩展包，并自动安装于Python安装目录下，省去手动下载以及选择安装位置等操作。

* + 1. Python科学计算环境Anaconda

Python语言提供了许多集成式的科学计算环境，将许多常用的工具与科学计算扩展包都集成于其中。用户只需要安装一个科学计算包就拥有了所有主流扩展包。这是由于手动安装Python扩展包是一件十分繁琐的工作，因为许多扩展包都有其依赖项，需要先安装依赖项才能运行，而不少依赖项又有自己的依赖项，使用pip命令安装工作量很大。安装一个科学计算环境可一次性解决这些问题。现在主流的科学计算环境有以下三种：

Pythonxy是一种集成式科学计算环境，内置了很多科学计算常用的扩展包，如Numpy，matbplotlib，spyder等，《用Python做科学计算》一书对此有详细介绍。其缺点是开发人员流失，更新速度慢。

Enthought Python Distrubution ( EDP ) 是一个商业的Python发行版本，包括了众多的科学计算包，作为教学使用是免费的。但在非教学开发时有复杂的版权问题，因此不建议使用。

Anaconda是由Python之父吉多·范罗苏姆作为核心成员开发并维护的科学计算环境。涵盖了Python 2 . x 和Python 3 . x 两个版本，覆盖Windows，Linux，Mac OS系统，包含大量科学计算包，方便安装第三方扩展包。更新速度快，完全开源免费，是最合适做数据处理的Python科学计算环境。

* + 1. Eclipse+PyDev搭建Python开发环境

使用一款实用高效的开发环境（IDE）可大大提高开发效率，降低开发难度。Eclipse是著名的跨平台的自由集成IDE。最初主要用来Java语言开发，通过安装不同的插件Eclipse可以支持不同的计算机语言，比如C++和Python等开发工具。Eclipse的本身只是一个框架平台，但是众多插件的支持使得Eclipse拥有其他功能相对固定的IDE软件很难具有的灵活性。许多软件开发商以Eclipse为框架开发自己的IDE。

PyDev便是用于将 Eclipse当作 Python IDE 的插件，在Eclipse的扩展选项中填入PyDev官方网站地址即可自动安装该插件于Eclipse之中。

* + 1. Eric+PyQt搭建GUI开发环境

Eric是一种Python和Ruby语言使用的IDE，具有变量补全，自动缩进，错误检查与运行分析等功能，完美支持Python的GUI开发工具PyQt的使用。

Python的GUI开发工具主要有Thinter、WxPython、Pygtk、PyQt、PySide、Kivy这几种。其中Thinter是Python内嵌的GUI环境，布局全靠代码实现，效果简陋；WxPython由C++编写，代码布局控件，不直观；Pygtk在Windows下兼容性有一定的问题；PySide效果与PyQt类似，但是也是由代码布局，不易开发；Kivy布局使用专用语言kivy language，GUI布局环境尚不成熟，中文支持差。

PyQt是Python对GUI设计程序QT的包装，源码为C++，空间丰富，函数\方法多，拖拽布局，开发简易，方便打包成二进制文件，因此是当前最实用的Python GUI开发工具。其中PyQt 4针对Python 2 . x 版本，PyQt 5针对Python 3 . x 版本。

1. 电池数据分析
   1. 数据的提取、清洗与转化

对数据的组织形式进行分析。电池数据一般存放于各种表格式文件之中，通常包括xls、xlsx、csv。若有多个数据文件，其命名通常具有相似性，可作为多文件读取的依据。Python语言自带有xlrd模块可读取Excel表，Pandas模块中也有一系列加载函数，可以读取各类表格文件并将其组织为Dataframe格式。

对数据进行清洗转化。由于数据通常被存储为带单位的字符串格式，分析数据单位所占字符，利用字符切片功能将单位切除，将数据转化为utf-8编码，再将其转化为对应的数据类型，如int、float等。

将数据组织为预期形式，一维数据可直接组成数组，多位数据可转化为Series、DataFrame格式数据。试举一例，流程与代码如下：

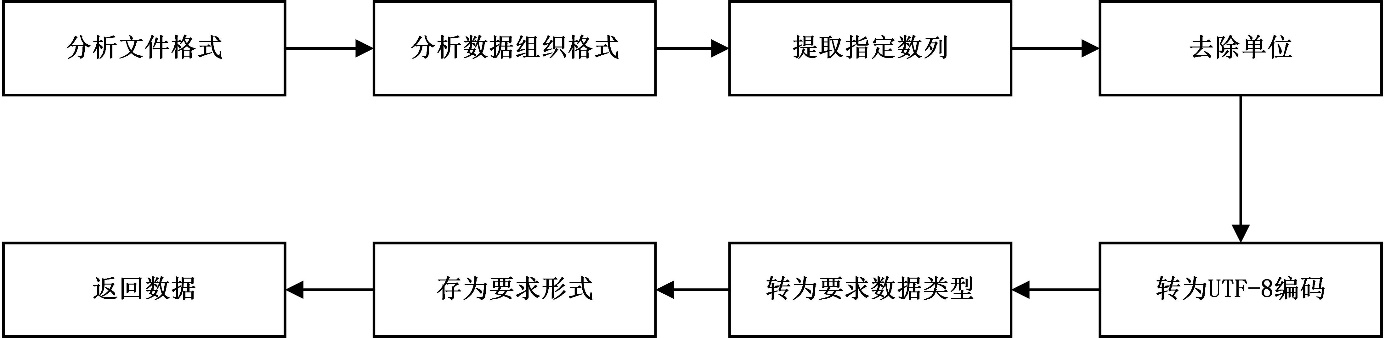


图 3–1

def readata (path,sheet,col,chop,str=0):

dt = [ ]

data = xlrd. open \_workbook (path)

table = data. sheet \_by \_index (sheet)

value = table. col \_values (col)

value. pop(0)

for val in value:

if str = = 1:

dt. append(val)

elif chop = = 0:

dt. append (float (val. encode ('utf-8')))

else:

dt. append (float (val [:-chop]. encode ('utf-8')))

return dt

这是一个将数据读取及格式清洗转换封装起来的函数，调用此函数，传入数据所在文件路径、所在表格、所在列、单位字符数，即可返回一个由float类型数据组成的数组；若不需要进行格式转换，则在第5个参数输入1，即可返回原格式数据组成的数组。

如需对多个文件进行分析，则利用OS模块提供的各种系统文件目录函数读取指定的路径即可。如图3 – 2，本例使用2015年4月2日某电动汽车公司120辆电动汽车的当天行使数据，所有数据文件都存放于一个文件夹中，据观察，这些文件命名方式具有一定的相似性。

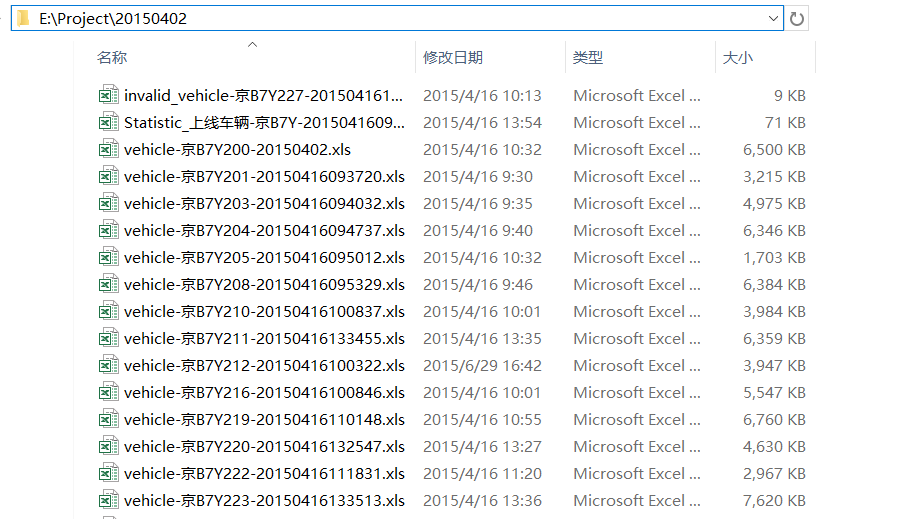


图 3 - 2

def getpath (dir):

path = [ ]

for file in os. listdir (dir):

if file [:7] = = 'vehicle':

path. append (os. path. join (dir,file))

return path

这是根据文件存储路劲以及文件命名特点制定的多文件读取函数，传入文件所在的目录，返回全部文件的路径，并将所有需要处理的文件路径添加到一个叫path的list之中，后续处理中，只需用循环操作轮流处理path当中每一个路径的文件，即可达到多文件处理的功能。

* 1. 数据去坏值、去零值

根据数据处理的要求，可以在数据分析前将数据坏值或无效的零值找出并去除。判定坏值的标准可在代码中制定。利用数组循环操作容易实现此功能。

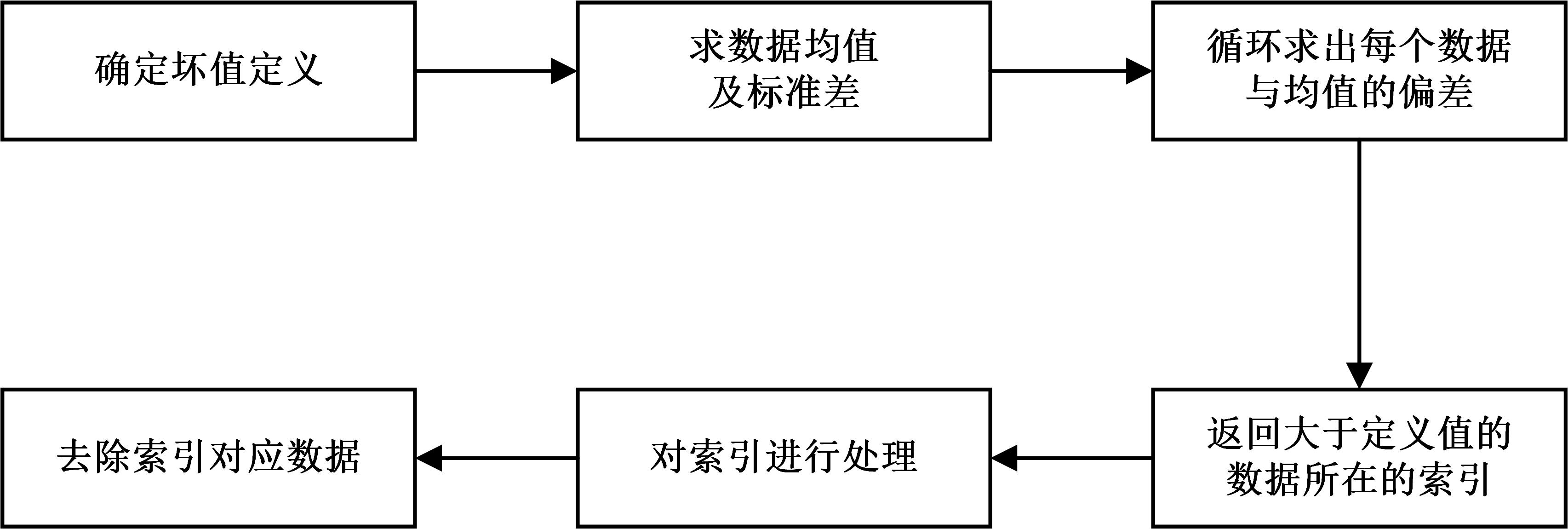


图 3 - 3

def badpop (lt): #去坏值

I = 0

p = [ ]

pop = [ ]

mean = np. mean (lt)

std = np. std (lt)

for x in lt:

if abs (mean-x) > std \* 10:

p. append (i)

I + = 1

for i in range (len (p)):

pop. append (p [i] - i)

return pop

def zeropop (lt): #去零值

I = 0

p = [ ]

pop = [ ]

for x in lt:

if x = = 0:

p. append (i)

I + = 1

for i in range (len (p)):

pop. append ( p [i]-i)

return pop

这里分别是去坏值与去零值的两个函数，传入需分析的数组，返回坏值/零值所在索引，此后只需利用该索引将数组中的坏值/零值弹出即可。所谓零值是指由于系统时间不同步、传感器故障等原因造成的在某些时间段上数据全是0的情况，应当作为无效数据排除。需要注意的是，并不是所有数值为0的数据都是无效了，在进行去零值操作前应当仔细分析数据的实际情况来确定要不要进行该项操作。在该函数中，对坏值的定义是数据与平均值之差的绝对值大于10倍标准差视为坏值。在实际应用中，该定义可根据实际情况修改或重制，或不进行坏值分析。

* 1. 特征参量提取

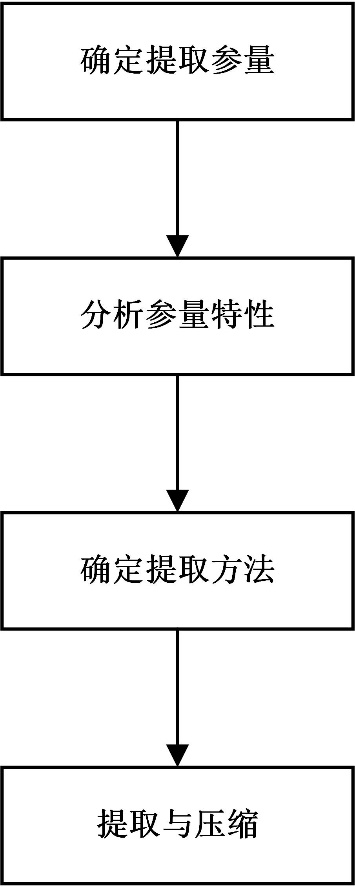
 特征参量是指能够反应电池特定参数的量，通常情况下会体现出对数据的压缩，为后续步骤的分析提供一些可规范化的量。流程如图3 - 4

图 3 - 4

这里试举几个特征参量提取的例子。

* + 1. 最值（以温度为例）

利用Numpy包中的统计函数可以很容易地找出数组中的最大最小值，调用上文中封装好的读取函数可以将文件读取为数组。

使用Numpy中的max函数即可提出去最大值。

path = u 'E: \\ project\\ 20150402\\ vehicle- 京B7Y204- 20150416094737. xls'

data = readata(path,0,12,2)

maxtmp = np. max (data)

print maxtmp

若要一次性读取所有文件的温度最大值，则使用递归的方法，将上述步骤在所有文件中使用一遍即可。

maxtmp = [ ]

dir = u 'E:\\ project\\ 20150402\\ '

for file in getpath (dir):

tmp = readata (file,0,12,2)

maxtmp. append (np. max (tmp))

print maxtmp

程序运行结果为：

[25.0, 20.0, 20.0, 24.0, 16.0, 19.0, 22.0, 18.0, 23.0, 24.0, 25.0, 22.0, 22.0, 23.0, 23.0, 19.0, 21.0, 19.0, 18.0, 22.0, 19.0, 23.0, 25.0, 23.0, 25.0, 24.0, 24.0, 22.0, 21.0, 21.0, 22.0, 22.0, 23.0, 25.0, 21.0, 21.0, 23.0, 19.0, 20.0, 24.0, 24.0, 20.0, 21.0, 23.0, 24.0, 19.0, 19.0, 19.0, 19.0, 20.0, 12.0, 19.0, 22.0, 23.0, 23.0, 20.0, 19.0, 21.0, 21.0, 20.0, 19.0, 19.0, 20.0, 22.0, 22.0, 11.0, 20.0, 27.0, 26.0, 22.0, 24.0, 18.0, 28.0, 21.0, 22.0, 22.0, 24.0, 18.0, 25.0, 25.0, 24.0, 20.0, 21.0, 21.0, 21.0, 20.0, 21.0, 21.0, 21.0, 21.0, 24.0, 20.0, 22.0, 19.0, 20.0, 23.0, 18.0, 23.0, 21.0, 20.0, 26.0, 23.0, 24.0, 25.0, 22.0, 24.0, 23.0, 21.0, 24.0, 26.0, 19.0, 19.0, 23.0, 20.0, 17.0, 21.0, 18.0, 20.0, 25.0, 25.0]

为了便于观看也可对其进行格式化输出处理。代码如下：

maxtmp = [ ]

dir = u 'E:\\ project\\ 20150402\\ '

for file in getpath (dir):

tmp = readata (file,0,12,2)

maxtmp. append (np. max (tmp))

for i in range (len (maxtmp)):

print u'%s 的电池最大温度值为%.0f'℃% (getpath(dir) [i] [28:25], maxtmp[i])

运行结果如下：

京B7 Y200 的电池最大温度值为25℃

京B7 Y201 的电池最大温度值为20℃

京B7 Y203 的电池最大温度值为20℃

京B7 Y204 的电池最大温度值为24℃

京B7 Y205 的电池最大温度值为16℃

京B7 Y208 的电池最大温度值为19℃

京B7 Y210 的电池最大温度值为22℃

京B7 Y211 的电池最大温度值为18℃

京B7 Y212 的电池最大温度值为23℃

京B7 Y216 的电池最大温度值为24℃

京B7 Y219 的电池最大温度值为25℃

京B7 Y220 的电池最大温度值为22℃

京B7 Y222 的电池最大温度值为22℃

……

* + 1. 均方差、极差（以总电流为例）

Numpy科学计算包中包含许多常用的数学计算，以分析总电流数据为例，用Numpy提取数据的均方差，极差。

path = u' E:\\ project\\ 20150402\\ vehicle- 京B7Y204- 20150416094737. xls'

I = readata (path,3,3)

print u' 总电流均方差为%. 2fA,极差为%. 2fA' % (np. std (I), max (I) – min (I))

运行结果显示：

总电流均方差为30.50A，极差为210.10A

* + 1. 数据压缩（以能量消耗为例）

特征参量提取的一大特点便是要体现对数据的压缩，将一个数组或者数据矩阵的特点进行总结并将众多数据转化成单一的值。我们利用提取电动汽车的能耗来体现这一特点。

path = u' E:\\ Project\\ 20150402\\ vehicle-京B7Y223 - 20150416133513. xls'

energy = readata (path,0,19,4)

zero = zeropop (energy)

for i in range (len (zero)):

energy. pop (zero [i])

energycost = 0

x = 0

m = [ ]

n = [ ]

l = [ ]

charge = 0

for i in range (len (energy) - 1):

if energy [i] < energy [i+1] and x = = 0:

m. append (i)

x = 1

if energy [i] > energy [i+1] and x = = 1:

n. append (i)

x = 0

if len (m) ! = len (n):

n. append (len (energy) - 1)

for i in range (len (m)):

if n [i] – m [i] > 20:

l. append ( [m [i], n [i]] )

for i in range (len (l)):

charge + = energy [l [i] [1]] – energy [ l [i] [0] ]

energycost = energy [1] – energy [ len (energy) - 1] + charge

print u' %s全日能耗为% .1fkW•h' % ( path [28:35], energycost )

运行结果为：

京B7 Y223全日能耗为28.5 kW • h

若需提取全体车辆的单日能耗，使用上文的文件遍历即可实现，此处不再累述。

* 1. 相关性分析

在有了对单个多个文件进行特称参量提取方法的基础上，我们可以进行更多的数据分析。流程如图3 – 2。

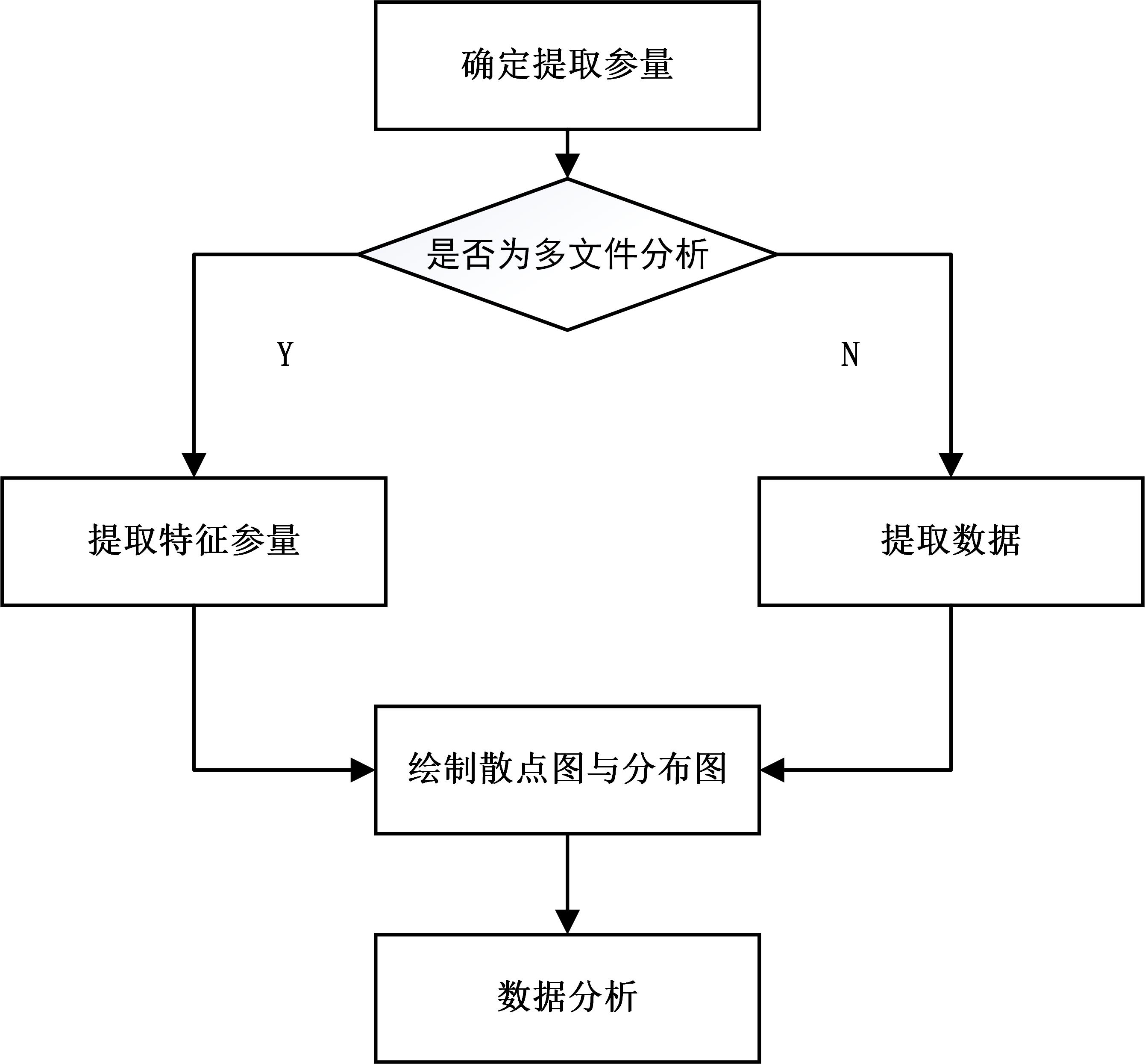


图 3 – 5

其中关键的绘图部分，借助Python中的数据绘图包Matplotlib的帮助，我们可以直观的进行数据变量之间的相关性分析。对于两组数据分别使其作为X轴可Y轴绘制散点图，通过其点分布观察两组变量是否具有相关性。对于多组数据可以使其两两一组绘制散点图阵，变量与自身的交叉初绘制变量自身的分布柱状图。定义一个函数correlation\_analysis来实现此功能，传入由需分析变量组成的数组，返回一个散点图阵。

def correlation \_analysis (datalist,labels):

n = len (labels)

fig,axes = plt. subplots (n,n,sharex = False,sharey = False)

fig. set \_size \_inches (16,10)

plt. subplots \_adjust (wspace = 0,hspace = 0)

for i in range (n):

for j in range (n):

axes [i,j]. set \_xticks ([ ])

axes [i,j]. set \_yticks ([ ])

if i = = 0:

axes [i,j]. set \_title (labels [j],fontsize = 'large')

if j = = 0:

axes [i,j]. set \_ylabel (labels [i],fontsize = 'large')

if I = = j:

axes [i,i]. hist (datalist [i],30,color = 'c',alpha = 0.6)

else:

axes [i,j]. scatter (datalist [j],datalist [i],marker = '.', s=5, color = 'r', alpha = 0.3)

* + 1. 单个文件数据相关性分析

下面取一辆电动汽车中的总电流、总电压、剩余电量（SOC）和行驶里程四个变量做相关性分析。

path = u 'E:\\ Project\\ 20150402\\ vehicle-京B7Y241 - 20150416133654. xls'

Vdata = readata (path,0,9,1)

Idata = readata (path,0,17,1)

SOC = readata (path,0,16,1)

mileage = readata (path,1,4,2)

datalist = [Vdata,Idata,SOC,mileage]

labels = ['V','I','SOC','mileage']

for i in range (4): #去坏值

bad = badpop (datalist [i])

for j in range (len (bad)):

datalist [0]. pop (bad [j])

datalist [1]. pop (bad [j])

datalist [2]. pop (bad [j])

datalist [3]. pop (bad [j])

correlation \_analysis (datalist,labels)

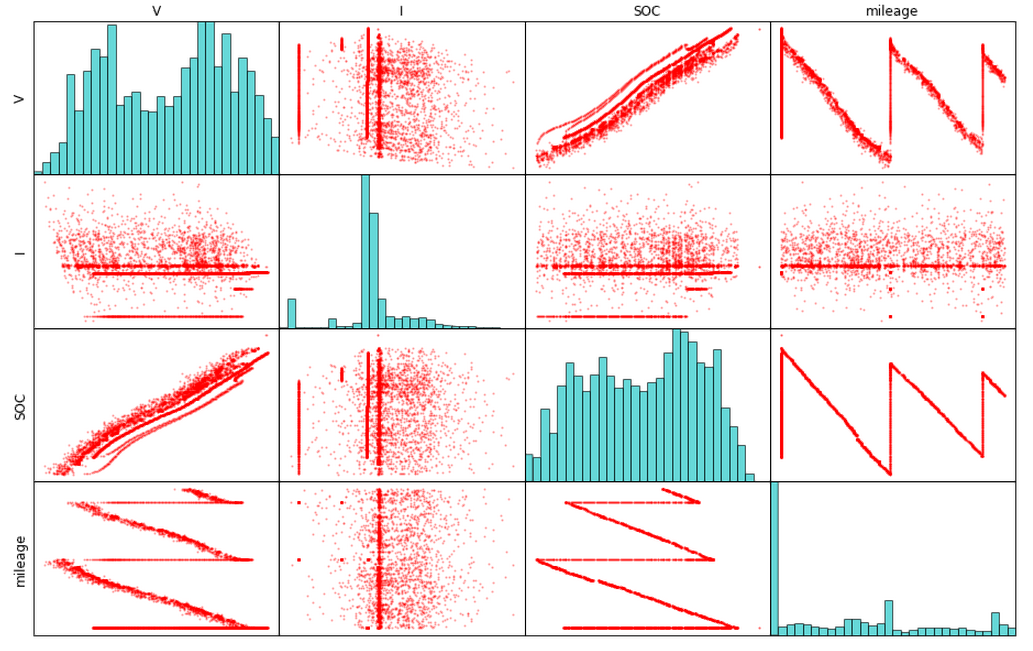
 绘图结果如下

图 3 - 6

图3 – 4中对角线上是各个变量的分布图，非对角线上是与其他变量组成的散点图。坐标轴上V为总电压，I为总电流，SOC为剩余电量，mileage为累计里程数。由此图可见，电池的总电压与剩余电量（SOC）具有较高的线性相关性，并且是正相关，意即电池剩余电量越充足，电池电压越高。同时SOC与行驶里程数也有很高的线性相关性，也就是里程越大剩余电量越低，图中直线部分，即SOC增加而里程不变的部分，分析可知是电池正在充电，所以通过此图也可看出电池一天的充电次数。从柱状分布图可看出总电流集中的分布在一个区间，而与其他量没有相关性，这是因为电池有大量电流为0的时刻。

* + 1. 结合特征参量提取的多文件数据相关性分析

上一个案例体现了对单个电池中的数据进行相关性分析的方法，结合上文提出的对单个文件进行特征参量提取的方法，我们可以进行对所有电池数据的特称参量进行相关性分析。

下面对全部电动汽车的各项参数进行分析。提取单量汽车全日能耗、电池组最大极差、行驶里程、电池组平均温度四个变量进行绘图分析。

allcost = [ ]

for file in path:

energy = readata (file,0,19,4)

zero = zeropop (energy)

for i in range (len (zero)):

energy. pop (zero [i])

encost = 0

for i in range (len (energy)-1):

if energy [i] > energy [i+1]:

encost + = (energy [i]-energy [i+1])

allcost. append (encost)

alltmp = [ ]

for file in path:

tmp = readata (file,0,12,2)

alltmp. append (sum (tmp) / len (tmp))

allrange = [ ]

for file in path:

vt = readlist (file,4,3)

rg = [ ]

for v in vt:

rg. append (max (v) - min (v))

allrange. append (max (rg))

allmile = [ ]

for file in path:

mile = readata (file,1,4,2)

zero = zeropop (mile)

for i in range (len (zero)):

mile. pop (zero [i])

cum = 0

for i in range (len (mile) -1):

if mile [i] < mile [i+1]:

cum + = (mile [i+1] – mile [i])

allmile. append (cum)

datalist = [allcost,allrange,alltmp,allmile]

labels = [ 'Energy \_cost' , 'Voltage \_range' , 'Temperature' , 'Mileage' ]

for i in range (4):

bad = badpop (datalist [i])

for j in range (len (bad)):

datalist [0]. pop (bad [j])

datalist [1]. pop (bad [j])

datalist [2]. pop (bad [j])

datalist [3]. pop (bad [j])

correlation \_analysis (datalist,labels)

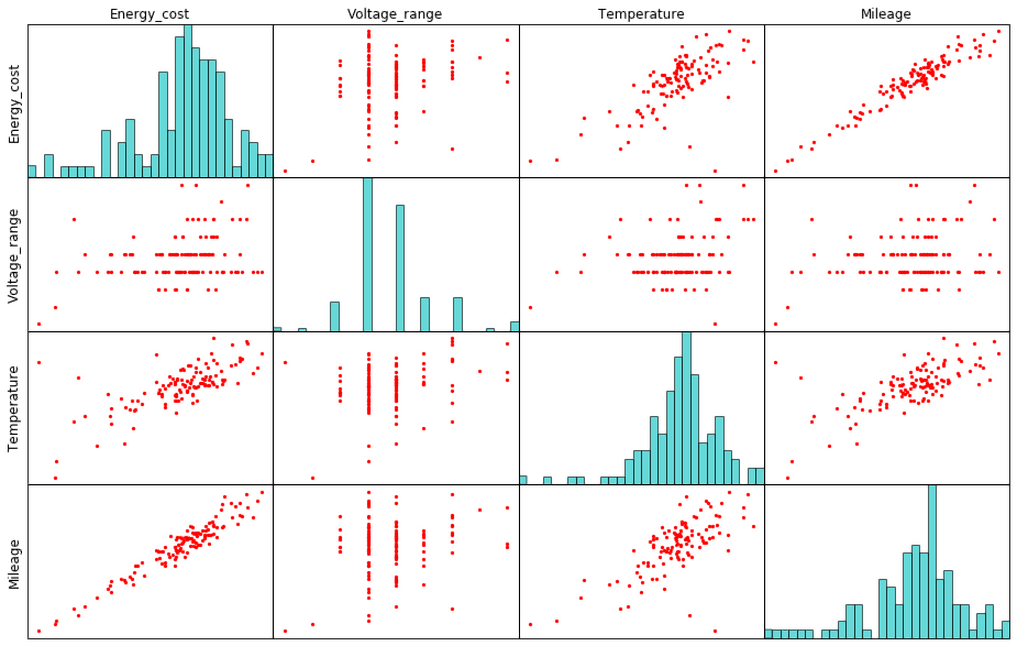
绘图结果如下：

图 3 - 7

由图3 - 5可见，电动汽车的全日能耗与行驶里程成正相关，全日能耗与电池组平均温度也有不明显的线性相关性，有进一步做线性回归分析的研究价值。电池组最大极差与其他变量没有明显的相关性，并且可以看出最大极差变量呈离散分布，这是因为探测电池组外电压的传感器最大精度为20毫伏。

* 1. 电池组电压一致性分析

一致性分析是指对多个变量在不同时间段的表现差异进行分析，在电池数据分析中体现为对电池组中每个电池的外电压与电池组整体进行比较，对外电压的不一致性进行分析。

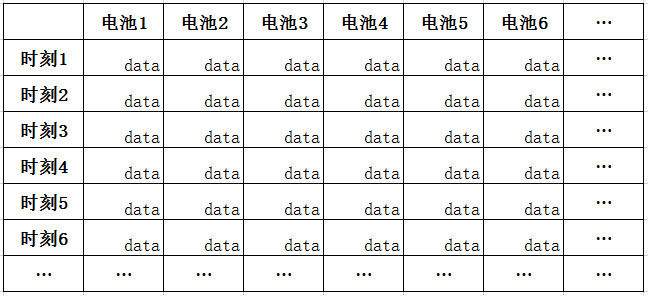
我们常常将多节电池串并联成组使用，使电池组达到一定的电压、功率和能量等级，而由于电池组中各单体电池之间存在的初始性能不一致问题，使得在充电的时候，电池组内的部分电池就会先于其他电池充满电，使得电池组不能继续充电，导致电池组的总储存能量较少；在放电的时候，电池组内的福分电池先于其他电池放完电，达到放电终止电压，使得电池组不能继续放电，使得电池组存储的容量和能量不能完全利用。但是如果忽略单只电池之间的差异而采用基于电池组端电压的充放电控制模式，继续对电池组进行充电或者放电，则会导致部分电池出现过充电和过放电，致使寿命缩短，甚至出现安全隐患；如果为了保护这些电池，而停止充电或者放电,就会导致电池组的容量和能量利用不充分，车辆的运行经济性和效率下降-随着电池的使用次数的增加，容量和性能差的电池更加容易出现过充电和过放电，使得这部分电池的容量和性能以更快的速度衰退，与其它电池之间的差异逐渐增加，从而导致电池之间的一致性越来越差。连续的充放电循环导致的单体电池的差异放大，将使某些单体电池容量加速衰减，从而使电池组过早失效。因此电池的组合应用不仅要求单体电池的性能指标达到规定的要求，对参与配组的单体电池性能的匹配也有严格的要求。在分析数据时，分析电池组的一致性具有重要的意义。

图 3 - 8

如图3 – 8，在本例中电池组由91个电池组成，传感器记录下了这91个电池在每个时刻的外电压，总共记录了约6000个时刻的数据，也就是数据是一个91×6000的矩阵。

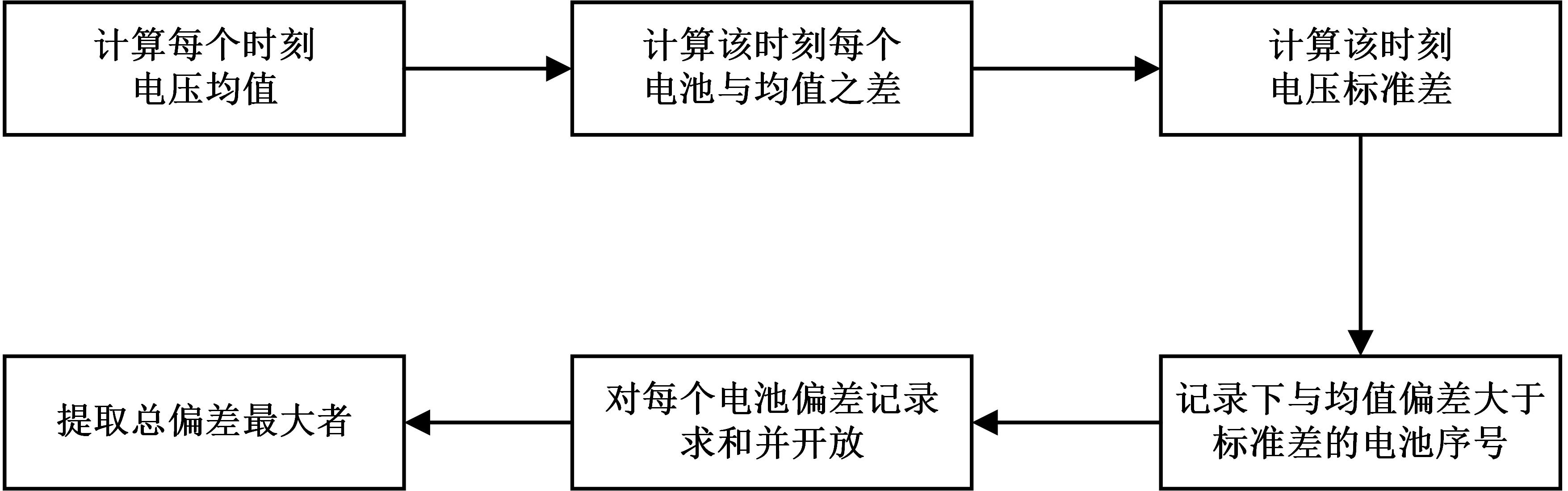


图 3 - 9

如图3 – 9，本例中使用的算法是，算出每个时刻91个电池外电压的均值，记录下该时刻每个电池与该均值的偏差，若该值超过了该时刻全部电池外电压的2倍标准差，则视为该电池在该时刻表现不一致。然后对每个电池全部时刻的不一致性进行统计，将其偏差的平方求和再开方，分析结果的分布情况得出当日运行过程中一致性较差的电池。实现算法如下：

path = u 'E:\\ project\\ 20150402\\ vehicle-京B7Y204 - 20150416094737. xls'

data = readlist (path,4,3)

ex = [ ] #各时间段电池组电压均值

voltage = [ ]

for value in data:

s = 0

for i in value:

s + = i

e = float (s) / len (value)

ex. append (e)

for j in range (len (data [0])): #单个电池与均值差值总和

s = 0

for i in range (len (data)):

s + = np. square ((data [i] [j] – ex [i]))

voltage. append (round ((np. sqrt (s)),2))

I = 0

overvt = [ ]

for vt in voltage:

i + = 1

if (vt-sum(voltage) / len(voltage)) > np. std (voltage) \* 2:

overvt. append ([vt,i])

fig = plt. figure ()

fig. set \_size \_inches (16,18)

ax1 = fig .add \_subplot (211)

ax2 = fig .add \_subplot (212)

ax1. set \_title ('Voltage Deviation of Every Battery',fontsize = 'large')

ax2. set \_title ('Voltage Deviation Distribution',fontsize = 'large')

m = ax1. bar (range (len (voltage)),voltage,0.5,color='r',alpha=0.6)

n = ax2. hist (voltage,50,color='c',alpha=0.8)

print u' \n 与其他电池电压一致性较差的电池有：'

for i in overvt:

print u' 第%d号电池，其电压偏差的均方差为%. 2f毫伏'% ( i [1],i [0])

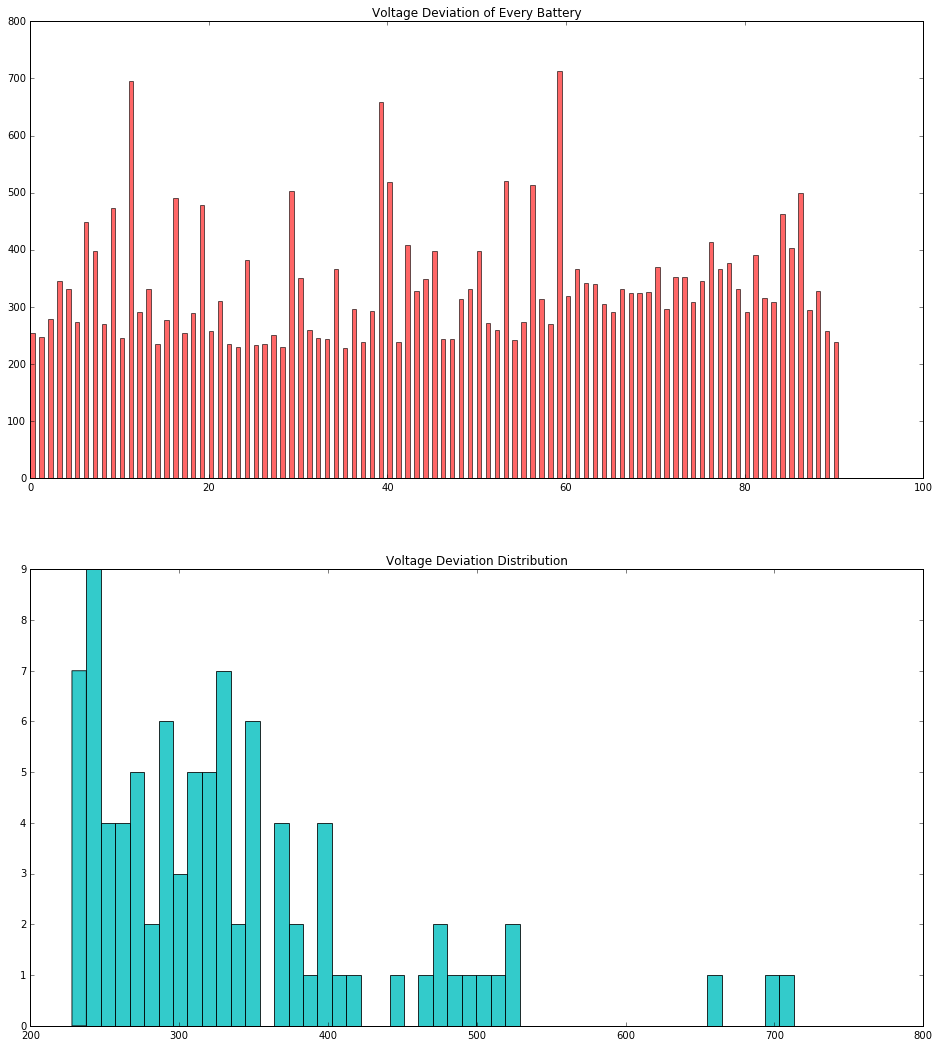
 程序运行结果：

图 3 - 10

图 3 - 11

与其他电池电压一致性较差的电池有：

第12号电池，其电压偏差的均方差为695.06毫伏

第40号电池，其电压偏差的均方差为657.94毫伏

第60号电池，其电压偏差的均方差为713.16毫伏

文字输出为一致性分析结果，图3 - 10显示了从1号到91号每个电池的电压偏差值。图3 - 11显示出所有电池电压偏差的分布情况，由图可见有三块电池电压偏差远大于其他电池。通过此分析可快捷直观地发现在电池组中与其他电池外电压表现不一样的电池。对表现不一致的定义可以根据实际情况修改或重写。

* 1. 驾驶行为分析

借助电池数据文件中的其他数据，我们还可以对每辆电动汽车的驾驶行为进行分析。此处试举两例。

* + 1. 停车时间

借助数据中的电机转速，或者汽车时速，及其相对应的时间戳，我们可以提取出电动汽车停车的次数、时间及时间段。我们还可以设置阈值，仅提取超过设定时间的停车行为。

if \_\_name \_\_ = = " \_\_main\_\_":

try:

path = u 'E:\\ Project\\ 20150402\\ vehicle -京B7Y238 - 20150416130635. xls'

speed = readata (path,1,12,4)

time = readata (path,1,15,0,1)

n = 10

stoptime = [ ]

turn = 1

for i in range (len (speed)):

if (speed [i] = = 0) and (turn = =1):

begin = time [i]

turn = 0

elif (speed [i] ! = 0) and (turn = = 0):

end = time [i]

turn = 1

if timediff (end,begin) [0] > = 60 \* n:

stoptime. append ( [begin,end,timediff (end, begin) [1], timediff (end, begin) [2], timediff (end , begin) [3] ] )

print u' % s 超过%d分钟停车时间：'% (path [28:35] , n)

for j in range (len (stoptime)):

print u'从%s到%s，共%d小时%d分%d秒'% ( stoptime [j] [0] , stoptime [j] [1] , stoptime [j] [2] , stoptime [j] [3] , stoptime [j] [4] )

except:

print 'Error'

程序运行结果：

京B7Y238 超过10分钟停车时间：

从09:36:31到11:56:50，共2小时20分19秒

从12:00:20到12:37:43，共0小时37分23秒

从12:56:23到13:12:03，共0小时15分40秒

从15:01:23到15:20:13，共0小时18分50秒

从16:51:56到19:03:59，共2小时12分3秒

本例设置阈值为10分钟，可以根据实际所需情况更改。借助上文中的多文件处理方法，可以提取出所有电动汽车的停车次数和停车时间段。

* + 1. 驾驶路径

利用数据中的坐标信息，我们可以通过散点图画出电动汽车的驾驶路径。若使用时间戳将不同时间段的路径用不同的颜色标出，可以得到电动汽车在不同时段的驾驶路径和活动范围。利用坐标数据所对应的时间戳可以将活动范围拆分成任意部分，如将每个小时的行驶路径单独提取绘图。若利用坐标结合地图，即可得到更加详细的电动车活动范围图。

* 1. 温场分析

温度是电池组的重要参数，不同的温度下，电池组的内阻，外电压，放电能力都有所不同。电池组温度过高或者过低都会影响电池的性能。

上文已经对温度的最值、极值进行了提取，也分析了电池组温度与其他参数的相关性。但这都是将电池组视作一个整体来进行的。由于电池组中的每个电池位置不同，散热条件不一样，导致不同的电池温度不同，也就是存在温场分布[11]。利用已有的方法，我们可以对温场分布对电池组各性能造成的影响进行研究。

首先分析各个传感器探测到的温度分布。

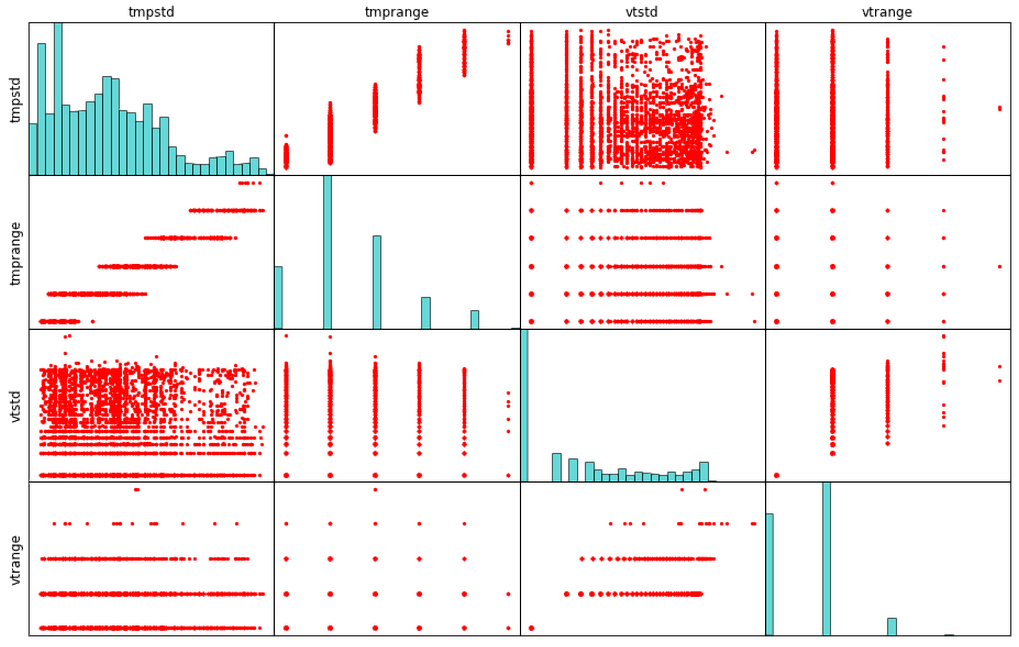
 使用提取特征参量的方法，提取电池组温场分布的均方差，极差和电池组电压的均方差、极差进行相关性分析。分析结果如下：

图 3 - 12

由图3 - 8可见电池组的温场分布与电压分布并没有明显的相关性出现，仅有电池温场的均方差和极差有较明显的线性相关性。从图中可以看出变量均呈现跳跃式分布，说明传感器的探测精度有限，尚不足以支撑本项研究。此时应该重新设置研究方案，选取不同变量或者加入其他变量进行协助分析。

1. GUI设计

图形用户界面（Graphical User Interface，GUI），是指将计算机操作以图形的方式显示给用户，以便用户直观地进行操作的计算机程序[12]。又称图形用户接口、程序前端。利用GUI设计，可以做出漂亮直观的用户界面，让使用者不必对着IDE中的一堆代码使用程序。

* 1. 交互界面布置

根据上文的分析，交互界面设计的任务由Eric + PyQt的工具来完成，其中Eric是作为开发及测试用的IDE来使用，而PyQt用于图形界面的布置设计。PyQt是著名的GUI交互界面设计软件Qt设计师的Python语言定制版，编写的源码为C++，空间丰富，函数和方法数量多，该软件使用拖拽布局，开发简易，设计完成后的GUI程序经处理方便打包成二进制文件，是当前使用最广泛的Python GUI开发工具。其中PyQt 4针对Python 2 . x 版本，PyQt 5针对Python 3 . x 版本，本文中使用的是PyQt 4。至于为什么要使用Eric作为GUI设计的IDE而不是用之前Eclipse + PyDev工具组合，原因在于Eric原本就是作为GUI设计专用的IDE而被开发出来的，其完美整合了PyQt的开发功能，许多功能进行了针对PyQt使用的优化，避免了兼容性问题。并且其具有变量补全，自动缩进，错误检查与运行分析等功能，有不输于常规IDE的高效性，因此是Python GUI开发的最优选择。

GUI设计的理念是将程序的众多技术细节封装起来，将程序源代码和运行过程在后台隐藏，对用户呈现程序的主要功能，提供具有简明性，易学性和一致性的程序输入/操作方法，并且返回简洁可读的运行结果。对于程序运行中出现的问题要及时返回，但不返回对于一般用户没有意义的错误提示信息。本着用户友好的原则，我们需要对上文中以完成的程序进行封装，做出GUI界面，使数据处理过程对于研究人员更加简明易用。

我们取上文中几段代码做GUI设计示范。

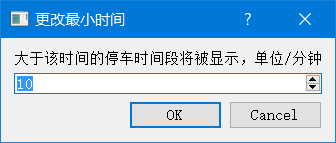
 图 4 - 1

图 4 - 2

图4 – 1是使用Eric和PyQt做出的中查询停车时间的交互界面，在输入框中输入所需要查询的车辆数据文件所在的路径，就会在文本框中输出该车该天的停车行为。程序默认输出超过10分钟的停车时间，点击左下角的更改时间会弹出一个小窗口，如图4 – 2，在里面可以修改停车时间的数值。

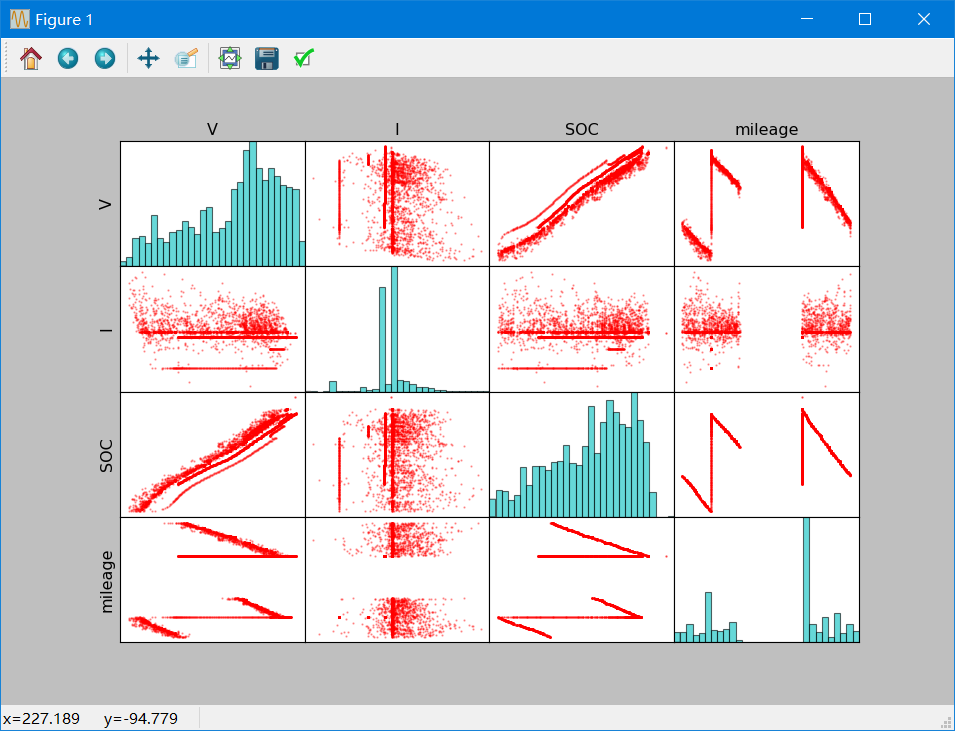
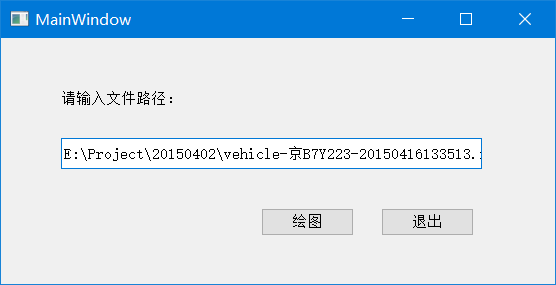
图 4 - 3

图 4 - 4

图4 - 3是相关性分析的GUI界面，同样的输入路径，会输出一张多变量相关性分析的散点/分布图（4 – 4）。图片窗口具有保存、缩放与局部缩放、设置坐标、改变边框大小、撤销与重做等功能，能够满足对绘图的一般需求。从图中我们除了可以得出3.4.1中的结论外，还可以由这辆车的里程—剩余电量关系图可以看出本车丢失了某一大段时间数据，即里程—剩余电量关系图中空白的部分。可见对同样的分析方法对于类似的分析对象也会得出不同的结果来。

* 1. 程序打包

程序打包又称程序封装，是指将已写好的代码封装成一个.exe文件，并且可以脱离程序的开发环境使用而不需要借助解释器。Python提供了程序打包工具PyInstaller，可以选择将程序打包成单个文件或一个文件夹。与早期的Python打包工具Py2exe相比，Pyinstaller操作更加方便，指令更加简单，可以在Windows、Linux、Unix、Mac Os等各种平台上运行，并且有多种参数选项可以设置，还可以指定二进制文件的图标。而Py2exe只能在Windows上运行，不能打包成单个文件，打包后是一个文件夹，包含各种dll和资源。因此选择PyInstaller作为我们的打包工具。

在由Eric与PyQt设计好的GUI程序所存放的文件夹中打开命令提示符，键入Pyinstaller –w –F xxx.py即可由工具自行打包。其中xxx.py是需要打包的程序的文件名，-w参数用于隐藏控制台，若不加该参数，打包完成后的exe程序在运行时桌面上会显示控制台，也就是一个类似命令提示符的黑色窗口；–F参数用于将程序集成于一个单独的exe文件之中，若不加该参数，程序相关资源不会自动集成，因此打包完成后是一个带有各种组件的文件夹。

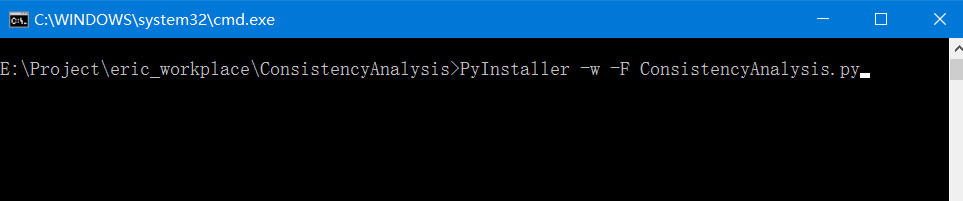
 以上文中的一致性分析为例，其GUI设计已经完成，现在对其进行程序打包。

 图 4 - 5

图 4 - 6

打包完成后，原目录中会多出两个文件夹，其中build文件夹里是各种相关的资源和中间文件，在完成后可以删除；dist文件夹中存放有打包完成后的程序，即图4 – 6中的ConsistencyAnalysis.exe文件。这个文件集成了程序运行所需要的所有条件，可以不依赖平台而被使用，具有良好的可移植性。

1. 经济性分析

在当今能源危机与环境污染成为人类社会所面临的主要问题的时代背景下，发展电动汽车是目前世界各国重要的战略性产业之一。美国、日本、德国等传统汽车制造业大国都启动了对大容量动力电池的研究攻关，意图开发出满足电动汽车常规使用需求的动力电池。我国在该项研究上亦投入了大量的精力与资源。在未来的汽车市场上，电动汽车将会是重要组成部分，甚至有可能取代内燃汽车成为主流汽车类型，其经济规模之庞大可以成为国家经济的重要组成部分。可以预见，在这个尚未成熟的领域，谁能率先掌握核心技术、成为行业标准的制定者，谁就能占领先机，分走最大的一块蛋糕，其巨大的经济效益值得投入大量资源进行技术攻关。

众所周知，电动汽车的三大关键技术中，电池技术是制约电动汽车发展的最大瓶颈。电能的储存是一项世界性的难题，若能在这一领域上有所突破，就能在工业生产中带来巨大的经济效益，而电动汽车的发展也将得到长足的进步。出于此种考虑，对大容量大功率动力电池的研究一直是我国电气行业的重要科研项目之一。对电动汽车电池数据的分析是其中重要的组成部分，若能进行良好的数据处理，可以大大加快研究进展，节省科研经费，总结有效经验而少走弯路，避免故障与事故的出现，及其可能造成的经济损失，其经济效益不言而喻。

本文中使用Python语言及其科学计算包作为数据分析的工具，可以带来一下的经济性收益：

首先，Python语言是完全开源免费使用的，而相比之下，另一种常用于数据分析的软件Matlab则十分昂贵。Matlab作为一款商业软件，获得正版授权，价格不菲。最便宜的学生版，核心组件单个授权要99美元，想使用额外工具箱，则是每个工具箱29美元。而作为可以在研究所中使用Matlab中国大陆的标准版售价为一万五千元人民币，无疑给研究所带来了额外的经济负担。并且使用Matlab过程中还很有可能遇上版权问题， Mathworks论坛上很多有价值的代码版权归Mathworks公司，要想使用必须获得它的授权。

其次，相比于其他常用的编程语言诸如Java、C语言而言，Python语言语法简单，框架明晰，扩展包丰富而易用，对于非计算机专业的研究人员也十分容易学会并上手使用，而不需要聘请专业人员来做针对数据分析的程序开发，为研究室节省了人力资源上的开销。而且相比于其他编程开发语言，Python语言是一种相当高级的语言，也就是不需要考虑程序实现时的诸多底层细节（比如堆栈操作、存储寻址等），语法高度精炼集中，所以Python语言的一大特点就是代码量极少，实现同样的功能，Python程序只需要相当于Java、C语言十分之一甚至几十分之一的代码量。由于数据分析任务的多变性，每次任务基本都需要修改或者重写代码来实现，Python语言的这一特性使开发人员写代码的时间得到了极大程度上的节约。我们都知道时间成本也是经济成本中的一部分，节省下来的时间可以让研究人员利用起来做其他有意义的工作，Python语言在这上面为我们创造了一定的经济效益。

1. 结论

本文建立起了一套使用Python语言及其科学计算包进行电动汽车电池数据分析的方法，实现了对大量电动汽车电池数据特征参数的自动化提取，对大量数据进行批量自动化处理。体现出了Python语言在大量电池数据处理中的高效性。

本文回顾了电动汽车电池及其数据分析的研究现状，综合比较各类数据分析软件与主流编程语言在数据处理、科学计算上的优劣，提出使用Python语言及其科学计算包作为电池数据分析的计算机工具。本文遵循程序开发的Easy模式，搭建起了一套最优化的数据平台及开发环境，囊括了众多编程开发以及数据处理所需要的工具，降低了开发人员在程序编写过程中可能由于开发环境而导致的各种困难。

利用Python语言的OS模块可以进行目录操作，实现对单一或多个文件进行数据提取的任务，在掌握了数据的组织形式后，可以进行数据清洗、格式转化等工作，使数据以最有利于处理的方式组织起来。通过使用Python科学计算包，对电池数据做多样化的处理，包括特征参量提取、相关性分析、一致性分析、驾驶行为分析、温场分布分析等等。每种分析方法均可加入不同的变量进行，包括电池组的总电压、总电流、单个电池电压、温度、行驶里程、剩余电量、电机状况等等。通过不同数据与不同方法的组合，得出相关的结论，研究电池的各项参数性能。

设计出将数据进行可视化的方法，使对各个变量的分析更加简洁易读。设计出有利于研究人员进行直观操作的GUI程序，并将程序打包，使其不依赖与Python开发环境，在任意一台计算机上均可运行。

以本文为例，本次电动汽车行驶数据分析完成了对122个文件中电动汽车数据的提取，从中提取特征参量，完成各项分析。如果不借助计算机工具直接对Excel文件进行这种操作，工作量十分巨大。使用Python科学计算包不仅可以完成对多文件的读取操作，更能自动化完成各项数据分析。为大规模的电池数据分析提供了可靠高效的统一方法。

在本文的研究基础上，后续工作可以由一下两个方面展开：

1、对更多不同来源的电池数据进行分析，找出电动汽车电池数据分析的共性，在此基础上建立具有普遍性的分析方法和流程。

2、对于会反复使用的代码，进一步优化程序，简化算法，或利用C / C++ 对Python程序中运行速度慢的部分进行重写，提高程序运行效率。

参考文献

[1]谢文捷. 世界能源安全研究[D].北京.中共中央党校.2006.1-2

[2]李仲兴.电动汽车电池管理系统中电池均衡技术研究[D].江苏.江苏大学.2011.3-4

[3]纪淼淼.纯电动汽车电池管理数据分析的研究[D].哈尔滨:哈尔滨工业大学，2014.6-27.

[4]王纬.基于数据驱动的锂电池数据处理[D].北京:北京交通大学，2013:7-1.

[5]胡阳波.数据分析在锂电池品质中的应用[J].电子设计工程，2014年第22卷第18期：32-35 .

[6]任远博.基于Ajax技术的计量管理系统设计与实现[D].武汉.武汉理工大学.2012.6

[7]李真,斯旎.混合编程的网络设备自动备份的设计与实现[J].大众科技.2015年1期.37-39

[8] Wes McKinney. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Ipython[M]. California:O'Reilly Media,2012

[9]狄博,王晓丹.基于Python语言的面向对象程序设计课程教学[J].计算机工程与科学.2014年1期.122-125

[10]张雪峰,李文林.Matlab在高等数学中的几点应用[J].安阳师范学院学报.2009年5期.121-122

[11]李茂德,王峰,梁芃.动力电池散热数值模拟分析[J].热科学与技术.2011年2期.103-109

[12]陈涛.基于MVC模式的视频播放器图形用户界面的设计与实现[J].商丘师范学院学报.2010年6期.92-93

致 谢

本文是在我的导师张彩萍老师的悉心指导之下完成的，张老师严谨的治学态度、渊博的学识体系、精益求精的指导作风和孜孜不倦的科研精神都给我留下了深刻的印象。本次设计从选题解读、制定研究方案到解决各种问题、论文撰写无一不包含着导师的心血和汗水。得益于老师的悉心指导和严格要求，我不仅完成了本课题的研究内容，更重要的是学会的做学问的方法和态度。能够得到张彩萍老师的言传身教，我感到十分幸运。张老师值得我永远学习。

感谢北京交通大学新能源所的张腾师兄的指导和帮助，在本次设计期间，他以极大的耐心支持我，有问必答，帮助我克服了许多困难。师兄待人坦诚和蔼，治学严谨认真，工作富有热情，是我学习生活的榜样。

感谢我所有的老师，在四年的学习生活中给予了我悉心的教导，帮助我克服了众多难关，为我解答了许多问题，使我有能力完成本次设计。感谢我周围的同学，在学习生活中为我提供了许多的支持，在他们身上我学到了很多东西，使我在各个方面上都能获得进步。

最后感谢我的父母，他们无私的爱是我的坚强后盾。他们在过去二十余年中给予了我最好的家庭环境，以自己的一言一行教导我、支持我，让我在耳濡目染中获得进取的精神与独立的人格。他们的关心与鼓励是我不断前进的动力。

附 录

附录1 文献翻译

原文

**Reasonable Python or how to Integrate F-Logic**

**into an Object-Oriented Scripting Language**

Markus Schatten

Faculty of Organization and Informatics

University of Zagreb

Pavlinska 2, Varaˇzdin, HR-42000, Croatia

markus.schatten@foi.hr

Abstract: Python is an object-oriented scripting language known for its ability to support various programming paradigms. In Python one can write procedural, functional, object-oriented, and thanks to metaclasses even aspect-oriented code. Even if some efforts were done to support the last major programming paradigm, logic programming is still not supported in a Python programmer friendly way.

In this paper a solution that aims on this target using F-Logic (particularly FLORA-2), which syntax is much more compatible with the Python language than traditional Prolog syntax, is presented. In order to make such an integration useful ZODB (Zope Object Base) is used to facilitate permanent storage of Python objects, while the FLORA-2 engine built on XSB is used for reasoning facilities.

To take advantage of logic programming concepts like facts, rules, variables or queries, special logical Python object are introduced. In the end some examples of usage are shown and future development guidelines are given.

**I. INTRODUCTION**

Python is a well known dynamic object-oriented scripting language initially implemented by Guido van Rossum for the Amoeba operating system, later published on USENET and made public available. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types (like lists, tuples, dictionaries etc.), classes, and metaclasses. The language comes with a large standard library that covers areas such as string processing (regular expressions, Unicode, calculating differences between files etc.), operating system interfaces (system calls, filesystems, TCP/IP sockets etc.), Internet protocols (HTTP, FTP, SMTP, XML-RPC, POP, IMAP, CGI programming etc.), and software engineering (unit testing, logging, profiling, parsing Python code etc.) [7].

One of the great advantages of the Python programming language is the fact that it supports various programming paradigms like procedural, functional, object-oriented or even aspect oriented programming. Even if there were some interesting attempts [5] [1] [2] to introduce the logic programming paradigm into Python due to incompatibilities to traditional Prolog syntax this task is still not achieved.

In [5] Petuello and later Zagrodnick, created a Python interface to SWI Prolog called PyLog. It allows interaction between Python and Prolog code, loading of Prolog programs, issuing queries and the useage of query results. The impossibility to use Python objects in an intuitive way is a major drawback of this interface.

In [1] and [2] Berger and Coelho, respectively, propose a metaprogramming approach to introduce logic programming into Python. Berger uses Pythons method override facilities to provide a more intuitive way to produce Prolog-like code in Python. These solutions still lacks the possibilities to completely integrate logic programming into Python since logic programming concepts are in a certain way separated from the rest of the ’normal’ Python code. They also build upon traditional Prolog syntax which seems not to be the optimal solution for an object-oriented programming language like Python.

As opposed to traditional Prolog syntax frame logic based languages [3] seem to be much more compatible with Python syntax. Especially the object-oriented knowledge base language FLORA-2 [10] has very interesting (but rather coincidental) similarities in its syntax to the Python syntax. Additionally FLORA-2 integrates F-Logic, HiLog and Transaction Logic which makes it a powerful knowledge representation and inference language. In [6] Yang, Kifer and Zhao argue that FLORA-2 ”is a flexible and natural framework that combines rule-based and object-oriented paradigms”.

Another system to mention here is the Zope Objectbase (ZODB) developed by the Zope Corporation [12]. It’s a simple object-oriented database which has the capabilities to store Python objects in a permanent way. However it lacks any reasoning facilities since all object are stored in a rather large Python dictionary. By combining the reasoning engine of FLORA-2 and the capabilities of ZODB it is possible to use knowledge bases in Python, as argued further.

**II. INTRODUCING LOGIC PROGRAMMING CONCEPTS INTO PYTHON**

Most important logic programming concepts to introduce into Python addressed here are logic variables, facts, rules and queries. In FLORA-2 logic variables are denoted by any word starting with an uppercase letter or the underscore (’ ’). A similar way is used in Python, by creating a class of objects named Variable which behavior is shown in the following Python interactive shell session.

>>> X = f.Variable( ’X’ )

>>> X

<f.Variable instance at 0xb7dd854c>

>>> print X

X

>>> X == 27

>>> print X

X = 27

When analyzing facts, rules and queries we can concludethat they are all instances of a construct which has the form:

Head : −Body. (1)

: −Body. (2)

Head : − . (3)

(1) is a rule where Head is the rule head, and Body is the rule body (denoted by Head :- Body.). (2) is a query where Body is the query string (denoted by - Body.). (3) is a fact where Head is the actual fact (denoted by Head.). We can conclude that we need only one class when introducing these concepts into Python which behavior is shown in the following interactive shell session (where Bogus is a logical class as argued further).

>>> x = f.Construct()

>>> print x

>>> h, b1, b2 = f.Bogus(), f.Bogus(),

f.Bogus()

>>> x & h

>>> x. class

<class f.Fact at 0xb7ae356c>

>>> del x

>>> x = f.Construct()

>>> x << b1 & b2

>>> x. class

<class f.Query at 0xb7ae365c>

>>> del x

>>> x = f.Construct()

>>> ( x & h ) << b1 | b2

>>> x. class

<class f.Rule at 0xb7ac5dac>

One can see that x which is an instance of the Construct class dynamically changes its class to Fact, Rule or Query according to its content (e. g. its head and body). The bitwise shift (’<<’), the bitwise or (’|’) and the bitwise and (’&’) were overridden in order to allow logic programming constructs, and stand for ’:-’, ’;’ and ’,’ respectively. This means that a logic programming construct like:

X : −(Y, Z); W. (4)

in Python would be written like:

X << (Y&Z)|W. (5)

Which is similar to both FLORA-2 and Python syntax. In addition to these two classes some additional classes were defined to ease such behavior.

**III. TRANSLATING PYTHON OBJECTS INTO F-MOLECULES**

Another task to be accomplished is the translation of Python objects into F-molecules in FLORA-2 syntax in order to allow reasoning about them. Python has a feature which eases this translation and can be seen in the following interactive shell session.

>>> class a:

... def init ( self, b1 = 1, b2 = 2 ):

... self.b1 = b1

... self.b2 = b2

...

>>> y = a()

>>> dir( y )

[’ doc ’, ’ init ’, ’ module ’, ’b1’, ’b2’]

>>> y. dict

’b1’: 1, ’b2’: 2

>>> y. class

<class main .a at 0xb7a27b9c>

We can conclude that Python objects allow us to query their internal structure and content. Using this feature a multiple recursive algorithm was developed which translates Python objects into FLORA-2 F-molecules. For example the y object from the example when translated into FLORA-2 would be:

>>> tr = py2f.py2f()

>>> tr.f2obj( y )

’py0xb7db4e2cpyobj main py :

pyapyclass main py ["b1"->1, "b2"->2]’

To have unique object names in our knowledge base we use the objects internal memory addresses and some additional information about its module. When an object is translated it is automatically stored in a ZOBD object base for later retrieval and permanent storage. Python objects which are not logical constructs as argued earlier, are considered to be facts.

**IV. CONNECTING FLORA-2 AND PYTHON**

In order to use the FLORA-2 engine from Python an interface had to be developed. FLORA-2 is build upon the OpenSource XSB Prolog engine which is developed in C and thus has a C interface [8]. The Simplified Wrapper and Interface Generator (SWIG) was used to create an interface between XSB and Python (but could be easily extended to any other language supported by SWIG i. e. AllegroCL, C# - Mono, C# - MS .NET, CFFI, CHICKEN, CLISP, Guile, Java, Lua, MzScheme, Ocaml, Perl, PHP, Ruby, or Tcl/Tk). Through such an interface FLORA-2 is then loaded as a module into XSB. Additional wrapper classes were developed to ease communication between XSB and Python, and FLORA-2 and Python, as shown in the following interactive shell session.

>>> f = interface.Flora2()

[FLORA2 specific output]

>>> f.consult( ’test’ )

[FLORA2 specific output]

>>> f.query( ’X:person[ Y -> Z ].’, [’X’,

’Y’, ’Z’] )

[{’Y’: ’age’, ’X’: ’mirko1’, ’Z’: ’40’},{’Y’:

’age’, ’X’: ’mirko2’, ’Z’: ’42’}]

>>> f.close query()

We can conclude that such an interface allows for using XSB and FLORA-2 engines from Python. Compiling and loading of Prolog and FLORA-2 specific programs is also supported. Queries can be issued directly whereas return variables have to be provided in a Python list as the second argument. The results of queries are Python lists which elements are Python dictionaries where each dictionary represents one solution. Keys of each dictionary are the provided return variables, and the matching values are the returned values from the knowledge base.

**V. PUTTING IT ALL TOGETHER**

After describing the particular parts of this integration it is possible to connect all together in order to facilitate logic programming in Python. All previously described parts constitute a Python module. In addition to theses parts a new class was defined which integrates them. In particular this class represents an F-Logic base with storing and querying facilities.

Any object to be stored in to the knowledge base is first stored in the ZODB in order to be persistent. Afterwards it is translated into FLORA-2 syntax and loaded into the FLORA-2 engine. Additionally the FLORA-2 code is saved in an external file so the state can be restored later. This behavior of the system is transparent to the Python programmer.

To query the knowledge base one can either use FLORA-2 syntax or create special query objects using logical variables and logical constructs. In the following interactive shell session we first create a construct and define a logical class which we will use for querying the knowledge base.

>>> x = Construct()

>>> class query object( Logical ):

... def init ( self, a, b ):

... Logical. init ( self )

... self.a = a

... self.b = b

Now we can create a query object and modify it to fit our needs. Note how it is possible to use a logical object to query for any object of any class by overriding its type and class type attributes with logical variables. We can also override the names of attributes by inserting a logical variable in the place of the attributes name.

>>> q = query object( Variable( ’X’ ),

Variable( ’Y’ ) )

>>> x << q

>>> print x

?- py0xb7d6cdacpyobj main py :

pyquery objectpyclass main py [

"b"->Y, "a"->X].

>>> q. type = Variable( ’Z’ )

>>> print x

?- Z : pyquery objectpyclass main py [

"b"->Y, "a"->X].

>>> q. classtype = Variable( ’W’ )

>>> print x

?- Z:W[ "b"->Y, "a"->X].

>>> q. dict [ Variable( ’V’ ) ] = q. dict [

’a’ ]

>>> del q. dict [ ’a’ ]

>>> print x

?- Z:W[ "b"->Y, V->X].

>>> del q. dict [ ’b’ ]

>>> print x

?- Z:W[ V->X ].

To query the knowledge base some facts and/or rules have to be stored in it. A simple data object class which subclasses the Persistent class is created in the following. The subclassing allows all instances of the class to be stored in the ZODB. Instances to be stored in the knowledge base are also created in this interactive shell session.

>>> class data object( Persistent ):

... def init ( self, a, b ):

... self.a = a

... self.b = b

...

>>> d1 = data object( 1, 2 )

>>> d2 = data object( 3, 4 )

Now it is possible to create the knowledge base which is just an Python object like any other and insert the data objects.

>>> fb = FBase( ’my flbase’ )

[FLORA-2 specific output]

>>> fb.insert( d1 )

[FLORA-2 specific output]

>>> fb.insert( d2 )

[FLORA-2 specific output]

By using the previously created query object the following results are obtained. Note that the W variable returned the string ’class’ which is due to the impossibility of ZODB to store class objects (e. g. they are not persistent).

>>> fb.query( x )

[{’X’: ’1’, ’Z’: < main .data object

object at 0xb7dde96c>, ’W’: ’class’},

{’X’: ’2’, ’Z’: < main .data object object

at 0xb7dde96c>, ’W’: ’class’}, {’X’:

’3’, ’Z’: < main .data object object at

0xb7c52aac>, ’W’: ’class’}, {’X’: ’4’, ’Z’:

< main .data object object at 0xb7c52aac>,

’W’: ’class’}]

This simple example shows how transparent the knowledge base is to a Python programmer. The programmer just has to create an FBase object to store Python objects in the knowledgde base.

To query the knowledge base query objects have to be created. Using logic objects and logical variables this task is intuitive and fair easy to accomplish.

To add rules to the knowledge base one has to use constructs in order to create them and store them. These rules are similar to their Prolog and FLORA-2 counterparts with some minor syntax differences still preserving the flexibility of normal Python code.

**VI. CONCLUSION AND FUTURE WORK**

In this paper an integration of F-logic, especially FLORA- 2 and the dynamic object-oriented programming language Python was proposed. This integration showed some advantages to other attempts of supporting logic programming in the Python scripting language. These advantages include more intuitive syntax for Python programmers, complete integration and reasoning over Python objects, and permanent storage. This integration could allow Python programmers to easier create knowledge base, ontology and semantic web solutions. Other possible usage would include automated applications generation, reasoning about module capabilities, and automated software testing.

Even if this solution is a step forward it has still to be developed further. Future development will include a more network oriented system which should allow multiple knowledge base users, client server architecture and the possibility to exchange Python objects over the network. A main idea is to connect Net Work Spaces (NWS)[4] with the developed system in order to facilitate such a system.

**Acknowledgments**

This paper was written after a disscusion with Michael Kifer over the FLORA-2 mailing list and a discussion with my mentor Mirko Cˇ ubrilo. In this place I’d like to thank them for suggestions and ideas.

**REFERENCES**

[1] S. Berger, Pythologic – Prolog syntax in Python, on-line<http://aspn. activestate. com/ASPN/Cookbook/Python/Recipe/303057>, accessed: 21st November 2006.

[2] F. Coelho, Extending python with prolog syntax \*and resolution\*, on-line<http:// aspn. activestate.com/ASPN/Cookbook/Python/Recipe/360698>,accessed: 21st November 2006.

[3] M. Kifer, G. Lausen, J. Wu, Logical Foundations of Object-Orientedand Frame-Based Languages, Journal of the Association for Computing Machinery, May 1995.

[4] Nws-py.sourceforge.net, NetWork Spaces for Python, on-line<http://nws-py.sourceforge.net/>, accessed: 21st November2006.

[5] W. M. Petullo and later C. Zagrodnick, PyLog, on-line<http://www.gocept. com/open source software/Pylog>,accessed: 19th November 2006.

[6] M. Pilgrim, Dive into Python, on-line <http://diveintopython.org/>, accessed: 19th November 2006.

[7] Python.org, General Python FAQ, on-line

<http://www.python.org/doc/faq/general/>, accessed:24th March 2007.

[8] SWIG, Simplified Wrapper and Interface Generator, on-line<http://www.swig.org>, accessed: 19th November 2006.

[9] XSB, XSB API documentation, on-line <http://xsb.sourceforge.net/api/index.html>, accessed: 19th November 2006.

[10] G. Yang, M. Kifer, C. Zhao, FLORA-2: A Rule-Based Knowledge Representation

and Inference Infrastructure for the Semantic Web, In SecondInternational Conference on Ontologies, Databases and Applications ofSemantics (ODBASE), Catania, Sicily, Italy, November 2003.

[11] G. Yang, M. Kifer, C. Zhao, V. Chowdhary Flora-2 :User’s Manual, Version 0.94, (Narumigata), on-line <http://flora2.sourceforge.net>, accessed: 29th November 2006.

[12] Zope.org, ZODB, on-line <http://www.zope.org/Wikis/ZODB/FrontPage>, accessed: 21st November 2006.

译文

**优化Python：**

**如何将框架逻辑融入面向对象的脚本语言**

Markus Schatten

萨格勒布大学

组织与信息学学院

Pavlinska 2, Varaˇzdin, HR-42000, 克罗地亚

markus.schatten@foi.hr

**摘要**：Python是一种面向对象的脚本语言，支持多种编程范式。使用者可以通过Python开发程序，定义函数，编写面向对象，甚至面向方面的代码。然而对最后一个主要的编程范式——逻辑编程，就算开发者付出诸多努力希望对其提供支持，Python仍然不能对程序员友好地支持逻辑编程。本文提出一个解决方案，针对这一目标使用框架逻辑（尤其是FLORA-2），其语法结合Python语言比传统Prolog的语法更兼容。为了使这种方法整合有用的ZODB（Zope对象库），帮助Python对象永久存储， FLORA-2或被用于推理功能。利用逻辑编程的概念，如事实、规则、变量或查询，介绍Python对象的特殊逻辑。最后展示一些例子，并给出未来的发展方向。

**一、简介**

Python是一个著名的动态、面向对象的脚本语言，初步实现了由Guido van Rossum的提出操作体系，并发表在Usenet，对所有人可用的。它包含了模块，动态类型，高级别的动态数据类型（如列表、元组、字典等），类和元类。该语言包含一个庞大的标准库，包括字符串处理（正则表达式，Unicode文件之间的差异，科学计算等），操作系统接口（系统调用、文件系统、TCP/IP套接字等）、网络协议（HTTP、FTP、SMTP、POP、IMAP、XML-RPC，CGI编程等），软件工程（单元测试、记录、分析、解析Python代码等）[7]。

Python编程语言的一个巨大优势在于它支持多种编程范式，即面向对象的程序，甚至是面向方法的编程。即使有一些有趣的尝试[5][1][2]介绍逻辑编程范式引入Python由于不兼容传统的Prolog的语法这个任务还没有实现。

在[5] 中Petuello和后来的Zagrodnick，创造了一个Python接口称为pylog SWI Prolog。它允许Python和Prolog代码之间的交互，Prolog程序加载、查询和查询结果的使用。无法在一个更直观的方式使用Python对象是该接口的一个主要缺点。

在[1]和[2]中Berger 和Coelho，分别提出了元编程的方法引入Python逻辑编程。伯杰用Python方法重写设在Python产生Prolog代码提供一种更直观的方式。这些解决方案还缺乏的可能性完全整合的逻辑编程Python从逻辑编程的概念以一定的方式从“正常”的Python代码的其余部分分开。他们还建立在传统Prolog的语法，然而这并不是一个面向对象的编程语言如Python的最优解。

相对于传统的Prolog的语法，框架逻辑为基础的语言[3]似乎与Python语法更兼容。尤其是面向对象的知识库语言FLORA-2 [10]与Python语法有非常有趣的（并非巧合）相似之处。此外FLORA-2集成框架逻辑，高阶逻辑和事务逻辑，使得一个强大的知识表示和推理语言。在[6]中，Yang, Kifer 和 Zhao认为FLORA-2“是一个将基于规则和面向对象的范式相结合的灵活和自然的框架”。

另一个系统，这里指的是Zope的对象库ZODB，由Zope公司[12]发展。这是一个简单的面向对象的数据库，存储在一个永久性的Python对象的函数中。然而，它缺乏任何推理的设置，因为所有的对象都存储在一个相当大的Python字典里。通过结合FLORA-2的推理引擎和ZODB的能力让使用Python中的知识库成为可能。

**二、将逻辑编程的概念引入到Python**

将逻辑编程的概念引入Python最需要解决的问题是逻辑变量，事实、规则和查询。在FLORA-2逻辑变量任何字出发，以一个大写字母或下划线表示(’ ’)。类似的方法是用Python创建一个类的对象命名变量在Python交互式会中，如下所示。

>>> X = f.Variable( ’X’ )

>>> X

<f.Variable instance at 0xb7dd854c>

>>> print X

X

>>> X == 27

>>> print X

X = 27

当分析事实、规则和查询我们可以得出这样的结论：他们是一种具有形式的框架的所有实例：

Head : −Body. (1)

: −Body. (2)

Head : − . (3)

（1）是一个规则，其头部是规则头部，其本体是规则本体（由头部表示：Head :- Body.）。（2）是查询的查询字符串（由- Body表示）。（3）是一个事实，头是实际的事实（由Head表示）。我们可以得出这样的结论：我们在把这些概念引进Python的时候只需要一个类即可，其行为在如下交互式会话所示（其中逻辑类论证作为后续研究）。

>>> x = f.Construct()

>>> print x

>>> h, b1, b2 = f.Bogus(), f.Bogus(),

f.Bogus()

>>> x & h

>>> x. class

<class f.Fact at 0xb7ae356c>

>>> del x

>>> x = f.Construct()

>>> x << b1 & b2

>>> x. class

<class f.Query at 0xb7ae365c>

>>> del x

>>> x = f.Construct()

>>> ( x & h ) << b1 | b2

>>> x. class

<class f.Rule at 0xb7ac5dac>

我们可以看到，构造类的实例是动态的，它可以根据其内容（例如它的头部和本体）来动态改变其类的事实、规则或查询。位移（“<<”），按位或（“|”）和按位与（“&”）被重写以使逻辑编程构造，并且支持'：'，'；'和'，'等运算符。这意味着一个逻辑编程结构：

X : −(Y, Z); W. (4)

在Python中被写作：

X << (Y&Z)|W. (5)

这是FLORA-2和Python相似的语法。除了这2类，还定义了一些额外的类来减少这类行为。

**三、翻译Python对象到框架模块**

另一个要完成的任务是将Python对象翻译成框架模块，以便在FLORA-2语法下了让他们进行推理。Python有一个特性，可以简化这种翻译过程，在下面的交互式会话可以看到。

>>> class a:

... def init ( self, b1 = 1, b2 = 2 ):

... self.b1 = b1

... self.b2 = b2

...

>>> y = a()

>>> dir( y )

[’ doc ’, ’ init ’, ’ module ’, ’b1’, ’b2’]

>>> y. dict

’b1’: 1, ’b2’: 2

>>> y. class

<class main .a at 0xb7a27b9c>

我们可以得出这样的结论：Python对象允许我们查询其内部的结构和内容。使用此功能可发展出多个递归算法，用于将Python对象转化成FLORA-2下的框架逻辑。例如下面所示的将y对象转化为FLORA-2：

>>> tr = py2f.py2f()

>>> tr.f2obj( y )

’py0xb7db4e2cpyobj main py :

pyapyclass main py ["b1"->1, "b2"->2]’

为了使我们的知识库中有唯一的对象名称，我们使用对象的内部内存地址和它的模块的一些附加信息。当一个对象被转化时，它将被自动存储在一个zobd对象库供以后检索和永久存储。之前有所争议的Python对象不是逻辑结构这一命题，被认为是事实。

**四、连接FLORA-2和Python**

为了使用由Python驱动的FLORA-2，需要开发一个接口。FLORA-2是建立在开源引擎XSB Prolog之上，由于该引擎由C开发、鼓起具有C接口[8]。简化封装和接口生成器（SWIG）被用来创建XSB和Python之间的接口（但可以很容易地扩展到其他语言如AllegroCL, C# - Mono, C# - MS .NET, CFFI, CHICKEN, CLISP, Guile, Java, Lua, MzScheme, Ocaml, Perl, PHP, Ruby, or或Tcl/Tk）。通过这样一个接口FLORA-2作为一个模块被XSB加载。额外的封装类来缓解XSB与Python，FLORA-2与Python之间的通信，如下面的交互式会话所示。

>>> f = interface.Flora2()

[FLORA2 specific output]

>>> f.consult( ’test’ )

[FLORA2 specific output]

>>> f.query( ’X:person[ Y -> Z ].’, [’X’,

’Y’, ’Z’] )

[{’Y’: ’age’, ’X’: ’mirko1’, ’Z’: ’40’},{’Y’:

’age’, ’X’: ’mirko2’, ’Z’: ’42’}]

>>> f.close query()

我们可以得出这样的结论：该接口可使用由Python驱动的 XSB和FLORA-2。编译Prolog和FLORA-2具体方案加载也被支持。查询可以直接发出而返回变量必须在Python列表作为第二个参数。查询的结果是Python列表元素，每个代表一个Python字典词典解决方案。每个字典的键提供返回变量，匹配值是从知识库返回的值。

**五、集成已实现功能**

在详述了该项结合的每个特定部分后，我们上述全部结合起来以助长Python的逻辑编程功能。前述的每一个部分都作为Python的一个模块，除此之外还需要定义一个新类以集成这些模块，这个类即是基于框架逻辑的提供存储和查询的工具。

为使存储持久，任何对象在被存储于知识库之前都会前辈存储于ZODB中，在此之后它将被翻译成FLORA-2语法并在FLTRA-2引擎中被加载。此外，FLORA-2代码保存在一个外部文件，所以声明稍后可以被重载。系统的这一过程对Python程序员是透明的。

在查询知识库时，我们既可以使用FLORA-2语法，也可以使用逻辑变量和逻辑框架创建特殊的查询对象。我们先创建一个构建并定义一个逻辑类，使用它来查询知识库。

>>> x = Construct()

>>> class query object( Logical ):

... def init ( self, a, b ):

... Logical. init ( self )

... self.a = a

... self.b = b

现在我们可以创建一个查询对象并修改它以适应我们的需求。请注意，使用逻辑变量的类型和类类型的属性，可以使用逻辑对象来查询任何类的对象。我们还可以通过在属性名的位置插入一个逻辑变量来重写属性的名称。

>>> q = query object( Variable( ’X’ ),

Variable( ’Y’ ) )

>>> x << q

>>> print x

?- py0xb7d6cdacpyobj main py :

pyquery objectpyclass main py [

"b"->Y, "a"->X].

>>> q. type = Variable( ’Z’ )

>>> print x

?- Z : pyquery objectpyclass main py [

"b"->Y, "a"->X].

>>> q. classtype = Variable( ’W’ )

>>> print x

?- Z:W[ "b"->Y, "a"->X].

>>> q. dict [ Variable( ’V’ ) ] = q. dict [

’a’ ]

>>> del q. dict [ ’a’ ]

>>> print x

?- Z:W[ "b"->Y, V->X].

>>> del q. dict [ ’b’ ]

>>> print x

?- Z:W[ V->X ].

查询知识库中的事例或规则必须存储在它的知识库中。下面创建一个简单的数据对象类，它包含了一个持久类作为子类。子类允许类的所有实例存储在ZODB之中。存储在知识库中的实例也在这个交互式会话中创建。

>>> class data object( Persistent ):

... def init ( self, a, b ):

... self.a = a

... self.b = b

...

>>> d1 = data object( 1, 2 )

>>> d2 = data object( 3, 4 )

现在可以创建知识库了，它作为Python的一个对象和其他对象没有区别，我们可以在这个库中导入数据对象。

>>> fb = FBase( ’my flbase’ )

[FLORA-2 specific output]

>>> fb.insert( d1 )

[FLORA-2 specific output]

>>> fb.insert( d2 )

[FLORA-2 specific output]

通过使用先前创建的查询对象，可以得到以下结果。注意，W变量返回的字符串使得ZODB无法存储类的对象（它们并不持续）。

>>> fb.query( x )

[{’X’: ’1’, ’Z’: < main .data object

object at 0xb7dde96c>, ’W’: ’class’},

{’X’: ’2’, ’Z’: < main .data object object

at 0xb7dde96c>, ’W’: ’class’}, {’X’:

’3’, ’Z’: < main .data object object at

0xb7c52aac>, ’W’: ’class’}, {’X’: ’4’, ’Z’:

< main .data object object at 0xb7c52aac>,

’W’: ’class’}]

这个简单的例子说明知识库对Python程序员是透明的，程序员只需要创建一个框架对象来将Python对象存储在知识中。

查询知识库时需要创建一个查询对象，使用逻辑对象和逻辑变量可以简易直观地完成这个任务。

在添加规则时，我们需要使用框架来创建并存储这些规则。这些规则与Prolog和FLORA-2是相同的，一些小的语法差异以用来保证正常Python代码的灵活性。

**六、结论与规划**

本文提出了一种框架逻辑的集成方法，主要支持FLORA- 2和动态的面向对象编程语言Python。这种整合相较于其他对Python脚本语言支持逻辑编程的尝试而言具有一定的优势。这些优势包括对Python程序员更直观的语法，完整的集成，在Python对象中进行推理，以及更持久的存储。这种集成使程序员可以容易地创建知识库，本体和语义网站解决方法。其他可能的用法包括自动应用程序生成、模块功能推理和自动化软件测试。

即使这个解决方案是向前发展的一步，它仍然需要进一步发展。未来的发展将包括更多的网络化系统，用以支持多个知识库的用户、客户端-服务器架构，为Python对象在网络上进行交换提供可能。一个可行的想法是将网络空间（NWS）[4]与已开发的系统连接起来以促进这样一个系统。

**致谢**

本文是在与Michael Kifer进行邮件探讨以及与我的导师Mirko Cˇ ubrilo进行讨论后写出的，在此我对他们的建议和想法表示感谢。

**参考文献**

[1] S. Berger, Pythologic – Prolog syntax in Python, on-line<http://aspn. activestate. com/ASPN/Cookbook/Python/Recipe/303057>, accessed: 21st November 2006.

[2] F. Coelho, Extending python with prolog syntax \*and resolution\*, on-line<http:// aspn. activestate.com/ASPN/Cookbook/Python/Recipe/360698>,accessed: 21st November 2006.

[3] M. Kifer, G. Lausen, J. Wu, Logical Foundations of Object-Orientedand Frame-Based Languages, Journal of the Association for Computing Machinery, May 1995.

[4] Nws-py.sourceforge.net, NetWork Spaces for Python, on-line<http://nws-py.sourceforge.net/>, accessed: 21st November2006.

[5] W. M. Petullo and later C. Zagrodnick, PyLog, on-line<http://www.gocept. com/open source software/Pylog>,accessed: 19th November 2006.

[6] M. Pilgrim, Dive into Python, on-line <http://diveintopython.org/>, accessed: 19th November 2006.

[7] Python.org, General Python FAQ, on-line

<http://www.python.org/doc/faq/general/>, accessed:24th March 2007.

[8] SWIG, Simplified Wrapper and Interface Generator, on-line<http://www.swig.org>, accessed: 19th November 2006.

[9] XSB, XSB API documentation, on-line <http://xsb.sourceforge.net/api/index.html>, accessed: 19th November 2006.

[10] G. Yang, M. Kifer, C. Zhao, FLORA-2: A Rule-Based Knowledge Representation

and Inference Infrastructure for the Semantic Web, In SecondInternational Conference on Ontologies, Databases and Applications ofSemantics (ODBASE), Catania, Sicily, Italy, November 2003.

[11] G. Yang, M. Kifer, C. Zhao, V. Chowdhary Flora-2 :User’s Manual, Version 0.94, (Narumigata), on-line <http://flora2.sourceforge.net>, accessed: 29th November 2006.

[12] Zope.org, ZODB, on-line <http://www.zope.org/Wikis/ZODB/FrontPage>, accessed: 21st November 2006.

附录2 程序代码

自定义函数集

#数据读取函数 参数为 路径 表格 数列 单位字符数 原格式输入

def readata(path,sheet,col,chop,str=0):

dt=[]

data=xlrd.open\_workbook(path)

table=data.sheet\_by\_index(sheet)

value=table.col\_values(col)

value.pop(0)

for val in value:

if str == 1:

dt.append(val)

elif chop==0:

dt.append(float(val.encode('utf-8')))

else:

dt.append(float(val[:-chop].encode('utf-8')))

return dt

#列表读取函数 参数为 路径 表格 数列

def readlist(path,sheet,col):

data=xlrd.open\_workbook(path)

table=data.sheet\_by\_index(sheet)

value=table.col\_values(col)

value.pop(0)

dataframe=[]

for i in value:

dt=[]

for j in i.split(';'):

dt.append(int(j))

dataframe.append(dt)

return dataframe

#去坏值函数 传入数组返回坏值所在索引

def badpop(lt):

i=0

p=[]

pop=[]

mean=np.mean(lt)

std=np.std(lt)

for x in lt:

if abs(mean-x)>std\*10:

p.append(i)

i+=1

for i in range(len(p)):

pop.append(p[i]-i)

return pop

#去零值函数 传入数组返回零值所在索引

def zeropop(lt):

i=0

p=[]

pop=[]

for x in lt:

if x==0:

p.append(i)

i+=1

for i in range(len(p)):

pop.append(p[i]-i)

return pop

#多变量相关性分析绘图函数 参数为 变量组 变量标签

def claly(datalist,labels):

n=len(labels)

fig,axes=plt.subplots(n,n,sharex=False,sharey=False)

fig.set\_size\_inches(16,10)

plt.subplots\_adjust(wspace=0,hspace=0)

for i in range(n):

for j in range(n):

axes[i,j].set\_xticks([])

axes[i,j].set\_yticks([])

if i==0:

axes[i,j].set\_title(labels[j],fontsize='large')

if j==0:

axes[i,j].set\_ylabel(labels[i],fontsize='large')

if i==j:

axes[i,i].hist(datalist[i],30,color='c',alpha=0.6)

else:

axes[i,j].scatter(datalist[j],datalist[i],marker='.',s=5,color='r',alpha=0.3)

#计算时间差函数 返回总秒数

def timediff(x,y):

h=int(x[:2].encode('utf-8'))-int(y[:2].encode('utf-8'))

m=int(x[3:5].encode('utf-8'))-int(y[3:5].encode('utf-8'))

s=int(x[-2:].encode('utf-8'))-int(y[-2:].encode('utf-8'))

if s<0:

s=60+s

m=m-1

if m<0:

m=m+60

h=h-1

return h\*3600+m\*60+s,h,m,s

#传入目录 返回全部文件路径

def getpath(dir):

path=[]

for file in os.listdir(dir):

if file[:7]=='vehicle':

path.append(os.path.join(dir,file))

return path

特征参量提取

import numpy as np

import matplotlib.pyplot as plt

import os,xlrd

def readata(path,sheet,col,chop,str=0):

dt=[]

data=xlrd.open\_workbook(path)

table=data.sheet\_by\_index(sheet)

value=table.col\_values(col)

value.pop(0)

for val in value:

if str == 1:

dt.append(val)

elif chop==0:

dt.append(float(val.encode('utf-8')))

else:

dt.append(float(val[:-chop].encode('utf-8')))

return dt

def zeropop(lt):

i=0

p=[]

pop=[]

for x in lt:

if x==0:

p.append(i)

i+=1

for i in range(len(p)):

pop.append(p[i]-i)

return pop

path=u'E:\\Project\\20150402\\vehicle-京B7Y223-20150416133513.xls'

energy=readata(path,0,19,4)

zero=zeropop(energy)

for i in range(len(zero)):

energy.pop(zero[i])

energycost=0

x=0

m=[]

n=[]

l=[]

charge=0

for i in range(len(energy)-1):

if energy[i]<energy[i+1] and x==0:

m.append(i)

x=1

if energy[i]>energy[i+1] and x==1:

n.append(i)

x=0

if len(m)!=len(n):

n.append(len(energy)-1)

for i in range(len(m)):

if n[i]-m[i]>20:

l.append([m[i],n[i]])

for i in range(len(l)):

charge+=energy[l[i][1]]-energy[l[i][0]]

energycost=energy[1]-energy[len(energy)-1]+charge

print u'%s全日能耗为%.1fkW•h'%(path[28:35],energycost)

def getpath(dir):

path=[]

for file in os.listdir(dir):

if file[:7]=='vehicle':

path.append(os.path.join(dir,file))

return path

path=getpath(u'E:\\Project\\20150402')

allcost=[]

for file in path:

energy=readata(file,0,19,4)

zero=zeropop(energy)

for i in range(len(zero)):

energy.pop(zero[i])

encost=0

for i in range(len(energy)-1):

if energy[i]>energy[i+1]:

encost+=(energy[i]-energy[i+1])

allcost.append(encost)

print allcost

# -\*- coding: utf-8 -\*-

%matplotlib inline

import numpy as np

import matplotlib.pyplot as plt

import os,xlrd

def readata(path,sheet,col,chop,str=0):

dt=[]

data=xlrd.open\_workbook(path)

table=data.sheet\_by\_index(sheet)

value=table.col\_values(col)

value.pop(0)

for val in value:

if str == 1:

dt.append(val)

elif chop==0:

dt.append(float(val.encode('utf-8')))

else:

dt.append(float(val[:-chop].encode('utf-8')))

return dt

def getpath(dir):

path=[]

for file in os.listdir(dir):

if file[:7]=='vehicle':

path.append(os.path.join(dir,file))

return path

maxtmp=[]

dir=u'E:\\project\\20150402\\'

for file in getpath(dir):

tmp=readata(file,0,12,2)

maxtmp.append(np.max(tmp))

for i in range(len(maxtmp)):

print u'%s 的电池最大温度值为%.0f℃'%(getpath(dir)[i][28:35],maxtmp[i])

相关性分析

# -\*- coding: utf-8 -\*-

%matplotlib inline

import numpy as np

import matplotlib.pyplot as plt

import os,xlrd

def correlation\_analysis(datalist,labels):

n=len(labels)

fig,axes=plt.subplots(n,n,sharex=False,sharey=False)

fig.set\_size\_inches(16,10)

plt.subplots\_adjust(wspace=0,hspace=0)

for i in range(n):

for j in range(n):

axes[i,j].set\_xticks([])

axes[i,j].set\_yticks([])

if i==0:

axes[i,j].set\_title(labels[j],fontsize='large')

if j==0:

axes[i,j].set\_ylabel(labels[i],fontsize='large')

if i==j:

axes[i,i].hist(datalist[i],30,color='c',alpha=0.6)

else:

axes[i,j].scatter(datalist[j],datalist[i],marker='.',s=20,color='r',alpha=1)

def readata(path,sheet,col,chop,str=0):

dt=[]

data=xlrd.open\_workbook(path)

table=data.sheet\_by\_index(sheet)

value=table.col\_values(col)

value.pop(0)

for val in value:

if str == 1:

dt.append(val)

elif chop==0:

dt.append(float(val.encode('utf-8')))

else:

dt.append(float(val[:-chop].encode('utf-8')))

return dt

def readlist(path,sheet,col): #读组电压 参数为 路径 表格 数列

data=xlrd.open\_workbook(path)

table=data.sheet\_by\_index(sheet)

value=table.col\_values(col)

value.pop(0)

dataframe=[]

for i in value:

dt=[]

for j in i.split(';'):

dt.append(int(j))

dataframe.append(dt)

return dataframe

def badpop(lt): #去坏值

i=0

p=[]

pop=[]

mean=np.mean(lt)

std=np.std(lt)

for x in lt:

if abs(mean-x)>std\*10:

p.append(i)

i+=1

for i in range(len(p)):

pop.append(p[i]-i)

return pop

def zeropop(lt):

i=0

p=[]

pop=[]

for x in lt:

if x==0:

p.append(i)

i+=1

for i in range(len(p)):

pop.append(p[i]-i)

return pop

def getpath(dir):

path=[]

for file in os.listdir(dir):

if file[:7]=='vehicle':

path.append(os.path.join(dir,file))

return path

path=getpath(u'E:\\Project\\20150402')

allcost=[]

for file in path:

energy=readata(file,0,19,4)

zero=zeropop(energy)

for i in range(len(zero)):

energy.pop(zero[i])

encost=0

for i in range(len(energy)-1):

if energy[i]>energy[i+1]:

encost+=(energy[i]-energy[i+1])

allcost.append(encost)

alltmp=[]

for file in path:

tmp=readata(file,0,12,2)

alltmp.append(sum(tmp)/len(tmp))

allrange=[]

for file in path:

vt=readlist(file,4,3)

rg=[]

for v in vt:

rg.append(max(v)-min(v))

allrange.append(max(rg))

allmile=[]

for file in path:

mile=readata(file,1,4,2)

zero=zeropop(mile)

for i in range(len(zero)):

mile.pop(zero[i])

cum=0

for i in range(len(mile)-1):

if mile[i]<mile[i+1]:

cum+=(mile[i+1]-mile[i])

allmile.append(cum)

datalist=[allcost,allrange,alltmp,allmile]

labels=['Energy\_cost','Voltage\_range','Temperature','Mileage']

for i in range(4): #去坏值

bad=badpop(datalist[i])

for j in range(len(bad)):

datalist[0].pop(bad[j])

datalist[1].pop(bad[j])

datalist[2].pop(bad[j])

datalist[3].pop(bad[j])

correlation\_analysis(datalist,labels)

一致性分析

# -\*- coding: utf-8 -\*-

%matplotlib inline

import numpy as np

import matplotlib.pyplot as plt

import os,xlrd

def readlist(path,sheet,col): #读组电压 参数为 路径 表格 数列

data=xlrd.open\_workbook(path)

table=data.sheet\_by\_index(sheet)

value=table.col\_values(col)

value.pop(0)

dataframe=[]

for i in value:

dt=[]

for j in i.split(';'):

dt.append(int(j))

dataframe.append(dt)

return dataframe

path=u'E:\\project\\20150402\\vehicle-京B7Y204-20150416094737.xls'

data=readlist(path,4,3)

ex=[] #各时间段电池组电压均值

voltage=[]

for value in data:

s=0

for i in value:

s+=i

e=float(s)/len(value)

ex.append(e)

for j in range(len(data[0])): #单个电池与均值差值总和

s=0

for i in range(len(data)):

s+=np.square((data[i][j]-ex[i]))

voltage.append(round((np.sqrt(s)),2))

i=0

overvt=[]

for vt in voltage:

i+=1

if (vt-sum(voltage)/len(voltage))>np.std(voltage)\*2:

overvt.append([vt,i])

fig=plt.figure()

fig.set\_size\_inches(16,18)

ax1=fig.add\_subplot(211)

ax2=fig.add\_subplot(212)

ax1.set\_title('Voltage Deviation of Every Battery',fontsize='large')

ax2.set\_title('Voltage Deviation Distribution',fontsize='large')

m=ax1.bar(range(len(voltage)),voltage,0.5,color='r',alpha=0.6)

n=ax2.hist(voltage,50,color='c',alpha=0.8)

print u'\n 与其他电池电压一致性较差的电池有：'

for i in overvt:

print u' 第%d号电池，其电压偏差的均方差为%.2f毫安'%(i[1],i[0])

温场分析

# -\*- coding: utf-8 -\*-

%matplotlib inline

import numpy as np

import matplotlib.pyplot as plt

import os,xlrd

def readlist(path,sheet,col): #读组电压 参数为 路径 表格 数列

data=xlrd.open\_workbook(path)

table=data.sheet\_by\_index(sheet)

value=table.col\_values(col)

value.pop(0)

dataframe=[]

for i in value:

dt=[]

for j in i.split(';'):

dt.append(int(j))

dataframe.append(dt)

return dataframe

def correlation\_analysis(datalist,labels):

n=len(labels)

fig,axes=plt.subplots(n,n,sharex=False,sharey=False)

fig.set\_size\_inches(16,10)

plt.subplots\_adjust(wspace=0,hspace=0)

for i in range(n):

for j in range(n):

axes[i,j].set\_xticks([])

axes[i,j].set\_yticks([])

if i==0:

axes[i,j].set\_title(labels[j],fontsize='large')

if j==0:

axes[i,j].set\_ylabel(labels[i],fontsize='large')

if i==j:

axes[i,i].hist(datalist[i],30,color='c',alpha=0.6)

else:

axes[i,j].scatter(datalist[j],datalist[i],marker='.',s=20,color='r',alpha=1)

def badpop(lt): #去坏值

i=0

p=[]

pop=[]

mean=np.mean(lt)

std=np.std(lt)

for x in lt:

if abs(mean-x)>std\*3:

p.append(i)

i+=1

for i in range(len(p)):

pop.append(p[i]-i)

return pop

path=u'E:\\project\\20150402\\vehicle-京B7Y204-20150416094737.xls'

tmp=readlist(path,3,3)

vt=readlist(path,4,3)

tmpstd=[]

tmprg=[]

vtstd=[]

vtrg=[]

for i in tmp:

tmpstd.append(np.std(i))

tmprg.append(max(i)-min(i))

for i in vt:

vtstd.append(np.std(i))

vtrg.append(max(i)-min(i))

datalist=[tmpstd,tmprg,vtstd,vtrg]

labels=['tmpstd','tmprange','vtstd','vtrange']

bad=badpop(datalist[2])

for j in range(len(bad)):

datalist[0].pop(bad[j])

datalist[1].pop(bad[j])

datalist[2].pop(bad[j])

datalist[3].pop(bad[j])

correlation\_analysis(datalist,labels)

驾驶行为分析

# -\*- coding:utf-8 -\*-

%matplotlib inline

import os,xlrd

import numpy as np

def readata(path,sheet,col,chop,str=0): #参数为 路径 表格 数列 单位字符数 原格式输入

i=0

dt=[]

data=xlrd.open\_workbook(path)

table=data.sheet\_by\_index(sheet)

value=table.col\_values(col)

for val in value:

i+=1

if i>1:

if str == 1:

dt.append(val)

elif chop==0:

dt.append(float(val.encode('utf-8')))

else:

dt.append(float(val[:-chop].encode('utf-8')))

return dt

def timediff(x,y): #计算时间差 x-y

h=int(x[:2].encode('utf-8'))-int(y[:2].encode('utf-8'))

m=int(x[3:5].encode('utf-8'))-int(y[3:5].encode('utf-8'))

s=int(x[-2:].encode('utf-8'))-int(y[-2:].encode('utf-8'))

if s<0:

s=60+s

m=m-1

if m<0:

m=m+60

h=h-1

return h\*3600+m\*60+s,h,m,s

if \_\_name\_\_ == "\_\_main\_\_":

try:

path=u'E:\\Project\\20150402\\vehicle-京B7Y238-20150416130635.xls'

speed = readata(path,1,12,4)

time = readata(path,1,15,0,1)

n=10

stoptime=[]

turn=1

for i in range(len(speed)):

if (speed[i]==0)and(turn==1):

begin=time[i]

turn=0

elif (speed[i]!=0)and(turn==0):

end=time[i]

turn=1

if timediff(end,begin)[0]>=60\*n:

stoptime.append([begin,end,timediff(end,begin)[1],timediff(end,begin)[2],timediff(end,begin)[3]])

print u'%s 超过%d分钟停车时间：'%(path[28:35],n)

for j in range(len(stoptime)):

print u'从%s到%s，共%d小时%d分%d秒'% (stoptime[j][0],stoptime[j][1], stoptime[j][2],stoptime[j][3],stoptime[j][4])

except:

print 'Error'

# -\*- coding: utf-8 -\*-

%matplotlib inline

import numpy as np

import matplotlib.pyplot as plt

import os,xlrd

def readata(path,sheet,col,chop,str=0): #参数为 路径 表格 数列 单位字符数 原格式输入

i=0

dt=[]

data=xlrd.open\_workbook(path)

table=data.sheet\_by\_index(sheet)

value=table.col\_values(col)

for val in value:

i+=1

if i>1:

if str == 1:

dt.append(val)

elif chop==0:

dt.append(float(val.encode('utf-8')))

else:

dt.append(float(val[:-chop].encode('utf-8')))

return dt

def badpop(lt): #输入list输出坏值所在index

i=0

p=[]

pop=[]

mean=np.mean(lt)

std=np.std(lt)

for x in lt:

if abs(mean-x)>std\*10:

p.append(i)

i+=1

for i in range(len(p)):

pop.append(p[i]-i)

return pop

def zeropop(lt): #去零值

i=0

p=[]

pop=[]

for x in lt:

if x==0:

p.append(i)

i+=1

for i in range(len(p)):

pop.append(p[i]-i)

return pop

path=u'E:\\Project\\20150402\\vehicle-京B7Y233-20150416134115.xls'

E=readata(path,2,3,0)

N=readata(path,2,4,0)

time=readata(path,2,7,0,1)

m='none' #分时段

n='none'

for i in range(len(time)):

if m=='none' and int(time[i][:2])==12:

m=i

if n=='none' and int(time[i][:2])==18:

n=i

break

bad=zeropop(E)

for j in range(len(bad)):

E.pop(bad[j])

N.pop(bad[j])

fig=plt.figure(111)

fig.set\_size\_inches(18,18)

ax=fig.add\_subplot(2,2,1)

ax1=fig.add\_subplot(2,2,2)

ax2=fig.add\_subplot(2,2,3)

ax3=fig.add\_subplot(2,2,4)

ax.set\_xticks([])

ax.set\_yticks([])

ax1.set\_xticks([])

ax1.set\_yticks([])

ax2.set\_xticks([])

ax2.set\_yticks([])

ax3.set\_xticks([])

ax3.set\_yticks([])

plt.subplots\_adjust(wspace=0,hspace=0)

ax.scatter(E[:m],N[:m],marker='.',s=10,color='b',alpha=1)

ax.scatter(E[m+1:n],N[m+1:n],marker='.',s=10,color='r',alpha=1)

ax.scatter(E[n+1:],N[n+1:],marker='.',s=10,color='k',alpha=1)

ax1.scatter(E[:m],N[:m],marker='.',s=20,color='b',alpha=1)

ax2.scatter(E[m+1:n],N[m+1:n],marker='.',s=20,color='r',alpha=1)

ax3.scatter(E[n+1:],N[n+1:],marker='.',s=20,color='k',alpha=1)