环境

项目依赖的第三方库都写在项目的根目录中的requirements.txt文件中，使用python的包管理工具"pip"可以很方便的安装这些依赖。

```
python -m pip install -r requirements.txt
```

> *如果没有安装pip，可以在这里下载。*

系统使用的数据库为sqlite和redis，其中sqlite用于存储用户信息以及数据集等需要持久化存储的数据，sqlite是python内置的数据库，因此我们无需安装它。而redis只是用来存储数据分析结果（一组序列化的python对象），可视化系统从redis中取出序列化的python对象，通过反序列化再将它加载为python对象。因此系统还依赖redis数据库。这里建议在linux环境下搭建redis数据库。我使用的是debian的linux发行版，可以在这里下载。其中debian-9.9-amd64-netinst.iso是x86架构下的64位纯命令行linux系统，它占用的资源很小，比较有利于开发。

系统简介

项目的目录结构是：

```
mine_sys
|-- mine
|
|-- super_dash
|
|-- mine_sys
|
|-- manage.py
|
|-- db.sqlite3
```

该项目包含两个系统，分别是 super_dash和mine，super_dash是可视化系统，主要负责结果展示，mine是数据分析系统，主要负责算法执行。

系统采用的技术栈为Python3.7+Django，Django是一个使用python实现的web框架，Django提供了一个manage.py脚本文件，改脚本的功能包括添加管理员账户，迁移数据库，启动调试服务器等指令。

如何启动系统

当第一部分的依赖都安装好后，便可以启动系统，通过使用manage.py，使用命令

```
python manage.py runserver localhost:8000
```

就可以启动系统，并且通过浏览器访问"http://localhost:8000"就可以访问可视化系统。

如果出现 ConnectionError异常，很有可能是redis服务未启动，或设置的redis服务器地址无效。如果确保了redis服务器已经正常工作，请检查super_dash/settings.py设置文件中REDIS字段的值是否设置正确。

> *tip: linux 下的redis服务器启动时默认监听的网卡是本地回环，因此外部是无法访问的，可以在/etc/redis/redis.conf文件中修改监听网卡。*

数据分析的配置

数据分析系统可以对上传的数据集进行分析，为了更好的控制分析过程，这里采用了配置信息的形式对这一流程进行手动调控，每个算法插件通过使用系统提供的signal来注册它自己的配置信息。需要注意的一点是配置信息中的chart字段，该字段并不是通过算法插件注册的，而是提供给可视化系统使用的，意思是该数据集需要展示何种图表，包括散点图、饼状图、柱状图、决策树。由于数据分析系统的分析结果对象和echart图表中的图表是对应的，因此可以做成系统自动识别使用何种图表来展示，但是目前系统还未实现该功能。

```
{
  "n_clusters": 2,
  "axis": [ "opp_point"],
  "chart": ["scatter"]
}
```

算法插件如何与系统对接

驱动

算法与系统对接的部分，我们在这里可以称之为驱动，也就是说，我们需要提供给数据分析系统一个驱动程序（函数）来使数据分析系统可以调用算法插件。

那么如何编写数据分析系统的插件呢？数据分析系统默认通过调用驱动的entry函数，来调用算法，当然函数名还可以通过配置文件的function_name字段指定。entry函数遵循以下协议：

- 接收两个入参，第一个入参是数据集文件路径，第二个入参是一个python的字典对象，存储了配置信息。
- 返回值是一个二元组，第一个元素是可迭代对象，里面存储了一个或多个继承自mine.models.Model类的数据模型对象，该对象存储了数据分析结果，并且该对象应当与echart中的图表数据相对应。第二个元素是一个可调用对象，通常是一个函数，用于“预测”。如果算法没有预测功能，那么该值应当是None。

注册配置信息格式

在驱动中还进行了配置信息的注册，首先根据JsonSchema语法编写一个jsonschema，然后调用super_dash.signals.register_jsonschema函数将该jsonschema注册即可。

这是一个驱动的例子

```
import pandas

# c45_pandas是算法程序，将真正的算法程序导入进来
from . import c45_pandas

# 这是一些数据模型类，将使用上面导入的算法进行数据分析，并转储到这些对象中去
from mine.algorithm.models import \
    (DecisionTree, Node, Line, PieGraph, ThreeDHistogram)

# 这是一个函数，用于注册jsonschema
from super_dash.signals import register_jsonschema

# jsonschema语法，定义了配置信息的格式
config_schema = {
    "title": "C45 algorithm config",
    "description": "",
    "type": "object",
    "properties": {
        # 配置信息的字段 preprocess_columns,类型是数组，每个元素的类型是string类型。
        "preprocess_columns": {
            "type": "array",
            "items": {
                "type": "string"
            }
        },
        # 配置信息的字段 result_column,类型是字符串。
        "result_column": {
            "type": "string"
        },
        # 配置信息的字段 keys,类型是数组，每个元素的类型是string类型。
        "keys": {
            "type": "array",
            "items": {
                "type": "string"
            }
        }
    },
    # 定义了result_column是必须字段。
    "required": ["result_column"]
}

# 调用函数讲jsonschema注册给可视化系统，可视化系统在存储用户输入的配置信息的时候会使用该
jsonschema对象对数据进行校验
register_jsonschema.send(sender=None, schema=config_schema,
                         import_path="mine.algorithm.C45.interface")

# 一个工具函数，将上面导入的算法返回的结果封装成一棵决策树。
def build_tree(old, new):
    for line in old.lines:
        new_line = Line(new, Node(line.child.name))
```

```
            new_line.name = line.name
            new.add_line(new_line)
            build_tree(line.child, new_line.child)


    # 驱动入口函数，在这里进行了c45算法的调用，并将返回结果封装成数据模型对象，然后返回一个二元组。
    def entry(ds, cfg):
        ds = pandas.read_csv(ds)
        res_col = cfg.get('result_column')
        c45_pandas.preprocess(ds, cfg.get('preprocess_columns'))
        res = c45_pandas.mine_c45(ds, cfg.get('result_column'),
                                  keys=cfg.get('keys'))
        models = []

        tree = DecisionTree(cfg.get('name'))
        tree.data = Node(res.name)
        build_tree(res, tree.data)
        models.append(tree)

        for index in ds.columns:
            pie = PieGraph(index)
            vc = ds[index].value_counts()
            for series_index in vc.index:
                pie.add({"name": series_index, "value": int(vc[series_index])})
            models.append(pie)

        for index in ds.columns:
            if index == res_col:
                continue
            threeD_histogram = ThreeDHistogram(index)
            sub_group = ds.groupby(index)
            for k, v in sub_group.groups.items():
                sub_ds = ds.loc[v]
                sub_ds_sub_group = sub_ds.groupby(res_col)
                for k1, v1 in sub_ds_sub_group.groups.items():
                    threeD_histogram.add((k, k1, len(v1)))
            models.append(threeD_histogram)
        return models, None
```

数据分析系统如何找到驱动

数据分析系统调用驱动，驱动负责调用算法，通过这样的方式实现了数据分析系统与算法插件的对接，那么自己实现了一个算法，并且实现了它的驱动，如何让数据分析系统发现这个插件并使用呢？通过在可视化系统的配置界面中写入驱动的路径（python的点分路径）之后数据分析系统会尝试动态导入该该路径指向的驱动程序，以此来发现驱动。

结语

系统开发仓促，仍有很多不完善的地方，比如数据模型不够全面，展示的图表类型有限，系统无法自动根据数据模型对象来展示对应的图表等等问题。但系统是具备可拓展能力的，上述需求并非无法实现，我自以为代码具有一定的可读性和可维护性，望读者可以通过以该系统为引，完善或实现自己的数据分析系统。