MLPractice-01 神舟优车订单预测（CSV）

讲义

1. 任务描述

在网约车业务中，由于系统中的专车资源是有限的，而乘客资源也是有限的，但是这两类有限的资源是各自散布在一个城市的若干区域中，并且由于城市的不同区域的功能不同，乘客资源散布在各个区域的密度有所差别，如何使专车的散布区域密度尽量符合乘客散布密度是其中很重要的一个需求。因此我们提出这样的需求，预测某个时间段内从区域A出发到区域B的订单数量。

现在给定2017年7月份和8月份的部分订单数据作为训练数据，预测8月份规定时间段内的数据。

2. 数据集

2.1. 训练集

我们以七月份的订单数据作为训练集，其中包含的各个字段的含义如下：

字段名称	字段含义	数据示例
id	订单id的hash值	583411b46a31bcc5d12d4402c928a146
driver_id	司机id的hash值（未出行成功则为一个特殊司机编号）	3e69e17a6e5a726fe44d71896bee4f32
menber_id	乘客id的hash值	6b4d6e4992191fe96b9f27921520d551
create_date	订单创建日期	2017-07-01

字段名称	字段含义	数据示例
create_hour	订单创建时间（0-23小时）	00
status	订单状态：0是未预约成功，1是预约取消，2是出行成功	2
estimate_money	预估行程金额（元）	140.00
estimate_distance	预估行程距离（米）	20099.00
estimate_term	预估行程时间（分钟）	18.00
start_geo_id	起点区域id的hash值	6d7827e8dcfa09497954a31e6f7e6ee6
end_geo_id	终点区域id的hash值	85e49ded1fa70a7bfa01ab0212a6e538

见附件：train_July.csv

2.2. 测试集

我们以八月份第一周内随机抽取的若干条数据作为测试集。测试集字段含义如下所示：

字段名称	字段含义	数据示例
test_id	该条测试用例的id	1
start_geo_id	起点区域id的hash值	6d7827e8dcfa09497954a31e6f7e6ee6
end_geo_id	终点区域id的hash值	85e49ded1fa70a7bfa01ab0212a6e538
create_date	订单创建日期	2017-08-01
create_hour	订单创建时间（0-23小时）	01

见附件：test_Aug.csv

2.3. 其他数据

2.3.1. 区域内设施数据

包含某一区域内公共设施的数量描述，其中第一列为区域id的hash值（与训练及测试数据中的区域对应），后面分别为各类设施的类型及数量，例如：

```
3d99665144344fc090b5b7450ffe72f5,加油站,4,超市,43,住宅区,151,地铁站,4,公交站,36,咖啡厅,22,中餐厅,597,ATM,59,写字楼,47,酒店,45
```

表示该区域内有加油站4个，超市43个，等等。

见附件：poi.csv

2.3.2. 天气数据

训练集与测试集所涉及的时间段内的天气情况（有极少部分缺失）。各个字段含义如下：

字段名称	字段含义	数据示例
date	日期（精确至半小时）	2017-7-10:30
text	天气现象文字	晴
code	天气现象代码	1
temperature	温度	29
feels_like	体感温度	28
pressure	气压	998
humidity	相对湿度	62
visibility	能见度	9.3
wind_direction	风向文字	南
wind_direction_degree	风向角度，范围0-360，0为正北，90为正东，180为正南，270为正西	200
wind_speed	风速	8
wind_scale	风力等级	2

见附件：weather.csv

3. 解决过程

在尝试解决一个问题之前，首先需要明确我们有哪些资源，然后考虑如何利用这些资源解决问题，因此第一步是将我现有的数据，特别是各个表头，表之间的关系梳理一下，然后再考虑要解决什么问题。

3.1. 目标是什么？

首先，该问题是一个回归问题，即根据测试集给定的字段，包括起点、重点、日期、小时来预测对应的订单数量。初一看应该是以这四个字段作为输入，作为预测的条件。那么对应的训练数据也应该与这四个字段对应。因此我们到训练集中去找到这四个对应的字段，看看有没有。

我们可以很容易的就在七月份的订单表中找到如下的几个字段，与测试集的输入相对应：

id	create_date	create_hour	start_geo_id	end_geo_id
d91bda...	2017-08-01	12	25998a0...	4176d25...
844444...	2017-08-01	12	d42290f...	7469adf...
67eb5b...	2017-08-01	14	2da36a4...	2da36a4...
28cb07...	2017-08-01	14	56b85fc...	56b85fc...

但是，有一个问题。我们知道回归问题作为监督学习的一种，训练集数据除了有输入特征之外，还要有标签，对应到这个问题就应该是特定的时间段内，对应的一组起点和终点的订单数量。但是在这个测试集中，我们并没有找到一个字段是用来描述某个时间段内的订单数量的字段，那怎么办呢？

考虑下，这个数据集中记录的是七月份所有的订单，每个订单一条数据，也就是说这个标签需要我们自己统计一下某个时间段内，从区域A到区域B的订单数量。

根据这个思路我们应该已经能够获取到训练集了。

3.2. 真正参与计算的特征？

当我们拿到看似完整的训练集之后，开始着手训练模型时，会发现一个问题，训练集中的

这四个特征，与我们需要计算的目标之间存在什么直接的关联么？

比如起点这个字段，是一个地区的编号，应该与订单数量是没有直接的关联的，而是这个地区背后所表示的含义与订单的数量有关系。比如说这个地区内是否有很多住宅区？是否有很多写字楼？是否有很多商业区等等。

比如日期这个字段，日期直接与订单数量存在什么必然的联系么？一般不会，而是日期所表示的各方面含义与订单数量有关系。比如某个日期对应的天气状况。

我们需要将这几个表面特征转换为其背后所表示的各方面数据，然后分析这些数据与订单数量之间的关系，以此来训练模型。

3.2.1. 地区所表示的含义

根据之前的思路，现在要做的是将训练集中的地区对应的特征数据表示出来，首先我们将训练集读取到内存中。

3.2.1.1. 读取训练集

首先利用`csv`模块将数据从CSV文件中读取出来，在读取训练集时，将训练集读取成如下结构：

```
[
    {
        "order": {
                "id":"c538ad66d710f99ad0ce951152da36a4",
                "driver_id": ...,
                "member_id": ...,
                "create_date" ...,
                "create_hour" ...,
                ...
                "start_geo_id": ...,
                "end_geo_id" ...:
            }
    }, {
        "order": {
                "id":"546e5411dd4bf116e28d3a7f7995e255",
                "driver_id": ...,
                ...
                "end_geo_id" ...:
            }
        }
]
```

最外层是一个列表，列表中的每个值是一个字典，将这个订单信息存放在“order”下面，这样设计是为了便于后续将地区公共设施和天气等内容组装在一起。接下来读取订单信息：

```
def TRAIN_read(fileName):
    csv_reader = csv.reader(open(fileName, encoding='utf-8'))
    csv_reader.__next__()
    trainSet = list()
    trainSetOrder = set()

    for row in csv_reader:
        item = {}
        item["order_id"] = row[0]
        item["driver_id"] = row[1]
        item["member_id"] = row[2]
        item["create_date"] = row[3]
        item["create_hour"] = row[4]
        item["status"] = row[5]
        item["estimate_money"] = row[6]
        item["estimate_distance"] = row[7]
        item["estimate_term"] = row[8]
        item["start_geo_id"] = row[9]
        item["end_geo_id"] = row[10]

        if item["order_id"] not in trainSetOrder:
            el = dict()
            el["order"] = item
            trainSetOrder.add(item["order_id"])
            trainSet.append(el)
        else:
            print(item)

    return trainSet
```

在这段读取订单信息的代码中，我们可以看到在每次添加订单信息时会有一个检查：

```
if item["order_id"] not in trainSetOrder:
    ...
```

目的是为了防止重复的订单，我们以一个订单号视作一条独立的订单，因此要把重复的订单剔除掉。这个函数接受一个参数是训练集所在的文件名称。

3.2.1.2. 读取公共设施数据

同样的思路读取公共设施数据，将公共设施数据同样构建成字典的形式：

```
{
    "687ea595a01fdc18ad438f0d0a325102":{
        "rid": "687ea595a01fdc18ad438f0d0a325102",
        "petrol": ...,
        "market": ...,
        "uptown": ...,
        "metro": ...,
        "bus": ...,
        "cafe": ...,
        "restruant": ...,
        "atm": ...,
        "office": ...,
        "hotel": ...,
    },
    "f80c4ceeb36264b42e34d6c4c2cb9b4c":{
        "rid": "f80c4ceeb36264b42e34d6c4c2cb9b4c",
        ...
    }
}
```

代码：

```
def POI_read():
    csv_reader = csv.reader(open('poi.csv', encoding='gbk'))

    poi = dict()

    for row in csv_reader:
        item = dict()
        item["rid"] = row[0]
        item["petrol"] = row[2]
        item["market"] = row[4]
        item["uptown"] = row[6]
        item["metro"] = row[8]
        item["bus"] = row[10]
        item["cafe"] = row[12]
        item["restruant"] = row[14]
        item["atm"] = row[16]
        item["office"] = row[18]
        item["hotel"] = row[20]
```

```
19.
20.             if item["rid"] not in poi.keys():
21.                 poi[item["rid"]] = item
22.
23.         return poi
```

3.2.1.3. 连接公共设施与测试集

在将训练集和公共设施数据都读取之后，需要根据每个订单中起点和终点的地区编号，组合相关的设施情况。连接后形成结果如下：

```
1.  [
2.      {
3.          "order": {
4.              ...
5.          },
6.          "startpoi": {
7.              'rid': '59e9ad69a9b98a75107b2a4ba96cd765',
8.              'petrol': '6',
9.              'market': '26',
10.             'uptown': '236',
11.             'metro': '6',
12.             ...
13.             'hotel': '141'
14.         },
15.         "endpoi": {
16.                 ...
17.         }
18.     }, {
19.         "order": {
20.                 ...
21.         },
22.         "startpoi": {
23.             ...
24.         },
25.         "endpoi": {
26.                 ...
27.         }
28.     }
29. ]
```

代码如下：

```
def combine_POI(train, poi):
    for i in range(len(train)):
        train[i]["startpoi"] = poi.get(train[i].get("order").get("start_geo_id"))
        train[i]["endpoi"] = poi.get(train[i].get("order").get("end_geo_id"))
    return train
```

3.2.2. 日期所表示的天气

根据之前的思路，现在要做的是将训练集中的日期对应的天气数据表示出来，此处采用与之前相同的思路将天气信息与订单信息整合。

3.2.2.1. 读取天气数据

为了便于后续将天气数据和训练集合并，此处我们设计的读取天气的数据格式为：

```
{
    "2017-01-01: {
        "0": [{},{}],
        "1": [],
        ...
        "23: []
    },
    "2017-01-02: {
        "0": [],
        "1": [],
        ...
        "23: []
    }
}
```

我们以一个字典存放该数据，以日期作为key，由于天气数据中的日期格式与训练集中的日期格式不同，所以在读取的时候进行格式转换。

然后每一天分为24个小时，每个小时内则有两条天气数据，使用列表存放。代码为：

```
def Weather_read():
    csv_reader = csv.reader(open('weather.csv', encoding='utf-8'))
    csv_reader.__next__()
    weather = dict()

    for row in csv_reader:
```

```
            item = {}
            date = row[0].split()
            # 将 2017-1-2 转为 2017-01-02 的形式
            item["date"] = stringToDate(date[0]).strftime("%Y-%m-%d")
            hour = date[1].split(":")
            item["hour"] = hour[0]
            item["min"] = hour[1]

            item["text"] = row[1]
            item["code"] = row[2]
            item["temperature"] = row[3]

            if len(row[4]) > 0:
                item["feels_like"] = row[4]
            else:
                item["feels_like"] = '-1'

            item["pressure"] = row[5]
            item["humidity"] = row[6]

            if len(row[7]) > 0:
                item["visibility"] = row[7]
            else:
                item["visibility"] = '-1'

            item["wind_direction"] = row[8]
            item["wind_direction_degree"] = row[9]
            item["wind_speed"] = row[10]
            item["wind_scale"] = row[11]

            if item["date"] not in weather.keys():
                weather[item["date"]] = dict()

            if item["hour"] not in weather.get(item["date"]).keys():
                weather[item["date"]][item["hour"]] = list()

            weather[item["date"]][item["hour"]].append(item)

        return weather
```

3.2.2.2. 连接数据

接下来将天气数据与训练集进行连接。从训练集中获取日期和小时，然后从天气数据中获取对应的数据，进行合并：

```
def combine_weather(train, weather):
    for i in range(len(train)):
        train[i]["weather"] = weather[train[i]["order"]["create_date"]]
[str(int(train[i]["order"]["create_hour"]))]
    return train
```

3.3. 如何得到标签？

经过上述一系列的处理之后，我们得到了一个有订单、区域公共设施和天气构成的数据集合，但是这个集合是有一定的复杂结构的，不同于我们常规的数据集组织方式，因此我们需要从中将需要的字段提取出来构成一个常规的训练集。

同时我们还需要统计订单，形成标签。此处需要注意的是，按照我们之前的思路是通过统计订单（即某个时间段内从某个区域到另一个区域的订单数量）。但现在我们将区域和时间分别转换成其背后所表示的含义了，那么我们统计订单数量时，所依据的就应该是这些关联信息的组合，而不再是时间、起点和重点的组合了。

我们以如下的例子进行说明：

订单编号	日期	小时	起点	终点
1	2017-07-01	1	hashcode01	hashcode02
2	2017-07-01	1	hashcode01	hashcode02
3	2017-07-01	1	hashcode02	hashcode03
2	2017-07-01	2	hashcode01	hashcode02
3	2017-07-01	2	hashcode02	hashcode03

对于这样一组数据，如果按照时间、起点和终点进行统计订单数量的话，其结果应该是如下：

日期	小时	起点	终点	数量
2017-07-01	1	hashcode01	hashcode02	2
2017-07-01	1	hashcode02	hashcode03	1
2017-07-01	2	hashcode01	hashcode02	1

日期	小时	起点	终点	数量
2017-07-01	2	hashcode02	hashcode03	1

但我们将时间、地区分别转换成其背后所表示的含以后：

订单编号	温度	风速	起点地铁	起点住宅	终点地铁	终点住宅
1	28	3	2	25	3	10
2	29	5	0	15	3	5
3	28	3	2	25	3	10
4	29	5	0	15	3	5
5	28	3	2	25	3	10

统计订单数量时，应该是以转换后的特征的组合进行统计：

温度	风速	起点地铁	起点住宅	终点地铁	终点住宅	数量
28	3	2	25	3	10	3
29	5	0	15	3	5	2

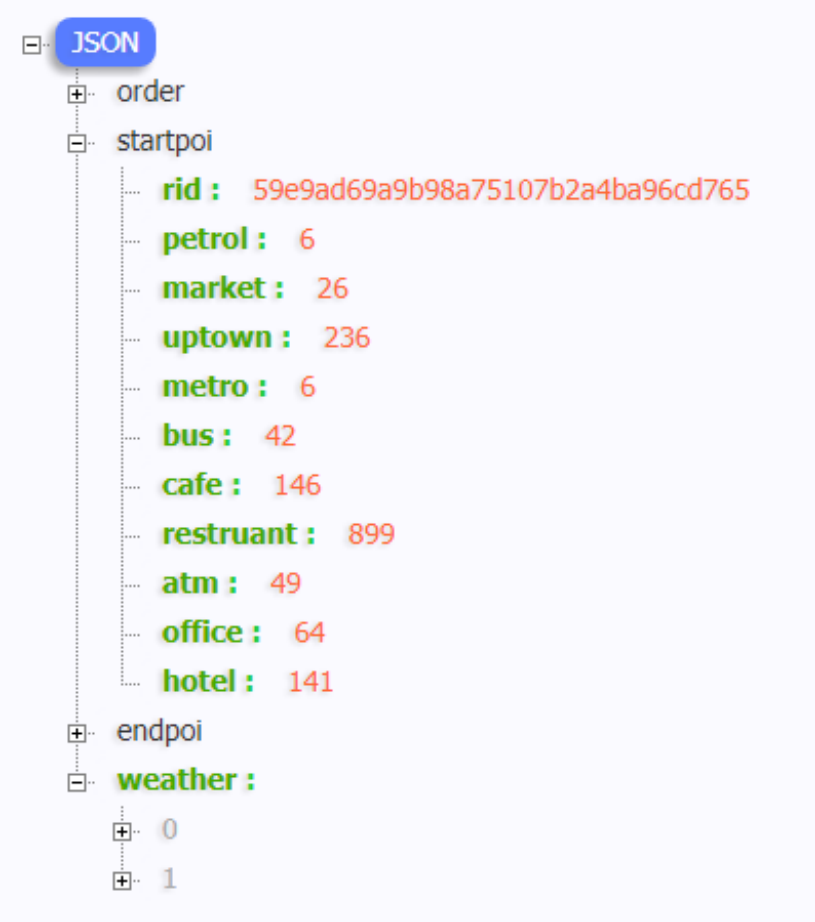
3.3.1. 提取数据并统计

现在我们从之前组合好的数据中提取所需要的数据，目前整合的数据结构最外层是一个列表，列表中每个元素的结构如下所示：

每条数据中包含四个元素，其中“order”中是订单原本的信息，“startpoi”是起点区域的设施信息，“endpoi”是终点区域内的设施信息，而“weather”则是该订单时间段内的

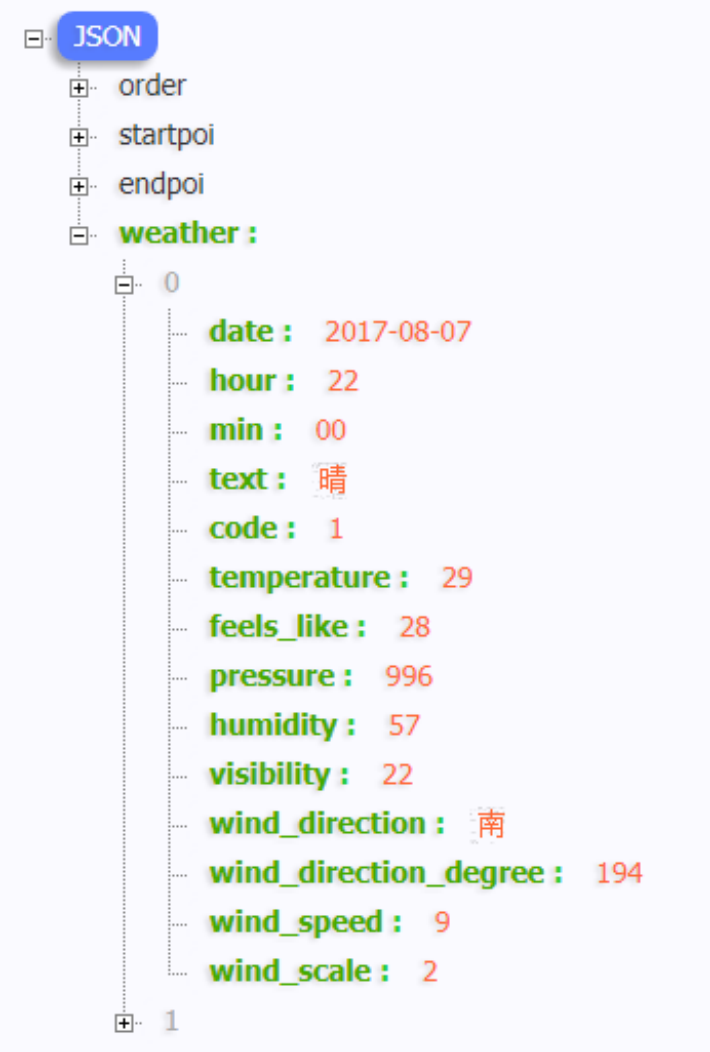
天气信息，因为每个小时内包含两条天气信息，因此是一个数组。下面具体看一看每个元素内部的结构，首先是订单信息，其结构如下：

在提取信息时，订单信息中的内容并不需要，我们不予关注，下面看看设施信息：



设施分为起点和终点区域的设施，分为两组，其内容部结构是一样的。在提取信息时，这两部分的信息都需要提取。

然后是提取天气信息，由于天气信息有两条，在提取的时候可以将两条天气信息取平均值，或者任意取一条，此处我们默认提取第一条。



在提取和统计数据时，如果出现设施或天气数据为空的情况时，默认处理是认为该条数据为无效数据，直接跳过去，代码如下：

```
def extract_statis(train):
    trainDic = dict()
    for item in train:
        try:
            row = list()
            row.append(item["startpoi"]["petrol"])
            row.append(item["startpoi"]["market"])
            row.append(item["startpoi"]["uptown"])
            row.append(item["startpoi"]["metro"])
            row.append(item["startpoi"]["bus"])
            row.append(item["startpoi"]["cafe"])
            row.append(item["startpoi"]["restruant"])
            row.append(item["startpoi"]["atm"])
            row.append(item["startpoi"]["office"])
```

```
            row.append(item["startpoi"]["hotel"])

            row.append(item["endpoi"]["petrol"])
            row.append(item["endpoi"]["market"])
            row.append(item["endpoi"]["uptown"])
            row.append(item["endpoi"]["metro"])
            row.append(item["endpoi"]["bus"])
            row.append(item["endpoi"]["cafe"])
            row.append(item["endpoi"]["restruant"])
            row.append(item["endpoi"]["atm"])
            row.append(item["endpoi"]["office"])
            row.append(item["endpoi"]["hotel"])

            row.append(item["weather"][0]["date"])
            row.append(item["weather"][0]["hour"])
            row.append(item["weather"][0]["min"])
            row.append(item["weather"][0]["code"])
            row.append(item["weather"][0]["temperature"])
            row.append(item["weather"][0]["feels_like"])
            row.append(item["weather"][0]["pressure"])
            row.append(item["weather"][0]["humidity"])
            row.append(item["weather"][0]["visibility"])
            row.append(item["weather"][0]["wind_direction_degree"])
            row.append(item["weather"][0]["wind_speed"])
            row.append(item["weather"][0]["wind_scale"])
            key = "@".join(row)

            if key not in trainDic.keys():
                trainDic[key] = dict()
                trainDic[key]["features"] = row
                trainDic[key]["count"] = 0

            trainDic[key]["count"] += 1
        except:
            continue

    trainList = list()

    for k in trainDic.keys():
        trainDic[k]["features"].append(trainDic[k]["count"])
        trainList.append(trainDic[k]["features"])

    columnNames = ["s_petrol", "s_market", "s_uptown", "s_metro", "s_bu
s", "s_cafe", "s_restruant",
                   "s_atm", "s_office", "s_hotel",
```

```
                    "e_petrol", "e_market", "e_uptown", "e_metro",
    "e_bus", "e_cafe", "e_restruant",
                    "e_atm", "e_office", "e_hotel",
                    "date", "hour", "min", "code", "temperature", "feels_
    like", "pressure", "humidity",
                    "visibility", "wind_direction_degree", "wind_speed",
    "wind_scale", "count"
                    ]

        return trainList, columnNames
```

最终得到的训练集如下：

```
   s_petrol  s_market  s_uptown  s_metro  s_bus  s_cafe  s_restruant  s_atm  s_office  \
0         1        39       222        5     35      28          649     59        93
1         4        61       233        6     40     102          900    136       162
2         3        63       272        7     49     334          898    189       310
3         0        39       316        3     43      33          811    106       117
4         2        28       139        2     55      21          525     50        37

   s_hotel  ...  code  temperature  feels_like  pressure  humidity  visibility  \
0       79  ...     4           30          30      1004        68         4.7
1      279  ...     4           30          30      1004        68         4.7
2      206  ...     4           32          31      1003        62         7.8
3      133  ...     4           32          31      1003        62         7.8
4       61  ...     4           32          32      1002        58        10.7

   wind_direction_degree  wind_speed  wind_scale  count
0                    233        3.24           1      1
1                    233        3.24           1      3
2                    174       13.68           3      7
3                    174       13.68           3      5
4                     98        9.72           2      1
```

3.4. 训练模型

如此，经过上面一系列的步骤，我们最终得到了可以用来训练模型的训练集，接下来利用sklearn训练模型。

拆分训练集，将特征和标签进行拆分：

```
x = data.drop(labels="count", axis=1)
y = data[["count"]]
```

使用sklearn训练模型：

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(x, y)
```

3.5. 预测验证

在利用训练好的模型对测试数据进行预测时，需要将测试数据进行相同的处理，即将日期和地区转换为其背后的含义。

该过程请参考上述过程自行完成。

附

整合订单信息结构完整示例

```
{
    "order": {
        "order_id": "9ad75820917ceedcc17aaba446a14b80",
        "driver_id": "975c4edacf558695d2f442d39c60a866",
        "member_id": "160a762f027e185b3e06ed5f5fa26223",
        "create_date": "2017-08-07",
        "create_hour": "22",
        "status": "2",
        "estimate_money": "36.00",
        "estimate_distance": "5799.00",
        "estimate_term": "11.00",
        "start_geo_id": "59e9ad69a9b98a75107b2a4ba96cd765",
        "end_geo_id": "27d75f17e61587172fe7a6827bbaa198"
    },
    "startpoi": {
        "rid": "59e9ad69a9b98a75107b2a4ba96cd765",
        "petrol": "6",
        "market": "26",
        "uptown": "236",
        "metro": "6",
```

```
            "bus": "42",
            "cafe": "146",
            "restruant": "899",
            "atm": "49",
            "office": "64",
            "hotel": "141"
        },
        "endpoi": {
            "rid": "27d75f17e61587172fe7a6827bbaa198",
            "petrol": "2",
            "market": "47",
            "uptown": "244",
            "metro": "6",
            "bus": "35",
            "cafe": "157",
            "restruant": "898",
            "atm": "107",
            "office": "177",
            "hotel": "186"
        },
        "weather": [
            {
                "date": "2017-08-07",
                "hour": "22",
                "min": "00",
                "text": "晴",
                "code": "1",
                "temperature": "29",
                "feels_like": "28",
                "pressure": "996",
                "humidity": "57",
                "visibility": "22",
                "wind_direction": "南",
                "wind_direction_degree": "194",
                "wind_speed": "9",
                "wind_scale": "2"
            },
            {
                "date": "2017-08-07",
                "hour": "22",
                "min": "30",
                "text": "多云",
                "code": "4",
                "temperature": "29",
                "feels_like": "28",
```

```
            "pressure": "996",
            "humidity": "60",
            "visibility": "21.8",
            "wind_direction": "西南",
            "wind_direction_degree": "203",
            "wind_speed": "8.28",
            "wind_scale": "2"
        }
    ]
}
```