# 摩拜单车分析和可视化报告

## 一、项目背景

1、本文对项目中给出的训练集数据mobike_shanghai_sample_updated.csv进行数据的探索性分析并可视化

- 不同时间段的订单数量分布图（柱形图）
- 骑行距离分布图（柱形图）
- 骑行时长分布图（柱形图）
- 单车初始位置分布图 （散点图）
- 空闲单车分布图（散点图）

2、分析的目的：获取用户出行的规律，主要分析维度是时间，日期，骑行距离等

3、工具： Pycharm

4、技术栈：numpy、pandas（用于数据处理），KMeans、flask（后端）、echarts（前端）用于数据可视化

## 二、数据预处理

**1、读取数据**

用pandas的read_csv方法读取csv文件mobike_shanghai_sample_updated.csv， 该数据集为摩拜提供的上海城区2016年8月随机抽样百万条用户使用数据，包含起点、目的地、租赁时间、还车时间、用户ID、车辆ID、交易编号等

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn import cluster, metrics
from sklearn.neighbors import NearestNeighbors

data = pd.read_csv('data/mobike_shanghai_sample_updated.csv')
data.head()
```

**2、更改列的数据类型**

更改'orderid', 'bikeid', 'userid'列的数据类型为str类型

```
tobestr = ['orderid', 'bikeid', 'userid']
data[tobestr] = data[tobestr].astype('str')
data['start_time'] = pd.to_datetime(data['start_time'])
data['end_time'] = pd.to_datetime(data['end_time'])
data.info()
```

**3、添加骑行时间列**

新增骑行时长duration等列（通过起始时间计算），并通过字符串拆分将骑行时长单位统一为分钟（命名为ttl_min列）

```
data['duration'] = data.end_time - data.start_time
data['dur_day'] = data.duration.apply(lambda x: str(x).split(' ')[0])
data['dur_hr'] = data.duration.apply(lambda x: str(x).split(' ')[-1][:2])
data['dur_min'] = data.duration.apply(lambda x: str(x).split(':')[-2])
data['dur_sec'] = data.duration.apply(lambda x: str(x).split(':')[-1])
tobeint = ['dur_day', 'dur_hr', 'dur_min', 'dur_sec']
data[tobeint] = data[tobeint].astype('int')
data['ttl_min'] = data.dur_day * 24 * 60 + data.dur_hr * 60 + data.dur_min +
data.dur_sec / 60
```

#### 4、 新增dayid和daytype列

获取每条记录是星期几，并根据工作日和周末进行分类， 新增hourid和hourtype列，获取每条记录是几点开始的，并根据早晚高峰和平峰时段进行分类

```
# datetime.datetime.isoweekday（）返回的1-7代表周一--周日；
data['dayId'] = data.start_time.apply(lambda x: x.isoweekday())
# dayType 工作日
data['dayType'] = data.dayId.apply(lambda x: 'weekends' if x == 6 or x == 7 else
'weekdays')

data['hourId'] = data.start_time.apply(lambda x: x.utctimetuple().tm_hour)
# rush hours：7-8，17-20 上班时间 其余时间为non-rush hours
data['hourType'] = data.hourId.apply(lambda x: 'rush hours' if (7 <= x <= 8) or
(17 <= x <= 20) else 'non-rush hours')
```

#### 5、 新增distance（骑行距离）和distocenter列

获取每条记录骑行起始点的直线距离和距离上海中心点的距离（km）

```
# 按每条记录的起点位置，作为发起订单所处位置的数据依据
from math import radians, cos, sin, asin, sqrt

# 自定义函数，通过两点的经纬度计算两点之间的直线距离
def geodistance(lng1, lat1, lng2, lat2):
    lng1_r, lat1_r, lng2_r, lat2_r = map(radians, [lng1, lat1, lng2, lat2])  # 经
纬度转换成弧度
    dlon = lng1_r - lng2_r
    dlat = lat1_r - lat2_r
    dis = sin(dlat / 2) ** 2 + cos(lat1_r) * cos(lat2_r) * sin(dlon / 2) ** 2
    distance = 2 * asin(sqrt(dis)) * 6371 * 1000  # 地球平均半径为6371km
    distance = round(distance / 1000, 3)
    return distance

data['distance'] = np.nan
data['disToCenter'] = np.nan

# 自定义函数，通过调用geodistance获取每条记录骑行始末点和起点距中心点的直线距离
def get_dis(item):
    # distance：点和起点距中心点的直线距离
    item['distance'] = geodistance(item['start_location_x'],
item['start_location_y'],
                                   item['end_location_x'],
item['end_location_y'])  # 计算骑行始末点经纬度的直线距离
    # 国际饭店一般被认为是上海地理中心坐标点，计算骑行起始点经纬度和国际饭店经纬度的直线距离
    # disToCenter：国际饭店的距离
```

```
    item['disToCenter'] = geodistance(item['start_location_x'],
item['start_location_y'], 121.471632, 31.233705)
    return item

data = data.apply(get_dis, axis=1)


# 骑行距离分组数量统计
gbDis = data1.groupby(by="distance")
gbDisList = []
for g in gbDis:
    gbDisList.append((g[0], len(g[1])))
gbDisListToDF = pd.DataFrame.from_records(gbDisList, columns=["distance",
"distanceNum"]).sort_values(by="distanceNum",

                      ascending=True)
gbDisListToDF.to_csv("data/gbDisListToDF.csv", index=False)
```

### 6、 确定空闲单车位置

```
locList = []  # 经纬度
for l in zip(data["end_location_x"], data["end_location_y"]):
    locList.append((l[0], l[1]))

locDisDF = pd.DataFrame(locList, columns=["longitude", "latitude"])
locDisDF.to_csv("data/locDisDF.csv", index=False)

dataLoc = pd.read_csv("data/locDisDF.csv")
nbrs = NearestNeighbors(n_neighbors=200).fit(dataLoc)  # 对每个点距离最近的200个点分
为一组
distances, indices = nbrs.kneighbors(dataLoc)  # 计算每个点最近的200个点的距离，并返
回索引

dataLocList = np.array(dataLoc).tolist()

# 画出每个点附近距离第200个点的距离
dist = distances[:, 199]
dist_ = np.sort(dist)

# 出现拐点的标志是说偏离集群点到远集群点的距离,大概率比较平整，  因为在一定区域内
plt.plot(dist_)
plt.show()
```

**DBSCAN算法将数据点分为三类：**

- **核心点：在半径Eps内含有超过MinPts数目的点**
- **边界点：在半径Eps内点的数量小于MinPts，但是落在核心点的邻域内**
- **噪音点：既不是核心点也不是边界点的点**

**通过获取到点与最大距离，来进行分组**

```
# eps:两个样本之间的最大距离，即扫描半径
#  min_samples ：作为核心点的话邻域(即以其为圆心，eps为半径的圆，含圆上的点)中的最小样本数(包括点本身)。
mdl_dbscan = cluster.DBSCAN(eps=0.0031, min_samples=200).fit(
    dataLoc)  # eps 看上图的拐点的大概值，可以查看模型的评分进行调整,min_samples=200并不是固定的
# davies_bouldin_score:
# 分数定义为每个群集与其最相似群集的平均相似性度量，其中相似度是群集内距离与群集间距离之比。
# 因此，距离较远且分散程度较低的群集将获得更好的分数。最低分数为零，值越低表示聚类越好。
metrics.davies_bouldin_score(dataLoc, mdl_dbscan.labels_)  # 1.5311 模型评分

# 展示有多少类，每类有多少个样本    70个样本类，即70个单车停放点
count = pd.Series(mdl_dbscan.labels_).value_counts()
```

**kmeans计算获得的大致分组数进行计算中心点**

```
from sklearn.cluster import KMeans

mdl_kmeans = KMeans(n_clusters=70).fit(dataLoc)

count2 = pd.Series(mdl_kmeans.labels_).value_counts()

km_centor = pd.DataFrame(mdl_kmeans.cluster_centers_, columns=['longitude',
'latitude'])  # 获取70个单车停放点的经纬度

km_centorList = np.array(km_centor).tolist()
```

**获取空闲单车坐标**

```
# 计算相邻的数据点的欧式距离
def dist(p1, p2):
    return ((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2) ** (0.5)

border_points = []  # 空闲单车坐标集合
for core in km_centorList:
    for other in dataLocList:
        if 0.0029 <= dist(core, other) <= 0.0031:  # 即不靠近中心点的位置，又在边缘处
            border_points.append(other)

# 去重
temp = []
for item in border_points:
    if not item in temp:
        temp.append(item)

borderPointsDF = pd.DataFrame(temp, columns=["longitude", "latitude"])
borderPointsDF.to_csv("data/borderPointsDF.csv", index=False)
```

### 7、data数据集结构说明

主要数据字典如下：

- orderid：交易编号

- bikeid：车辆ID
- userid：用户ID
- start_time：订单发起时间
- start_location_x：骑行起点经度
- start_location_y：骑行起点纬度
- end_time：订单结束时间
- end_location_x：骑行终点经度
- end_location_y：骑行终点纬度
- track：骑行轨迹
- duration：骑行时长，按订单发起和结束时间间隔计算，单位按天/时/分/秒表示
- ttl_min：骑行时长，通过duration数据进行单位统一，因原始数据中的时间精度仅到分，故该变量单位精确到分
- dayid：星期编号（按订单发起时间统计）
- daytype：工作日/双休日（按订单发起时间统计，周一至周五为工作日，周六及周日为双休日）
- hourid：小时编号（按订单发起时间统计，早7-8点及晚17-20点为高峰时段，其他时段为非高峰时段）
- hourtype：高峰时段/非高峰时段（按订单发起时间统计）
- distance：骑行始末点的直线距离（km）
- disToCenter：骑行起始点经纬度和国际饭店经纬度的直线距离

## 三、获取可视化数据

**1、每个小时的订单量，保为hour_num_df.csv文件**

```
data["hourId"] = data["hourId"].apply(lambda x: x + 1)
hour_group = data.groupby("hourId")
hour_num_df = hour_group.agg({"orderid": "count"}).reset_index()  # 计算分组后的单车数
hour_num_df.to_csv("data/hour_num_df.csv", index=None)
```

**2、骑行距离的单车数量分布，保存为gbDisListToDF.csv文件**

```
gbDis = data.groupby(by="distance")
gbDisList = []
for g in gbDis:
    gbDisList.append((g[0], len(g[1])))
gbDisListToDF = pd.DataFrame.from_records(gbDisList, columns=["distance", "distanceNum"]).sort_values(by="distanceNum",ascending=True)
gbDisListToDF.to_csv("data/gbDisListToDF.csv", index=False)
```

**3、骑行时长的单车数量分布，保存为gbtimeListToDF.csv文件**

```
# 骑行时间单车数量分布
gbtime = data.groupby(by="ttl_min")
gbtimeList = []
for g in gbtime:
    gbtimeList.append((g[0], len(g[1])))
gbtimeListToDF = pd.DataFrame.from_records(gbtimeList, columns=["time", "timeNum"]).sort_values(by="time",ascending=True)
gbtimeListToDF.to_csv("data/gbtimeListToDF.csv", index=False)
```

**4、确定单车的初始位置，保存为locDisDF.csv文件**

```
# 确定单车位置
locList = []  # 经纬度
for l in zip(data["end_location_x"], data["end_location_y"]):
    locList.append((l[0], l[1]))
locDisDF = pd.DataFrame(locList, columns=["longitude", "latitude"])
locDisDF.to_csv("data/locDisDF.csv", index=False)
```

**5、单车空闲坐标，保存文件borderPointsDF.csv文件**

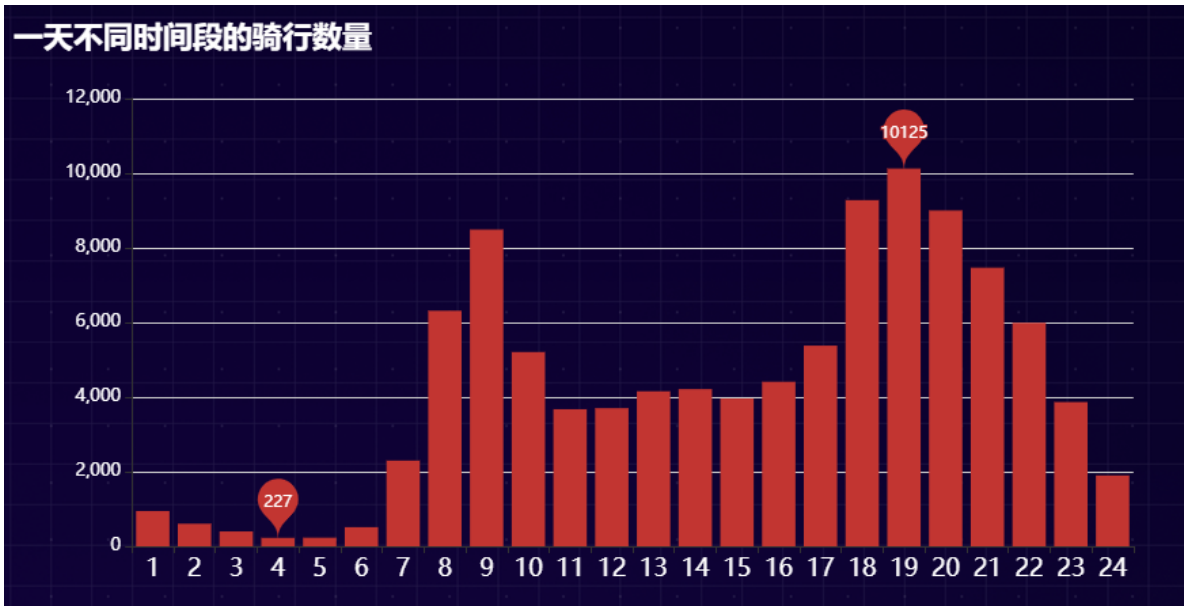
```
borderPointsDF = pd.DataFrame(temp, columns=["longitude", "latitude"])
borderPointsDF.to_csv("data/borderPointsDF.csv", index=False)
```

# 四、数据可视化

**1、不同时间段的订单数量分布图**

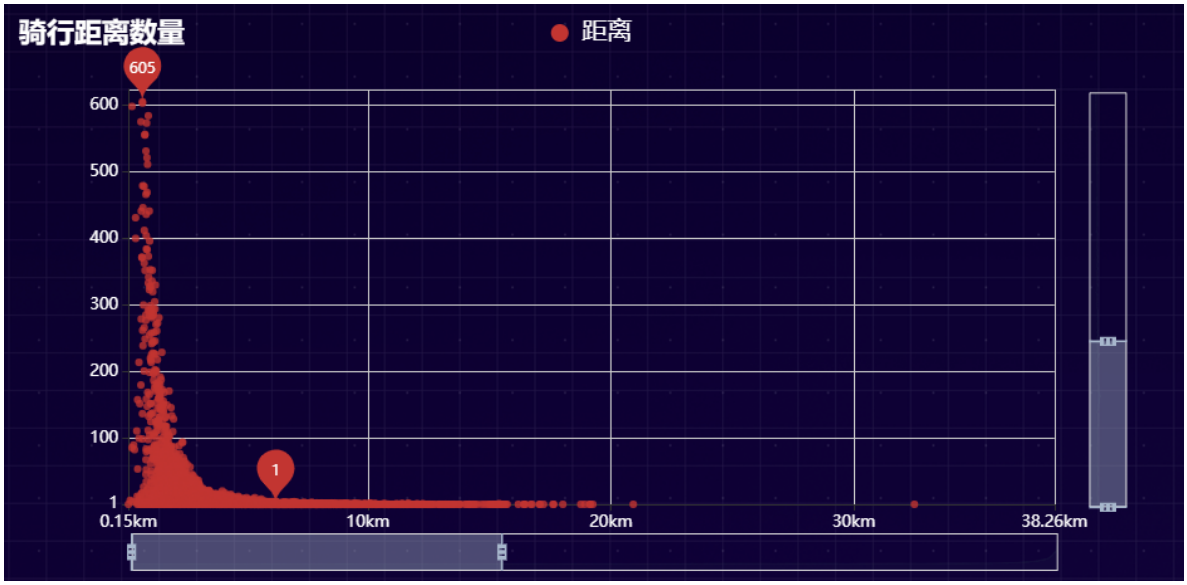
分析：

（1）、23-5点这段时间，人们大多在休息，使用单车出行的订单数极少。
（2）、6点开始，活跃用户增多。在7-9点、17-19点上下班时段，单车订单量迅速增长，呈现明显的骑行早晚高峰。
（3）、11-13点时段，出现局部午高峰，这和中午外出就餐或者休息时间活动有一定关系。




**2、骑行距离分布图**

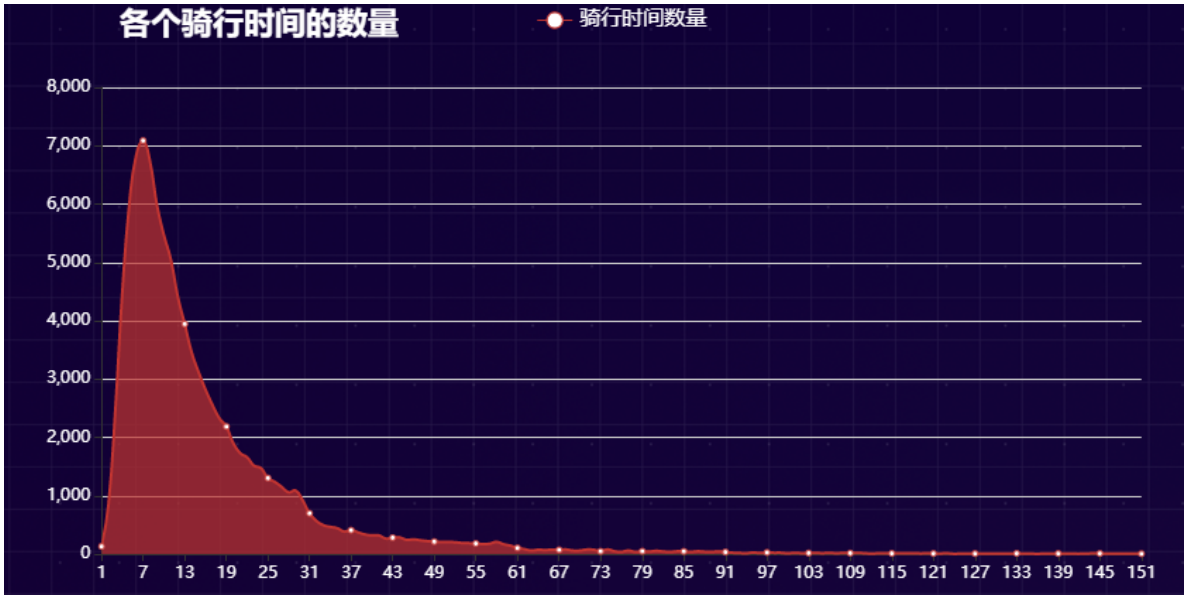
- 分析： 骑行距离数据同样呈现长尾分布，绝大多数骑行距离较短，极少数骑行距离较长。由图可以发现峰值出现在0.7-1.3公里之间



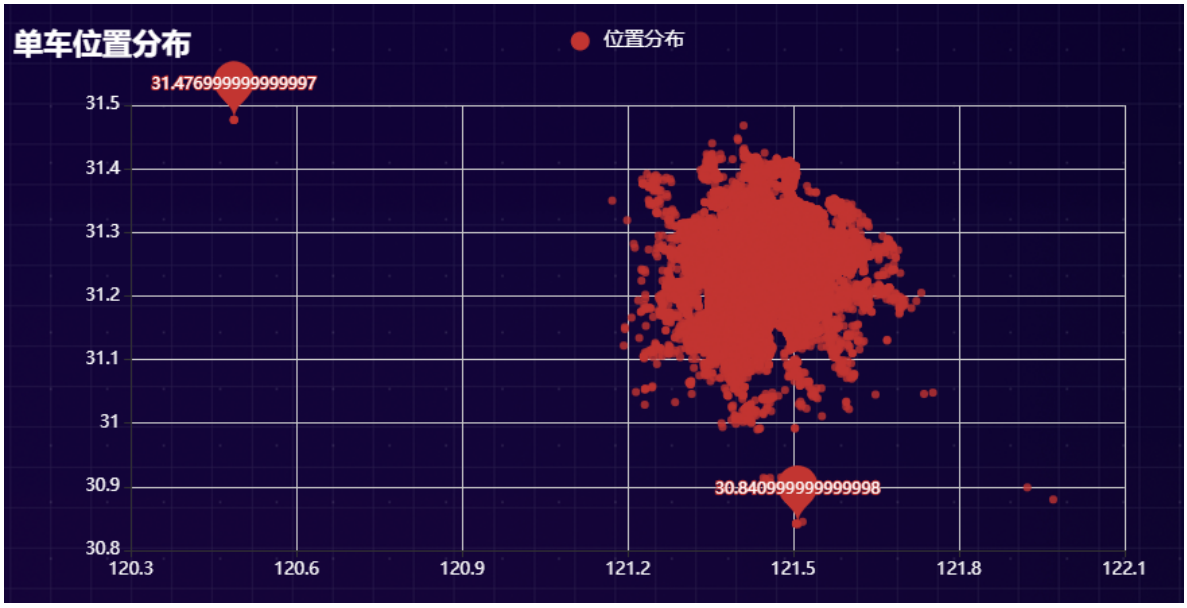

**3、骑行时长分布图**

分析：

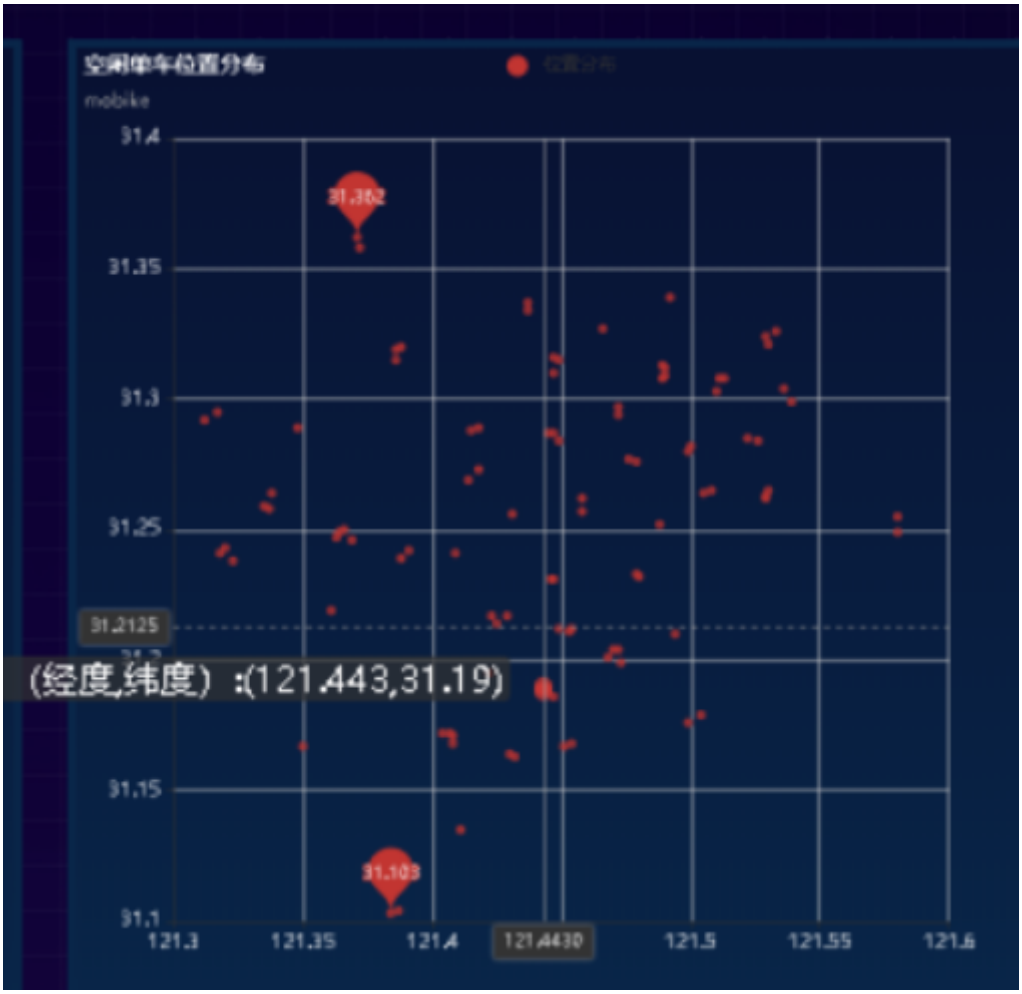
- 骑行时长数据呈现长尾分布，绝大多数骑行时长较短，极少数骑行时间较长。由图可以看出峰值出现在7-10分钟之间




**4、单车骑行结束位置分布图**




**5、单车空闲位置分布图**




flask代码

```
import json
from flask import Flask, redirect, render_template, url_for, request, flash
from sqlalchemy import create_engine
import pymysql
import pandas as pd
import numpy as np
```

```python
app = Flask(__name__)


@app.route("/mobike")
def opinion():
    return render_template("dataVisiable.html")

# 骑行距离分布数据
@app.route("/distance", methods=["GET", "POST"])
def distance():
    res = pd.read_csv("../data/gbDisListToDF.csv")
    total = {}
    dis = []
    for i in zip(res["distance"], res["distanceNum"]):
        dis.append(list(i))
    total = {"dis": dis}
    return total

# 每小时的订单量数据
@app.route("/hour", methods=["POST", "GET"])
def hour():
    res = pd.read_csv("../data/hour_num_df.csv")
    res = np.array(res).tolist()
    hour_res = []
    hour_num = []
    hour_res_dict = {}

    for i in res:
        hour_res.append(i[0])
        hour_num.append(i[1])

    hour_res_dict["hour_res"] = hour_res
    hour_res_dict["hour_num"] = hour_num

    return hour_res_dict

# 骑行时间单车数量分布数据
@app.route("/time", methods=["GET", "POST"])
def time():
    res = pd.read_csv("../data/gbtimeListToDF.csv")
    res = np.array(res).tolist()
    time = []
    time_num = []
    time_dict = {}

    for i in res:
        time_num.append(i[0])
        time.append(i[1])

    time_dict["time"] = time
    time_dict["time_num"] = time_num

    return time_dict

# 单车骑行结束位置分布
@app.route("/local", methods=["GET", "POST"])
def local():
    locList = []
```

```
    locDic = {}
    res = pd.read_csv("../data/locDisDF.csv")
    for l in zip(res["longitude"], res["latitude"]):
        locList.append(list(l))

    locDic["loc"] = locList
    return locDic
# 空闲单车分布数据
@app.route("/free", methods=["GET", "POST"])
def tool():

    locList = []
    locDic = {}
    res = pd.read_csv("../data/borderPointsDF.csv")
    for l in zip(res["longitude"], res["latitude"]):
        locList.append(list(l))

    locDic["loc"] = locList
    return locDic

if __name__ == "__main__":
    app.run(host="localhost", port=5000, debug=True)
```

前端布局代码

```
<!-- 头部 -->
<header id="header">
    <p class="header-title">电影数据展示</p>
    <div class="header-info header-info-l">
        数据来源:IMDB影视
        <div class="header-info-l-weather-temperature">

            <p class="header-info-l-weather">天气情况</p>
            <p class="header-info-l-temperature">温度范围</p>

        </div>

    </div>
    <div class="header-info header-info-r">

        <div class="header-info-r-time"></div>
        <div class="header-info-r-day-year">
            <p class="header-info-r-day">今天是星期几</p>
            <p class="header-info-r-year">这是具体日期</p>
        </div>

    </div>
</header>

<!-- 数据可视化 -->
<div class="container-fluid">
    <div class="row">
        <div class="col-md-6">
            <div id="distance" style="height: 450px"></div>
        </div>
```

```
        <div class="col-md-6">
            <div id="hour" style="height: 450px;"></div>
        </div>
    </div>
</div>
<div class="container-fluid">
    <div class="row">
        <div class="col-md-4">
            <div id="time" style="height: 450px"></div>
        </div>
        <div class="col-md-4">
            <div id="local" style="height: 450px"></div>
        </div>
        <div class="col-md-4">
            <div id="free" style="height: 450px"></div>
        </div>
    </div>
</div>
```

echarts和获取系统当前时间代码

```
<script>
    // 骑行距离数量
    const distance_chart = echarts.init(document.getElementById('distance'));
    $.post('/distance', function (data) {
        data = data["dis"];
        const distance_option = {
            title: {
                text: '骑行距离数量',
                textStyle: {
                    fontSize: 22,
                    color: "white"
                }
            },
            legend: {
                data: ["距离"],
                textStyle: {
                    color: "white",
                    fontSize: 20
                }
            },
            dataZoom: [
                {
                    type: 'slider',
                    show: true,
                    xAxisIndex: [0],
                    start: 0,
                    end: 40
                },
                {
                    type: 'slider',
                    show: true,
                    yAxisIndex: [0],
                    left: '93%',
```

```
                start: 0,
                end: 40
            },
            {
                type: 'inside',
                xAxisIndex: [0],
                start: 0,
                end: 40
            },
            {
                type: 'inside',
                yAxisIndex: [0],
                start: 0,
                end: 40
            }
        ],
        xAxis: [{
            textStyle: {
                color: "white",
                fontSize: 13
            },
            type: 'value',
            scale: true,
            axisLabel: {
                show: true,
                formatter: '{value}km',
                textStyle: {
                    color: "white",
                    fontSize: 13
                }
            }
        }],
        yAxis: [{
            textStyle: {
                color: "white",
                fontSize: 13
            },
            type: 'value',
            scale: true,
            axisLabel: {
                show: true,
                formatter: '{value}',
                textStyle: {
                    color: "white",
                    fontSize: 13
                }
            }
        }],
        tooltip: {
            trigger: 'axis',
            axisPointer: {
                show: true,
                type: 'cross',
                linesStyle: {
                    type: 'dashed',
                    width: 1
                }
            },
```

```
            formatter: "（骑行距离（km），数量）：（{c}）"
        },
        series: [{
            symbolSize: 6,   // 定义散点的大小
            name: '距离',
            type: 'scatter',
            data: data,
            markPoint: {
                data: [
                    {type: 'max', name: '最大值'},
                    {type: 'min', name: '最小值'}
                ]
            },
        }]
    };
    distance_chart.setOption(distance_option);
});

// 一天不同时间段的骑行数量
const hour_chart = echarts.init(document.getElementById('hour'));
$.post("/hour", function (data) {
    const hour = [];
    const hour_option = {
        title: {
            text: '一天不同时间段的骑行数量',
            textStyle: {
                fontSize: 22,
                color: "white"
            }
        },
        legend: {
            data: ["时间段"],
            textStyle: {
                color: "white",
                fontSize: 20
            }
        },
        xAxis: [
            {
                type: 'category',
                data: data.hour_res,
                color: "white",
                axisLabel: {
                    show: true,
                    textStyle: {
                        formatter: '{value}h',
                        color: "white",
                        fontSize: 18
                    },
                    interval: 0  // 横坐标名全显示
                }
            }
        ],
        yAxis: {
            type: 'value',
            axisLabel: {
                show: true,
                formatter: '{value}',
```

```
                textStyle: {
                    color: "white",
                    fontSize: 13
                }
            }
        },
        calculable: 'true',
        tooltip: {
            trigger: 'item',
            formatter: "骑行时间段(h)：{b},:数量：{c}"
        },

        series: [
            {
                name: '数量',
                type: 'bar',

                data: data.hour_num,
                markPoint: {                    //设置定点的值
                    data: [
                        {type: 'max', name: '最大值'},
                        {type: 'min', name: '最小值'}
                    ]
                }
            }
        ]
    };
    hour_chart.setOption(hour_option);
});

// 各个骑行时间的数量
const time_chart = echarts.init(document.getElementById('time'));
$.post("/time", function (data) {
        const time_option = {
            title: {
                text: '各个骑行时间的数量',
                textStyle: {
                    fontSize: 22,
                    color: "white"
                },
                left: 100
            },
            legend: {
                data: ["骑行时间数量"],
                textStyle: {
                    color: "white",
                    fontSize: 15
                }
            },

            dataZoom: [{
                type: 'slider',
                show: true, //flase直接隐藏图形
                xAxisIndex: [0],
                left: '9%', //滚动条靠左侧的百分比
                bottom: -5,
                start: 0,//滚动条的起始位置
                end: 50 //滚动条的截止位置（按比例分割你的柱状图x轴长度）
```

```
                }],

                xAxis: {
                    type: 'category',
                    boundaryGap: false,                 //不设置间距
                    data: data.time_num,
                    axisLabel: {
                        textStyle: {
                            show: true,
                            color: "white"
                        }
                    }
                },
                yAxis: {
                    type: 'value',
                    axisLabel: {

                        textStyle: {
                            show: true,
                            color: "white"
                        }

                    }
                },
                calculable: 'true',
                tooltip: {
                    trigger: 'axis',
                    formatter: "骑行时间(min)：{b}, 该时间数量：{c}"
                },
                //
                series: [
                    {
                        name: '骑行时间数量',    // series和legend上面的data的name要
对应
                        type: 'line',
                        smooth: 'true',                         //设置平滑曲线
                        itemStyle: {                            //设置阴影
                            normal: {areaStyle: {type: 'default'}}
                        },
                        data: data.time
                    }
                ]
            };
            time_chart.setOption(time_option);
        }
    )

    // 单车位置分布
    const local_chart = echarts.init(document.getElementById('local'));
    $.post("/local", function (data) {
        data = data["loc"];
        const local_option = {
                title: {
                    text: '单车位置分布',
                    textStyle: {
                    fontSize: 22,
                    color: "white"
                }
```

```
        },
        legend: {
            data: ["位置分布"],
            textStyle: {
                color: "white",
                fontSize: 15
            }
        },
        xAxis: [{

            type: 'value',
            scale: true,
            axisLabel: {
                show: true,
                formatter: '{value}',
                textStyle: {
                    color: "white",
                    fontSize: 13
                }
            }
        }],
        yAxis: [{
            textStyle: {
                color: "white",
                fontSize: 13
            },
            type: 'value',
            scale: true,
            axisLabel: {
                show: true,
                formatter: '{value}',
                textStyle: {
                    color: "white",
                    fontSize: 13
                }
            }
        }],
        tooltip:{
            trigger: 'axis',
            axisPointer:
                {
                    show: true,
                    type:'cross',
                    linesStyle:{
                        type: 'dashed',
                        width: 1
                    }
                },
            formatter: "（经度,纬度）:({c})"
        },
        series: [{
            symbolSize: 6,   // 定义散点的大小
            name: '位置分布',
            type: 'scatter',
            data: data,
            markPoint: {
                data: [
                    {type: 'max', name: '最大值'},
```

```
                        {type: 'min', name: '最小值'}
                    ]
                },
            }]
        }
    ;
    local_chart.setOption(local_option);
});


//空闲单车分布图
var free_chart = echarts.init(document.getElementById('free'));
$.post("/free", function (data) {
    data = data["loc"];
    const free_option = {
        title: {
            text: '空闲单车位置分布',
            subtext: 'mobike',
            textStyle: {
                color: "white",
                fontSize: 15
            }
        },
        legend: {
            data: ["位置分布"]
        },
        xAxis: [{

            type: 'value',
            scale: true,
            axisLabel: {
                show: true,
                formatter: '{value}',
                textStyle: {
                    color: "white",
                    fontSize: 13
                }
            }
        }
        ],
        yAxis: [{
            textStyle: {
                color: "white",
                fontSize: 13
            },
            type: 'value',
            scale: true,
            axisLabel: {
                show: true,
                formatter: '{value}',
                textStyle: {
                    color: "white",
                    fontSize: 13
                }
            }
        }],
        tooltip:{
            trigger: 'axis',
```

```
                axisPointer:{
                    show: true,
                    type:'cross',
                    linesStyle:{
                        type: 'dashed',
                        width:1
                    }
                },
                formatter: "（经度,纬度）:({c})"
            },
            series: [{
                symbolSize: 6,   // 定义散点的大小
                name: '位置分布',
                type: 'scatter',
                data: data,
                markPoint: {
                    data: [
                        {type: 'max', name: '最大值'},
                        {type: 'min', name: '最小值'}
                    ]
                },
            }]
        };
        free_chart.setOption(free_option);
    });
    // 处理时间函数stringify()
    function stringify(date, format) {
        if (!date) {
            return;
        }
        const year = date.getFullYear();   //年份
        const month = date.getMonth() + 1;  //月份
        const day = date.getDate();   //获取星期
        const hours = date.getHours();  //获取小时数
        const min = date.getMinutes();  //获取分钟数
        const seconds = date.getSeconds();    //获取秒

        return format.replace(/yyyy/g, year)    ///将年份以格式化形式表示

            .replace(/MM/g, ("0" + month).slice(-2))  // slice(-2)   取后面的两位
  /g 表示进行全局匹配
            .replace(/dd/g, ("0" + day).slice(-2))
            .replace(/hh/g, ("0" + hours).slice(-2))
            .replace(/ss/g, ("0" + min).slice(-2))
            .replace(/kk/g, ("0" + seconds).slice(-2))
            .replace(/yy/g, year)
            .replace(/M(?!a)/g, month)
            .replace(/d/g, day);
    }

    // 获取实时时间
    function setCurrentTime(obj) {
        $(obj).text("系统时间: " + stringify(new Date(), "hh:ss:kk"));
    }
    var current_timer = setInterval("setCurrentTime('.header-info-r-time')",
1000);
    $(".header-info-r-day-year .header-info-r-year").text(stringify(new Date(),
"yyyy-MM-dd"));
```

```
    $(".header-info-r-day-year .header-info-r-day").text("星期" + "日一二三四五
六".charAt(new Date().getDay()));       //获取当前星期数

    $.get("http://wthrcdn.etouch.cn/weather_mini?city=" + encodeURIComponent("番
禺"), function (data) {
        // console.log(data);
        var data1 = JSON.parse(data);
        // console.log(data1);
        if (data1.status === 1000) {
            var weather = data1.data.forecast[0];
            var temperature = weather.low.replace(/[^0-9]/ig, "") + "-" +
weather.high.replace(/[^0-9]/ig, "") + "°C";     //获取当日的温度
            $(".header-info-l-weather").text("天气情况：" + weather.type);
            $(".header-info-l-temperature").text("实时温度：" + temperature);
            // $(".nav_temperature .nav_left_content_green").html(data.data.wendu
+ "<span>°C</span>");
        }
    })
</script>
```

## 五、项目小结

本项目通过Python进行对mobike_shanghai_sample_updated.csv数据进行预处理，最后使用flask+echarts对数据进行可视化分析，在前端页面进行展示。