服务器日志数据分析

log.txt 文件记录了某个项目中某个 api 的调用情况，采样时间为每分钟一次，包括调用次数、响应时间等信息，大约18万条数据。下面进行探索性数据分析：

导入数据

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

plt.rc('font', **{'family':'SimHei'})

# 从log.txt导入数据

data = pd.read_table('log.txt', header=None,
                     names=['id', 'api', 'count', 'res_time_sum', 'res_time_min',
                            'res_time_max', 'res_time_avg', 'interval', 'created_at'])
# 或者分开来
data = pd.read_table('log.txt', header=None)
data.columns = ['id', 'api', 'count', 'res_time_sum', 'res_time_min',
                'res_time_max', 'res_time_avg', 'interval', 'created_at']

# 看一下前面几个
data.head()
# 随机抽取5个查看
data.sample(5)
```

	id	api	count	res_time_sum	res_time_min	res_time_max	res_time_avg	interval	created_at
0	162542	/front-api/bill/create	8	1057.31	88.75	177.72	132.0	60	2017-11-01 00:00:07
1	162644	/front-api/bill/create	5	749.12	103.79	240.38	149.0	60	2017-11-01 00:01:07
2	162742	/front-api/bill/create	5	845.84	136.31	225.73	169.0	60	2017-11-01 00:02:07
3	162808	/front-api/bill/create	9	1305.52	90.12	196.61	145.0	60	2017-11-01 00:03:07
4	162943	/front-api/bill/create	3	568.89	138.45	232.02	189.0	60	2017-11-01 00:04:07

数据清洗

检查异常

```
# 检查是否有重复值
data.duplicated().sum()
```

```
# 检查是否有空值
data.isnull().sum()

# 分析 api 和 interval 这两列的数据是否对分析有用
len(data)  # 得到 179496

len(data[data['interval'] == 60])  # 得到 179496
len(data[data['api'] == '/front-api/bill/create'])  # 得到 179496

# 查看api字段信息，可以发现unique=1，也就是说只有一个值，所以是没有意义的
data['api'].describe()
# 删除api一列
data = data.drop('api', axis=1)

# 还发现 interval 的值全是60
data.interval.unique()
# 把 id 和 interval 字段都删掉
data = data.drop(['id', 'interval'], axis=1)

# 发现数据中每一行的 api 和 interval 字段的值都一样，所以丢弃这两列
data2 = data.drop(columns=['api', 'interval'])
data2.head()


# 查看维度信息
data2.shape
# 查看字段类型
data2.dtypes
data2.info()
data2.describe()
```

	id	count	res_time_sum	res_time_min	res_time_max	res_time_avg	interval
count	1.794960e+05	179496.000000	179496.000000	179496.000000	179496.000000	179496.000000	179496.0
mean	6.866490e+06	7.175909	1393.177370	108.419620	359.880351	187.812208	60.0
std	3.686579e+06	4.325160	1499.485881	79.640559	638.919769	224.464813	0.0
min	1.625420e+05	1.000000	36.550000	3.210000	36.550000	36.000000	60.0
25%	3.825183e+06	4.000000	607.707500	83.410000	198.280000	144.000000	60.0
50%	6.811432e+06	7.000000	1154.905000	97.120000	256.090000	167.000000	60.0
75%	9.981397e+06	10.000000	1834.117500	116.990000	374.410000	202.000000	60.0
max	1.343909e+07	31.000000	142650.550000	18896.640000	142468.270000	71325.000000	60.0

可以发现，这份数据其实已经很规整了！

时间索引

为方便分析，使用 created_at 这一列的数据作为时间索引

```
# 查看时间字段，会发现count=unique=179496，说明没有重复值
```

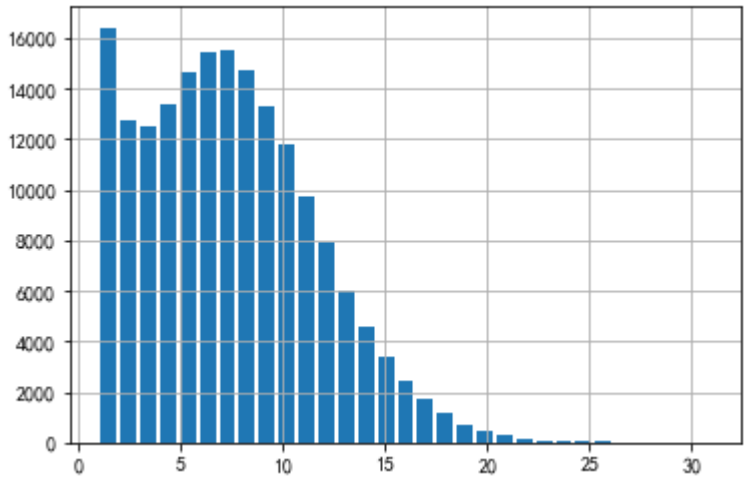
```
data2['created_at'].describe()

# 选取 2018-05-01 的数据，但是没有显示
data2[data2.created_at == '2018-05-01']

# 这样就可以，但是这样选取毕竟挺麻烦的
data2[(data2.created_at >= '2018-05-01') & (data2.created_at < '2018-05-01')]

# 所以，将时间序列作为索引
data2.index = data2['created_at']

# 为了能 data['2018-05-01'] 这样选取数据，我们还要将时间序列由字符串转为时间索引
data2.index = pd.to_datetime(data2['created_at'])

# 有了时间索引，后面的操作就方便多了
data2['2018-05-01']
```

created_at	id	count	res_time_sum	res_time_min	res_time_max	res_time_avg	created_at
2017-11-01 00:00:07	162542	8	1057.31	88.75	177.72	132.0	2017-11-01 00:00:07
2017-11-01 00:01:07	162644	5	749.12	103.79	240.38	149.0	2017-11-01 00:01:07
2017-11-01 00:02:07	162742	5	845.84	136.31	225.73	169.0	2017-11-01 00:02:07
2017-11-01 00:03:07	162808	9	1305.52	90.12	196.61	145.0	2017-11-01 00:03:07
2017-11-01 00:04:07	162943	3	568.89	138.45	232.02	189.0	2017-11-01 00:04:07

数据分析

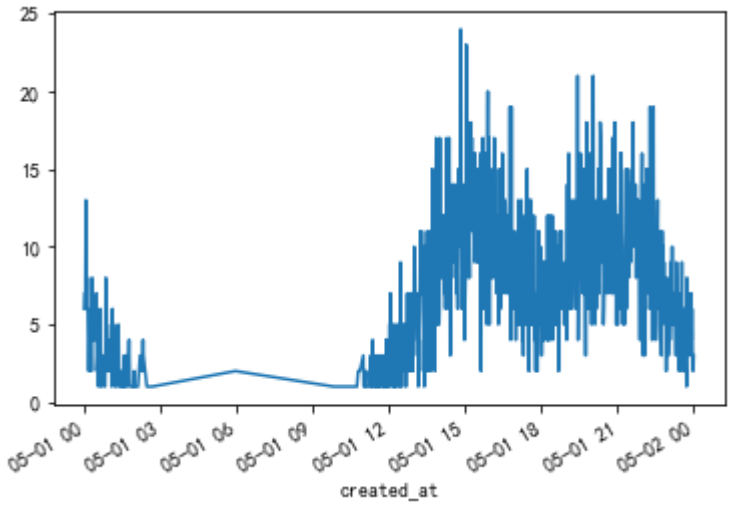
分析 api 调用次数

```
# 分析 api 调用次数情况
# 下面直方图表示单位时间调用api的次数，最大值为31，所以就分31组吧

data['count'].hist(bins=31, rwidth=0.8)
plt.show()
```



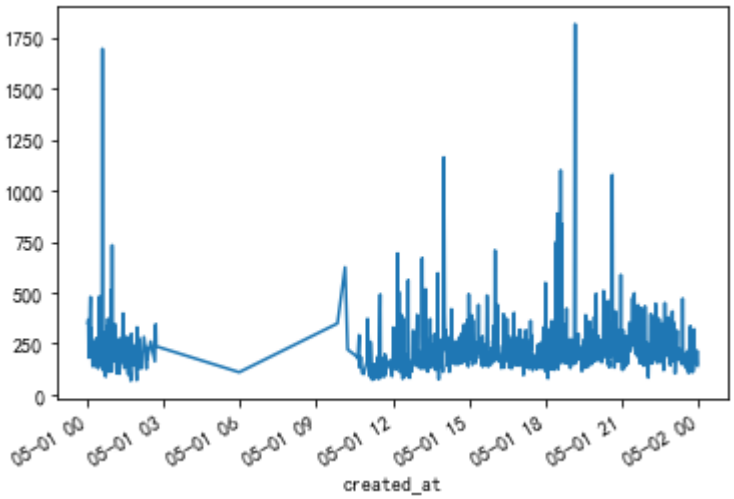
分析访问高峰时段

```
# 分析 api 调用次数情况，例如，在2018-5-1这一天中，哪些时间是访问高峰，哪些时间段访问比较少
# 如下图所示，从凌晨2点到11点访问少，业务高峰出现在下午两三点，晚上八九点。

data2['2018-5-1']['count'].plot()
plt.show()
```

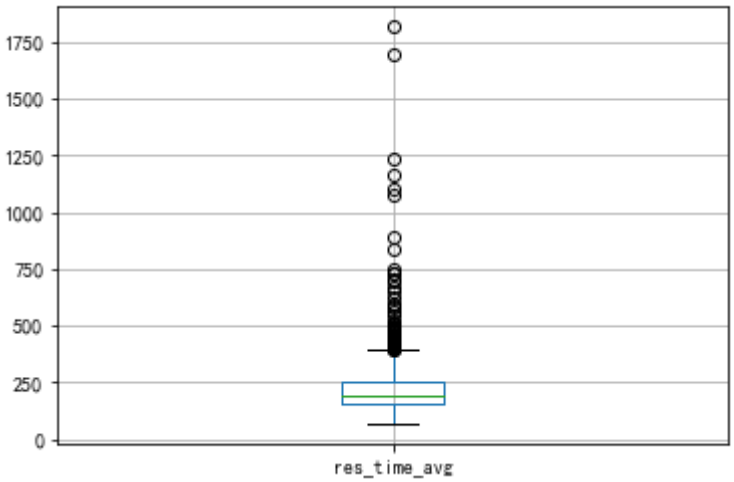


分析 api 响应时间

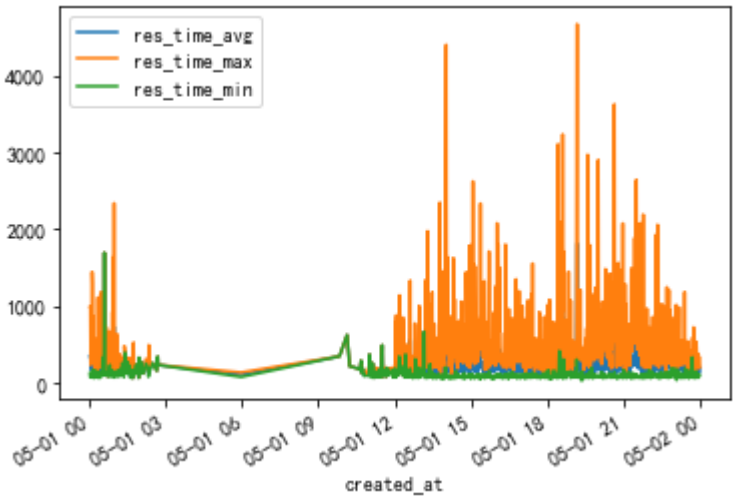
```
data2['2018-5-1'].describe()

# 分析一天中 api 响应时间
data2['2018-5-1']['res_time_avg'].plot()
plt.show()
```



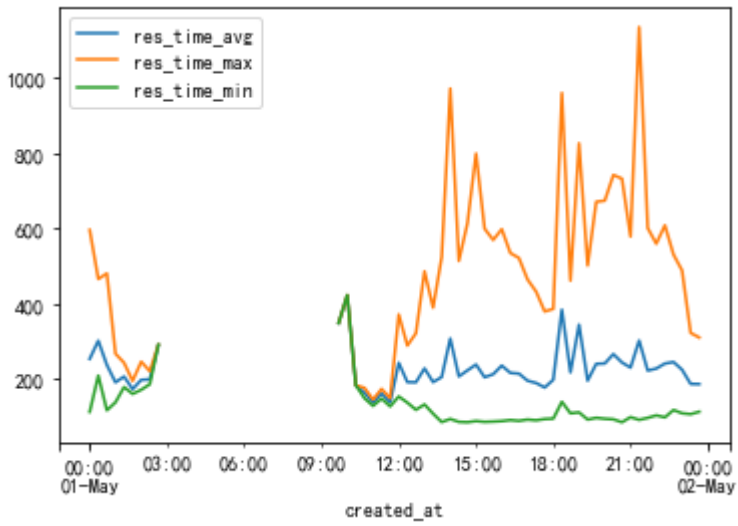

```
data2['2018-5-1'][['res_time_avg']].boxplot()
plt.show()
```




```
data3 = data2['2018-5-1']
#len(data3)
data3[data3['res_time_avg']>1000]


#data2['2018-5-1'][['res_time_avg','res_time_max','res_time_min','res_time_sum']].plot()
data2['2018-5-1'][['res_time_avg','res_time_max','res_time_min']].plot()
plt.show()
```

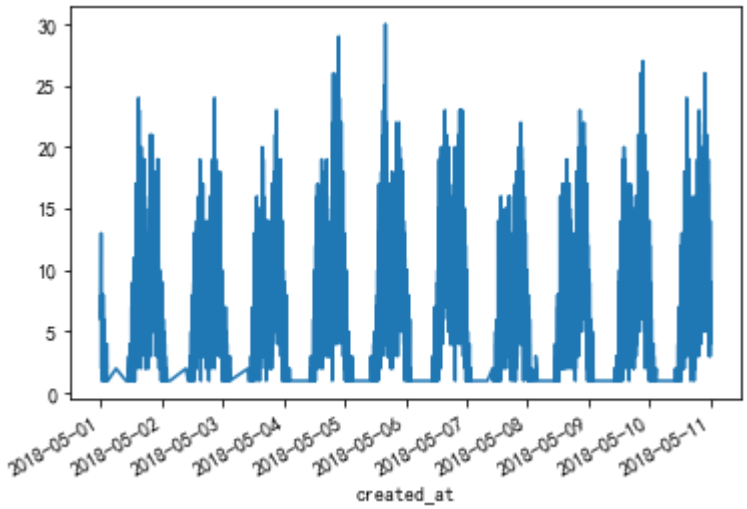



```
# 以20分钟为单位重新采样，可以看到在业务高峰时间段，最大响应时间和平均响应时间都有所上升

#data2['2018-5-1'].resample('20T').mean()
[['res_time_avg','res_time_max','res_time_min','res_time_sum']].plot()
data2['2018-5-1'].resample('20T').mean()
[['res_time_avg','res_time_max','res_time_min']].plot()
plt.show()
```



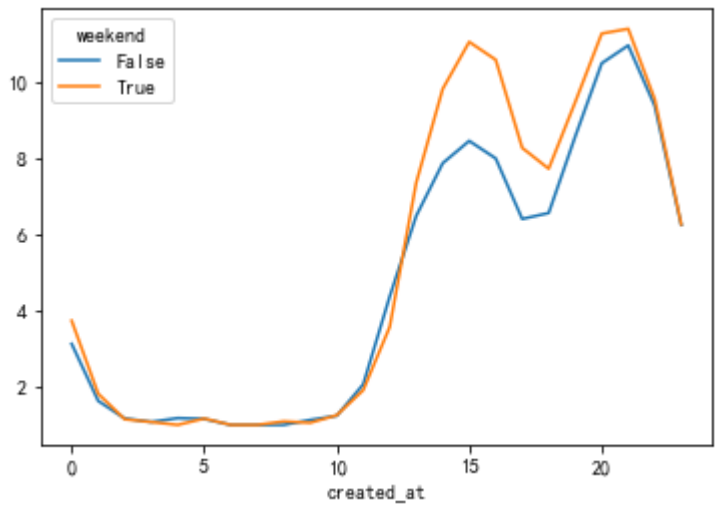
分析连续几天数据

```
# 分析连续的几天数据，可以发现，每天的业务高峰时段都比较相似

data2['2018-5-1':'2018-5-10']['count'].plot()
plt.show()
```



分析周末访问量增加情况

```
# 分析周末访问量是否有增加

data2['weekday'] = data2.index.weekday
data2.head()

# weekday从0开始，5和6表示星期六和星期天
data2['weekend'] = data2['weekday'].isin({5,6})
data2.head()

data2.groupby('weekend')['count'].mean()
data2.head()

#data2.groupby(['weekend', data2.index.hour])['count'].mean().plot()
#plt.show()

data2.groupby(['weekend', data2.index.hour])['count'].mean().unstack(level=0).plot()
plt.show()
```



可以发现，周末的下午和晚上，比非周末的访问量多一些。