智能数据监控日报系统

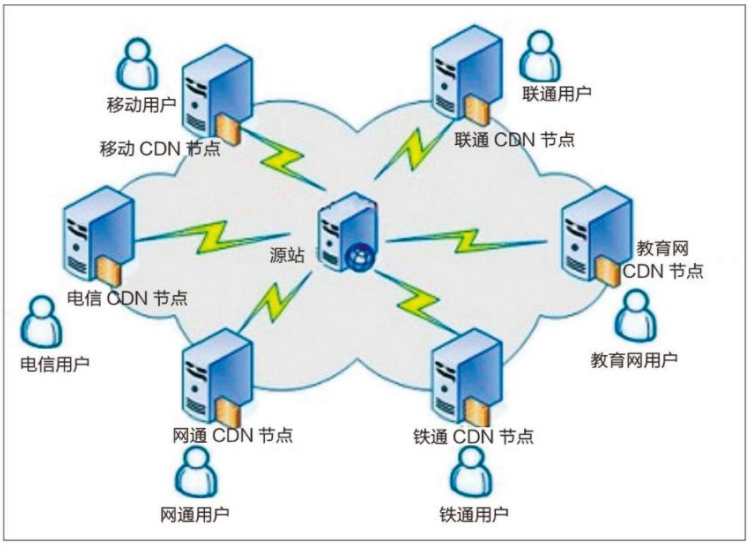
目录

【一】项目背景

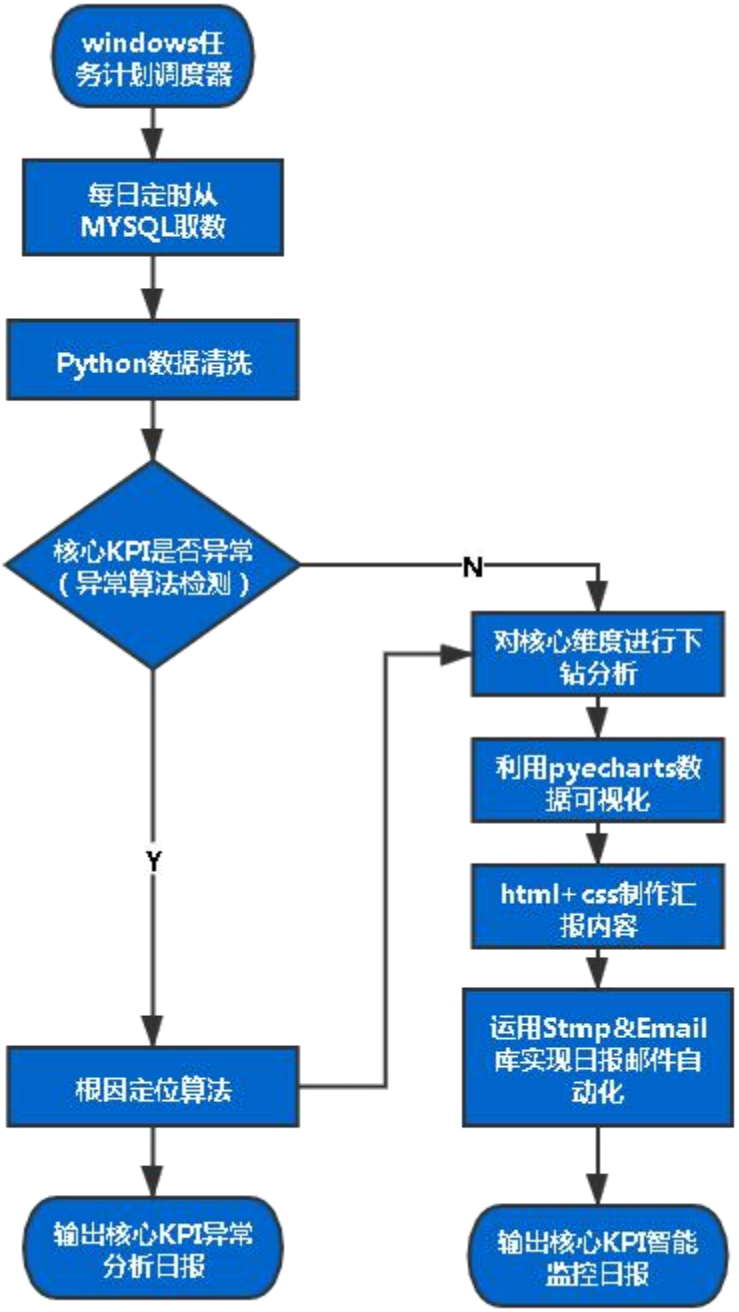
用户观看体验是某视频公司 A 的生命线，而卡顿是影响用户观众体验的头号杀手。我们作为数据分析师，因对卡顿等核心指标进行数据监控，利用异常检测算法自动发现异常点，根因定位算法智能分析其原因，并采取多维度下钻分析及数据可视化的手段形成每日智能监控邮件日报，为公司生命线保驾护航。

【二】数据介绍

- **user_plat**：用户使用什么平台观看视频，例如 adr/ios
- **roomid**：视频 id
- **province**：用户所在省份
- **isp**：用户所用运营商
- **ka_people**：当天总卡顿人数
- **all_people**：当天总观看视频人数
- **cdn_name**: CDN 的全称是 Content Delivery Network, 即内容分发网络。CDN 的基本原理是广泛采用各种缓存服务器，将这些缓存服务器分布到用户访问相对集中的地区或网络中，在用户访问网站时，利用全局负载技术将用户的访问指向距离最近的工作正常的缓存服务器上，由缓存服务器直接响应用户请求。



【三】流程框架


windows任务计划调度器
每日定时从MYSQL取数
Python数据清洗
核心KPI是否异常（异常算法检测）
N
Y
对核心维度进行下钻分析
利用pyecharts数据可视化
html+css制作汇报内容
运用Stmp&Email库实现日报邮件自动化
根因定位算法
输出核心KPI异常分析日报
输出核心KPI智能监控日报

【四】Python 连接 mysql 取数

通过 Pymysql 连接 MYSQL 数据库，设置 day 变量实现动态取数。

```
# mysql取数
def mysql_datagain(day):
    '''
    目的：从mysql中取数明细数据
    day：时间，例'2021-02-23'
    '''
    db = pymysql.connect(host='127.0.0.1',port=3306,user='root',password='Abc123456',db='report',charset=
    cursor =db.cursor()
    start_time = (datetime.datetime.strptime(day,'%Y-%m-%d') - timedelta(days=22)).strftime('%Y-%m-%d')
    sql = '''select * from sp_karate_detail where day between %s and %s'''
    cursor.execute(sql,(start_time,day))
    data = cursor.fetchall()
    col = [i[0] for i in cursor.description]
    df = pd.DataFrame(list(data),columns=col)
    db.close()
    return df
```

【五】Prophet 时序异常检测算法

由于大多类似卡顿率等核心指标均周期性，传统的异常检测算法难以学习周期性，所以我们选择 FaceBook 开源的时序异常检测算法 Prophet。

```
# 异常检测算法判断异常
def prophet_error(df,day):
    '''
    目的：判断当天卡顿率是否异常

    df : 明细数据
    day : 时间，例'2021-02-23'
    '''

    #转换成prophet所需格式
    df = df[['day','ka_rate']]
    df.columns = ['ds','y']

    m_e = Prophet(interval_width = 0.93)
    m_e.fit(df)
    future_e = m_e.make_future_dataframe(periods=0)
    forecast_e = m_e.predict(future_e)

    df['yhat_lower'] = forecast_e['yhat_lower']
    df['yhat_upper'] = forecast_e['yhat_upper']
    day_error = list(df[((df['y'] - df['yhat_lower'])<0) | ((df['y'] - df['yhat_upper'])>0)]['ds'])

    result = 1 if day in day_error else 0
    return result
```

【六】卡顿率多维度下钻分析报表

报表由当天**卡顿率情况文字描述、CDN 质量排名表格、卡顿率多维度下钻图表、当天明细数据附件**组成。

当天卡顿率情况文字描述：

基于数据实现动态判断上升/下降，并且当卡顿率大于业务阈值 10%后自动标红。

```
# 相比昨天环比{上升/下降/持平}xxx%
yesterday_dif = []
if today_karate <= yesterday_karate:
    yes_text = '下降'
    yesterday_dif.append(round((yesterday_karate - today_karate)*100/yesterday_karate,2))

elif today_karate >= yesterday_karate:
    yes_text = '上升'
    yesterday_dif.append(round((today_karate-yesterday_karate)*100/yesterday_karate,2))
else:
    yes_text = '持平'
    yesterday_dif.append('')
```

```
def generate_table(data):
    '''
    目的：将CDN质量排名表格dataframe的数据转换为html
    data：CDN质量排名表格dataframe
    '''
    html = ''
    for index in range(data.shape[0]):
        # 选取行的索引，一行一行建设
        html += '<tr>'
        for col in range(data.shape[1]):
            # 选取列的索引，一列一列的填入
            if col==3:
            # 如果col，列索引为3的时候
                if np.float(data.iloc[index,3].strip('%')) >=10:
                # 如何卡顿率>=10
                    html += '<td class="tg-vd9z">'+ str(data.iloc[index,col]) + '</td>'
                else:
                    html += '<td class="tg-9wq8">'+ str(data.iloc[index,col]) + '</td>'
            else:
                html += '<td class="tg-9wq8">'+ str(data.iloc[index,col]) + '</td>'
        html += '</tr>'
    return html
```

CDN 质量排名表格：

动态取出当天 CDN 质量排名表格，并利用 html+css 封装成智能与美观的表格。

```
# 多维度下钻分析 - 图表
def analyse_tablemake(df,day):
    '''
    目的：构建邮件日报的CDN质量排名表格
    df：明细数据
    day：时间，例'2021-02-23'
    '''
    df_today = df[df['day']==day]
    df_cdn = df_today.groupby('cdn_name',as_index=False)[['ka_people','all_people']].sum()
    df_cdn['ka_rate'] = df_cdn['ka_people'] / df_cdn['all_people']
    df_cdn['rank'] = df_cdn['ka_rate'].rank().astype(int)
    df_cdn['ka_rate'] = round(df_cdn['ka_rate']*100,2).astype(str) +'%'
    df_cdn.columns = ['cdn','卡顿人数','观众总人数','卡顿率','质量排名']
    return df_cdn
```

```
def generate_table(data):
    '''
    目的：将CDN质量排名表格dataframe的数据转换为html
    data：CDN质量排名表格dataframe
    '''
    html = ''
    for index in range(data.shape[0]):
        # 选取行的索引，一行一行建设
        html += '<tr>'
        for col in range(data.shape[1]):
            # 选取列的索引，一列一列的填入
            if col==3:
            # 如果col，列索引为3的时候
                if np.float(data.iloc[index,3].strip('%')) >=10:
                # 如何卡顿率>=10
                    html += '<td class="tg-vd9z">'+ str(data.iloc[index,col]) + '</td>'
                else:
                    html += '<td class="tg-9wq8">'+ str(data.iloc[index,col]) + '</td>'
            else:
                html += '<td class="tg-9wq8">'+ str(data.iloc[index,col]) + '</td>'
        html += '</tr>'
    return html
```

卡顿率多维度下钻图表：

利用 pyecharts 实现卡顿率多维度下钻图表，让数据监控更直观。包括近 20 天卡顿率趋势图、近 20 天观看&卡顿总人数、当天各平台观众分布、当天各运营商观众分布、当天各省份观看人数等。

```
# 近20天卡顿率趋势图
x_data = list(df_kpi['day'])
y_data = list(df_kpi['ka_rate'].round(3))
line = Line(init_opts=opts.InitOpts(theme='light',bg_color=JsCode(bg_color_js),width='700px',height=
line.add_xaxis(x_data)
line.add_yaxis('卡顿率',y_data)
line.set_series_opts(label_opts=opts.LabelOpts(is_show=False),itemstyle_opts=opts.ItemStyleOpts(colo
                     markarea_opts=opts.MarkAreaOpts(data=[opts.MarkAreaItem(name="春节", x=("2021-02-
line.set_global_opts(legend_opts=opts.LegendOpts(is_show=False),title_opts=opts.TitleOpts(title="近20
make_snapshot(snapshot,line.render(),"1.png")
```

```
# 近20天总人数与卡顿人数联合表
y_data1 = list(df_kpi['ka_people'])
y_data2 = list(df_kpi['all_people'])
bar = Bar(init_opts=opts.InitOpts(theme='light', bg_color=JsCode(bg_color_js),width='700px', height='
bar.add_xaxis(x_data)
bar.add_yaxis('观看总人数', y_data1)
bar.add_yaxis('卡顿总人数', y_data2)
bar.set_series_opts(label_opts=opts.LabelOpts(is_show=False))
bar.set_global_opts(title_opts=opts.TitleOpts(title="近20天观看&卡顿总人数"),legend_opts=opts.LegendOpt
make_snapshot(snapshot,bar.render(),"2.png")
```

```
# 当天各平台观众分布
df_today = df[df['day']==day]
df_plat_pie = df_today.groupby('user_plat',as_index=False)['all_people'].sum()
x_plat_data = list(df_plat_pie ['user_plat'])
y_plat_data = list(df_plat_pie ['all_people'])
pie_plat = (Pie(init_opts=opts.InitOpts(theme='infographic',bg_color=JsCode(bg_color_js),width='700px
   .add('观众人数', [list(z) for z in zip(x_plat_data, y_plat_data)],
   label_opts=opts.LabelOpts(
                formatter="\n\n\n{a|{a}}{abg|}\n{hr|}\n {b|{b}: }{c}\n{per|{d}%}    ",
                rich=rich_text))
   )
pie_plat.set_global_opts(title_opts=opts.TitleOpts(title="%s各平台观众分布"%day),legend_opts=opts.Leger
make_snapshot(snapshot,pie_plat.render(),"3.png")
```

```
# 当天各isp观众分布
df_isp_pie = df_today.groupby('isp',as_index=False)['all_people'].sum()
x_isp_data = df_isp_pie['isp']
y_isp_data = df_isp_pie['all_people']
pie_isp = (Pie(init_opts=opts.InitOpts(theme='essos',bg_color=JsCode(bg_color_js),width='700px', heig
   .add('观众人数', [list(z) for z in zip(x_isp_data, y_isp_data)],
   label_opts=opts.LabelOpts(position='outsiede',
                formatter="\n\n\n\n\n\n{a|{a}}{abg|}\n{hr|}\n       {b|{b}: }{c}  {per|{d}%}       "
                rich=rich_text))
   )
pie_isp.set_global_opts(title_opts=opts.TitleOpts(title="%s各运营商观众分布"%day),legend_opts=opts.Lege
make_snapshot(snapshot,pie_isp.render(),"4.png")
```

【七】基于影响度的根因定位算法

其核心借鉴随机森林的特征重要性计算思想。
在随机森林中某个特征 X 的重要性的计算方法如下:
1: 对于随机森林中的每一颗决策树,使用相应的 OOB(袋外数据)数据来计算它的袋外数据误差,记为 errOOB1。
2: 随机地对袋外数据 OOB 所有样本的特征 X 加入噪声干扰(相当于删除该列。),再次计算它的袋外数据误差,记为 errOOB2。
3: 假设随机森林中有 Ntree 棵树,那么对于特征 X 的重要性 = ∑(errOOB2-errOOB1)/Ntree,袋外的准确率大幅度变化,则说明这个特征对于样本的分类结果影响很大,也就是说它的重要程度比较高。

yxd = 去掉之前的卡顿率 - 去掉某维度的卡顿率，yxd 越大说明该维度越容易让指标突增，yxd 越小说明该维度越容易让指标突降，yxd 接近于 0，说明该维度完全不影响指标。

```
# 根因定位算法
def yxd_rootcause_function(data,columns,day):

    '''
    目的：找出异常的根因
    data：明细数据
    columns：根因定位维度
    day：时间，例'2021-02-23'
    '''
```

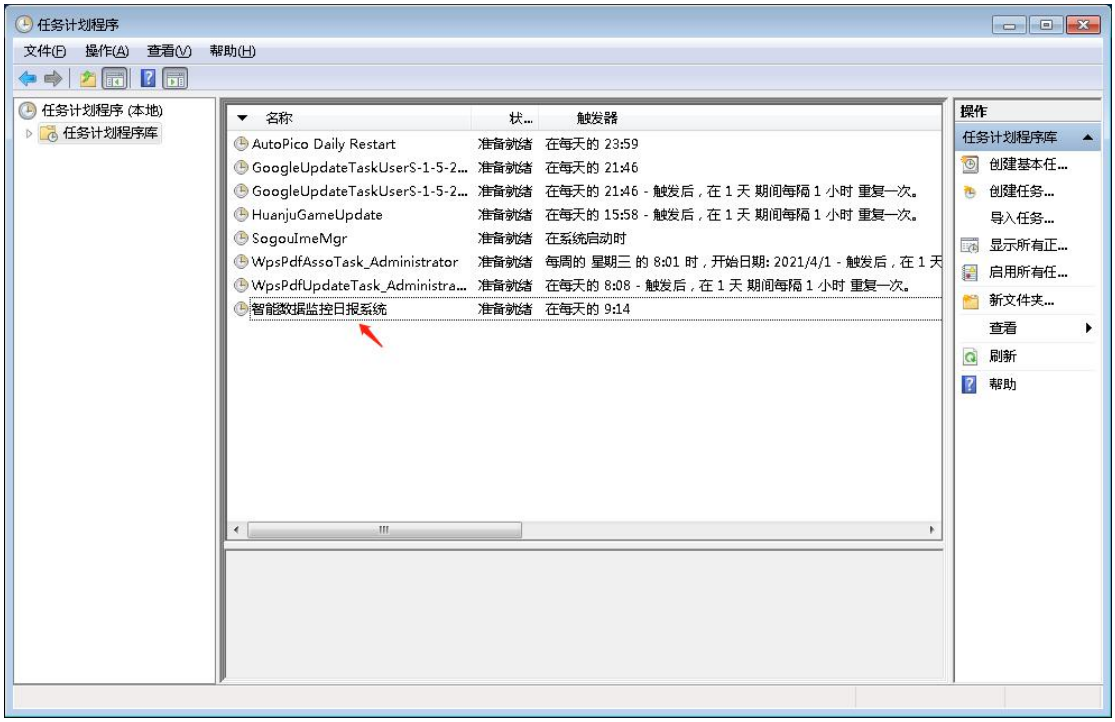
当 Prophet 时序异常检测算法识别出存在异常时，程序会运行根因定位算法智能分析出可能的 TOP3 根因，并绘制影响度表格给予各根因对异常的影响度，以及根因验证图表。

```
# 根因定位报表 - 图表

df_top1 = rootdata_deal(df,df_reason['维度'].iloc[0],df_reason['条件'].iloc[0])
df_top2 = rootdata_deal(df,df_reason['维度'].iloc[1],df_reason['条件'].iloc[1])
df_top3 = rootdata_deal(df,df_reason['维度'].iloc[2],df_reason['条件'].iloc[2])
x_data = list(df_kpi['day'])
y_data = list(df_kpi['ka_rate'])
y_data1 = df_top1['%s_ka_rate'%df_reason['维度'].iloc[0]]
y_data2 = df_top2['%s_ka_rate'%df_reason['维度'].iloc[1]]
y_data3 = df_top3['%s_ka_rate'%df_reason['维度'].iloc[2]]

line2 = Line(init_opts=opts.InitOpts(theme='light',bg_color=JsCode(bg_color_js),width='700px',hei
line2.add_xaxis(x_data)
line2.add_yaxis('卡顿率',y_data)
line2.add_yaxis('去除"%s=%s"数据后的卡顿率'%(df_reason['维度'].iloc[0],df_reason['条件'].iloc[0]),y_
line2.add_yaxis('去除"%s=%s"数据后的卡顿率'%(df_reason['维度'].iloc[1],df_reason['条件'].iloc[1]),y_
line2.add_yaxis('去除"%s=%s"数据后的卡顿率'%(df_reason['维度'].iloc[2],df_reason['条件'].iloc[2]),y_
line2.set_series_opts(label_opts=opts.LabelOpts(is_show=False))
line2.set_global_opts(legend_opts=opts.LegendOpts(pos_bottom='80%',pos_left='10%'),title_opts=opt
make_snapshot(snapshot,line2.render(),"6.png")
```

【八】使用 STMP+EMAIL 库实现邮件发送

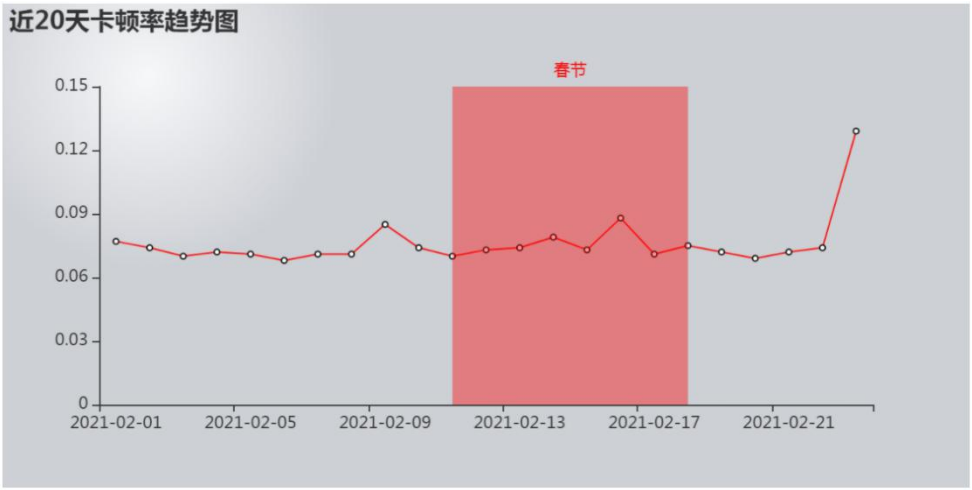
```
# 邮件发送
def mail_send(msg):
    '''
    目的：发送邮件
    msg：msg_make函数的输出
    '''

    mail_host = "smtp.qq.com"
    mail_pass = "xxx"

    s = smtplib.SMTP()
    s.connect(mail_host)
    s.login(mail_sender,mail_pass)
    s.sendmail(mail_sender,to_list,msg.as_string())
    print('发送成功')
```

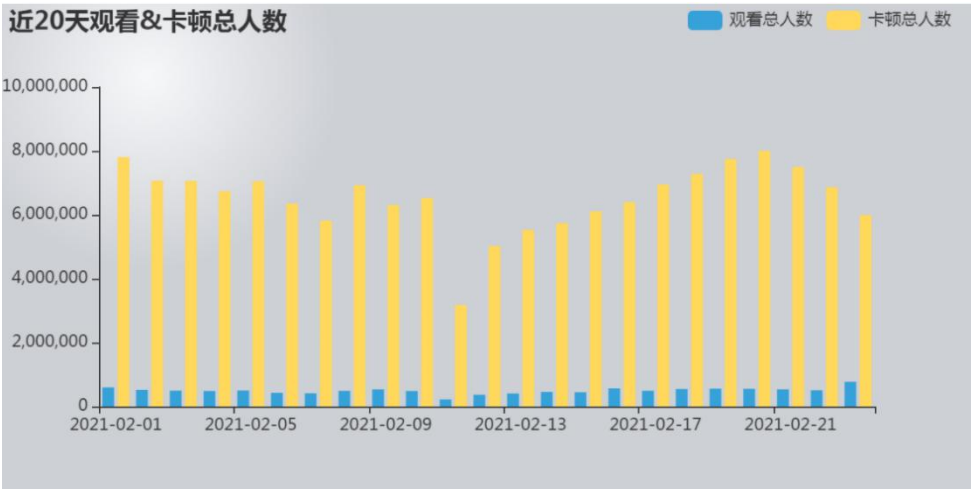
【九】使用 windows 任务计划程序实现全自动

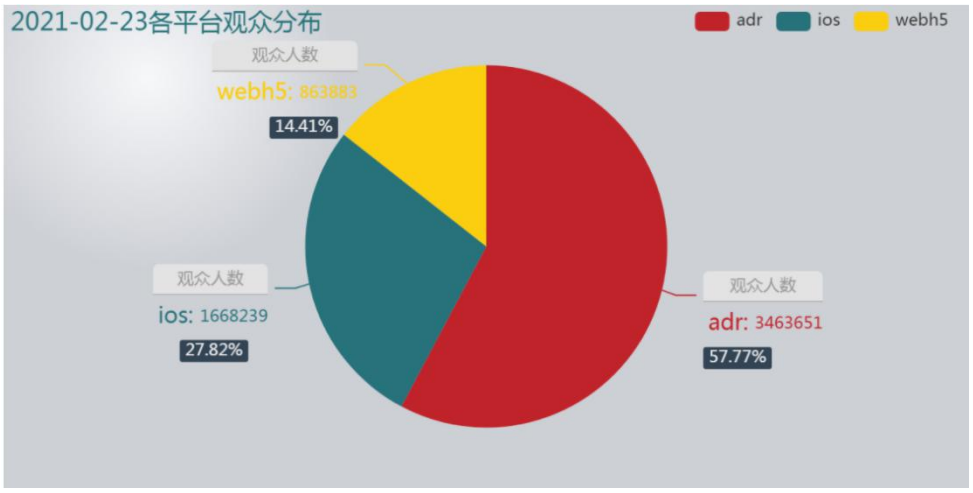


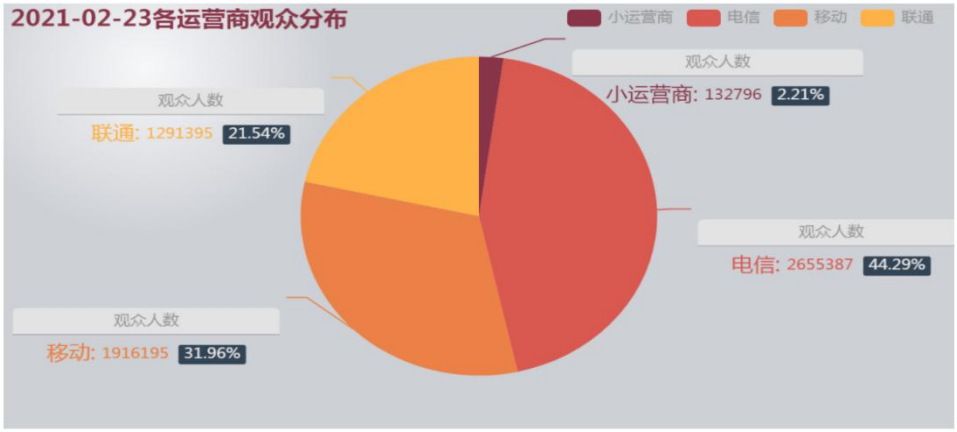
【十】实现效果

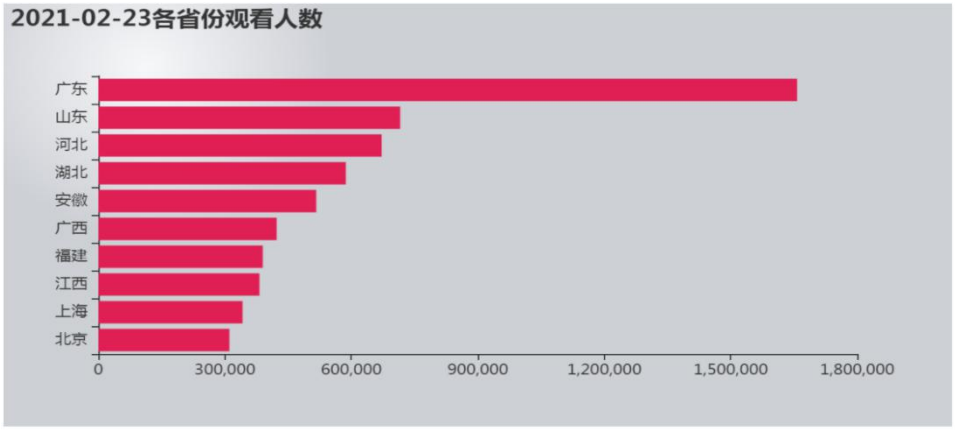
每日固定时间点，自动发出智能数据监控邮件日报，实时掌握用户体验最新情况，并当核心指标出现异常时，算法智能分析出其原因，为业务不断迭代优化提供智能化的支撑。


近20天卡顿率趋势图
春节
0.15
0.12
0.09
0.06
0.03
0
2021-02-01
2021-02-05
2021-02-09
2021-02-13
2021-02-17
2021-02-21


近20天观看&卡顿总人数
观看总人数
卡顿总人数
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
2021-02-01
2021-02-05
2021-02-09
2021-02-13
2021-02-17
2021-02-21


2021-02-23各平台观众分布
adr
ios
webh5
观众人数
webh5: 863883
14.41%
观众人数
ios: 1668239
27.82%
观众人数
adr: 3463651
57.77%





智能根因定位日报

我们基于Prophet时间序列异常检测算法发现当天(2021-02-23)的卡顿率(12.9%)为异常值，调用基于影响度的根因定位算法分析出以下几种根因可能是引起异常的关键点，麻烦相关业务同学进一步核实。

维度	条件	卡顿人数	总人数	该条件下卡顿率	全网卡顿率	该条件影响度	去掉该条件数据后卡顿率	根因排序
roomid	8711409	351497	574379	61.2%	12.9%	5.12%	7.79%	Top1
cdn_name	七牛云	503387	2630067	19.14%	12.9%	4.87%	8.03%	Top2
user_plat	adr	510854	3463651	14.75%	12.9%	2.52%	10.38%	Top3