游乐园客流疏导及酒店预订预测模型

小组成员

1452118 汽车学院 罗怡晨 18217775112

1452132 汽车学院 任晓舟 15618635582

1452822 软件学院 洪嘉勇 15900582673

目录

摘要

游乐园在人数过饱和的情况下，由于游乐项目的位置分布和不同项目的进行时间不同将导致部分游乐项目处发生拥挤，游客的平均等待时间过长，针对该问题，以游园规划图为主要研究对象，利用相关理论，最终得出了一个合理的游园疏导方案；游乐园附近的酒店预定量随季节、假期、周末等因素影响，具有一定的可预测性，利用 2015 年全年房间订单信息并详尽考虑季节、假期、是否为周末等相关因素对预订房间可能产生的影响，大胆引入虚拟变量，最终对 2016 年 1-3 月份的房间预订数得到较为可信的预测，具有较高的现实意义。

针对游园疏导问题，首先分析题目所给信息，作出合理的假设，对游客进入游乐园采用泊松流的模型，对游客游玩各个游乐项目采用排队论模型(M/D/c/∞)。并根据游乐园规划图利用 Floyd 算法计算出任意两个游乐项目点之间的最短径。综合利用这些模型和算法，采用一定的调度策略，通过计算机仿真模拟出游客在游乐园内的走动、等待与游玩状况，并可以较为精准地计算出在没有任何疏导的情况下各个游乐项目点的平均等待队长，人均等待时间。再以缩短人均等待时间为主要优化目的，缩短平均等待队长为参考，结合实际经过合理的计算和调整优化，最终得到每一个游乐项目点等待队长的警戒值，给出了一套完整的游客提醒和疏导方案，并提出了可改进的方向。

针对酒店订单预测问题，首先可以确定，各影响因素直接影响的是每天住店人数，而非预定人数。于是第一步根据题目所给数据处理出2015年每天入住房间数，完成数据准备；第二步创新性地采用聚类分析的思想，分析出在不同情况下，预定提前天数的概率分布，并以此作为最终预测的依据之一；第三步通过建立多元线性回归的模型，选定基变量，提取虚拟变量（经过细致地观察15年数据的波动，还合理地引入描述外部环境剧烈变化的虚拟变量）利用回归分析软件stata得到科学合理的回归拟合函数式。综合利用上述分析结果与数据，先分析计算出2016年相关日期的住店人数，再结合提前预定天数的概率分布，通过计算机编程处理出所需预测的结果。之后又进行了模型误差可信度分析与模型检验，证明其合理性与可靠性，并提出了可改进的方向。

最后对于游乐园疏导问题，为每一个游乐项目点设置了一个由相关理论计算得到的警戒值，将过量的游客疏导往计算机模拟出来的相邻较闲项目点对游客有效地进行了分流。经计算机模拟，优化前A点平均等待时间为4700s，经优化之后降低到1270s，其他利用率不高的点利用率相应上升，游客游园体验大幅上升。

最后对于酒店订单预测问题，计算得到了多元线性回归函数，将它与15年的数据相拟合，有很高的可信度，最后利用该函数拟合出2016年1月1日至5月10日的到店人数，并通过统计得到的每一天提前预订房间天数的概率预测出2016年1月1日止3月31日的房间订单数。比如15年清明节前夕到店人数激增，三月份的订单受到清明影响激增。16年2月份和3月份，我们预测的结果3月份数据相对2月份逐步上升，这与15年三月份数据受到清明节假期影响上升的情况极其相近。

关键词：图论 Floyd 算法 仿真 排队论 多元线性回归 聚类分析 统计预测

一、问题的提出

1.1 背景介绍

Youth 游乐园即将盛大开园，作为本市建有最多过山车的游乐园，受到了青少年的热捧。预计届时园区将迎来每天 1 万的大客流。如何根据客流情况，及时分流人群，为顾客提供游园线路引导，保障游客的游园体验显得尤为重要。

1.2 问题重述

试就园区的整体规划，建立数学模型分析研究下面的问题：

问题一：Youth 乐园共设 A-J 共 10 个项目点，游客可沿着给定的线路往返下个游乐项目。在保障每位游客体验游乐设施的前提下，建立对每个游乐项目的等候游客进行游览提醒和疏导的模型，以达到游园体验最优。

问题二：皇冠假日酒店是游乐园内的酒店，目前已开业，为有需要的游客提供住宿便利。请根据该酒店历史预订数据信息,综合考虑影响房间预定量的主要因素(比如季节,工作日/周末,法定假日,暑期等)建立数学模型。并根据酒 2015 年全年预定数据,预测 2016 年 1 月至 3 月每天预定房间数。

二、问题分析

2.1 问题一的分析：

题目已经给出全天客流量，以及各个游乐项目之间的连通线路和距离。本文先考虑对游乐项目的等待游客不进行任何提醒和疏导，对进园客流量和游客对各个游乐项目的选择采用合理的假设，运用计算机仿真模拟出全天内各个游乐项目点的平均等待队长，以及平均每位游客游玩遍全部游乐项目所需要的等待时间，建立原始模型 1.1;由于 Youth 游乐园的主要面向对象是青少年,他们体力旺盛，但内心浮躁，所以本文主要以缩短人均等待时间为主要优化目的，缩短平均等待队长为参考，建立优化后的模型 1.2 并确定较优解。

2.2 问题二的分析：

题目已给出 2015 年全年所有订单的详情（包括预定时间，预定房间数，到店时间，离店时间），考虑到每天的季节,工作日/周末,法定假日,暑期等直接影响的是每天入住房间数。我们可以从题目所给数据中整理出 2015 年每天的入住房间数，并分析其与当天所处的季节，周末/工作日，是否是寒暑假，是否是法定节假日等的关系，由此预测 2016 年部分日期的入住房间数。之后再分析在不同情况下，游客们提前多少天预定与上述若干因素的关系。综合上述两项分析可以预测 2016 年 1 至 3 月的每天预定房间数。并有一定的可信度。

三、模型的基本假设

3.1 游乐园疏导模型的基本假设：

1.游乐园开放时间为9:00—22:00（20:00开始不再允许游客进入）；

2.在一天内到达游乐园的游客数符合泊松分布；

3.园区各个景点对于游客的吸引力相同；

4.园区各个景点外等待场所足够大，并忽略道路上的拥堵；

5.游客步行平均速度1.5m/s ；

6.每个游乐项目的进场与出场时间忽略不计。

3.2 酒店预测模型的基本假设：

1.酒店房间充足；

2.所有游客在预定房间时都已经掌握了届时的季节，周末/工作日，是否是寒暑假，是否是法定节假日等信息；

3.所有下订单的游客都会按时到达酒店并入住；

4.由于2015年4月2日开始，入住房间数出现了骤增并维持在较高水平，我们假设此时外部环境发生了变化——存在竞争对手突然倒闭的情况。

5.影响因素有且仅有季节，周末/工作日，是否是寒暑假，是否是法定节假日，竞争对手倒闭与否；

6.寒假日期为1月31-2月28日，暑假日期为7月12日-8月31日；

7.春季为3月21日-6月21日，夏季为6月22日-9月23日，秋季为9月24日-12月22日，冬季为12月23日-次年3月20日；

四、符号约定

4.1 游乐园疏导模型的符号约定：

W_{ri}（$\mathrm{i}=1,2,3\ldots\ldots10$）	任意一个时刻各个游乐项目点的等待队长
W_{ti}（$\mathrm{i}=1,2,3\ldots\ldots10$）	任意一位游客在各个游乐项目点的等待时间
E_{ri}（$\mathrm{i}=1,2,3\ldots\ldots10$）	各个游乐项目点等待队长的期望值
E_{ti}（$\mathrm{i}=1,2,3\ldots\ldots10$）	每个游客游玩各个游乐项目所需等待时间的期望值
T_{i}（$\mathrm{i}=1,2,3\ldots\ldots10$）	各个游乐项目进行一场所需要的时间（题目已经给出）
N_{i}（$\mathrm{i}=1,2,3\ldots\ldots10$）	各个游乐项目每场所能容纳的游客数（题目已经给出）
N_{i}（$\mathrm{i}=1,2,3\ldots\ldots10$）	各个游乐项目每场所能容纳的游客数（题目已经给出）
d_{ij}（$\mathrm{i,j}=1,2,3\ldots\ldots10$）	两个游乐项目之间的最短路径长度

4.2 酒店预测模型的符号

N	任意一天入住房间数
M	任意一天预定房间数

五、模型的建立与求解

5.1 游乐园疏导模型的建立与求解：

5.1.1 游乐园的游客进入情况

根据问题一基本假设 2，进入游乐园的游客以泊松流到达，既时间 t 内进入游乐园的游客总数为 n 的概率满足泊松流公式

$$P_n(\mathrm{t}) = \frac{(\lambda \mathrm{t})^{n}}{\mathrm{n!}} \times e^{-\lambda t}$$

（公式 1）

其中参数λ为单位时间内平均到达的游客人数。根据假设 1，有游客进入的时间为 9:00—20:00 是 11 个小时，总客流量 1 万人，计算得λ=0.2525 人/s。即进入游乐园的游客数满足

$$P_n(\mathrm{t}) = \frac{(0.2525\mathrm{t})^{n}}{\mathrm{n!}} \times e^{-0.2525t}$$

（公式 2）

进而我们运用逆变换法计算任意两位游客到达游乐园的时间间隔：已知，进入游乐园的游客数满足公式 2 的泊松分布。则两个顾客的到达时间间隔服从参数为λ=0.2525 的负指数分布，其概率密度函数为：

$$\mathrm{f(t)} = \begin{cases} 0.2525e^{-0.2525t}, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

（公式 3）

其中，$\frac{1}{0.2525} = 3.96(\mathrm{s})$为平均到达时间间隔。

易知其分布函数为$\mathrm{F(t)} = 1 - e^{-0.2525t}$，$\mathrm{t} \geq 0$. 令 R= $\mathrm{F(t)} = 1 - e^{-0.2525t}$，则 R 服从(0,1)上均匀分布。做逆变换得

$$\mathrm{T} = -\frac{1}{0.2525}\ln(1-\mathrm{R}) = -3.96\ln(1-\mathrm{R}).$$

令 U=1-R，则 U 也是(0,1)上的均匀随机数。于是，T=$-3.96\ln$ U是服从于指数分布的随机数，即任意两位游客到达游乐园的时间间隔为 T=$-3.96\ln$ U（U 是服从(0,1)上均匀分布的随机数）。

5.1.2 游客在游园内任意两个游乐项目之间的路线选择

题目已给出各个游乐项目之间的连接关系和距离，根据以上信息，考虑运用Floyd 算法去计算任意两个景点间的最近路线以模拟游乐园中游客的走动，并可用程序将其实现。

1、Floyd 算法基本思想：

令 D_m 表示一个N×N矩阵，它的(i,j)元素是 d_{ij}^m 。已知图中每条线段的长度，则可以确定矩阵 D_0，最终希望得到最短路长度的矩阵 D_N。Floyd算法从 D_0 开始，由 D_0 计算 D_1，然后Floyd算法再由 D_1 计算 D_2。将这个过程重复进行下去，直至由 D_{N-1} 求得 D_N 为止。

计算思路如下，设已知：

（1）游乐项目点i到游乐项目点m的最短路，其中只容许前m-1个游乐项目点即1，2，…，m-1作为中间游乐项目点。

（2）从游乐项目点m到游乐项目点j的最短路，其中只容许前m-1个游乐项目点即1，2，…，m-1作为中间游乐项目点。

（3）从游乐项目点i到游乐项目点j的最短路，其中只容许前m-1个游乐项目点即1，2，…，m-1作为中间游乐项目点。因为不存在有负长度的回路，所以（4）项与（5）项中给出的2条路中较短的1条一定是从i到j的最短路，其中只容许前m个游乐项目点即游乐项目点1，2，…，m作为中间游乐项目点。

（4）上述（1）项和（2）项2条路的并。

（5）上述（3）项的路。

因此，$\mathrm{d}_{\mathrm{ij}}^{\mathrm{m}} = \min\left\{d_{im}^{m-1} + d_{mj}^{m-1}, d_{ij}^{m-1}\right\}$

从以上方程可以看出，只需要 D_{m-1} 矩阵的各个元素，就可以计算出矩阵 D_m 的各个元素；而且，无需参看基本图就可以进行计算。现在即使用Floyd算法求图中每一对游乐项目点之间最短路。

2、Floyd算法基本步骤：

Step 1：将图中各游乐项目点编为1，2，…，N。确定矩阵 D_0，其中(i,j)元素等于从游乐项目点i到游乐项目点j最短线段的长度(如果有最短线段的话)。对于i，令 $d_{ii}^0 = \infty$

Step 2：对m=1，2，…，N，依次由 D_{m-1} 的元素确定 D_m 的元素，应用下列递归公式 $\mathrm{d}_{\mathrm{ij}}^{\mathrm{m}} = \min\left\{d_{m}^{m-1} + d_{mj}^{m-1}, d_{ij}^{m-1}\right\}$ 每当确定一个元素时，就记下它所表示的路。在算法终止时，矩阵 D_n 的元素(i,j)元素就表示从游乐项目点i到游乐项目点j最短路的长度。

注意：

对所有的i和m ，$d_{ii}^{m}=\infty$，矩阵$D_1, D_2, ..., D_n$的对角线元素都无需计算，而且，对所有的i =1，2，…，n，$d_{im}^{m-1}=d_{im}^{m}$和$d_{mi}^{m-1}=d_{mi}^{m}$。这是因为不存在有负长度的回路，所以在游乐项目点m 处起始的任一最短路中，游乐项目点m 不是中间。因此，在矩阵D_m的计算中，第m 行和m 列都不需计算。在每一个矩阵D_m中，不在对角线上，也不在第 m 行和第 m 列的(N - 1) (N - 2)个元素需要计算。

根据以上算法思想对每一个游乐项目点到其它10个游乐项目点(将出入口假设为一个游乐项目点0)的最短距离利用Floyd算法进行计算(代码见附录代码1)，结果见表1：

	O	A	B	C	D	E	F	G	H	I	J
O	∞	300	400	700	1150	650	1650	1200	1350	900	550
A	300	∞	300	600	1050	350	1550	900	1050	600	250
B	400	300	∞	300	750	350	1250	900	1250	800	550
C	700	600	300	∞	450	500	950	1050	1400	950	850
D	1150	1050	750	450	∞	950	500	1150	1550	1400	1300
E	650	350	350	500	950	∞	1200	550	900	450	600
F	1650	1550	1250	950	500	1200	∞	650	1050	1500	1800
G	1200	900	900	1050	1150	550	650	∞	400	850	1150
H	1350	1050	1250	1400	1550	900	1050	400	∞	450	800
I	900	600	800	950	1400	450	1500	850	450	∞	350
J	550	250	550	850	1300	600	1800	1150	800	350	∞

表 1 各个游乐项目点到其它 10 个游乐项目点的最短距离

5.1.3 模型的建立

5.1.3.1 模型 1.1 的建立

根据以上信息进行计算机编程模拟仿真全天内游客的走动，具体编程思想如下：

1、游客从开园 9:00 开始至 20:00，按照时间间隔 T=−3.96ln U（U 是服从(0,1)上均匀分布的随机数）逐个进入游乐园；

2、假设每位游客手中都有一份游乐园地图，进入游乐园的游客在每次面临接下来去哪个游乐项目的选择时，优先在 800m 范围内的若干游乐项目中随机选择一个，如果 800m 范围内的所有游乐项目均已玩过，那么选择距离最近的一个未玩过的项目前往。具体行走的路径和距离根据 Floyd 算法计算。

3、游乐园模拟程序设计如下（使用 JAVA 语言 代码见附录代码 2）：

（1）类的说明：

该模拟程序应用了游客类（visitor）和游乐项目点类（youPoint）。

①游客类简要说明：

游客类包含该游客的基本信息并可以调用相应方法决定前往哪一个没有游玩过的游玩项目。游客类在每游玩一个项目将自检是否遍历所有游乐项目点，若已完成游园将自行出园。

②游乐项目点类简要说明：

游乐项目点中包含该游乐项目点的基本信息和在场内观赏的游客队列、等待中的游客队列和从该游乐项目点出发但还未到达目标游乐项目点的游客队列。游乐项目点类将在每一秒检查其他游乐项目点的出发队列中是否有游客在该秒到达该游乐项目点，并根据具体情况决定对该游客将采取何种操作。

注意：

游客自行出院选择路径为 Floyd 给出的最短路径；

园区大门是一个特殊的游乐项目点，假设它的观赏队列最大容量为 10000，保证不会有游客在大门口等待，它的项目总时间设置为 1 秒，使得在该游乐项目点中的游客在一秒后就离开该游乐项目点。

（2）程序运行概述：

①模拟程序以秒为时间单位，在每一秒执行相应的类操作实现游乐园的原始模拟。

②程序开始之前，根据公式 T=−3.96ln U（U 是服从(0,1)上均匀分布的随机数），将相邻游客的时间间隔加入一个队列之中。每当时钟节拍与之相等，实例化一个 visitor，设置初始信息之后，将其加入园区大门（大门也属于 youPoint 类）的观赏队列。之后，从 A 游乐项目点开始，依次调用所有游乐项目点的相应方法，对当前园内的所有游客进行相应的操作和调度，直至时钟节拍走完设定的总时间。

（3）程序记录的数据概述：

①游客游园总时间

②游客在每个游乐项目点等待的时间

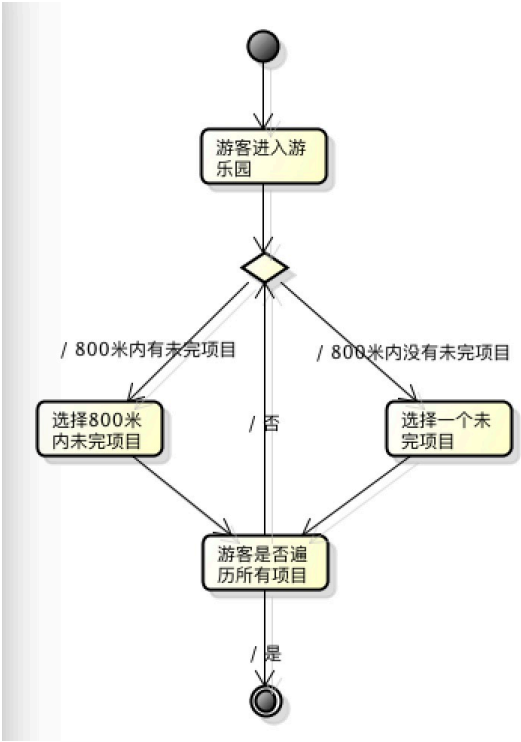
③每个项目游乐项目点在每一时钟节拍时对应的等待队列长度

（4）程序运行必要算法具体说明：

①游客如何决定下一点的走向（详见流程图）：

基于游客想遍历所有游乐项目的主观意愿和游客手中持有园区地图的基本假设，游客必将首先选择近距离内（800 米范围内）未游玩的项目游乐项目点。

- 如果存在满足该条件的游乐项目点，将它们全部找出作为可选择游乐项目点。如果不存在，则将所有未游玩的点全部找出选择一个作为可选择游乐项目点。
- 如果存在 800 米范围内未游玩的项目游乐项目点，则在这若干个游乐项目点中进行一次等概率的随机，选中一个点作为下一游乐项目点。如果 800 米范围内的游乐项目点均已游玩，则选定一个未游玩过的项目点作为下一游乐项目点。
- 以 1.5m/s 的步速和由 Floyd 算法得到的点对点最短路径计算出游客到达下一游乐项目点的具体时间，与当前时刻相加，得到游客到达下一游乐项目点的预估时间，作为路途中游客的属性。

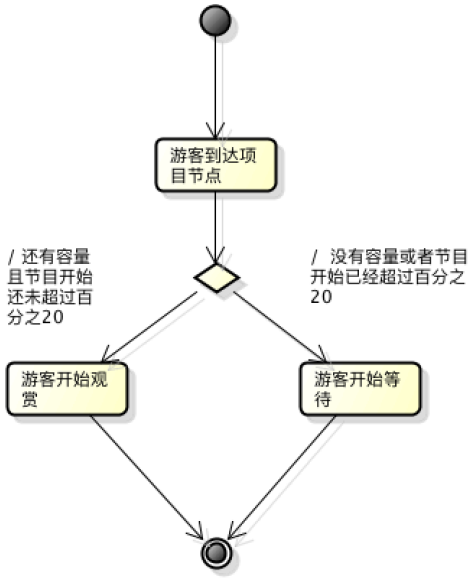


游客走向决定流程图

②对于到达某游乐项目点的游客，游乐项目点调度算法（详见流程图）：

基于游客想尽兴观赏的基本假设，游客到达游乐项目点时若节目已开场百分之二十，假设游客将宁愿等待。

- 如果该游客到达某游乐项目点时该游乐项目点的观赏游客数量未达到最大容量且节目开场时间还未超过总项目时长的百分之二十：游客将被该游乐项目点加入该游乐项目点的观赏队列。
- 如果该游客到达某游乐项目点时该游乐项目点的观赏游客数量未达到最大容量且节目开始时间超过节目总时长的百分之二十：游客将直接被该游乐项目点加入该游乐项目点的等待队列。
- 如果该游客到达某游乐项目点时该游乐项目点的观赏游客数量达到最大容量，由于缺乏疏导，游客将进入该游乐项目点的等待队列。



项目调度游客算法流程图

③游乐项目点对项目的管理：

在每一个时钟节拍中，游乐项目点都会检查当前项目是否到达散场时间，若在某时刻需要进行散场，将执行以下操作：

- 将观赏队列中的游客依次取出，并调用该游客的相应方法让游客自行决定接下来的去向。
- 将该游客加入该项目游乐项目点的出发队列。
- 将等待队列中的游客按先进先出的原则取出，置入观赏队列，直到观赏队列满员或者等待队列被取空

（5）数据获取：

至此得到原始模型 1.1；在进行过上百次仿真模拟后得到各个游乐项目点的等待队长随时间的全天分布，见图 1 中的红色曲线。并计算得出各个游乐项目点等待队长的期望值E_{ri}（$\mathrm{i}=1,2,3\ldots\ldots10$），以及每个游客游玩各个游乐项目所需等待时间的期望值E_{ti}（$\mathrm{i}=1,2,3\ldots\ldots10$）见图 2 中的红色柱状体。

（6）数据分析和问题的发现：

对由模型 1.1 得到的数据我们进行简要分析。分析发现：

①A 点、D 点、I 点的人员堆积情况由为严重。

②A 点、D 点、I 点周边点的等待人数却非常少。

由此我们提出以下几点设想：

①如果可以将 A 点、D 点、I 点的等待游客进行合理的分流，当等待时间明显过长时将他们疏导至其他距离范围合理的未玩游乐项目点，我们认为这样可以

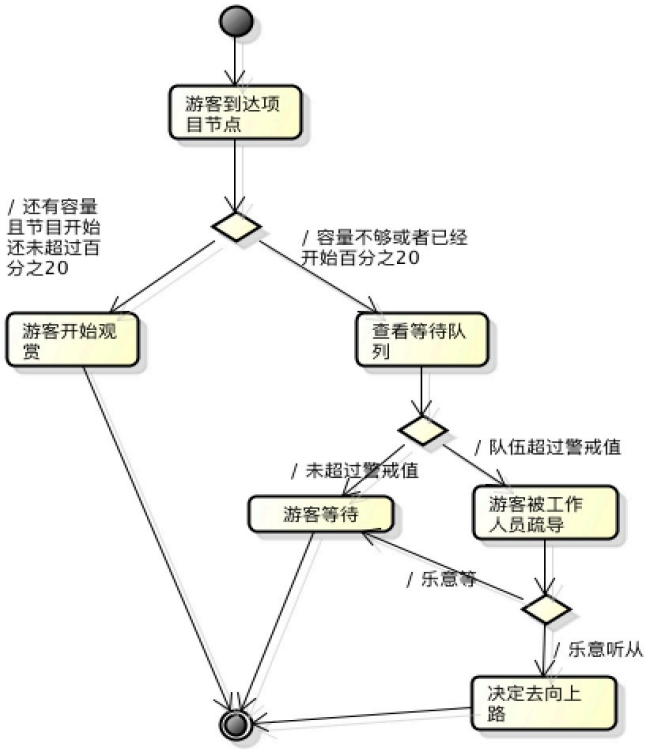
很大程度上减少拥挤、游乐项目点的平均队长和每位游客的总等待时长，从而优化游园体验。

②为每一个游乐项目点设置一个合理的等待人数警戒值W_{rimax}，当达到该值之后进行疏导。警戒值将根据模型 1.1 算法得到的数据分析计算得出，具体计算将在后文给出。

5.1.3.2 模型 1.2 的建立

针对问题分析，我们对模型 1.1 中的游乐项目点调度算法做出相应调整，调整如下：

② 添加当等待队列长度超出警戒值的情况（具体代码见附录代码 3 和流程图）



优化流程图

②针对模型 1.1 的结果进行调整计算，计算的结果是为了得到每一个游乐项目点的等待人数警戒值W_{rimax}，等待人数达到该值，工作人员就开始对游客进行疏导。初步调整计算以实现每一个游乐项目所需等待的时间期望值相等为目的，得到调整的大致范围。即$\frac{W_{\mathrm{rmaxi}}}{N_i}\times T_i \approx \mathrm{k}$（$\mathrm{i}=1,2,3\ldots\ldots 10;\mathrm{k}$ 为一待定常数）再根据上式反解出的 W_{rimax}（是一个包含 k 的式子）由$\sum_1^{10} W_{\mathrm{rimax}} = \sum_1^{10} E_{\mathrm{ri}}$得出初步的 $W_{\mathrm{rimax}}(\mathrm{i}=1,2,3\ldots\ldots 10)$值，计算结果见表 2.

A	B	C	D	E	F	G	H	I	J
104	206	172	103	172	172	129	172	114	215

表 2 初步计算得到的各游乐项目外等待人数警戒值

出于对实际情况的考虑，游乐园中往往有游客对某些项目抱有非常大的热情而忽视工作人员疏导意见的情况，此处我们赋予游客听从工作人员疏导的一个固定的概率值 90%。该值的设定同时也规避了在某游客只剩下一个游乐项目点需游玩而遭到疏导的尴尬局面。

更改优化程序段经过类似上述的仿真模拟后得到各个游乐项目点等待队长随时间的全天分布，并计算得出各个游乐项目点等待人数的期望值E_{ri}（$\mathrm{i}=1,2,3\ldots\ldots10$），以及每个游客游玩各个游乐项目所需等待时间的期望值E_{ti}（$\mathrm{i}=1,2,3\ldots\ldots10$）结合实际情况，反复调整各个警戒值并适当圆整，进行仿真模拟最终得到各个警戒值的一个较优解见表 3.

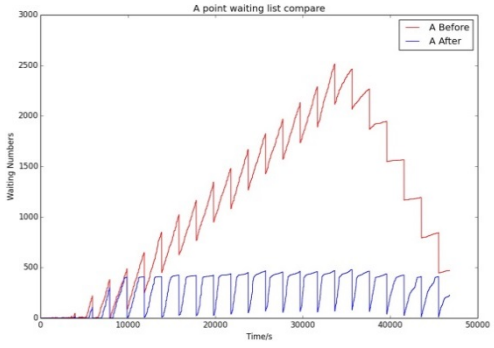
表 3 优化计算后得到的各游乐项目外等待人数警戒值

A	B	C	D	E	F	G	H	I	J
400	150	300	100	300	200	100	200	100	200

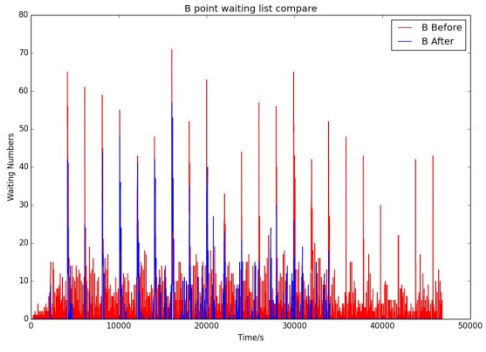
至此我们得到优化模型 1.2；在进行过上百次仿真模拟后得到各个游乐项目点的等待队长随时间的全天分布，见图 1 中的蓝色曲线。并计算得出每个游客游玩各个游乐项目所需等待时间的期望值E_{ti}（$\mathrm{i}=1,2,3\ldots\ldots10$）见图 2 中的蓝色柱状体。

注：在此我们对基本假设 1 做合理性检验：

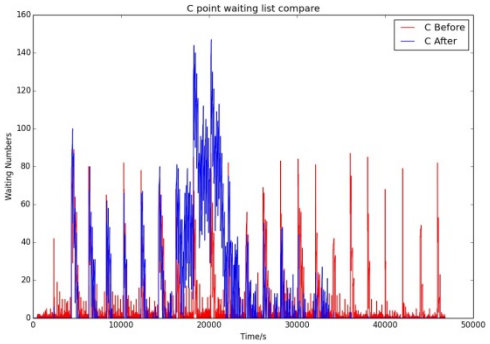
基于已建立的优化后模型 1.2，我们通过计算机编程计算得到平均每位游客游玩遍整个游乐园所需要的时间总共为 133 分钟，而在基本假设 1 中我们假设提前 2 个小时禁止游客入内，此提前量恰好约等于游客游遍整个游乐园所需时间，不会出现游客未玩遍整个游乐园就已经到关门时间的情况，从侧面证明了我们假设的合理性以及模型的合理性。



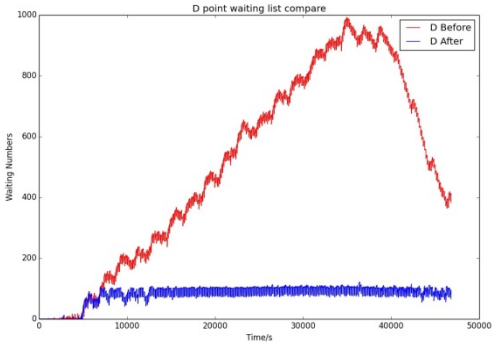
A



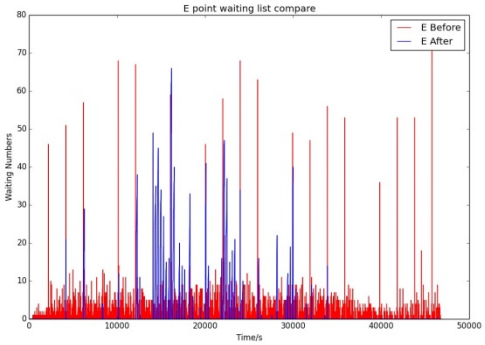
B



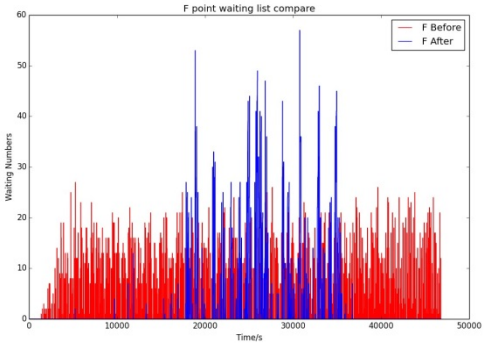
C



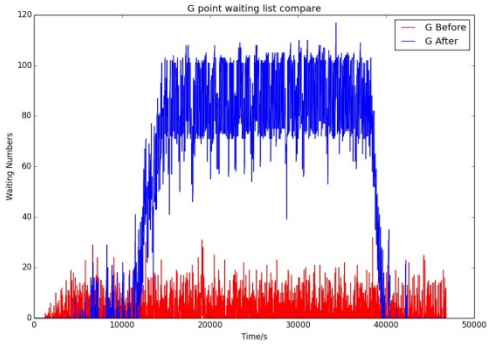
D



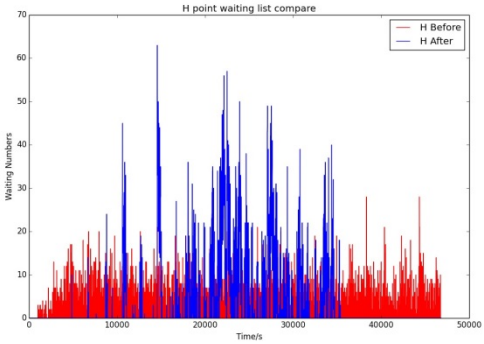
E



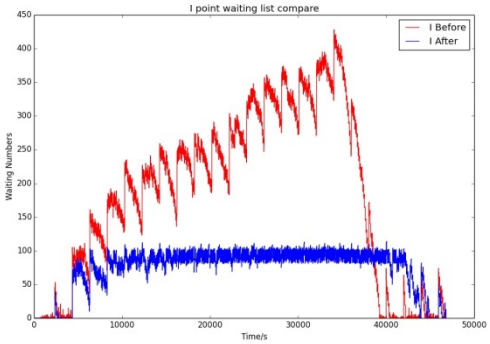
F



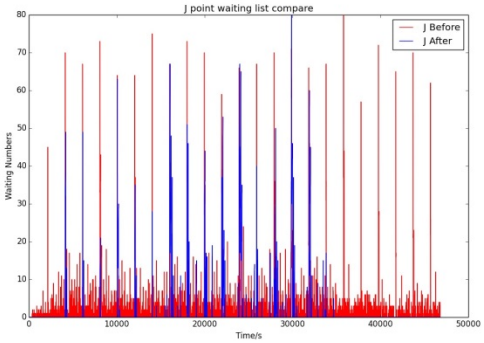
G



H



I



J

图 1 各个游乐项目点等待人数随时间的全天分布优化前后对比
（红色为优化之前，蓝色为优化之后）

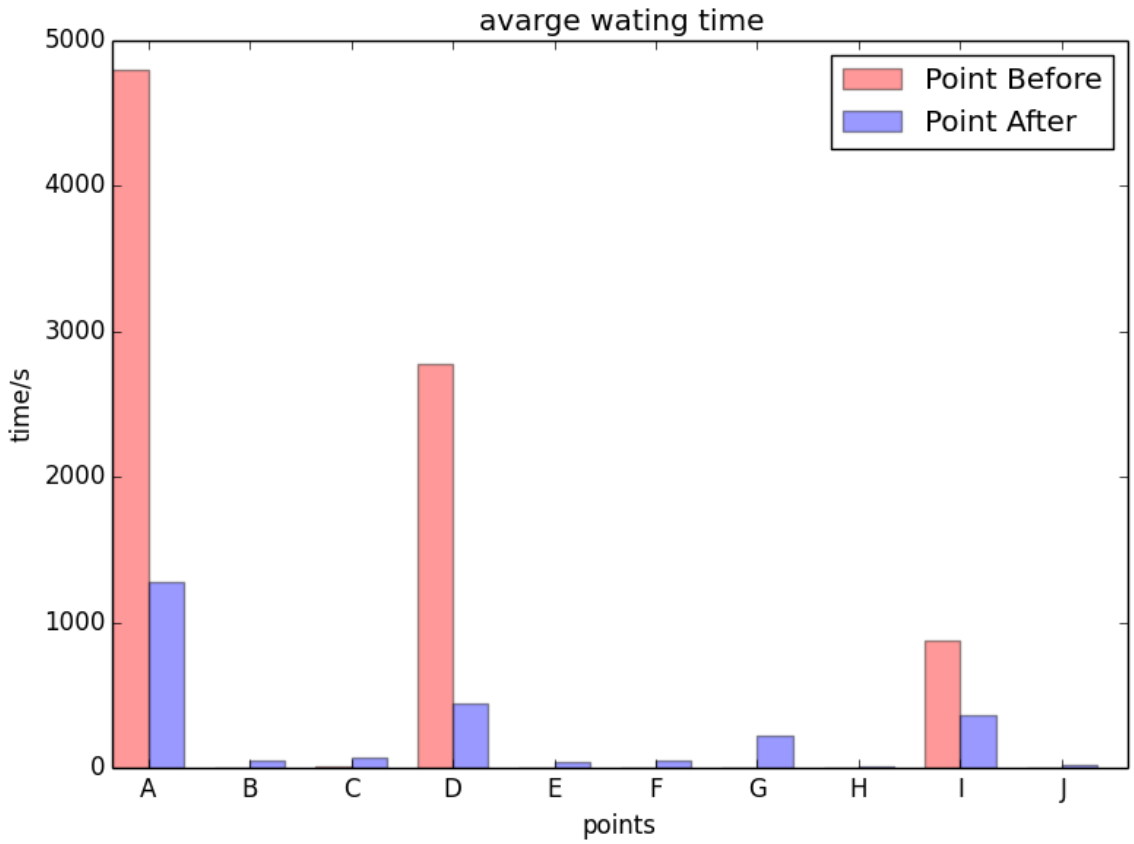


图 2 每个游客游玩各个游乐项目所需等待时间的期望值优化前后对比
（红色为优化之前，蓝色为优化之后）

5.1.4 游乐园疏导模型的结论总结

综上，根据我们建立起的数学模型，对每个游乐项目的等候游客进行游览提醒和疏导的具体方案如下：

1、在游乐项目的等候游客数不超过警戒值时，不进行提醒和疏导；

2、当游乐项目的等候游客数超过警戒值时，工作人员开始提醒并疏导堆积的游客前往下一个游乐项目（各游乐项目外等待人数警戒值见表 3）

5.2 酒店预测模型的建立与求解：

5.2.1 数据准备：

从题目所给数据中处理出 2015 年每天的入住房间数 N（详见附录表 1）N 在全年的分布见图 3

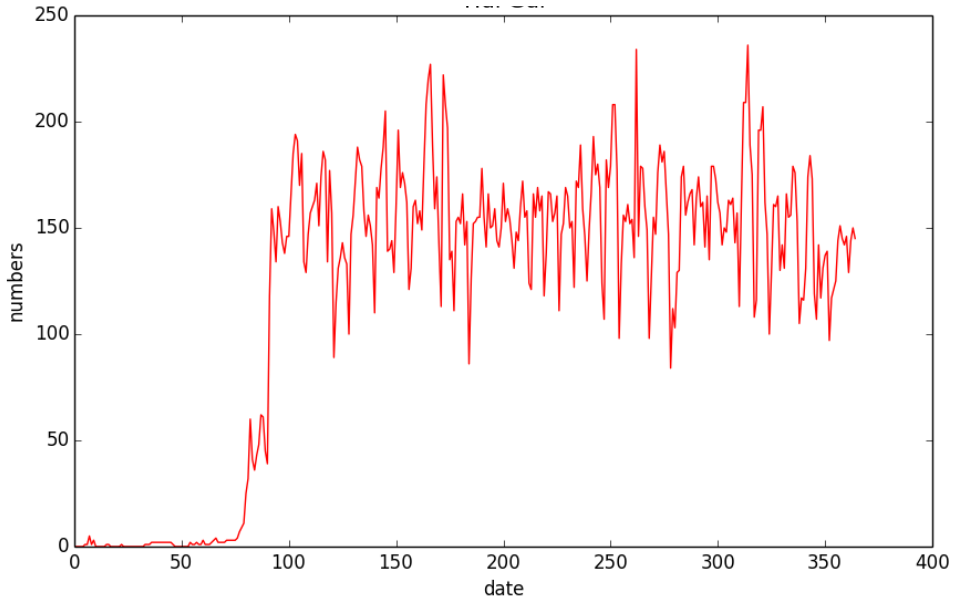


图 3 2015 年每天的入住房间数 N 全年的分布

5.2.2 关于直接影响 2016 年 1 至 3 月预定房间数的日期区间的分析计算

根据问题二基本假设 2，每位旅客在预订房间时都已经掌握了游玩当日的季节，周末/工作日，是否是寒暑假，是否是法定节假日这些信息。我们有理由相信这些因素会直接影响每天的入住房间数，进而影响每天的预定房间数。为了预测 2016 年 1 至 3 月每天的预定房间数，我们需要参照逆推的思想，先确定旅客会在 2016 年 1 至 3 月预定哪些天的房间（设为日期区间$[T_1, T_2]$，显然T_1已知为 2016 年 1 月 1 日）。关于T_2的计算，我们首先采用聚类分析的思想做了以下分析处理：

1、对于入住日期为工作日的情况：

先求得 2015 全年所有工作日的入住房间数总和S_1，再计算提前 n 天预定工作日入住的房间总数s_n(n=0, 1, 2, 3……)，求$\frac{S_n}{S_1}$即为提前 n 天预定工作日入住发生的频率，可视为这些事件发生的概率，详见附录表 2，分布见图 4。

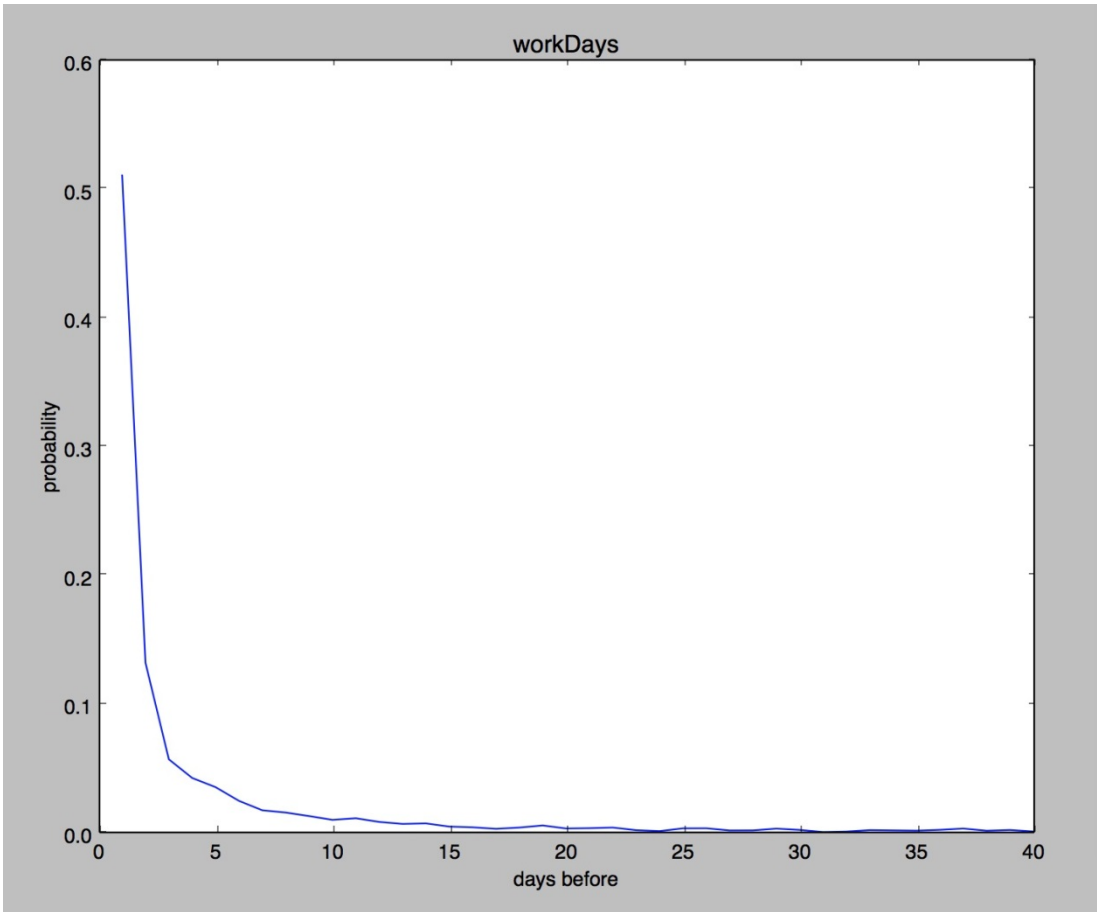


图 4 提前 n 天预定工作日入住的概率分布

2、对于入住日期为周末的情况：

先求得 2015 全年所有周末的入住房间数总和S_2，再计算提前 m 天预定周末入住的房间总数s_m(m=0,1,2,3……)，求$\frac{s_m}{S_2}$即为提前 m 天预定周末入住发生的频率，可视为这些事件发生的概率，详见附录表 3，分布见图 5。

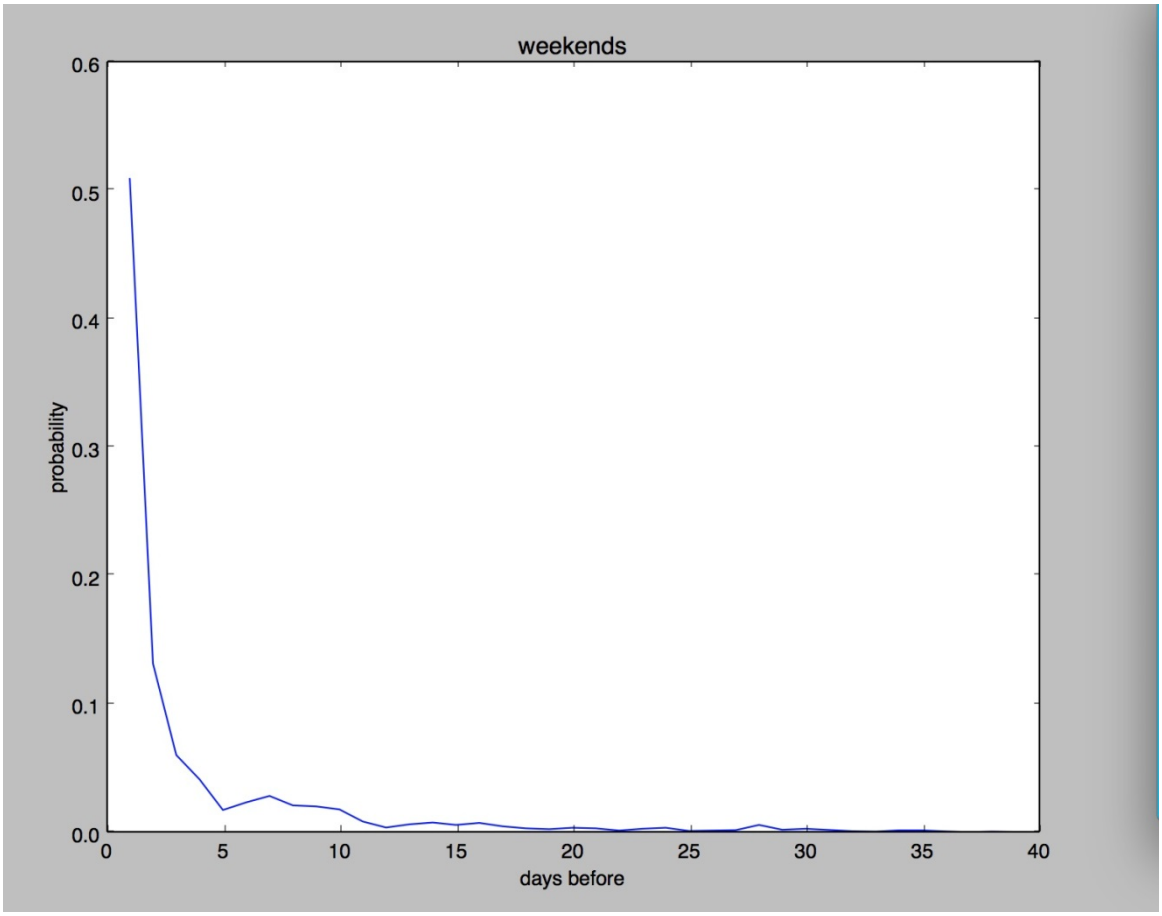


图 5 提前 m 天预定周末入住的概率分布

经过以上分析，我们可以看出无论是周末还是工作日，提前 40 以上天预定房间的概率很小，接近于 0，且提前 1-40 天预定的概率总和经过计算，工作日已达到 97.39%，周末已达到 96.42%。所以，我们默认所有人的提前预定天数都不会超过 40 天。由此我们得出T_2为 2016 年 5 月 10 日。即旅客在 2016 年 1 至 3 月预定的都是 2016 年 1 月 1 日-2016 年 5 月 10 日的房间。

5.2.3 2016 年 1 月 1 日-2016 年 5 月 10 日入住量的分析预测

1、回归拟合函数式的获得：

根据 5.2.1 的分析计算，我们得知要预测 2016 年 1 至 3 月每日预定房间数就首先要预测 2016 年 1 月 1 日-2016 年 5 月 10 日每日入住房间数。为此本文建构多元线性回归的统计预测模型 2，将季节，周末/工作日，是否是寒暑假，是否是法定节假日，外界环境是否发生剧烈变化等影响因素提取为：竞争对手是否倒闭（R），是否为夏季（S），是否为秋季（F），是否为冬季(WI)，是否为周末(WE)，是否处在三天法定节假日内(H_3)，是否处在七天法定节假日内（H_7），是否处在寒假（WH），是否处在暑假(SH)这 9 个虚拟变量（其定义域只有 0 和 1 两个数值，是则为 1，否则为 0。而其它诸如春季，工作日，非节假日以及非寒暑假分别为各组组合虚拟变量中的基变量）结合 2015 年每天的入住房间数 N（已整理为附录表 1）利用这些数据导入 stata 软件进行回归处理。处理结果见图 6。

people	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
rival	127.7256	5.097181	25.06	0.000	117.7011	137.7501
summer	-2.051406	4.154168	-0.49	0.622	-10.22128	6.118467
autumn	-8.324522	3.395906	-2.45	0.015	-15.00315	-1.645899
winter	-29.46454	5.150197	-5.72	0.000	-39.59328	-19.33581
weekend	-13.17855	2.591519	-5.09	0.000	-18.27521	-8.081891
holiday3	-12.12209	5.965011	-2.03	0.043	-23.85329	-.3908884
holiday7	4.573348	6.439767	0.71	0.478	-8.091541	17.23824
sumholiday	-6.063032	4.576105	-1.32	0.186	-15.06271	2.936651
winholiday	-3.191638	5.310092	-0.60	0.548	-13.63483	7.251556
_cons	36.67665	5.014367	7.31	0.000	26.81505	46.53825

图 6 stata 软件进行回归处理结果

得到 N 关于 9 个虚拟变量的函数关系如下：

N=36.67665+127.7256R-2.051406S-8.324522F-29.46454WI-13.17855WE-12.12209H_3+4.573348H_7-6.063032SH -3.191638WH.

5.2.4 关于各种情况下提前预定天数的预测分析

由于在 1 月 1 日-5 月 10 日所包含的节假日较少，且 2015 年所给相关数据很少。因此为了简化模型，我们在此仅考虑工作日与周末对于提前预定天数的影响。且提前 n 天预定工作日入住的概率分布和提前 m 天预定周末入住的概率已经 5.2.1 中计算过，详见附录表 2，表 3。

至此，我们已建立起完整的多元线性回归的统计预测模型 2。

5.2.5 最终预测结果的计算

基于 2016 年的日历和基本假设，可以确定 2016 年 1 月 1 日-5 月 10 日每一天竞争对手是否倒闭（R），是否为夏季（S），是否为秋季（F），是否为冬季(WI)，是否为周末(WE)，是否处在三天法定节假日内(H_3)，是否处在七天法定节假日内（H_7），是否处在寒假（WH），是否处在暑假(SH)这 9 个变量的值，整理见附录表 4;

接下来通过计算机编程处理：（代码见附录代码 4）

- 利用函数关系式

  N=36.67665+127.7256R-2.051406S-8.324522F-29.46454WI-13.17855WE-12.12209H_3+4.573348H_7-6.063032SH -3.191638WH.

预测计算出 2016 年 1 月 1 日-5 月 10 日每一天的入住房间数，装入数组 Num。整理见附录表 4。

- 记每一天的预订提前概率为 Pi，i 为提前量。
- 假设 Day 为储存每天预定量的数组，记 now 为当天，则有以下关系成立，Day[now - i] = Day[now - i] + Num[now] * Pi。
- 二重循环遍历 now 与 i 即可得到 Day 数组，即我们的最终预测结果，详见表 4

表 4 预定房间数的最终预测结果

日期	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10

预定量	110	112	131	133	130	131	130	118	118	131
日期	1.11	1.12	1.13	1.14	1.15	1.16	1.17	1.18	1.19	1.20
预定量	133	130	131	130	117	118	131	133	130	131
日期	1.21	1.22	1.23	1.24	1.25	1.26	1.27	1.28	1.29	1.30
预定量	130	117	118	131	133	130	131	130	117	118
日期	1.31	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9
预定量	129	131	129	130	129	117	118	133	135	132
日期	2.10	2.11	2.12	2.13	2.14	2.15	2.16	2.17	2.18	2.19
预定量	133	132	119	119	130	133	130	131	130	118
日期	2.20	2.21	2.22	2.23	2.24	2.25	2.26	2.27	2.28	2.29
预定量	119	133	136	133	135	135	122	125	145	162
日期	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	3.10
预定量	159	160	159	145	146	160	162	159	160	159
日期	3.11	3.12	3.13	3.14	3.15	3.16	3.17	3.18	3.19	3.20
预定量	145	146	160	162	159	160	159	145	146	160
日期	3.21	3.22	3.23	3.24	3.25	3.26	3.27	3.28	3.29	3.30
预定量	163	160	161	160	146	147	161	163	161	162
日期	3.31									
预定量	159									

六、误差可信度分析

6.1 酒店预测模型误差可信度分析：

根据 stata 回归结果，我们可以发现 9 个虚拟变量中显著性较强的分别有对手是否倒闭（R），是否为冬季(WI)，是否为周末(WE)，是否为三天假期（H_3），这也就说明了竞争对手是否倒闭对于酒店入住量是具有显著影响的，同时是否为周末、是否为冬季，是否为三天假期这三个性质也会明显影响入住量。剩下的变量虽然可信度没有那么强，但我们可以将他们包括在内，来更好地拟合这一函数关系。

七、模型的检验

7.1 酒店预测模型检验：

（1）拟合度检验：

利用 5.2.2 回归分析出的 N 关于 9 个虚拟变量的函数关系：

$N=36.67665+127.7256R-2.051406S-8.324522F-29.46454WI-13.17855WE-12.12209H_3+4.573348H_7-6.063032SH-3.191638WH.$

结合 2015 年全年每天的 9 个虚拟变量的具体数值（见附录表 1）可以绘制按照上式函数关系得到的 2015 年每天入住房间数，与实际入住量对比见图 7.

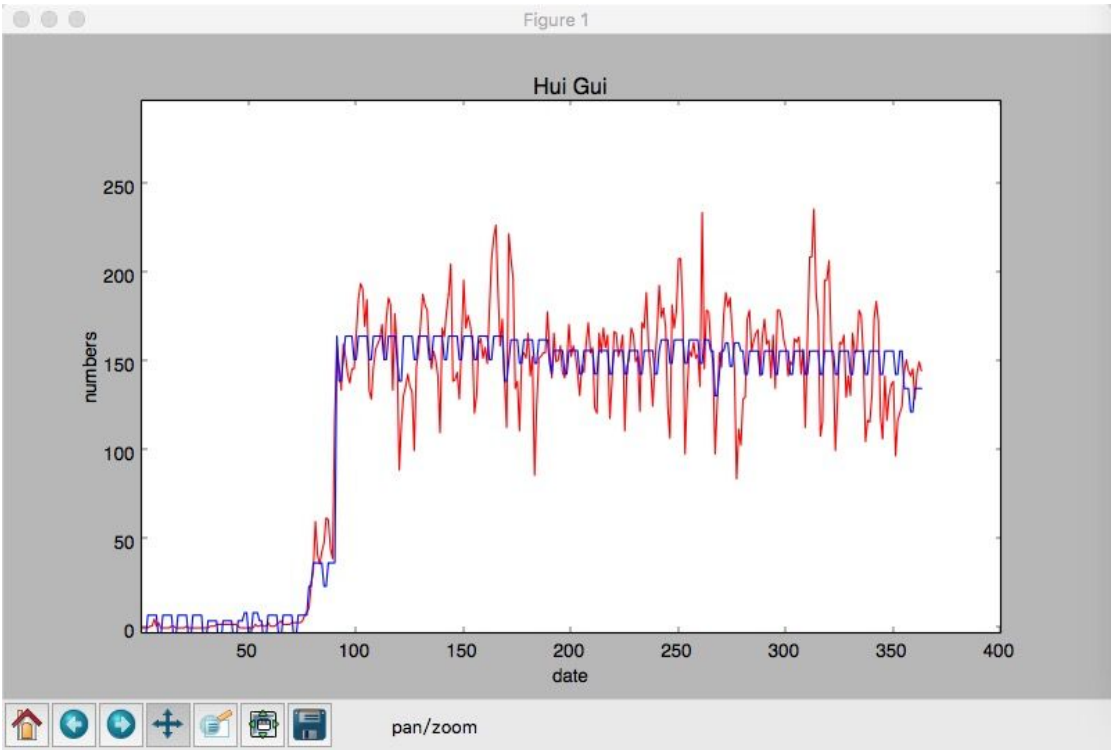


图 7 2015 年每天入住房间数回归分析（蓝色）与实际数据（红色）对比

由上图可见，本文所做的回归分析与实际情况较为贴合，因此运用模型 2 进行预测分析具有较高的可信度。

（2）实际情况检验：

我们可以从题目所给信息，在 2015 年末已经有一部分顾客预定了 2016 年初的房间，且已经给出了到店日期和离店日期，由此我们可以计算出 2016 年初几天的实际最小入住量，且根据模型 2 可以给出 2016 年初几天的预测入住量，现对比见表 5.

表 5 2016 年初几天的实际最小入住量与预测入住量对比

	2016/1/1	2016/1/2	2016/1/3	2016/1/4	2016/1/5	2016/1/6
实际最小入住	140	99	38	25	24	9
预测入住	123	110	110	135	135	135

由对比结果，我们可以发现，根据模型 2 预测出的入住量绝大部分大于根据题目信息推算的实际最小入住量，并考虑相关的概率，我们发现这两个数据在误差允许范围内满足模型 2 中的相关关系。

这再一次验证了我们所建立模型的合理性，可靠性。

八、模型的评价与改进

8.1 游乐园疏导模型的评价与改进

8.1.1 评价

该问题的最终目的是为了实现每位游客游园体验最佳，这涉及到很多因素，比如游乐项目外等待区的舒适程度，比如出入口以及道路的拥挤程度等等，本问题所建立的最终优化模型 1.2 是在将这些无法量化的因素忽略后建立的较为理想化的模型，主要参照了图论与排队论的思想，运用了计算机仿真模拟，并解决了原始模型 1.1 中不加以疏导，将会导致的极个别游乐项目前人流大量堆积的问题，使得每个游乐项目前的队长都得到了大大的优化。具有一定的理论价值和实用意义。

8.1.2 改进

我们发现当前模型 1.2 还有一些可改进的点，现总结如下：

（1）本文在对游客在游乐园内的走动进行仿真模拟时，对游客听取工作人员引导的概率设置为固定值 90%，而现实情况中此项概率是会随着等待人数的不同，去下一个游乐项目的距离的不同而变化的，即等待的人数越少，去下一个游乐项目的距离越远，游客不听从指导的概率越大。该概率和队列长度、距离远近的关系可以经过大量问卷调查和样本分析后得到。若能应用于模型则可进一步提高仿真度。

（2）本文假设所有游乐项目对游客的吸引力是相同的，但实际情况显然不是这样的，在游客面临随机选择游乐项目的时候，赋予他各个游乐项目的吸引力系数可以大大提高仿真度。具体各个游乐项目对于游客的吸引力还需要大量的现场观测以及问卷调查获得。

（3）本文设定游客在面临接下来前往哪一个游乐项目点时，会优先在 800m 范围内随机选择，而进一步的模型优化可以令他们在 800m 范围内的游乐项目选择符合一定的概率分布，选择距离较近的游乐项目概率更高，反之较低。这样优化模型会更加符合实际。

（4）考虑到实施成本问题，我们在整个分析，建模，优化的过程中都没有考虑实时通信的可能，如果要进一步优化模型和方案，可以在计算机仿真模拟系统中加入实施通信，即将每一个游乐项目点的等待人数及项目开始结束时间通报给每一位游客，再来根据每位游客自身的属性决定下一步最佳的行动方案。这样不仅仅会进一步优化每个游乐项目点前的平均等待时间，而且会优化每位游客游遍全园的总体时间，以及在路上行走所耗费的时间等，达到最优。

（5）本文所建立的模型设定等待人数一旦达到警戒值值，工作人员就开始对游客进行疏导，在进一步优化中可以综合考虑等待人数超出警戒值的多少，以及此时距离该游乐项目下一场表演开始的时间得到工作人员进行疏导的强度，即等待人数越少，距离下一场表演开始时间越短，工作人员进行疏导的积极性越低。这样更加符合实际，可以提高模型的可信度。

8.2 问题二模型的评价与改进

8.2.1 评价

该问题的最终目的是实现对未来一段时间酒店预定量的预测，而实际情况中影响一个酒店预定量的因素有很多，比如天气，竞争对手的新引力，房间价格的

调整等等。而模型 2 是在忽略这些无法量化的因素后建立的模型。主要运用多元线性回归预测，合理地将题目要求并给出的各个影响因素综合考虑在内。在分析提前预定天数时创新性地使用了聚类分析的思想，在不影响预测结果的前提下合理地简化了模型 2，可以说在题目信息允许的条件下，模型 2 的预测结果已经是较为合理和可信的了。

8.2.2 改进

（1）在模型 2 的建立过程中，关于周末的处理仅设置了是否为周末（WE）这一个虚拟变量，但实际情况中周六和周日的影响是不同的，因此，为了进一步优化模型 2，还可以将其设置为两个变量，即是否为周六(SA)和是否为周日(SU)。这样，再运用 stata 软件进行回归处理后，会得到更为精确的回归方程式。且对于季节的处理也可以更加细化，以提高预测的准确度。

（2）如果拥有足够充分的数据，比如天气，市场竞争情况等等，我们可以添加若干变量，进一步优化预测模型 2，可以得到更为精准的预测模型。

（3）本文的模型 2 的建立我们仅仅尝试了线性回归，已经得到了较为理想的拟合曲线，如果时间足够，我们还可以用单一变量法分析每一个变量对结果的影响。进而用不同的函数关系（可以是非线性的）去拟合每一个变量，这样可以得到近乎完美的回归函数，实现误差最小的预测。

（4）模型 2 中在计算 15 年年初的入住量时并没有考虑 14 年预定但是 15 年入住的情况，会导致 15 年年初的入住量偏小，如果有 14 年的预定情况，预测结果将为更加理想。

九、参考文献

[1]埃克尔. Java 编程思想. 上海，机械工业出版社，2007.6
[2]戴明强，宋业新. 数学模型及其应用—2 版. 北京，科学出版社，2015.2
[3]郭亚军. 旅游景区管理—1 版. 北京，高等教育出版社，2006.5
[4]蒋启源，谢金星，叶俊. 数学模型—4 版. 北京，高等教育出版社，2011.1
[5]同济大学概率统计教研组. 概率统计—4 版. 上海，同济大学出版社，2009.6
[6]伊德里斯. Python 数据分析基础教程:NumPy 学习指南—2 版. 人民邮电出版社,2014.1
[7]詹姆斯·斯托克，马克·W.沃森. 计量经济学. 上海，格致出版社，2012.3
[8]百度用户. 旅游景点最优化模型.
http://wenku.baidu.com/link?url=IvhCu7DQLqVk23Vf-RxqBODENVrtUs-E0j8WgkX-Q-e1FPNMtbPEY97f-yeL-XIZfONIAsU2GOYWAYlsOWhhOUVhc6qaruNtX3FL_R14FF3&qq-pf-to=pcqq.group. 2016.4.30

十、附录

代码 1. Folyd 算法代码如下：

```
#include <iostream>
#include <vector>
```

```
#include <memory>
#include <cstdio>

using namespace std;

int Map[11][11]={
                10000, 300, 400, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
                300, 10000, 300, 10000, 10000, 350, 10000, 10000, 10000, 250,
                400, 300, 10000, 300, 10000, 350, 10000, 10000, 10000, 10000,
                10000, 10000, 300, 10000, 450, 500, 10000, 10000, 10000, 10000,
                10000, 10000, 10000, 450, 10000, 10000, 500, 10000, 10000, 10000,
                10000, 350, 350, 500, 10000, 10000, 10000, 550, 10000, 450, 10000,
                10000, 10000, 10000, 10000, 500, 10000, 10000, 650, 10000, 10000,
                10000, 10000, 10000, 10000, 10000, 550, 650, 10000, 400, 10000,
                10000, 10000, 10000, 10000, 10000, 10000, 10000, 400, 10000, 450, 10000,
                10000, 10000, 10000, 10000, 10000, 450, 10000, 10000, 450, 10000, 350,
                10000, 250, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 350, 10000
};

char x[11] = {'0', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};

int Path[11][11];

int main() {
        for (int l = 0; l < 11; ++l) {
                for (int i = 0; i < 11; ++i) {
                        Path[l][i] = i;
                }
        }
        for (int i = 0; i < 11; ++i) {
```

```
                for (int j = 0; j < 11; ++j) {
                        for (int k = 0; k < 11; ++k) {
                                if (k == j)
                                        continue;
                                if (Map[j][i] + Map[i][k] < Map[
j][k]){
                                        Map[j][k] = Map[j][i] + M
ap[i][k];
                                        Path[j][k] = Path[j][i];
                                }
                        }
                }
        }
        cout << "        ";
        for (int n = 0; n < 11; ++n) {
                cout << x[n] << "        ";
        }
        cout << endl;
        for (int m = 0; m < 11; ++m) {
                cout << x[m] << ": ";
                for (int i = 0; i < 11; ++i) {
                        cout << Map[m][i] << " ";
                }
                cout << endl;
        }

        int f,en;
        while (cin >> f >> en) {
                while (f != en) {
                        printf("%d->", f);
                        f = Path[f][en];
                }
                printf("%d\n", en);
        }
        return 0;
}
```

代码 2.游乐园模拟程序关键代码如下：

1.1 游客类属性代码展示

```
public class visitor {
    // 已经玩过的点
```

```
    private ArrayList<Integer> playedPoint;
    // 未玩过的点
    private boolean [] unPlayedPoint;
    // 当前所在节点
    private int currentPos;
    // 入园时间
    private int inTime;
    // 出园时间
    private int outTime;
    // 游客去往下一点的时间代价
    public double value;
    // 游客决定前往下一节点的时间节点
    private int decideTime;
    // 游客决定下一节点的编号
    private int to;
    // 游客是否在出园过程中
    private boolean exit;
    // 记录游客的等待时间
    public Integer [] waitTime = new Integer[11];
}
```

1.2 节点类代码展示

```
public class youPoint{
    // 节点名称
    private  int name;
    // 项目当前进行时间
    public   int goTime;
    // 项目所需总时长
    private  int runTime;
    // 游客等待队列
    public Stack<visitor> waitList;
    // 游客观赏队列
    private Stack<visitor> inList;
    public ArrayList<visitor> passList;
    // 游客出发队列
    private int fullNum;

    /*优化方案中添加的警戒值*/
    public int warnNum;
}
```

1.3 游客选取节点代码展示

```
    public void next() throws IOException {
        // 如果游客遍历玩所有节点
```

```
        if(playedPoint.size() == 11){
            int chuqu = (int) (outTime + youleyuan.minRoad[currentPos][0]
/ 1.5 - inTime);
            for (int i = 1; i <= 10; i++){
                writer.write(waitTime[i] + " ");
            }
            exit = true;
            return;
        }
        ArrayList<Integer> can = new ArrayList<Integer>();
        // 优先选取 800 米以内可行节点，随机前往
        for (int i = 0; i < 11; i++){
            if(unPlayedPoint[i] && youleyuan.minRoad[currentPos][i] <=
800){
                can.add(i);
            }
        }
        int num = (int) (Math.random() * can.size());
        if(num >= 0 && can.size() != 0) {
            int choose = can.get(num);
            decideTime = youleyuan.staticTime;
            to = choose;
            value = youleyuan.minRoad[currentPos][to] / 1.5;
        }
        // 800 米内无解，选取就近点
        else{
            can.clear();
            for (int i = 0; i < 11; i++){
                if(unPlayedPoint[i]){
                    can.add(i);
                }
            }
            int choose = can.getMin()
            decideTime = youleyuan.staticTime;
            to = choose;
            value = youleyuan.minRoad[currentPos][to] / 1.5;
        }
    }
```

1.4 节点原始调度模拟算法代码展示：

```
    public void go() throws IOException {
        goTime++;
        // 取等待队列
```

```
        while (inList.size() < fullNum && !waitList.isEmpty()){
                inList.add(waitList.pop());
        }

        // 取点
        for (int i = 0; i < 11; i++){
            for (int k = 0; k < youleyuan.points.get(i).passList.size();
k++){
                visitor temp =
youleyuan.points.get(i).passList.get(k);
                // 获取在该时刻到达该点的游客
                if(temp.getValue() + temp.getDecideTime() <
youleyuan.staticTime && temp.getTo() == name){
                    temp.setCurrentPos(name);
                    // 如果观赏队列没有满
                    if(inList.size() < fullNum){
                        inList.add(temp);
                    }else{
                        // 否则加入等待队列
                        temp.waitTime[name] = waitList.size() / fullNum
* runTime;
                        waitList.add(temp);

                    }
                    youleyuan.points.get(i).passList.remove(k);
                    k--;
                }
            }
        }


        // 散场
        if(goTime == runTime){
            goTime = 0;
            for (int i = 0; i < fullNum; i++){
                if(inList.isEmpty()){
                    break;
                }
                visitor temp = inList.pop();
                temp.addPlayed(name);
                temp.setUnPlayedPoint(name);
                try {
```

```
                    temp.setOutTime(youleyuan.staticTime);
                    temp.setTo(-1);
                } catch (IOException e) {
                    e.printStackTrace();
                }
                temp.next();
                if(!temp.getExit()) {
                    passList.add(temp);
                }
            }
            // 取等待队列
            while (inList.size() < fullNum && !waitList.isEmpty()){
                inList.add(waitList.pop());
            }
        }
    }
```

代码 3.节点优化调度算法代码如下：

```
// 取点
        for (int i = 0; i < 11; i++){
            for (int k = 0; k < youleyuan.points.get(i).passList.size();
k++){
                visitor temp =
youleyuan.points.get(i).passList.get(k);
                // 获取在该时刻到达该点的游客
                if(temp.getValue() + temp.getDecideTime() <
youleyuan.staticTime && temp.getTo() == name){
                    if(inList.size() < fullNum){
                        temp.setCurrentPos(name);
                        inList.add(temp);
                        youleyuan.points.get(i).passList.remove(k);
                        k--;
                    }else{
                        // 如果等待队列中的等待人数未超出我们设置的警
戒值
                        if(waitList.size() <= warnNum) {
                            temp.setCurrentPos(name);
                            temp.waitTime[name] = (waitList.size() /
fullNum + 1) * runTime - goTime;
                            waitList.add(temp);

youleyuan.points.get(i).passList.remove(k);
                            k--;
```

```
                        }else {
                            // 如果超出我们设置的境界值
                            while (waitList.size() > warnNum) {
                                // 90%听从建议
                                double choose = Math.random();
                                if (choose <= 0.1) {
                                    // 不听从则直接进入等待队列
                                    temp.setCurrentPos(name);
                                    temp.waitTime[name] =
(waitList.size() / fullNum + 1) * runTime - goTime;
                                    waitList.add(temp);

youleyuan.points.get(i).passList.remove(k);
                                    k--;
                                    break;
                                } else {

youleyuan.points.get(i).passList.remove(k);
                                    k--;
                                    temp.setCurrentPos(name);
                                    // 再次决定去向
                                    temp.next();
                                    passList.add(temp);
                                    break;
                                }
                            }
                        }

                    }
                }
            }
        }
```

游乐园运作展示：

```
// 应用泊松流计算出相邻游客的到达时间间隔
        Queue<Integer> people = new LinkedList<Integer>();
        int count = 0;
        for (int i = 0; i < 10000; i++) {
            int dl = (int)(-3.88 * Math.log(Math.random()));
            count += dl;
            people.add(count);
        }
        while(staticTime <= 46800){
```

```
            // 实例化游客并加入大门节点中
            while(!people.isEmpty() && staticTime == people.peek()){
                visitor vis = new visitor(staticTime, visitorTimeLog);
                vis.addPlayed(0);
                vis.setUnPlayedPoint(0);
                vis.next();
                points.get(0).getIn(vis);
                count++;
                people.poll();
            }
            // 时钟前进一秒
            staticTime++;
            // A 点进行操作
            A.go();
            // B 点进行操作
            B.go();
            // C 点进行操作
            C.go();
            // D 点进行操作
            D.go();
            // E 点进行操作
            E.go();
            // F 点进行操作
            F.go();
            // G 点进行操作
            G.go();
            // H 点进行操作
            H.go();
            // I 点进行操作
            I.go();
            // J 点进行操作
            J.go();
        }
```

代码 4.预定量计算代码如下：

```
data = xlrd.open_workbook('2016.xlsx')
    table = data.sheets()[0]
    days = []
    stay = []
    for day in range(START1, START5):
        days.append(0)
```

```
    for day in range(0, 131):
        a = table.row_values(day)[0]
        b = table.row_values(day)[1]
        c = table.row_values(day)[2]
        d = table.row_values(day)[3]
        e = table.row_values(day)[4]
        f = table.row_values(day)[5]
        g = table.row_values(day)[6]
        h = table.row_values(day)[7]
        i = table.row_values(day)[8]
        y =
36.67665+127.7256*a+2.051406*b-8.324522*c-29.46454*d-13.17855*e+12.12
209*f-4.573348*g-6.063032*h-3.191638i
        stay.append(y)
        # print y
        for i in range(0, 40):
            if (day + 1) % 7 == 0 or (day + 2) % 7 == 0:
                try:
                    days[day - i - 1] += y * weekend[i]
                except:
                    pass
            else:
                try:
                    days[day - i - 1] += y * workDay[i]
                except:
                    pass
    print days
```

表 1：2015 年每天 9 个虚拟变量的值与实际入住房间数

日期	竞争对手是否倒闭	夏天	秋天	冬天	周末	3 天假期	7 天假期	暑假	寒假	入住房间数
2015 年 1 月 1 日	0	0	0	1	0	1	0	0	0	0
2015 年 1 月 2 日	0	0	0	1	0	1	0	0	0	0
2015 年 1 月 3 日	0	0	0	1	1	1	0	0	0	0
2015 年 1 月 4 日	0	0	0	1	1	0	0	0	0	0
2015 年 1 月 5 日	0	0	0	1	0	0	0	0	0	0
2015 年 1 月 6 日	0	0	0	1	0	0	0	0	0	0
2015 年 1 月 7 日	0	0	0	1	0	0	0	0	0	1
2015 年 1 月 8 日	0	0	0	1	0	0	0	0	0	1
2015 年 1 月 9 日	0	0	0	1	0	0	0	0	0	5
2015 年 1 月 10 日	0	0	0	1	1	0	0	0	0	1

2015年1月11日	0	0	0	1	1	0	0	0	0	3
2015年1月12日	0	0	0	1	0	0	0	0	0	0
2015年1月13日	0	0	0	1	0	0	0	0	0	0
2015年1月14日	0	0	0	1	0	0	0	0	0	0
2015年1月15日	0	0	0	1	0	0	0	0	0	0
2015年1月16日	0	0	0	1	0	0	0	0	0	0
2015年1月17日	0	0	0	1	1	0	0	0	0	1
2015年1月18日	0	0	0	1	1	0	0	0	0	1
2015年1月19日	0	0	0	1	0	0	0	0	0	0
2015年1月20日	0	0	0	1	0	0	0	0	0	0
2015年1月21日	0	0	0	1	0	0	0	0	0	0
2015年1月22日	0	0	0	1	0	0	0	0	0	0
2015年1月23日	0	0	0	1	0	0	0	0	0	0
2015年1月24日	0	0	0	1	1	0	0	0	0	1
2015年1月25日	0	0	0	1	1	0	0	0	0	0
2015年1月26日	0	0	0	1	0	0	0	0	0	0
2015年1月27日	0	0	0	1	0	0	0	0	0	0
2015年1月28日	0	0	0	1	0	0	0	0	0	0
2015年1月29日	0	0	0	1	0	0	0	0	0	0
2015年1月30日	0	0	0	1	0	0	0	0	0	0
2015年1月31日	0	0	0	1	1	0	0	0	1	0
2015年2月1日	0	0	0	1	1	0	0	0	1	0
2015年2月2日	0	0	0	1	0	0	0	0	1	0
2015年2月3日	0	0	0	1	0	0	0	0	1	0
2015年2月4日	0	0	0	1	0	0	0	0	1	1
2015年2月5日	0	0	0	1	0	0	0	0	1	1
2015年2月6日	0	0	0	1	0	0	0	0	1	1
2015年2月7日	0	0	0	1	1	0	0	0	1	2
2015年2月8日	0	0	0	1	1	0	0	0	1	2
2015年2月9日	0	0	0	1	0	0	0	0	1	2
2015年2月10日	0	0	0	1	0	0	0	0	1	2
2015年2月11日	0	0	0	1	0	0	0	0	1	2
2015年2月12日	0	0	0	1	0	0	0	0	1	2
2015年2月13日	0	0	0	1	0	0	0	0	1	2
2015年2月14日	0	0	0	1	1	0	0	0	1	2
2015年2月15日	0	0	0	1	1	0	0	0	1	2
2015年2月16日	0	0	0	1	0	0	0	0	1	2
2015年2月17日	0	0	0	1	0	0	0	0	1	1
2015年2月18日	0	0	0	1	0	0	0	0	1	0
2015年2月19日	0	0	0	1	0	0	1	0	1	0
2015年2月20日	0	0	0	1	0	0	1	0	1	0

2015年2月21日	0	0	0	1	1	0	1	0	1	0
2015年2月22日	0	0	0	1	1	0	1	0	1	0
2015年2月23日	0	0	0	1	0	0	1	0	1	0
2015年2月24日	0	0	0	1	0	0	1	0	1	0
2015年2月25日	0	0	0	1	0	0	1	0	1	2
2015年2月26日	0	0	0	1	0	0	0	0	1	1
2015年2月27日	0	0	0	1	0	0	0	0	1	1
2015年2月28日	0	0	0	1	1	0	0	0	1	2
2015年3月1日	0	0	0	1	1	0	0	0	0	1
2015年3月2日	0	0	0	1	0	0	0	0	0	1
2015年3月3日	0	0	0	1	0	0	0	0	0	3
2015年3月4日	0	0	0	1	0	0	0	0	0	1
2015年3月5日	0	0	0	1	0	0	0	0	0	1
2015年3月6日	0	0	0	1	0	0	0	0	0	1
2015年3月7日	0	0	0	1	1	0	0	0	0	2
2015年3月8日	0	0	0	1	1	0	0	0	0	3
2015年3月9日	0	0	0	1	0	0	0	0	0	4
2015年3月10日	0	0	0	1	0	0	0	0	0	2
2015年3月11日	0	0	0	1	0	0	0	0	0	2
2015年3月12日	0	0	0	1	0	0	0	0	0	2
2015年3月13日	0	0	0	1	0	0	0	0	0	2
2015年3月14日	0	0	0	1	1	0	0	0	0	3
2015年3月15日	0	0	0	1	1	0	0	0	0	3
2015年3月16日	0	0	0	1	0	0	0	0	0	3
2015年3月17日	0	0	0	1	0	0	0	0	0	3
2015年3月18日	0	0	0	1	0	0	0	0	0	3
2015年3月19日	0	0	0	1	0	0	0	0	0	4
2015年3月20日	0	0	0	1	0	0	0	0	0	7
2015年3月21日	0	0	0	0	1	0	0	0	0	9
2015年3月22日	0	0	0	0	1	0	0	0	0	11
2015年3月23日	0	0	0	0	0	0	0	0	0	25
2015年3月24日	0	0	0	0	0	0	0	0	0	32
2015年3月25日	0	0	0	0	0	0	0	0	0	60
2015年3月26日	0	0	0	0	0	0	0	0	0	41
2015年3月27日	0	0	0	0	0	0	0	0	0	36
2015年3月28日	0	0	0	0	1	0	0	0	0	43
2015年3月29日	0	0	0	0	1	0	0	0	0	48
2015年3月30日	0	0	0	0	0	0	0	0	0	62
2015年3月31日	0	0	0	0	0	0	0	0	0	61
2015年4月1日	0	0	0	0	0	0	0	0	0	45
2015年4月2日	0	0	0	0	0	0	0	0	0	39

2015年4月3日	1	0	0	0	0	0	0	0	0	118
2015年4月4日	1	0	0	0	1	1	0	0	0	159
2015年4月5日	1	0	0	0	1	1	0	0	0	148
2015年4月6日	1	0	0	0	0	1	0	0	0	134
2015年4月7日	1	0	0	0	0	0	0	0	0	160
2015年4月8日	1	0	0	0	0	0	0	0	0	153
2015年4月9日	1	0	0	0	0	0	0	0	0	143
2015年4月10日	1	0	0	0	0	0	0	0	0	138
2015年4月11日	1	0	0	0	1	0	0	0	0	146
2015年4月12日	1	0	0	0	1	0	0	0	0	146
2015年4月13日	1	0	0	0	0	0	0	0	0	166
2015年4月14日	1	0	0	0	0	0	0	0	0	185
2015年4月15日	1	0	0	0	0	0	0	0	0	194
2015年4月16日	1	0	0	0	0	0	0	0	0	191
2015年4月17日	1	0	0	0	0	0	0	0	0	170
2015年4月18日	1	0	0	0	1	0	0	0	0	185
2015年4月19日	1	0	0	0	1	0	0	0	0	134
2015年4月20日	1	0	0	0	0	0	0	0	0	129
2015年4月21日	1	0	0	0	0	0	0	0	0	147
2015年4月22日	1	0	0	0	0	0	0	0	0	157
2015年4月23日	1	0	0	0	0	0	0	0	0	160
2015年4月24日	1	0	0	0	0	0	0	0	0	163
2015年4月25日	1	0	0	0	1	0	0	0	0	171
2015年4月26日	1	0	0	0	1	0	0	0	0	151
2015年4月27日	1	0	0	0	0	0	0	0	0	174
2015年4月28日	1	0	0	0	0	0	0	0	0	186
2015年4月29日	1	0	0	0	0	0	0	0	0	182
2015年4月30日	1	0	0	0	0	0	0	0	0	134
2015年5月1日	1	0	0	0	0	1	0	0	0	177
2015年5月2日	1	0	0	0	1	1	0	0	0	157
2015年5月3日	1	0	0	0	1	1	0	0	0	89
2015年5月4日	1	0	0	0	0	0	0	0	0	115
2015年5月5日	1	0	0	0	0	0	0	0	0	131
2015年5月6日	1	0	0	0	0	0	0	0	0	136
2015年5月7日	1	0	0	0	0	0	0	0	0	143
2015年5月8日	1	0	0	0	0	0	0	0	0	136
2015年5月9日	1	0	0	0	1	0	0	0	0	133
2015年5月10日	1	0	0	0	1	0	0	0	0	100
2015年5月11日	1	0	0	0	0	0	0	0	0	147
2015年5月12日	1	0	0	0	0	0	0	0	0	156
2015年5月13日	1	0	0	0	0	0	0	0	0	172

2015 年 5 月 14 日	1	0	0	0	0	0	0	0	0	188
2015 年 5 月 15 日	1	0	0	0	0	0	0	0	0	182
2015 年 5 月 16 日	1	0	0	0	1	0	0	0	0	179
2015 年 5 月 17 日	1	0	0	0	1	0	0	0	0	158
2015 年 5 月 18 日	1	0	0	0	0	0	0	0	0	146
2015 年 5 月 19 日	1	0	0	0	0	0	0	0	0	156
2015 年 5 月 20 日	1	0	0	0	0	0	0	0	0	151
2015 年 5 月 21 日	1	0	0	0	0	0	0	0	0	142
2015 年 5 月 22 日	1	0	0	0	0	0	0	0	0	110
2015 年 5 月 23 日	1	0	0	0	1	0	0	0	0	169
2015 年 5 月 24 日	1	0	0	0	1	0	0	0	0	164
2015 年 5 月 25 日	1	0	0	0	0	0	0	0	0	178
2015 年 5 月 26 日	1	0	0	0	0	0	0	0	0	188
2015 年 5 月 27 日	1	0	0	0	0	0	0	0	0	205
2015 年 5 月 28 日	1	0	0	0	0	0	0	0	0	139
2015 年 5 月 29 日	1	0	0	0	0	0	0	0	0	140
2015 年 5 月 30 日	1	0	0	0	1	0	0	0	0	144
2015 年 5 月 31 日	1	0	0	0	1	0	0	0	0	129
2015 年 6 月 1 日	1	0	0	0	0	0	0	0	0	160
2015 年 6 月 2 日	1	0	0	0	0	0	0	0	0	196
2015 年 6 月 3 日	1	0	0	0	0	0	0	0	0	169
2015 年 6 月 4 日	1	0	0	0	0	0	0	0	0	176
2015 年 6 月 5 日	1	0	0	0	0	0	0	0	0	171
2015 年 6 月 6 日	1	0	0	0	1	0	0	0	0	162
2015 年 6 月 7 日	1	0	0	0	1	0	0	0	0	121
2015 年 6 月 8 日	1	0	0	0	0	0	0	0	0	131
2015 年 6 月 9 日	1	0	0	0	0	0	0	0	0	160
2015 年 6 月 10 日	1	0	0	0	0	0	0	0	0	163
2015 年 6 月 11 日	1	0	0	0	0	0	0	0	0	152
2015 年 6 月 12 日	1	0	0	0	0	0	0	0	0	158
2015 年 6 月 13 日	1	0	0	0	1	0	0	0	0	149
2015 年 6 月 14 日	1	0	0	0	1	0	0	0	0	180
2015 年 6 月 15 日	1	0	0	0	0	0	0	0	0	208
2015 年 6 月 16 日	1	0	0	0	0	0	0	0	0	220
2015 年 6 月 17 日	1	0	0	0	0	0	0	0	0	227
2015 年 6 月 18 日	1	0	0	0	0	0	0	0	0	189
2015 年 6 月 19 日	1	0	0	0	0	0	0	0	0	159
2015 年 6 月 20 日	1	0	0	0	1	1	0	0	0	174
2015 年 6 月 21 日	1	0	0	0	1	1	0	0	0	142
2015 年 6 月 22 日	1	1	0	0	0	1	0	0	0	113
2015 年 6 月 23 日	1	1	0	0	0	0	0	0	0	222

2015年6月24日	1	1	0	0	0	0	0	0	0	208
2015年6月25日	1	1	0	0	0	0	0	0	0	197
2015年6月26日	1	1	0	0	0	0	0	0	0	135
2015年6月27日	1	1	0	0	1	0	0	0	0	139
2015年6月28日	1	1	0	0	1	0	0	0	0	111
2015年6月29日	1	1	0	0	0	0	0	0	0	153
2015年6月30日	1	1	0	0	0	0	0	0	0	155
2015年7月1日	1	1	0	0	0	0	0	0	0	152
2015年7月2日	1	1	0	0	0	0	0	0	0	166
2015年7月3日	1	1	0	0	0	0	0	0	0	142
2015年7月4日	1	1	0	0	1	0	0	0	0	153
2015年7月5日	1	1	0	0	1	0	0	0	0	86
2015年7月6日	1	1	0	0	0	0	0	0	0	126
2015年7月7日	1	1	0	0	0	0	0	0	0	152
2015年7月8日	1	1	0	0	0	0	0	0	0	153
2015年7月9日	1	1	0	0	0	0	0	0	0	155
2015年7月10日	1	1	0	0	0	0	0	0	0	155
2015年7月11日	1	1	0	0	1	0	0	0	0	178
2015年7月12日	1	1	0	0	1	0	0	1	0	155
2015年7月13日	1	1	0	0	0	0	0	1	0	141
2015年7月14日	1	1	0	0	0	0	0	1	0	166
2015年7月15日	1	1	0	0	0	0	0	1	0	150
2015年7月16日	1	1	0	0	0	0	0	1	0	151
2015年7月17日	1	1	0	0	0	0	0	1	0	159
2015年7月18日	1	1	0	0	1	0	0	1	0	144
2015年7月19日	1	1	0	0	1	0	0	1	0	141
2015年7月20日	1	1	0	0	0	0	0	1	0	150
2015年7月21日	1	1	0	0	0	0	0	1	0	171
2015年7月22日	1	1	0	0	0	0	0	1	0	153
2015年7月23日	1	1	0	0	0	0	0	1	0	159
2015年7月24日	1	1	0	0	0	0	0	1	0	154
2015年7月25日	1	1	0	0	1	0	0	1	0	144
2015年7月26日	1	1	0	0	1	0	0	1	0	131
2015年7月27日	1	1	0	0	0	0	0	1	0	148
2015年7月28日	1	1	0	0	0	0	0	1	0	144
2015年7月29日	1	1	0	0	0	0	0	1	0	160
2015年7月30日	1	1	0	0	0	0	0	1	0	172
2015年7月31日	1	1	0	0	0	0	0	1	0	155
2015年8月1日	1	1	0	0	1	0	0	1	0	158
2015年8月2日	1	1	0	0	1	0	0	1	0	124
2015年8月3日	1	1	0	0	0	0	0	1	0	121

2015年8月4日	1	1	0	0	0	0	0	1	0	166
2015年8月5日	1	1	0	0	0	0	0	1	0	155
2015年8月6日	1	1	0	0	0	0	0	1	0	169
2015年8月7日	1	1	0	0	0	0	0	1	0	158
2015年8月8日	1	1	0	0	1	0	0	1	0	165
2015年8月9日	1	1	0	0	1	0	0	1	0	118
2015年8月10日	1	1	0	0	0	0	0	1	0	138
2015年8月11日	1	1	0	0	0	0	0	1	0	167
2015年8月12日	1	1	0	0	0	0	0	1	0	166
2015年8月13日	1	1	0	0	0	0	0	1	0	153
2015年8月14日	1	1	0	0	0	0	0	1	0	157
2015年8月15日	1	1	0	0	1	0	0	1	0	165
2015年8月16日	1	1	0	0	1	0	0	1	0	111
2015年8月17日	1	1	0	0	0	0	0	1	0	147
2015年8月18日	1	1	0	0	0	0	0	1	0	152
2015年8月19日	1	1	0	0	0	0	0	1	0	169
2015年8月20日	1	1	0	0	0	0	0	1	0	165
2015年8月21日	1	1	0	0	0	0	0	1	0	150
2015年8月22日	1	1	0	0	1	0	0	1	0	153
2015年8月23日	1	1	0	0	1	0	0	1	0	122
2015年8月24日	1	1	0	0	0	0	0	1	0	172
2015年8月25日	1	1	0	0	0	0	0	1	0	169
2015年8月26日	1	1	0	0	0	0	0	1	0	189
2015年8月27日	1	1	0	0	0	0	0	1	0	159
2015年8月28日	1	1	0	0	0	0	0	1	0	146
2015年8月29日	1	1	0	0	1	0	0	1	0	125
2015年8月30日	1	1	0	0	1	0	0	1	0	149
2015年8月31日	1	1	0	0	0	0	0	1	0	167
2015年9月1日	1	1	0	0	0	0	0	0	0	193
2015年9月2日	1	1	0	0	0	0	0	0	0	175
2015年9月3日	1	1	0	0	0	0	0	0	0	180
2015年9月4日	1	1	0	0	0	0	0	0	0	169
2015年9月5日	1	1	0	0	1	0	0	0	0	124
2015年9月6日	1	1	0	0	1	0	0	0	0	107
2015年9月7日	1	1	0	0	0	0	0	0	0	182
2015年9月8日	1	1	0	0	0	0	0	0	0	169
2015年9月9日	1	1	0	0	0	0	0	0	0	179
2015年9月10日	1	1	0	0	0	0	0	0	0	208
2015年9月11日	1	1	0	0	0	0	0	0	0	208
2015年9月12日	1	1	0	0	1	0	0	0	0	177
2015年9月13日	1	1	0	0	1	0	0	0	0	98

2015 年 9 月 14 日	1	1	0	0	0	0	0	0	0	133
2015 年 9 月 15 日	1	1	0	0	0	0	0	0	0	156
2015 年 9 月 16 日	1	1	0	0	0	0	0	0	0	153
2015 年 9 月 17 日	1	1	0	0	0	0	0	0	0	161
2015 年 9 月 18 日	1	1	0	0	0	0	0	0	0	152
2015 年 9 月 19 日	1	1	0	0	1	0	0	0	0	154
2015 年 9 月 20 日	1	1	0	0	1	0	0	0	0	136
2015 年 9 月 21 日	1	1	0	0	0	0	0	0	0	234
2015 年 9 月 22 日	1	1	0	0	0	0	0	0	0	146
2015 年 9 月 23 日	1	1	0	0	0	0	0	0	0	179
2015 年 9 月 24 日	1	0	1	0	0	0	0	0	0	178
2015 年 9 月 25 日	1	0	1	0	0	0	0	0	0	160
2015 年 9 月 26 日	1	0	1	0	1	1	0	0	0	149
2015 年 9 月 27 日	1	0	1	0	1	1	0	0	0	98
2015 年 9 月 28 日	1	0	1	0	0	1	0	0	0	127
2015 年 9 月 29 日	1	0	1	0	0	0	0	0	0	155
2015 年 9 月 30 日	1	0	1	0	0	0	0	0	0	147
2015 年 10 月 1 日	1	0	1	0	0	0	1	0	0	176
2015 年 10 月 2 日	1	0	1	0	0	0	1	0	0	189
2015 年 10 月 3 日	1	0	1	0	1	0	1	0	0	181
2015 年 10 月 4 日	1	0	1	0	1	0	1	0	0	186
2015 年 10 月 5 日	1	0	1	0	0	0	1	0	0	167
2015 年 10 月 6 日	1	0	1	0	0	0	1	0	0	147
2015 年 10 月 7 日	1	0	1	0	0	0	1	0	0	84
2015 年 10 月 8 日	1	0	1	0	0	0	0	0	0	112
2015 年 10 月 9 日	1	0	1	0	0	0	0	0	0	103
2015 年 10 月 10 日	1	0	1	0	1	0	0	0	0	129
2015 年 10 月 11 日	1	0	1	0	1	0	0	0	0	130
2015 年 10 月 12 日	1	0	1	0	0	0	0	0	0	174
2015 年 10 月 13 日	1	0	1	0	0	0	0	0	0	179
2015 年 10 月 14 日	1	0	1	0	0	0	0	0	0	156
2015 年 10 月 15 日	1	0	1	0	0	0	0	0	0	162
2015 年 10 月 16 日	1	0	1	0	0	0	0	0	0	166
2015 年 10 月 17 日	1	0	1	0	1	0	0	0	0	168
2015 年 10 月 18 日	1	0	1	0	1	0	0	0	0	142
2015 年 10 月 19 日	1	0	1	0	0	0	0	0	0	164
2015 年 10 月 20 日	1	0	1	0	0	0	0	0	0	174
2015 年 10 月 21 日	1	0	1	0	0	0	0	0	0	160
2015 年 10 月 22 日	1	0	1	0	0	0	0	0	0	162
2015 年 10 月 23 日	1	0	1	0	0	0	0	0	0	141
2015 年 10 月 24 日	1	0	1	0	1	0	0	0	0	165

2015年10月25日	1	0	1	0	1	0	0	0	0	135
2015年10月26日	1	0	1	0	0	0	0	0	0	179
2015年10月27日	1	0	1	0	0	0	0	0	0	179
2015年10月28日	1	0	1	0	0	0	0	0	0	173
2015年10月29日	1	0	1	0	0	0	0	0	0	162
2015年10月30日	1	0	1	0	0	0	0	0	0	157
2015年10月31日	1	0	1	0	1	0	0	0	0	142
2015年11月1日	1	0	1	0	1	0	0	0	0	150
2015年11月2日	1	0	1	0	0	0	0	0	0	148
2015年11月3日	1	0	1	0	0	0	0	0	0	163
2015年11月4日	1	0	1	0	0	0	0	0	0	161
2015年11月5日	1	0	1	0	0	0	0	0	0	164
2015年11月6日	1	0	1	0	0	0	0	0	0	143
2015年11月7日	1	0	1	0	1	0	0	0	0	157
2015年11月8日	1	0	1	0	1	0	0	0	0	113
2015年11月9日	1	0	1	0	0	0	0	0	0	163
2015年11月10日	1	0	1	0	0	0	0	0	0	209
2015年11月11日	1	0	1	0	0	0	0	0	0	209
2015年11月12日	1	0	1	0	0	0	0	0	0	236
2015年11月13日	1	0	1	0	0	0	0	0	0	189
2015年11月14日	1	0	1	0	1	0	0	0	0	175
2015年11月15日	1	0	1	0	1	0	0	0	0	108
2015年11月16日	1	0	1	0	0	0	0	0	0	116
2015年11月17日	1	0	1	0	0	0	0	0	0	196
2015年11月18日	1	0	1	0	0	0	0	0	0	196
2015年11月19日	1	0	1	0	0	0	0	0	0	207
2015年11月20日	1	0	1	0	0	0	0	0	0	162
2015年11月21日	1	0	1	0	1	0	0	0	0	147
2015年11月22日	1	0	1	0	1	0	0	0	0	100
2015年11月23日	1	0	1	0	0	0	0	0	0	129
2015年11月24日	1	0	1	0	0	0	0	0	0	161
2015年11月25日	1	0	1	0	0	0	0	0	0	160
2015年11月26日	1	0	1	0	0	0	0	0	0	165
2015年11月27日	1	0	1	0	0	0	0	0	0	130
2015年11月28日	1	0	1	0	1	0	0	0	0	142
2015年11月29日	1	0	1	0	1	0	0	0	0	131
2015年11月30日	1	0	1	0	0	0	0	0	0	166
2015年12月1日	1	0	1	0	0	0	0	0	0	155
2015年12月2日	1	0	1	0	0	0	0	0	0	156
2015年12月3日	1	0	1	0	0	0	0	0	0	179
2015年12月4日	1	0	1	0	0	0	0	0	0	176

2015 年 12 月 5 日	1	0	1	0	1	0	0	0	0	153
2015 年 12 月 6 日	1	0	1	0	1	0	0	0	0	105
2015 年 12 月 7 日	1	0	1	0	0	0	0	0	0	117
2015 年 12 月 8 日	1	0	1	0	0	0	0	0	0	116
2015 年 12 月 9 日	1	0	1	0	0	0	0	0	0	131
2015 年 12 月 10 日	1	0	1	0	0	0	0	0	0	174
2015 年 12 月 11 日	1	0	1	0	0	0	0	0	0	184
2015 年 12 月 12 日	1	0	1	0	1	0	0	0	0	173
2015 年 12 月 13 日	1	0	1	0	1	0	0	0	0	119
2015 年 12 月 14 日	1	0	1	0	0	0	0	0	0	107
2015 年 12 月 15 日	1	0	1	0	0	0	0	0	0	142
2015 年 12 月 16 日	1	0	1	0	0	0	0	0	0	117
2015 年 12 月 17 日	1	0	1	0	0	0	0	0	0	131
2015 年 12 月 18 日	1	0	1	0	0	0	0	0	0	137
2015 年 12 月 19 日	1	0	1	0	1	0	0	0	0	139
2015 年 12 月 20 日	1	0	1	0	1	0	0	0	0	97
2015 年 12 月 21 日	1	0	1	0	0	0	0	0	0	117
2015 年 12 月 22 日	1	0	1	0	0	0	0	0	0	121
2015 年 12 月 23 日	1	0	0	1	0	0	0	0	0	125
2015 年 12 月 24 日	1	0	0	1	0	0	0	0	0	144
2015 年 12 月 25 日	1	0	0	1	0	0	0	0	0	151
2015 年 12 月 26 日	1	0	0	1	1	0	0	0	0	145
2015 年 12 月 27 日	1	0	0	1	1	0	0	0	0	142
2015 年 12 月 28 日	1	0	0	1	0	0	0	0	0	146
2015 年 12 月 29 日	1	0	0	1	0	0	0	0	0	129
2015 年 12 月 30 日	1	0	0	1	0	0	0	0	0	144
2015 年 12 月 31 日	1	0	0	1	0	0	0	0	0	150

表 2：提前 n 天预定工作日入住的概率

提前预定天数	概率
0	0.51094363
1	0.132304357
2	0.057350816
3	0.042845344
4	0.03572166
5	0.025087756
6	0.017757588
7	0.016002478
8	0.01326657
9	0.010324179
10	0.011563081

11	0.008723931
12	0.007175305
13	0.007588272
14	0.005110469
15	0.004542639
16	0.003355358
17	0.004336155
18	0.005988024
19	0.003613463
20	0.003871567
21	0.004336155
22	0.00232294
23	0.001600248
24	0.003716705
25	0.003768325
26	0.002116457
27	0.002168078
28	0.003510221
29	0.002529424
30	0.000877555
31	0.001238902
32	0.002374561
33	0.002168078
34	0.001961594
35	0.002581045
36	0.003561842
37	0.001909973
38	0.002477803
39	0.001290522
总和	0.973983068

表 3：提前 m 天预定周末入住发生的概率

提前预定天数	概率
0	0.50925345
1	0.131587202
2	0.060382685
3	0.04140527
4	0.017565872
5	0.023525721
6	0.028544542
7	0.021173149
8	0.020388959

9	0.018036386
10	0.008626098
11	0.003920954
12	0.006430364
13	0.007841907
14	0.005959849
15	0.007528231
16	0.004861982
17	0.003293601
18	0.002666248
19	0.003764115
20	0.003293601
21	0.001568381
22	0.002979925
23	0.003764115
24	0.001254705
25	0.001568381
26	0.001882058
27	0.005959849
28	0.002195734
29	0.002979925
30	0.002038896
31	0.001097867
32	0.000784191
33	0.00172522
34	0.00172522
35	0.000941029
36	0.000313676
37	0.000784191
38	0.000470514
39	0.000156838
总和	0.964240903

表 4：2016 年 1 月 1 日-5 月 10 日 8 个虚拟变量的值与预测入住房间数

日期	竞争对手是否倒闭	夏天	秋天	冬天	周末	3 天假期	7 天假期	暑假	寒假	入住房间数
2016 年 1 月 1 日	1	0	0	1	0	1	0	0	0	123
2016 年 1 月 2 日	1	0	0	1	1	1	0	0	0	110
2016 年 1 月 3 日	1	0	0	1	1	1	0	0	0	110
2016 年 1 月 4 日	1	0	0	1	0	0	0	0	0	135

2016 年 1 月 5 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 6 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 7 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 8 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 9 日	1	0	0	1	1	0	0	0	0	122
2016 年 1 月 10 日	1	0	0	1	1	0	0	0	0	122
2016 年 1 月 11 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 12 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 13 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 14 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 15 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 16 日	1	0	0	1	1	0	0	0	0	122
2016 年 1 月 17 日	1	0	0	1	1	0	0	0	0	122
2016 年 1 月 18 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 19 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 20 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 21 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 22 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 23 日	1	0	0	1	1	0	0	0	0	122
2016 年 1 月 24 日	1	0	0	1	1	0	0	0	0	122
2016 年 1 月 25 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 26 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 27 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 28 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 29 日	1	0	0	1	0	0	0	0	0	135
2016 年 1 月 30 日	1	0	0	1	1	0	0	0	0	122
2016 年 1 月 31 日	1	0	0	1	1	0	0	0	0	122
2016 年 2 月 1 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 2 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 3 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 4 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 5 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 6 日	1	0	0	1	1	0	0	0	1	119
2016 年 2 月 7 日	1	0	0	1	1	0	0	0	1	119
2016 年 2 月 8 日	1	0	0	1	0	0	1	0	1	136
2016 年 2 月 9 日	1	0	0	1	0	0	1	0	1	136
2016 年 2 月 10 日	1	0	0	1	0	0	1	0	1	136
2016 年 2 月 11 日	1	0	0	1	0	0	1	0	1	136
2016 年 2 月 12 日	1	0	0	1	0	0	1	0	1	136
2016 年 2 月 13 日	1	0	0	1	1	0	1	0	1	123
2016 年 2 月 14 日	1	0	0	1	1	0	1	0	1	123

2016 年 2 月 15 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 16 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 17 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 18 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 19 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 20 日	1	0	0	1	1	0	0	0	1	119
2016 年 2 月 21 日	1	0	0	1	1	0	0	0	1	119
2016 年 2 月 22 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 23 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 24 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 25 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 26 日	1	0	0	1	0	0	0	0	1	132
2016 年 2 月 27 日	1	0	0	1	1	0	0	0	1	119
2016 年 2 月 28 日	1	0	0	1	1	0	0	0	1	119
2016 年 2 月 29 日	1	0	0	1	0	0	0	0	0	135
2016 年 3 月 1 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 2 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 3 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 4 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 5 日	1	0	0	0	1	0	0	0	0	151
2016 年 3 月 6 日	1	0	0	0	1	0	0	0	0	151
2016 年 3 月 7 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 8 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 9 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 10 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 11 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 12 日	1	0	0	0	1	0	0	0	0	151
2016 年 3 月 13 日	1	0	0	0	1	0	0	0	0	151
2016 年 3 月 14 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 15 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 16 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 17 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 18 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 19 日	1	0	0	0	1	0	0	0	0	151
2016 年 3 月 20 日	1	0	0	0	1	0	0	0	0	151
2016 年 3 月 21 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 22 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 23 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 24 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 25 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 26 日	1	0	0	0	1	0	0	0	0	151

2016 年 3 月 27 日	1	0	0	0	1	0	0	0	0	151
2016 年 3 月 28 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 29 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 30 日	1	0	0	0	0	0	0	0	0	164
2016 年 3 月 31 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 1 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 2 日	1	0	0	0	1	1	0	0	0	139
2016 年 4 月 3 日	1	0	0	0	1	1	0	0	0	139
2016 年 4 月 4 日	1	0	0	0	0	1	0	0	0	152
2016 年 4 月 5 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 6 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 7 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 8 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 9 日	1	0	0	0	1	0	0	0	0	151
2016 年 4 月 10 日	1	0	0	0	1	0	0	0	0	151
2016 年 4 月 11 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 12 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 13 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 14 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 15 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 16 日	1	0	0	0	1	0	0	0	0	151
2016 年 4 月 17 日	1	0	0	0	1	0	0	0	0	151
2016 年 4 月 18 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 19 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 20 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 21 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 22 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 23 日	1	0	0	0	1	0	0	0	0	151
2016 年 4 月 24 日	1	0	0	0	1	0	0	0	0	151
2016 年 4 月 25 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 26 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 27 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 28 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 29 日	1	0	0	0	0	0	0	0	0	164
2016 年 4 月 30 日	1	0	0	0	1	0	0	0	0	151
2016 年 5 月 1 日	1	0	0	0	1	1	0	0	0	139
2016 年 5 月 2 日	1	0	0	0	0	1	0	0	0	152
2016 年 5 月 3 日	1	0	0	0	0	1	0	0	0	152
2016 年 5 月 4 日	1	0	0	0	0	0	0	0	0	164
2016 年 5 月 5 日	1	0	0	0	0	0	0	0	0	164
2016 年 5 月 6 日	1	0	0	0	0	0	0	0	0	164

2016年5月7日	1	0	0	0	1	0	0	0	0	151
2016年5月8日	1	0	0	0	1	0	0	0	0	151
2016年5月9日	1	0	0	0	0	0	0	0	0	164
2016年5月10日	1	0	0	0	0	0	0	0	0	164