出租车机场载客决策方案

摘 要

机场“蓄车池”出租车排队等待乘客乘车是一个复杂的排队系统，本文基于收益优化决策模型、排队论、相关分析，综合出租车排队载客系统的出租车、乘客、机场管理者三方建立出租车排队决策模型，根据出租车收益、等待时间、通行效率等指标可对出租车是否进入“蓄车池”、如何制定“上车点”上车方案、如何给与优先权做出决策。最后结合所搜收集的数据应用 Arena 软件进行仿真模拟，给出科学可行的决策方案。

针对问题一，首先确定影响出租车司机作出决定的影响因素，如航班数量、乘客数量变化、正在等待出租车数量等。通过收益成本函数来表达相关因素与出租车司机最终决策的定量关系，得到出租车司机可以接受的最长等待队伍，并以收益成本函数的正负性来为出租车司机作出决策。

针对问题二，数据来源为上海市某天 4316 辆出租车的 GPS 记录，处理后约为 40 万条行程。首先进行描述性统计，初步验证模型的合理性，后经计算得到 74%司机选择的支持，最后检验出租车司机选择方案与航班数量即乘客到达数量的强相关性。

针对问题三，为结合系统中出租车排队等待载客和乘客等待上车两个排队情景，本文在 M/M/c 排队模型的基础上构建多窗口双向等待制排队模型，并得到出租车平均等待时间、平均等待车数、乘客平均等待时间、平均等待人数、“上车点”泊位平均使用率、通行效率 6 个指标的计算公式。结合所收集数据，利用 Arena 软件进行仿真模拟，通过比较不同泊位数下上述指标的表现确定出最优泊位数，从而给出“上车点”设计方案。

针对问题四，首先明确管理部门对短途载客再次返回的出租车给予优先权的程度，在于找到判断从机场载客的里程数为短途或长途的界定点。再根据带有优先权的排队模型，计算出两个优先级（长途、短途出租车）的平均等待时间。由收益均衡的条件，可得到长途、短途出租车里程数期望差与二者平均等待时间差的关系。利用 Arena 软件仿真模拟得到整个排队系统通行效率最大时二者平均等待时间，从而求解出长途、短途出租车里程数期望差，再根据条件期望公式，可求出判定短途、长途的里程数界定点，得出了可行的优先安排方案。

关键词：决策、排队、数据挖掘、收益、仿真模拟

一、 问题重述

1.1 背景

出租车载客到达机场后，面临着到“蓄车池”等待载客和空载回市区两个选择。选择到“蓄车池”等待载客的司机需要排队等待载客，空载回市区则会损失空车费用以及潜在的载客收益。出租车司机需要根据到达航班数、“蓄车池”排队车辆数等相关信息及经验进行决策。同时到“蓄车池”乘车的乘客也面临这排队乘车的问题，合理的“上车点”上车规则可以降低出租车和乘客的等待时间，同时保证“上车点”的通行效率。由于乘客所要到达的目的地由远有近，出租车接单的收益存在较大差异，但出租车不能选择乘客或拒载。为平衡出租车的收益，可以给短途载客再次返回的出租车给予一定的“优先权”。

1.2 需要解决的问题

问题一

确定出租车司机的决策模型，综合考虑航班数量、乘客数量变化、已等待出租车的数量等因素对出租车司机进行决策（选择等待载客或直接返回市区）的影响，确定出租车司机决策收益成本函数，帮助出租车司机进行判断。

问题二

收集国内某一机场及其所在城市出租车的相关数据，需要对数据进行清洗、整理、分析，得到和本题相关的指标数据，如航班数量、从机场出发的载客时间、在机场等待的时间等。定量分析模型一中的相关因素对实际情况中出租车司机最终决策的依赖性，并进一步说明验证模型的合理性。

问题三

在机场“蓄车池”，往往存在出租车排队载客和乘客排队等车的情况，在保证出租车和乘客安全的前提下，制定一个“上车点”设计方案，使得“蓄车池”的乘车效率最优。“上车点”泊位数量是研究重点，确定一个合理的泊位数量既能很好的服务离港乘客又能减少待客出租车的蓄车时间，提高系统的通行效率。

问题四

确定管理部门对短途载客再次返回的出租车给予优先权的程度，在于找到判断从机场载客的路程是否为短途或长途的界定点。通过控制界定点的大小来控制短途出租车进入优先车道的速率，节省短途出租车的等待时间以弥补载客收益，来保证长途、短途出租车的收益达到均衡。

二、 问题分析

2.1 问题一的分析

利用影响出租车司机决策的相关因素得到出租车司机收益成本函数，即用出租车司机在机场等待载客的收益减去沉没成本和等待成本，比较差值的正负，为出租车司机提出选择建议。当差值为正时表示出租车在机场等待载客的收益大于直接返回市区的收益，建议出租车司机在机场等待；当差值为负时表示出租车在机场等待载客的收益小于直接返回市区的收益，建议出租车司机返回市区。

2.2 问题二的分析

通过清洗整理后的数据带回到问题一的模型中，为出租车司机提供决策，考虑到大多数出租车司机有一定的经验，能够在实际情况中作出正确的判断，如果得到大部分留在机场等待载客的出租车司机的收益成本函数为正，则说明符合生活实际，验证了模型的合理性。同时将模型一预判的影响因素的实际数据与出租车司机的最终决策进行相关分析，得到具体的相关系数，说明相关因素的依赖性。

2.3 问题三的分析

在“上车点”的设计中，“上车点”的泊位数量研究的重点。综合考虑“上车点”受到建筑空间和道路的限制、出租车和乘客的等待时间以及“蓄车去”的通行效率，确定一个合理的泊位数量既能很好的服务离港乘客又能减少待客出租车的蓄车时间，提高系统的通行效率。问题三情形可看成出租车和乘客多窗口双向等待制排队系统，通过构建合理的排队系统可以求出平均等待时间、平均通行能力等指标。通过建立多窗口双向等待制排队模型，综合考虑平均等待人数、平均等待车数、乘客平均等待时间、出租车平均等待时间、平均绝对通行能力等指标，确定最优窗口数。最后结合数据进行仿真模拟，给出即“上车点”设计方案。

2.4 问题四的分析

建立带有优先权的排队模型，求解得到第一优先级（短途出租车）和第二优先级（长途出租车）平均等待时间。为了让出租车收益均衡，即让长途出租车载客收益与等待时间成本差值与短途出租车保持一致，可以得到长、短途出租车载客里程期望差与二者平均等待时间之差的关系，再利用软件 Arena 仿真模拟开放优先车道的情况，求出整个系统通行效率最大时长、短途出租车载客里程期望差，利用条件期望公式求解出长、短途的界定点。

三、 模型假设

（1） 出租车选择一种决策后不能改变。

（2） 每组乘客上车用时为确定的常数。

（3） 机场客流量较大，到达航班数、天气等因素对客流量和车流量的影响可忽略，客流量和车流量是平稳的。

（4） 在充分小的时间间隔内只有一组乘客到达系统，两组乘客同时到达系统的概率为 0；在充分小的时间间隔内只有一辆出租车到达系统，两辆及两辆以上出租车到达系统的概率为 0。

（5） 在互不相交的时间内，乘客到达“蓄车池”是独立同分布的随机事件，与已到达多少乘客无关；出租车到达“蓄车池”同理。

（6） 乘客和出租车进入等待区后按先后顺序排队等待，不能离开。

（7） 系统容量、乘客源及出租车源无限。

（8） 存在两个优先级（1 级代表短途优先级；2 级代表长途优先级）。

（9） 服务顺序首先基于优先级，同一优先级内，依据“先到先服务”原则。

（10）对任意优先级，出租车到达服从 Poisson 分布，服务时间服从负指数分布。

（11）具有优先权的短途出租车，也不能强制让一个正在提供服务的长途出租车返回排队。

四、 符号说明

符号	说明
$\mathbf{W}_{c}$	在排队系统中出租车司机预期等待的时间
$\mathbf{N}_{\mathbf{c}}$	在排队系统中的等待载客的出租车数量
$\frac{1}{\mu}$	在排队系统中每组乘客上车用时
s	在排队系统中“上车点”内的泊位数量

五、 模型的建立与求解

5.1 问题一的求解

5.1.1 出租车司机决策行为模型的构建

根据题目要求，选择收益成本函数来比较出租车司机两种决策，收益成本函数如下

$$\mathrm{U}(\mathbf{W_c}) = \mathrm{R} + C_{sunk} - \mathrm{C} * \mathbf{W_c} \quad (1)$$

其中，$\mathrm{U}(\mathbf{W_c})$表示在排队系统中出租车司机的收益成本函数，R表示出租车司机在排队系统中载客获得的期望收益，C 表示出租车司机在排队系统中单位时间的等候成本（单位时间内回市区的期望收益），C_{sunk}表示出租车司机在排队系统中的沉没成本（决策已经发生，不能由将来的任何决策而改变的成本，如心理成本、油费差价等）。

基于收益成本函数的模型认为出租车司机仅在$\mathrm{U}(\mathbf{W_c}) \geq 0$的情况下进入“蓄车池”排队等待，否则直接放空返回市区拉客。令T_{max}表示出租车司机可以接受的最长等待时间，此时收益成本函数$\mathrm{U}(\mathbf{W_c}) = 0$，从而得到

$$\mathrm{T}_{max} = \frac{R+C_{sunk}}{\mathrm{C}} \quad (2)$$

5.1.2 出租车司机决策行为模型的求解

首先设N_p表示出租车前面无等待载客的其他出租车时乘客数量（以组为单位），N_a表示在排队系统中抵达的航班数量。可将出租车司机预期等待时间分为两种情况：

第一种情况：$\mathrm{N}_p > 0$

$$\mathbf{W_c} = \frac{\mathrm{N}_c+1}{\mu * \mathrm{N}_a} \quad (3)$$

带入式（2）到式（3），则出租车司机可接受的最长等待车队长为

$$\mathrm{N}_q = \mu * \mathrm{N}_a * \mathrm{T}_{max} - 1 \quad (4)$$

带入式（2）到式（4）

$$\mathrm{N}_q = \mu * \mathrm{N}_a * (\frac{R+C_{sunk}}{\mathrm{C}}) - 1 \quad (5)$$

第二种情况：$\mathrm{N}_p = 0$

此时，出租车在等待其前面的出租车载客完成后没有乘客，需继续等待下一组乘客的到来，设这段时间为t_0，则

$$\mathbf{W_c} = \frac{\mathrm{N}_c+1}{\mu * \mathrm{N}_a} + t_0 \quad (6)$$

带入式（2）到式（6），则出租车司机可接受的最长等待车队长为

$$\mathrm{N}_q = \mu * \mathrm{N}_a * (\mathrm{T}_{max} - t_0) - 1 \quad (7)$$

带入式（2）到式（7）得到

$$\mathrm{N}_q = \mu * \mathrm{N}_a * (\frac{R+C_{sunk}}{\mathrm{C}} - t_0) - 1 \quad (8)$$

由于泊松分布的无记忆性，设服从泊松分布的参数为λ，λ表示为单位时间内乘客到达的平均数量，则下一组乘客到来的时间可近似为

$$t_0 = \frac{1}{\lambda} \tag{9}$$

带入式（9）到式（8）得到

$$N_q = \mu * N_a * (\frac{R+C_{sunk}}{C} - \frac{1}{\lambda}) - 1 \tag{10}$$

所以出租车司机的决策如下：

当出租车司机到达时车队中排队的出租车数量$N_c \leq N_q$时，选择进入“蓄车池”排队；当出租车司机到达时车队中排队的出租车数量$N_c > N_q$时，选择直接放空返回市区拉客。

5.2 问题二的求解

5.2.1 数据收集与处理

5.2.1.1 数据简介

选自上海市2007年2月20日的出租车GPS记录，特征包括：出租车ID、时间、经度、纬度、速度、角度、载客状态（‘0’为无客载，‘1’为有客载），共4316辆出租车，每隔1min左右更新一次车的GPS记录，记录时长为24h,每辆出租车的GPS轨迹数据量多为千条左右，数据量大概四百万左右。

5.2.1.2 数据处理

首先对数据进行清洗，将经度、纬度、行驶状态（速度是否大于0）、载客状态未发生变化的数据删除，整理出每辆出租车出发-到达的时刻表，将数据量减少了一个量级。

通过经纬度计算公式，筛选出出租车在机场等待、出发、载下一单客的数据集以及出租车全天在市区拉客的数据集。

5.2.1.3 数据检查

抽取一部分从机场出发的行程并标记出它们的目的地，做出如下所示的热力分布图，可以看到，对数据的整理与清洗基本正确，结果与经验相符。选取上海虹桥国际机场的原因在于其离市区较近，数据集中的符合分析条件的数据项相较离市区很远的浦东机场要充足，且面临决策时司机的选择更为灵活，更有参考与研究价值。

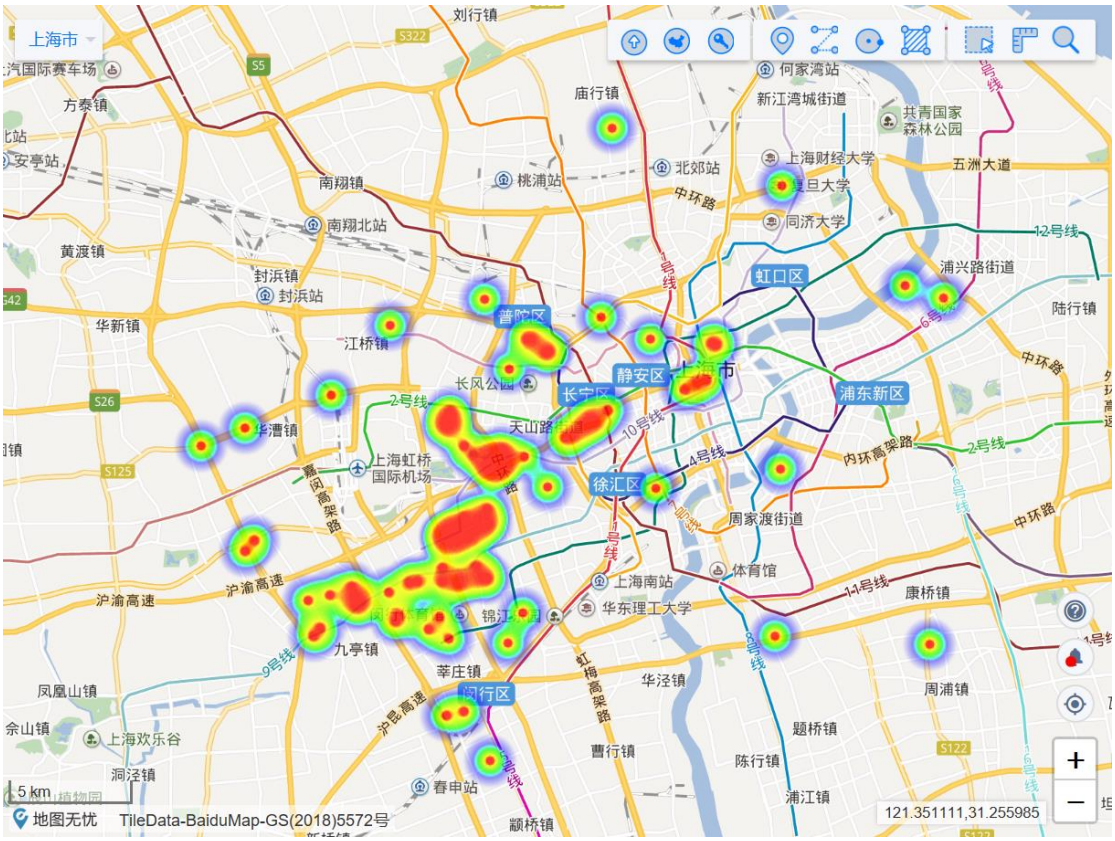


图 5.2.1.3-1 机场出发行程目的地的热力分布图

从机场出发的行程的时长、里程分布情况与市区内的行程时长、里程分布情况均与预期相符。

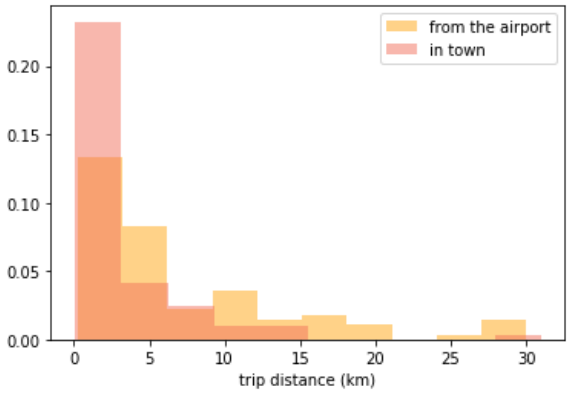


图 5.2.1.3-2 距离分布

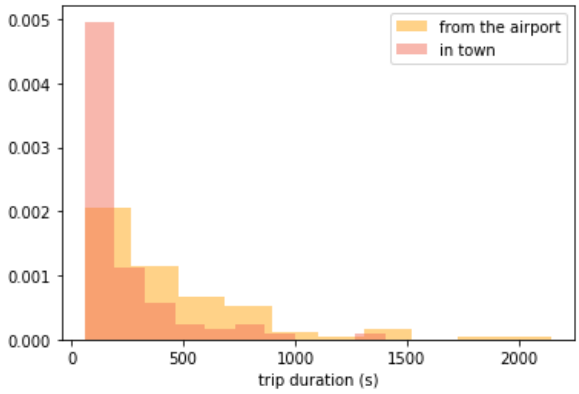


图 5.2.1.3-3 时长分布

可以看到大致的趋势为 9-24 时为上海虹桥国际机场的高峰期，其他时间为低峰期。高峰期中不同时间段的机场繁忙程度是波动的，不过相差并不太大。司机的到达情况与航班的到达情况基本符合，并稍有时间上的延后。

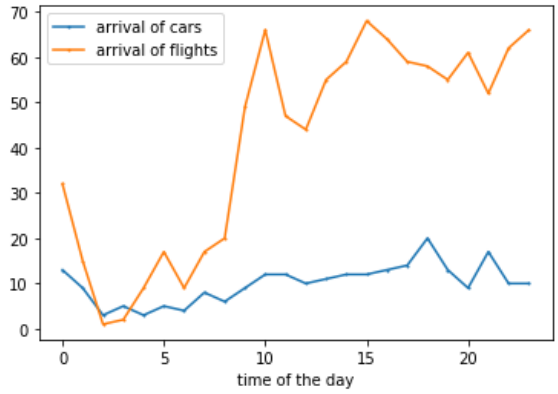
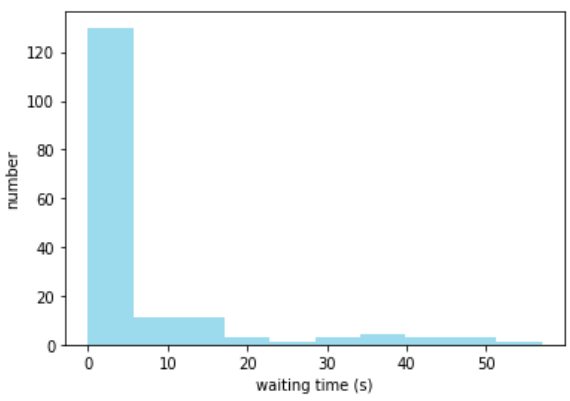


图 5.2.1.3-4 航班及出租车到达随时间分布情况

分析抽取出从机场出发的行程，处理得到司机在机场的等待时间，可以验证其大致服从指数分布，符合模型的假设。




```
        Distribution Summary

Distribution:    Exponential
Expression:      -0.001 + EXPO(345)
Square Error:    0.095134

Chi Square Test
  Number of intervals   = 4
  Degrees of freedom    = 2
  Test Statistic        = 54.5
  Corresponding p-value < 0.005

Kolmogorov-Smirnov Test
  Test Statistic        = 0.505
  Corresponding p-value < 0.01
```

图 5.2.1.3-5 出租车机场等待载客时长分布图及检验结果

5.2.1.4 数据分析

由于出租车各时段行驶的里程数未知，近似将到达时间与出发时间之差作为衡量出租车收益的指标。

根据第一问建立的理论模型，在可获得的数据的支持下，可近似将决定司机车选择方案的收益成本函数（R-C*t）等价为等待载客时长、载客时长和平均载客率的函数$k_1\mathrm{T_c}-\alpha\cdot k_2\mathrm{T_r}$（其中，$k_1$为从机场载客的平均收益率，$k_2$为在市区载客的平均收益率，α为在市区的平均载客时长与行驶总时长的比，T_c为从机场出发的载客行程的平均时长，T_r为在机场等待的平均时长）

5.2.1.5 数据使用

利用从机场出发的载客数据计算出每辆出租车在机场的等待时间t_c、载下一单客的时间t_r、并利用极大似然的思想，分别求得期望。利用在市区载客的数据计算出全天在市区的载客率α。

收益成本大于 0 时，则建议出租车司机进入“蓄车池”等待，反之，则建议出租车司机空载回市区拉客。

5.2.2 相关性依赖分析

由第一问建立的模型可知，乘客数量及排队等候的车辆数决定了司机的选择方案，对这三个变量进行相关性依赖的分析。以 1 小时为划分的单位，统计出整点附近一小时到达的航班数量以及从数据集中可统计到的出租车数量，计算相关系数矩阵。

表 5.2.2-1 时间、出租车数量、航班数量相关系数矩阵

	时间	出租车数量	航班数量
时间	1	0.60	0.83
出租车数量	0.60	1	0.75
航班数量	0.83	0.75	1

可以看到，时间与出租车数量，时间与航班数量，出租车数量与航班数量之间都具有很强的相关性，航班数量确实很大程度上影响了出租车司机的决策，让他们做出等待的选择。

5.2.3 模型合理性分析

抽取从机场出发的行程，将对应的这些司机各自所有的从机场出发的行程的载客时间以及之前等待的时间分别求和，带入公式$k_1\mathrm{T_c}-\alpha\cdot k_2\mathrm{T_r}$中，如果结果为正值，则代表该司机的选择支持了模型中给出的方案，如果结果为负值则代表该司机的选择不同于模型给出的方案。

最终算出，收集到数据的司机中选择支持模型中给出方案的比例为 0.74，在司机大都经验丰富，并且大部分时间做出理性决策的条件下可以看出所建立模型对司机给出的选择方案的合理性。

5.3 问题三的求解

设N_r表示系统中排队等待乘车的乘客数的期望值；W_r表示某组乘客在系统中排队等候的时间的期望值；T_0表示乘客等到出租车后的上车时间，T_0与“上车点”内的泊位数量成函数关系即$T_0=as+b$；η表示停车点上车人数与停车点泊位之比的均值（停车点平均利用率）；N_c表示系统中排队载客的出租车数的期望值；W_c表示出租车在系统中排队等候的时间；A_r、A_c表示单位时间内能被服务完的乘客或出租车的期望值；t 时刻系统中有 n 位乘客的概率为$P_n(t)$,有 k 辆出租车的概率为$E_k(t)$。通过建立多窗口双向等待制排队模型，综合考虑等待人数、等待车数、乘客等待时

间、出租车等待时间、绝对通行能力等指标，给出最优窗口数及上车点设计方案。由于乘客等到出租车后上车的时间较短且稳定，本系统中只等待与逗留行为差别不大，且逗留时间可在通行能力中有所体现，故不考虑逗留指标。最后，结合数据，运用 Arena 软件对本模型进行仿真模拟。

5.3.1 多窗口双向等待制排队模型的建立

由假设可知单位时间内到达的乘客数和出租车数都服从泊松分布，从而乘客到达时间间隔和出租车到达时间间隔都服从指数分布。又有假设系统容量和顾客源无限，排队系统服从先到先得的规则，本题可看作一个多服务台、系统容量和顾客源无限、先到先得的排队系统。注意到，系统中出租车和乘客都存在排队行为，若将乘客看作服务对象，出租车看作服务窗口则顾客按泊松分布到达系统且相互独立，有 s 个服务窗口进行服务，服务时间服从指数分布，构成一个 M/M/s 排队模型；若将出租车看作服务对象，乘客看作服务窗口，同样构成一个顾客按泊松分布到达系统且相互独立，有 s 个服务窗口进行服务，服务时间服从指数分布的 M/M/s 排队模型。又考虑到“上车点”泊位的多少会影响乘客等到出租车后上车需要的时间，因为增加“上车点”泊位会增加乘客选择出租车的时间和上车需要行走的路程。将上车时间用函数关系与泊位数量联系起来并加入服务时间。

平均等待人数计算公式为：

$$N_r = \sum_{k=s}^{\infty}(k-s)\,P_k \tag{11}$$

停车点平均利用率计算公式为为：

$$\eta = \sum_{k=0}^{\infty} k\,P_k \tag{12}$$

乘客平均等待时间计算公式为：

$$W_r = \frac{N_r}{\lambda} \tag{13}$$

乘客等待概率计算公式为：

$$P\{x > s\} = \sum_{k=s}^{\infty} P_k \tag{14}$$

平均等待车数计算公式为：

$$N_c = \sum_{k=s}^{\infty}(k-s)\,E_k \tag{15}$$

出租车平均等待时间计算公式为：

$$W_{cq} = \frac{N_c}{\tau} \tag{16}$$

出租车等待概率计算公式为：

$$E\{x > s\} = \sum_{k=s}^{\infty} E_k \tag{17}$$

5.3.2 多窗口双向等待制排队模型的求解

设乘客到达时间间隔与出租车到达时间间隔分布服从参数为λ、τ的指数分布。以将乘客看作服务对象，出租车看作服务窗口的 M/M/s 排队模型为例计算指标参数。首先计算。首先计算系统中有 k 位乘客的概率，取初始时刻为$t=0$，由全概公式可得系统在任意时刻t的状态概率

$$\begin{cases} P_0(t+\Delta t)=P_0(t)(1-\lambda\Delta t)+P_1(t)(\tau+T_0)\Delta t+o(\Delta t) \\ P_k(t+\Delta t)=P_{k-1}(t)\lambda\Delta t+P_k(1-\lambda\Delta t-k(\tau+T_0)\Delta t)+P_{k+1}(t)(k+1)(\tau+T_0)\Delta t+o(\Delta t),1\le k\le s-1 \\ P_k(t+\Delta t)=P_{k-1}(t)\lambda\Delta t+P_k(t)(1-\lambda\Delta t-s\tau\Delta t)++P_{k+1}(t)n(\tau+T_0)\Delta t+o(\Delta t),k\ge s \end{cases} \quad (18)$$

可得稳态方程为

$$\begin{cases} \lambda P_0=P_1(\tau+T_0) \\ P_{k+1}(k+1)(\tau+T_0)+\lambda P_{k-1}=P_k(\lambda+k(\tau+T_0)),1\le k\le s-1 \\ s\tau P_{k+1}+\lambda P_k=P_k(\lambda+s(\tau+T_0)),k\ge s \end{cases} \quad (19)$$

由递推 2 关系求得系统的状态概率为：

$$\begin{cases} \frac{\rho^k}{k!}P_0=\frac{s^k}{k!}\rho^kP_0,0\le k\le s-1 \\ \frac{\rho^k}{s!s^{k-s}}P_0=\frac{s^s}{s!}\rho^kP_0,k\ge s-1 \end{cases}, \text{其中}\rho=\lambda/s(\tau+T_0) \quad (20)$$

当$\rho<1$时，有

$$1=\sum_{k=0}^{\infty}P_k=P_0\left(\sum_{n=0}^{s-1}\frac{\rho^k}{k!}+\sum_{k=s}^{\infty}\frac{\rho^k}{s!s^{k-s}}\right)=P_0\left(\sum_{k=0}^{s-1}\frac{\rho^k}{k!}+\frac{\rho^s}{s!(1-s\rho)}\right) \quad (21)$$

得

$$P_0=\left(\sum_{k=0}^{s-1}\frac{\rho^k}{k!}+\frac{\rho^s}{s!}\frac{1}{1-s\rho}\right)^{-1} \quad (22)$$

由此可得平均等待人数为：

$$N_r=\sum_{k=s}^{\infty}(k-s)\,P_k=\sum_{i=1}^{\infty}i\,P_{i+s}=\frac{s\rho(s^2\rho)^s}{s!(1-s\rho)^2}P_0 \quad (23)$$

停车点平均利用率为：

$$\eta=\sum_{k=0}^{\infty}k\,P_k=\sum_{k=0}^{s-1}k\,P_k+s\sum_{k=s}^{\infty}P_k=\rho \quad (24)$$

乘客平均等待时间为：

$$W_r = \frac{N_r}{\lambda} = \frac{s\rho(s^2\rho)^s}{s!(1-s\rho)^2\lambda}P_0 \tag{25}$$

乘客等待概率为：

$$P\{x > s\} = \sum_{k=s}^{\infty} P_k = \sum_{k=s}^{\infty} \frac{\rho^k}{s!s^{k-s}}P_0 = \sum_{k=s}^{\infty} \rho^{k-s}P_s = \frac{P_s}{1-s\rho} = \frac{sP_{s-1}}{s-\rho} \tag{26}$$

同理将出租车看作服务对象，乘客看作服务窗口的 M/M/s 排队模型为例计算指标参数。对于系统的状态概率为：

$$\begin{cases} \frac{\varphi^k}{k!}E_0 = \frac{s^k}{k!}\varphi_1^k E_0, 0 \le k \le s-1 \\ \frac{\varphi^k}{n!n^{k-n}}E_0 = \frac{n^n}{n!}\varphi_1^k E_0, k \ge s-1 \end{cases} (10), \text{其中} \varphi = \tau / s(\lambda + T_0)。$$

当$\varphi > 1$时，有

$$1 = \sum_{k=0}^{\infty} E_k = E_0\left(\sum_{k=0}^{s-1}\frac{\varphi^k}{k!} + \sum_{k=s}^{\infty}\frac{\varphi^k}{s!s^{k-s}}\right) = E_0\left(\sum_{k=0}^{s-1}\frac{\varphi^k}{k!} + \frac{\varphi^s}{s!(1-s\varphi)}\right) \tag{27}$$

得

$$E_0 = \left(\sum_{k=0}^{s-i}\frac{\varphi^k}{k!} + \frac{\varphi^s}{s!}\frac{1}{1-s\varphi}\right)^{-1} \tag{28}$$

由此可得平均等待车数为：

$$N_c = \sum_{k=s}^{\infty}(k-s)\,E_k = \sum_{i=1}^{\infty} i\,E_{i+s} = \frac{s\varphi(s^2\varphi)^s}{s!(1-s\varphi)^2}E_0 \tag{29}$$

出租车平均等待时间为：

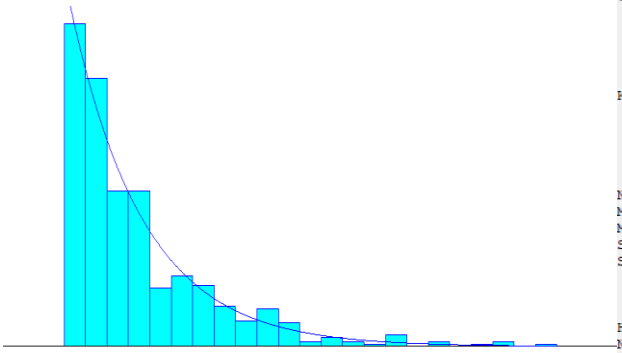
$$W_{cq} = \frac{N_c}{\tau} = \frac{s\varphi(s^2\varphi)^s}{s!(1-s\varphi)^2\tau}E_0 \tag{30}$$

出租车等待概率为：

$$E\{x > s\} = \sum_{k=s}^{\infty} E_k = \sum_{k=s}^{\infty} \frac{\varphi^k}{s!s^{k-s}}E_0 = \sum_{k=s}^{\infty} \varphi^{k-s}E_s = \frac{E_s}{1-s\varphi} = \frac{nE_{s-1}}{n-\varphi} \tag{31}$$

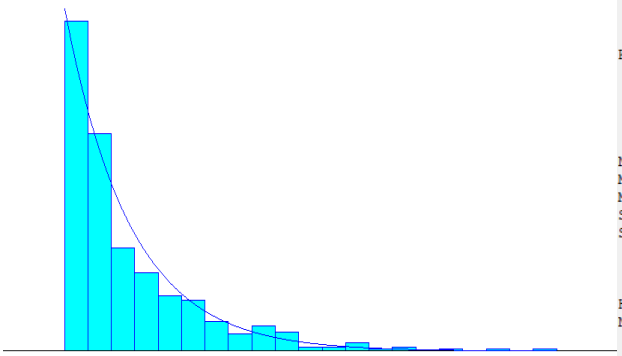
5.3.3 仿真模拟与结论

根据数据集（见支撑材料）中出租车到达时间间隔和乘客到达时间间隔的数据，利用 Arena 软件给出了出租车到达时间间隔和乘客到达时间间隔的拟合分布以及拟合报告。从拟合报告中可看到两个拟合分布的方差都较小，且 p-value 小于 0.005。可以认为出租车到达时间间隔服从参数为 0.893 的指数分布，乘客到达时间间隔服从参数为 1.06 的指数分布。根据实际经验将上车时间确定为$T_0 = 0.5s + 2$。由于蓄车去有两条并行车道，“上车点”的泊位只能为 2、4、6 等依次增长。



```
       Distribution Summary

Distribution:   Exponential
Expression:     -0.001 + EXPO(0.893)
Square Error:   0.002514

Chi Square Test
  Number of intervals   = 11
  Degrees of freedom    = 9
  Test Statistic        = 18.4
  Corresponding p-value = 0.0324

Kolmogorov-Smirnov Test
  Test Statistic        = 0.0343
  Corresponding p-value > 0.15

       Data Summary

Number of Data Points   = 538
Min Data Value          = 0
Max Data Value          = 5.82
Sample Mean             = 0.892
Sample Std Dev          = 0.931

       Histogram Summary

Histogram Range         = -0.001 to 6
Number of Intervals     = 23
```

图 5.3.3-1 车数据拟合情况



```
       Distribution Summary

Distribution:   Exponential
Expression:     EXPO(1.06)
Square Error:   0.004614

Chi Square Test
  Number of intervals   = 10
  Degrees of freedom    = 8
  Test Statistic        = 17.4
  Corresponding p-value = 0.0261

Kolmogorov-Smirnov Test
  Test Statistic        = 0.0641
  Corresponding p-value = 0.0472

       Data Summary

Number of Data Points   = 452
Min Data Value          = 0.00113
Max Data Value          = 7.94
Sample Mean             = 1.06
Sample Std Dev          = 1.18

       Histogram Summary

Histogram Range         = 0 to 8
Number of Intervals     = 21
```

图 5.3.3-2 乘客数据拟合情况

分别令 n=2、4、6、8、10，得到系统指标为如下表：

表 5.3.3-1 不同泊位系统指标对比表

s	2	4	6	8	10
N_r	17.7851	4.5608	1.5528	0.336	0.0980
W_r	23.8421	4.5899	1.3481	0.2912	0.0776
N_c	29.588	12.9618	2.5598	0.6254	1.566
W_c	26.1101	13.5749	2.6712	0.7606	1.1241
η	0.99845	0.935525	0.82735	0.757175	0.6046
A_c	58	90	96	96	90

从上表可看出“上车点”泊位数量小于等于 8 时，在随着泊位数量的增加，系统内的等待人数、等待车数、乘客等待时间、出租车等待时间都减小，系统绝对通行率上升，但泊位利用率减小。同时泊位数量较小时等待车数和人数及等待时间的减小和绝对通行率的增加较为显著，。当泊位大于 8 时，综合考虑乘客和出租车的等待行为等待时间并没有减少，绝对通行率和泊位利用率也有所降低。说明适当增加“上车点”泊位数量可以减少出租车和乘客的等待时间并提供通行效率，但是泊位并不是越多越好，较低的泊位利用率不仅不利于“蓄车池”通行而且造成机场资源浪费。综合考虑乘客和出租车的等待行为、泊位利用率和绝对通行率，机场“上车点”的泊位数量定在 4 至 6 个之间较为合适。

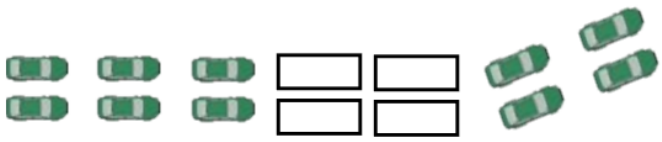


图 5.3.3-3 机场“上车点”设计方案

5.4 问题四的求解

5.4.1 带有优先权的排队模型的构建与求解

首先设 λ_1 表示第一优先级短途出租车到达服从泊松分布的参数，即短途车道到达的间隔时间；λ_2 表示第二优先级长途出租车到达服从泊松分布的参数，即长途车道到达的间隔时间；W_1 表示稳定状态下第一优先级即短途出租车平均等待时间；W_2 表示稳定状态下第二优先级即长途出租车平均等待时间；K_1 表示短途出租车载客的里程数；K_2表示长途出租车载客的里程数；m 表示单位里程的期望收益；q 表示时间成本的相关常数。

基于假设，综合带有优先权的排队模型，即在排队系统中的出租车出发时间顺序基于他们具有优先级（符合非强占性优先权），于是可得到

$$W_1 = \frac{1}{TB_0B_1} + \frac{1}{\mu} \tag{32}$$

$$W_2 = \frac{1}{TB_1B_2} + \frac{1}{\mu} \tag{33}$$

考虑到原有车道为并行车道，并再设立一个优先车道，得到一个平均到达时间λ

$$\lambda = \lambda_1 + 2 * \lambda_2 \tag{34}$$

再令

$$r = \frac{\lambda}{\mu} \tag{35}$$

同时满足

$$B_0 = 1 \tag{36}$$

$$B_k = 1 - \frac{\sum_{i=1}^{k} \lambda_i}{\mathrm{s}\mu} \tag{37}$$

此时T 满足如下式子

$$\mathrm{T} = \mathrm{s}! \frac{\mathrm{s}\mu - \lambda}{\mathrm{r}^{\mathrm{s}}} \sum_{\mathrm{j}=0}^{\mathrm{s}-1} \frac{\mathrm{r}^{\mathrm{j}}}{\mathrm{j}!} + \mathrm{s}\mu \tag{38}$$

注：$\sum_{i=1}^{k} \lambda_i < \mathrm{s}\mu$使两个优先级能够达到稳定状态

在该模型的基础下，从管理部门的角度考虑，选择使得载到长途、短途客人的出租车的收益尽可能均衡，短途客车的单位里程载客期望收益为$\mathrm{E}K_1$，长途客车的单位里程载客期望收益为$\mathrm{E}K_2$，结合收益函数得到

$$\mathrm{m} * \mathrm{E}K_1 \ - q * \ W_1 = \ \ \mathrm{m} * \mathrm{E}K_2 \ - q * \ W_2 \tag{39}$$

将式（32）-（38）带入式（39）得到

$$\mathrm{E}K_2 - \mathrm{E}K_1 = \frac{\mathrm{s}\mu(\lambda_1+\lambda_2)}{m*[\mathrm{s}\mu-(\lambda_1+\lambda_2)](\mathrm{s}\mu-\lambda_1)\left(\mathrm{s}!\frac{\mathrm{s}\mu-\lambda}{\mathrm{r}^{\mathrm{s}}}\sum_{\mathrm{j}=0}^{\mathrm{s}-1}\frac{\mathrm{r}^{\mathrm{j}}}{\mathrm{j}!}+\mathrm{s}\mu\right)} \tag{40}$$

由于指数分布的性质，可得

$$\frac{1}{\lambda_1} + \frac{1}{\lambda_2} = \frac{1}{\lambda_0} \tag{41}$$

注：λ_0为一条普通车道出租车的到达间隔时间，为确定的常数。

5.4.2 仿真模拟与结论

假设

$$\lambda_1 = n * \lambda_2 \tag{42}$$

通过 Arena 软件继续仿真模拟训练，得到当 $n = 4$，整个系统的通行效率最大，部分结果如下表（其余结果见附录）：

表 5.4.2-1 仿真模拟训练带优先车道的排队情况

k	λ_1	λ_2	A_c	W_1	W_2
1	1.786	1.786	89	20.826	8.988
2	2.679	1.340	88	12.900	11.613
3	3.572	1.191	97	8.485	12.566
4	4.465	1.116	115	2.148	10.176
5	5.358	1.072	99	1.910	11.011
6	6.251	1.042	89	1.477	10.101

将式（42）带回式（41）得到$\lambda_1 = 5\lambda_0$，$\lambda_2 = \frac{5}{4}\lambda_0$时，带回式（40），在其他参数假定已知的情况下，可求出此时的$\mathrm{E}K_2 - \mathrm{E}K_1$

根据里程数的概率密度函数，设为$\mathrm{P}(x)$，短途长途的分界点为K

$$\int_K^{\infty} x\, P_{x|\mathrm{x}>K}(x|\mathrm{x} > K)dx - \int_0^K x\, P_{x|\mathrm{x}\le K}(x|\mathrm{x} \le K)dx = \mathrm{E}K_2 - \mathrm{E}K_1 \tag{43}$$

解式（43）方程，可以得到分界点 K 的值，即判别出租车进入普通或优先车道的依据，当载客里程数大于 K 时，出租车属于长途载客，走普通车道；当载客里程数小于等于 K 时，出租车属于短途载客，走优先车道，其余正常排在普通车道，在这种方案下出租车的收益近似均衡。

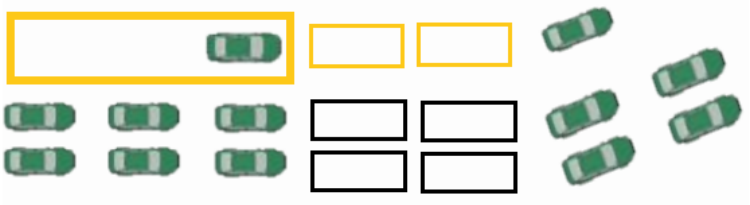


图 5.4.2-1 机场优先车道设计方案

六、 模型的评估与优化

6.1 模型优点

（1） 模型综合了出租车、乘客、机场管理部门三方对出租车是否排队、“上车点”设置的泊位数量以及如何开放短途载客返回出租车的优先权问题进行决策。

（2） 引入时间成本、机会成本概念对出租车收益计算全面、精确。

（3） 模型直观，给出的决策便于理解，可行性强。

（4） 仿真模拟给出的决策较为科学可信。

6.2 模型缺点

（1） 数据维度单一，有一定偏差。

（2） 模型适用于客流量大的机场。

6.3 模型优化

（1） 可将影响出租车是否排队的相关因素进一步细分。

（2） 可通过收集更加多维的数据对决策结果进行修正

七、 参考文献

[1]孙健. 基于排队论的航空枢纽陆侧旅客服务资源建模与仿真[D].中国矿业大学(北京),2017.

[2]孙健,丁日佳,陈艳艳.基于排队论的单车道出租车上客系统建模与仿真[J].系统仿真学报,2017,29(05):996-1004.

[3]李娜,贾博,江志斌,谢梦得.考虑顾客体验的排队系统研究[J].工业工程与管理,2012,17(03):36-40+46.

[4]黄业文,邝神芬.非强占有限优先权 M/M/n/m 排队系统[J].应用概率统计,2018,34(04):364-380.

[5]陈潇.在超市排队系统中运用 Arena 仿真建模[J].河北企业,2017(04):51-52.

附录

使用软件：Python、Arena

程序：

```
import os
import pandas as pd
import numpy as np
from math import sqrt
import datetime

os.chdir("C:/Taxi_070220")

"""
calculate each car's total waiting time at the airport
"""
duration = []
for p in range(1,101):
    train = pd.read_csv('Taxi (%d)' %
p,header=None,names=['car_id','time','longitude','latitude','speed','angle','flag'])
    car_id = train['car_id'][0]
    def haversine_array(lat1, lng1, lat2, lng2):
        lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
        AVG_EARTH_RADIUS = 6371  # in km
        lat = lat2 - lat1
        lng = lng2 - lng1
        d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
        h = 2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(d))
        return h

    def geo_distance(lat1, lng1, lat2, lng2):
        a = haversine_array(lat1, lng1, lat1, lng2)
        b = haversine_array(lat1, lng1, lat2, lng1)
```

```
        return a + b

    def bridge_trip(x):
        return x['distance_to_bridge'] < 5

    train['bridge_lat'] = 31.19590
    train['bridge_lon'] = 121.34113

    train.loc[:, 'distance_to_bridge'] =  geo_distance(train['latitude'].values, train['longitude'].values,
train['bridge_lat'].values, train['bridge_lon'].values)
    train['is_bridge'] = train.apply(bridge_trip,axis=1)
    test = train[train['is_bridge']]

    if not test.empty:
        r1 =  pd.DataFrame(columns=train.columns)
        r2 =  pd.DataFrame(columns=train.columns)
        for i in range(1,len(train)-1):
            if train['is_bridge'][i] == True and train['is_bridge'][i-1] == False:
                r1 = r1.append(train[i:i+1])
            if train['is_bridge'][i] == True and train['is_bridge'][i+1] == False:
                r2 = r2.append(train[i:i+1])
        r1['timescale'] = 0

        for j in range(len(r1)-1):
            t1 = datetime.datetime.strptime(r1['time'].values[j], "%Y-%m-%d %H:%M:%S")
            t2 = datetime.datetime.strptime(r2['time'].values[j], "%Y-%m-%d %H:%M:%S")
            total_interval_time = (t2 - t1).total_seconds()
            r1['timescale'].values[j] = total_interval_time

        duration.append([car_id,r1['timescale'].sum()])

d = np.matrix(duration)
```

```
d = pd.DataFrame(d)
d.to_csv("d.csv",index=False)


"""
calculate each car's total carrying time of routes starting from the airport
"""
timesum = []
for i in range(1,101):
    train = pd.read_csv('Taxi (%d)' %
i,header=None,names=['car_id','time','longitude','latitude','speed','angle','flag'])
    car_id = train['car_id'][0]

    recordpick = pd.DataFrame(columns=['car_id','time','longitude','latitude','speed','angle','flag'])
    recorddrop = pd.DataFrame(columns=['car_id','time','longitude','latitude','speed','angle','flag'])
    for i in range(len(train)):
        if i == 0:
            recordpick = recordpick.append(train[i:i+1])

        if train.iloc[i,6]!=recordpick.iloc[len(recordpick)-1,6]:
            recordpick = recordpick.append(train[i:i+1])
            recorddrop = recorddrop.append(train[i-1:i])
        else:
            if recordpick.iloc[len(recordpick)-1,4] != 0 :
                if train.iloc[i,4] == 0 :
                    recordpick = recordpick.append(train[i:i+1])
                    recorddrop = recorddrop.append(train[i-1:i])
            if recordpick.iloc[len(recordpick)-1,4] == 0 :
                if train.iloc[i,4] != 0 :
                    recordpick = recordpick.append(train[i:i+1])
                    recorddrop = recorddrop.append(train[i-1:i])
    if len(recordpick) > len(recorddrop):
        recorddrop = recorddrop.append(train[len(train)-1:len(train)])
```

```
recordpick['trip_id'] = range(len(recordpick))
recorddrop['trip_id'] = range(len(recordpick))


train = pd.merge(recordpick,recorddrop,on='trip_id')

def haversine_array(lat1, lng1, lat2, lng2):
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    AVG_EARTH_RADIUS = 6371  # in km
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
    h = 2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(d))
    return h

def geo_distance(lat1, lng1, lat2, lng2):
    a = haversine_array(lat1, lng1, lat1, lng2)
    b = haversine_array(lat1, lng1, lat2, lng1)
    return a + b

train['bridge_lat'] = 31.19590
train['bridge_lon'] = 121.34113

train.loc[:, 'pickup_distance_to_bridge'] =  geo_distance(train['latitude_x'].values, train['longitude_x'].values,
train['bridge_lat'].values, train['bridge_lon'].values)
train.loc[:, 'dropoff_distance_to_bridge'] =  geo_distance(train['latitude_y'].values, train['longitude_y'].values,
train['bridge_lat'].values, train['bridge_lon'].values)


def bridge_trip(x):
    return int(x['pickup_distance_to_bridge'] < 5) and int(x['dropoff_distance_to_bridge'] > 5)
```

```
    train = train.drop(['bridge_lat','bridge_lon'],axis=1)
    train['is_bridge'] = train.apply(bridge_trip,axis=1)

    train = train.drop(['car_id_y','flag_y','pickup_distance_to_bridge','dropoff_distance_to_bridge'], axis=1)
    train.columns=['car_id','pickup_time','pickup_longitude','pickup_latitude','p_speed','p_angle','flag','trip_id','dropoff_time','dropoff_longitude','dropoff_latitude','d_speed','d_angle','is_bridge']

    train = train[['car_id','trip_id','flag','p_speed','p_angle','d_speed','d_angle','pickup_time','dropoff_time','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','is_bridge']]
    train['pickup_time'] = pd.to_datetime(train.pickup_time)
    train['dropoff_time'] = pd.to_datetime(train.dropoff_time)

    train['duration'] = 0
    for i in range(len(train)):
        train['duration'][i] = (train['dropoff_time'][i] - train['pickup_time'][i]).seconds

    train = train[train['duration'] > 0]
    train = train[['car_id','trip_id','flag','pickup_time','dropoff_time','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','is_bridge','duration']]

    r1 = train[train['is_bridge'] == 1]
    if not r1.empty:
        r2 = r1[r1['flag'] == 1]

        timesum.append([car_id,r2['duration'].sum()])

s = np.matrix(timesum)
s = pd.DataFrame(s)
s.to_csv("s.csv",index=False)
```

```
"""
calculate the ratio of time cars running with passengers in town against total time
"""
res =
pd.DataFrame(columns=['car_id','trip_id','status','pickup_time','dropoff_time','pickup_longitude','pickup_latitude','
dropoff_longitude','dropoff_latitude','is_bridge','duration'])

for i in range(1,701):
    train = pd.read_csv('Taxi (%d)' %
i,header=None,names=['car_id','time','longitude','latitude','speed','angle','flag'])

    recordpick = pd.DataFrame(columns=['car_id','time','longitude','latitude','speed','angle','flag'])
    recorddrop = pd.DataFrame(columns=['car_id','time','longitude','latitude','speed','angle','flag'])
    for i in range(len(train)):
        if i == 0:
            recordpick = recordpick.append(train[i:i+1])

        if train.iloc[i,6]!=recordpick.iloc[len(recordpick)-1,6]:
            recordpick = recordpick.append(train[i:i+1])
            recorddrop = recorddrop.append(train[i-1:i])
        else:
            if recordpick.iloc[len(recordpick)-1,4] != 0 :
                if train.iloc[i,4] == 0 :
                    recordpick = recordpick.append(train[i:i+1])
                    recorddrop = recorddrop.append(train[i-1:i])
            if recordpick.iloc[len(recordpick)-1,4] == 0 :
                if train.iloc[i,4] != 0 :
                    recordpick = recordpick.append(train[i:i+1])
                    recorddrop = recorddrop.append(train[i-1:i])
    if len(recordpick) > len(recorddrop):
        recorddrop = recorddrop.append(train[len(train)-1:len(train)])
```

```
recordpick['trip_id'] = range(len(recordpick))
recorddrop['trip_id'] = range(len(recordpick))


train = pd.merge(recordpick,recorddrop,on='trip_id')

def haversine_array(lat1, lng1, lat2, lng2):
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    AVG_EARTH_RADIUS = 6371  # in km
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
    h = 2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(d))
    return h

def geo_distance(lat1, lng1, lat2, lng2):
    a = haversine_array(lat1, lng1, lat1, lng2)
    b = haversine_array(lat1, lng1, lat2, lng1)
    return a + b

train['bridge_lat'] = 31.19590
train['bridge_lon'] = 121.34113


train.loc[:, 'pickup_distance_to_bridge'] =  geo_distance(train['latitude_x'].values, train['longitude_x'].values,
train['bridge_lat'].values, train['bridge_lon'].values)
train.loc[:, 'dropoff_distance_to_bridge'] =  geo_distance(train['latitude_y'].values, train['longitude_y'].values,
train['bridge_lat'].values, train['bridge_lon'].values)


def bridge_trip(x):
```

```
        return int(x['pickup_distance_to_bridge'] < 5) or int(x['dropoff_distance_to_bridge'] < 5)

    train = train.drop(['bridge_lat','bridge_lon'],axis=1)
    train['is_bridge'] = train.apply(bridge_trip,axis=1)

    train = train.drop(['car_id_y','flag_y','pickup_distance_to_bridge','dropoff_distance_to_bridge'], axis=1)
    train.columns=['car_id','pickup_time','pickup_longitude','pickup_latitude','p_speed','p_angle','status','trip_id','dropoff_time','dropoff_longitude','dropoff_latitude','d_speed','d_angle','is_bridge']

    # status 0 空载
    # status 1 载客
    # status 2 停下
    for i in range(len(train)):
        if train['p_speed'][i] + train['status'][i] == 0:
            train['status'][i] = 2

    train = train[['car_id','trip_id','status','p_speed','p_angle','d_speed','d_angle','pickup_time','dropoff_time','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','is_bridge']]
    train['pickup_time'] = pd.to_datetime(train.pickup_time)
    train['dropoff_time'] = pd.to_datetime(train.dropoff_time)

    train['duration'] = 0
    for i in range(len(train)):
        train['duration'][i] = (train['dropoff_time'][i] - train['pickup_time'][i]).seconds

    train = train[train['duration'] > 0]
    train = train[['car_id','trip_id','status','pickup_time','dropoff_time','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','is_bridge','duration']]

    common = train[train['is_bridge'] == 0]
```

```
    res = res.append(common)


# status 1
time1 = res[res['status'] == 1].groupby(['car_id'])['duration'].sum()
# status not 2
time2 = res[res['status'] != 2].groupby(['car_id'])['duration'].sum()


time = pd.merge(time1,time2,on='car_id')
time.loc[:,'ratio'] = time['duration_x']/time['duration_y']
a = time['ratio'].mean()


"""
calculate the number of cars arriving at the airport every hour
"""
res = pd.DataFrame(columns=['car_id_x','dropoff_hour'])
for p in range(1,101):
    train = pd.read_csv('Taxi (%d)' %
p,header=None,names=['car_id','time','longitude','latitude','speed','angle','flag'])


    car_id = train['car_id'][0]
    #train = pd.read_csv('Taxi (1)',header=None,names=['car_id','time','longitude','latitude','speed','angle','flag'])
    recordpick = pd.DataFrame(columns=['car_id','time','longitude','latitude','speed','angle','flag'])
    recorddrop = pd.DataFrame(columns=['car_id','time','longitude','latitude','speed','angle','flag'])
    for i in range(len(train)):
        if i == 0:
            recordpick = recordpick.append(train[i:i+1])


        if train.iloc[i,6]!=recordpick.iloc[len(recordpick)-1,6]:
            recordpick = recordpick.append(train[i:i+1])
```

```
            recorddrop = recorddrop.append(train[i-1:i])
        else:
            if recordpick.iloc[len(recordpick)-1,4] != 0 :
                if train.iloc[i,4] == 0 :
                    recordpick = recordpick.append(train[i:i+1])
                    recorddrop = recorddrop.append(train[i-1:i])
            if recordpick.iloc[len(recordpick)-1,4] == 0 :
                if train.iloc[i,4] != 0 :
                    recordpick = recordpick.append(train[i:i+1])
                    recorddrop = recorddrop.append(train[i-1:i])
    if len(recordpick) > len(recorddrop):
        recorddrop = recorddrop.append(train[len(train)-1:len(train)])

    recordpick['trip_id'] = range(len(recordpick))
    recorddrop['trip_id'] = range(len(recordpick))


    train = pd.merge(recordpick,recorddrop,on='trip_id')


    def haversine_array(lat1, lng1, lat2, lng2):
        lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
        AVG_EARTH_RADIUS = 6371  # in km
        lat = lat2 - lat1
        lng = lng2 - lng1
        d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
        h = 2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(d))
        return h

    def geo_distance(lat1, lng1, lat2, lng2):
        a = haversine_array(lat1, lng1, lat1, lng2)
        b = haversine_array(lat1, lng1, lat2, lng1)
```

```
        return a + b

    def bridge_trip(x):
        return x['dropoff_distance_to_bridge'] < 5

    train['bridge_lat'] = 31.19590
    train['bridge_lon'] = 121.34113

    train.loc[:, 'dropoff_distance_to_bridge'] =  geo_distance(train['latitude_y'].values, train['longitude_y'].values,
train['bridge_lat'].values, train['bridge_lon'].values)
    train['dropoff_hour'] = pd.to_datetime(train.time_y).dt.hour + pd.to_datetime(train.time_y).dt.minute//30

    train['is_bridge'] = train.apply(bridge_trip,axis=1)
    train = train[train['is_bridge']]

    if not train.empty:
        subdf = train.drop_duplicates(['car_id_x','dropoff_hour'])

        subdf = subdf[['car_id_x','dropoff_hour']]

        res = res.append(subdf)

see = res.groupby('dropoff_hour')['car_id_x'].count()

see.to_csv("hour_cars.csv",index=False)

"""
calculate the correlation coefficient matrix
"""
cars = pd.read_csv("hour_cars.csv")
flights = pd.read_csv("hour_flights.csv")
res = pd.merge(cars,flights,on='time')[['time','cars','flights']]
```

```
"""
calculate the proportion of car drivers support the model
"""
a = 0.5465517027089453
d = pd.read_csv("d.csv")
s = pd.read_csv("s.csv")

to_process = pd.merge(s,d,how='left',on='car_id')

def cal_gain(r,c):
    # timesum - a * duration
    return r - a * c

to_process.loc[:,'gain'] = cal_gain(to_process['timesum'].values,to_process['duration'].values)
to_test = to_process[['gain']]

to_test['is_gain'] = (to_test['gain'] > 0).astype('int')
proportion = len(to_test[to_test['gain'] < 0]) / len(to_test)

"""
plot the distribution of time cars spent on waiting to get customers
"""
train['dropoff_time'] = pd.to_datetime(train.dropoff_time)
new = train.groupby(['car_id','dropoff_hour']).apply(lambda dropoff_time:(dropoff_time.max()-
dropoff_time.min()))

new['waiting_time'] = 0
for i in range(len(new)):
    new['waiting_time'][i] = new['dropoff_time'][i].seconds

time = new[['waiting_time']]
```

```
time = time.reset_index()


to_draw = time[time['dropoff_hour'] > 9]

to_draw = to_draw[to_draw['dropoff_hour'] < 23]


X = []

for i in range(len(to_draw)):

    X.append(i)


import matplotlib.pyplot as plt

plt.hist(to_draw['waiting_time']/60)

plt.xlabel("waiting time (s)")

plt.ylabel("number")
```