Optimization of Ride Hailing Operations in NYC: A Model-Based Approach

Julia VanPutte (JHV42)

November 19, 2023

Abstract

This report analyzes the ride-hailing landscape in New York City, with a special focus on non-shared rides, excluding Staten Island. The analysis employs a model loosely based on Uber's operations, considering 20,000 ride requests per hour, each averaging a 15-minute duration, with riders willing to wait a maximum of 7 minutes before potentially canceling. The study involves 4,600 drivers, evenly distributed across 30 city zones, to evaluate service efficiency and customer satisfaction. A significant aspect of the analysis is the visualization and quantification of rider cancellation rates, attributed to factors such as driver availability and prolonged pickup times. Further, a scatter plot analysis is conducted to determine the optimal number of drivers for maximizing time spent on routes, an essential factor in operational efficiency.

The model is enhanced by integrating a pricing strategy based on riders' perceived value of their journey, modeled as a shifted gamma distribution. This addition allows for a dynamic pricing model that varies with the number of occupied drivers, implementing two distinct pricing tiers. The recalibrated model comprehensively evaluates key performance indicators within this pricing framework, including the critical aspect of hourly revenue. It scrutinizes the impact of pricing changes on ride cancellations and riders' acceptance rates, aiming to balance profitability with service quality. The study provides strategic insights into optimizing pricing and driver allocation to enhance the overall efficacy of the ride-hailing system.

1 Introduction

New York City, as the most densely populated urban area in the United States, presents unique challenges for urban mobility, particularly in the ride-hailing sector using platforms like Lyft and Uber. The efficiency of such a system is influenced by various factors, including the number of drivers, riders' behavior, and the dynamics of ride pricing. This report aims to optimize a simplified ride-sharing model, maximizing profitability while adhering to data constraints and considering riders' sensitivity to pricing. A stochastic model is developed to analyze the arrival rates of riders and the service rates of drivers. This approach includes a thorough exploration of rider traffic within the system, employing a deterministic birth-death process flow. Such modeling allows for an in-depth understanding of the interplay between rider behavior and system efficiency, critical for operational optimization.

Further, the model delves into the intricate relationship between riders' valuation of rides and the pricing strategy. Insights gained from this analysis are crucial for developing pricing strategies that appeal to riders while ensuring profitability for service providers. The report also explores a hypothetical scenario where individual riders' valuations for each ride are known, suggesting that profits could be significantly improved by dynamically adjusting prices based on these valuations. This aspect underscores the potential advantages of personalized pricing strategies, which not only aim to maximize profits but also lead to a more customer-centric approach. Tailoring prices to individual preferences and valuations could revolutionize pricing strategies in the ride-hailing industry, potentially leading to enhanced customer satisfaction and increased revenue for service providers.

2 Analysis: Modeling

The primary objective of this analysis is to ascertain the optimal number of drivers, the most efficient distribution of these drivers across various city regions, the required adjustments to driver allocation during peak and off-peak hours, and the ideal pricing strategy to encourage community-driven ride-sharing efficiency. This encompasses determining the right balance between supply (drivers) and demand (riders) across different areas of the city, identifying key times for strategic driver reallocation to maximize service availability and minimize wait times, and formulating a pricing model that adapts to fluctuating demand while incentivizing driver participation and customer satisfaction.

2.1 Assumptions

Before constructing the model there are a number of assumptions that must be made on the model.

- The area of analysis is NYC, excluding Staten Island. This represents a land area of about 630 square kilometers.
- The model will ignore geography.
- The model will ignore time-of-day effects. The model will apply to a single period within a day.
- The model will ignore deadheading.
- There are 20,000 ride requests per hour. New riders arrive according to a Poisson Process with constant rate $\lambda = 20,000$ rides per hour
- None of the rides are shared.
- Drivers travel in a straight line between their current location and their destination.

- The average ride length is 15 minutes. We assume that the time to the next completion is exponentially distributed
- Drivers drive at a speed of 7.1 miles per hour, which is 0.190439 km/min [Met23].
- Riders will wait at most 7 minutes for pickup before they cancel.
- There are $c = 4600$ drivers.
- There are n riders in the system at any given time. If $n = c$ (all drivers are busy), then new rides are blocked.
- NYC is divided into 30 regions, each of equal land area and call rate. This is to partially account for the regional nature of the city, in that not all drivers can reach all calls when drivers and calls are distributed across the city.
  - Each region has an area a of 21 square kilometers.
  - Each region receives $\lambda = 666.67$ calls per hour.
  - Each region has $c = 153.33$ drivers.
  - We then model the ride-hailing operation within a single region, and when we want to report city-scale results we multiply the results for a single region by the number of regions as appropriate.
- Riders have a (private) random valuation of their ride, V. Riders will only take a ride if their private value V is larger than p, the price for the ride AND their pickup time is 7 minutes or less.
  - t V has a "shifted gamma" distribution with shift \$15, shape parameter 4 and scale parameter 3. V is given by \$15 plus the sum of 4 exponential random variables, each of which has mean \$3
- For the first pricing scenario, we will take the price p to depend on n, the number of busy drivers in each region at the time the call is received. We will assume $p = p_0 = \$20$ when n is $n_0 = 140$or below and $p = p_1 = \$30$ when n is above n_0

2.2 CTMC Birth-Death Model

Given the defined assumptions on the model, a birth-death model is constructed. The model will be constructed for a given region. This can be defined as a Continuous Time Markov Chain (CTMC). Let $X(t)$ be the number of rides at time t. Then, $(X(t) : t \geq 0$ is a CTMC. In particular, it is a birth-death process. We will assume riders arrive with rate λ666.67 calls per hour, if $n < c$, and $\lambda_n = 0$ calls per hour, if $n \geq c$. For the service rate μ_n, the calculation is more complex. $\frac{n}{\mu_n}$ is the mean service time given n riders in the system, which is equal to the mean time to do pickup + the mean time en-route. The n on the numerator is because the time until the next ride to complete is the minimum of n exponential random variables. The mean time en-route is $\tau = 15$ minutes. To get the mean pickup time, we will use Poisson processes.

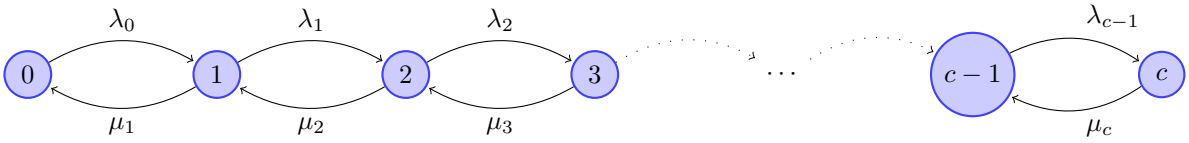
From a past homework in ORIE 4130, we can get the mean pickup time. Free drivers in the ride-sharing system are located in a NYC according to a Poisson process with constant intensity γ, measured in units of drivers per square km. Suppose that a ride request arises at a fixed location within the city. Let T be the random time till the closest free driver reaches the ride request, measured in minutes. From our assumption that drivers travel at a constant speed of 0.19 km/min, we can use a property of poison processes to get the distance to the closest point to a fixed location, which will then be converted to speed. For a Poisson with mean function λ. Let R be the distance to the nearest point. Then, $P(R > r) = e^{-\lambda(C)}$, where C is the disc centered at x with radius r.

Applying this to our model, since the driver travels at 0.19 km/min, the driver must be within a circle of radius $0.19t$ to reach the request in time t. Since the radius of the circle is $0.19t$, the area of the circle is $\pi t^2 0.19^2$. Thus, we have that $P(T > t) = e^{-\gamma\pi t^2 0.19^2}$, where T here is the pickup time. To get the average pickup time, $E[T] = \int_0^\infty P(T > t) = \int_0^\infty e^{-\gamma\pi t^2 0.19^2} \approx \frac{2}{\gamma}$. Where γ is the intensity of drivers. We will think of drivers as points in a poisson process in space. We have c total drivers and n rides in the system. So, the number of free drivers will be $c - n$. These drivers are evenly distributed in an area a in the region. Thus, $\gamma = \frac{c-n+1}{a}$. Finally. $E[T] = \frac{2\sqrt{a}}{\sqrt{c-n}}$

So, we have that $\mu_n = \frac{n}{E[T]+\tau} = \frac{n}{\frac{2\sqrt{a}}{\sqrt{c-n+1}}+\tau}$.

Further, a rider will cancel if $E[T] = \frac{2\sqrt{a}}{\sqrt{c-n+1}} \geq 7$ minutes. Thus, the arrival rate λ_n can be modified to be $\lambda_n = \lambda * P(T > t) = \lambda e^{-\gamma\pi t^2 0.19^2} = \lambda e^{-(c-n)/a\pi t^2 0.19^2}$, where here $t = 7$ minutes is the max pickup time. An extension to pricing will be discussed in section 2.4.

We can then construct a CTMC diagram, with the parameters μ_n being the dynamic service rate, and λ_n being the dynamic arrival rate. The states here are $1, 2, 3, ..., c$, corresponding to the number of riders in the system.



2.3 Performance Measures

In order to evaluate the efficiency of our designed system, there are a number of performance measures to compute. Firstly, we must compute the steady-state probability distribution. This can be computed by setting $\sum_i \pi_i = 1$ and starting with the first transition. We can start by setting $\pi_0 = 1$ and renormalize at the end. We know that the rate into each state must equal the

rate out of each state, so $\pi_0\lambda_0 = \pi_1\mu_1$. Thus, $\pi_{i+1} = \frac{\lambda_i}{\mu_{i+1}}\pi_i$ and $\pi_i = \frac{\mu_{i+1}}{\lambda_i}\pi_i + 1$.

Then, we can easily compute the performance measures of our system.

- Offered Load = λ
- Fraction of Calls completed = $\overline{\lambda} = \sum_{n=0}^{c} \lambda_n \pi_n = f$
- Fraction of calls Canceled = $1 - f$
- Throughput = Offered Load * fraction completed = $\lambda * f$
- Fraction of Driver Time En Route (Drivers want to maximize this) = $\overline{\lambda}\tau/c$
- Total Driver Time En Route per hour = $\overline{\lambda}\tau$
- Utilization of Drivers (En Route or pick up) = Average number of busy drivers / total drivers = $\frac{1}{c}\sum_{n=0}^{c} \pi_n n = \rho$
- Fraction of time idle = $1 - \rho$
- Average Pickup Time (Riders care a lot about this) = total pickup time per call / handled calls per hour = $c($1 − fraction idle - fraction en-route$)/\overline{\lambda} = \frac{(\rho - \frac{\overline{\lambda}\tau}{c})c}{\overline{\lambda}} = \frac{c\rho - \overline{\lambda}\tau}{\overline{\lambda}}$

2.4 Model Analysis and Graphs

Graphical representations and performance metrics derived from the model provide insights into the operational dynamics of the NYC ride-hailing system. The adjusted arrival and service rates, plotted against the number of riders, show the interaction between demand and the system's ability to service it. An equilibrium point is indicated where these rates intersect, suggesting a balanced state of the system.

The steady-state distribution, visualized through a plot, reflects the probabilities of the system being at various levels of rider occupancy. This distribution is crucial for understanding the most likely states of the system at any given time.

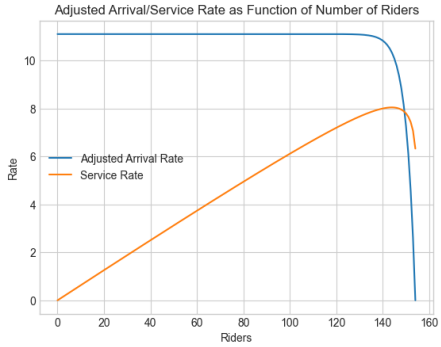


Figure 1: Arrival and Service Rates

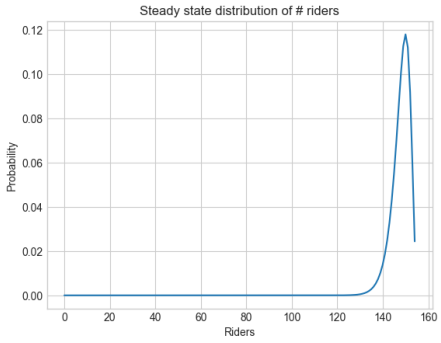


Figure 2: Steady State Distribution

Key performance indicators include the fraction of rides canceled due to the system reaching capacity or riders' unwillingness to wait, and the throughput, which indicates the actual number of rides serviced per unit time. These measures are essential for assessing the efficiency of the system and the utilization of drivers.

By extending these calculations to the entire city, the analysis provides a city-wide perspective on ride-hailing operations, thereby facilitating strategic decisions to enhance system-wide efficiency.

Regional Analysis	Offered demand in region	11.111 rides/min = 666.667 rides/hr
	Fraction of offered demand served	0.7
	Fraction of rides canceled	0.3
	Handled Load (throughput) in region	7.782 rides/min = 466.947 rides/hr
	Fraction of driver time spent en route	0.763
	Total driver time spent en route in region	116.7 min = 7004.2 hr

Driver Utilization	Utilization of Drivers	0.961
	Fraction of driver time spent in pickup	0.198
	Fraction of driver time spent idle	0.039
	Average pickup time	3.9 minutes

City-Wide Analysis	Offered demand in city	333.333 rides/min = 20000 rides/hr
	Handled Load (throughput) in city	233.474 rides/min = 14008.418 rides/hr
	Total driver time spent en route in city	3502.1 min = 210126.3 hr

2.4.1 Maximizing the Driver Time en route

A crucial factor to consider is the optimization of drivers' time spent en route. To explore this, we plot the fraction of time drivers spend en route against both the average number of drivers en route and the total number of drivers. This examination spanned 50 to 10,000 drivers in the city.

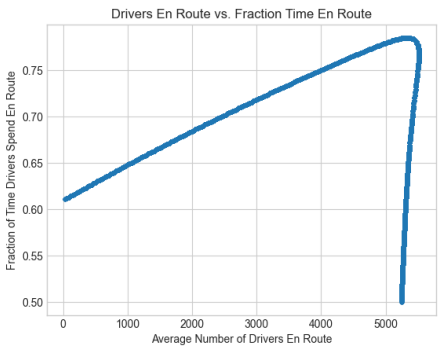


Figure 3: Fraction Time En Route

The plot shown contrasts the average number of drivers en route with the fraction of their time spent en route. This reveals how efficiently drivers are being utilized as their numbers change. This examines how the system manages the balance between available rides and drivers on the road.

A critical question is determining the optimal number of drivers that maximizes the time they spend en route. This is essential for efficiency, ensuring that drivers are engaged in rides, rather waiting for ride requests. The optimal number of drivers, achieving the maximum fraction of time spent en route, is 5,790.

The analysis of the ride-hailing system in New York City reveals a significant gap between the current number of drivers (4,600) and the optimal (5,790). This shortfall manifests in two critical aspects of the system's operation. Firstly, the steady-state distribution skewed towards the right suggests that a substantial portion of the time, a high number of drivers are occupied. This scenario leads to what can be described as a "wild goose chase," where the available drivers are spread thin across the city, resulting in prolonged pickup times. Such inefficiencies not only diminish the service quality but also contribute to rider dissatisfaction due to extended waiting periods. Secondly, the fact that the actual number of drivers is substantially lower than the optimal number further exacerbates these challenges. The system, therefore, operates under a consistent strain, struggling to meet demand efficiently, which could lead to increased cancellations and a potential loss of revenue and customer trust. Addressing this gap by aligning the number of drivers closer to the optimal level identified could significantly enhance operational efficiency, reduce pickup times, and improve overall service quality.

2.5 Pricing and Rider Valuation

Now, the model will be extended to take into account the pricing of rides. Riders have a private random valuation of their ride, V. Riders will only take a ride if their private value V is larger than p, the price for the ride and their pickup time is 7 minutes or less. From the assumptions listed above, V has a "shifted gamma" distribution with shift \$15, shape parameter 4 and scale parameter 3. In other words, V is given by \$15 plus the sum of 4 exponential random variables, each of which has mean \$3. We will take the price p to depend on n, the number of busy drivers in each region at the time the call is received. In particular, we will assume p = p0 = \$20 when n is n0 = 140 or below and p = p1 = \$30 when n is above n0.

The adjusted arrival rate λ_n at state n takes into account both the likelihood of riders accepting the ride based on their valuation and the probability of a ride request being within the acceptable pickup time. The probability that a rider's valuation is greater than the price $p(n)$, denoted as $P(V > p(n))$, is given by the survival function of the shifted gamma distribution: $P(V > p(n)) = 1 - F_{\text{gamma}}(p(n) - 15; \alpha, \theta)$ where F_{gamma} is the cumulative distribution function (CDF) of the gamma distribution. The probability that the pickup time is less than the maximum wait time, denoted as $P(T \leq k)$, where k is the maximum wait time (7 minutes), is approximated based on the system's current state. For simplicity, let's denote this probability as $P_{\text{pickup}}(n)$. The adjusted arrival rate λ_n is then the product of the base arrival rate, $P_{\text{pickup}}(n)$, and $P(V > p(n))$:

$$\lambda_n = \text{arrival rate per region per minute } \times P_{\text{pickup}}(n) \times P(V > p(n))$$

The service rate μ_n is assumed to be dependent on the number of occupied drivers and the mean time to complete a ride. It remains unchanged from the previous model.

The revenue per unit time can be quantified by a formula that integrates the system's state probabilities, arrival rates, and pricing strategy. Specifically, the formula $\sum_n \pi_n \lambda_n \text{price}(n)$ captures the essence of revenue generation in this context. n represents the different states of the system, each corresponding to a specific number of busy drivers. p_n is the steady-state probability of the system being in state n. This probability reflects how frequently the system is expected to find itself with n busy drivers. λ_n denotes the adjusted arrival rate of ride requests in state n, which factors in both the riders' willingness to wait and their acceptance of the ride price. price(n) is a function defining the ride price based on the number of busy drivers, n. Here, price(n) = $p_0$if $n \leq n_0$ And, price(n) = $p_1$if $n > n_0$. where p_0 and p_1 are different pricing tiers, and n_0 is the threshold number of drivers. The revenue per unit time is calculated by summing over all possible states. For each state, the product of the steady-state probability, the adjusted arrival rate, and the price gives the expected revenue contribution from that state. Essentially, the formula assesses how often each state occurs and multiplies this frequency by the revenue generated in that state, thereby aggregating the total expected revenue across all states. This approach effectively captures the dynamic interplay between driver availability, demand, and pricing within the ride-hailing market, providing a comprehensive view of the system's revenue-generating potential.

Similar to before, we can gain insights from graphs and performance metrics. We again show the adjusted arrival and service rates, plotted against the number of riders, to show the interaction between demand and the system's ability to service it. The steady-state distribution and the probability of ride acceptance based on price are also shown.

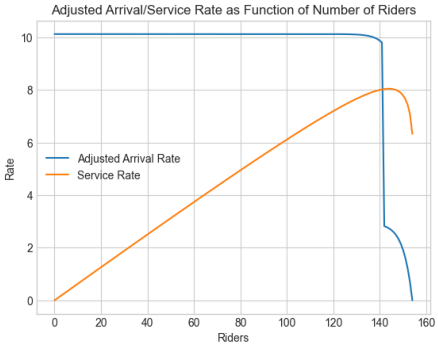


Figure 4: Arrival and Service Rates against the Number of Riders in the System, With pricing

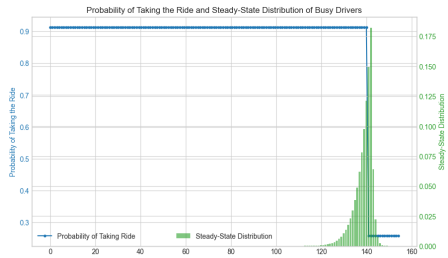


Figure 5: Steady State Distribution of Number of Riders in the system and Probability of Ride Acceptance with pricing

Key performance are shown below:

Regional Analysis	Offered demand in region	11.111 rides/min = 666.667 rides/hr
	Fraction of offered demand served	0.716
	Fraction of rides canceled	0.284
	Handled Load (throughput) in region	7.957 rides per minute = 477.428 rides per hour
	Fraction of driver time spent en route	0.78
	Total driver time spent en route in region	119.4 min = 7161.4 hr
	Revenue per hour	$10,018.35
Driver Utilization	Utilization of Drivers	0.901
	Fraction of driver time spent in pickup	0.121
	Fraction of driver time spent idle	0.099
	Average pickup time	2.3 minutes
City-Wide Analysis	Offered demand in city	333.333 rides/min = 20000 rides/hr
	Handled Load (throughput) in city	238.714 rides/min = 14332.854 rides/hr
	Total driver time spent en route in city	3580.7 min = 214842.8 hr
	Revenue per hour	$300,550.50

2.5.1 Maximizing the Driver Time en route

Similar to the previous model, we can vary the number of drivers in the system to maximize drivers' time spent en route.

The optimal number of drivers that maximizes the time they spend en route is 4470, which is closer to the 4600 we have currently. This indicates that dynamic pricing enhances the model's efficiency. A plot is shown in Figure 6.

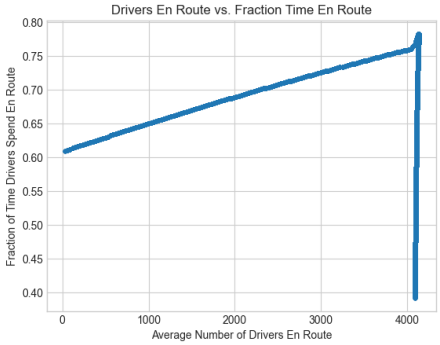


Figure 6: Average number of Drivers En Route vs Fraction Time En Route with pricing

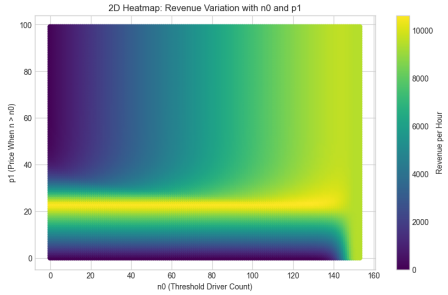


Figure 7: Heatmap Showing Price vs n0 and p1, for pricing optimization

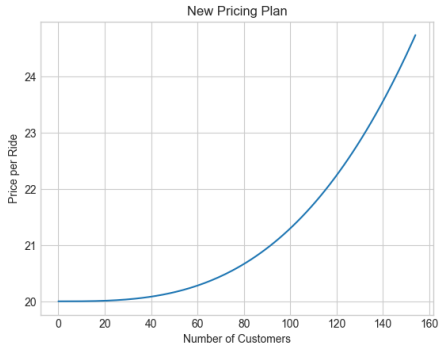


Figure 8: New Dynamic Pricing Model

2.5.2 Optimizing Current Pricing Model

Next, the model was used to come up with a pricing scheme (values of p_1 and n_0) to maximize revenue per hour, taking the base price p_0 as fixed. This was a simple optimization of two parameters, shown in the heatmap in Figure 7. It was important in this optimization to recalculate the arrival rates with each pricing model, due to the effect that prices have on arrival rates. It was found that the optimal p_1 and n_0 were $n_0 = 121$ and $p)1 = 23$. This led to a maximum regional revenue of $10,629.93 per hour

or citywide $318,897.90 hourly. Plots for the new Arrival/service rates and stready state distribution are shown below. We see with these optimized pricing values the steady state distribution has much more time at the higher price through the change of n_0, but because the price p_1 has decreased, riders are more likely to pay.

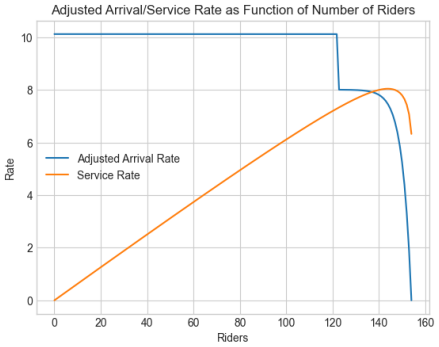


Figure 9: Arrival and Service Rates p1=23 n0=121

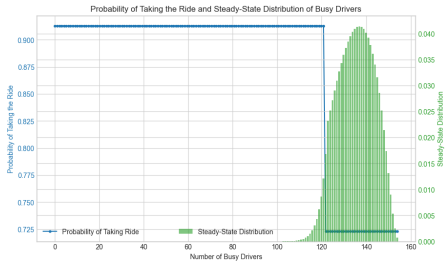


Figure 10: Steady State Distribution p1=23 n0=121

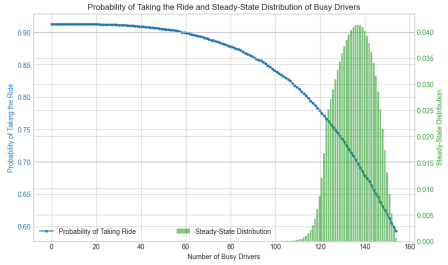


Figure 11: Steady State Distribution for New Model

2.5.3 A New Pricing Model

Further, a new more dynamic pricing model was created. Here, the price of the ride is according to this function depending on the number of riders: $p(n) = n^3/770000 + 20$. This is shown in figure 8 at the top of the page, showing the variation of price vs number of riders. The minimum price is $20 and the price increases as n increases, but the cubic relationship makes the price a bit more dynamic. Here, the revenue was further optimized to $10709.55 per hour per region, or $321,286.50 hourly citywide. The steady state distribution using this new pricing model is shown in Figure 11. See the appendix for the arrival and service rates (very similar to previous rates).

2.5.4 The Value of Learning a Riders' Value

The extension of the current ride-hailing model to incorporate personalized pricing, where the platform can ascertain a rider's valuation (V) for each ride, presents an intriguing opportunity for enhancing revenue and service optimization. If the platform knew the riders' value for every ride, they could offer prices tailored to individual riders, where the prices are slightly less than the riders' valuations. This would potentially maximize profits by capturing more consumer demand.

To assess the value of personalized pricing in a ride-hailing model, it's essential to extend the current model to compare key metrics under standard and personalized pricing scenarios. The model would incorporate a function that simulates personalized pricing based on each rider's known valuation (V). This function would dynamically adjust the price of each ride to approach, but not exceed, the rider's maximum willingness to pay. The model would then analyze the impact of this new pricing on revenue, ride acceptance rates, and overall system efficiency. Key performance indicators such as total revenue, average wait times, and driver earnings would be evaluated under both the personalized pricing and the function-based pricing strategies. It is logical from analysis that personalized pricing would result in higher revenue, because more rides would be accepted

However, in practice, personalized pricing relies heavily on the accurate prediction of each rider's valuation. Implementing such a personalized pricing strategy would require a careful balance, as it could raise concerns among users regarding fairness and privacy. Transparency in how prices are determined and ensuring that the pricing algorithm does not discriminate against certain user groups would be crucial. The extension must also consider the long-term implications on rider loyalty and market dynamics, as personalized pricing could significantly influence user behavior and platform perception. This could lead to a more nuanced, demand-responsive ride-hailing service, potentially setting new standards in the industry for customer-centric pricing strategies.

3 Conclusion

This study of NYC's ride-hailing system has provided insights into optimizing efficiency and profitability. The developed model revealed the significance of managing 4,600 drivers across 30 city zones to balance demand and service capacity effectively. A pivotal finding was identifying the optimal driver count, 5,790 without pricing and 4470 with pricing, to maximize drivers' active time, indicating a current shortfall in the existing driver pool. The integration of a dynamic pricing model, sensitive to riders' valuation following a shifted gamma distribution, led to adjustments in pricing tiers based on driver availability, enhancing both service quality and revenue generation. This study shows understanding rider behavior, optimizing driver allocation, and implementing adaptive pricing strategies are key to ensuring customer satisfaction, reducing wait times, and increasing profitability. The exploration of a more dynamic pricing model further highlights the potential for innovative approaches to meet the evolving demands of urban mobility, positioning ride-hailing services as a critical component in the future of urban transportation.

References

[Hen23] Shane Henderson. Course notes in orie 4130: Service system modeling, 2023. Unpublished course notes, Cornell University.

[Met23] Metropolitan Transportation Authority. Why nyc needs central business district tolling, 2023.

[Ope23] OpenAI. Chatgpt, 2023. Source for code and writing help.

[Wik22] Wikipedia. New york city, 2022. Available: https://en.wikipedia.org/wiki/New_York_City.

Ope23 Hen23 Wik22

Appendix: Ride Hailing in NYC

In this case we want to develop a model to evaluate and potentially modify the ridehailing system in NYC, excluding Staten Island. This represents a land area of about **630 square kilometers**. We will adopt a (greatly) simplified model of operations that **doesn't incorporate geography** (as in the models we built in class), but with additional features that you will develop. Our model also **ignores time-of-day effects**. You can think of this as meaning our model applies to a single period within a day.

The following estimates are loosely based on publicly available data for Uber's operations in NYC. We assume **20,000 ride requests are received per hour**. Assume that *none of these rides are shared*. Assume an **average ride length of 15 minutes**. We will *ignore the need for deadheading* throughout this case. Assume riders will **wait at most 7 minutes for pickup** before they cancel. Assume **4600 drivers**. To partially account for the regional nature of the city, in that not all drivers can reach all calls when drivers and calls are distributed across the city, we adopt a regional model as follows. We assume **NYC is divided into 30 regions, each of equal land area and call rate**. We then *model the ridehailing operation within a single region, and when we want to report city-scale results we multiply the results for a single region by the number of regions as appropriate*.

Compute the equilibrium points (if they exist) and the stationary distribution for a birth-death process that models ride sharing. Based on work by Lobel et al.

Modify the python notebook provided in class so that it handles the situation where riders will cancel at the time of their request if their pickup time is excessive. Plot the resulting arrival rate (that decreases as cancellations increase) and the service rate as a function of n, the number of riders in the system.

traffic speed from https://new.mta.info/project/CBDTP/why-NYC-needs-central-business-district-tolling

and from the HW2Q4 where assuming speed is 0.25 km/min

In [211…]
```
4600/30
```

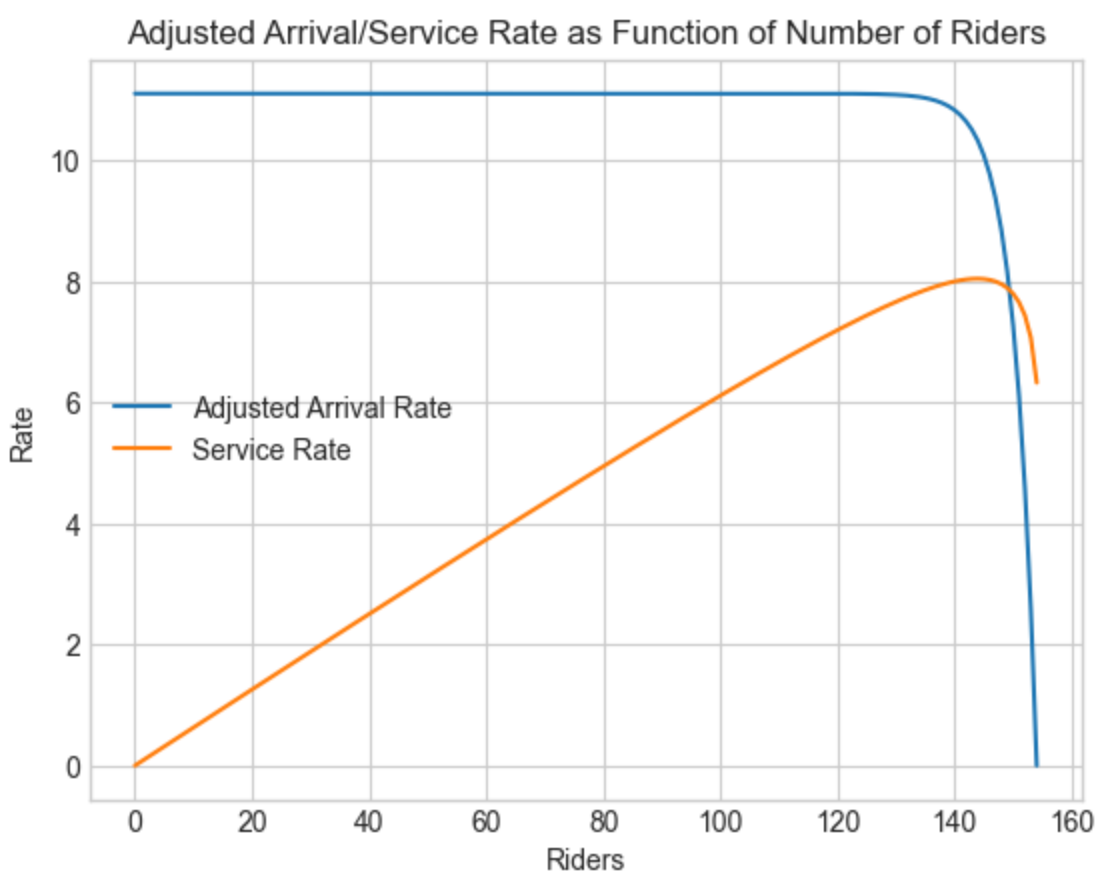
Out[211]:
```
153.33333333333334
```

In [1]:
```
area_nyc = 630 #sq_km
arrival_rate = 20000 #per_hour
arrival_rate_min = arrival_rate/60 #per_minute
drivers = 4600
mean_ride_len = 15 #minutes
max_wait_time = 7 #minutes
regions_nyc = 30
speed = 0.190439 #km/min


num_drivers_region = drivers/regions_nyc
area_region = area_nyc/regions_nyc #sq_km
arrival_rate_region_minute = arrival_rate_min/regions_nyc #people per minute per region
```

cite hw2 Q4

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import math

# Constants
city_area = area_region  # this many square kilometers
c = math.floor(num_drivers_region) # Number of drivers
mean_en_route = mean_ride_len # Mean time spent riding from origin to destination
pickup_base = 2 * math.sqrt(city_area) # Mean time to do pickup in minutes is pickup_base / sqrt
max_n = c + 1  # Explore number of riders n in system ranging from 0 to this number-1. Loss syst

# Adjusted Arrival Rate
def adjusted_arrival_rate(n, c, max_wait_time, pickup_base):
    P_t_greater_k = np.exp( (-(c-n))/city_area * math.pi*(speed**2)*(max_wait_time**2))
    return arrival_rate_region_minute*(1-P_t_greater_k)

# Compute Service Rate
mu = np.zeros(max_n)
adjusted_arrival_rates = np.zeros(max_n)
for n in range(max_n):
    adjusted_arrival_rates[n] = adjusted_arrival_rate(n, c, max_wait_time, pickup_base)
    mu[n] = np.minimum(n, c) / (mean_en_route + pickup_base / math.sqrt(c - n + 1))

# Plotting
plt.figure()
plt.title("Adjusted Arrival/Service Rate as Function of Number of Riders")
plt.xlabel("Riders")
plt.ylabel("Rate")
x_nums = np.linspace(0, max_n, max_n)
plt.plot(x_nums, adjusted_arrival_rates, label='Adjusted Arrival Rate')
plt.plot(x_nums, mu, label='Service Rate')
plt.legend()
plt.show()
```

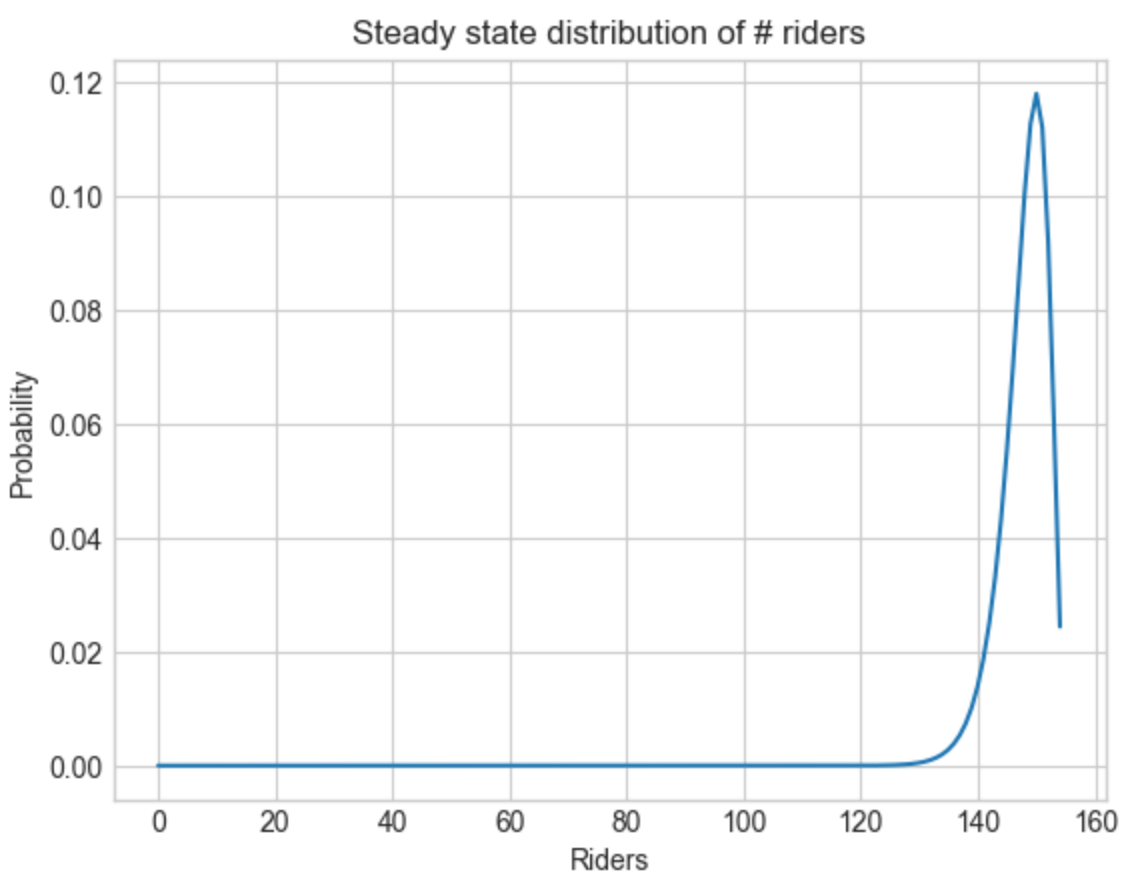
```
C:\Users\julia\anaconda3\envs\engri_1101\lib\site-packages\numpy\_distributor_init.py:32: UserWa
rning: loaded more than 1 DLL from .libs:
C:\Users\julia\anaconda3\envs\engri_1101\lib\site-packages\numpy\.libs\libopenblas.QVLO2T66WEPI7
JZ63PS3HMOHFEY472BC.gfortran-win_amd64.dll
C:\Users\julia\anaconda3\envs\engri_1101\lib\site-packages\numpy\.libs\libopenblas.XWYDX2IKJW2NM
TWSFYNGFUWKQU3LYTCZ.gfortran-win_amd64.dll
  stacklevel=1)
```



Generate plot of the arrival and service rates

Compute steady-state distribution

In [3]:
```
rate_differences = np.abs(adjusted_arrival_rates - mu)
start_index = np.argmin(rate_differences)
# start_index gives the location of the closest match between arrival and service rates.
# Start the recursion for birth-death processes there and work out to edges.
# Numerically sensible.

pi = np.zeros(max_n) # Vector giving the stationary distribution
pi[start_index] = 1 # Temporary value. We rescale to prob distribution later
for i in range(start_index, max_n-1):
    pi[i+1] = pi[i] * adjusted_arrival_rates[i] / mu[i+1]
for i in range(start_index, 0, -1):
    pi[i-1] = pi[i] * mu[i] / adjusted_arrival_rates[i-1]
sum_pi = pi.sum()
pi = pi / sum_pi
#print(pi)
plt.plot(x_nums, pi)
plt.title("Steady state distribution of # riders")
plt.xlabel("Riders")
plt.ylabel("Probability")
plt.show()
```



Compute performance measures

In addition to the other performance measures computed in the python notebook, compute the fraction of riders who cancel. Assume that riders who are blocked due to all drivers being busy are classified as cancellations.

In [4]:
```
offered_load = arrival_rate_region_minute
print(f'Offered demand in region = {offered_load:.3f} rides per minute')

print(f'Offered demand in region = {offered_load*60:.3f} rides per hour')
print(' ')
fraction_handled = 1/offered_load*(pi@adjusted_arrival_rates)
print(f'Fraction of offered demand served = {fraction_handled:.3f}')
print(f'Fraction of rides canceled = {1-fraction_handled:.3f}')
print(' ')
throughput = offered_load*fraction_handled
print(f'Handled Load (throughput) in region = {throughput:.3f} rides per minute')
print(f'Handled Load (throughput) in region= {throughput*60:.3f} rides per hour')
print(' ')
fraction_enroute = throughput * mean_en_route / c # fraction of driver time en_route
print('Fraction of driver time spent en_route = %.3f' %fraction_enroute)
print('Total driver time spent en route per minute in region = %.1f' %(fraction_enroute * c))
print('Total driver time spent en route per hour in region = %.1f' %(fraction_enroute*60 * c))
print(' ')

busy_drivers = 0
for i in range(max_n):
    busy_drivers += pi[i] * np.minimum(i, c)
fraction_busy = busy_drivers / c
fraction_idle = 1-fraction_busy
fraction_pickup = fraction_busy - fraction_enroute
print('Utilzation of Drivers = %.3f' %fraction_busy)
print('Fraction of driver time spent in pickup = %.3f' %fraction_pickup)
print('Fraction of driver time spent idle = %.3f' %fraction_idle)
```

```
average_pickup_time = c * ( fraction_busy - fraction_enroute ) / throughput
print('Average pickup time = %.1f minutes' %average_pickup_time)


print(f'Offered demand in city = {offered_load*30:.3f} rides per minute')

print(f'Offered demand in city = {offered_load*30*60:.3f} rides per hour')
print(' ')

print(f'Handled Load (throughput) in city = {throughput*30:.3f} rides per minute')
print(f'Handled Load (throughput) in city= {throughput*60*30:.3f} rides per hour')
print(' ')
print('Total driver time spent en route per minute in city = %.1f' %(fraction_enroute*30 * c))
print('Total driver time spent en route per hour in city = %.1f' %(fraction_enroute*60*30 * c))
print(' ')
```

```
Offered demand in region = 11.111 rides per minute
Offered demand in region = 666.667 rides per hour

Fraction of offered demand served = 0.700
Fraction of rides canceled = 0.300

Handled Load (throughput) in region = 7.782 rides per minute
Handled Load (throughput) in region= 466.947 rides per hour

Fraction of driver time spent en_route = 0.763
Total driver time spent en route per minute in region = 116.7
Total driver time spent en route per hour in region = 7004.2

Utilzation of Drivers = 0.961
Fraction of driver time spent in pickup = 0.198
Fraction of driver time spent idle = 0.039
Average pickup time = 3.9 minutes
Offered demand in city = 333.333 rides per minute
Offered demand in city = 20000.000 rides per hour

Handled Load (throughput) in city = 233.474 rides per minute
Handled Load (throughput) in city= 14008.418 rides per hour

Total driver time spent en route per minute in city = 3502.1
Total driver time spent en route per hour in city = 210126.3
```

Plot a scatter plot of the fraction of time drivers spend en route versus the average number of drivers en route over a wide range of total drivers, where each point in the scatter plot corresponds to a different number of drivers. (For this question and the next you should vary the number of drivers.)

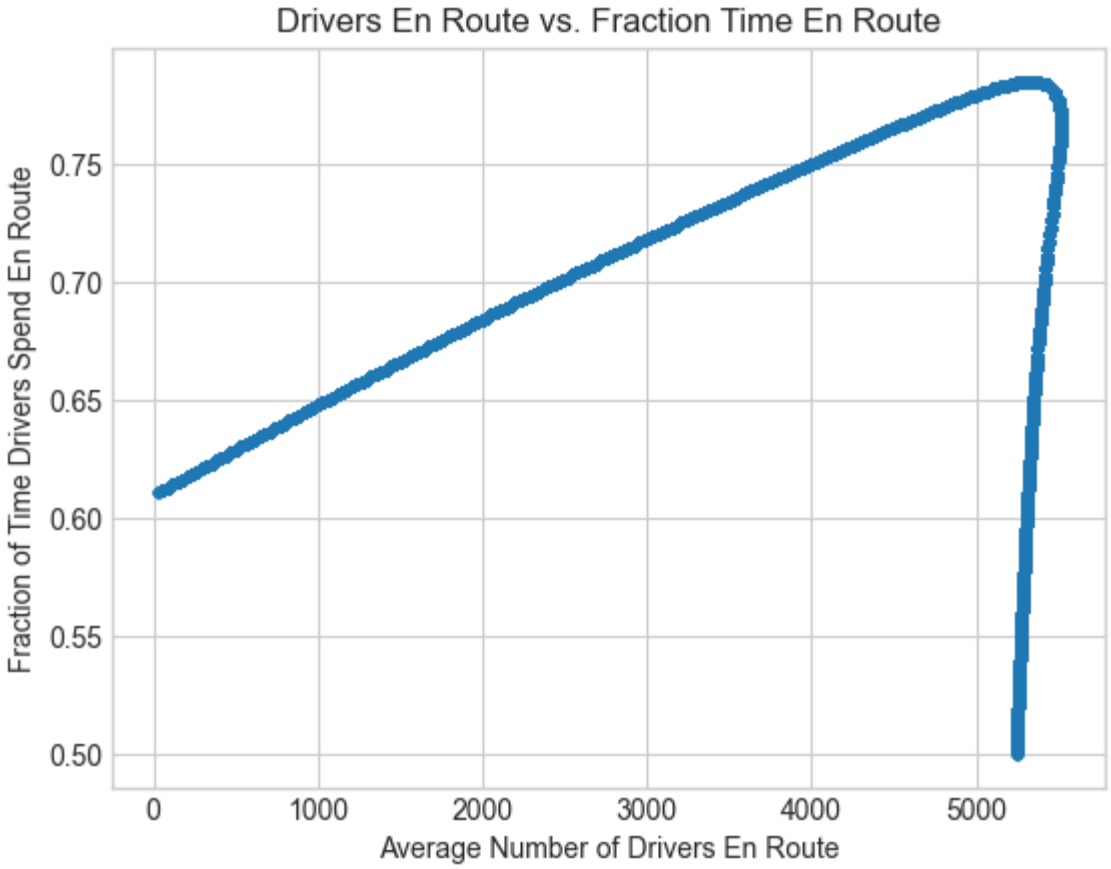
In [5]:

```
drivers_in_route = []
fraction_time_enroute = []
for i in range(50,10000):

    area_nyc = 630 #sq_km
    arrival_rate = 20000 #per_hour
    arrival_rate_min = arrival_rate/60 #per_minute
    drivers = i
    mean_ride_len = 15 #minutes
    max_wait_time = 7 #minutes
    regions_nyc = 30
    speed = 0.190439 #km/min


    num_drivers_region = drivers/regions_nyc
    area_region = area_nyc/regions_nyc #sq_km
```

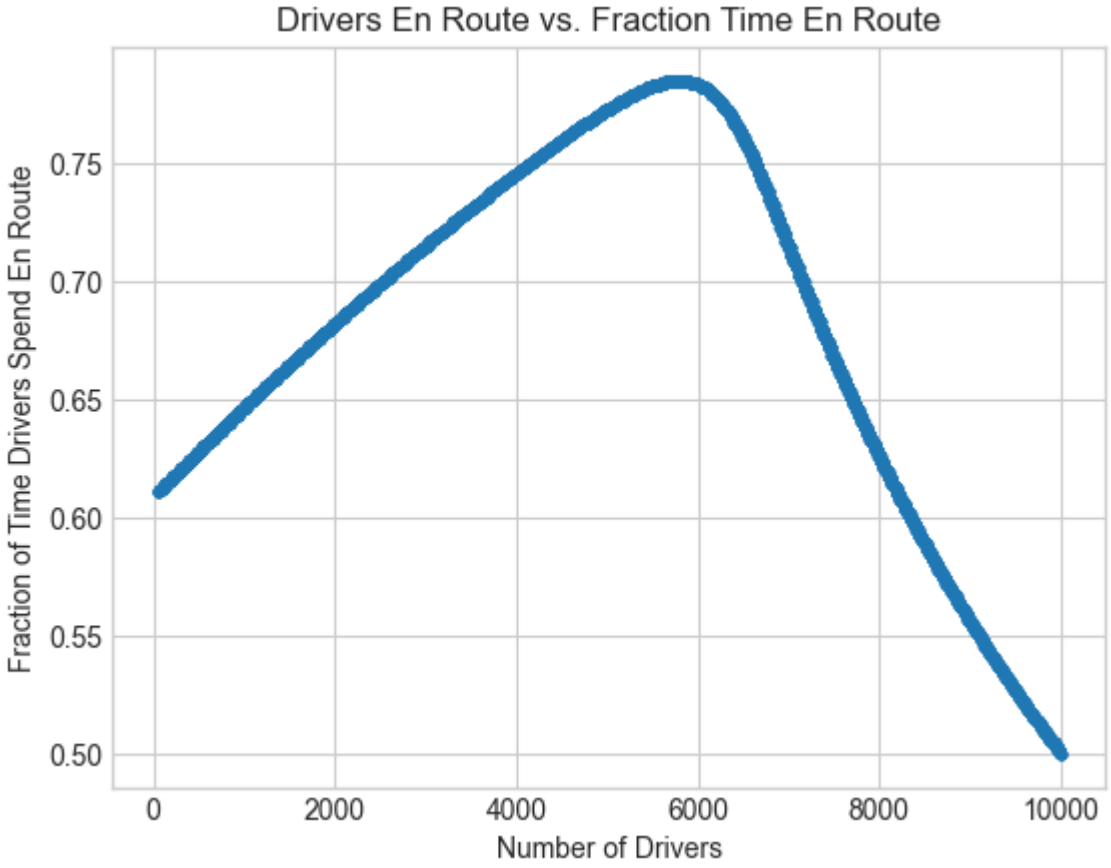
```
    arrival_rate_region_minute = arrival_rate_min/regions_nyc #people per minute per region

    # Constants
    city_area = area_region  # this many square kilometers
    c = math.floor(num_drivers_region) # Number of drivers
    mean_en_route = mean_ride_len # Mean time spent riding from origin to destination
    pickup_base = 2 * math.sqrt(city_area) # Mean time to do pickup in minutes is pickup_base /
    max_n = c + 1  # Explore number of riders n in system ranging from 0 to this number-1. Loss

    # Adjusted Arrival Rate
    def adjusted_arrival_rate(n, c, max_wait_time, pickup_base):
        P_t_greater_k = np.exp( (-(c-n))/city_area * math.pi*(speed**2)*(max_wait_time**2))
        return arrival_rate_region_minute*(1-P_t_greater_k)

    # Compute Service Rate
    mu = np.zeros(max_n)
    adjusted_arrival_rates = np.zeros(max_n)
    for n in range(max_n):
        adjusted_arrival_rates[n] = adjusted_arrival_rate(n, c, max_wait_time, pickup_base)
        mu[n] = np.minimum(n, c) / (mean_en_route + pickup_base / math.sqrt(c - n + 1))

    rate_differences = np.abs(adjusted_arrival_rates - mu)
    start_index = np.argmin(rate_differences)
    # start_index gives the location of the closest match between arrival and service rates.
    # Start the recursion for birth-death processes there and work out to edges.
    # Numerically sensible.

    pi = np.zeros(max_n) # Vector giving the stationary distribution
    pi[start_index] = 1 # Temporary value. We rescale to prob distribution later
    for i in range(start_index, max_n-1):
        pi[i+1] = pi[i] * adjusted_arrival_rates[i] / mu[i+1]
    for i in range(start_index, 0, -1):
        pi[i-1] = pi[i] * mu[i] / adjusted_arrival_rates[i-1]
    sum_pi = pi.sum()
    pi = pi / sum_pi



    offered_load = arrival_rate_region_minute
    fraction_handled = 1/offered_load*(pi@adjusted_arrival_rates)
    throughput = offered_load*fraction_handled
    fraction_enroute = throughput * mean_en_route / c # fraction of driver time en_route
    average_drivers_en_route = 0
    # Loop through each state of the system
    for i in range(max_n):
        # The number of drivers busy with riders is the minimum of i and c
        drivers_busy_with_riders = min(i, c)
        # Accumulate the weighted number of drivers busy with riders
        average_drivers_en_route += (pi[i] * drivers_busy_with_riders)*30
    drivers_in_route.append(average_drivers_en_route)
    fraction_time_enroute.append(fraction_enroute)
```

In [6]:

```
plt.plot(drivers_in_route, fraction_time_enroute,'.')
plt.xlabel('Average Number of Drivers En Route')
plt.ylabel('Fraction of Time Drivers Spend En Route')
plt.title('Drivers En Route vs. Fraction Time En Route')
plt.show()
```



In [7]:
```
plt.plot(range(50,10000), fraction_time_enroute,'.')
plt.xlabel('Number of Drivers')
plt.ylabel('Fraction of Time Drivers Spend En Route')
plt.title('Drivers En Route vs. Fraction Time En Route')
plt.show()
```



Approximately what number of drivers maximizes the fraction of time that drivers spend en route?

In [8]:
```
index = np.argmax(fraction_time_enroute)
range(50,10000)[index]
```

Out[8]:
```
5790
```

Now suppose that in addition to cancellation, riders have a (private) random valuation of their ride, V say. Riders will only take a ride if their private value V is larger than p, the price for the ride AND their pickup time is 7 minutes or less. Assume that V has a "shifted gamma" distribution with shift 15, shape parameter 4 and scale parameter 3. In other words, V is given by 15 plus the sum of 4 exponential random variables, each of which has mean 3. We will take the price p to depend on n, the number of busy drivers in each region at the time the call is received. In particular, we will assume p = p0 = 20 when n is n0 = 140 or below and p = p1 = 30 when n is above n0. This setting can be modeled by changing the arrival rates ln ; **explain in your report how they are now computed**. Modify your python notebook to compute all performance measures in this new setting. In addition to all previous performance measures, compute the revenue received per hour. Explain why the revenue per unit time is given by ån pn ln price(n), for a suitably defined function (you should define it) "price" that is a function of n. Use your model to provide new answers to Questions 1 through 4 above. (For Question 2 where we ask you to report the cancellation fraction, instead report one number giving the fraction of rides that are not taken. Rides are not taken because either the rider cancels or the rider thinks the price is too high.)

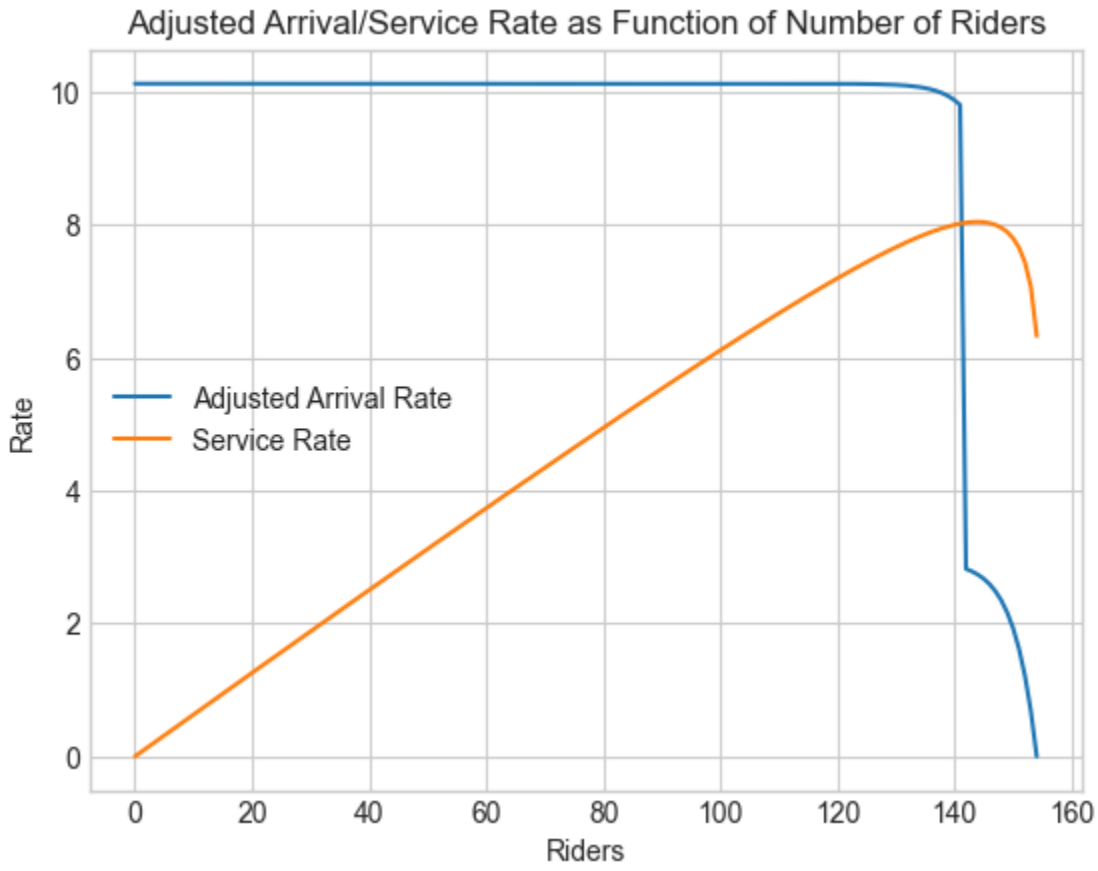
In [150…
```
def price(n, n0=140, p0=20, p1=30):
    return p0 if n <= n0 else p1
```

In [151…
```
area_nyc = 630 #sq_km
arrival_rate = 20000 #per_hour
arrival_rate_min = arrival_rate/60 #per_minute
drivers = 4600
mean_ride_len = 15 #minutes
max_wait_time = 7 #minutes
regions_nyc = 30
speed = 0.190439 #km/min


num_drivers_region = drivers/regions_nyc
area_region = area_nyc/regions_nyc #sq_km
arrival_rate_region_minute = arrival_rate_min/regions_nyc #people per minute per region
```

cite hw2 Q4

In [152…
```
import scipy.stats

# Constants
city_area = area_region  # this many square kilometers
c = math.floor(num_drivers_region) # Number of drivers
mean_en_route = mean_ride_len # Mean time spent riding from origin to destination
pickup_base = 2 * math.sqrt(city_area) # Mean time to do pickup in minutes is pickup_base / sqrt
max_n = c + 1  # Explore number of riders n in system ranging from 0 to this number-1. Loss syst

# Adjusted Arrival Rate

def adjusted_arrival_rate(n, c, max_wait_time, pickup_base, n0=140, p0=20, p1=30):
    # Probability that pickup time is greater than max_wait_time
    P_t_greater_k = np.exp( (-(c-n))/city_area * math.pi*(speed**2)*(max_wait_time**2))
```

```
    # Probability that a rider's valuation is higher than the price
    current_price = price(n, n0, p0, p1)
    P_v_greater_p = 1 - scipy.stats.gamma.cdf(current_price, a=4, scale=3, loc=15)
    return arrival_rate_region_minute*(1-P_t_greater_k)*P_v_greater_p

# Compute Service Rate
mu = np.zeros(max_n)
adjusted_arrival_rates = np.zeros(max_n)
for n in range(max_n):
    adjusted_arrival_rates[n] = adjusted_arrival_rate(n, c, max_wait_time, pickup_base)
    mu[n] = np.minimum(n, c) / (mean_en_route + pickup_base / math.sqrt(c - n + 1))

# Plotting
plt.figure()
plt.title("Adjusted Arrival/Service Rate as Function of Number of Riders")
plt.xlabel("Riders")
plt.ylabel("Rate")
x_nums = np.linspace(0, max_n, max_n)
plt.plot(x_nums, adjusted_arrival_rates, label='Adjusted Arrival Rate')
plt.plot(x_nums, mu, label='Service Rate')
plt.legend()
plt.show()
```



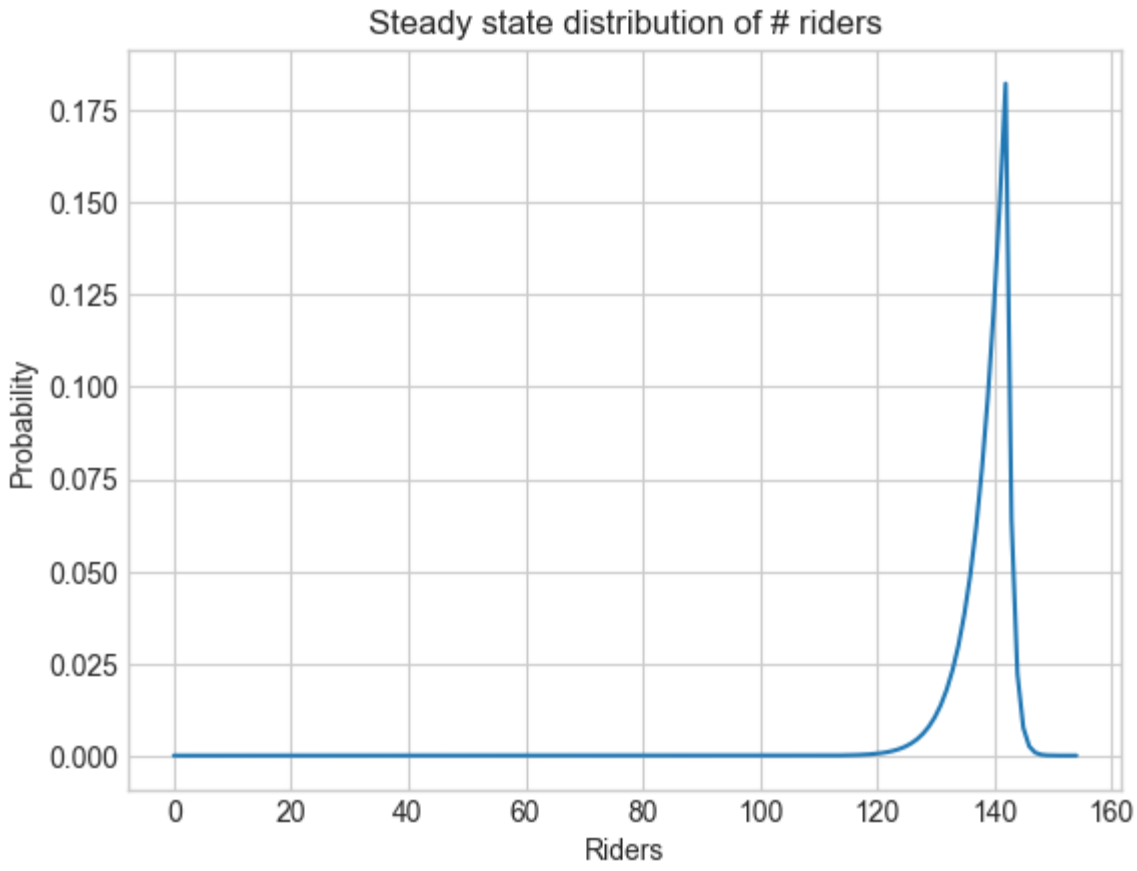
Generate plot of the arrival and service rates

Compute steady-state distribution

In [153…

```
rate_differences = np.abs(adjusted_arrival_rates - mu)
start_index = np.argmin(rate_differences)
# start_index gives the location of the closest match between arrival and service rates.
# Start the recursion for birth-death processes there and work out to edges.
# Numerically sensible.

pi = np.zeros(max_n) # Vector giving the stationary distribution
pi[start_index] = 1 # Temporary value. We rescale to prob distribution later
```

```
for i in range(start_index, max_n-1):
    pi[i+1] = pi[i] * adjusted_arrival_rates[i] / mu[i+1]
for i in range(start_index, 0, -1):
    pi[i-1] = pi[i] * mu[i] / adjusted_arrival_rates[i-1]
sum_pi = pi.sum()
pi = pi / sum_pi
#print(pi)
plt.plot(x_nums, pi)
plt.title("Steady state distribution of # riders")
plt.xlabel("Riders")
plt.ylabel("Probability")
plt.show()
```



Compute performance measures

In [154…

```
offered_load = arrival_rate_region_minute
print(f'Offered demand in region = {offered_load:.3f} rides per minute')

print(f'Offered demand in region = {offered_load*60:.3f} rides per hour')
print(' ')
fraction_handled = 1/offered_load*np.sum(pi*adjusted_arrival_rates)
print(f'Fraction of offered demand served = {fraction_handled:.3f}')
print(f'Fraction of rides not taken = {1-fraction_handled:.3f}')
print(' ')
throughput = offered_load*fraction_handled
print(f'Handled Load (throughput) in region = {throughput:.3f} rides per minute')
print(f'Handled Load (throughput) in region= {throughput*60:.3f} rides per hour')
print(' ')
fraction_enroute = throughput * mean_en_route / c # fraction of driver time en_route
print('Fraction of driver time spent en_route = %.3f' %fraction_enroute)
print('Total driver time spent en route per minute in region = %.1f' %(fraction_enroute * c))
print('Total driver time spent en route per hour in region = %.1f' %(fraction_enroute*60 * c))
print(' ')

busy_drivers = 0
for i in range(max_n):
```

```
    busy_drivers += pi[i] * np.minimum(i, c)
fraction_busy = busy_drivers / c
fraction_idle = 1-fraction_busy
fraction_pickup = fraction_busy - fraction_enroute
print('Utilzation of Drivers = %.3f' %fraction_busy)
print('Fraction of driver time spent in pickup = %.3f' %fraction_pickup)
print('Fraction of driver time spent idle = %.3f' %fraction_idle)

average_pickup_time = c * ( fraction_busy - fraction_enroute ) / throughput
print('Average pickup time = %.1f minutes' %average_pickup_time)


print(f'Offered demand in city = {offered_load*30:.3f} rides per minute')

print(f'Offered demand in city = {offered_load*30*60:.3f} rides per hour')
print(' ')

print(f'Handled Load (throughput) in city = {throughput*30:.3f} rides per minute')
print(f'Handled Load (throughput) in city= {throughput*60*30:.3f} rides per hour')
print(' ')
print('Total driver time spent en route per minute in city = %.1f' %(fraction_enroute*30 * c))
print('Total driver time spent en route per hour in city = %.1f' %(fraction_enroute*60*30 * c))
print(' ')
```

```
Offered demand in region = 11.111 rides per minute
Offered demand in region = 666.667 rides per hour

Fraction of offered demand served = 0.716
Fraction of rides not taken = 0.284

Handled Load (throughput) in region = 7.957 rides per minute
Handled Load (throughput) in region= 477.428 rides per hour

Fraction of driver time spent en_route = 0.780
Total driver time spent en route per minute in region = 119.4
Total driver time spent en route per hour in region = 7161.4

Utilzation of Drivers = 0.901
Fraction of driver time spent in pickup = 0.121
Fraction of driver time spent idle = 0.099
Average pickup time = 2.3 minutes
Offered demand in city = 333.333 rides per minute
Offered demand in city = 20000.000 rides per hour

Handled Load (throughput) in city = 238.714 rides per minute
Handled Load (throughput) in city= 14322.854 rides per hour

Total driver time spent en route per minute in city = 3580.7
Total driver time spent en route per hour in city = 214842.8
```

In [155…

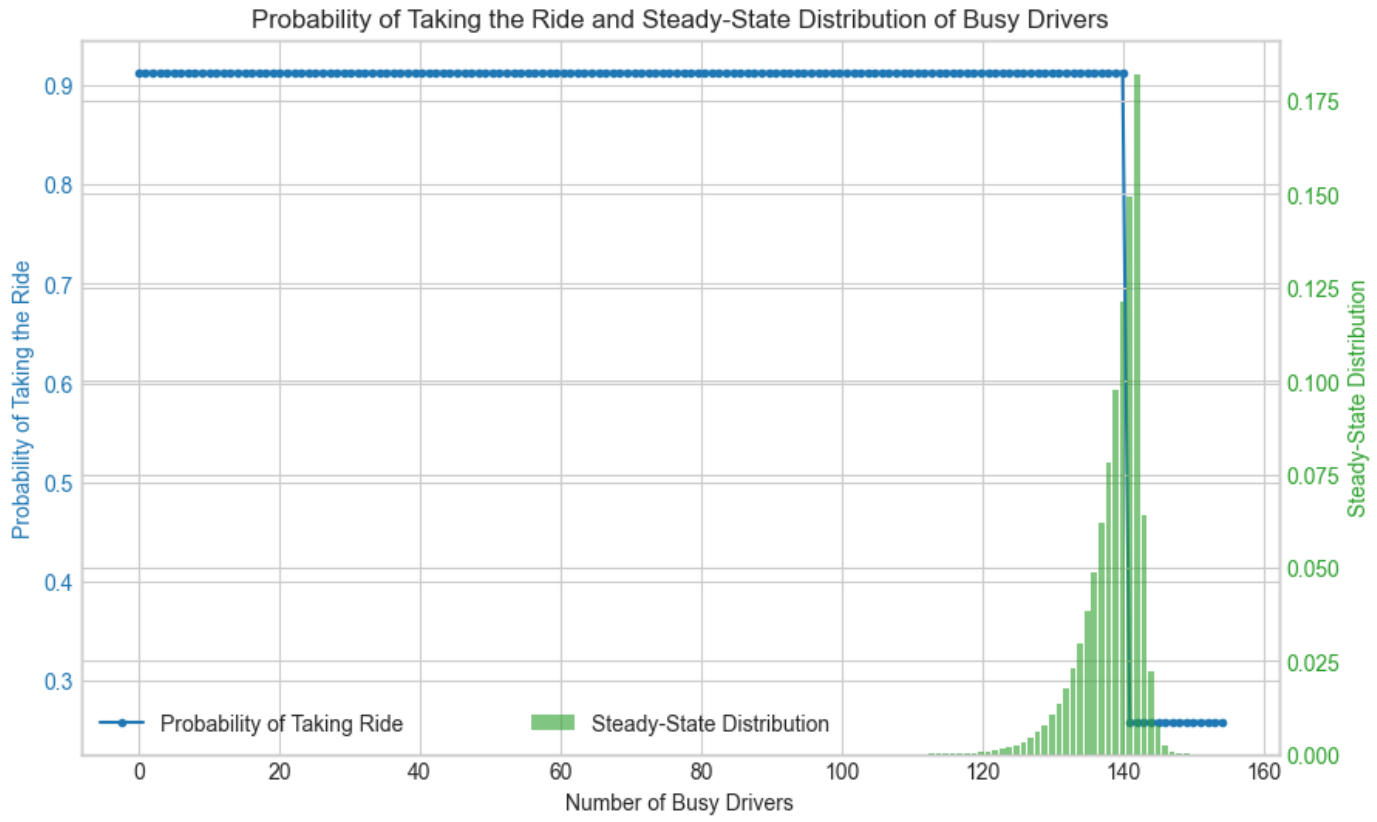
```
def compute_revenue(pi, n0, p0, p1):
    revenue = 0
    for n in range(len(pi)):
        revenue += price(n, n0, p0, p1) * pi[n] * adjusted_arrival_rates[n]
    return revenue * 60  # Convert to per hour

revenue_per_hour = compute_revenue(pi, 140, 20, 30)
print(f'Revenue per hour: {revenue_per_hour:.2f}')
```

```
Revenue per hour: 10018.35
```

In [171…

```
#chat GPT aid in plotting
# Pricing model parameters
p0 = 20  # price when number of busy drivers is 140 or below
```

```
p1 = 30  # price when number of busy drivers is above 140
n0 = 140



# Determining the price based on the number of busy drivers
prices_based_on_busy_drivers = np.where(x_nums <= n0, p0, p1)

# Calculating the probability for each scenario based on the number of busy drivers
probabilities_based_on_busy_drivers = [probability_take_ride(price, valuations) for price in pri

# Creating a two-y-axis plot
fig, ax1 = plt.subplots(figsize=(10, 6))

# Plotting the probability that rider takes the ride (left y-axis)
color = 'tab:blue'
ax1.set_xlabel('Number of Busy Drivers')
ax1.set_ylabel('Probability of Taking the Ride', color=color)
ax1.plot(x_nums, probabilities_based_on_busy_drivers, color=color, marker='.')
ax1.tick_params(axis='y', labelcolor=color)

# Creating a second y-axis for the steady-state distribution
ax2 = ax1.twinx()
color = 'tab:green'
ax2.set_ylabel('Steady-State Distribution', color=color)
ax2.bar(x_nums, pi, color=color, alpha=0.6)
ax2.tick_params(axis='y', labelcolor=color)

# Adding a legend and grid
ax1.legend(['Probability of Taking Ride'], loc='lower left')
ax2.legend(['Steady-State Distribution'], loc='lower center')
ax1.grid(True)

plt.title('Probability of Taking the Ride and Steady-State Distribution of Busy Drivers')
plt.show()
```



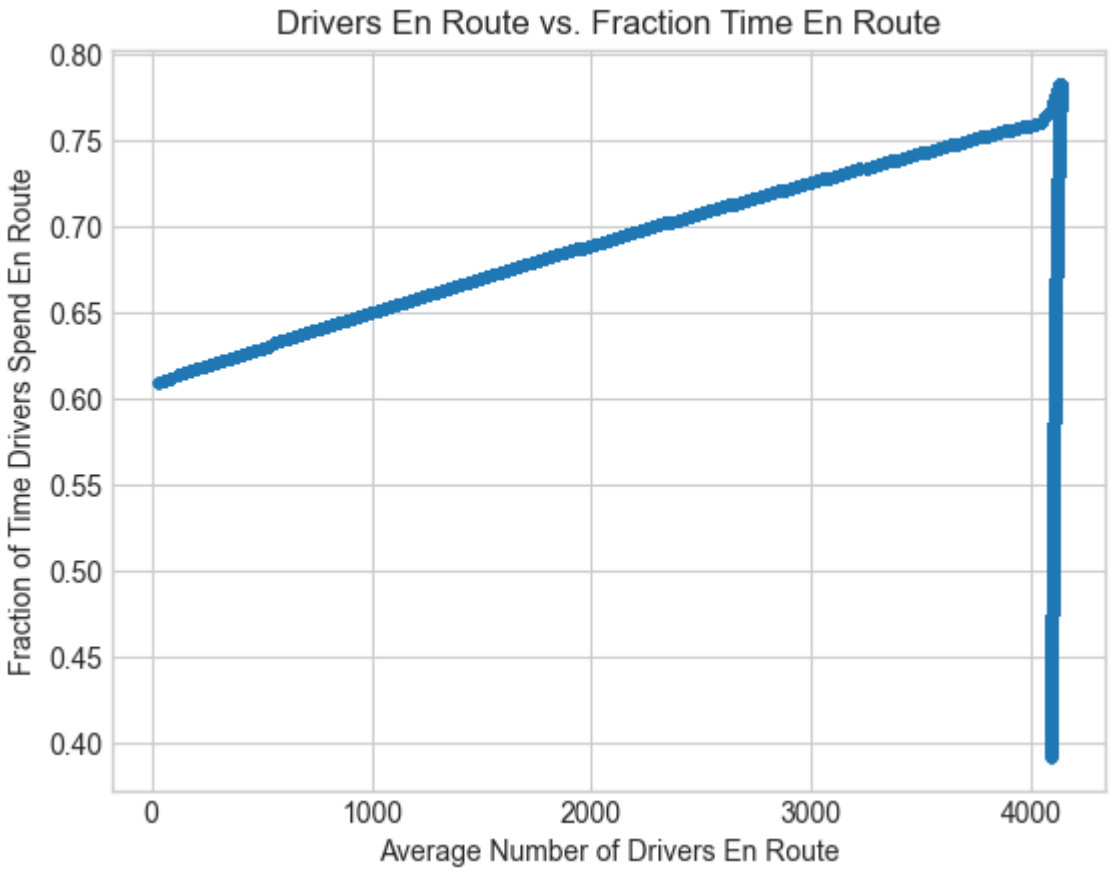
In [15]:

```
drivers_in_route = []
fraction_time_enroute = []
```

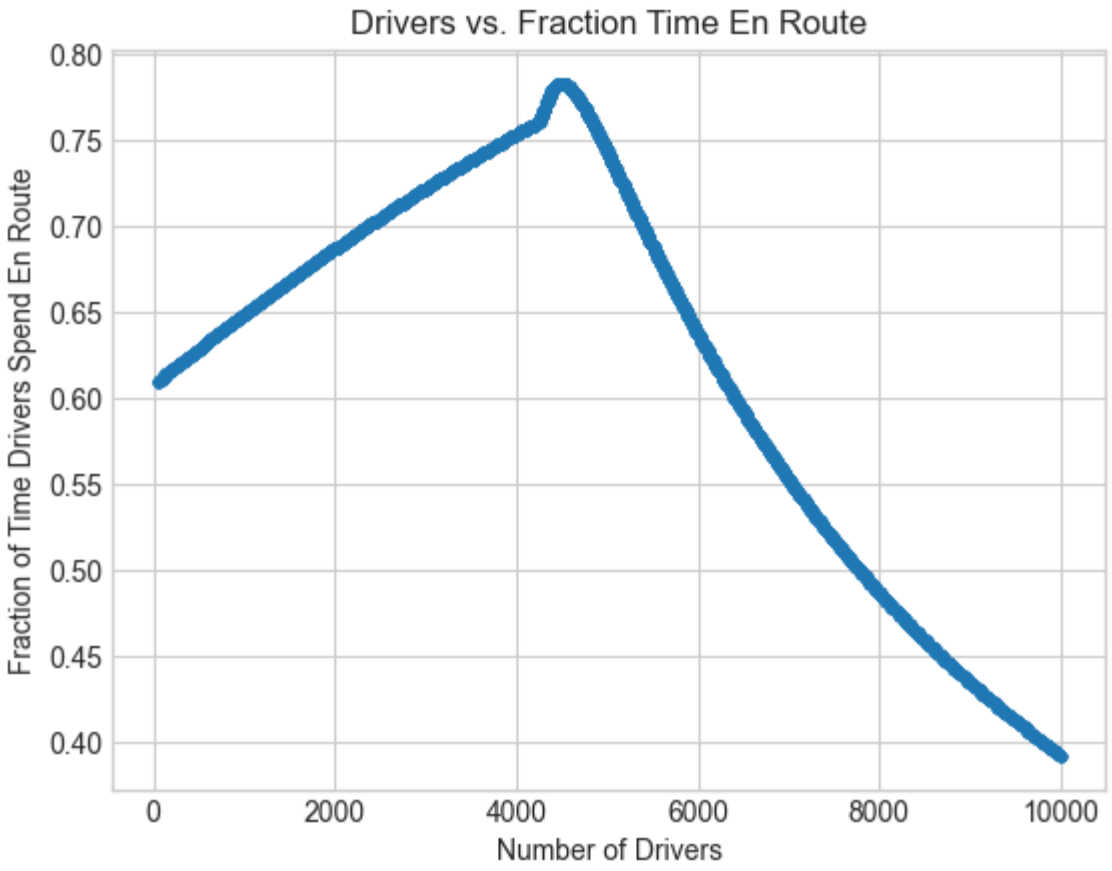
```
revenue = []
for i in range(50,10000):

    area_nyc = 630 #sq_km
    arrival_rate = 20000 #per_hour
    arrival_rate_min = arrival_rate/60 #per_minute
    drivers = i
    mean_ride_len = 15 #minutes
    max_wait_time = 7 #minutes
    regions_nyc = 30
    speed = 0.190439 #km/min


    num_drivers_region = drivers/regions_nyc
    area_region = area_nyc/regions_nyc #sq_km
    arrival_rate_region_minute = arrival_rate_min/regions_nyc #people per minute per region

    # Constants
    city_area = area_region  # this many square kilometers
    c = math.floor(num_drivers_region) # Number of drivers
    mean_en_route = mean_ride_len # Mean time spent riding from origin to destination
    pickup_base = 2 * math.sqrt(city_area) # Mean time to do pickup in minutes is pickup_base /
    max_n = c + 1  # Explore number of riders n in system ranging from 0 to this number-1. Loss

    # Adjusted Arrival Rate

    def adjusted_arrival_rate(n, c, max_wait_time, pickup_base, n0=140, p0=20, p1=30):
        # Probability that pickup time is greater than max_wait_time
        P_t_greater_k = np.exp( (-(c-n))/city_area * math.pi*(speed**2)*(max_wait_time**2))
        # Probability that a rider's valuation is higher than the price
        current_price = price(n, n0, p0, p1)
        P_v_greater_p = 1 - scipy.stats.gamma.cdf(current_price, a=4, scale=3, loc=15)
        return arrival_rate_region_minute*(1-P_t_greater_k)*P_v_greater_p

    # Compute Service Rate
    mu = np.zeros(max_n)
    adjusted_arrival_rates = np.zeros(max_n)
    for n in range(max_n):
        adjusted_arrival_rates[n] = adjusted_arrival_rate(n, c, max_wait_time, pickup_base)
        mu[n] = np.minimum(n, c) / (mean_en_route + pickup_base / math.sqrt(c - n + 1))

    rate_differences = np.abs(adjusted_arrival_rates - mu)
    start_index = np.argmin(rate_differences)
    # start_index gives the location of the closest match between arrival and service rates.
    # Start the recursion for birth-death processes there and work out to edges.
    # Numerically sensible.

    pi = np.zeros(max_n) # Vector giving the stationary distribution
    pi[start_index] = 1 # Temporary value. We rescale to prob distribution later
    for i in range(start_index, max_n-1):
        pi[i+1] = pi[i] * adjusted_arrival_rates[i] / mu[i+1]
    for i in range(start_index, 0, -1):
        pi[i-1] = pi[i] * mu[i] / adjusted_arrival_rates[i-1]
    sum_pi = pi.sum()
    pi = pi / sum_pi



    offered_load = arrival_rate_region_minute
    fraction_handled = 1/offered_load*(pi@adjusted_arrival_rates)
    throughput = offered_load*fraction_handled
    fraction_enroute = throughput * mean_en_route / c # fraction of driver time en_route
    average_drivers_en_route = 0
    # Loop through each state of the system
```

```
    for i in range(max_n):
        # The number of drivers busy with riders is the minimum of i and c
        drivers_busy_with_riders = min(i, c)
        # Accumulate the weighted number of drivers busy with riders
        average_drivers_en_route += (pi[i] * drivers_busy_with_riders)*30
    drivers_in_route.append(average_drivers_en_route)
    revenue_per_hour = compute_revenue(pi, 140, 20, 30)
    revenue.append(revenue_per_hour)
    fraction_time_enroute.append(fraction_enroute)
```

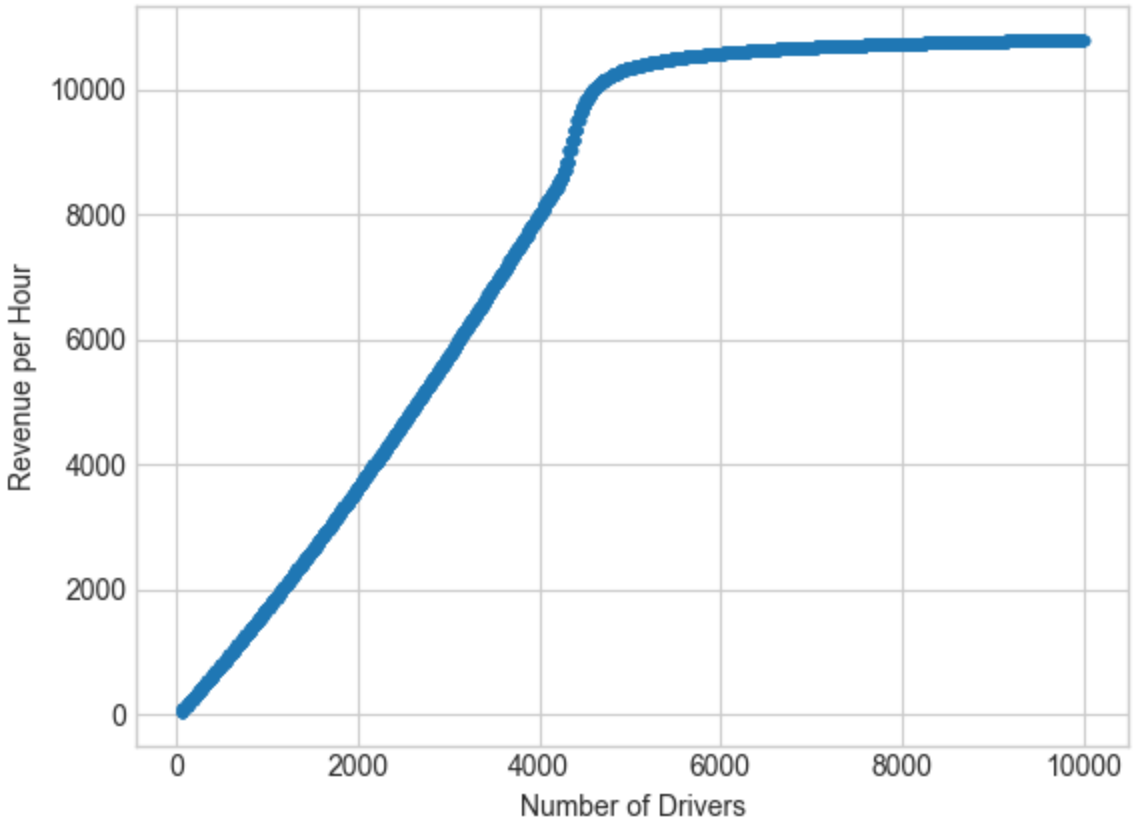
In [16]:
```
plt.plot(drivers_in_route, fraction_time_enroute,'.')
plt.xlabel('Average Number of Drivers En Route')
plt.ylabel('Fraction of Time Drivers Spend En Route')
plt.title('Drivers En Route vs. Fraction Time En Route')
plt.show()
```



In [17]:
```
plt.plot(range(50,10000), fraction_time_enroute,'.')
plt.xlabel('Number of Drivers')
plt.ylabel('Fraction of Time Drivers Spend En Route')
plt.title('Drivers vs. Fraction Time En Route')
plt.show()
```



In [18]:
```
plt.plot(range(50,10000), revenue,'.')
plt.xlabel('Number of Drivers')
plt.ylabel('Revenue per Hour')
plt.show()
```



Approximately what number of drivers maximizes the fraction of time that drivers spend en route?

In [19]:
```
index = np.argmax(fraction_time_enroute)
range(50,10000)[index]
```

Out[19]:
```
4470
```

In [20]:
```
area_nyc = 630 #sq_km
arrival_rate = 20000 #per_hour
arrival_rate_min = arrival_rate/60 #per_minute
drivers = 4600
mean_ride_len = 15 #minutes
max_wait_time = 7 #minutes
regions_nyc = 30
speed = 0.190439 #km/min


num_drivers_region = drivers/regions_nyc
area_region = area_nyc/regions_nyc #sq_km
arrival_rate_region_minute = arrival_rate_min/regions_nyc #people per minute per region
```

In [21]:
```
# Constants
city_area = area_region  # this many square kilometers
c = math.floor(num_drivers_region) # Number of drivers
mean_en_route = mean_ride_len # Mean time spent riding from origin to destination
pickup_base = 2 * math.sqrt(city_area) # Mean time to do pickup in minutes is pickup_base / sqrt
max_n = c + 1  # Explore number of riders n in system ranging from 0 to this number-1. Loss syst

# Adjusted Arrival Rate

def adjusted_arrival_rate(n, c, max_wait_time, pickup_base, n0=140, p0=20, p1=30):
    # Probability that pickup time is greater than max_wait_time
    P_t_greater_k = np.exp( (-(c-n))/city_area * math.pi*(speed**2)*(max_wait_time**2))
    # Probability that a rider's valuation is higher than the price
    current_price = price(n, n0, p0, p1)
    P_v_greater_p = 1 - scipy.stats.gamma.cdf(current_price, a=4, scale=3, loc=15)
    return arrival_rate_region_minute*(1-P_t_greater_k)*P_v_greater_p
```

In addition to the other performance measures computed in the python notebook, compute the fraction of riders who cancel. Assume that riders who are blocked due to all drivers being busy are classified as cancellations.

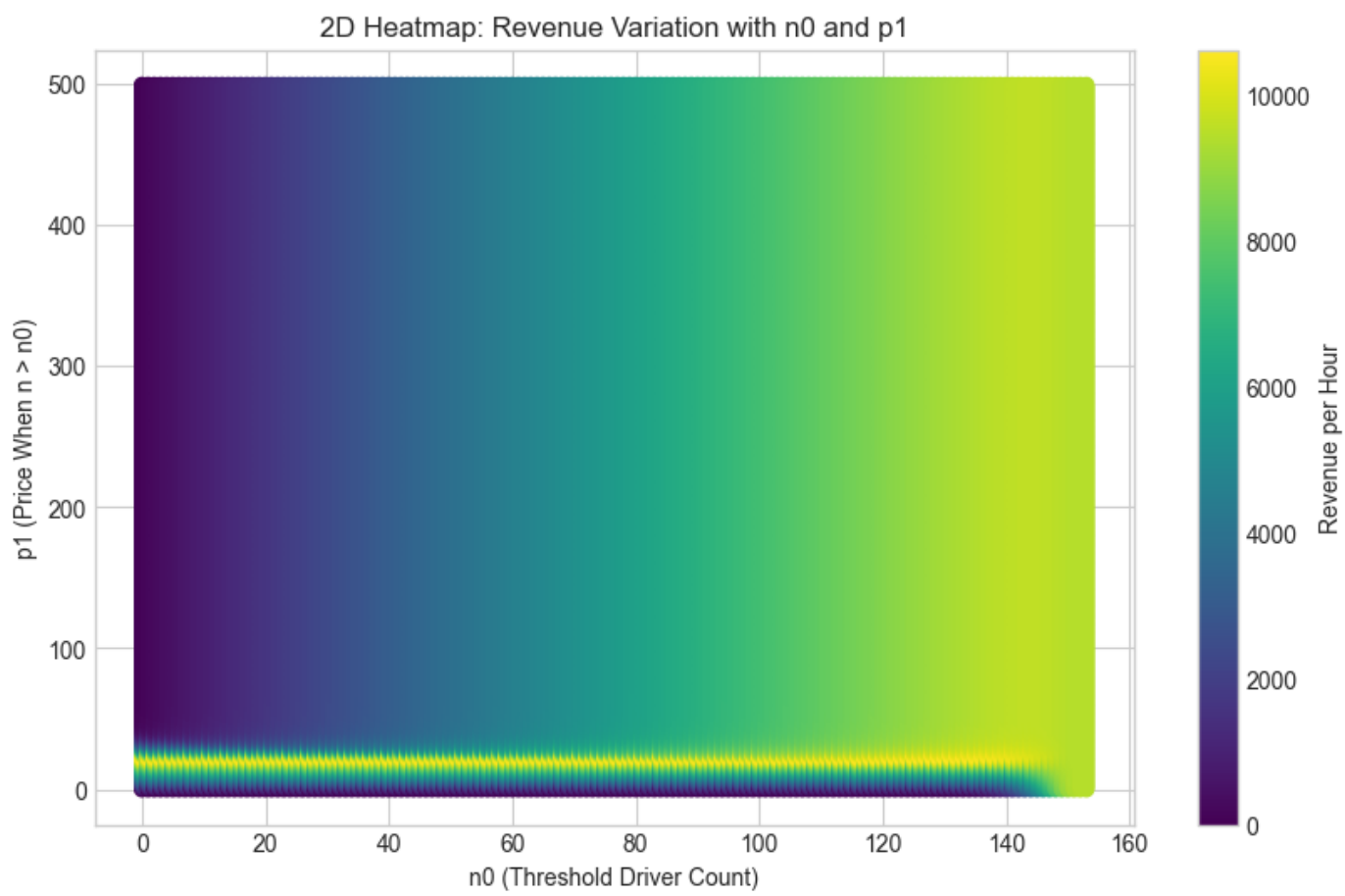
In [38]:
```
def compute_revenue(n0, p0, p1):
    revenue = 0
    mu = np.zeros(max_n)
    adjusted_arrival_rates = np.zeros(max_n)
    for n in range(max_n):
        adjusted_arrival_rates[n] = adjusted_arrival_rate(n, c, max_wait_time, pickup_base,n0,p0
        mu[n] = np.minimum(n, c) / (mean_en_route + pickup_base / math.sqrt(c - n + 1))
    rate_differences = np.abs(adjusted_arrival_rates - mu)
    start_index = np.argmin(rate_differences)
    # start_index gives the location of the closest match between arrival and service rates.
    # Start the recursion for birth-death processes there and work out to edges.
    # Numerically sensible.

    pi = np.zeros(max_n) # Vector giving the stationary distribution
    pi[start_index] = 1 # Temporary value. We rescale to prob distribution later
    for i in range(start_index, max_n-1):
        pi[i+1] = pi[i] * adjusted_arrival_rates[i] / mu[i+1]
    for i in range(start_index, 0, -1):
        pi[i-1] = pi[i] * mu[i] / adjusted_arrival_rates[i-1]
    sum_pi = pi.sum()
    pi = pi / sum_pi
```

```
    for n in range(len(pi)):
        # Compute Service Rate
        revenue += price(n, n0, p0, p1) * pi[n] * adjusted_arrival_rates[n]
    return revenue * 60  # Convert to per hour

revenue_per_hour = compute_revenue(140, 20, 30)
print(f'Revenue per hour: {revenue_per_hour:.2f}')
```

```
Revenue per hour: 10018.35
```

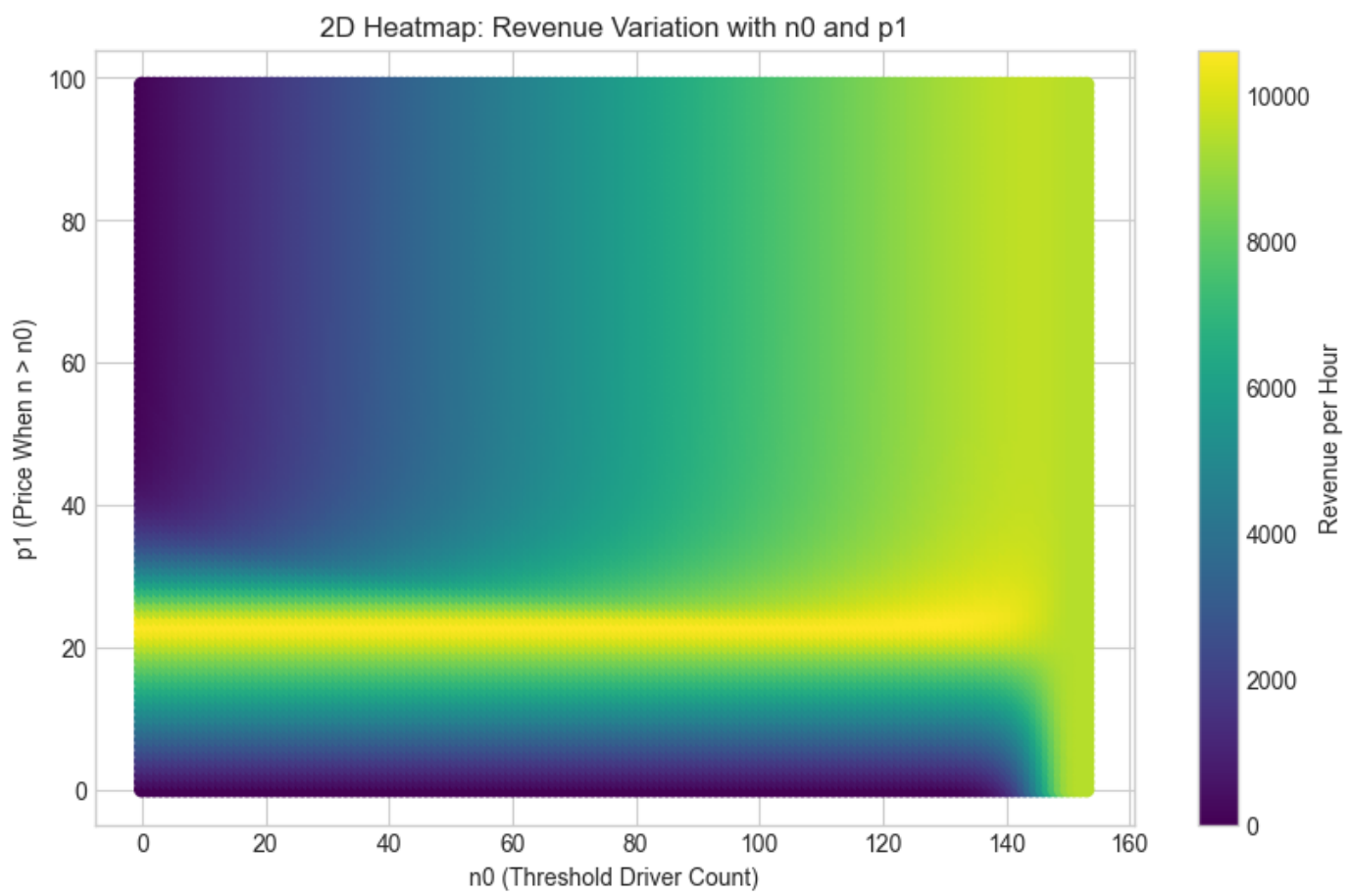
In [23]:

```
#chat GPT helped make heat map

revenue_hourly = []
n0_values = []
p1_values = []

for n0 in range(0, max_n):
    for p1 in range(0, 500):
        revenue_per_hour = compute_revenue(n0, 20, p1)
        revenue_hourly.append(revenue_per_hour)
        n0_values.append(n0)
        p1_values.append(p1)

# Assuming revenue_hourly, n0_values, and p1_values are already calculated
revenue_hourly = np.array(revenue_hourly)
n0_values = np.array(n0_values)
p1_values = np.array(p1_values)

# Creating the 2D plot
plt.figure(figsize=(10, 6))

# Scatter plot with color based on revenue
plt.scatter(n0_values, p1_values, c=revenue_hourly, cmap='viridis')
plt.colorbar(label='Revenue per Hour')

# Setting labels and title
plt.xlabel('n0 (Threshold Driver Count)')
plt.ylabel('p1 (Price When n > n0)')
plt.title('2D Heatmap: Revenue Variation with n0 and p1')

# Show plot
plt.show()
```



In [39]:

```python
#chat GPT helped make heat map

revenue_hourly = []
n0_values = []
p1_values = []

for n0 in range(0, max_n):
    for p1 in range(0, 100):
        revenue_per_hour = compute_revenue(n0, 20, p1)
        revenue_hourly.append(revenue_per_hour)
        n0_values.append(n0)
        p1_values.append(p1)

# Assuming revenue_hourly, n0_values, and p1_values are already calculated
revenue_hourly = np.array(revenue_hourly)
n0_values = np.array(n0_values)
p1_values = np.array(p1_values)

# Creating the 2D plot
plt.figure(figsize=(10, 6))

# Scatter plot with color based on revenue
plt.scatter(n0_values, p1_values, c=revenue_hourly, cmap='viridis')
plt.colorbar(label='Revenue per Hour')

# Setting labels and title
plt.xlabel('n0 (Threshold Driver Count)')
plt.ylabel('p1 (Price When n > n0)')
plt.title('2D Heatmap: Revenue Variation with n0 and p1')

# Show plot
plt.show()
```



In [41]:
```
index = np.argmax(revenue_hourly)
optimal_n0 = n0_values[index]
optimal_p1 = p1_values[index]
print(f'The maximum revenue is achieved with n0 = {optimal_n0} and p1 = {optimal_p1}')
print(f'Maximum Revenue per Hour: {revenue_hourly[index]:.2f}')
```

```
The maximum revenue is achieved with n0 = 121 and p1 = 23
Maximum Revenue per Hour: 10629.93
```

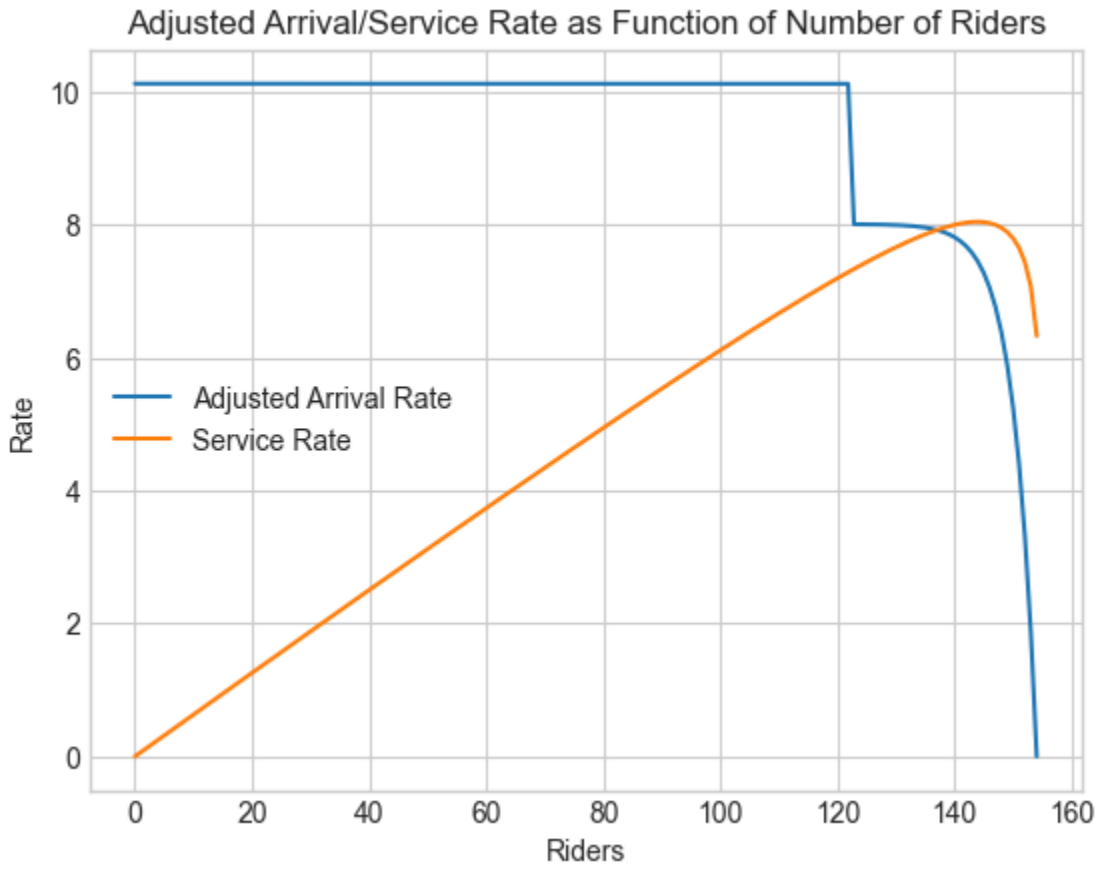
In [172…
```
def price(n, n0=121, p0=20, p1=23):
    return p0 if n <= n0 else p1
```

In [173…
```
area_nyc = 630 #sq_km
arrival_rate = 20000 #per_hour
arrival_rate_min = arrival_rate/60 #per_minute
drivers = 4600
mean_ride_len = 15 #minutes
max_wait_time = 7 #minutes
regions_nyc = 30
speed = 0.190439 #km/min


num_drivers_region = drivers/regions_nyc
area_region = area_nyc/regions_nyc #sq_km
arrival_rate_region_minute = arrival_rate_min/regions_nyc #people per minute per region
```

cite hw2 Q4

In [174…
```
import scipy.stats

# Constants
city_area = area_region  # this many square kilometers
```

```
c = math.floor(num_drivers_region) # Number of drivers
mean_en_route = mean_ride_len # Mean time spent riding from origin to destination
pickup_base = 2 * math.sqrt(city_area) # Mean time to do pickup in minutes is pickup_base / sqrt
max_n = c + 1  # Explore number of riders n in system ranging from 0 to this number-1. Loss syst

# Adjusted Arrival Rate

def adjusted_arrival_rate(n, c, max_wait_time, pickup_base, n0=121, p0=20, p1=23):
    # Probability that pickup time is greater than max_wait_time
    P_t_greater_k = np.exp( (-(c-n))/city_area * math.pi*(speed**2)*(max_wait_time**2))
    # Probability that a rider's valuation is higher than the price
    current_price = price(n, n0, p0, p1)
    P_v_greater_p = 1 - scipy.stats.gamma.cdf(current_price, a=4, scale=3, loc=15)
    return arrival_rate_region_minute*(1-P_t_greater_k)*P_v_greater_p

# Compute Service Rate
mu = np.zeros(max_n)
adjusted_arrival_rates = np.zeros(max_n)
for n in range(max_n):
    adjusted_arrival_rates[n] = adjusted_arrival_rate(n, c, max_wait_time, pickup_base)
    mu[n] = np.minimum(n, c) / (mean_en_route + pickup_base / math.sqrt(c - n + 1))

# Plotting
plt.figure()
plt.title("Adjusted Arrival/Service Rate as Function of Number of Riders")
plt.xlabel("Riders")
plt.ylabel("Rate")
x_nums = np.linspace(0, max_n, max_n)
plt.plot(x_nums, adjusted_arrival_rates, label='Adjusted Arrival Rate')
plt.plot(x_nums, mu, label='Service Rate')
plt.legend()
plt.show()
```

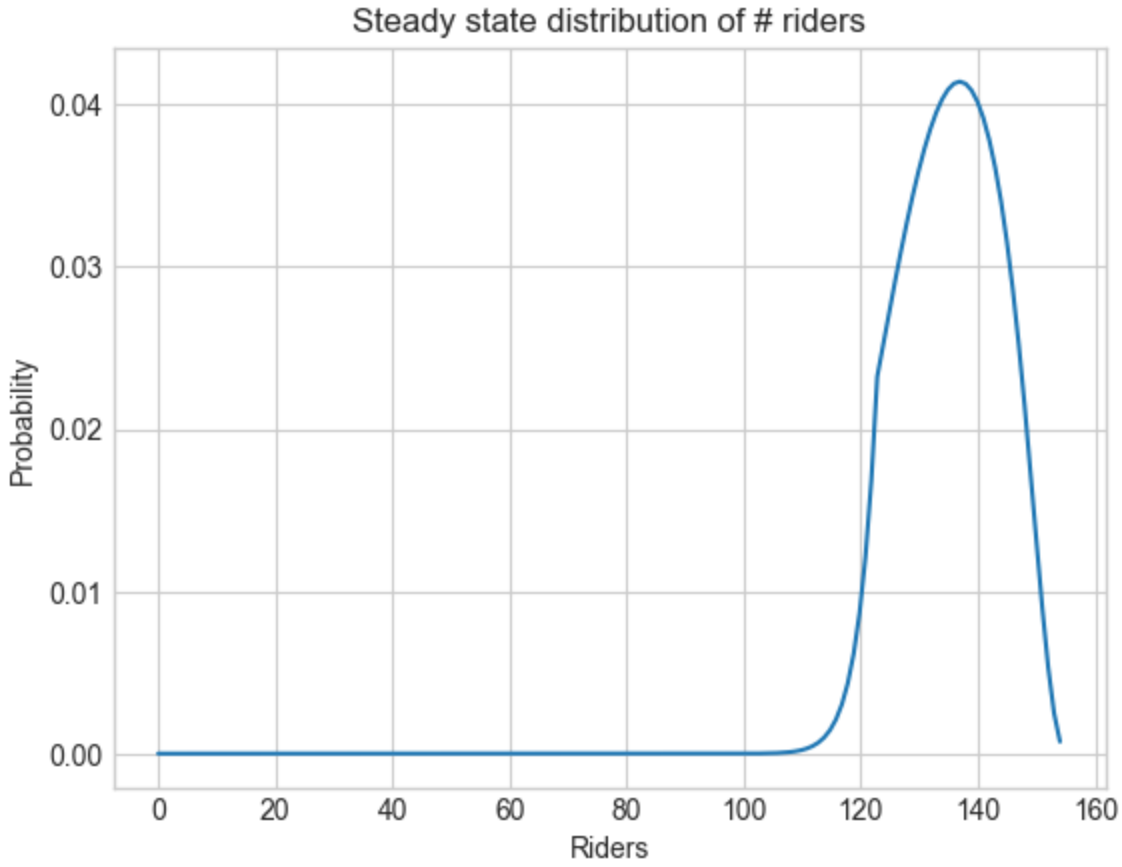


Generate plot of the arrival and service rates

Compute steady-state distribution

In [175…

```
rate_differences = np.abs(adjusted_arrival_rates - mu)
start_index = np.argmin(rate_differences)
# start_index gives the location of the closest match between arrival and service rates.
# Start the recursion for birth-death processes there and work out to edges.
# Numerically sensible.

pi = np.zeros(max_n) # Vector giving the stationary distribution
pi[start_index] = 1 # Temporary value. We rescale to prob distribution later
for i in range(start_index, max_n-1):
    pi[i+1] = pi[i] * adjusted_arrival_rates[i] / mu[i+1]
for i in range(start_index, 0, -1):
    pi[i-1] = pi[i] * mu[i] / adjusted_arrival_rates[i-1]
sum_pi = pi.sum()
pi = pi / sum_pi
#print(pi)
plt.plot(x_nums, pi)
plt.title("Steady state distribution of # riders")
plt.xlabel("Riders")
plt.ylabel("Probability")
plt.show()
```



Compute performance measures

In [176…

```
offered_load = arrival_rate_region_minute
print(f'Offered demand in region = {offered_load:.3f} rides per minute')

print(f'Offered demand in region = {offered_load*60:.3f} rides per hour')
print(' ')
fraction_handled = 1/offered_load*np.sum(pi*adjusted_arrival_rates)
print(f'Fraction of offered demand served = {fraction_handled:.3f}')
print(f'Fraction of rides not taken = {1-fraction_handled:.3f}')
print(' ')
throughput = offered_load*fraction_handled
print(f'Handled Load (throughput) in region = {throughput:.3f} rides per minute')
```

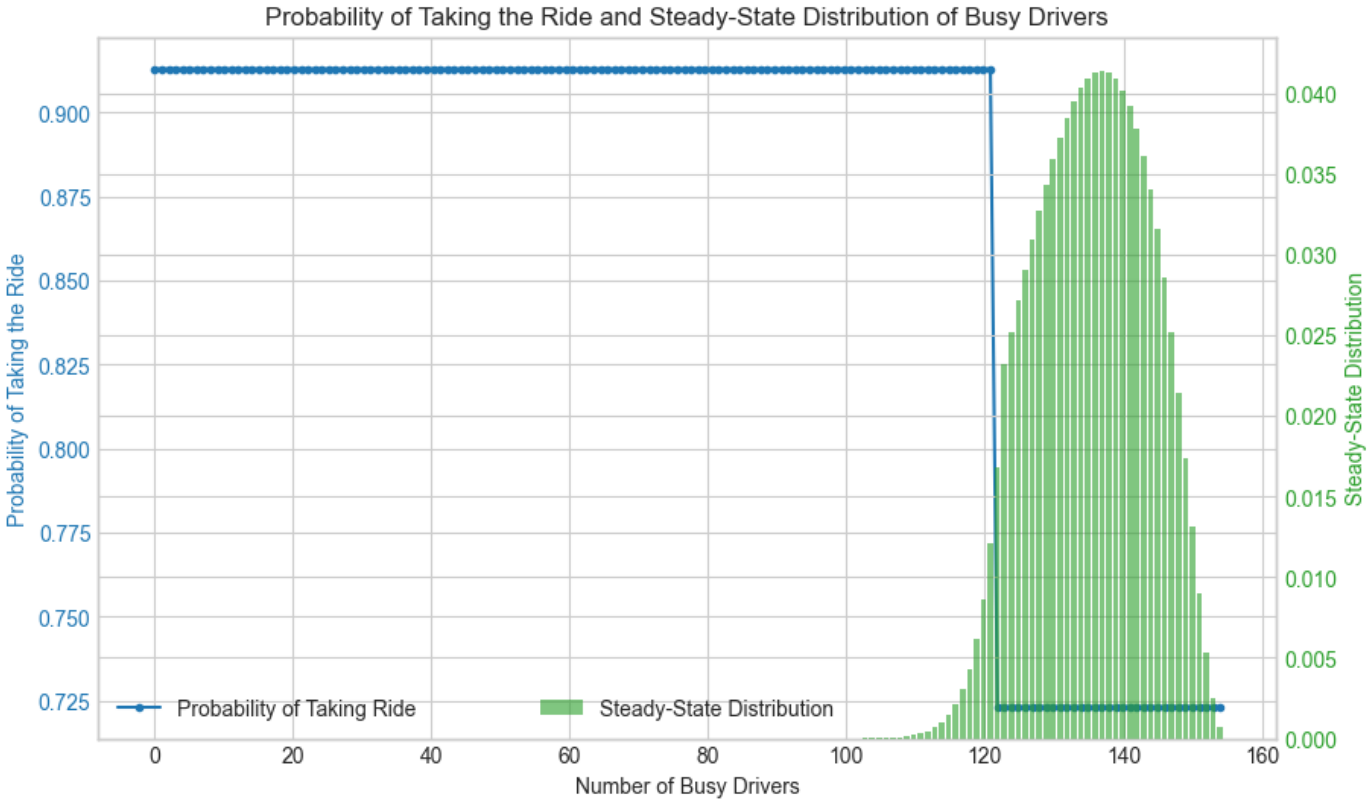
```
print(f'Handled Load (throughput) in region= {throughput*60:.3f} rides per hour')
print(' ')
fraction_enroute = throughput * mean_en_route / c # fraction of driver time en_route
print('Fraction of driver time spent en_route = %.3f' %fraction_enroute)
print('Total driver time spent en route per minute in region = %.1f' %(fraction_enroute * c))
print('Total driver time spent en route per hour in region = %.1f' %(fraction_enroute*60 * c))
print(' ')

busy_drivers = 0
for i in range(max_n):
    busy_drivers += pi[i] * np.minimum(i, c)
fraction_busy = busy_drivers / c
fraction_idle = 1-fraction_busy
fraction_pickup = fraction_busy - fraction_enroute
print('Utilzation of Drivers = %.3f' %fraction_busy)
print('Fraction of driver time spent in pickup = %.3f' %fraction_pickup)
print('Fraction of driver time spent idle = %.3f' %fraction_idle)

average_pickup_time = c * ( fraction_busy - fraction_enroute ) / throughput
print('Average pickup time = %.1f minutes' %average_pickup_time)


print(f'Offered demand in city = {offered_load*30:.3f} rides per minute')

print(f'Offered demand in city = {offered_load*30*60:.3f} rides per hour')
print(' ')

print(f'Handled Load (throughput) in city = {throughput*30:.3f} rides per minute')
print(f'Handled Load (throughput) in city= {throughput*60*30:.3f} rides per hour')
print(' ')
print('Total driver time spent en route per minute in city = %.1f' %(fraction_enroute*30 * c))
print('Total driver time spent en route per hour in city = %.1f' %(fraction_enroute*60*30 * c))
print(' ')
```

```
Offered demand in region = 11.111 rides per minute
Offered demand in region = 666.667 rides per hour

Fraction of offered demand served = 0.700
Fraction of rides not taken = 0.300

Handled Load (throughput) in region = 7.779 rides per minute
Handled Load (throughput) in region= 466.746 rides per hour

Fraction of driver time spent en_route = 0.763
Total driver time spent en route per minute in region = 116.7
Total driver time spent en route per hour in region = 7001.2

Utilzation of Drivers = 0.879
Fraction of driver time spent in pickup = 0.116
Fraction of driver time spent idle = 0.121
Average pickup time = 2.3 minutes
Offered demand in city = 333.333 rides per minute
Offered demand in city = 20000.000 rides per hour

Handled Load (throughput) in city = 233.373 rides per minute
Handled Load (throughput) in city= 14002.392 rides per hour

Total driver time spent en route per minute in city = 3500.6
Total driver time spent en route per hour in city = 210035.9
```

In [177…

```
#chat GPT aid in plotting
# Pricing model parameters
p0 = 20  # price when number of busy drivers is 140 or below
p1 = 23  # price when number of busy drivers is above 140
```

```
n0 = 121



# Determining the price based on the number of busy drivers
prices_based_on_busy_drivers = np.where(x_nums <= n0, p0, p1)

# Calculating the probability for each scenario based on the number of busy drivers
probabilities_based_on_busy_drivers = [probability_take_ride(price, valuations) for price in pri

# Creating a two-y-axis plot
fig, ax1 = plt.subplots(figsize=(10, 6))

# Plotting the probability that rider takes the ride (left y-axis)
color = 'tab:blue'
ax1.set_xlabel('Number of Busy Drivers')
ax1.set_ylabel('Probability of Taking the Ride', color=color)
ax1.plot(x_nums, probabilities_based_on_busy_drivers, color=color, marker='.')
ax1.tick_params(axis='y', labelcolor=color)

# Creating a second y-axis for the steady-state distribution
ax2 = ax1.twinx()
color = 'tab:green'
ax2.set_ylabel('Steady-State Distribution', color=color)
ax2.bar(x_nums, pi, color=color, alpha=0.6)
ax2.tick_params(axis='y', labelcolor=color)

# Adding a legend and grid
ax1.legend(['Probability of Taking Ride'], loc='lower left')
ax2.legend(['Steady-State Distribution'], loc='lower center')
ax1.grid(True)

plt.title('Probability of Taking the Ride and Steady-State Distribution of Busy Drivers')
plt.show()
```



Devise and evaluate the revenue per hour from a fancier pricing scheme that you think would be better than the two-price scheme above

In [221…

```
def price(n):
    return n**3/770000+20
area_nyc = 630 #sq_km
arrival_rate = 20000 #per_hour
arrival_rate_min = arrival_rate/60 #per_minute
drivers = 4600
mean_ride_len = 15 #minutes
max_wait_time = 7 #minutes
regions_nyc = 30
speed = 0.190439 #km/min


num_drivers_region = drivers/regions_nyc
area_region = area_nyc/regions_nyc #sq_km
arrival_rate_region_minute = arrival_rate_min/regions_nyc #people per minute per region
```

In [222…

```
# Constants
city_area = area_region  # this many square kilometers
c = math.floor(num_drivers_region) # Number of drivers
mean_en_route = mean_ride_len # Mean time spent riding from origin to destination
pickup_base = 2 * math.sqrt(city_area) # Mean time to do pickup in minutes is pickup_base / sqrt
max_n = c + 1  # Explore number of riders n in system ranging from 0 to this number-1. Loss syst

# Adjusted Arrival Rate

def adjusted_arrival_rate(n, c, max_wait_time, pickup_base, n0=140, p0=20, p1=30):
    # Probability that pickup time is greater than max_wait_time
    P_t_greater_k = np.exp( (-(c-n))/city_area * math.pi*(speed**2)*(max_wait_time**2))
    # Probability that a rider's valuation is higher than the price
    current_price = price(n)
    P_v_greater_p = 1 - scipy.stats.gamma.cdf(current_price, a=4, scale=3, loc=15)
    return arrival_rate_region_minute*(1-P_t_greater_k)*P_v_greater_p
```

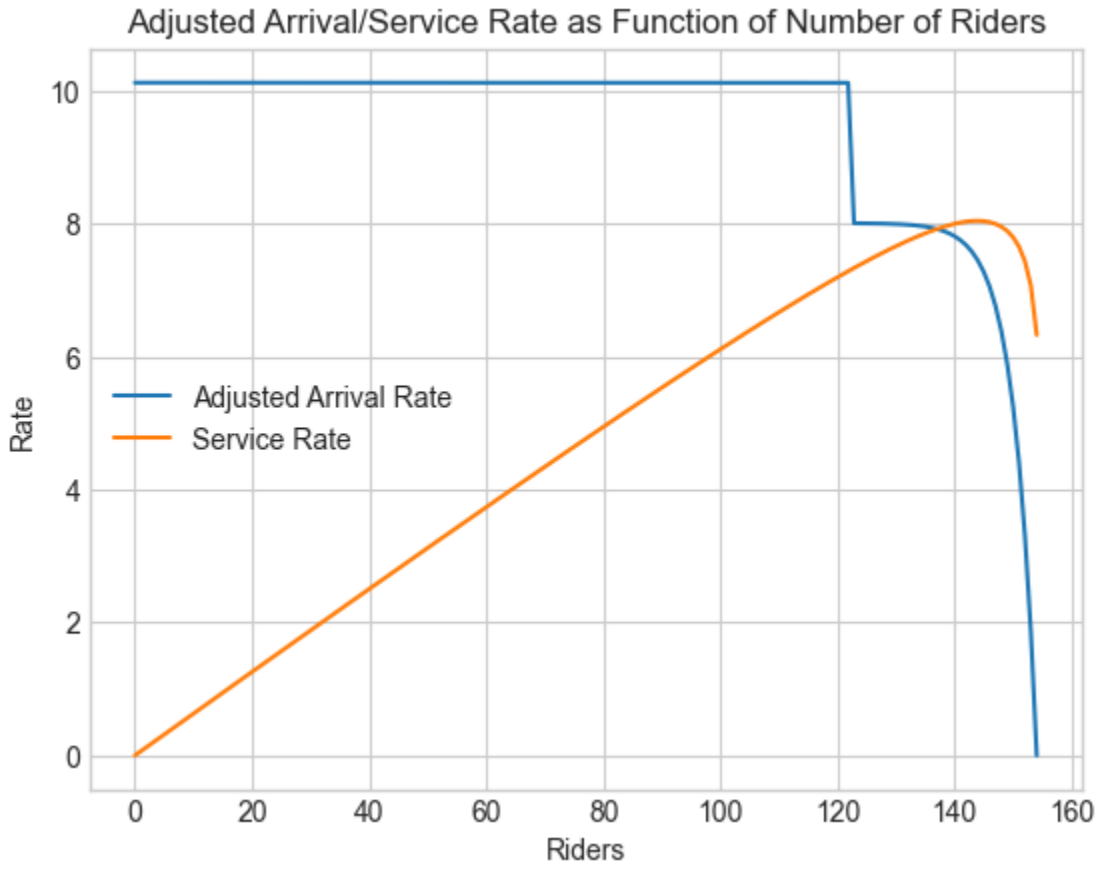
In [223…

```
def compute_revenue(n0, p0, p1):
    revenue = 0
    mu = np.zeros(max_n)
    adjusted_arrival_rates = np.zeros(max_n)
    for n in range(max_n):
        adjusted_arrival_rates[n] = adjusted_arrival_rate(n, c, max_wait_time, pickup_base,n0,p0
        mu[n] = np.minimum(n, c) / (mean_en_route + pickup_base / math.sqrt(c - n + 1))
    rate_differences = np.abs(adjusted_arrival_rates - mu)
    start_index = np.argmin(rate_differences)
    # start_index gives the location of the closest match between arrival and service rates.
    # Start the recursion for birth-death processes there and work out to edges.
    # Numerically sensible.

    pi = np.zeros(max_n) # Vector giving the stationary distribution
    pi[start_index] = 1 # Temporary value. We rescale to prob distribution later
    for i in range(start_index, max_n-1):
        pi[i+1] = pi[i] * adjusted_arrival_rates[i] / mu[i+1]
    for i in range(start_index, 0, -1):
        pi[i-1] = pi[i] * mu[i] / adjusted_arrival_rates[i-1]
    sum_pi = pi.sum()
    pi = pi / sum_pi
    for n in range(len(pi)):
        # Compute Service Rate
        revenue += price(n) * pi[n] * adjusted_arrival_rates[n]
    return revenue * 60  # Convert to per hour

revenue_per_hour = compute_revenue(140, 20, 30)
print(f'Revenue per hour: {revenue_per_hour:.2f}')
```

```
Revenue per hour: 10709.55
```

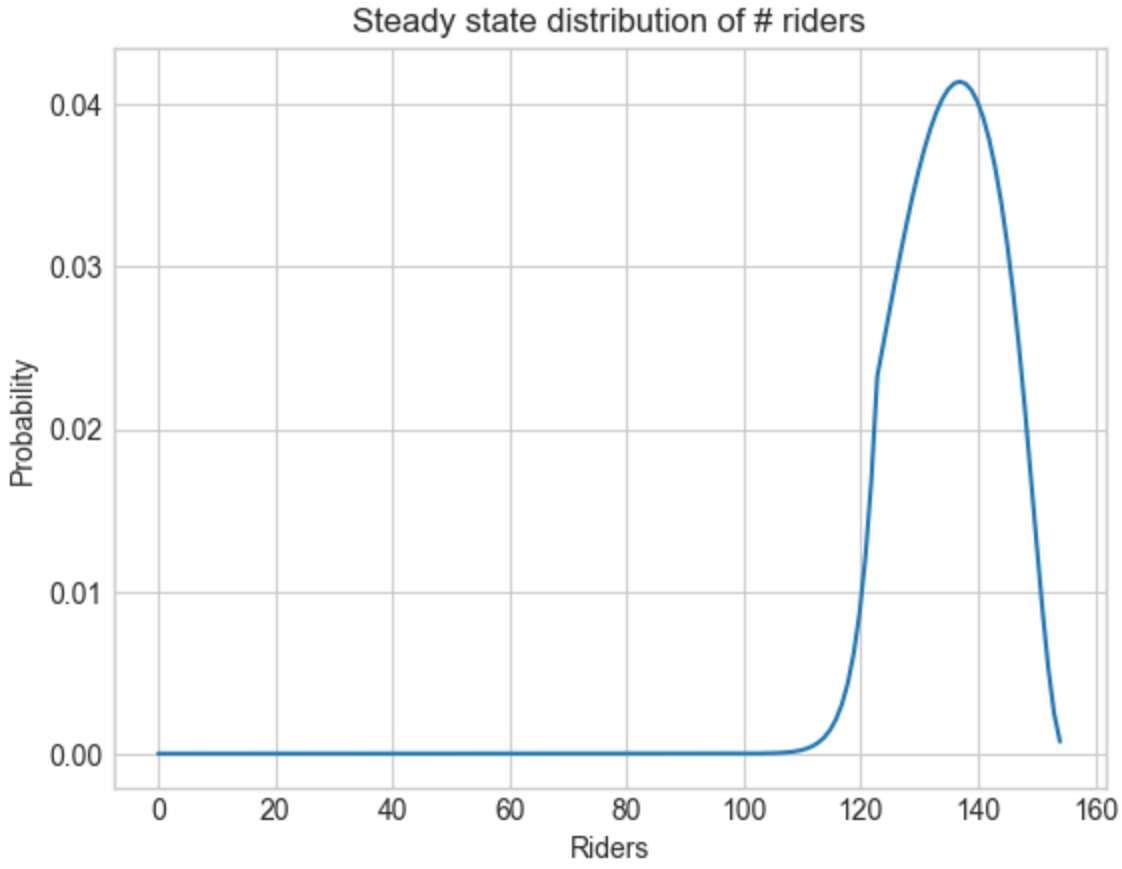
In [224…

```
# Plotting
plt.figure()
plt.title("Adjusted Arrival/Service Rate as Function of Number of Riders")
plt.xlabel("Riders")
plt.ylabel("Rate")
x_nums = np.linspace(0, max_n, max_n)
plt.plot(x_nums, adjusted_arrival_rates, label='Adjusted Arrival Rate')
plt.plot(x_nums, mu, label='Service Rate')
plt.legend()
plt.show()
```



In [225…

```
plt.plot(x_nums, pi)
plt.title("Steady state distribution of # riders")
plt.xlabel("Riders")
plt.ylabel("Probability")
plt.show()
```

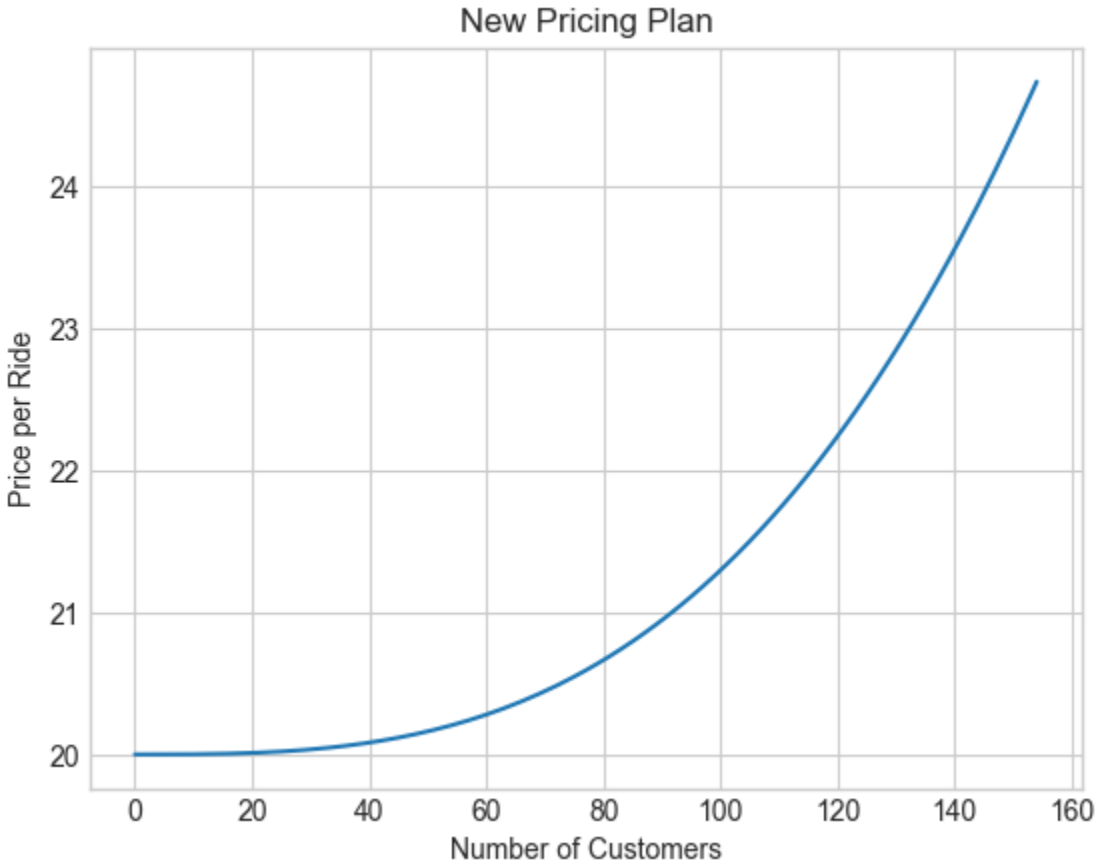


In [207…

```
plt.plot(range(max_n+1),price(np.arange(max_n+1)))
plt.xlabel('Number of Customers')
plt.ylabel('Price per Ride')
plt.title('New Pricing Plan')
```

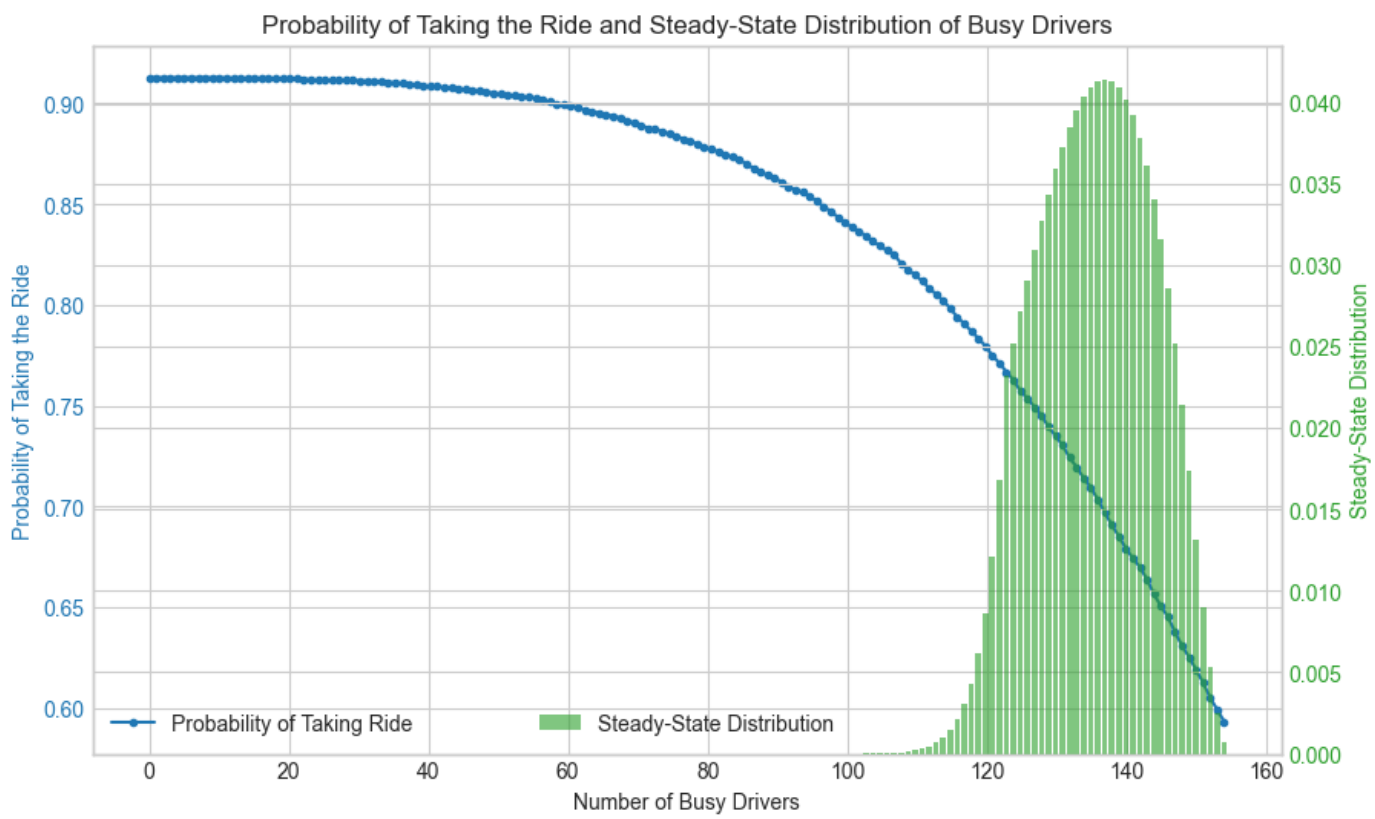
Out[207]:

```
Text(0.5, 1.0, 'New Pricing Plan')
```



In [208…

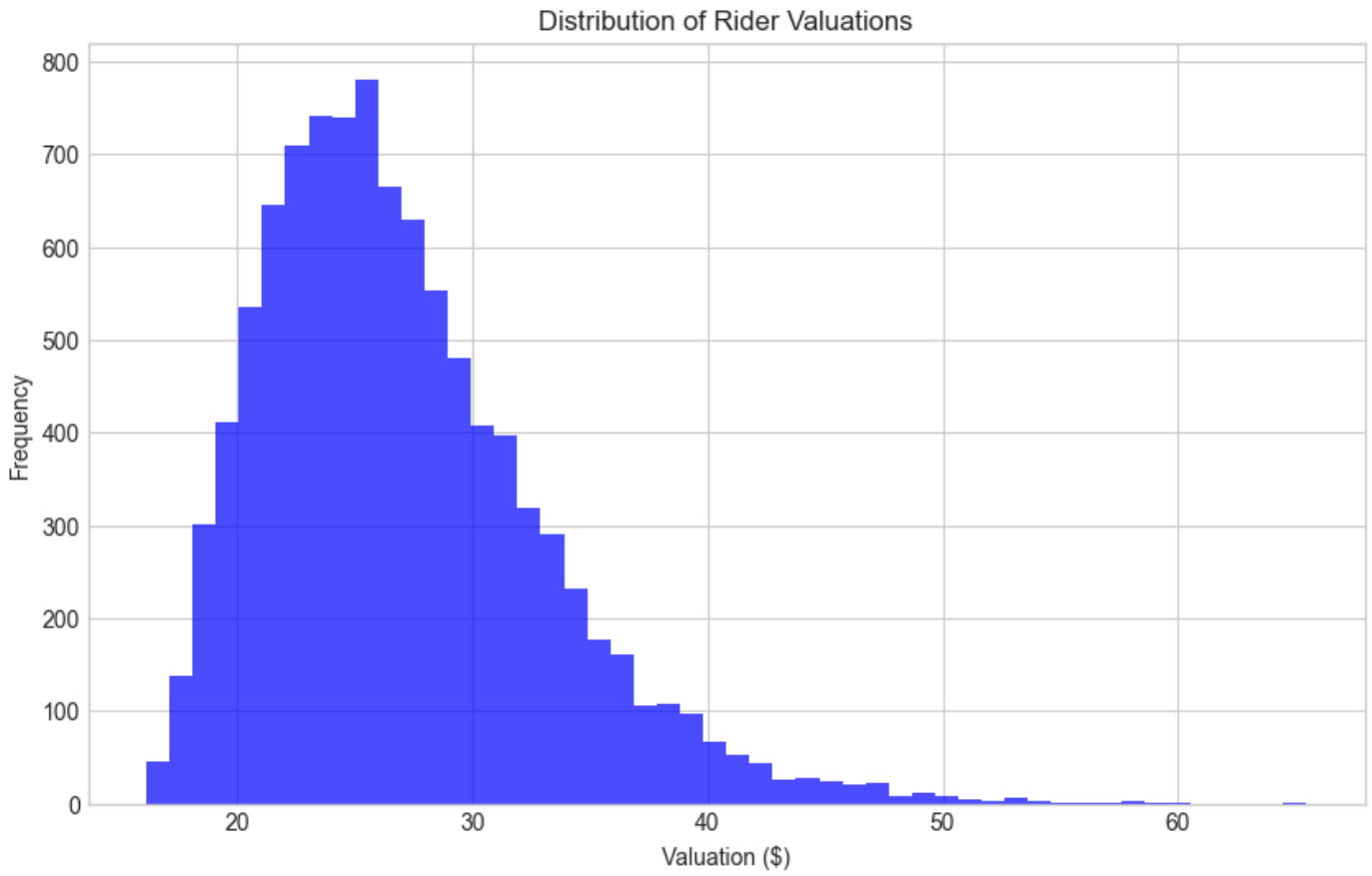
```
#chat GPT aid in plotting
# Pricing model parameters
p0 = 20  # price when number of busy drivers is 140 or below
p1 = 30  # price when number of busy drivers is above 140
n0 = 140



# Determining the price based on the number of busy drivers
prices_based_on_busy_drivers = price(x_nums)

# Calculating the probability for each scenario based on the number of busy drivers
probabilities_based_on_busy_drivers = [probability_take_ride(price, valuations) for price in pri

# Creating a two-y-axis plot
fig, ax1 = plt.subplots(figsize=(10, 6))

# Plotting the probability that rider takes the ride (left y-axis)
color = 'tab:blue'
ax1.set_xlabel('Number of Busy Drivers')
ax1.set_ylabel('Probability of Taking the Ride', color=color)
ax1.plot(x_nums, probabilities_based_on_busy_drivers, color=color, marker='.')
ax1.tick_params(axis='y', labelcolor=color)

# Creating a second y-axis for the steady-state distribution
ax2 = ax1.twinx()
color = 'tab:green'
ax2.set_ylabel('Steady-State Distribution', color=color)
ax2.bar(x_nums, pi, color=color, alpha=0.6)
ax2.tick_params(axis='y', labelcolor=color)

# Adding a legend and grid
ax1.legend(['Probability of Taking Ride'], loc='lower left')
ax2.legend(['Steady-State Distribution'], loc='lower center')
ax1.grid(True)

plt.title('Probability of Taking the Ride and Steady-State Distribution of Busy Drivers')
plt.show()
```

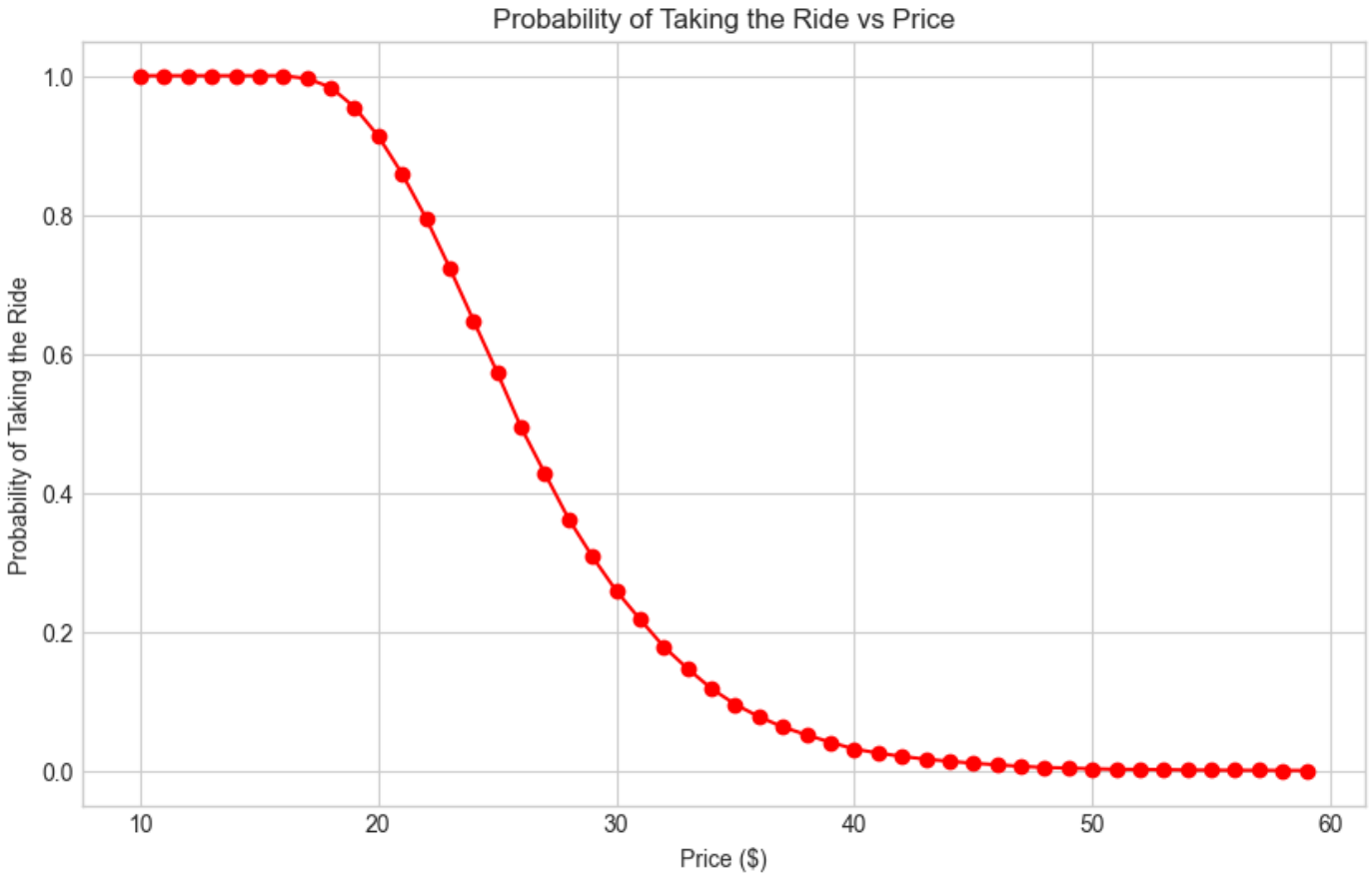


In [194…

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gamma

# Parameters for the shifted gamma distribution
shift = 15
shape = 4
scale = 3

# Generating random valuations for riders using shifted gamma distribution
np.random.seed(0)
sample_size = 10000
valuations = shift + gamma.rvs(shape, scale=scale, size=sample_size)

# Plotting the distribution of rider valuations
plt.figure(figsize=(10, 6))
plt.hist(valuations, bins=50, color='blue', alpha=0.7)
plt.title('Distribution of Rider Valuations')
plt.xlabel('Valuation ($)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Function to calculate the probability that the rider takes the ride based on price
def probability_take_ride(price, valuations):
    return np.mean(valuations > price)

# Different price points
prices = np.arange(10, 60, 1)

# Calculating the probability for each price point
probabilities = [probability_take_ride(price, valuations) for price in prices]

# Plotting the probability that rider takes the ride vs price
plt.figure(figsize=(10, 6))
plt.plot(prices, probabilities, color='red', marker='o')
plt.title('Probability of Taking the Ride vs Price')
plt.xlabel('Price ($)')
```

```
plt.ylabel('Probability of Taking the Ride')
plt.grid(True)
plt.show()
```





In [ ]: