Online Marketplaces

Part 2: Topics in Marketplace Design

Bar Light
National University of Singapore

January 14, 2026

Random Variables Recap

Informally, a random variable can be described by a function that assigns probabilities to outcomes.

Types of Random Variables:

- **Discrete Random Variables:** Defined by a **probability mass function (PMF)** $P(Y = x)$. Example: Rolling a fair dice

$$P(Y = x) = \frac{1}{6}, \quad x \in \{1, 2, 3, 4, 5, 6\}$$

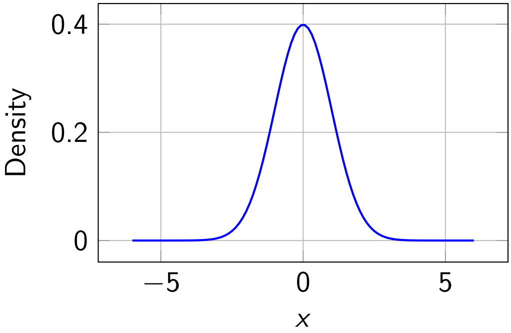
- **Continuous Random Variables:** Defined by a **probability density function (PDF)** $f_Y(x)$. Probability over an interval:

$$P(a \leq Y \leq b) = \int_a^b f_Y(x)\, dx$$

Examples of Continuous Distributions

1. **Normal Distribution:**
   - Bell-shaped curve, symmetric around the mean μ.
   - PDF:
   $$f_Y(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$
   - Example: Heights of people.

Normal Distribution $(\mu = 0, \sigma^2 = 1)$



Examples of Continuous Distributions

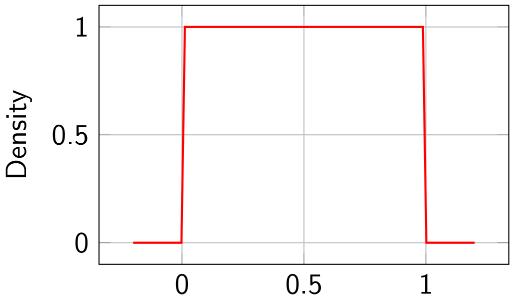
2. Uniform Distribution:

- Equal probability over a range $[a, b]$.
- PDF:

$$f_Y(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{otherwise} \end{cases}$$

- Example: Random Number Generators, e.g., rand() that generate numbers uniformly between 0 and 1.

Uniform Distribution ($[0, 1]$)



Python Code: Generating Random Variables

Code Example: Generating Normal and Uniform Samples

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

# Normal Distribution
mu, sigma = 0, 1  # Mean and standard deviation
normal_samples = np.random.normal(mu, sigma, 10000)

# Uniform Distribution
a, b = 0, 1  # Range [a, b]
uniform_samples = np.random.uniform(a, b, 10000)

# Plot Normal Distribution
plt.hist(normal_samples, bins=50, density=True, color=
    'blue')
plt.title('Normal Distribution (\u03bc=0, \u03c3=1)')
plt.xlabel('Value')
plt.ylabel('Density')
plt.show()
```

Mathematical Notation Refresher

1. **Summation Notation:**
   - Summation:
$$\sum_{i=1}^{n} a_i = a_1 + a_2 + \cdots + a_n$$
2. **Inner Product ($x^\top \theta$):**
   - Definition:
$$x^\top \theta = \sum_{i=1}^{d} x_i \theta_i$$
     where $x = [x_1, x_2, \ldots, x_d]$ and $\theta = [\theta_1, \theta_2, \ldots, \theta_d]$.
3. **Matrices:**
   - Matrix Inverse: For a square matrix A, A^{-1} satisfies:
$$AA^{-1} = I$$
     where I is the identity matrix.
   - Example: For $A = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$, its inverse is: $A^{-1} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$.

Python Code: Notation

```python
import numpy as np

# Vectors
x = np.array([1, 2, 3])
theta = np.array([4, 5, 6])

# Compute summation
summation = np.sum(x)

# Compute dot product
dot_product = np.dot(x, theta)

# Define a square matrix
A = np.array([[2, 0],
              [0, 2]])

# Compute the inverse
A_inverse = np.linalg.inv(A)
```

Platforms as Market Designers: Examples

We focus on ridesharing.

- Platforms like Uber, and Grab design markets through:
  1. **Matching:** How to efficiently connect supply (drivers) with demand (riders) in real-time.
  2. **Quality Selection:** Ensuring service standards and optimal price-quality menus.
  3. **Demand Choice Modeling:** Using data and modeling to estimate demand.
  4. **Pricing and Incentives:** Using surge pricing to incentivize drivers.
  5. **Spatial Pricing:** Adjusting prices based on geographical factors to account for shortages.

- The examples discussed are stylized but introduce core challenges.
- Data science teams play a critical role in addressing these problems through algorithm design, predictive modeling, and real-time optimization.

1. Ride-Sharing Matching Example

Consider N drivers and N riders. The drivers and riders are each represented as random points uniformly distributed on $[0, 1]^2$.

- In the traditional taxi industry we can think of the matching between drivers and riders as random.

The value of a designer

- Simplest operations: In the greedy algorithm, the driver-rider pair with the shortest distance is matched and removed from the pool in each iteration.

- Advanced operations: Finding the optimal matching that minimizes the total distance using optimization.

Greedy Algorithm: Simplest Approach to Matching

Greedy Algorithm.

- Drivers and riders are represented as points in a 2D space.
- At each step:
  1. Find the driver-rider pair with the shortest Euclidean distance.
  2. Match them and remove them from the pool.
- Repeat until all drivers and riders are matched.

Advantages:

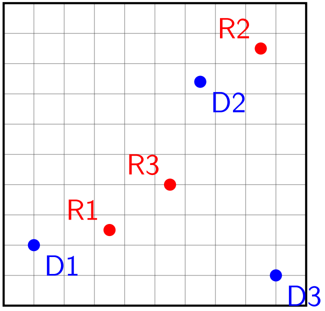
- Simple to implement and computationally fast.
- Provides reasonable matches for small-scale problems.

Limitation:

- Not guaranteed to minimize the total distance.

Example: Visualizing Greedy Algorithm Matching (3x3)

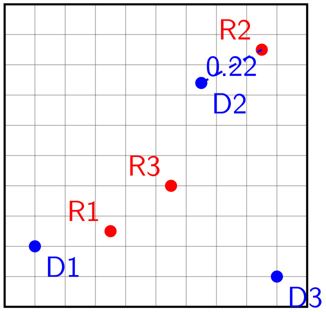
Driver and Rider Locations in 2D Space:



Greedy algorithm computes all possible distances between drivers and riders.

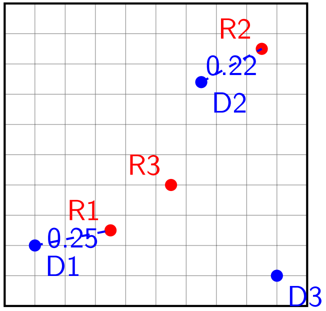
Example: Visualizing Greedy Algorithm Matching (3x3)

Driver and Rider Locations in 2D Space:



- D2 is matched to R2 as the closest pair (distance 0.22).
- Remaining drivers and riders will be matched in subsequent steps.

Greedy Algorithm

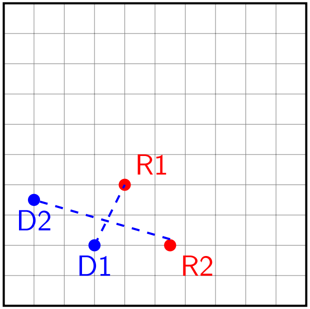


Greedy Matching Steps:

- Step 1: Match D2 - R2 (0.22).
- Step 2: Match D1 - R1 (0.25).
- Step 3: Match D3 - R3 (0.47).

Greedy Algorithm Not Optimal

Driver and Rider Locations in 2D Space:



Distances:
R1 to D1 (0.22), D2 to R2 (0.48), D1 to R2 (0.25), D2 to R1 (0.3).

- Greedy algorithm matches D1 to R1 and D2 to R2 (0.22 + 0.48).
- Optimal matching matches D1 to R2 and D2 to R1 (0.25 + 0.3).

Optimal Matching

What is the Optimal Matching Problem?

- Goal: Minimize the total distance across all matches.

How can we solve this?

1. Define decision variables: $x_{ij} = \begin{cases} 1, & \text{if driver } i \text{ is matched to rider } j \\ 0, & \text{otherwise} \end{cases}$
2. Objective: Minimize $\sum_{i,j} d_{ij} x_{ij}$, where d_{ij} is the distance.
3. Constraints:
   - Each driver is matched to exactly one rider: $\sum_j x_{ij} = 1$.
   - Each rider is matched to exactly one driver: $\sum_i x_{ij} = 1$.
   - $x_{ij} \in \{0, 1\}$.

Greedy vs Optimization

- This problem is a special case (assignment problem) where the relaxation to continuous variables $x_{ij} \in [0, 1]$ produces integer solutions.

Trade-off: Optimal but Engineering Intensive

- Guarantees globally optimal matches.
- Useful in large, dense urban areas like NYC where:
  - Riders and drivers can be batched efficiently within regions.
- Requires:
  - Strong data science capabilities.
  - Algorithms to handle trade-offs between batching and waiting.

Discussion: Which algorithm would you choose in a new platform?

Matching Exercise: Network Effect

- We have n **riders** and n **drivers**.
- Each rider needs exactly one driver, and each driver can serve at most one rider.

- Think of a **unit square map**: $[0,1] \times [0,1]$.
- Assume rider and driver locations are drawn **independently and uniformly at random** on this square.
- We vary market thickness by trying $n \in \{10, 30, 50\}$ and repeat the experiment many times (e.g., 1000 simulations).

- Suppose the cost of matching rider i to river j is the Euclidean distance

$$d_{ij} = \sqrt{(x_i - x'_j)^2 + (y_i - y'_j)^2}$$

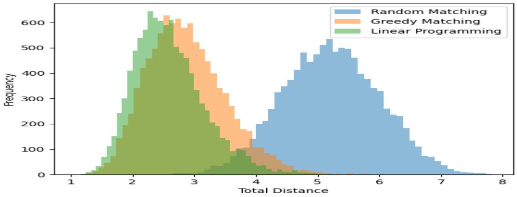
Generate Locations + Compute Distances in Python

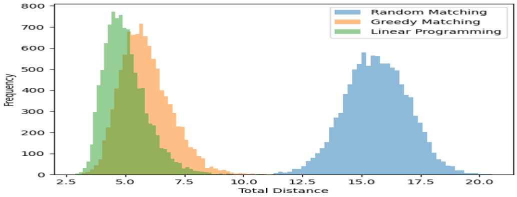
```
import numpy as np
from scipy.spatial.distance import cdist

def generate_positions(n):
    # Generate n random points in the unit square [0, 1]^2
    return np.random.rand(n, 2)

def compute_distances(riders, drivers):
    # Compute Euclidean distances between riders and drivers
    return cdist(riders, drivers, metric="euclidean")

# Define different values for n
n_values = [10, 30, 50]

for n in n_values:
    riders = generate_positions(n)
    drivers = generate_positions(n)

    distances = compute_distances(riders, drivers)
```

Each possible matching has a different total distance.

The Value of Platforms: Ride-Sharing Matching Example

Simulate 10,000 times and compute the total distance using the three methods for N=10 and N=30. Network effect?

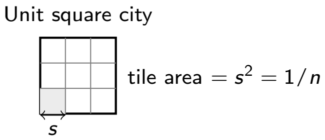




A thicker market makes ("good") matching more efficient.

Why the Network Effect Shows Up (Intuition)

- Riders and drivers are placed uniformly in the **same unit square** (the city does not get bigger).
- Increasing n means **more agents packed into the same area**.



- Split the city into n equal square tiles. Each tile has area $1/n$. Side length s is $s = 1/\sqrt{n}$.
- A typical nearby rider–driver distance is on the order of the tile size: $\approx 1/\sqrt{n}$.

- LP tries to match **mostly locally** when the market is thick.
- Total distance $\approx$ (number of matches) $\times$ (distance per match) $n \cdot 1/\sqrt{n} = \sqrt{n}$
- On the other hand, random matching ignores geography.

Ride-Sharing Matching Example: Complexities

The ride-sharing matching problem in real life is significantly more complex (examples?). Good news for data scientists – greater need for optimizing!

- Geographic Space: Real-world geographic space is complex due to the presence of roads, traffic conditions, buildings, etc. The shortest path between two points can be complicated to compute.
- Multiple Objectives: Ride-sharing platforms may need to consider multiple objectives when matching drivers and riders, such as minimizing wait times, maximizing driver earnings, rider safety, etc.
- Rebalancing Problem: Over time, the distribution of drivers might become imbalanced, with too many drivers in some areas and too few in others.
- Strategic Behavior: Both drivers and riders can behave strategically.

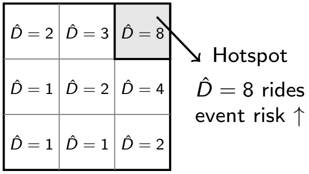
Example: Future Demand (Forecast by Tile)

What platforms do in practice:

- The city is split into areas. For each area we forecast future demand $\hat{D}$ (expected ride requests at some future time).
- Forecasts come from ML using historical patterns, weather, and other features.
- Can we fully trust historical data? Imagine a big concert (e.g., Taylor Swift / Blackpink)..

Platform's response: **reposition / nudge idle drivers** toward predicted hotspots (often via incentives).

This becomes (MUCH) harder than one-shot matching (why)?



$n = 9$ tiles

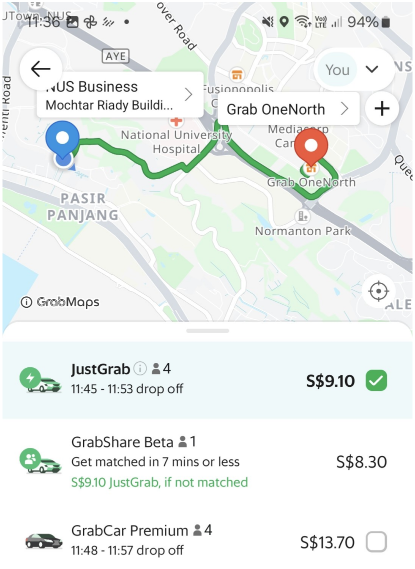
Discussion: Centralized vs Decentralized Matching

- We have seen how centralized matching (e.g., ride-sharing platforms) benefits network effects and direct optimization of matches.
- On the other hand, many freelance platforms (e.g., Upwork) rely on decentralized matching with assortment or recommendations.

Discussion:

- Why do you think centralized matching works well for some platforms (e.g., ride-sharing), while decentralized matching is preferred in others (e.g., labor platforms)?
- What factors (e.g., scale, real-time, commoditization, user preferences) influence these design choices?

2. Quality Selection



NUS Business
Mochtar Riady Buildi...

JustGrab 4
11:45 - 11:53 drop off
S$9.10 ✓

GrabShare Beta 1
Get matched in 7 mins or less
S$9.10 JustGrab, if not matched
S$8.30

GrabCar Premium 4
11:48 - 11:57 drop off
S$13.70 ☐

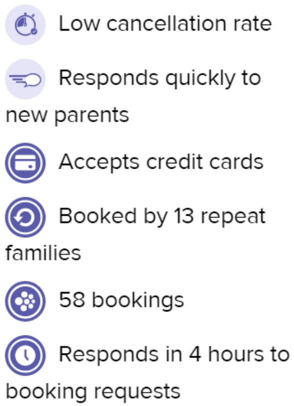
Quality Selection General Platforms

Platforms use **many sources of information** to classify the quality of their sellers.

Platforms **share some** of this information with buyers.

Buyers' and Sellers' Behavior on the Platform

urbansitter



Low cancellation rate

Responds quickly to new parents

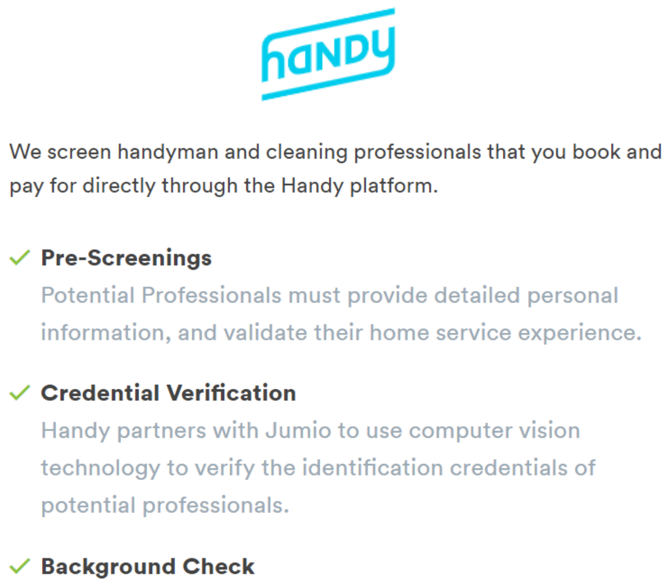
Accepts credit cards

Booked by 13 repeat families

58 bookings

Responds in 4 hours to booking requests

Pre-Screening



We screen handyman and cleaning professionals that you book and pay for directly through the Handy platform.

✓ **Pre-Screenings**

Potential Professionals must provide detailed personal information, and validate their home service experience.

✓ **Credential Verification**

Handy partners with Jumio to use computer vision technology to verify the identification credentials of potential professionals.

✓ **Background Check**

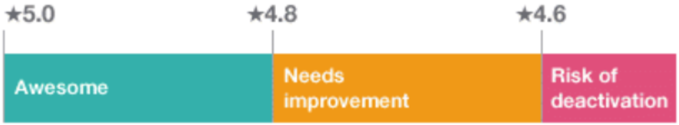
Quality Selection General Platforms: Information Design

Platforms **collect** information about sellers.

Platforms **share** information about sellers with buyers.

$\rightarrow$ Platforms **influence** the participation of sellers based on their quality.

Platforms Ban Low Quality Sellers



"If Lyft's records show that your **car has become too old to use**, they'll deactivate you."

"Lyft will deactivate you for **unsafe driving behavior**."

Platforms Boost the Visibility of High Quality Sellers

bonanza

Ancient Egypt Papyrus Hieroglyphics Womens Coin Bag Purse

$4.72 + $2.99 ship

Ancient Egypt Papyrus Hieroglyphics Framed Tile Wall Picture

$9.39 + $3.99 ship

Ancient Egypt Papyrus Hieroglyphics Hardshell Case for HTC Windows Phone 8s

$14.07 + $3.99 ship

Grab's Quality Selection

This raises crucial design questions: how much of its **available** information about sellers' quality a platform should share with buyers?

Which sellers should it allow to participate?

Example: Grab's Dilemma:

- Option 1: Remove **JustGrab**, keeping only **GrabCar Premium**.
- Option 2: Offer both **GrabCar Premium** and **JustGrab**.

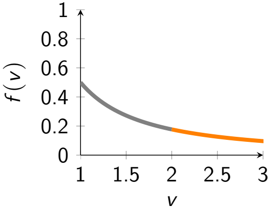
Questions:

- How does the demand structure affect revenue under different options?
- What trade-offs arise from removing the lower-tier service (**JustGrab**)?

Option 1: High-Quality Menu (**GrabCar Premium Only**)

Option 1:

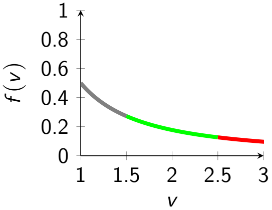
- Removes **JustGrab**, offering only **GrabCar Premium** for price p_H.
  v reflects how riders value quality vs. price.
- Potential riders are split:
  - **Black:** Close the app.
  - **Orange:** Choose **GrabCar Premium**.



Option 2: Full Menu (**GrabCar Premium + JustGrab**)

Option 2:

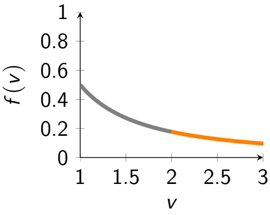
- Offers both options to consumers. Prices $p_L < p_H$.
- Riders are split.
  - **Black:** Close the app.
  - **Green:** Choose **JustGrab**.
  - **Red:** Choose **GrabCar Premium**.



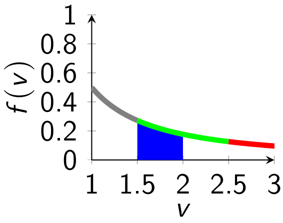
Trade-Off:

- **Advantage:** Attracts price-sensitive consumers.
- **Disadvantage:** Lower revenue per ride from **JustGrab**.

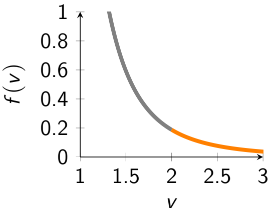
Elasticity: Graphical Intuition



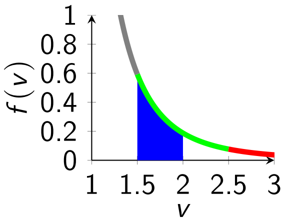
(a) Low Elasticity



(b) Low Elasticity



(c) High Elasticity



(d) High Elasticity

Trade-Off Analysis

Key Factors to Consider:

- **Revenue Gains:**
  - Removing **JustGrab** increases price per ride for remaining customers.
- **Revenue Losses:**
  - Removing **JustGrab** causes some price-sensitive customers to leave.

Outcome:

- If demand is less elastic (e.g., fewer buyers leave when removing JustGrab), removing **JustGrab** is more profitable.
- If demand is more elastic (e.g., many buyers leave when removing JustGrab), offering both options is better.

Price Impact: Removing JustGrab

We assumed that removing JustGrab drivers does not change prices (which are set to balance supply and demand).

Is this realistic?

Removing **JustGrab** increases demand for high-quality service (**GrabCar Premium**) because:

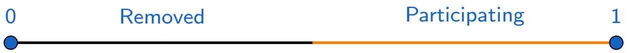
- Some riders who previously selected **JustGrab**, now shift to **GrabCar Premium**.

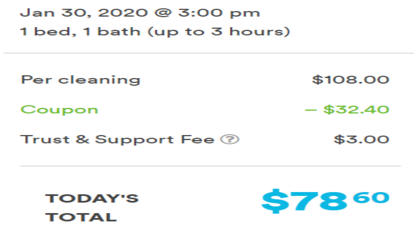
Impact on Revenue:

- Higher demand for **GrabCar Premium** increases price (p_H).

  Revenues increase or decrease?

Simple Menu Discussion?





Jan 30, 2020 @ 3:00 pm
1 bed, 1 bath (up to 3 hours)

Per cleaning	$108.00
Coupon	– $32.40
Trust & Support Fee ?	$3.00
TODAY'S TOTAL	$78.60

Why choose this 1 price 1 quality menu?

3. Choice Modeling

Utility Function: Each rider has a parameter m representing their price-quality sensitivity. Given a price-quality pair (p, q), the rider's utility is $U(v, q, p)$ and 0 for not buying. Vertical differentiation example:

$$U(v, p, q) = vq - p.$$

Rider Decision Rule: A buyer chooses the price-quality pair (p_i, q_i) from the menu that maximizes their utility:

$$(vq_i - p_i) \geq (vq_j - p_j), \quad \text{for all } (p_j, q_j) \text{ available in menu.}$$

If for all pairs (p, q), $(vq - p < 0)$, the buyer does not buy.

Demand. Riders have density $f(v)$. The *demand* for a price-quality pair is the total mass of buyers whose utility is maximized by that pair.

Buyer Choice Modeling: Horizontal Differentiation

- Unlike vertical differentiation, horizontal differentiation focuses on **taste preferences** rather than intrinsic quality.

Example: Logit Model

- Utility for product j by customer i:

$$\beta_p p_j + \beta_q q_j + \gamma x_j + \epsilon_{ij}$$

  p_j: Price, q_j: Quality, x_j: Contextual features, ϵ_{ij}: Random taste capturing unobserved features.

- Example: In e-commerce, one customer might prefer a product because of brand loyalty or aesthetic appeal.
- Example: In a food delivery platform, customers might prefer one restaurant over another because of familiarity with the cuisine, emotional association, random craving, etc.

Choice Modeling Example

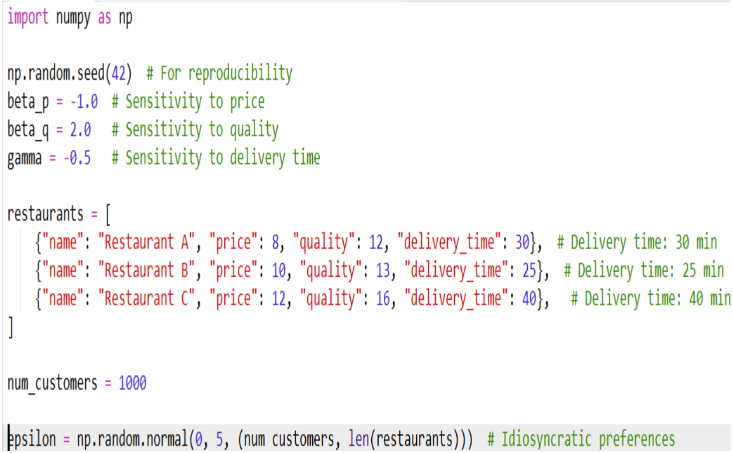
Scenario: Customer choice in food delivery platform (e.g., GrabFood)

- Three restaurants compete for customers based on:
  - **Price**: The cost of the meal.
  - **Quality**: Customer ratings of the restaurant.
  - **Delivery Time**: Estimated time to deliver the order.

Class Exercise

$U_{ij} = \beta_p p_j + \beta_q q_j + \gamma x_j + \epsilon_{ij}.$.
How to calculate the percentage of customers choosing each restaurant?

First assume parameters β_p, β_q, γ are estimated from data (how?).



```
import numpy as np

np.random.seed(42)  # For reproducibility
beta_p = -1.0  # Sensitivity to price
beta_q = 2.0   # Sensitivity to quality
gamma = -0.5   # Sensitivity to delivery time

restaurants = [
    {"name": "Restaurant A", "price": 8, "quality": 12, "delivery_time": 30},  # Delivery time: 30 min
    {"name": "Restaurant B", "price": 10, "quality": 13, "delivery_time": 25},  # Delivery time: 25 min
    {"name": "Restaurant C", "price": 12, "quality": 16, "delivery_time": 40},   # Delivery time: 40 min
]

num_customers = 1000

epsilon = np.random.normal(0, 5, (num customers, len(restaurants)))  # Idiosyncratic preferences
```

Class Exercise

Generate utilities[i, j] to represent the utility of restaurant *j* for customer *i*.

$$\text{utilities} = \begin{bmatrix} 3.2 & 4.1 & 2.8 \\ 5.0 & 3.3 & 4.7 \\ 2.1 & 2.5 & 3.9 \\ 4.0 & 3.6 & 2.7 \\ 3.8 & 4.2 & 3.0 \end{bmatrix}$$

Find the restaurant with the highest utility for each customer:

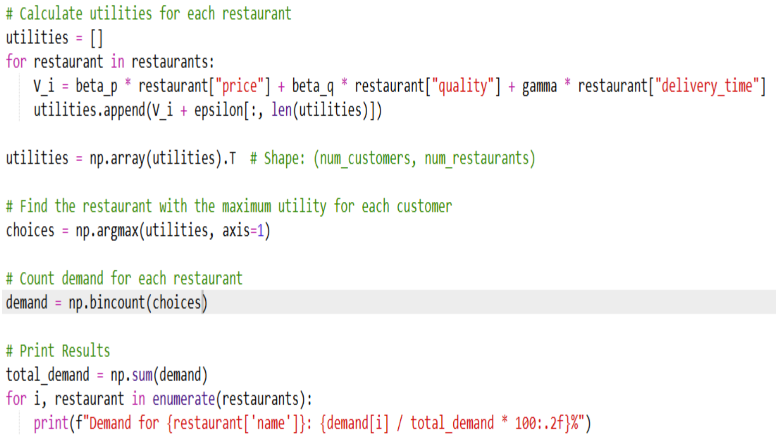
$$\text{choices} = \text{np.argmax(utilities, axis=1)}.$$

Output: choices $= [1, 0, 2, 0, 1]$.
Count the occurrences of each restaurant index in `choices`:

$$\text{demand} = \text{np.bincount(choices)}$$

Output: demand $= [2, 2, 1]$.
Final demand probabilities are 40%, 40%, 20%.

Class Exercise



```
# Calculate utilities for each restaurant
utilities = []
for restaurant in restaurants:
    V_i = beta_p * restaurant["price"] + beta_q * restaurant["quality"] + gamma * restaurant["delivery_time"]
    utilities.append(V_i + epsilon[:, len(utilities)])

utilities = np.array(utilities).T  # Shape: (num_customers, num_restaurants)

# Find the restaurant with the maximum utility for each customer
choices = np.argmax(utilities, axis=1)

# Count demand for each restaurant
demand = np.bincount(choices)

# Print Results
total_demand = np.sum(demand)
for i, restaurant in enumerate(restaurants):
    print(f"Demand for {restaurant['name']}: {demand[i] / total_demand * 100:.2f}%")
```

Output: Demand for Restaurant A: 29.30%
Demand for Restaurant B: 50.10%
Demand for Restaurant C: 20.60%

Choice Modeling Application: Personalization

We model the utility of customer i choosing restaurant j as:

$$U_{ij} = \beta_p \cdot \text{price}_j + \beta_q \cdot \text{quality}_j + \gamma \cdot \text{delivery time}_j + \boldsymbol{\theta}^T \boldsymbol{f}_{ij} + \epsilon_{ij}$$

where $\boldsymbol{f}_i$ is a vector of features specific to customer i (e.g., demographics, behavior).

Estimate Model:
Data: Observed customer choices (which restaurant each customer chose from an assortment).

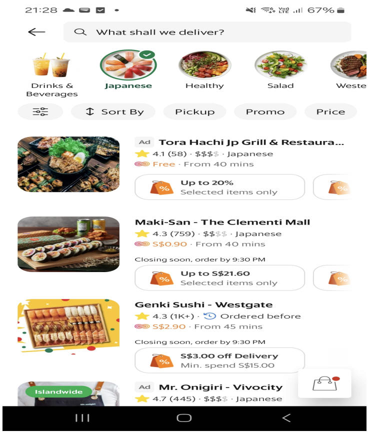
Contextual assortment: the set of restaurants available to each customer when they made their choice.

Estimate: Use Maximum Likelihood Estimation (MLE) or other method to estimate parameters.

Choice Modeling Application: Assortment Optimization

Platform's strategy:

- **Estimate:** Estimate choice model from data.
- **Assortment Optimization:** Choose assortment based on utilities (e.g., select the N restaurants with the highest predicted utilities). Not optimal probably (why?).. see Multi Armed Bandits next lecture.



Ad Tora Hachi Jp Grill & Restaura...
4.1 (58) · $$$$ · Japanese
Free · From 40 mins
Up to 20%
Selected items only

Maki-San - The Clementi Mall
4.3 (759) · $$$$ · Japanese
S$0.90 · From 40 mins
Closing soon, order by 9:30 PM
Up to S$21.60
Selected items only

Genki Sushi - Westgate
4.3 (1K+) · Ordered before
S$2.90 · From 45 mins
Closing soon, order by 9:30 PM
S$3.00 off Delivery
Min. spend S$15.00

Ad Mr. Onigiri - Vivocity
4.7 (445) · $$$$ · Japanese

Exercise: From Data to Assortment

The Data Context

- **Observations:** We have $N = 5,000$ historical choice events. Assume that the agent sees **4 Restaurants** chosen from a larger catalog each time for simplicity.
- **Agents:** In practice, users are heterogeneous. We group them into **10 Representative Types** (e.g., using K-Means clustering on past behavior).

  Why clustering?

Model Assumption: Multinomial Logit (MNL)

- Utility of Restaurant j: $V_{ij} = \beta_p P_j + \beta_q Q_j + \gamma T_j + \theta f_{ij}$
- Utility of Outside Option (None): $V_{i0} = 0$ (Reference Level)

Choice Probabilities Model (Softmax)

$$P(\text{Choice} = j) = \frac{e^{V_{ij}}}{1 + \sum_{k=1}^{4} e^{V_{ik}}}$$

Note: The '1' in the denominator represents $e^{V_{i0}} = e^0 = 1$.

Exercise: From Data to Assortment

Estimation: Maximum Likelihood (MLE)

- **Idea:** We search for the parameter values $\alpha = (\beta_p, \beta_q, \gamma, \theta)$ that make the *observed* customer choices most probable.
- **Likelihood Function:** We maximize the sum of log-probabilities for all N observed choices y_n:

$$\hat{\alpha}_{MLE} = \text{argmax}_\alpha \sum_{n=1}^{N} \ln \left(\frac{e^{V_{y_n}}}{1 + \sum_{j \in \text{Menu}_n} e^{V_j}} \right)$$

  The (log)-likelihood is the sum, over all choice events, of the log of the model's predicted probability of the option that was actually chosen, and we estimate the parameters by choosing the values that make those chosen-option probabilities as large as possible.
- **Tractability:** This is known to be tractable computationally even for big data (convex optimization).

Exercise: From Data to Assortment

Application: Assortment Optimization

- Using estimated parameters, we predict utilities for the **entire** catalog (including items not seen before).
  Possible strategy: for each customer type i, select the top 4 restaurants with the highest predicted utilities.
- Data-driven assortment optimization: User arrives $\rightarrow$ Classified to Type $\rightarrow$ Shown the optimal model-based assortment.

Choice Modeling: Vertical Differentiation Again

Setup:

- Utility model: $u = vq - p$. v: Buyer valuation (exponential distribution $f(v) = exp(-v)$).
- Low-quality: $q_L = 1$, $p_L = 1$. High-quality: $q_H = 2$, $p_H = 4$.

Utilities:

$$\begin{aligned} u_0 &= 0 \quad \text{(Not buying)} \\ u_L &= v \cdot q_L - p_L = v - 1 \quad \text{(Low quality)} \\ u_H &= v \cdot q_H - p_H = 2v - 4 \quad \text{(High quality)} \end{aligned}$$

Can you find the demand in Python?

Practical Demand Estimation Model

Utility Framework:

$$u = vq - p + \beta_{\text{public}} x_{\text{public}} + \beta_{\text{private}} x_{\text{private}} + \epsilon$$

- v: Buyer valuation (random variable)
- x_{public}: Public contextual features (e.g., location, time, promotions)
- x_{private}: Private contextual features (e.g., past experiences)
- $\beta_{\text{public}}, \beta_{\text{private}}$: Coefficients to be estimated from data
- ϵ: Random (e.g., capturing unobserved preferences)

Differentiation:

- **Vertical Differentiation:** $vq - p$ captures differences in perceived product quality (q) and willingness to pay (v).
- **Horizontal Differentiation:** $\beta_{\text{private}} x_{\text{private}} + \epsilon$ introduces randomness and individual-specific preferences.

Choice Modeling: Discussion on “Black-Box” Methods

Recall the utility model for demand for restaurants:

$$U_{ij} = \beta_p \cdot \text{price}_j + \beta_q \cdot \text{quality}_j + \gamma \cdot \text{delivery time}_j + \boldsymbol{\theta}^T \boldsymbol{f}_i + \epsilon_{ij}$$

Would you prefer to use a “black-box” neural network / random forest model instead of a structural form model to improve predictions?

Interpretability: The structural model explicitly defines how price, quality, delivery time, and agent-specific features influence choices.

Limited engineering resources: Structural models are simpler and computationally cheaper.

Data Constraints: If the dataset is small, the predictive power of black-box methods may be limited.

4. Surge Pricing and Behavior of Drivers

Drivers in Ride-Hailing Marketplaces:

- Make decisions about whether to accept or reject trip requests.
- Aim to maximize their earnings over time.

Strategic Behavior:

- Reject short trips during surge to wait for higher-paying long trips.
- Avoid long trips in non-surge periods to become available during the next surge.
- Such strategies can disrupt platform operations.

Surge Pricing

MARINA
RUSSIAN HILL
101
CHINATOWN
NOB HILL
California St
$3.40
Bush St
Geary St
UNION SQUARE
WESTERN ADDITION
SAN FRANCISCO
$7.80
6th St
80
Oak St
HAIGHT ASHBURY
Market St
280
16th St
The Embarca

Surge Pricing Mechanisms

Dynamic Pricing in Ride-Hailing:

- Platforms use surge pricing to balance supply (drivers) and demand (riders).
- Two common mechanisms:
  - **Multiplicative Surge:** Payout scales with trip length ($m \times$ base).
  - **Additive Surge:** Fixed payout added to the base fare ($base + a$).
- Multiplicative pricing benefits long trips but undervalues short trips.

Paper: "Driver Surge Pricing"

Evidence from Uber:

- Multiplicative Surge:
  - Encourages drivers to reject short trips during surge.
  - Incentivizes long-trip rejection in non-surge periods, waiting for surge.
- Additive Surge:
  - Increases trip acceptance rates for both short and long trips.

Impact on Marketplace:

- Additive surge reduces "cherry-picking" behavior.
- Multiplicative surge, historically the standard on ride-hailing platforms, was replaced by an additive surge mechanism by Uber.

Takeaways and Questions

Design Implications:

- Pricing models should account for driver temporal and spatial opportunity costs.

Do you foresee any issues with these surge pricing mechanisms in terms of balancing supply and demand?

Drivers complain on ghost surges..

Ride-hailing platforms use nudging instead of heat-maps. Trade-off?

What other aspects of driver behavior should a platform take into account?

Paper: “Algorithm Aversion: Evidence from Ridesharing Drivers”

Key Empirical Findings:

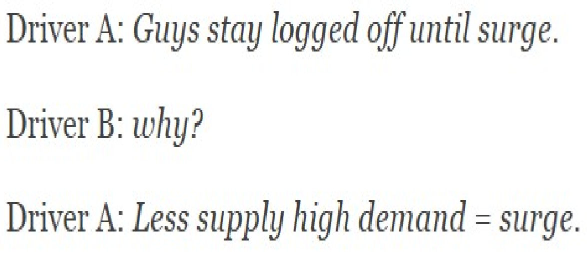
- Drivers show aversion to algorithmic recommendations even when designed to maximize system-wide efficiency.
- Two main drivers of algorithm aversion:
  - **Contextual Experience:** Drivers prefer their own past experience over algorithmic suggestions.
  - **Herding Behavior:** Drivers follow peers’ actions, relying on informal communication to infer demand.

Empirical Evidence

Findings:

- Drivers are less likely to follow recommendations when:
  - Their past experience contradicts the algorithm's suggestion.
  - Peers choose to remain at the current location instead of moving as recommended.
- Over time, drivers' algorithm adoption improves with:
  - Positive personal experience with the algorithm.
  - Increased familiarity and trust in the algorithm's performance.

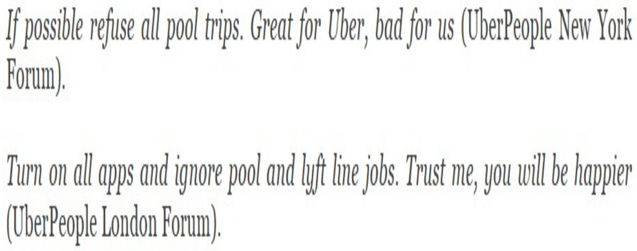
Drivers are Strategic



Driver A: *Guys stay logged off until surge.*

Driver B: *why?*

Driver A: *Less supply high demand = surge.*



If possible refuse all pool trips. Great for Uber, bad for us (UberPeople New York Forum).

Turn on all apps and ignore pool and lyft line jobs. Trust me, you will be happier (UberPeople London Forum).

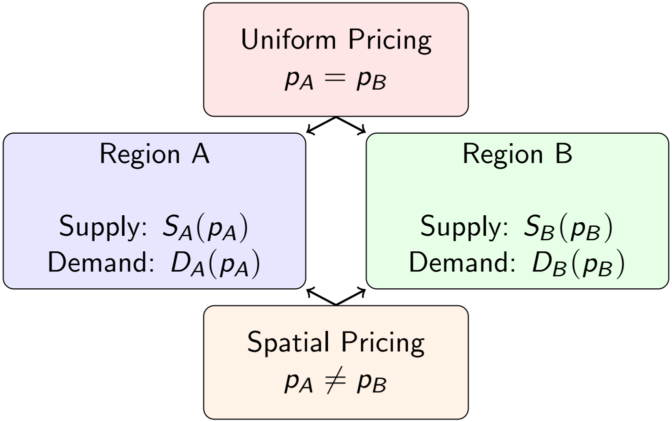
Managerial Insights

Recommendations for Platforms:

- Personalize Recommendations:
  - Align suggestions with drivers' contextual experiences to increase acceptance.
- Address Herding:
  - Educate drivers about system-wide benefits of the algorithm.
  - Use targeted incentives to encourage exploration and adoption.
- Leverage Positive Feedback Loops:
  - Showcase algorithm success stories to build trust.
  - Gradually increase reliance on algorithmic decisions as adoption grows.

5. Spatial Pricing

- **Spatial price discrimination**: ridesharing platforms set different prices for rides originating from various locations.



Imbalances

- Demand patterns are imbalanced in cities—for example:
- **Morning:** Suburban areas tend to have an excess supply of drivers after riders travel downtown for work.

- Spatial pricing can improve profit and welfare by balancing demand and supply across locations.
- Why other marketplaces cannot use spatial pricing (e.g., e-commerce)?

Public backlash. Legal risks. Market structure (competition).

Challenges with Spatial Pricing

Good Machine Learning is Critical

- Accurate demand and supply forecasts are required for each region.
- Platforms need to predict rider elasticity to price changes (how demand responds to pricing adjustments).

Algorithmic Challenges:

- Real-time optimization of prices across multiple regions.
- Avoiding unintended consequences, such as driver gaming or strategic positioning.

Summary: Role of Data Science in Ridesharing

Key Teams and Their Applications:

1. **Machine Learning (ML) Teams:**
   - Predict demand and supply patterns using historical data.
2. **Matching Teams:**
   - Develop algorithms to pair drivers with riders efficiently.
   - Solve large-scale optimization problems in real-time.
3. **Pricing Teams:**
   - Design dynamic pricing algorithms to balance demand and supply.
   - Implement spatial pricing algorithms.

Key Takeaway:

- Investing in data science and optimization capabilities is crucial for success of an online marketplace!

Discussion Question

Uber and the Threat of Disruption

- Uber has experienced a decline in stock price in Q4 2024.
- Key reasons cited include:
  - The rise of autonomous vehicle companies like Google's Waymo and Tesla's Robotaxi.
  - Advancements in AI potentially disrupting traditional ride-sharing operations.

Thoughts ?