# Food Delivery Dispatch System

# Overview

This simulation models a food delivery platform (similar to UberEats, PickMe, DoorDash, or Deliveroo) to analyze dispatch strategies and fleet management performance.

# System Description

### Components

1. **Orders**: Customer food orders with restaurant and delivery locations
2. **Drivers**: Fleet of delivery personnel who pick up and deliver orders
3. **Dispatch System**: Assigns orders to drivers using different strategies
4. **Service Area**: 10km × 10km geographic region

### Key Features

* **Realistic order generation**: Poisson arrival process
* **Multiple dispatch strategies**: Nearest, FCFS, Balanced
* **Geographic simulation**: 2D coordinates for restaurants, customers, drivers
* **Time-based metrics**: Wait times, delivery times, driver utilization
* **Workload balancing**: Fair distribution across drivers

## Installation & Requirements

### Required Libraries

pip install numpy matplotlib pandas seaborn

### Python Version

Python 3.7 or higher

## Usage Instructions

### Quick Start

python food\_delivery\_simulation.py

This will run all 5 pre-configured scenarios and generate visualizations.

### Running Custom Scenarios

from food\_delivery\_simulation import \*

*# Create custom simulation*

system = run\_simulation(

num\_drivers=12, *# Number of drivers*

dispatch\_strategy=DispatchStrategy.NEAREST, *# Assignment strategy*

duration=480, *# 8 hours in minutes*

order\_rate=0.5, *# Orders per minute (30/hour)*

service\_area\_size=10.0, *# 10km × 10km*

avg\_speed=0.5 *# 30 km/hour*

)

*# Analyze results*

results = analyze\_results(system, "My Custom Scenario")

*# Visualize*

plot\_driver\_distribution(system, "My Custom Scenario")

### Dispatch Strategies

1. **NEAREST** (DispatchStrategy.NEAREST)
   * Assigns orders to the driver closest to the restaurant
   * Minimizes travel distance and delivery time
   * Best for customer satisfaction
2. **FCFS** (DispatchStrategy.FCFS)
   * First-Come-First-Served: assigns to first available driver
   * Simplest implementation
   * May result in longer delivery distances
3. **BALANCED** (DispatchStrategy.BALANCED)
   * Distributes workload evenly across drivers
   * Prevents driver burnout
   * May sacrifice some delivery time efficiency

## Scenarios Explained

### Scenario 1: Normal Demand - Nearest

* **Purpose**: Baseline performance with optimal strategy
* **Parameters**: 10 drivers, 24 orders/hour, nearest dispatch
* **Expected**: Good delivery times, balanced workload

### Scenario 2: Normal Demand - FCFS

* **Purpose**: Compare simpler dispatch strategy
* **Parameters**: 10 drivers, 24 orders/hour, FCFS dispatch
* **Expected**: Similar to Scenario 1 but potentially higher variance

### Scenario 3: Normal Demand - Balanced

* **Purpose**: Test workload balancing strategy
* **Parameters**: 10 drivers, 24 orders/hour, balanced dispatch
* **Expected**: Most even distribution, slightly longer delivery times

### Scenario 4: Peak Demand - Understaffed

* **Purpose**: Analyse system under stress
* **Parameters**: 10 drivers, 42 orders/hour (high demand)
* **Expected**: Long wait times, high driver utilization, pending orders

### Scenario 5: Peak Demand - Well-Staffed

* **Purpose**: Show proper capacity planning
* **Parameters**: 15 drivers, 42 orders/hour
* **Expected**: Improved metrics compared to Scenario 4

## Performance Metrics

### Time Metrics

* **Wait Time**: Time from order placement to driver assignment
* **Delivery Time**: Total time from order to delivery completion
* **Preparation Time**: Food cooking time at restaurant (15 min average)

### Driver Metrics

* **Utilization Rate**: Percentage of time drivers are busy
* **Orders per Driver**: Total orders completed by each driver
* **Distance Traveled**: Total kilometers driven per driver
* **Workload Standard Deviation**: Measure of workload balance (lower = more balanced)

### System Metrics

* **Completion Rate**: Percentage of orders completed
* **Pending Orders**: Orders waiting for drivers at simulation end
* **Throughput**: Total orders processed per hour

## Output Files

### Generated Visualizations

1. **delivery\_comparison.png**: 4-panel comparison chart
   * Average delivery time across scenarios
   * Wait time vs total delivery time
   * Driver utilization rates
   * Workload balance (standard deviation)
2. **driver\_distribution\_[scenario].png**: Driver workload analysis
   * Orders completed per driver
   * Distance traveled per driver

### Console Output

* Detailed metrics for each scenario
* Performance summary table
* Key insights and recommendations

## Interpreting Results

### Good Performance Indicators

* Average delivery time < 30 minutes
* Driver utilization 60-80% (sustainable)
* Completion rate > 95%
* Low workload standard deviation (< 3 orders)
* Minimal pending orders

### Warning Signs

* Driver utilization > 85% (burnout risk)
* Completion rate < 90% (capacity issues)
* High workload variance (unfair distribution)
* Wait times > 10 minutes

## Customization Guide

### Adjusting System Parameters

*# Modify simulation settings*

system = run\_simulation(

num\_drivers=20, *# Scale fleet size*

dispatch\_strategy=DispatchStrategy.BALANCED,

duration=720, *# Longer simulation (12 hours)*

order\_rate=0.6, *# Adjust demand (36/hour)*

service\_area\_size=15.0, *# Larger city (15km × 15km)*

avg\_speed=0.6 *# Faster drivers (36 km/hour)*

)

### Creating New Scenarios

*# Rush hour scenario*

rush\_hour\_sim = run\_simulation(

num\_drivers=20,

dispatch\_strategy=DispatchStrategy.NEAREST,

duration=120, *# 2-hour rush period*

order\_rate=1.0, *# 60 orders/hour!*

service\_area\_size=10.0,

avg\_speed=0.4 *# Traffic slows drivers*

)

*# Late night scenario*

late\_night\_sim = run\_simulation(

num\_drivers=5, *# Fewer drivers at night*

dispatch\_strategy=DispatchStrategy.NEAREST,

duration=240, *# 4 hours*

order\_rate=0.15, *# 9 orders/hour (quiet)*

service\_area\_size=12.0, *# Larger area (spread out)*

avg\_speed=0.7 *# Less traffic, faster*

)

### Testing Different Order Patterns

*# Modify generate\_orders function for custom patterns*

def generate\_lunch\_rush\_orders(duration=120):

"""Generate orders with lunch rush peak"""

orders = []

for t in range(duration):

*# Peak between 12:00-13:00*

if 30 <= t <= 90: *# Minutes 30-90*

rate = 1.2 *# High rate*

else:

rate = 0.3 *# Normal rate*

if np.random.random() < rate:

*# Create order (similar to generate\_orders)*

pass

return orders

## Case Study Report Structure

### Suggested Report Outline

1. **Introduction** (1 page)
   * Background on food delivery industry
   * Problem statement
   * Objectives of the study
2. **System Description** (1 page)
   * Components and workflow
   * Stakeholders
   * Performance requirements
3. **Methodology** (1-2 pages)
   * Simulation model design
   * Parameters and assumptions
   * Dispatch strategies explained
   * Validation approach
4. **Experimental Design** (1 page)
   * Scenario descriptions
   * Rationale for each test
   * Parameter values
5. **Results** (2-3 pages)
   * Performance metrics tables
   * Visualizations (3+ charts)
   * Comparative analysis
6. **Discussion** (1-2 pages)
   * Key findings
   * Trade-offs between strategies
   * Recommendations for operators
   * Limitations
7. **Conclusion** (0.5 page)
   * Summary of insights
   * Future improvements
8. **Appendix** (separate)
   * Full code listing
   * Sample output data
   * Additional visualizations

## Key Findings to Highlight

### Strategy Comparison

* **Nearest**: Best delivery times, moderate balance
* **FCFS**: Simplest, but inconsistent performance
* **Balanced**: Fairest to drivers, slightly slower deliveries

### Capacity Planning

* 10 drivers handle ~24 orders/hour comfortably
* Peak demand (42 orders/hour) requires 15+ drivers
* Understaffing causes exponential wait time increases

### Driver Utilization

* Target: 65-75% for sustainable operations
* Below 50%: Overstaffed (inefficient)
* Above 85%: Burnout risk, quality issues

## Troubleshooting

### No orders completed

* **Cause**: Too few drivers or too high demand
* **Solution**: Increase num\_drivers or decrease order\_rate

### All drivers equally utilized

* **Expected**: With BALANCED strategy
* **Not a bug**: This is the intended behavior

### Visualizations not showing

* **Cause**: Missing matplotlib backend
* **Solution**: Add plt.show() or save files with plt.savefig()

### Very long wait times

* **Cause**: System overload (demand > capacity)
* **Solution**: Increase drivers or reduce order rate

## Advanced Modifications

### Adding Multiple Order Types

@dataclass

class Order:

*# Add priority field*

priority: str = "standard" *# "express" or "standard"*

### Implementing Batch Deliveries

class Driver:

*# Allow multiple orders*

current\_orders: List[Order] = []

max\_orders: int = 3 *# Multi-order capacity*

### Adding Traffic Patterns

def get\_speed\_factor(time\_of\_day):

"""Slow down during rush hours"""

if 12 <= time\_of\_day <= 13 or 18 <= time\_of\_day <= 20:

return 0.6 *# 60% normal speed*

return 1.0

### Dynamic Driver Availability

def update\_fleet\_size(current\_time):

"""Adjust drivers based on time"""

if 11 <= current\_time <= 14: *# Lunch*

return 15

elif 17 <= current\_time <= 21: *# Dinner*

return 20

else:

return 8 *# Off-peak*

## Expected Output Example

======================================================================

Results for: Scenario 1: Normal Demand - Nearest Driver

======================================================================

Strategy: NEAREST

Number of drivers: 10

Total orders received: 192

Orders completed: 189

Orders pending: 3

Completion rate: 98.4%

--- Time Metrics ---

Average wait time: 2.34 minutes

Maximum wait time: 8.21 minutes

Average delivery time: 24.67 minutes

Maximum delivery time: 42.15 minutes

--- Driver Metrics ---

Average orders per driver: 18.9

Std dev orders per driver: 2.14

Average distance per driver: 67.32 km

Average driver utilization: 68.5%

## References and Resources

### Similar Real-World Systems

* UberEats dispatch algorithm
* DoorDash batching system
* Deliveroo rider allocation

### Academic Resources

* Queueing theory for service systems
* Vehicle routing problem (VRP)
* Multi-agent dispatch optimization

### Further Reading

* "The Vehicle Routing Problem" - Toth & Vigo
* "Queueing Systems" - Leonard Kleinrock
* Research on gig economy logistics

## Support

For questions or issues:

1. Check this README thoroughly
2. Review code comments
3. Experiment with different parameter values
4. Compare your results with expected outputs

## License

Educational use - Case Study Project 2025

**Last Updated**: October 2025  
**Version**: 1.0