CS/ECE/ISyE 524 — Introduction to Optimization — Spring 2023

Final Course Project: Due 5/5/23

Chicago Divvy Bike Supply & Station Management

Stephen Ling (jling9@wisc.edu)

Hongtao Zhang (hzhang784@wisc.edu)

Table of Contents

---

1. Introduction

The station-based bike rental system, which allows people to rent bikes from designated locations, or stations, and return them to another station in a different location, has become an increasingly popular mode of transportation in cities around the world. The Chicago Department of Transportation (CDOT) recently announced that the Divvy Bikeshare System has become the biggest bikeshare in North America by service area (Brewster, 2023). This expansion demonstrates the profitability and potential of the market for station-based bike rentals.

Optimization modeling has been widely used to improve the efficiency and profitability of bike-sharing systems. The modeling typically consists of two parts: strategic planning, which involves route design, facility location, and facility capacity allocation, and operational planning, which involves bike positioning and bike repositioning (Nath & Rambha, 2019). In the beginning, researchers mainly focusing minimizing the cost of transporting bikes between locations (Yan et al., 2017) and rebalancing bike storage between different stations and time period (Paul & Zhang, 2017). However, in recent years, researchers have begun to incorporate environmental criteria like CO2 emmission (Wang & Szeto, 2018) and social welfare criteria (Qian et al., 2022) into station location optimization.

This project focuses on the facility location problem in strategic planning and the bike positioning problem in operational planning but has a much simpler objective function comparing with models used in relevant research. Specifically, we aim to explore how **bike supply arrangement**, **station location**, and **pricing strategy** can affect the revenue of a station-based bike rental firm, using data from the Divvy Bikeshare System. Our goal is to determine the optimal bike supply arrangement, station location, and pricing strategy that can **maximize revenue**.

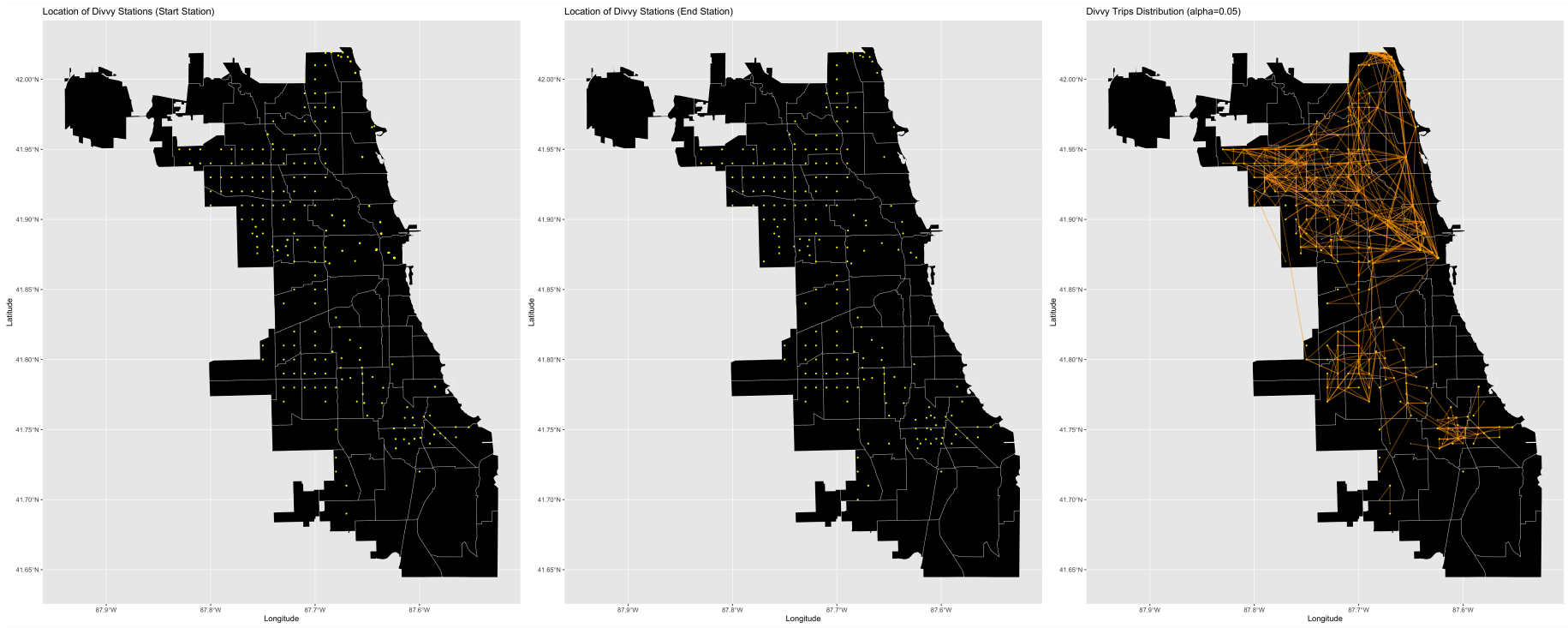
Our data is composed of two parts. The first part is the Trips data, which contains information on all Divvy Bike trips in May 2022, including start and end times, start and end station IDs, and trip durations. The second part is the Stations data, which contains the unique station IDs and the maximum number of bikes that can be parked in each station.

In the first section of our report, we will outline the mathematical models (IP, MIP) we use and demonstrate the network model with graphs. In the second section, we will provide the code implementation of each mathematical model. Finally, we

will visualize and discuss the results in the last section.

In summary, this project aims to optimize station-based bike rental systems by maximizing revenue. Our findings could have implications for the future development and management of bike-sharing systems in urban areas.

The image below visualize the distribution of Divvy Bike Stations and Divvy Trips between these stations in May 2022:



---

2. Mathematical Model

(a) Model 1: Bike Supply Model (IP)

Assumption

- 1 bike could be only used 1 time in an hour.
- A new bike could be used for 24 months (2 years).
- The station capacity is the number of docks in a station.
- The flow capacity from station i to station j will not exceed the flow from i to station j in May 2022
- Stations are already constructed so that we do not need to pay extra fee to construct a station.
- No maintainance fee for either station or bike.

Parameters:

- c_b: cost of a bike ($1000)
- l_b: lifetime of a bike (in terms of month) (24 months)
- s_j: the dock size of station j (maximum number of bikes could be stored in station j)
- r_{activate}: revenue of unlocking a bike ($1.0)
- r_{ride}: revenue of one minute ride ($0.17/min)
- $t_{i,j,k}$: the average length of trip in hour i from station j to station k
- $g_{i,j,k}$: the maximum flow (number of bikes) in hour i from station j to station k

Variables

- $b_{i,j}$: An integer variable representing the number of bike of Station j at Hour i. ($b \in \mathbb{N}$)
- $f_{i,j,k}$: the actual flow in hour i from station j to station k. ($f \in \mathbb{N}$)

Integer Program (IP) Model:

$$\max_{b_{1,j}} \sum_{i=1}^{24}\sum_{j}\sum_{k}(f_{i,j,k}\cdot r_{\text{activate}} + f_{i,j,k}\cdot t_{i,j,k}\cdot r_{\text{ride}}) - \sum_{i}(c_b/l_b)\cdot b_{1,i}$$

$s.t.$

$$\text{Station Capacity Limit:} \qquad b_{i,j} \le s_j \qquad \forall i,j$$

$$\text{Capacity Constraints} \qquad f_{i,j,k} \le g_{i,j,k} \qquad \forall i \in [1,24], j, k$$

$$\text{Conservation:} \qquad b_{i,j} = b_{i-1,j} + \sum_{k}(f_{i-1,k,j} - f_{i-1,j,k}) \qquad \forall i \in [2,24], j$$

(b) Model 2: Bike Supply & Station Management Model (IP)

Assumption

- 1 bike could be only used 1 time in an hour.
- A new bike could be used for 24 months (2 years).
- A new station could be used for 60 months (5 years).
- The construction cost of each station is fixed.
- The station capacity is the number of docks in a station.
- The flow capacity from station i to station j will not exceed the flow from i to station j in May 2022
- No maintainance fee for either station or bike.

Parameters:

- c_b: cost of a bike ($1000)
- c_s: cost of a station ($10000)
- l_b: lifetime of a bike (in terms of month) (24 months)
- l_s: lifetime of a station (in terms of month) (60 months)
- s_j: the dock size of station j (maximum number of bikes could be stored in station j)
- r_{activate}: revenue of unlocking a bike ($1.0)
- r_{ride}: revenue of one minute ride ($0.17/min)
- $t_{i,j,k}$: the average length of trip in hour i from station j to station k
- $g_{i,j,k}$: the maximum flow (number of bikes) in hour i from station j to station k

Variables

- $b_{i,j}$: An integer variable representing the number of bike of Station j at Hour i. ($b \in \mathbb{N}$)
- $f_{i,j,k}$: the actual flow in hour i from station j to station k. ($f \in \mathbb{N}$)
- z_i: A binary variable (0,1) determines whether station i is built. ($z \in \{0,1\}$)

Integer Program (IP) Model:

$$\max_{b_{1,j}} \sum_{i=1}^{24}\sum_{j}\sum_{k}(f_{i,j,k}\cdot r_{\text{activate}} + f_{i,j,k}\cdot t_{i,j,k}\cdot r_{\text{ride}}) - \sum_{i}(c_b/l_b)\cdot b_{1,i} - \sum_{i}(c_s/l_s)\cdot z_i$$

$s.t:$

$$\text{Station Capacity Limit:} \qquad b_{i,j} \le s_j \cdot z_j \qquad \forall i,j$$

$$\text{Capacity Constraints} \qquad f_{i,j,k} \le g_{i,j,k} \cdot z_j \qquad \forall i \in [1,24], j, k$$

$$\text{Conservation:} \qquad b_{i,j} = b_{i-1,j} + \sum_{k}(f_{i-1,k,j} - f_{i-1,j,k}) \qquad \forall i \in [2,24], j$$

$$\text{Binary:} \qquad z_i \in \{0,1\} \qquad \forall i$$

(c) Model 3: Bike Supply, Pricing & Station Management Model (MIP)

Assumption

- 1 bike could be only used 1 time in an hour.
- A new bike could be used for 24 months (2 years).
- A new station could be used for 60 months (5 years).
- The construction cost of each station is fixed.
- The station capacity is the number of docks in a station.

- The flow capacity from station i to station j will not exceed the flow from i to station j in May 2022
- No maintainance fee for either station or bike.
- Suppose there is another Bike rental firm in Chicago named *Bivvy Bike*, which owns exactly the same bike stations (identical) as *Divvy Bike*. Since 2 firms are identical, and we assume there are no membership users, consumers' choice is only affected by price. We assume **proportion of total demand that *Bivvy Bike* shared** could be formulated as $D_{Bivvy} = \frac{1}{1+\exp(-1+\frac{P_{Bivvy}}{P_{Divvy}})}$.

Parameters:

- c_b: cost of a bike ($1000)
- c_s: cost of a station ($10000)
- l_b: lifetime of a bike (in terms of month) (24 months)
- l_s: lifetime of a station (in terms of month) (60 months)
- s_j: the dock size of station j (maximum number of bikes could be stored in station j)
- r_{activate}: revenue of unlocking a bike ($1.0)
- r_{ride}: revenue of one minute ride ($0.17/min)
- $t_{i,j,k}$: the average length of trip in hour i from station j to station k
- $g_{i,j,k}$: the maximum flow (number of bikes) in hour i from station j to station k

Variables

- $b_{i,j}$: An integer variable representing the number of bike of Station j at Hour i. ($b \in \mathbb{N}$)
- $f_{i,j,k}$: the actual flow in hour i from station j to station k. ($f \in \mathbb{N}$)
- z_i: A binary variable (0,1) determines whether station i is built. ($z \in \{0,1\}$)
- P: the price for one minute ride. ($P \in \mathbb{R}^+$)
- $d = \frac{1}{1+\exp(-1+\frac{P}{r_{ride}})}$: proportion of total demand that *Bivvy Bike* shared.

Mixed Integer Program (MIP) Model:

$$\max_{b_{1,j}} \sum_{i=1}^{24}\sum_{j}\sum_{k}(f_{i,j,k}\cdot r_{\text{activate}} + f_{i,j,k}\cdot t_{i,j,k}\cdot P) - \sum_{i}(c_b/l_b)\cdot b_{1,i} - \sum_{i}(c_s/l_s)\cdot z_i$$

$s.t:$

$$\text{Station Capacity Limit:} \quad b_{i,j} \le s_j \cdot z_j \qquad \forall i,j$$

$$\text{Capacity Constraints} \quad f_{i,j,k} \le g_{i,j,k}\cdot z_j \cdot d \qquad \forall i \in [1,24], j, k$$

$$\text{Conservation:} \quad b_{i,j} = b_{i-1,j} + \sum_{k}(f_{i-1,k,j} - f_{i-1,j,k}) \qquad \forall i \in [2,24], j$$

$$\text{Binary:} \quad z_i \in \{0,1\} \qquad \forall i$$

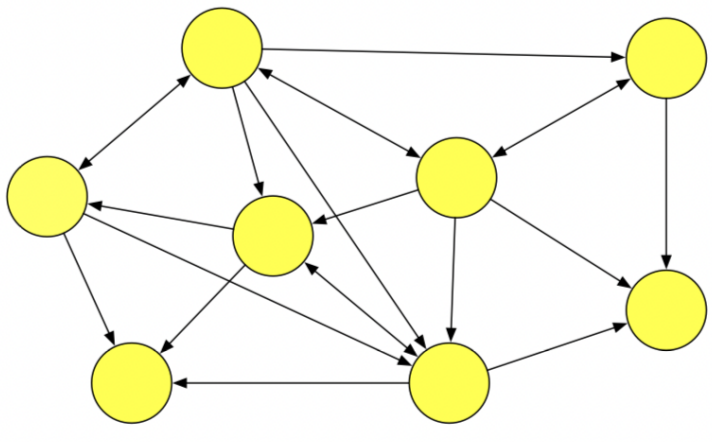
This is a Non-Convex Model

(d) Model Graphing

The following 2 graphs would help to illustrate the Network Flow Model used in this project.

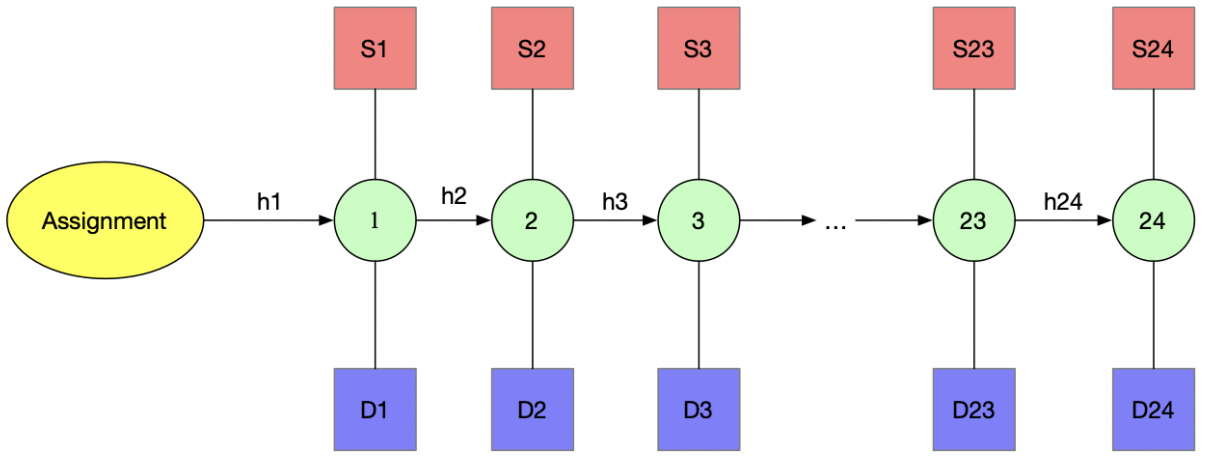
(ii) Graph 1

Graph 1 illustrate the network flow model from the perspective of **multiple stations & 1 hour**. Edges have flow costs and capacity constraints. Each node can supply/consume/conserve a flow by hour:



(i) Graph 2

Graph 2 illustrate the network flow model from the perspective of **a single station & 24 hours**. Each node i indicates hour i. At hour i, there are S_i bikes are returned to the station, and D_i bikes are borrowed from the station. If there are h_i bikes left in station at the end of hour $i-1$, then h_i bikes are passed to hour i, and h_1 indicates the endowment to each station at the beginning of a day.



---

3. Solution

(a) Install & Load Required Packages

- Packages for Data Processing: CSV, DataFrames, DataFramesMeta, Dates, Statistics
- Packages for Optimization Models: JuMP, Gurobi (for IP, MIP), Ipopt (for Non-convex)
- Packages for Visualization: PrettyTables, StatsPlots, Plots

In [ ]:
```julia
using Pkg
Pkg.add("CSV")
Pkg.add("DataFrames")
Pkg.add("DataFramesMeta")
Pkg.add("StatsBase")
Pkg.add("PrettyTables")
Pkg.add("Alpine")
Pkg.add("Plots")
Pkg.add("StatsPlots")
```

In [ ]:
```julia
using CSV, DataFrames, DataFramesMeta, Dates, Statistics, JuMP, Ipopt, Gurobi, PrettyTables, StatsPlots,
```

(b) Processing Data

(i) Read & Clean Data

In [ ]:
```julia
# Data for ALL
trips = CSV.read("202205-divvy-tripdata.csv", DataFrame)
stations = CSV.read("Divvy_Bicycle_Stations.csv", DataFrame)

# rename upper case sepearated by space to lower case separated by underscore
function snake_case(camelstring::S) where S<:AbstractString
    wordpat = r"
    ^[a-z]+ |                   #match initial lower case part
    [A-Z][a-z]+ |               #match Words Like This
    \d*([A-Z](?=[A-Z]|$))+ |    #match ABBREV 30MW
    \d+                         #match 1234 (numbers without units)
    "x
    smartlower(word) = any(islowercase, word) ? lowercase(word) : word
    words = [smartlower(m.match) for m in eachmatch(wordpat, camelstring)]
    join(words, "_")
end
for name in names(stations)
    rename!(stations, name => snake_case(name))
end
rename!(stations, "ID" => "id")
@transform!(stations, :id = string.(stations.id))

# handling missing values, keeping the valid records
```

```
trips = @orderby(trips, :start_station_id)

dropmissing!(trips)

trips = trips[findall(x->insorted(x, trips.start_station_id), trips.end_station_id), :]

trips = @chain trips begin
    innerjoin(stations, on = [:start_station_id => :id])
end

# transform time string to date format
date_format = dateformat"yyyy-mm-dd HH:MM:SS"
@transform!(trips, :started_at = DateTime.(trips.started_at, date_format))
@transform!(trips, :ended_at = DateTime.(trips.ended_at, date_format))
@transform!(trips, :trip_length = trips.ended_at - trips.started_at);

pretty_table(first(trips, 5))
pretty_table(first(stations, 5))
```

(i) Preparing Data for `MODEL 1`

The data we have is the data for each trip and data for each station. However, to implement Model 2, we need a data contains following variables:

- `Hour` : The hour of day ranges from 1 to 24.
- `Start Station` : The start station id.
- `End Station` : The end station id.
- `n` : number of trips from "start station" to "end station" at "hour".
- `Average Trip Duration` : Average trip length for trip from "start station" to "end station" at "hour".

In [ ]:
```
# Based on scope (hour) assumption, extract hour for each trip
@transform! trips begin
    :hour = hour.(trips.ended_at)
end

# Create a basic data frame for Model 2 & 3:
# vars: start station id, end station id, count, average trip duration, docks in service
base_data = @chain trips begin
    groupby([:start_station_id, :hour, :end_station_id])
    @combine begin
        :n = length(:ride_id)
        :avg_duration = @chain :trip_length begin
            round.(Dates.Minute)
            Dates.value.(_)
            mean()
        end
        :start_station_name = first(:start_station_name)
        :end_station_name = first(:end_station_name)
        :start_lat = first(:start_lat)
        :end_lat = first(:end_lat)
        :start_lng = first(:start_lng)
        :end_lng = first(:end_lng)
        :docks_service = first(:docks_service)
    end
    @orderby(-:n)
end

# Some station id are not numbers, let's map it into integer
# This could make coding of model more convinent
station_id = unique(intersect(base_data.start_station_id, base_data.end_station_id))
station_id_map = Dict(zip(station_id, 1:length(station_id)))

# Construct the data for model by generatig useful data from basic data frame
data = @chain base_data begin
    @transform(:start_station_id = get.(Ref(station_id_map), :start_station_id, 0),
               :end_station_id = get.(Ref(station_id_map), :end_station_id, 0))
    groupby([:start_station_id, :hour, :end_station_id])
    @combine begin
        :n = sum(:n)
        :avg_duration = mean(:avg_duration)
        :start_station_name = first(:start_station_name)
        :end_station_name = first(:end_station_name)
        :start_lat = first(:start_lat)
        :end_lat = first(:end_lat)
```

```
        :start_lng = first(:start_lng)
        :end_lng = first(:end_lng)
        :docks_service = first(:docks_service)
    end
    @orderby(:start_station_id)
end

# Extract useful data to build a dictionary based on station id
# (avoid associating a station with wrong data)
start_hour_dict = Dict((data.hour .=> data.start_station_id .=> data.end_station_id) .=> zip(data.n, dat

# create data to print the model output
visual_data = @chain data begin
    @select(:start_station_id, :start_station_name, :start_lat, :start_lng, :docks_service)
    @distinct(:start_station_id)
    @orderby(:start_station_id)
end

pretty_table(first(data, 5)) # show first 5 records of the data
```

(iii) Preparing Data for `MODEL 2`

The Model 3 is Model 2 adding the logic constraints. So, all data we need for Model 3 is the data from Model 2 and data for the logical constraints:

- `Hour` : The hour of day ranges from 1 to 24.
- `Start Station` : The start station id.
- `End Station` : The end station id.
- `n` : number of trips from "start station" to "end station" at "hour".
- `Average Trip Duration` : Average trip length for trip from "start station" to "end station" at "hour".
- `Dock in Service` : Number of docks (place to park bike) for each station.

In [ ]:
```
# create a dictionary that map station id into number of docks in service for that station
service_dict = Dict(data.start_station_id .=> data.docks_service)
first(service_dict, 5) # show first 5 records of the data
```

(iv) Preparing Data for `MODEL 3`

`Model 3` shares the same data with `Model 2`

(c) Model Implementation

(i) Model 1: Bike Supply Model (IP)

In [ ]:
```
m1 = Model(Gurobi.Optimizer)
set_silent(m1)

c_b = 1000 / 24 # cost of bike / life of bike
r_ride = 0.17 # cost per minute of ride
r_activate = 1 # cost to unlock a bike
N_station = length(station_id) # number of stations

@variable(m1, flow[1:24, 1:N_station, 1:N_station] >= 0)
@variable(m1, 0 <= num_hourly[1:24, 1:N_station])

# dock constraint
@constraint(m1, dock_contraint[i=1:N_station], num_hourly[:, i] .<= service_dict[i])
# conservation constraint
@constraint(m1, conservation[i=2:24, j=1:N_station], num_hourly[i, j] == num_hourly[i-1, j] .+ sum(flow[
# flow capacity constraint
@constraint(m1, flow_capacity[i=1:24, j=1:N_station], sum(flow[i, j, :]) <= num_hourly[i, j])
@constraint(m1, flow_capacity2[i=1:24, j=1:N_station, k=1:N_station], sum(flow[i, j, k]) <= get(start_ho

@expression(m1, revenue[i=1:24, j=1:N_station, k=1:N_station], flow[i, j, k] * r_activate + flow[i, j, k
@objective(m1, Max, sum(revenue) - sum(num_hourly[1, :]) * c_b) # maximize total revenue

optimize!(m1)
```

In [ ]:
```
using DataFrames, DataFramesMeta, PrettyTables, Printf

# Define a function to print results
```

```
function pretty_print_result(bikes_num)
    result = @chain bikes_num begin
        value.()
        enumerate()
        map(((i, x), ) -> hcat(i, x, visual_data[i, :].start_station_name, visual_data[i, :].start_lat,
        reduce(vcat, _)
        DataFrame([:station_id, :initial_bike, :station_name, :station_lat, :station_lng])
        @orderby(-:initial_bike)
        first(15)
    end

    pretty_table(result)
end

pretty_print_result(value.(num_hourly[1, :]))
@printf("Total revenue: \$%.2f\n", objective_value(m1))
sum(value.(num_hourly[1, :]))
```

(ii) Model 2: Bike Supply & Station Management Model (IP)

In [ ]:
```
m2 = Model(Gurobi.Optimizer)
set_silent(m2)

c_b = 1000 / 24 # cost of bike / life of bike (24 months)
c_station = 10000 / 60 # cost of station / life of station (60 months)
r_ride = 0.17 # cost per minute of ride
r_activate = 1 # cost to unlock a bike
N_station = length(station_id) # number of stations

@variable(m2, flow[1:24, 1:N_station, 1:N_station] >= 0)
@variable(m2, 0 <= num_hourly[1:24, 1:N_station])
@variable(m2, build[1:N_station], Bin) # binary variable (0,1) indicate whether station is in service

@constraint(m2, valid[i=1:N_station], num_hourly[:, i] .<= service_dict[i] * build[i]) # station can onl
@constraint(m2, conservation[i=2:24, j=1:N_station], num_hourly[i, j] == num_hourly[i-1, j] .+ sum(flow[
@constraint(m2, flow_capacity[i=1:24, j=1:N_station], sum(flow[i, j, :]) <= num_hourly[i, j] * build[j])
@constraint(m2, flow_capacity2[i=1:24, j=1:N_station, k=1:N_station], sum(flow[i, j, k]) <= get(start_ho

@expression(m2, revenue[i=1:24, j=1:N_station, k=1:N_station], flow[i, j, k] * r_activate + flow[i, j, k
@objective(m2, Max, sum(revenue) - sum(num_hourly[1, :]) * c_b - sum(build) * c_station)

optimize!(m2)
```

In [ ]:
```
pretty_print_result(value.(num_hourly[1, :]))
@printf("Total revenue: \$%.2f\n", objective_value(m2))
sum(value.(num_hourly[1, :]))
```

(iii) Model 3: Model 3: Bike Supply, Pricing & Station Management Model (MIP)

This chunk tried to use `Alpine` to find an optimal value (this is a non-convex model), but it takes too long & too much memory to obtain the results.

Code could run, but failed to obtain final (aborted) since taking too much computation time & recourses.

In [ ]:
```
# DO NOT RUN FOLLOWING CODE IF USING LAPTOP
using JuMP, Ipopt, Gurobi, Alpine
const gurobi = optimizer_with_attributes(Gurobi.Optimizer, MOI.Silent() => true, "Presolve" => 1)
const ipopt = optimizer_with_attributes(Ipopt.Optimizer, MOI.Silent() => true, "sb" => "yes", "max_iter"
const alpine = optimizer_with_attributes(Alpine.Optimizer, "nlp_solver" => ipopt, "mip_solver" => gurobi

m = Model(alpine)

c_b = 1000 / 24
c_station = 10000 / 60
r_ride = 0.17

r_activate = 1
N_station = length(station_id)

# manual price
# price = 0.19

@variable(m, 0 <= price)

@NLexpression(m, d_prop, 1 / (1 + exp(-1 + price / r_ride)))
```

```
@variable(m, flow[1:24, 1:N_station, 1:N_station] >= 0)
@variable(m, 0 <= num_hourly[1:24, 1:N_station])
@variable(m, build[1:N_station], Bin) # binary variable (0,1) indicate whether station is in service

@constraint(m, valid[i=1:N_station], num_hourly[:, i] .<= service_dict[i] * build[i]) # station can only
@constraint(m, conservation[i=2:24, j=1:N_station], num_hourly[i, j] == num_hourly[i-1, j] .+ sum(flow[i
@NLconstraint(m, flow_capacity[i=1:24, j=1:N_station], sum(flow[i, j, k] for k in 1:N_station) <= num_ho
@constraint(m, flow_capacity2[i=1:24, j=1:N_station, k=1:N_station], sum(flow[i, j, k]) <= get(start_hou

@NLobjective(m, Max, sum(flow[i, j, k] * r_activate + flow[i, j, k] * get(start_hour_dict, i=>j=>k, (0,0

optimize!(m)

pretty_print_result(first(value.(num_hourly[1, :]), 5))
@printf("Total revenue: \$%.2f\n", objective_value(m))
```

This chunk uses a sequence of values for `Price` instead using the price variable. We will plot `Price vs. Total` Revenue to see the influence of Price over total revenue.

In [ ]:
```
# THIS CODE TAKES MORE THAN 1 HOUR
const gurobi = optimizer_with_attributes(Gurobi.Optimizer, MOI.Silent() => true, "Presolve" => 2)

c_b = 1000 / 24
c_station = 10000 / 60
r_ride = 0.17
r_activate = 1
N_station = length(station_id)
price_choice = collect(0.1:0.01:0.40) # sequence of prices

revenues = []

for price in price_choice
    m = Model(gurobi)

    @expression(m, d_prop, 1 / (1 + exp(-1 + price / r_ride)))

    @variable(m, flow[1:24, 1:N_station, 1:N_station] >= 0, Int)
    @variable(m, 0 <= num_hourly[1:24, 1:N_station], Int)
    @variable(m, build[1:N_station], Bin) # binary variable (0,1) indicate whether station is in service

    @constraint(m, valid[i=1:N_station], num_hourly[:, i] .<= service_dict[i] * build[i]) # station can
    @constraint(m, conservation[i=2:24, j=1:N_station], num_hourly[i, j] == num_hourly[i-1, j] .+ sum(fl
    @constraint(m, flow_capacity[i=1:24, j=1:N_station], sum(flow[i, j, k] for k in 1:N_station) <= num_
    @constraint(m, flow_capacity2[i=1:24, j=1:N_station, k=1:N_station], flow[i, j, k] <= get(start_hour


    @expression(m, revenue[i=1:24, j=1:N_station, k=1:N_station], flow[i, j, k] * r_activate + flow[i, j
    @objective(m, Max, sum(revenue) - sum(num_hourly[1, i] for i in 1:N_station) * c_b - sum(build[i] fo
    optimize!(m)

    push!(revenues, objective_value(m))
end

revenues
println("price: ", price_choice[argmax(revenues)])
println("profit: ", maximum(revenues))
```

(d) Results Visualization

The code chunk below try to visualize the Optimal Model's Supply & Demand.

In [ ]:
```
# For model 1
using StatsPlots, Plots

function num_bikes_trend(num_hourly, flow)
    sorted_initial = sortperm(num_hourly[1, :], rev = true)

    # plot the trend of number of bikes in the station with the most bikes
    plots = []

    for i in 1:12
        station = sorted_initial[i]

        p1 = plot(1:24, num_hourly[:, station], label = "Station $station", xlabel = "Hour", ylabel = "N
```

```
        plot!(1:24, sum(flow[:, station, :], dims = 2), label = "Flow Out", xlabel = "Hour", ylabel = "N
        plot!(1:24, sum(flow[:, :, station], dims = 2), label = "Flow In", xlabel = "Hour", ylabel = "Nu
        push!(plots, p1)
    end

    plot(plots..., layout = (2, 6), size = (1500, 400))
end

num_bikes_trend(value.(num_hourly), value.(flow))
```

In [ ]:
```
# For model 2
sorted_station = sortperm(value.(num_hourly[1, :]), rev = true)
id_station_map = Dict(value => key for (key, value) in station_id_map)
for i in 1:10
    println(id_station_map[sorted_station[i]])
end
```

In [ ]:
```
# For model 2
using DataFramesMeta

start_data = @chain data begin
    @subset(:start_station_id .!= 0)
    groupby([:start_station_id, :hour])
    @combine begin
        :n = sum(:n)
    end
    @orderby(-:n)
end

end_data = @chain data begin
    @subset(:end_station_id .!= 0)
    groupby([:end_station_id, :hour])
    @combine begin
        :n = sum(:n)
    end
    @orderby(-:n)
end

new_data = innerjoin(start_data, end_data, on = [:start_station_id => :end_station_id, :hour], makeuniqu

function origin_data_trends(data)
    station_group = groupby(data, :start_station_id)

    sorted_initial = @chain station_group begin
        @combine(:n_sum = sum(:n), :n_1_sum = sum(:n_1))
        @transform(:n_mean = :n_sum / sum(:n_sum), :n_1_mean = :n_1_sum / sum(:n_1_sum))
        @orderby(-(:n_mean + :n_1_mean))
        @select(:start_station_id)
    end

    # plot the trend of number of bikes in the station with the most bikes
    plots = []


    for i in 1:10
        station = station_group[(start_station_id = sorted_initial[i, :].start_station_id,)]

        sort!(station, :hour)

        p1 = plot(station.hour, station.n, label = "Flow Out (Station $(sorted_initial[i, :].start_stati
        plot!(station.hour, station.n_1, label = "Flow In", xlabel = "Hour", ylabel = "Number of bikes")
        push!(plots, p1)
    end

    plot(plots..., layout = (2, 5), size = (1200,500))
end

origin_data_trends(new_data)
```

In [ ]:
```
# For Model 3
plot(price_choice, revenues, ylabel = "Revenue", xlabel="Price", label = "Revenue")
```

---

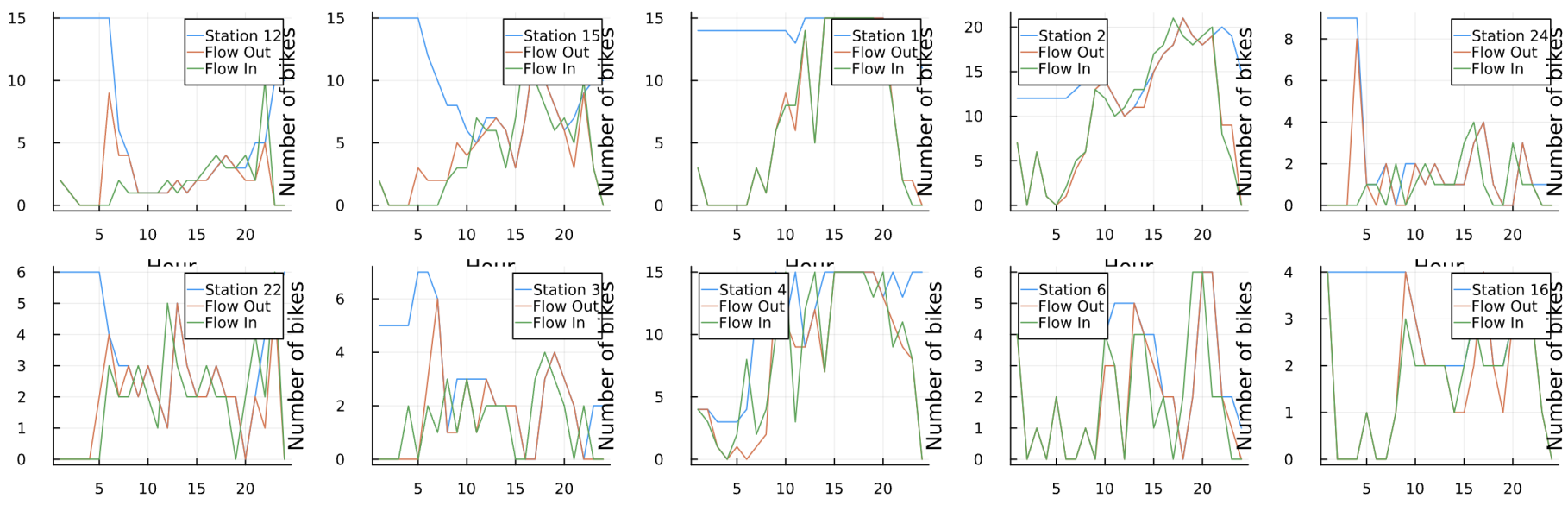
4. Results and discussion

Model 1: Bike Supply Model (IP)

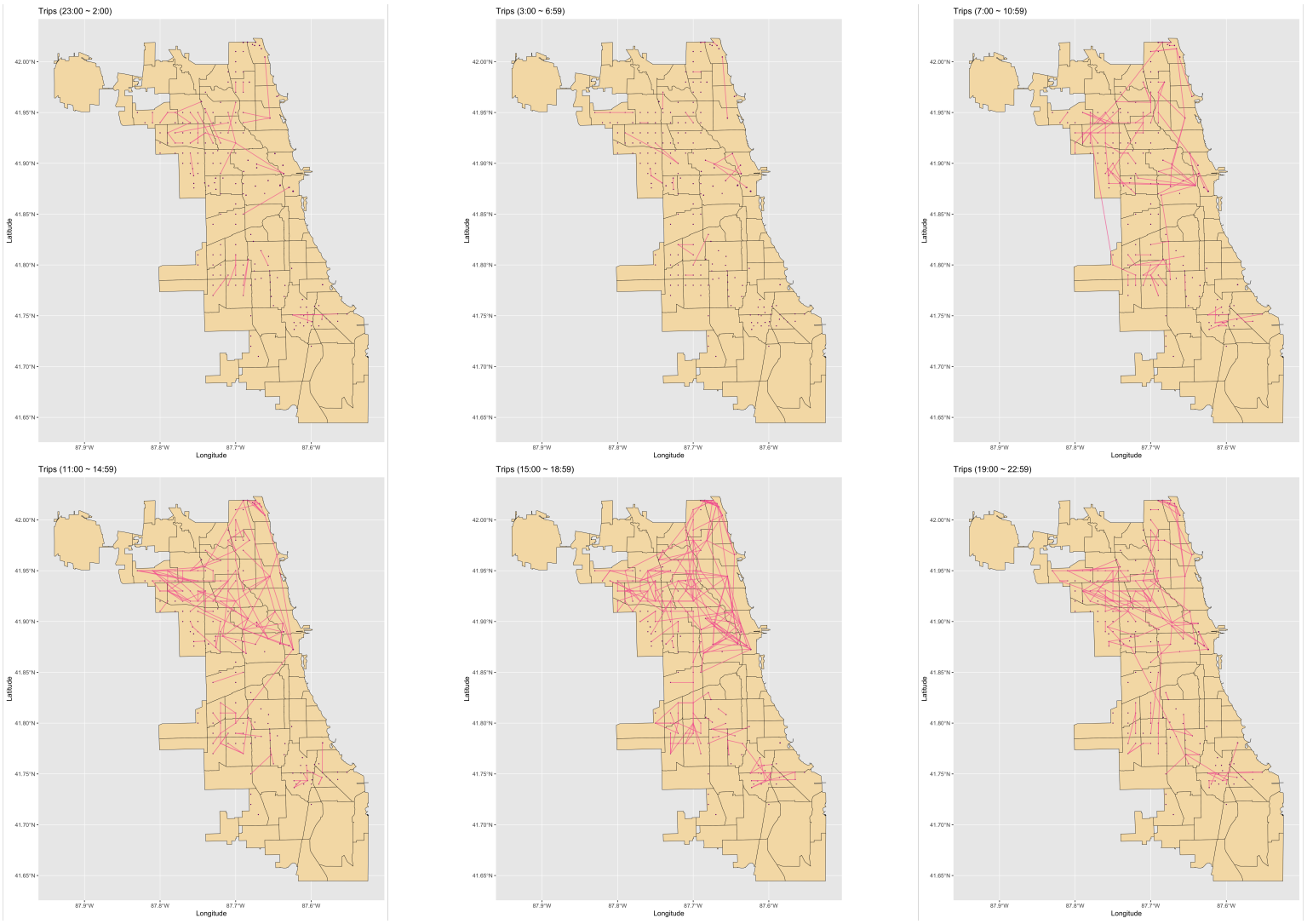
- From 244 possible stations, the model choose to give 66 stations a non-zero endowment of bikes.
- The total supply of bikes is 174.
- The total profit for current network is $9059.92 per month.
- The table below demonstrate the stations with endowment > 2.

Station ID	Bike Endowment	Station Name
12	15.0	Halsted St & Clybourn Ave
15	15.0	University Library (NU)
1	14.0	Jonathan Y Scammon Public School
2	12.0	Lakefront Trail & Wilson Ave
24	9.0	Greenview Ave & Jarvis Ave
53	9.0	Cornell Dr & Hayes Dr
22	6.0	Hoyne Ave & Balmoral Ave
3	5.0	Michigan Ave & 8th St
4	4.0	Clinton St & Jackson Blvd
6	4.0	Racine Ave & Washington Blvd
16	4.0	Eastlake Ter & Rogers Ave
49	4.0	Keystone Ave & North Ave
25	3.0	Ridge Blvd & Howard St
58	3.0	Central Ave & Roscoe St
66	3.0	Hamlin Ave & 62nd Pl

In addition, we would like to visualize the optimal flow generated by the model in the figure below (blue line indicates the remaining bike in the station in a given hour, orange line indicates out flow in a given hour, and green line indicates in flow in a given hour).



We also visualize the distribution of Divvy Bike Stations and Divvy Trips between these stations in May 2022 by Hour in the image below. This suggests the optimal flow generated by the model matched the real data since we assumed that the flow capacity from station i to station j will not exceed the flow from i to station j in May 2022.



Summary

To optimize profit for Divvy Bike in May, the company should give 66 stations a non-zero endowment of bikes. Assuming each bike can only be used once per hour and there are no additional maintenance fees, a new bike can be used for 24 months, each bike station should be endowed with the number of bikes listed in the table above at 0:00 each day. The total supply of bike would be 174. Ultimately, this network of bike stations would generate a total profit of $9059.92 per month.

Model 2: Bike Supply & Station Management Model (IP)

General Information

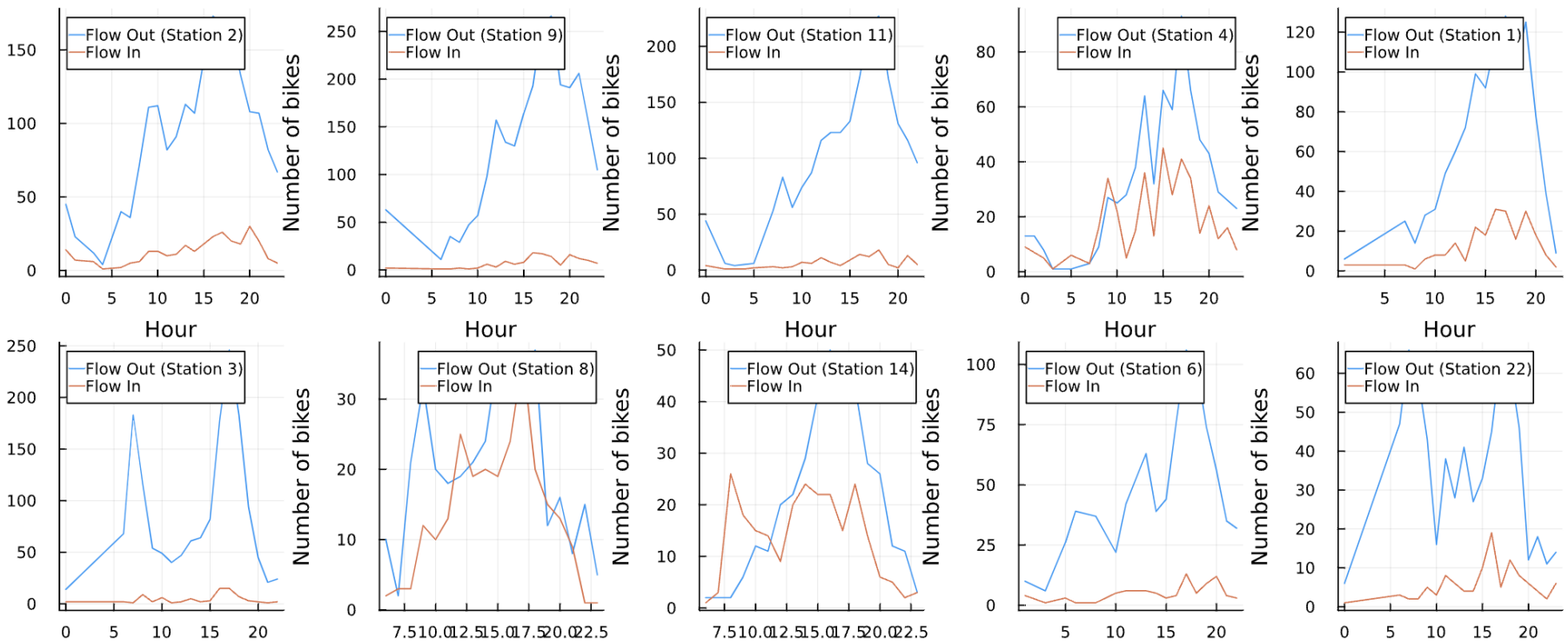
- From 244 possible stations, the model choose to build 10 stations and supply a total of 85 bikes.
- The total supply of bike is 85.
- The total profit for current network is $3957.72 per month.
- The model does not have a dual solution since for this IP problem.

Station ID	Bike Endowment	Station Name
1	15.0	Jonathan Y Scammon Public School
12	15.0	Halsted St & Clybourn Ave
15	15.0	University Library (NU)
2	14.0	Lakefront Trail & Wilson Ave
8	8.0	Wood St & Chicago Ave
4	5.0	Clinton St & Jackson Blvd
53	5.0	Cornell Dr & Hayes Dr
6	4.0	Racine Ave & Washington Blvd
23	2.0	Wood St & Augusta Blvd
159	2.0	Greenwood Ave & 79th St

Why these stations?

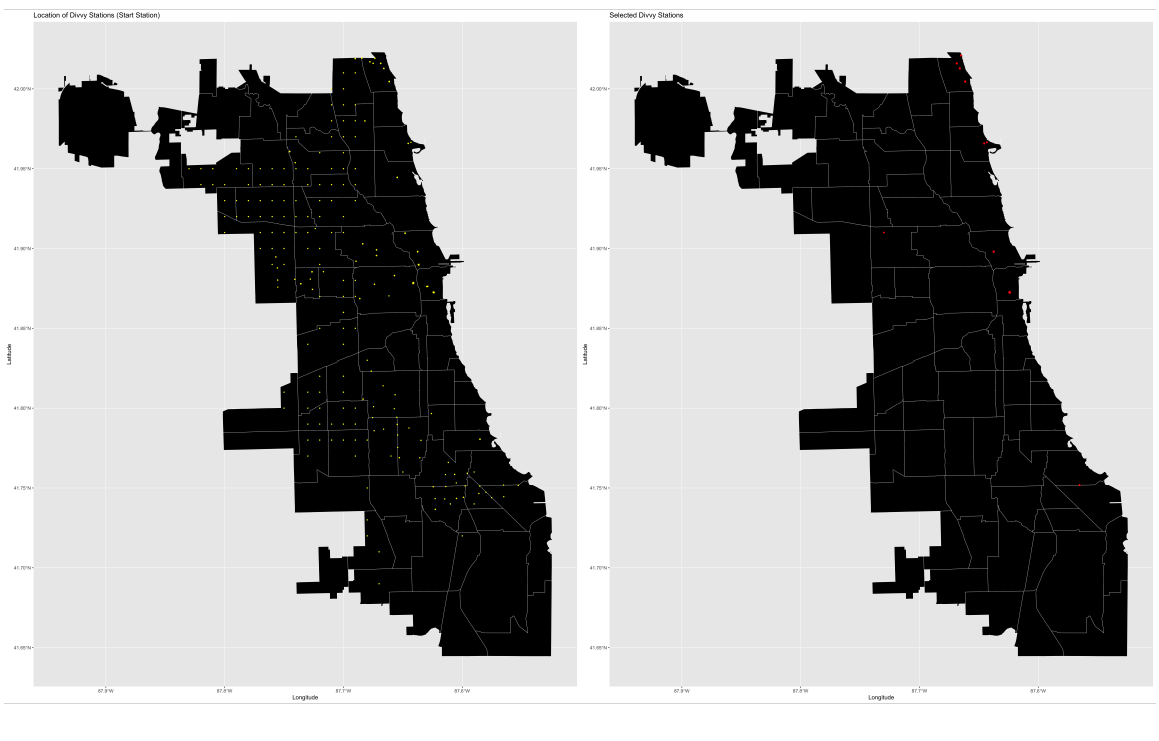
We would like to further explore why model would choose 10 stations in the table above by answering the question that whether the selected stations have any common harateristics. We would first visualize the **Orginal Station Flow Data** (not the models'). Here, we plot 10 stations that have the highest average throughput

$$\text{Average Throughput} = \frac{\text{In Flow} + \text{Out Flow}}{\text{Total Number of Flows for all stations in May}}.$$



Based on the plots presented above, we can infer that the stations selected by the model, including Station 1, Station 2, Station 4, Station 6, and Station 8, also have the highest average total flow. This suggests that stations with a "high average throughput" are likely to be selected by the model. However, it remains unclear why Station 12, Station 15, Station 53, Station 23, and Station 159 were also chosen. One possible explanation is that if a station is not selected for construction, all trips associated with that station would be invalid. Therefore, after selecting stations with high average throughput, the model may have selected additional stations to ensure that there are enough valid trips to generate optimal revenue.

The graph below demonstrates the locations of selected stations (*left*: all stations, *right*: selected (built) station). Comparing the graph below with "distribution of Divvy Bike Stations and Divvy Trips between these stations in May 2022" in the introduction part could partially support the claim above.



Comparing with Model 1

When comparing with Model 1, we observe that the total number of bikes and profit are significantly reduced. This outcome is not unexpected, given that each new station incurs a cost of $10,000, which represents a substantial reduction in the overall revenue available to purchase additional bikes. Consequently, the limited budget results in a reduced number of bikes and subsequently, a lower overall profit.

Summary

To optimize profit for Divvy Bike in May, the company should construct 10 new bike stations according to the above table. Assuming each bike can only be used once per hour, there are no additional maintenance fees, and a new bike can be used for 24 months while a new station can be used for 60 months, each bike station should be endowed with the number of bikes listed in the table below at 0:00 each day. The total supply of bikes would be 85. Ultimately, this network of bike stations would generate a total profit of $3,957.72 per month.

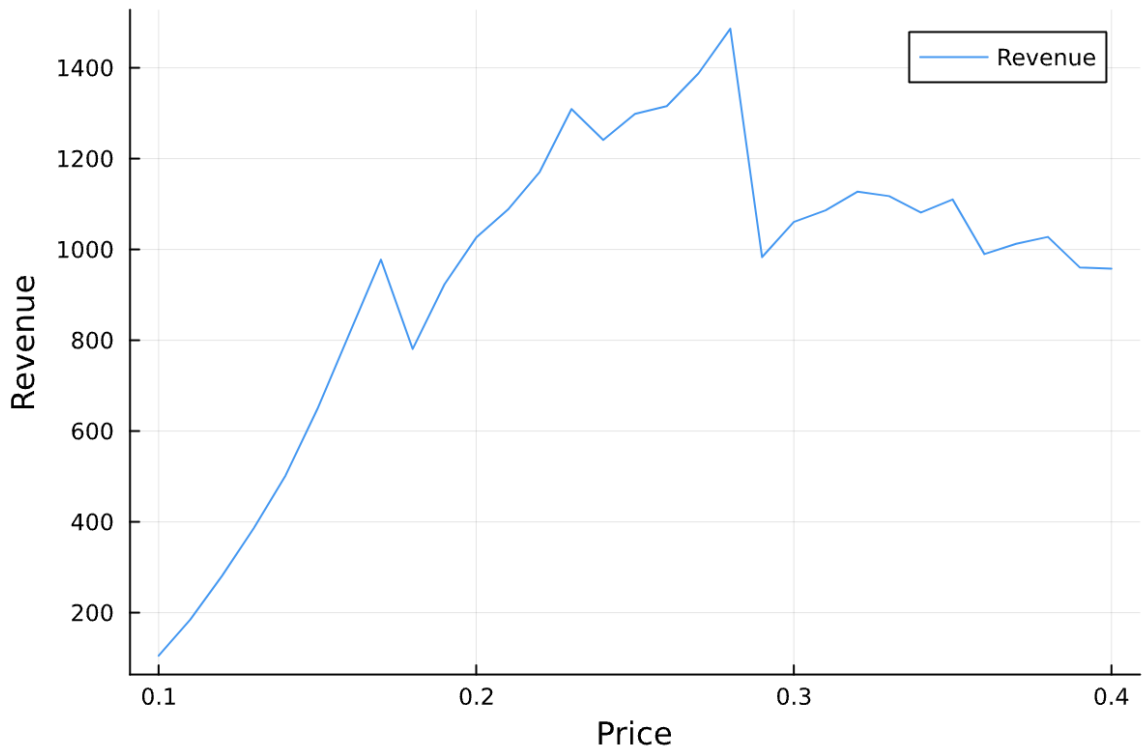
Additionally, the optimization model favors a combination of stations that can generate "valuable" trips (high trip counts & long duration). Typically, this involves selecting stations with high throughput as well as other relevant stations that have a large number of trips with the high-throughput stations.

Model 3: Bike Supply, Pricing & Station Management Model (MIP)*

The non-convexity of Model 3 makes it difficult to find a global or local maximum using optimization techniques in a limited amount of time and computation resources. However, instead of using price as a variable, we can create a sequence of prices ranging from 0.1 to 0.40 with an interval of 0.01 and calculate the optimal profit for each price.

The graph below shows that the total profit is not a monotonically increasing function of price due to the trade-off between price and flow size. As mentioned earlier, the proportion of total demand that Divvy Bike can capture can be expressed as a function of its price relative to its competitors (proportion of total demand that *Bivvy Bike* shared could be formulated as $D_{Bivvy} = \frac{1}{1+\exp(-1+\frac{P_{Bivvy}}{P_{Divvy}})}$), and a lower price can lead to greater flow size but each trip would bring in less revenue, and vice versa.

From the plot below, we also find that at price of `$0.28/min`, the firm would obtain the profit of `$1486.14`. This is reasonable, since at p = 0.28, Bivvy Bike would only have 34.37% of total demands in bike share markest (assume there are only two bike share firms named Divvy and Bivyy in the bike share market).



---

5. Conclusion

In conclusion, **Model 1** suggests that Divvy Bike should endow 66 bike stations with bikes to optimize profit, resulting in a total supply of 174 bikes and a monthly profit of $9059.92. **Model 2** recommends constructing 10 new bike stations with an endowment of 85 bikes to generate a monthly profit of $3,957.72. Valuable trips are favored by the optimization model, which involves selecting stations with high throughput and large numbers of trips with high-throughput stations. **Model 3** is non-convex, making optimization difficult, but a sequence of prices can be used to calculate the optimal profit for each price. The total profit is not a monotonically increasing function of price due to the trade-off between price and flow (trips) size.

The primary limitation of this model is that it does not account for the imbalancing problem between stations that may arise over time. While the model assumes that each station will have the designated number of bikes at the start of every day and that there are no transportation costs associated with redistributing bikes between stations, this is not the case in reality. In a Bike-share system, the imbalancing problem is a critical issue that requires attention, and future models should factor in the costs associated with transporting bikes between stations. Another limitation is that the model's time scope is limited to a single day, whereas in reality, the imbalancing problem may manifest over several days or even weeks. To address this limitation, future models should consider a longer time horizon to handle the imbalancing problem between stations more effectively. Furthermore, the current model uses May 2022 data to optimize strategies for May 2023, but it would be more effective to use a negative binomial or time series model to forecast the flow capacity between stations accurately. By incorporating forecasting models, future models can better anticipate changes in demand and make more accurate predictions to optimize the allocation of resources between stations.

Reference

Brewster, C. (2023, May 2). *Chicago's divvy to become biggest bikeshare in North America by service area*. WGN. Retrieved May 2, 2023, from https://wgntv.com/news/chicago-news/chicago-expands-divvy-bikeshare-system/

Divvy. (2022, September 10). *Divvy bicycle stations: City of Chicago: Data Portal*. Chicago Data Portal. Retrieved May 2, 2023, from https://data.cityofchicago.org/Transportation/Divvy-Bicycle-Stations/bbyy-e7gq

Divvy Data. Home. (n.d.). Retrieved May 2, 2023, from https://divvybikes.com/system-data

Pal, A., & Zhang, Y. (2017). Free-floating bike sharing: Solving real-life large-scale static rebalancing problems. *Transportation Research Part C: Emerging Technologies*, 80, 92-116.

Qian, X., Jaller, M., & Circella, G. (2022). Equitable distribution of bikeshare stations: An optimization approach. *Journal of transport geography*, 98, 103174.

Wang, Y., & Szeto, W. Y. (2018). Static green repositioning in bike sharing systems with broken bikes. *Transportation Research Part D: Transport and Environment*, 65, 438-457

Yan, S., Lin, J. R., Chen, Y. C., & Xie, F. R. (2017). Rental bike location and allocation under stochastic demands. *Computers & Industrial Engineering*, 107, 1-11.