The Transportation Problem

Problem Statement

In the dynamic environment of fashion retail, optimizing transportation logistics emerges as a strategy for enhancing efficiency and reducing costs.
The transportation problem focuses on minimizing costs and itinerary time for two distinct groups involved in showroom appointments: buyers and salespeople.

Buyers from prominent retail companies navigate through the city to attend 6-8 meetings daily across various locations during the fashion week season. Their primary objectives are to:

1. streamline their routes effectively;
2. adjust in real-time to traffic;
3. strategically cluster meetings within proximate areas to maximize productivity.

Salespeople representing elite brands must:

1. efficiently calculate travel times
2. adapt to unforeseen schedule changes due to external events e.g. cancellations
3. organize their meetings to optimize their daily schedule, which translates into reducing operational costs.

By addressing these tailored needs through refined transportation strategies, companies can significantly bolster profitability and operational efficacy.

Algorithm Rationale

The rationale behind the development of the algorithm begins with the reducibility theorem and the foundational concept of the Traveling Salesman Problem (TSP), where the objective is to find the shortest possible route that visits a set of locations exactly once and returns to the origin point. Extending this problem to accommodate scenarios where there are multiple buyers, each responsible for covering distinct sets of showrooms without overlap with their colleagues, the problem evolves into the Multiple Traveling Salesmen Problem (mTSP). This adaptation distributes the workload among several agents, thereby complicating the route optimization process due to the introduction of parallel operational constraints.

Further complexity is introduced with the integration of specific time windows for each showroom, turning the problem into the Multiple Traveling Salesmen Problem with Time Windows (mTSPTW). Each showroom, treated as a node in the problem, has an associated single time window characterized by a predefined opening and closing time. The algorithm must, therefore, not only determine the optimal route for each buyer to minimize travel time and distance but also synchronize these routes with the operational schedules of the showrooms. This synchronization ensures that each buyer arrives at each showroom within the permissible time window, making the problem both a routing and a scheduling challenge. The inclusion of time windows adds a layer of temporal constraints that requires the algorithm to

balance between spatial efficiency and time-specific feasibility, thus significantly enhancing the complexity and practical relevance of the solution.

Construction Hypothesis

The introduction of the following set of hypotheses is strategically aligned with the goal of optimizing the transportation logistics in the context of fashion retail showroom visits, where efficiency and cost reduction are paramount.

1. A list of brands' showroom addresses is provided, from which the travel costs i.e. average travel time can be retrieved
2. Being employees paid by hour, the unit cost of travelling i.e. distance would coincide with the unit cost of waiting, hence this reduces to the Makespan Problem with Time Window. Therefore, the ultimate objective would be just to minimize the tour duration.
3. Buyers depart from the same starting point as their colleagues', which is the retail company they work for i.e. depot.
4. The TSPTW at hand is stated to be symmetric, setting $t_{ij}=t_{ji}$. Hence, arcs in the complete graph [Subsection *Index Sets* of *Notations*] will be undirected.
5. Brands' showrooms are considered to be non-stop i.e. an only time window will be associated to each node.
6. Time to visit a brand showroom is measured based a worst-case scenario fashion, in order to get covered as many scenarios as possible.

Notations

1. Index Sets

Let m be the number of buyers within the same retail company, n the number of brands' showrooms and $G=(V,A)$ a complete graph, such that:

- $V=(0,1,..,n)$ is the set of the nodes i.e. brands' showrooms;
- Node 0 is the Depot;
- $A=((i,j)\vee i\neq j, j\in V)$ is the set of undirected arcs [4^{th} hypothesys in Section *'Construction Hypothesis'*].

2. Parameters

- t_{ij} is the travel time from the i^{th} showroom to the j^{th} [4^{th} hypothesys in Section *'Construction Hypothesis'*];
- d_i is the meeting duration at the i^{th} showroom [6^{th} hypothesys in Section *'Construction Hypothesis'*];
- a_i is the earliest possible visit to the i^{th} showroom i.e. opening hour during its working dates;
- b_i is the latest possible visit to the i^{th} showroom i.e. closing hour subtracted by its meetiong duration, during working dates.

Therefore, $[a_i, b_i)$ consists in the time window related to node i [5^{th} hypothesys in Section *Construction Hypothesis*].

3. Decision Variables

- t_{ik} is the meeting time the k^{th} salesperson shows up at the i^{th} showroom;
- $x_{ijk} \in \{0, 1\}$ is the auxiliarity binary value. It is equals to 1 if and only if salesman k travels from showroom of brand i to j's.

Formulation - Integer Linear Programming [ILP]

[Objective] Find a set of (n) routes that services all at minimum times i.e. time to get from the i^{th} showroom to the j^{th} plus the meeting time at the j^{th}.

N.B. This minimization by time will contribute to maximize the number of meetings scheduled per day.

$$min\left(\sum_{k=1}^{m}\sum_{i=0}^{n}\sum_{j=0,\,j\neq i}^{n} x_{ijk}\cdot t_{ij} + \sum_{k=1}^{m}\sum_{i=1}^{n} x_{i0k}\cdot d_i\right)$$

[Constraint 1] Time window limitations

1. Triangle inequality depot w.r.t. depot

$$\begin{Bmatrix} t_{ik} - t_0 \geq t_{0i} \\ t_{(n+1)k} - t_{ik} \geq t_{i0} \end{Bmatrix}$$

2. (Lineralized) Triangle inequality between two consequently placed showrooms (within same salesperson's route)

   Where $M > 0$ and a sufficiently large number and $y_{ij}s$ are binary variables for all arcs of A

$$\begin{Bmatrix} t_{ik} - t_{jk} \geq t_{ij} - M\,y_{ij} \\ t_{jk} - t_{ik} \geq t_{ij} - M\,(1 - y_{ij}) \end{Bmatrix}$$

3. Meeting time compatibility w.r.t. opening hour

$$\begin{Bmatrix} t_{ik} \geq a_i \\ t_{ik} + d_i \leq b_i \\ t_{ik} \geq 0 \end{Bmatrix}$$

[Constraint 2 - BONUS not covered in the task] All buyers' routes start and end at their retail company

$$\left(\begin{array}{ll} \sum_{j \in V : (1,j) \in A} x_{1j} = m & depot_depart \\ \sum_{j \in V : (j,1) \in A} x_{j1} = m & depot_return \end{array} \right)$$

[Constraint 3] Each brand showroom must be visited exactly once by only one buyer from each retail company

¿

N.B. Last equation in the system checks whether the arriving buyier is consistent i.e. equals to the leaving one. It assures that the showroom has been visited by only one buyer from the retail company.

[Constraint 4] Subtour elimination

Being $\left | S \right | - 1 $ the number of hacks:

$$\sum_{ijk} x_{ijk} \leq |S) - 1 \quad \forall S \subseteq P(N), \quad depot \notin S$$

N.B. In ILP the number of subtour elimination constraints grows exponentially.

[Constraint 5]

$$x_{ijk} \in (0,1)$$

Live Traffic and Cancellations Handling

1. Adopted Method - Google Maps Cloud API

   In addressing the dynamic challenges of the Multiple Travelling Salesmen Problem with Time Windows (MTSPTW), this work incorporates of Google Maps Cloud API. This API provides real-time traffic data and information on route cancellations, which significantly enhances the resolution of the live traffic issues faced in mTSPTW. By dynamically updating the travel times between nodes based on current traffic conditions and immediately adjusting the routes in response to cancellations, this solution remains adaptive and efficient in real-time operational contexts.

2. Reinforcement Learning

   In this approach, the environment is defined by the cities, roads, traffic states and time windows, with salesmen serving as agents. Each agent's actions (choosing routes between cities) result in rewards (minimized travel time and penalties for time window breaches). A model-based RL can be employed where a simulation of the environment incorporates live data updates from the Google Maps API, allowing the RL agent to learn and refine policies that **anticipate** and **react to changes** in traffic patterns and route availability. This method leverages techniques like Q-

learning or Policy Gradient to iteratively improve the decision-making process, thus enhancing route optimization continuously.

3. Genetic Algorithms

   Genetic Algorithms (GAs), on the other hand, provide a **heuristic method** to explore a vast solution space through processes mirroring natural selection, including crossover, mutation and selection. Each potential solution, or individual in the population, represents a set of routes for all salesmen. Fitness functions are defined to evaluate solutions based on the total travel time, adherence to time windows and response to real-time traffic and route cancellations. GAs iteratively evolve the population towards better solutions by selecting high-fitness individuals to reproduce, crossing over their routes to generate new offspring, and introducing mutations to maintain genetic diversity. This method is particularly effective for finding near-optimal solutions quickly in complex, dynamic scenarios such as the mTSPTW, where traditional methods might falter due to computational infeasibility.

Reading Input Data

The brands with relative address, time window constraints and meeting duration are retrieved from a xlsx file.

```
import pandas as pd

df = pd.read_excel('Modacruising.xlsx', engine='openpyxl')
print(df.head())
```

```
             Marque                Adresse Date ouverture (MM/DD/AAAA)
\
0         schouler   15 Avenue Hoche,75008                   26/01/2024

1           Proenza  15 Avenue Hoche,75008                   26/01/2024

2         retrofete  15 Avenue Hoche,75008                   26/01/2024

3  Hemant & Nandita  15 Avenue Hoche,75008                   26/01/2024

4         Ser.Ho.Ya  15 Avenue Hoche,75008                   26/01/2024


  Date fermeture Opening hour Closing hour  Length
0     31/01/2024     09:00:00     19:00:00      30
1     31/01/2024     09:00:00     19:00:00      30
2     31/01/2024     09:00:00     19:00:00      30
3     31/01/2024     09:00:00     19:00:00      30
4     31/01/2024     09:00:00     16:00:00      30
```

```
n = len(df)    # Number of brands i.e. showrooms to be visited
m = 2          # Number of retail company buyers, it can be adapted
based on the retail company buyers' availability
```

Travel Costs

Being the task at hand a problem of itinerary optimization aimed at:

- Maximizing the number of buyer's meetings per day
- Minimizing the costs i.e. saving money, since employees are paied based by hours

The goal reduces to finding the minimum tour duration, added up with meetings duration, since the unit cost of travelling and unit cost of waiting result to be interchangeable.
Therefore, showrooms' locations are exploited for the retrieval of travel times (in minutes) that elapse in between two different showrooms.

```
"""
To use this script, you need a Google Maps API key. Follow these steps
to obtain one:
1. Go to the Google Cloud Console (https://console.cloud.google.com/).
2. Create a new project or select an existing one.
3. Navigate to 'APIs & Services' > 'Library', search for 'Google Maps
Directions API', and enable it for your project.
4. Go to 'Credentials', click on 'Create Credentials', and choose 'API
key'. Your new API key will be displayed.
5. Copy the API key and replace 'YOUR_API_KEY_HERE' with your actual
API key in the script.
"""

import googlemaps
from datetime import datetime

def get_travel_time(api_key, start_address, end_address,
mode='driving'):
    """
    Fetches the travel time in minutes between two addresses using the
Google Maps Directions API.

    Args:
    api_key (str): Google Maps API key
    start_address (str): The travel starting address
    end_address (str): The travel address of the journey
    mode (str): Transportation modality, it defaults to 'driving' -
other options include 'walking', 'bicycling' and 'transit'

    Returns:
    int: Travel time in minutes
    """
    gmaps = googlemaps.Client(key=api_key)

    # Request directions between tshowrooms' addresses
    now = datetime.now()
    directions_result = gmaps.directions(start_address,
                                         end_address,
```

```
                                            mode=mode,
                                            departure_time=now)

    # Extract the duration from the first route found and convert it
to minutes
    duration = directions_result[0]['legs'][0]['duration']['value']
# Travel time in seconds
    travel_time_minutes = duration // 60

    return travel_time_minutes

def calculate_brand_travel_time_matrix(data, api_key):
    """
    Calculates the travel time matrix for a list of brands and their
addresses using the Google Maps Directions API.

    Args:
    data (DataFrame): A DataFrame containing at least two columns
'Marque' for brand names and 'Adresse' for addresses
    api_key (str): Google Maps API key to authenticate the requests

    Returns:
    DataFrame: A square DataFrame where indices and columns are brand
names, and values are travel times in minutes
    """

    gmaps = googlemaps.Client(key=api_key)

    # Create mappings from brands to addresses
    brand_to_address = data.set_index('Marque')['Adresse'].to_dict()
    # Get unique brands
    unique_brands = data['Marque'].unique()
    travel_times = pd.DataFrame(index=unique_brands,
columns=unique_brands, dtype=int)
    cache = {}    # Store previously computed travel times

    for origin_brand in unique_brands:
        for destination_brand in unique_brands:
            # Avoid computing travel time from a brand to itself
            if origin_brand != destination_brand:
                origin_address = brand_to_address[origin_brand]
                destination_address =
brand_to_address[destination_brand]

                # In the Modacruising.xlsx dataset it was observed
that several showrooms shared the same address, leading to repeated
API calls
                # To enhance efficiency, the following optimizations
were applied:
```

```
                # 1) Travel times were calculated once for each unique address pair and reused for all brands at those addresses
                # 2) A caching mechanism was introduced to minimize redundant requests
                address_pair = tuple(sorted([origin_address, destination_address]))
                if address_pair not in cache:
                    travel_time = get_travel_time(api_key, origin_address, destination_address)
                    cache[address_pair] = travel_time
                else:
                    travel_time = cache[address_pair]

                travel_times.loc[origin_brand, destination_brand] = travel_time

    return travel_times

# Avoiding to expose the API key publicly to not incurr unauthorized use
api_key = "Please insert yours here"
travel_time_matrix = calculate_brand_travel_time_matrix(df, api_key)
print(travel_time_matrix)
```

	schouler	Proenza	retrofete	Hemant & Nandita
schouler	NaN	0.0	0.0	0.0
Proenza	0.0	NaN	0.0	0.0
retrofete	0.0	0.0	NaN	0.0
Hemant & Nandita	0.0	0.0	0.0	NaN
Ser.Ho.Ya	0.0	0.0	0.0	0.0
247 Showroom	0.0	0.0	0.0	0.0
Mint	0.0	0.0	0.0	0.0
The Known Agency	0.0	0.0	0.0	0.0
Nahmias	0.0	0.0	0.0	0.0
Willy Chavarria	0.0	0.0	0.0	0.0
Veronica Beard	0.0	0.0	0.0	0.0
Atelier	0.0	0.0	0.0	0.0

Cinq à Sept	0.0	0.0	0.0	0.0
Alexandre Vauthier	9.0	9.0	9.0	9.0
Vanessa Bruno	24.0	24.0	24.0	24.0
Yves Salomon	12.0	12.0	12.0	12.0
Fursac	14.0	14.0	14.0	14.0
Harris Wharf London	25.0	25.0	25.0	25.0
Labo.Art	24.0	24.0	24.0	24.0
Le Kasha	18.0	18.0	18.0	18.0
Soeur	16.0	16.0	16.0	16.0
Del Rainbow	5.0	5.0	5.0	5.0
Showroom Marcona3	24.0	24.0	24.0	24.0
ShowroomThomasDufour	22.0	22.0	22.0	22.0
Six London	23.0	23.0	23.0	23.0
Sphere Paris	8.0	8.0	8.0	8.0
Talk Studio	22.0	22.0	22.0	22.0
Vald Agency	23.0	23.0	23.0	23.0
032C	19.0	19.0	19.0	19.0
1972 Desa	8.0	8.0	8.0	8.0
44 Label Group	25.0	25.0	25.0	25.0
A.A Spectrum	24.0	24.0	24.0	24.0
Adidem Asterisks	22.0	22.0	22.0	22.0
Amish Supplies	22.0	22.0	22.0	22.0
Antik Batik	25.0	25.0	25.0	25.0
Ag-ency Showroom	24.0	24.0	24.0	24.0
Chevignon	22.0	22.0	22.0	22.0
Inoui Editions	16.0	16.0	16.0	16.0

	Ser.Ho.Ya	247 Showroom	Mint	The Known Agency
\				
schouler	0.0	0.0	0.0	0.0
Proenza	0.0	0.0	0.0	0.0
retrofete	0.0	0.0	0.0	0.0
Hemant & Nandita	0.0	0.0	0.0	0.0
Ser.Ho.Ya	NaN	0.0	0.0	0.0
247 Showroom	0.0	NaN	0.0	0.0
Mint	0.0	0.0	NaN	0.0
The Known Agency	0.0	0.0	0.0	NaN
Nahmias	0.0	0.0	0.0	0.0
Willy Chavarria	0.0	0.0	0.0	0.0
Veronica Beard	0.0	0.0	0.0	0.0
Atelier	0.0	0.0	0.0	0.0
Cinq à Sept	0.0	0.0	0.0	0.0
Alexandre Vauthier	9.0	9.0	9.0	9.0
Vanessa Bruno	24.0	24.0	24.0	24.0
Yves Salomon	12.0	12.0	12.0	12.0
Fursac	14.0	14.0	14.0	14.0
Harris Wharf London	25.0	25.0	25.0	25.0
Labo.Art	24.0	24.0	24.0	24.0
Le Kasha	18.0	18.0	18.0	18.0
Soeur	16.0	16.0	16.0	16.0
Del Rainbow	5.0	5.0	5.0	5.0
Showroom Marcona3	24.0	24.0	24.0	24.0
ShowroomThomasDufour	22.0	22.0	22.0	22.0
Six London	23.0	23.0	23.0	23.0

Sphere Paris	8.0	8.0	8.0	8.0
Talk Studio	22.0	22.0	22.0	22.0
Vald Agency	23.0	23.0	23.0	23.0
032C	19.0	19.0	19.0	19.0
1972 Desa	8.0	8.0	8.0	8.0
44 Label Group	25.0	25.0	25.0	25.0
A.A Spectrum	24.0	24.0	24.0	24.0
Adidem Asterisks	22.0	22.0	22.0	22.0
Amish Supplies	22.0	22.0	22.0	22.0
Antik Batik	25.0	25.0	25.0	25.0
Ag-ency Showroom	24.0	24.0	24.0	24.0
Chevignon	22.0	22.0	22.0	22.0
Inoui Editions	16.0	16.0	16.0	16.0

	Nahmias	Willy Chavarria	...	032C	1972 Desa \
schouler	0.0	0.0	...	19.0	8.0
Proenza	0.0	0.0	...	19.0	8.0
retrofete	0.0	0.0	...	19.0	8.0
Hemant & Nandita	0.0	0.0	...	19.0	8.0
Ser.Ho.Ya	0.0	0.0	...	19.0	8.0
247 Showroom	0.0	0.0	...	19.0	8.0
Mint	0.0	0.0	...	19.0	8.0
The Known Agency	0.0	0.0	...	19.0	8.0
Nahmias	NaN	0.0	...	19.0	8.0
Willy Chavarria	0.0	NaN	...	19.0	8.0
Veronica Beard	0.0	0.0	...	19.0	8.0

Atelier	0.0	0.0	...	19.0	8.0
Cinq à Sept	0.0	0.0	...	19.0	8.0
Alexandre Vauthier	9.0	9.0	...	17.0	6.0
Vanessa Bruno	24.0	24.0	...	8.0	31.0
Yves Salomon	12.0	12.0	...	10.0	19.0
Fursac	14.0	14.0	...	6.0	20.0
Harris Wharf London	25.0	25.0	...	9.0	31.0
Labo.Art	24.0	24.0	...	8.0	30.0
Le Kasha	18.0	18.0	...	1.0	26.0
Soeur	16.0	16.0	...	8.0	24.0
Del Rainbow	5.0	5.0	...	24.0	5.0
Showroom Marcona3	24.0	24.0	...	12.0	21.0
ShowroomThomasDufour	22.0	22.0	...	8.0	30.0
Six London	23.0	23.0	...	5.0	29.0
Sphere Paris	8.0	8.0	...	17.0	7.0
Talk Studio	22.0	22.0	...	6.0	28.0
Vald Agency	23.0	23.0	...	6.0	27.0
032C	19.0	19.0	...	NaN	26.0
1972 Desa	8.0	8.0	...	26.0	NaN
44 Label Group	25.0	25.0	...	9.0	26.0
A.A Spectrum	24.0	24.0	...	9.0	29.0
Adidem Asterisks	22.0	22.0	...	7.0	27.0
Amish Supplies	22.0	22.0	...	7.0	27.0
Antik Batik	25.0	25.0	...	10.0	30.0
Ag-ency Showroom	24.0	24.0	...	9.0	24.0
Chevignon	22.0	22.0	...	7.0	27.0

Inoui Editions	16.0	16.0	...	9.0	18.0

	44 Label Group	A.A Spectrum	Adidem Asterisks \
schouler	25.0	24.0	22.0
Proenza	25.0	24.0	22.0
retrofete	25.0	24.0	22.0
Hemant & Nandita	25.0	24.0	22.0
Ser.Ho.Ya	25.0	24.0	22.0
247 Showroom	25.0	24.0	22.0
Mint	25.0	24.0	22.0
The Known Agency	25.0	24.0	22.0
Nahmias	25.0	24.0	22.0
Willy Chavarria	25.0	24.0	22.0
Veronica Beard	25.0	24.0	22.0
Atelier	25.0	24.0	22.0
Cinq à Sept	25.0	24.0	22.0
Alexandre Vauthier	21.0	21.0	20.0
Vanessa Bruno	1.0	3.0	2.0
Yves Salomon	16.0	15.0	13.0
Fursac	12.0	12.0	10.0
Harris Wharf London	4.0	1.0	2.0
Labo.Art	0.0	3.0	2.0
Le Kasha	8.0	7.0	5.0
Soeur	12.0	15.0	11.0
Del Rainbow	30.0	30.0	27.0
Showroom Marcona3	19.0	16.0	18.0
ShowroomThomasDufour	4.0	3.0	2.0

Six London	4.0	3.0	1.0
Sphere Paris	20.0	21.0	20.0
Talk Studio	2.0	2.0	0.0
Vald Agency	7.0	7.0	6.0
032C	9.0	9.0	7.0
1972 Desa	26.0	29.0	27.0
44 Label Group	NaN	3.0	2.0
A.A Spectrum	3.0	NaN	3.0
Adidem Asterisks	2.0	3.0	NaN
Amish Supplies	2.0	3.0	0.0
Antik Batik	4.0	3.0	3.0
Ag-ency Showroom	12.0	14.0	10.0
Chevignon	2.0	3.0	0.0
Inoui Editions	13.0	14.0	11.0

	Amish Supplies	Antik Batik	Ag-ency Showroom \
schouler	22.0	25.0	24.0
Proenza	22.0	25.0	24.0
retrofete	22.0	25.0	24.0
Hemant & Nandita	22.0	25.0	24.0
Ser.Ho.Ya	22.0	25.0	24.0
247 Showroom	22.0	25.0	24.0
Mint	22.0	25.0	24.0
The Known Agency	22.0	25.0	24.0
Nahmias	22.0	25.0	24.0
Willy Chavarria	22.0	25.0	24.0

Veronica Beard	22.0	25.0	24.0
Atelier	22.0	25.0	24.0
Cinq à Sept	22.0	25.0	24.0
Alexandre Vauthier	20.0	21.0	19.0
Vanessa Bruno	2.0	4.0	13.0
Yves Salomon	13.0	16.0	15.0
Fursac	10.0	13.0	11.0
Harris Wharf London	2.0	2.0	13.0
Labo.Art	2.0	4.0	12.0
Le Kasha	5.0	8.0	6.0
Soeur	11.0	13.0	10.0
Del Rainbow	27.0	30.0	28.0
Showroom Marcona3	18.0	16.0	14.0
ShowroomThomasDufour	2.0	4.0	12.0
Six London	1.0	4.0	10.0
Sphere Paris	20.0	23.0	19.0
Talk Studio	0.0	3.0	10.0
Vald Agency	6.0	8.0	10.0
032C	7.0	10.0	9.0
1972 Desa	27.0	30.0	24.0
44 Label Group	2.0	4.0	12.0
A.A Spectrum	3.0	3.0	14.0
Adidem Asterisks	0.0	3.0	10.0
Amish Supplies	NaN	3.0	10.0
Antik Batik	3.0	NaN	15.0
Ag-ency Showroom	10.0	15.0	NaN

Chevignon	0.0	3.0	10.0
Inoui Editions	11.0	16.0	11.0

	Chevignon	Inoui Editions
schouler	22.0	16.0
Proenza	22.0	16.0
retrofete	22.0	16.0
Hemant & Nandita	22.0	16.0
Ser.Ho.Ya	22.0	16.0
247 Showroom	22.0	16.0
Mint	22.0	16.0
The Known Agency	22.0	16.0
Nahmias	22.0	16.0
Willy Chavarria	22.0	16.0
Veronica Beard	22.0	16.0
Atelier	22.0	16.0
Cinq à Sept	22.0	16.0
Alexandre Vauthier	20.0	12.0
Vanessa Bruno	2.0	13.0
Yves Salomon	13.0	8.0
Fursac	10.0	4.0
Harris Wharf London	2.0	14.0
Labo.Art	2.0	13.0
Le Kasha	5.0	6.0
Soeur	11.0	3.0
Del Rainbow	27.0	19.0
Showroom Marcona3	18.0	13.0
ShowroomThomasDufour	2.0	13.0
Six London	1.0	10.0
Sphere Paris	20.0	11.0
Talk Studio	0.0	11.0
Vald Agency	6.0	10.0
032C	7.0	9.0
1972 Desa	27.0	18.0
44 Label Group	2.0	13.0
A.A Spectrum	3.0	14.0
Adidem Asterisks	0.0	11.0
Amish Supplies	0.0	11.0
Antik Batik	3.0	16.0
Ag-ency Showroom	10.0	11.0
Chevignon	NaN	11.0
Inoui Editions	11.0	NaN

[38 rows x 38 columns]

Time Window Data

```
import datetime

# Function to convert time objects to timedelta
def time_to_timedelta(t):
    return pd.Timedelta(hours=t.hour, minutes=t.minute,
seconds=t.second)

df['Opening hour'] =  df['Opening hour'].apply(time_to_timedelta)
df['Closing hour'] = df['Closing hour'].apply(time_to_timedelta)
df['Length'] = pd.to_timedelta(df['Length'], unit='m')
```

Model Implementation and Optimization

```
from gurobipy import Model, GRB, quicksum
from itertools import combinations

# The model reduces to the multiple Travelling Salesmen Problem with
Time Window [mTSPTW]
model = Model("multiple_Travelling_Salesmen_Time_Window")

# DEFINE DECISION VARIABLES
x = model.addVars(n, n, m, vtype=GRB.BINARY, name="x")         #
[Constraint 5]
t_meeting_start = model.addVars(n, m, vtype=GRB.CONTINUOUS, name="t")
y = model.addVars(n, n, m, vtype=GRB.BINARY, name="y")         #
[Constraint 1.1] Binary variables indicating travel from showroom i to
j by buyer k

# DEFINE CONSTRAINTS
# [Constraint 1] Time window constraints
# 1.1 Linearized triangle inequality between two consequent showrooms
M = 1000             # Large constant, to be chosen based on the
context of travel times
for i in range(n):
    for j in range(n):
        if i != j:
            for k in range(m):
                model.addConstr(t_meeting_start[j, k] >=
t_meeting_start[i, k] + travel_time_matrix[i][j] - M * (1 - y[i, j,
k]))

# 1.2 Time compatibility w.r.t. showroom's time window
meeting_durations = []
for j in range(n):
    # Convert timedelta to hours
    a = df.at[j, 'Opening hour'].total_seconds() / 3600
    b = df.at[j, 'Closing hour'].total_seconds() / 3600
    d = df.at[j, 'Length'].total_seconds() / 3600
```

```
    meeting_durations.append(d)         # Will be exploited later in
the objective function

    for k in range(m):
        model.addConstr(t_meeting_start[j, k] >= a)
        model.addConstr(t_meeting_start[j, k] + d <= b)
        model.addConstr(t_meeting_start[j, k] >= 0)



# [Constraint 2] Each buyer must start and end at the depot
#for k in range(m):
#    model.addConstr(quicksum(x[0,j,k] for j in range(1, n+1)) == 1)
#    model.addConstr(quicksum(x[i,0,k] for i in range(1, n+1)) == 1)

# [Constraint 3] Each showroom is visited exactly once by one buyer
# Each buyer visits exactly one showroom and each showroom is visited
by exactly one buyer
for j in range(n):
    sum_entering = quicksum(quicksum(x[i, j, k] for i in range(n)) for
k in range(m))
    sum_exiting = quicksum(quicksum(x[j, i, k] for i in range(n)) for
k in range(m))
    model.addConstr(sum_entering == 1)                 # 3.1 Showroom
i is visited exactly once
    model.addConstr(sum_exiting == 1)                  # 3.2 Showroom
i is exited exactly once
    model.addConstr(sum_entering == sum_exiting)       # 3.3 Ensure
the same buyer who arrives also leaves


# [Constraint 4] Subtour elimination constraint for each buyer of the
retail company
for k in range(m):
    for S in range(2, n):  # Start from 2 to avoid trivial cases
        for subset in combinations(range(1, n+1), S):
            model.addConstr(quicksum(x[i, j, k] for i in subset for j
in subset if i != j) <= len(subset) - 1)

# DEFINE OBJECTIVE FUNCTION
objective = quicksum(quicksum(quicksum(x[i, j, k] *
travel_time_matrix[i][j] for j in range(n) if j != i) for i in
range(n)) for k in range(m))
objective += quicksum(quicksum(x[i, j, k] * meeting_durations[i] for j
in range(n) if j != i) for i in range(n) for k in range(m))

model.setObjective(objective, GRB.MINIMIZE)

# OPTIMIZE
model.optimize()
```