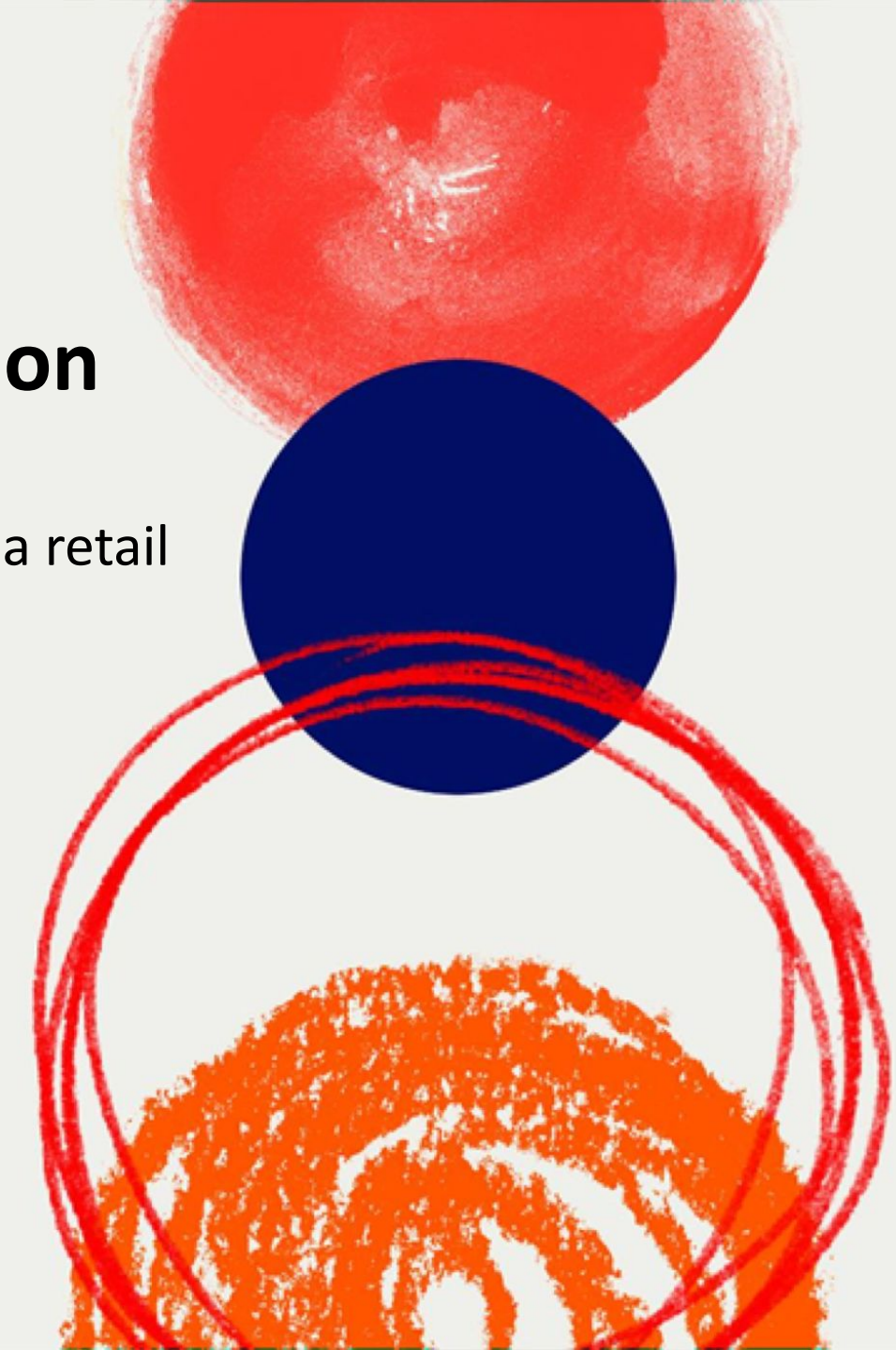
Distribution network Optimization

Definition of strategies to minimize total costs in a retail warehouse and maximize efficiency.

Professors: Pedro Amorim, Sérgio Castro, Daniela Fernandes

George Gittins | Judith Geissler | Konsta Saranpää | Pim Gäfvert

MC Sonae



Contents

- Problem Overview
- Task Breakdown
- Dataset Analysis & Data Preparation
- Solution Description
- Solution Results & Analysis
- Solution Evaluation

1. Problem Overview

MC Sonae

With over 35 years of experience, MC is leading company in the food retail sector in Portugal.

More than **5.978M €** **turnover**

+ 4M customers served weekly

Over the last three decades, MC has been strengthening its **leading position** in the **food retail sector** in Portugal.
MC's activity began in 1985, within the Sonae Group, with the opening of the first hypermarket in Portugal. Today MC owns **12 brands** and has a vast portfolio of high quality products and services.

Network with **+1.400 stores**

+ 38.000 employees

Addressing Sonae's Challenge: Optimizing Transportation Networks and Fleet for Expanding Proximity Stores

Challenge:
Optimize transport for cost-efficient expansion of proximity stores.

How?
Utilize data-driven logistics models to optimize routes and fleet allocation.

Main processes currently being used

Warehouses

Two picking methods:

(1) PBL: Supplier boxes sent to stores throughout the day (no stock)

(2) PBS: Supplier pallets stored in the warehouse, boxes picked per store

- Pallets grouped by store for shipment

Transport

- Warehouse staff load pallets
- Single truck may ship multiple stores
- Truck carries pallet groups for cross-docking

Cross-docking

- Pallets sent to cross-docking for grouping
- Platforms: Unloading, Ground Sorting, Loading
- Single truck may ship one or more stores

Store

- Daily deliveries at set time slots
- Separate slots for fresh and dry goods
- 6-day delivery frequency (Mon-Sat)

2. Task Breakdown

MC Sonae

Strategic Focus: Goals and Objectives

Minimize costs:
Find a solution with the lowest sum of fuel, driver, vehicle and warehouse costs.

How?
Optimize key cost drivers in logistics

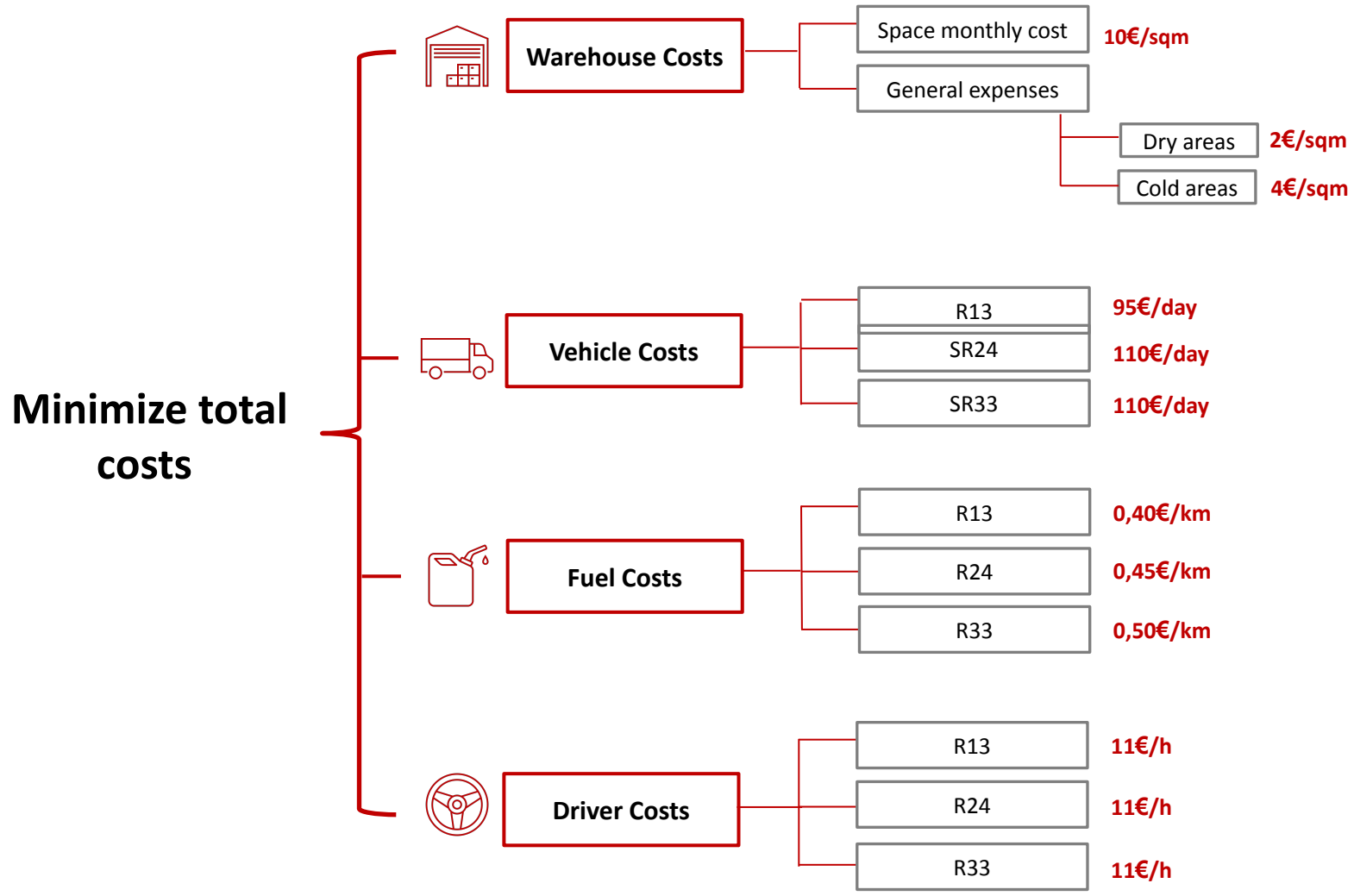
100% service level:
Optimize transport for cost-efficient expansion of proximity stores.

How?
Efficient Order Assignment, Timely Arrivals, Loading Optimization, and Delivery Time Adherence

Maximize efficiency:
Prioritize Early Deliveries, Load Trucks Fully, Optimize Fleet Size.

How?
Optimize delivery times, minimize underfilled trucks & number of trucks used

To minimize total costs, we must optimize key cost drivers in logistics



There are 3 aspects related to vehicles that need to be considered

- Pallet capacity
- Fuel costs
- Maintenance costs

Driver costs are 11€/h for all vehicles

Pallet capacity: 13

Fuel costs: 0,40€/km

Maintenance costs: 95€/day

Pallet capacity: 24

Fuel costs: 0,45€/km

Maintenance costs: 110€/day

Pallet capacity: 33

Fuel costs: 0,50€/km

Maintenance costs: 110€/day

Group Clustering Analysis: Directional Insights

North
Groups - 1, 2, 3, 4

West
Groups - 6, 7, 8

South
Groups - 5, 9, 10

- Deliveries to each direction could be delivered together and sorted at a central cross-docking warehouse.

However, **stores** of **different groups can't be grouped** in the **same truck.**

- This is also only beneficial if one or more of the groups in that direction have significant variation of store vehicle size restrictions.

Group Clustering Analysis: Distance from DC

Close < 100km

Groups: 1, 3, 4, 6, 7, 8

- For **nearby locations**, **fuel** and **driver costs** will usually be **lower** due to shorter trips.
- **Trips** to these locations will be **easier** to **plan** and **may not require cross docking** because cross docking should help save on the costs of travelling to the group.

Far > 100km away

Groups: 2, 5, 9, 10

- For **far away locations** it might be good to consider **2 drivers** to allow for 20h journeys and **save** on **fleet size** and **fuel costs**.
- Also **if** a **distant group** has **many vehicle size limits** then c**ross-docking might allow** for **completing** the long distance **trip** with **fewer vehicles**.

Group Clustering Analysis: Average store distance

Close < 20km

Groups: 1, 7, 8

Deliveries to these groups can likely **include many stores** as this would **not add much time** and **saves** on **vehicle costs** and **long distance journeys** to the group.

Medium < 50km

Groups: 3, 6, 9

The **advice** for these groups will lie **somewhere between** the **close** and **far** store **distances** and is case dependent.

Far ≤ 100km

Groups: 2, 4, 5, 10

For these groups **adding more stores** will **result** in **longer trips** which could **benefit** from adding a **second driver** to allow for more time.

Clustering Groups - Vehicle type restrictions

All SR33

Groups - 1, 3, 4, 5, 9

Since all truck requirements are SR33, the best approach is to perform all deliveries with SR33 trucks and distribute the pallets among them such that they have to visit as few stores as possible.

Mostly SR33

Groups – 2 (24:1), 6 (24:1,13:2), 8 (24:2,13:1), 10 (24:1)

Since in these cases the size limit is usually due to 1-3 stores, the recommendation would be to combine orders to these stores in particular. Making use of an approach such as cross-docking will not help.

Much Variation

Group - 7 (33:25,24:12,13:57)

In group 7 there is a significant amount of vehicle limited stores and therefore many smaller vehicles will have to make the long journey to the location. **Cross-docking might help here…**

Cost Analysis: Cross-Docking Facility

Given information:

- Warehouse space monthly cost 10€/sqm
- Minimum area 300 sqm
- Warehouse operator cost 10€/h
- Expenses for dry area 2€/sqm
- Expenses for fresh area 4€/sqm
- Average unloading time = 15min + 60sec/pal
- Average sorting time = 120sec/pal
- Average loading time = 15min + 60sec/pal

Let’s calculate costs of a cross-docking warehouse when dry and fresh areas are both 100 sqm.

Calculating the Costs of a cross-docking location

Total Warehouse Space Area

- In these calculations we use a total area of 500 sqm ⟶ This includes 100 sqm for both dry and fresh areas.

Total Warehouse Space Cost

- Total Warehouse Space Cost = 500 sqm x 10 €/sqm = **5000€**

Warehouse General Expenses

- Dry Area Expenses= 200€
- Fresh Area Expenses = 400€

Warehouse Space Monthly Cost

- Warehouse Space Monthly Cost = 500 sqm x 10 €/sqm = **5000€**

For group 7, the most pallets on a day that would go through the cross docking platform are:

- Dry: 942 pallets
- Fresh: 697 pallets

Total expenses

Construction Cost = Total Warehouse Space Cost + Dry Area Expenses + Fresh Area Expenses

Construction Cost = 5000€ + 200€ + 400€ = **5600€**

Monthly cost = 5000€ + pallet processing costs

Pallet processing costs:

= WH Operator cost x Time spent on processing

Calculating the Costs of a transport to the cross-docking center

55km at 75 km/h
from DC to CDP
= **0.733 hours**

Cap	Fuel	Driver	Vehicle
33	0,50€	11,0€	110,0€
24	0,45€	11,0€	110,0€
13	0,40€	11,0€	95,0€

These are optimal planning outcomes where all pallets perfectly fit and there is no mixing, but the impact should be comparable to reality!

<u>Maximum in a day:</u>
689 dry pallets with limit 13 ⟶ 53 trucks
529 fresh pallets with limit 13 ⟶ 41 trucks
295 dry pallets with limit 24 ⟶ 13 trucks
179 fresh pallets with limit 24 ⟶ 8 trucks

Total = 93 x cap13-Trucks & 21 x cap24-Trucks

<u>Those same pallets with 33 limit:</u>
942 dry pallets but 33 limit ⟶ 29
697 fresh pallets but 33 limit ⟶ 22

Total = 51 trucks

Fuel = 55 *(0.4 * 93 + 0.45 * 21) = **2565.75€**
Driver = 0.733 * 11 * (93 + 21) = **919.182€**
Vehicle = (93*95+21*110) * 6 = **66870€**

Fuel = 55 *0.5 * 51 = **1402.50€**
Driver = 0.733 * 11 * 51 = **411.21€**
Vehicle = 51*110 * 6 = **33660€**

Comparing Total costs for transport from DC to cross-docking centre

Without cross docking:
2565.75€ + 919.18€ + 66870€
= **70354.93€**

With cross docking:
1402.50€ + 411.21€ + 33660€
= **35473.71€**

Saving per week:
70354.93€ - 35473.71€ = **34881.22€**

We **assume** that there will be **no savings** driving from the cross docking location to the stores **as** the **original limited capacity fleet** is still **needed** for that…

However, the savings will be less than this due to the fleet size!

- We will still need to use the same fleet whether we use a cross docking centre or not.
  → The same trucks that would have take the goods to the cross-docking centre still need to transport it to the stores.
- This only adds some complication in tracking and planning where trucks need to be because we have to send the SR33 trucks back to the DC and make sure the smaller trucks are ready to pick up the goods from the cross-docking centre.
- It might even increase the overall fleet size to introduce cross docking as it occupies trucks that could have been used elsewhere.

Comparing fuel and driver costs for transport from DC to cross-docking centre

Without cross docking:
2565.75€ + 919.18€ =
3484.93€

With cross docking:
1402.50€ + 411.21€ =
1813.71€

Saving per week:
3484.93€ - 1813.71€ = **1671.22€**

Very low, does this compensate for the costs of building and operating a cross docking centre?

Delivery Priority Strategy: Location and Order Planning

Which locations should be delivered to first?
There are two factors to this decision:

Delivering to **nearby store groups first** means that

- trucks will **return faster** for the next delivery,
- thereby **reducing fleet size** and **costs**

or

Delivering to **far store groups first** means that

- the **long trips** will be **addressed first** and
- **increases** the **chance** that the **fresh** and **dry** daily **deadlines** are **met**

→ Since the **first goal** is to **minimize costs** we will **initially attempt** to **deliver** to **nearby groups** and **if time optimizations** are **still needed** we can **look** at **alternative approaches**

Within a group, which orders should be planned first?

We can plan them in order of two values: capacity limit and pallet volume.

- **Capacity Limit**: The order in which we plan these should be insignificant but we should try to combine orders with the same capacity limits to avoid transporting pallets with a smaller vehicle than necessary.
- **Pallet Volume**

To attain a 100% service level, specific success conditions were essential:

Full Assignment
All orders have been assigned to a delivery.

Timely arrival

- No fresh order arrives later than the fresh time limit
  → (8:00am on the day after the order)
- No dry order arrives later than the dry time limit
  → (23:59 on the day after the order)
- If a delivery contains part of an order, the remainder of the order must arrive within 4 hours

Loading dock capacity

- 40 loading docks can be in use at a time for dispatch

Delivery time limits: Single/Tandem Driver Constraints

- No delivery trip can take longer than 10 hours
- Trip limit may be extended to 20 hours if two drivers are assigned

If any of these are not met the solution fails

Enabling Success: Meeting 100% Service Level Conditions

Full Assignment
All orders have been assigned to a delivery.

- **Each row** in the **orders table** is **treated** as an **order**
  - whenever pallets are assigned to a row in the output table they are subtracted from the pallets column in the input table
- The **total number** of **fresh and dry pallets planned** is **summed** and **compared** to the **original sum** of fresh and dry demand
- Any **mismatch** is a **failure**

Timely arrival

- No fresh order arrives later than the fresh time limit
  → (8:00am on the day after the order)
- No dry order arrives later than the dry time limit
  → (23:59 on the day after the order)
- If a delivery contains part of an order, the remainder of the order must arrive within 4 hours

- All **deliveries starting after 19:00** must **contain fresh goods** unless fresh demand is satisfied
- **Between 17:00 and 19:00** try to plan **as many dry deliveries** as **possible**
  - But don't add part of an order too soon before 19:00 to a trip so that it has to wait until fresh demand is satisfied again is complete
- **When mixing**, **don't add dry goods if** it means **exceeding** the **fresh goods time limit** of 8am

Enabling Success: Meeting 100% Service Level Conditions

Loading dock capacity
40 loading docks can be in use at a time for dispatch.

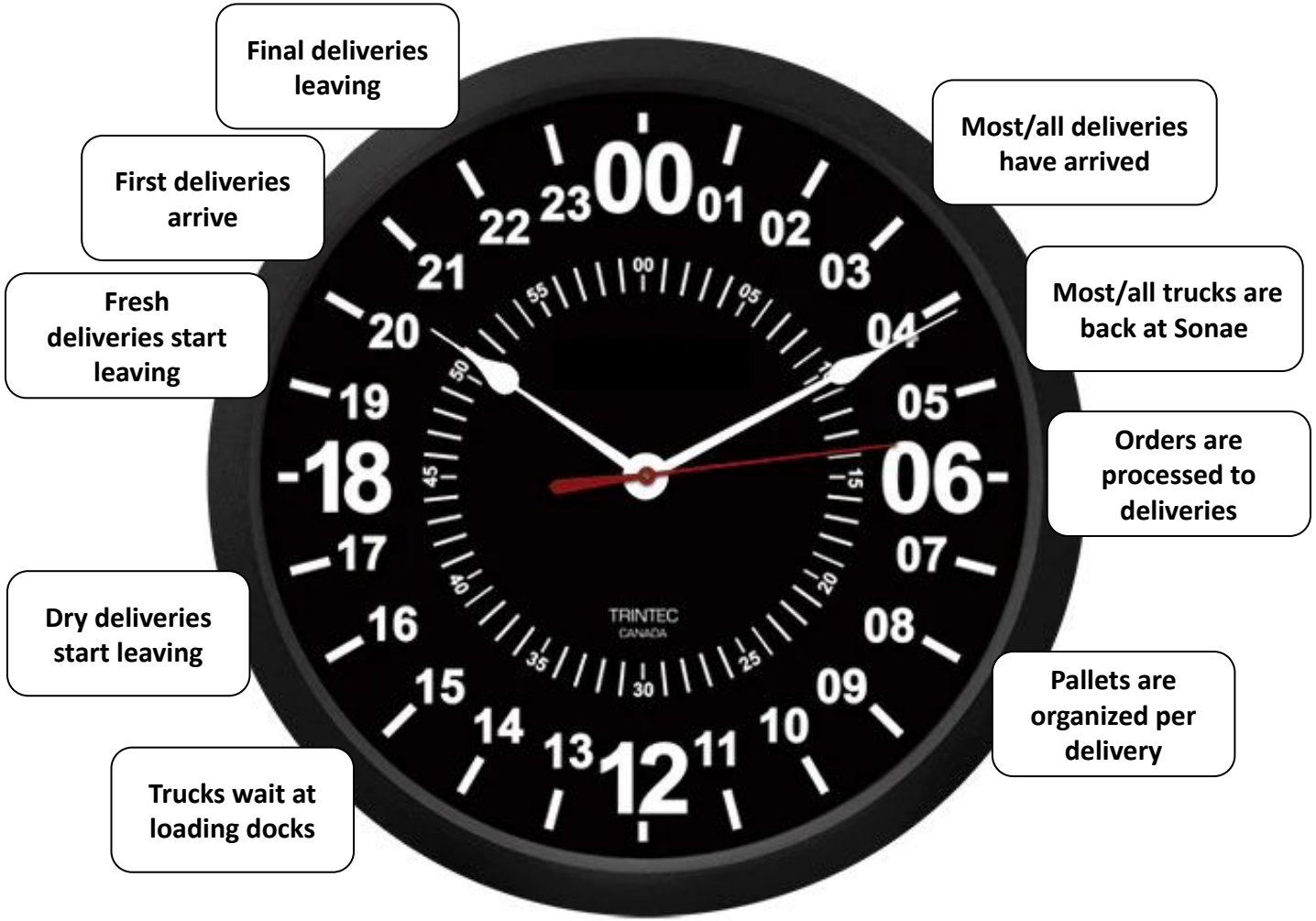
- The **first 40 deliveries** of the day **always start loading** at **17:00**
- **Any deliveries after** that can **start after any** of the **previous deliveries** have finished loading

Delivery time limits: Single/Tandem Driver Constraints

- No delivery trip can take longer than 10 hours
- Trip limit may be extended to 20 hours if two drivers are assigned

- **When adding pallets** or **adding** a **new store** to the list, always **check if** this **addition** will **lead** to the **trip time exceeding** 10 hours
- **If** there are **two drivers** available then **extend** the **limit** to **20 hours** but **double** the **driver costs** for that trip

Optimizing Efficiency: Daily Delivery Schedule & Data Insights



01/05/2023 is Monday

Picking

- Orders are picked 6 days a week (Mon-Sat)
- Dry goods are ready to be shipped everyday at 5 p.m (d)
- Fresh goods are ready to be shipped everyday at 7 p.m (d)

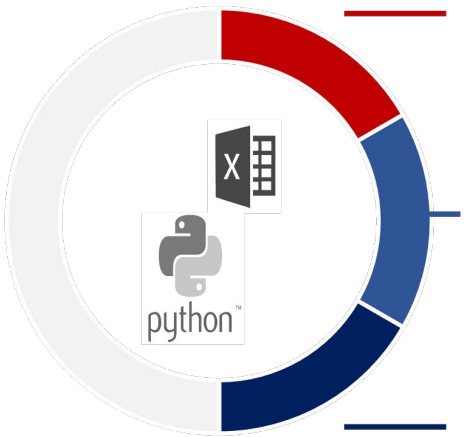
Store restrictions

- Dry goods may be delivered during all day (d or d+1)
- All fresh goods must be delivered until 8 a.m (d+1)
- On the same day, all goods of the same type must be delivered in a 4h time slot of more than one delivery is made

3. Dataset Analysis & Data Preparation

MC Sonae

The data available was treated and processed in two different work tools: Excel and Python



Distances.xlsx
Distances between warehouse, region center & stores

Pallets.xlsx
Number of pallets by store per week

Vehicle size limits.xlsx
Maximum vehicle size for the stores

We merged the three Excel files into one single document called MasterWorksheet(alldata).xlsx for easier data processing and analysis

Through Excel:

- Each order was matched to its respective store along with:
- Vehicle costs, which have been added by inserting a column each for **vehicle (pallet size)**, **fuel costs**, **driver costs** and **vehicle costs**.

Through Python:

- Create the initial `plan`-list and organize the orders for each days.
- Organising deliveries by store- groups.
- Planning orders for divided groups of dry and fresh goods.
- Calculating remaining demand and repeating until demand is satisfied.
- Add information to existing `Plan`-list.
- Using the codes to plan the different scenarios.

Visualizing Order Details: Exploring the Input Table

Columns of the input table – each row is an order

- **Delivery date (d+1):** Date the order should depart on
- **Store:** Unique number value to indicate the store
- **Type:** type of goods of the order Either Fresh or Dry
- **Pallets:** number of pallets in that order
- **Type:** Shop type e.g. Continente (not used)
- **Group:** Region group number of the store, determines distance
- **Vehicle size limit:** Maximum size label of vehicle that can perform the last-mile delivery to this store
- **Distance between DC and center of group [km]:** Distance to region
- **Average distance between stores [km]:** Distance between stores in region
- **Pallet Capacity:** numerical version of Vehicle size limit value
- **Fuel costs (€/km):** Fuel cost per kilometer of vehicle (relative to size limit value)
- **Driver Costs (€/h):** Wage per hour per driver
- **Vehicle Costs (€/day):** amount paid per day for entire fleet needed (price relative to size limit value)

delivery date (d+1)	store	type	pallets	Type	Group	Vehicle size limit
02/may	5	DRY GOODS	69	continente	1	SR33
02/may	5763	DRY GOODS	27	continente bom dia	1	SR33
02/may	269	DRY GOODS	26	continente modelo	1	SR33
02/may	5	FRESH GOODS	25	continente	1	SR33
02/may	494	DRY GOODS	22	continente modelo	1	SR33
02/may	3698	DRY GOODS	20	continente bom dia	1	SR33
02/may	4121	DRY GOODS	19	continente bom dia	1	SR33
02/may	269	FRESH GOODS	14	continente modelo	1	SR33
02/may	5763	FRESH GOODS	13	continente bom dia	1	SR33
02/may	494	FRESH GOODS	11	continente modelo	1	SR33
02/may	4121	FRESH GOODS	11	continente bom dia	1	SR33
02/may	3698	FRESH GOODS	9	continente bom dia	1	SR33
03/may	5	DRY GOODS	106	continente	1	SR33
03/may	269	DRY GOODS	30	continente modelo	1	SR33
03/may	4121	DRY GOODS	22	continente bom dia	1	SR33
03/may	5	FRESH GOODS	21	continente	1	SR33
03/may	494	DRY GOODS	21	continente modelo	1	SR33
03/may	5763	DRY GOODS	20	continente bom dia	1	SR33
03/may	3698	DRY GOODS	17	continente bom dia	1	SR33
03/may	5763	FRESH GOODS	13	continente bom dia	1	SR33
03/may	269	FRESH GOODS	12	continente modelo	1	SR33
03/may	494	FRESH GOODS	11	continente modelo	1	SR33
03/may	4121	FRESH GOODS	11	continente bom dia	1	SR33
03/may	3698	FRESH GOODS	10	continente bom dia	1	SR33
04/may	5	DRY GOODS	81	continente	1	SR33
04/may	269	DRY GOODS	32	continente modelo	1	SR33
04/may	494	DRY GOODS	31	continente modelo	1	SR33
04/may	5	FRESH GOODS	27	continente	1	SR33
04/may	4121	DRY GOODS	27	continente bom dia	1	SR33

Excel Input Sample: Total Orders and Pallet Count

delivery date (d+1)	store	type	pallets	Type	Group	Vehicle size limit	distance between DC and center of group [km]*	average distance between stores [km]	Pallet Capacity	Fuel costs (€/km)	Driver Costs (€/h)	Vehicle Costs (€/day)
02/may	5	DRY GOODS	69	continente	1	SR33	93	20	33	0,50	11	110
07/may	269	FRESH GOODS	17	continente modelo	1	SR33	93	20	33	0,50	11	110
05/may	212	FRESH GOODS	22	continente	2	SR33	212	100	33	0,50	11	110
…	…	…	…	…	…	…	…	…	…	…	…	…

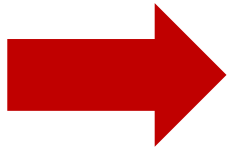
- In total there are **2425 orders** with in a sum of **43 995 pallets** that need to be delivered
- The **average truck size** is **26.35 pallets** which gives an optimistic **reference number** of **trips**:
  - 43995 / 26.35 = **1669.63**

4. Solution Description

MC Sonae

We wrote a python script that creates a table with each delivery trip as a row

Table of orders:
each row is an order



Python Algorithm:
Assigns each order's pallets to a trip

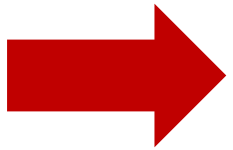


Table of trips:
each row is a planned delivery of 1 truck

delivery date (d+1)	store	type	pallets
02/may	5	DRY GOODS	69
02/may	269	FRESH GOODS	14

delivery date (d+1)	start_time	Pallets_in_vehicle	good_type
02/may	17	33	DRY
02/may	269	FRESH GOODS	14

All output columns on the next slide

Important:

The calculations and plans are based on average times for driving and packing. Room should be made for errors and delays.

Delivery Snapshot: Output Table Headers as Individual Deliveries

- **day**: Date the order should depart on (Date at 00:00)
- **start_time**: Time at which filling the truck should start (Date time)
- **loading_time**: Expected amount of time taken to fill the truck (hours)
- **v_capacity**: Type of vehicle used and it's capacity for pallets (units)
- **Pallets_in_vehicle**: how many pallets should be added to the vehicle (units)
- **good_type**: what type of pallets should be added (FRESH, DRY OR FRESH AND DRY)
- **group**: Number identifier of group (1-10)
- **Store_count**: Number of stores the vehicle will deliver to (count)
- **recieving_stores**: List of stores the vehicle will visit (list of number identifiers)
- **relative_pallets**: In the order of receiveing_stores, number of pallets to deliver each store (list of units)
- **trip_duration**: Expected time take to fill, drive, deliver and return to Sonae (hours)
- **return_time**: Time the vehicle is expected to arrive back at Sonae (Date time)

- **last_delivery_time**: Time the vehicle is expected to complete the final delivery (Date time)
- **Dry_demand**: Remaining dry demand for this group after assigning these pallets (units)
- **Fresh_demand**: Remaining fresh demand for this group after assigning these pallets (units)
- **reason_for_next**: An explanation for why more pallets were not added to the vehicle (text)
- **Fuel_cost**: Total cost of fuel for driving the vehicle (€)
- **Driver_cost**: total wage cost for the entire trip duration for the driver, multiplied by number of drivers (€)
- **active_13trucks**: Number of 13 capacity trucks that are currently in use (count)
- **active_24trucks**: Number of 24 capacity trucks that are currently in use (count)
- **active_33trucks**: Number of 33 capacity trucks that are currently in use (count)
- **remaining_v_space**: How much capacity of the vehicle was not utilized (units)

Example Output Showcase: Illustrating Results

ⓘ The table, featuring numerous columns, has its **second half displayed below** for enhanced readability

day	start_ time	loading_ time	v_ capacity	Pallets_ in_ vehicle	good_ type	group	Store_ count	recievin g_stores	relative_ pallets	trip_ duration	return_ time	last_ delivery _ time
2023-05-02 00:00:00	2023-05-02 17:00:00	0.538889	13	13	DRY	7	3	[9915, 9905, 4593]	[3, 9, 1]	3.038888889	2023-05-02 20:02:20	2023-05-02 19:18:20

day	Dry_ demand	Fresh_ demand	reason_ for_next	Fuel_ cost	Driver_ cost	active_ 13trucks	active_ 24trucks	active_ 33trucks	remaini ng_ v_spac e
2023-05-02 00:00:00	1652	1057	full vehicle	51.2	33.42778	12	1	20	0

Python Delivery Planner: A Deep Dive into its Operational Workflow

On each day:

Between 17:00 and 19:00
- Plan **trips** for **dry deliveries**

After 19:00
- Plan **trips** for **fresh deliveries**

After all Fresh goods are planned
- Plan **remaining trips** for **dry deliveries**

Considerations for deliveries:

- Total trip cannot exceed 10 hours
  - including (un)loading and driving between stores
  - A trip can deliver to multiple stores if adding that store doesn't exceed the time
- Number of pallets cannot exceed vehicle capacity
- Fresh and Dry deliveries cannot arrive later than their time limit of 8am and midnight on the day after, respectively

Delivery Planning Logic: Iterative Code Structure & Constraints

This **step-by-step explanation** covers the **main structure of the code** and the **logic** for **planning deliveries** based on fresh and dry goods demand, location groups, and various constraints.

It's designed to iteratively plan deliveries until the total demand for each day is satisfied.

1. Initialize Plan and Loop Through Delivery Days:
   a. Create an empty list called `plan` to store the delivery schedule for each day.
   b. Use a loop to iterate through each day in the `delivery_days`.
2. Filter Orders for the Current Day:
   a. Select orders from the dataframe (`df`) where the 'delivery date (d+1)' matches the current day.
   b. Create an empty dataframe called `deliveries` to store the planned deliveries for the day.
   c. Initialize a variable `demand_satisfied` to track the total demand satisfied.
3. Group Orders by Location Group:
   a. Group the orders for the day by the 'Group' column to plan deliveries for each location group separately.
4. Iterate Through Location Groups:
   a. Loop through each location group and plan deliveries for that group.
   b. Calculate the total demand for fresh and dry goods for the current group.
5. Plan Deliveries for Dry Goods (between 17:00 and 19:00 or after all Fresh has been planned):
   a. Iterate through orders for dry goods in the group.
   b. Plan trips to satisfy dry goods demand, considering time constraints and capacity limits.
   c. Calculate costs for each planned trip.
6. Plan Deliveries for Fresh Goods (after 19:00):
   a. Iterate through orders for fresh goods in the group.
   b. Plan trips to satisfy fresh goods demand, considering time constraints and capacity limits.
   c. Calculate costs for each planned trip.
7. Update Remaining Demand:
   a. Calculate how much fresh and dry goods demand remains after planning deliveries for the current group.
8. Repeat Until All Demand Satisfied:
   a. Continue planning deliveries for different groups until the total demand for the day is satisfied.
9. Add Planned Deliveries to the Plan List:
   a. Append the planned deliveries for the day to the `plan` list.
10. End of Loop:
    a. Once deliveries are planned for all days, the loop ends

Late-Day Mix: Fresh-Dry Fusion After 7 PM

On each day:

Between 17:00 and 19:00
- Plan **trips** for **dry deliveries**

After 19:00
- Plan **trips** for **fresh deliveries**
- **If the truck has space, check for dry goods ordered by the same store**

After all Fresh goods are planned
- Plan **remaining trips** for **dry deliveries**

Considerations for deliveries:

- Total trip cannot exceed 10 hours
  - including (un)loading and driving between stores
  - A trip can deliver to multiple stores if adding that store doesn't exceed the time
- Number of pallets cannot exceed vehicle capacity
- Fresh and Dry deliveries cannot arrive later than their time limit of 8am and midnight on the day after, respectively

Mixed Goods Delivery Logic: Code Structure & Iterative Planning

This **step-by-step explanation** covers the main structure of the code for **planning mixed deliveries of fresh and dry goods**, based on their location groups, and various constraints.

It's designed to iteratively plan deliveries until the total demand for each day is satisfied.

1. Initialize Plan and Loop Through Delivery Days:
   a. Create an empty list called `plan` to store the delivery schedule for each day.
   b. Use a loop to iterate through each day in the `delivery_days`.
2. Filter Orders for the Current Day:
   a. Select orders from the dataframe (`df`) where the 'delivery date (d+1)' matches the current day.
   b. Create an empty dataframe called `deliveries` to store the planned deliveries for the day.
   c. Initialize a variable `demand_satisfied` to track the total demand satisfied.
3. Group Orders by Location Group:
   a. Group the orders for the day by the 'Group' column to plan deliveries for each location group separately.
4. Iterate Through Location Groups:
   a. Loop through each location group and plan deliveries for that group.
   b. Calculate the total demand for fresh and dry goods for the current group.
5. Plan Deliveries for Dry Goods (between 17:00 and 19:00 or after all Fresh has been planned):
   a. Iterate through orders for dry goods in the group.
   b. Plan trips to satisfy dry goods demand, considering time constraints and capacity limits.
   c. Calculate costs for each planned trip.
6. Plan Deliveries for Fresh Goods (after 19:00):
   a. Iterate through orders for fresh goods in the group.
   b. Plan trips to satisfy fresh goods demand, considering time constraints and capacity limits.
   c. Add dry goods from the same store, considering time constraints and capacity limits.
   d. Calculate costs for each planned trip.
7. Update Remaining Demand:
   a. Calculate how much fresh and dry goods demand remains after planning deliveries for the current group.
8. Repeat Until All Demand Satisfied:
   a. Continue planning deliveries for different groups until the total demand for the day is satisfied.
9. Add Planned Deliveries to the Plan List:
   a. Append the planned deliveries for the day to the `plan` list.
10. End of Loop:
    a. Once deliveries are planned for all days, the loop ends

Step-by-Step Demonstration

Step-by-Step Demonstration

For each day:

```
#make a loop, it repeats everything separately for each day and adds the schedule to plan
for day in delivery_days:
  print(day)
  # Filter the DataFrame to select rows where 'delivery date (d+1)' matches the desired delivery day
  day_orders = all_orders[all_orders['delivery date (d+1)'] == day].copy()
  no = pd.to_datetime("2150") #no is a date value far in the future which is assigned to rows that are not partially satisfied (for sorting)
  yes = pd.to_datetime("2200") #yes is a date value far in the future which is assigned to rows that are fully satisfied (for sorting)
  day_orders['order_assigned'] = no #make a column with indication if the order is assigned to a truck [no, yes or a time when part of the order was picked up]
      #so when you sort the order list by order_assigned you get them in order partial, unassigned, assigned

  day = pd.to_datetime(day + " 2023") #convert the day value to a date number rather than the "may 01/2023" format

  #create an empty table for all deliveries on that day, each row is one truck
  deliveries = pd.DataFrame(columns=["day", "start_time", "loading_time", "V_capacity", "Pallets_in_vehicle", "good_type", "group", "Store_count","recieving_stores", "relative_pallets", "trip_duration", "return_time", "last_delivery_time","Dry_demand",
  #start the day with no deliveries planned (so no demand satisfied)
  demand_satisfied = 0

  #here we order the groups in which they will be planned
  if group_order == "close first": #we order the orders by distance of group from DC, so that we handle the nearest orders first ···

  elif group_order == "far first": #we order the orders by distance of group from DC, so that we handle the furthest orders first ···

  elif  group_order == "count up": #we order the orders by group number, so that we handle group 1's orders first ···

  elif  group_order == "random": #if interested in testing different group orders there is a random option too···

  # Group by the 'Group' column, to make a separate table for each location, we do this so that we plan each separately
  groups = []
  #makes a list of tables, one table for each group
  for group_name, group_data in day_orders.groupby('Group', sort=False):···

  delivery_count =  0 #no deliveries planned yet for this day
  start_time = day + pd.to_timedelta(Dry_starttime, unit='H') #set that start of planning to 17:00

  #sum how many fresh and dry goods pallets need to be delivered today
  Starting_fresh_demand = day_orders['pallets'][day_orders['type'] == "FRESH GOODS"].sum()
  Starting_dry_demand = day_orders['pallets'][day_orders['type'] == "DRY GOODS"].sum()
  total_day_demand = day_orders['pallets'].sum()

  #the fresh and dry_demand variables will be updated as we plan deliveries (demand will go down)
  Fresh_demand = Starting_fresh_demand
  Dry_demand = Starting_dry_demand

  #this loop below will repeat as long as all demand has not been satisfied
  while demand_satisfied < total_day_demand:···


  print("Ordered fresh:",Starting_fresh_demand)
  print("Ordered dry:",Starting_dry_demand)
  print("Ordered demand:",total_day_demand)

  print("Scheduled demand:",demand_satisfied)

  plan = pd.concat([plan, deliveries]) #add to the plan delivery table

return plan
```

Step-by-Step Demonstration

Until planned pallets equals total demand for that day:

```
#this loop below will repeat as long as all demand has not been satisfied
while demand_satisfied < total_day_demand:

  # So here we check through each group at a time to plan deliveries
  for group_data in groups:···


  demand_satisfied = deliveries["Pallets_in_vehicle"].sum() #calculate how maybe pallets are assigned to a truck

  # set values for remaining demand values
  remaining_demand = 0
  remaining_fresh_demand = 0
  remaining_dry_demand = 0

  #check through all groups and sum up how much demand remains in total for that day
  for group_data in groups:
    remaining_fresh_demand += group_data['pallets'][group_data['type'] == "FRESH GOODS"].sum()
    remaining_dry_demand += group_data['pallets'][group_data['type'] == "DRY GOODS"].sum()
    remaining_demand += group_data['pallets'].sum()
```

Step-by-Step Demonstration

For each group:

```
# So here we check through each group at a time to plan deliveries
for group_data in groups:
  group_name = group_data["Group"].iat[0] #set the current group we are checking


  #Calculate how much fresh and dry goods need to be assigned, this will change every time this loop is repeated
  Fresh_demand = group_data['pallets'][group_data['type'] == "FRESH GOODS"].sum()
  Dry_demand = group_data['pallets'][group_data['type'] == "DRY GOODS"].sum()

  #sort the group by dry first, then partial/unassigned orders and then ascending numbers of pallets, we use the first row of the table
  group_data.sort_values(by=['type', 'order_assigned', 'pallets'], ascending=[True, True, True], inplace=True)
  group_data.reset_index(drop=True, inplace=True) #since the order of rows has changed, this sets the row numbers back to 0, 1, 2, 3...(easier for code to know that row 0 is first row)

  #We will check if Dry demand needs to be satisfied in this group if the time is before 19:00 or no fresh demand remains
  #after 19:00, fresh demand needs to
  while Dry_demand > 0  and (Fresh_demand == 0 or start_time < day + pd.to_timedelta(Fresh_starttime, unit='H')):···

  #calculate the number of planned pallets we have
  demand_satisfied = deliveries["Pallets_in_vehicle"].sum()

  #if we have not been able to add partial orders due to it being too close to 19:00, past 19:00 or dry demand is fully planned before 19:00
  if reason_for_next == "too late to create partial dry trip" or reason_for_next == "19:00 threshold" or reason_for_next == "dry demand satisfied":
    reason_for_next = "moving to fresh" #next time we come to this line of code we move past, to fresh demand planning
    break #restart the for groups loop back to the first group in the list

  #sort the group by Fresh first, then partial/unassigned orders and then ascending numbers of pallets, we use the first row of the table
  group_data.sort_values(by=['type', 'order_assigned', 'pallets'], ascending=[False, True, True], inplace=True)
  group_data.reset_index(drop=True, inplace=True) #since the order of rows has changed, this sets the row numbers back to 0, 1, 2, 3...(easier for code to know that row 0 is first row)

  #Calculate how much fresh goods need to be assigned
  Fresh_demand = group_data['pallets'][group_data['type'] == "FRESH GOODS"].sum()

  #Check if more fresh demand needs to be satisfied in this group and if it's 19:00 or later
  while Fresh_demand > 0 and start_time >= day + pd.to_timedelta(Fresh_starttime, unit='H'):#everything in this loop is for one truck at a time and it repeats for the next truck···

  demand_satisfied = deliveries["Pallets_in_vehicle"].sum()

  #Calculate how much fresh goods need to be assigned
  Fresh_demand = group_data['pallets'][group_data['type'] == "FRESH GOODS"].sum()
  #Calculate how much dry goods need to be assigned
  Dry_demand = group_data['pallets'][group_data['type'] == "DRY GOODS"].sum()
```

Step-by-Step Demonstration

Between 17:00 and 19:00 and after all fresh demand is planned:

```
#We will check if Dry demand needs to be satisfied in this group if the time is before 19:00 or no fresh demand remains
#after 19:00, fresh demand needs to
while Dry_demand > 0  and (Fresh_demand == 0 or start_time < day + pd.to_timedelta(Fresh_starttime, unit='H')):
  #this is where the planning of a trip starts
  if delivery_count+1 > Loading_docks: #all loading docks start empty so once they are all in use we have to wait until one is free
    start_time = deliveries["start_time"].iat[delivery_count-Loading_docks] + pd.to_timedelta(deliveries["loading_time"].iat[delivery_count+1-Loading_docks], unit='H')
    #start_time is either set to 17 or once a loading_time has passed of a previous row it is set to the time a dock frees up
  else: start_time = day + pd.to_timedelta(Dry_starttime, unit='H')

  if start_time >= day + pd.to_timedelta(Fresh_starttime, unit='H') and Fresh_demand != 0:
    reason_for_next = "19:00 threshold"
    break

  Pallets_in_vehicle = 0 #set value to count how may pallets have been assigned to the vehicle
  delivery_count += 1 #start a new delivery += adds to the current value 1
  recieving_stores = [] #a list of the stores this truck will go to
  relative_pallets = [] #a list of the pallets per store (in order)

  v_capacity = group_data['Vehicle size limit'].iat[0]  #set the current delivery capacity limit to the first value in the orders table

  Store_count = 1 #count the number of stores to be visited

  #while loop to add goods to a truck, repeats until we can't add more to it (various reasons)
  while Pallets_in_vehicle < v_capacity: #is there still space in the truck?···

  if Pallets_in_vehicle > 0: #if there are any pallets added to this truck we can calculate costs
    #Calculate costs
    Fuel_cost = (group_data["distance between DC and center of group [km]*"].iat[0] * 2 + group_data["average distance between stores [km]"].iat[0] * (Store_count-1)) * truck_costs[v_capacity]["Fuel_Costs"]
    #if the trip takes less than an hour we only need 1 driver, if its more we pay for 2 drivers
    if trip_duration <= 10: Driver_cost = trip_duration * truck_costs[v_capacity]["Driver_Costs"]/drivers
    else: Driver_cost = trip_duration * truck_costs[v_capacity]["Driver_Costs"]


    # Define the data for the new row, "this is the column name" and to the right of those are the values for the row/column
    new_row_data = {"day": day, "start_time": start_time, "loading_time": loading_time, "v_capacity": v_capacity, "Pallets_in_vehicle": Pallets_in_vehicle, "good_type": "DRY", "group": group_name, "Store_count": Store_count,
        "recieving_stores": str(recieving_stores), "relative_pallets": str(relative_pallets), "trip_duration": trip_duration, "return_time": return_time, "last_delivery_time": last_delivery_time, "Dry_demand": Dry_demand,
        "Fresh_demand": Fresh_demand, "reason_for_next": reason_for_next, "Fuel_cost": Fuel_cost, "Driver_cost": Driver_cost,
    }

    # Create a new DataFrame row with the specified index and data
    new_row = pd.DataFrame(data=new_row_data, index=[delivery_count])

    # Concatenate the new row with the existing 'deliveries' DataFrame
    deliveries = pd.concat([deliveries, new_row])
  else: #if we did not manage to add pallets to this vehicle
    delivery_count -= 1 #subtract a delivery because no pallets could be assigned
    #it probably means its ready to wait for fresh orders to be ready, set start_time to 19:00
    start_time = day +  pd.to_timedelta(Fresh_starttime, unit='H') +  pd.to_timedelta(1, unit='m')
```

Step-by-Step Demonstration

Plan trips with dry goods

```
#while loop to add goods to a truck, repeats until we can't add more to it (various reasons)
while Pallets_in_vehicle < v_capacity: #is there still space in the truck?

  #need to check if adding pallets from the next row will change the size limit and then be too much
  if group_data['Vehicle size limit'].iat[0] < Pallets_in_vehicle:
    reason_for_next = "can't add more pallets due to size limit"
    Store_count -= 1 #we thought we could add another store after the last loop, but nvm, no space
    break #finish adding to truck and go to calculation of distance and costs of delivery
  added  = True #assume we will add pallets to this trip

  #Check if the current order is partially assigned, then calculate how long it has been waiting
  if group_data['order_assigned'].iat[0] < no: #less than no is any date and time earlier than 2150 (we set that vanlue for no), meaning it was partly planned at that time
    #check if any partial orders are being completed too late
    check_order(deliveries, group_data, 0, start_time, "DRY")

  #check if the store has ordered more than the truck can still hold, and that it's not too late to take part of an order (we don't want half orders to wait more than 4 hours)
  if group_data['pallets'].iat[0] > min(group_data['Vehicle size limit'].iat[0], v_capacity) - Pallets_in_vehicle and (start_time < day + pd.to_timedelta(18.3, unit='H') or start_time > day + pd.to_timedelta(19, unit='H')):
    #take the order volume the truck can still hold
    (variable) relative_pallets: list  size limit'].iat[0], v_capacity)  #update delivery capacity limit with the current store, capacity can only go down
    apacity - Pallets_in_vehicle #remove from order row, some goods still remain
    recieving_stores.append(group_data['store'].iat[0]) #add the current store number to the list (row 0 is first row)
    relative_pallets.append(v_capacity - Pallets_in_vehicle) #note the number of pallets for this store
    Pallets_in_vehicle += v_capacity - Pallets_in_vehicle #add to the vehicle (fill)
    group_data['order_assigned'].iat[0] = start_time #a partial order has been created at this start time


  #if there is enough space to hold the whole order, we can add it
  elif group_data['pallets'].iat[0] <= min(group_data['Vehicle size limit'].iat[0], v_capacity) - Pallets_in_vehicle:#if the truck can hold the whole order
    v_capacity = min(group_data['Vehicle size limit'].iat[0], v_capacity)  #update delivery capacity limit with the current store, capacity can only go down
    Pallets_in_vehicle += group_data['pallets'].iat[0] #add to the truck
    recieving_stores.append(group_data['store'].iat[0]) #add the current store number to the list (row 0 is first row)
    relative_pallets.append(group_data['pallets'].iat[0]) #note the number of pallets for this store
    group_data['pallets'].iat[0] = 0 #order of first row is satisfied
    group_data['order_assigned'].iat[0] = yes #this order is fully assigned

    #sort the group by dry first, then partial/unassigned orders and then ascending numbers of pallets, we use the first row of the table
    group_data.sort_values(by=['type', 'order_assigned', 'pallets'], ascending=[True, True, True], inplace=True)
    group_data.reset_index(drop=True, inplace=True) #since the order of rows has changed, this sets the row numbers back to 0, 1, 2, 3...(easier for code to know that row 0 is first row)

  #if there's not enough space for the whole order and it's too close to 19:00 add half, we add nothing
  else:
    Store_count -= 1 #we cancel adding this store's pallets because it's too late to add half an order
    added = False #indicate that nothing was added from this order

  #Calculate how many dry goods still need to be assigned in the current group
  Dry_demand = group_data['pallets'][group_data['type'] == "DRY GOODS"].sum()

  #calculate how long it would take to fill the truck
  loading_time = Average_loading_time + Pallets_in_vehicle * added_loading_time_per_pallet
  #calculate how long the entire trip will take
  trip_duration = loading_time + Pallets_in_vehicle * added_unloading_time_per_pallet + group_data["distance between DC and center of group [km]*"].iat[0] *2 / High_average_speed + group_data["average distance between stores [km]"].iat[0] * (Store_count-1) / Low_average_speed
  #calculate what time the driver completes the final delivery of their trip
  last_delivery_time = start_time +  pd.to_timedelta(trip_duration - group_data["distance between DC and center of group [km]*"].iat[0] / High_average_speed, unit='H')
  return_time = start_time + pd.to_timedelta(trip_duration, unit='H')  #and the time they arrive back at Sonae

  if Dry_demand == 0:#if there is no more dry demand then there is nothing left to add to the truck
    reason_for_next = "dry demand satisfied" #define why we stop filling
    break #stop filling this truck and leave the current loop to calculate trip costs

  #check if there is still space in the truck and if adding more would make the trip too long
  if Pallets_in_vehicle < v_capacity and trip_duration + group_data["average distance between stores [km]"].iat[0] / Low_average_speed + min((v_capacity - Pallets_in_vehicle),group_data['pallets'].iat[0]) * (added_unloading_time_per_pallet + added_loading_time_per_pallet) <= Maximium_trip_time:
    Store_count += 1 #We should be able to deliver to another store, so add one to the value

  #check if adding another store would make the trip too long
  if trip_duration + group_data["average distance between stores [km]"].iat[0] / Low_average_speed + min((v_capacity - Pallets_in_vehicle),group_data['pallets'].iat[0])  * (added_unloading_time_per_pallet + added_loading_time_per_pallet) > Maximium_trip_time:
    reason_for_next = "long trip" #define why we stop adding to this truck
    break #stop filling this truck and leave the current loop to calculate trip costs

  #check if adding another store would make the last delivery too late (deadline)
  if last_delivery_time + pd.to_timedelta(group_data["average distance between stores [km]"].iat[0] / Low_average_speed + min((v_capacity - Pallets_in_vehicle),group_data['pallets'].iat[0])  * (added_unloading_time_per_pallet + added_loading_time_per_pallet), unit='H') > day + pd.to_timedelta(Dry_limit_time, unit='H') :
    reason_for_next = "Adds too much time for dry trip limit" #define why we stopp adding to this truck
    break #stop filling this truck and leave the current loop to calculate trip costs

  if added == False: #nothing was added to the truck this loop
    reason_for_next = "too late to create partial dry trip" #adding part of the top order to this truck would make it wait too long for the second part
    break #so we can stop adding to this truck
  reason_for_next = "full vehicle" #if all other checks have passed, the only remaining reason we stop filling a truck is because it's full
```

Step-by-Step Demonstration

After 19:00:

```
#Check if more fresh demand needs to be satisfied in this group and if it's 19:00 or later
while Fresh_demand > 0 and start_time >= day + pd.to_timedelta(Fresh_starttime, unit='H'):#everything in this loop is for one truck at a time and it repeats for the next truck
  #this is where the planning of a trip starts
  if delivery_count+1 > Loading_docks: #all loading docks start empty so once they are all in use we have to wait until one is free
    start_time = max(day + pd.to_timedelta(Fresh_starttime, unit='H') ,deliveries["start_time"].iat[delivery_count-Loading_docks] + pd.to_timedelta(deliveries["loading_time"].iat[delivery_count+1-Loading_docks], unit='H') )
    #start_time is either set to 19 or once a loading_time has passed of a previous row it is set to the time a dock frees up
  else: start_time = day + pd.to_timedelta(Fresh_starttime, unit='H') #this only happens if there was not enough dry demand to use up 40 loading docks
  if start_time < day + pd.to_timedelta(Fresh_starttime, unit='H'): break #so we exit this fresh demand while loop, nothing below here happens and we eventually come back to the loops above

  Pallets_in_vehicle = 0 #set value to count how may pallets have been assigned to the vehicle
  delivery_count += 1 #start a new delivery += adds to the current value 1
  recieving_stores = [] #a list of the stores this truck will go to
  relative_pallets = []  #a list of the pallets per store (in order)

  type ="FRESH" #the type of good we are planning in this loop
  v_capacity = group_data['Vehicle size limit'].iat[0]  #set the current delivery capacity limit to the first value in the orders table

  Store_count = 1 #count the number of stores to be visited


  #while loop to add goods to a truck, repeats until we can't add more to it (various reasons)
  while Pallets_in_vehicle < v_capacity: #is there still space in the truck?···

  #Calculate costs
  Fuel_cost = (group_data["distance between DC and center of group [km]*"].iat[0] * 2 + group_data["average distance between stores [km]"].iat[0] * (Store_count-1)) * truck_costs[v_capacity]["Fuel_Costs"]
  #if the trip takes less than an hour we only need 1 driver, if its more we pay for 2 drivers
  if trip_duration <= 10:
    Driver_cost = trip_duration * truck_costs[v_capacity]["Driver_Costs"]/drivers
  else: Driver_cost = trip_duration * truck_costs[v_capacity]["Driver_Costs"]

            # Define the data for the new row, "this is the column name" and to the right of those are the values for the row/column
  new_row_data = {"day": day, "start_time": start_time, "loading_time": loading_time, "v_capacity": v_capacity, "Pallets_in_vehicle": Pallets_in_vehicle, "good_type": type, "group": group_name, "Store_count": Store_count,
        "recieving_stores": str(recieving_stores), "relative_pallets": str(relative_pallets), "trip_duration": trip_duration, "return_time": return_time, "last_delivery_time": last_delivery_time, "Dry_demand": Dry_demand,
        "Fresh_demand": Fresh_demand, "reason_for_next": reason_for_next, "Fuel_cost": Fuel_cost, "Driver_cost": Driver_cost,
    }


  # Create a new DataFrame row with the specified index and data
  new_row = pd.DataFrame(data=new_row_data, index=[delivery_count])

  # Concatenate the new row with the existing 'deliveries' DataFrame
  deliveries = pd.concat([deliveries, new_row])

demand_satisfied = deliveries["Pallets_in_vehicle"].sum()

#Calculate how much fresh goods need to be assigned
Fresh_demand = group_data['pallets'][group_data['type'] == "FRESH GOODS"].sum()
#Calculate how much dry goods need to be assigned
Dry_demand = group_data['pallets'][group_data['type'] == "DRY GOODS"].sum()
```

Step-by-Step Demonstration

Plan trips with fresh goods

```
#while loop to add goods to a truck, repeats until we can't add more to it (various reasons)
while Pallets_in_vehicle < v_capacity: #is there still space in the truck?
  #need to check if adding pallets from the next row will change the size limit and then be too much
  if group_data['Vehicle size limit'].iat[0] < Pallets_in_vehicle:
    reason_for_next = "can't add more pallets due to size limit change"
    Store_count -= 1 #we thought we could add another store after the last loop, but nvm, no space
    break #finish adding to truck and go to calculation of distance and costs of delivery

  #Check if the current order is partially assigned, then calculate how long it has been waiting
  if group_data['order_assigned'].iat[0] < no: #less than no is any date and time earlier than 2150 (we set that vanlue for no), meaning it was partly planned at that time
    #check if any partial orders are being completed too late
    check_order(deliveries, group_data, 0, start_time, "FRESH")
    check_order(deliveries, group_data, 0, start_time, "FRESH and DRY")

  v_capacity = min(group_data['Vehicle size limit'].iat[0], v_capacity)  #update delivery capacity limit with the current store, capacity can only go down

  #check if the store has ordered more than the truck can still hold
  if group_data['pallets'].iat[0] > v_capacity - Pallets_in_vehicle:
    #take the order volume the truck can still hold
    recieving_stores.append(group_data['store'].iat[0]) #add the current store number to the list (row 0 is first row)
    relative_pallets.append(v_capacity - Pallets_in_vehicle) #note the number of pallets for this store

    group_data['pallets'].iat[0] -= v_capacity - Pallets_in_vehicle #remove from order row, some goods still remain
    Pallets_in_vehicle += v_capacity - Pallets_in_vehicle #add to the vehicle (fill)
    group_data['order_assigned'].iat[0] = start_time #we took part of an order, note down the starting time

  else:#if the truck can hold the whole order
    Pallets_in_vehicle += group_data['pallets'].iat[0] #add to the truck

    recieving_stores.append(group_data['store'].iat[0]) #add the current store number to the list (row 0 is first row)
    relative_pallets.append(group_data['pallets'].iat[0]) #note the number of pallets for this store

    group_data['pallets'].iat[0] = 0 #order of first row is satisfied
    group_data['order_assigned'].iat[0] = yes #the whole order is planned


    if mixing: #if we allow mixing of dry and fresh we can add the dry from the same store to this truck ···

    #sort the group by Fresh first, then partial/unassigned orders and then ascending numbers of pallets, we use the first row of the table
    group_data.sort_values(by=['type', 'order_assigned', 'pallets'], ascending=[False, True, True], inplace=True)
    group_data.reset_index(drop=True, inplace=True) #since the order of rows has changed, this sets the row numbers back to 0, 1, 2, 3...(easier for code to know that row 0 is first row)

  #Calculate how much fresh goods still need to be assigned
  Fresh_demand = group_data['pallets'][group_data['type'] == "FRESH GOODS"].sum()

  #calculate how long it would take to fill the truck
  loading_time = Average_loading_time + Pallets_in_vehicle * added_loading_time_per_pallet
  #calculate how long the entire trip will take
  trip_duration = loading_time + Pallets_in_vehicle * added_unloading_time_per_pallet + group_data["distance between DC and center of group [km]**"].iat[0] *2 / High_average_speed + group_data["average distance between stores [km]"].iat[0] * (Store_count-1) / Low_average_speed
  #calculate what time the driver completes the final delivery of their trip
  last_delivery_time = start_time + pd.to_timedelta(trip_duration - group_data["distance between DC and center of group [km]**"].iat[0] / High_average_speed, unit='H')
  return_time = start_time + pd.to_timedelta(trip_duration, unit='H') #and the time they arrive back at Sonae

  if Fresh_demand == 0:#if there is no more fresh demand then there is nothing left to add to the truck
    reason_for_next = "demand satisfied" #define why we stop filling
    break #stop filling this truck and leave the current loop to calculate trip costs

  #check if there is still space in the truck and if adding more would make the trip too long
  if Pallets_in_vehicle < v_capacity and trip_duration + group_data["average distance between stores [km]"].iat[0] / Low_average_speed + min((v_capacity - Pallets_in_vehicle),group_data['pallets'].iat[0]) * (added_unloading_time_per_pallet + added_loading_time_per_pallet) <= Maximium_trip_time:
    Store_count += 1 #We should be able to deliver to another store, so add one to the value

  #check if adding another store would make the trip too long
  if trip_duration + group_data["average distance between stores [km]"].iat[0] / Low_average_speed + min((v_capacity - Pallets_in_vehicle),group_data['pallets'].iat[0])  * (added_unloading_time_per_pallet + added_loading_time_per_pallet) > Maximium_trip_time:
    reason_for_next = "long trip" #define why we stopp adding to this truck
    break #stop filling this truck and leave the current loop to calculate trip costs

  #check if adding another store would make the last delivery too late (deadline)
  if last_delivery_time + pd.to_timedelta(group_data["average distance between stores [km]"].iat[0] / Low_average_speed + min((v_capacity - Pallets_in_vehicle),group_data['pallets'].iat[0])  * (added_unloading_time_per_pallet + added_loading_time_per_pallet), unit='H') > day + pd.to_timedelta(Fresh_limit_time, unit='H'):
    reason_for_next = "too late for fresh trip" #define why we stop adding to this truck
    break #stop filling this truck and leave the current loop to calculate trip costs

  reason_for_next = "full vehicle" #if all other checks have passed, the only remaining reason we stop filling a truck is because it's full
```

Step-by-Step Demonstration

And add dry if allowed

```
if mixing: #if we allow mixing of dry and fresh we can add the dry from the same store to this truck
  #check in the current group if there is an order for dry goods from the same store
  dry_order_same_store = group_data[(group_data['store'] == group_data['store'].iat[0]) & (group_data['type'] == "DRY GOODS") & (group_data['pallets'] > 0)]

  if len(dry_order_same_store) > 0: #if there are orders from the same store
    row = dry_order_same_store.index #set a row number that has that store in the group table

    #calculate how long it would currently take to fill the truck
    loading_time = Average_loading_time + Pallets_in_vehicle * added_loading_time_per_pallet
    #calculate how long the entire trip will currently take
    trip_duration = loading_time + Pallets_in_vehicle * added_unloading_time_per_pallet + group_data["distance between DC and center of group [km]*"].iat[0] *2 / High_average_speed + group_data["average distance between stores [km]"].iat[0] * (Store_count-1) / Low_average_speed
    #calculate what time the driver completes the final delivery of their trip
    last_delivery_time = start_time + pd.to_timedelta(trip_duration - group_data["distance between DC and center of group [km]*"].iat[0] / High_average_speed, unit='H')
    return_time = start_time + pd.to_timedelta(trip_duration, unit='H') #and the time they arrive back at Sonae

    #check if adding dry goods would make the last delivery too late (deadline)
    if v_capacity > Pallets_in_vehicle and last_delivery_time + pd.to_timedelta(min((v_capacity - Pallets_in_vehicle), group_data['pallets'].iat[row[0]] )  * (added_unloading_time_per_pallet + added_loading_time_per_pallet), unit='H') < day + pd.to_timedelta(Fresh_limit_time, unit='H'):
      type ="FRESH and DRY" #we can mix and add from that order too

      #Check if the current order is partially assigned, then calculate how long it has been waiting
      if group_data['order_assigned'].iat[row[0]] < no: #less than no is any date and time earlier than 2150 (we set that vanlue for no), meaning it was partly planned at that time
        #check if any partial orders are being completed too late
        check_order(deliveries, group_data, row[0], start_time, "DRY")
        check_order(deliveries, group_data, row[0], start_time, "FRESH and DRY")

      #check if the truck can only hold part of the order
      if group_data['pallets'].iat[row[0]] > v_capacity - Pallets_in_vehicle:
        relative_pallets.append(f"dry:{v_capacity - Pallets_in_vehicle}") #note the number of dry pallets for this store
        group_data['pallets'].iat[row[0]] -= v_capacity - Pallets_in_vehicle #remove from order row, some goods still remain
        Pallets_in_vehicle += v_capacity - Pallets_in_vehicle #add to the vehicle (fill)
        group_data['order_assigned'].iat[row[0]] = start_time #we took part of an order, note down the starting time

      else:#if the truck can hold the whole order
        Pallets_in_vehicle += group_data['pallets'].iat[row[0]] #add to the truck
        relative_pallets.append(f"dry:{group_data['pallets'].iat[row[0]]}") #note the number of dry pallets for this store
        group_data['pallets'].iat[row[0]] = 0 #order of the row is satisfied
        group_data['order_assigned'].iat[row[0]] = yes #entire order has been satisfied
```

5. Solution Results & Analysis

MC Sonae

Deciphering Metrics: Analyzing and Explaining

Number of trips	Number of rows in the week plan, this is the total number of deliveries
fleet size 13	Total number of 13_trucks, equals the maximum number of active trucks at any time
fleet size 24	Total number of 24_trucks, equals the maximum number of active trucks at any time
fleet size 33	Total number of 33_trucks, equals the maximum number of active trucks at any time
total fuel costs (€)	Sum of the fuel price of each trip planned, which is distance travelled multiplied by price
total driver costs (€)	Sum of the driver price of each trip planned, which is the wage(s) multiplied by trip duration
total vehicle costs (€)	Each of the fleet sizes multiplied by their respective vehicle prices and the number of days
total costs (€)	Sum of all total costs
last_fresh_delivery (time)	The time taken for the final truck of the week with fresh goods to complete final last delivery*
last_dry_delivery (time)	The time taken for the final truck of the week with dry goods to complete final last delivery*

***Written as days HH:mm, where the deadline for fresh is 1 day 08:00 and dry is 2 days 00:00**

Function for making the results table:

```python
#a function to make a table with all the calculated metrics for any plan
def calculateResults(planResults, plan, planName):
    #number of rows is the number of trips planned
    planResults.at[planName, "trips"] = len(plan)
    #maximum value of the active_xxtrucks is the number of trucks we need in our fleet (assuming we won't experiment with fewer with increased costs)
    planResults.at[planName, "fleet_size_13"] = plan['active_13trucks'].max()
    planResults.at[planName, "fleet_size_24"] = plan['active_24trucks'].max()
    planResults.at[planName, "fleet_size_33"] = plan['active_33trucks'].max()

    #total fuel cost is the sum of the fuel cost of each trip
    planResults.at[planName, "total_fuel_costs"] = plan['Fuel_cost'].sum()
    #total driver cost is the sum of each trip's driver cost
    planResults.at[planName, "total_driver_costs"] = plan['Driver_cost'].sum()
    #total vehicle cost is the fleet size for each truck size multiplied by the price per vehicle type and the number of days
    planResults.at[planName, "total_vehicle_costs"] = (planResults.at[planName, "fleet_size_13"] * 95 + planResults.at[planName, "fleet_size_24"] * 110 + planResults.at[planName, "fleet_size_33"] * 110)*len(delivery_days)
    #total costs is the sum of all costs for that plan
    planResults.at[planName, "total_costs"] = planResults.at[planName, "total_vehicle_costs"] + planResults.at[planName, "total_driver_costs"] + planResults.at[planName, "total_fuel_costs"]

    #The highest value of all the "last_delivery_time" values will tell us what the latest time of delivery was and if all deadlines were met
    planResults.at[planName, "last_fresh_delivery"] = max(plan[plan["good_type"] != "DRY"]['last_delivery_time'] - plan[plan["good_type"] != "DRY"]['day'])
    planResults.at[planName, "last_dry_delivery"] = max(plan[plan["good_type"] != "FRESH"]['last_delivery_time'] - plan[plan["good_type"] != "FRESH"]['day'])
    #last delivery must be lower than 1 day and 8 hours for fresh and and 2 days for dry

    return planResults
```

```python
#measuring performance of each plan, each column is a metric
planResults = pd.DataFrame() #prepare the results table

plan_count = 0 #value to select the specific table from the plans list
for planName in planNames: #check each plan
    planResults = calculateResults(planResults, plans[plan_count], planName) #run the metrics function to make a row of results for any plan
    plan_count += 1
planResults.T #T is for transpose, that flips the table so that the plan names are the columns
```

Baseline Scenario Analysis: Single Driver, No Mix, Group Order Check

Parameters

- 1 driver
- No mixing dry and fresh
- Check groups in ascending number order

Results are for the entire week of orders:

	BASELINE CASE
Number of trips	2049
fleet size 13	88
fleet size 24	34
fleet size 33	165
total fuel costs (€)	219031.7
total driver costs (€)	106665.35
total vehicle costs (€)	181500
total costs (€)	507197.05
last_fresh_delivery (time)	1 day, 03:50:59
last_dry_delivery (time)	1 day, 06:09:39

Comparing Baseline to Initial Alternative: Single Driver, No Mixing

The **only difference** in setup is the **order** in which the **groups are treated**.

In the **baseline** it plans for **groups** in **numerical order**, for the **second option** they are **planned** in **order distance** to the **DC**, nearest first.

Results are similar:

- having **fewer trips** saves on fuel costs for the alternative
- due to a **larger fleet size** the alternative is **more expensive** overall

→ We keep the **baseline** for the next to minimize cost.

Green = improvement	Baseline Case	1 driver, no mixing
Number of trips	2049	2041
fleet size 13	88	97
fleet size 24	34	31
fleet size 33	165	165
total fuel costs (€)	219031.70	218727.05
total driver costs (€)	106665.35	106728.42
total vehicle costs (€)	181500.00	184650.00
total costs (€)	507197.05	510105.47
last_fresh_delivery (time)	1 day, 03:50:59	1 day 04:48:39
last_dry_delivery (time)	1 day, 06:09:39	1 day 05:57:39

Baseline vs. 2nd Alternative: Group Order Sequence Change

The **difference** in setup is the **order** in which the **groups** are **treated**.

In the **baseline** it **plans** for **groups** in **numerical order**, for the **second option** they are **planned** in **order distance** to the **DC**, nearest first.

Also when **planning fresh orders** it **checks** if **dry orders** can be **added** from the **same store**.

Results are better in the alternative:

- having fewer trips and a smaller fleet size saves on fuel costs
- mixing fresh and dry leads to later final deliveries but the deadlines are still adhered to

Green = improvement	Baseline Case	1 driver, no mixing
Number of trips	2049	1897
fleet size 13	88	96
fleet size 24	34	26
fleet size 33	165	152
total fuel costs (€)	219031.70	183397.95
total driver costs (€)	106665.35	94441.34
total vehicle costs (€)	181500.00	172200.00
total costs (€)	507197.05	450039.29
last_fresh_delivery (time)	1 day, 03:50:59	1 day 04:36:59
last_dry_delivery (time)	1 day, 06:09:39	1 day 06:20:19

Comparing 2nd to 3rd Alternative: Driver Flexibility & Order Adaptation

The **difference** in setup is the **number of drivers** permitted.

If a trip takes longer than 10 hours a second driver can be added.

Also the **first option checks** if **dry orders** can be **added** from the **same store** when **planning fresh orders.**

Results are better in the **2nd** alternative:

- **costs** are **lower** across the board, especially driver costs
- **last delivery times** are also **much later** in the **second alternative** due to allowing for 20 hour trips

Green = improvement	1 driver, mixing	2 drivers, no mixing
Number of trips	1897	1865
fleet size 13	96	97
fleet size 24	26	32
fleet size 33	152	147
total fuel costs (€)	183397.95	186609.95
total driver costs (€)	94441.34	135129.07
total vehicle costs (€)	172200.00	173430.00
total costs (€)	450039.29	495169.02
last_fresh_delivery (time)	1 day 04:36:59	1 days 07:59:16
last_dry_delivery (time)	1 day 06:20:19	1 days 10:36:15

Contrasting 2nd to 4th Alternative: Extended Trip Duration & Driver Flexibility

The **difference** in setup is the **number of drivers** permitted.

If a trip takes longer than 10 hours a second driver can be added and the limit becomes 20 hours.

Results are slightly better in the **2nd** alternative

- **driver costs** are **lower**, significantly **lowering total costs**
- for two drivers **fewer trips** are needed, **lowering fuel vehicle costs**
- **last delivery times** are also **much later** in the 4th alternative due to allowing for 20 hour trips

Green = improvement	1 driver, mixing	2 drivers, mixing
Number of trips	1897	1831
fleet size 13	96	96
fleet size 24	26	28
fleet size 33	152	141
total fuel costs (€)	183397.95	172305.35
total driver costs (€)	94441.34	112569.66
total vehicle costs (€)	172200.00	166260.00
total costs (€)	450039.29	451135.01
last_fresh_delivery (time)	1 day 04:36:59	1 days 06:02:19.
last_dry_delivery (time)	1 day 06:20:19	1 days 13:11:55.

Alternative Comparison: Exploring Delivery Planning Scenarios

	Baseline Case	1 driver, no mixing	1 driver, mixing	2 drivers, no mixing	2 driver, mixing
Number of trips	2049	2041	1897	1865	1831
fleet size 13	88	97	96	97	96
fleet size 24	34	31	26	32	28
fleet size 33	165	165	152	147	141
total fuel costs (€)	219031.70	218727.05	183397.95	186609.95	172305.35
total driver costs (€)	106665.35	106728.42	94441.34	135129.07	112569.66
total vehicle costs (€)	181500.00	184650.00	172200.00	173430.00	166260.00
total costs (€)	507197.05	510105.4667	**450039.29**	495169.02	451135.01
last_fresh_delivery (time)	1.160416667	1.200462963	1.192361111	1.332824074	1.25162037
last_dry_delivery (time)	1.256712963	1.24837963	1.26412037	1.441851852	1.549953704

Result-Driven Conclusions: Insights Drawn from Analysis

Allowing trips after 19:00 to **include dry** and **fresh** goods has the **most impact** on **fleet size** and **overall costs**, although slightly slowing down the arrival of the last fresh orders

Allowing 2 drivers on a delivery results in **fewer deliveries** being **needed** and a slightly smaller fleet size but due to increased driver costs the **total cost** is **increased**.

However, **allowing** for **2 drivers** might **perform better** with some parameter tuning, or even in combination with cross-docking

Next we can look at ordering the groups

Result-Driven Conclusions: Ordering Groups

- It appears to be **impactful which groups** are **addressed first**.
- The **baseline case** puts **groups** in **numerical order** and was **slightly cheaper**.

→ Using the best performing alternative we will do a quick comparison of:

- Counting up in group number
- Ordering by furthest first
- Random orders each day

Alternative Comparison: Different Group Orders

	no Mix, 1 driver, Close 1st	no Mix, 1 driver, Count up	no Mix, 1 driver, Far 1st	no Mix, 1 driver, Random
Number of trips	**1897**	1952	2046	1924
fleet size 13	99	87	81	108
fleet size 24	26	29	28	26
fleet size 33	153	162	182	164
total fuel costs (€)	**183393.9**	196144.1	214751.6	189078.3
total driver costs (€)	**94441.34**	98447.8	102914.9	95941.63
total vehicle costs (€)	**174570**	175650	184770	186960
total costs (€)	**452405.2**	470241.9	502436.5	471979.9
last_fresh_delivery (time)	1 days 04:36:59	1 days 04:49:59	**1 days 02:58:59**	1 days 04:22:19
last_dry_delivery (time)	1 days 06:20:19	1 days 05:37:59	1 days 08:24:27	1 days 07:04:47

The **original optimal** is **still** the **best**, **although delivering** to the **far locations first** makes the **final fresh delivery** arrive **earlier**.

Should we adjust Fleet size?

Right now fleet size for each truck type is the maximum number in use at any point in the week..

So we are paying daily price for trucks that are not always being used that day.

We could reduce the fleet size and do the following when the active trucks exceeds the fleet:

“Occasional trucks may be considered beyond the regular fleet for specific days. For those vehicles, all costs must be increased by +30%”

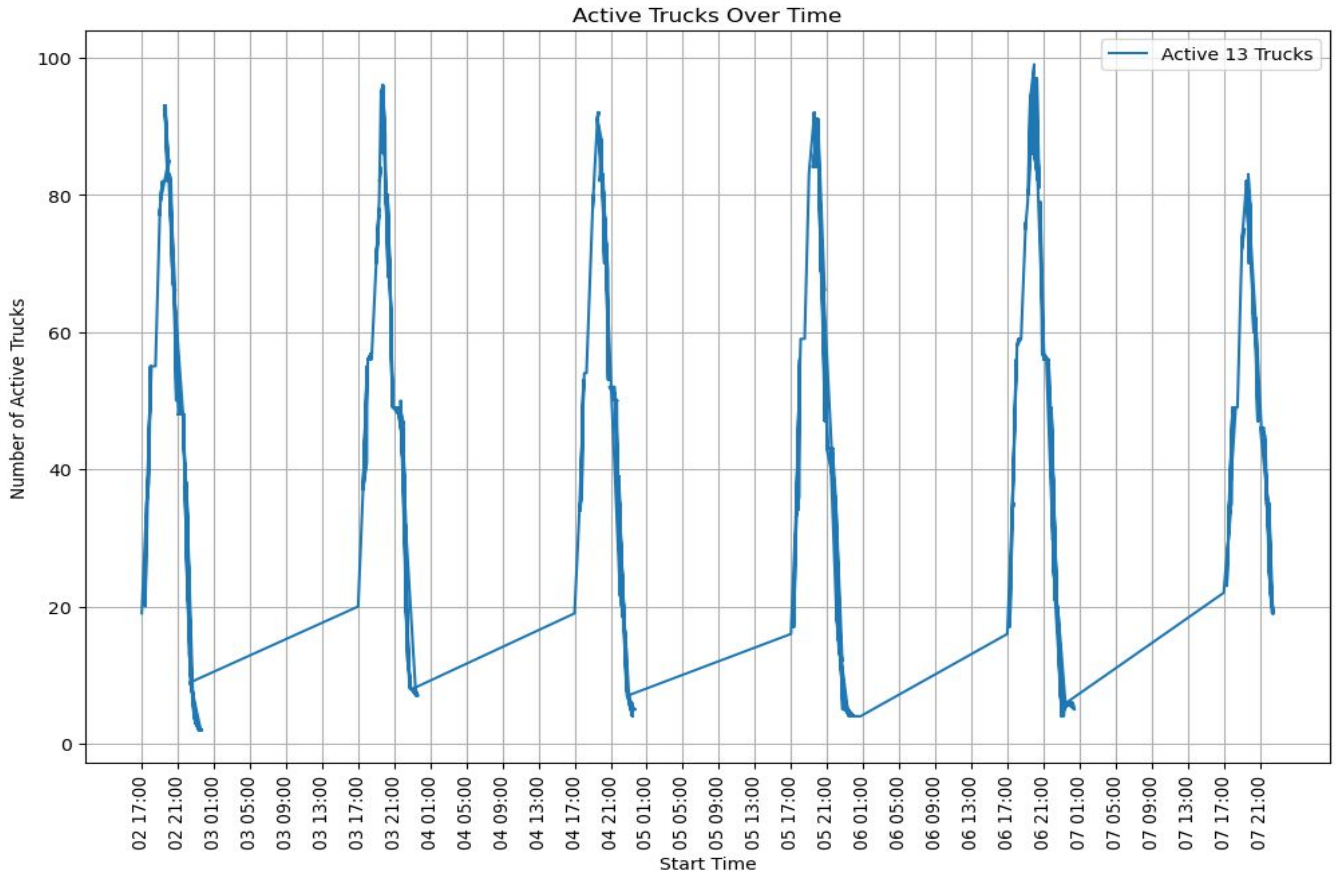
Function to calculate how many of each truck type is active when a delivery starts:

```python
#A function to calculate the active trucks at the start of any delivery
def getActiveTrucks(plan, v_capacity, row):
    # calculate the number of active trucks (currently filling or active)
    start_time = row["start_time"]

    # Count of trucks that started but haven't returned by the specified time
    started_trucks = plan[plan['start_time'] <= start_time]
    # counts how many of the planned trips have trucks that started already
    started_truck_count = len(started_trucks[started_trucks["v_capacity"] == v_capacity])

    # Count of trucks that have returned by the specified time
    returned_trucks = plan[plan['return_time'] < start_time]
    returned_truck_count = len(returned_trucks[returned_trucks["v_capacity"] == v_capacity])

    # Calculate the total number of active trucks
    active_trucks = started_truck_count - returned_truck_count
    return active_trucks

#A function to add the active trucks columns to any plan table
def getActiveTruckColumns(plan):
    #use the function to calculate the active trucks of each capacity at the start of every delivery
    plan["active_13trucks"] = plan.apply(lambda row: getActiveTrucks(plan, 13, row), axis=1)
    plan["active_24trucks"] = plan.apply(lambda row: getActiveTrucks(plan, 24, row), axis=1)
    plan["active_33trucks"] = plan.apply(lambda row: getActiveTrucks(plan, 33, row), axis=1)
    return plan

# run the function to add the activetruck columns for each of the plans we have created
for plan in plans:
    plan = getActiveTruckColumns(plan)
```

This **creates** the **columns** in the plan table with the **number** of **active trucks**.

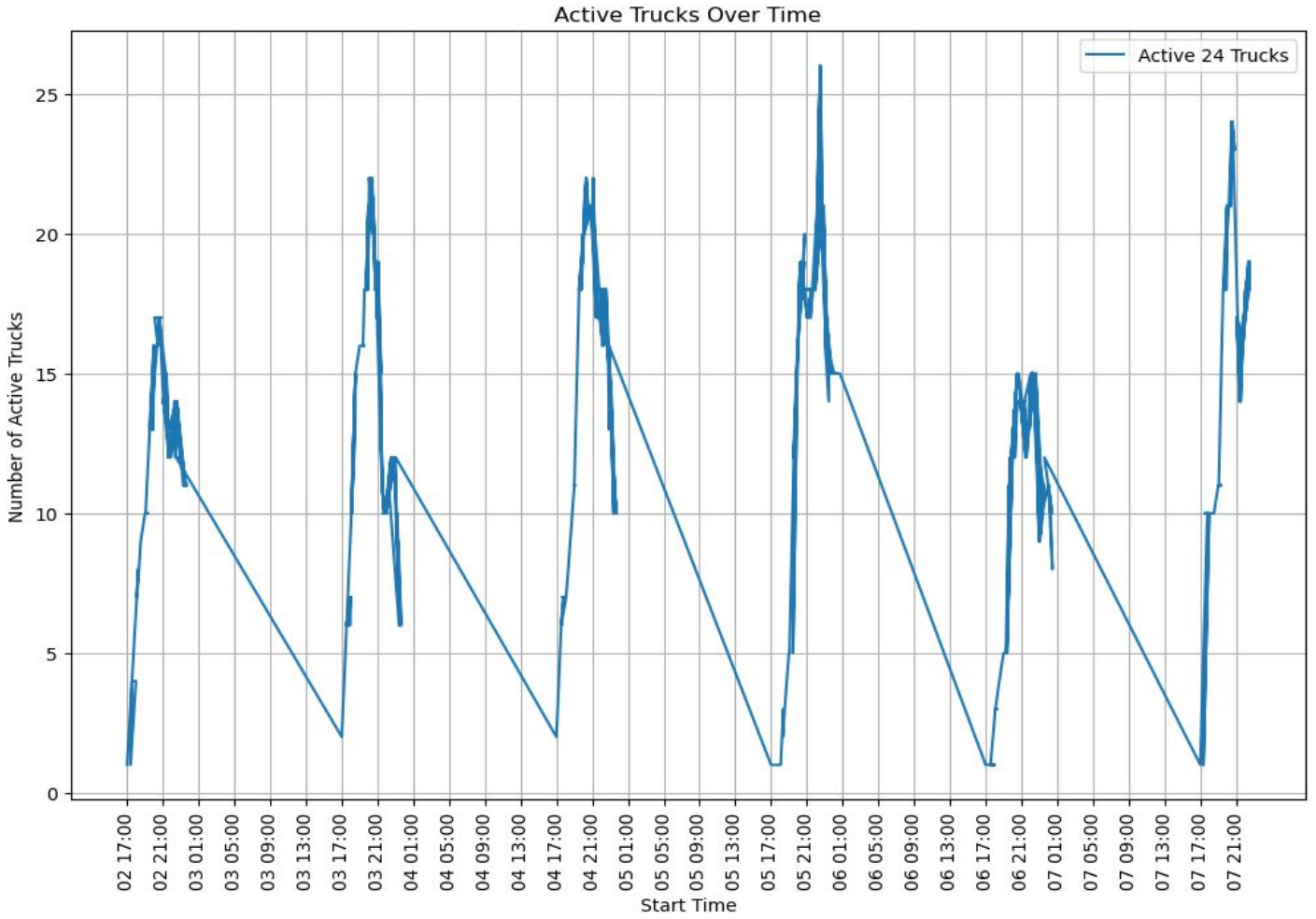
Each column is plotted over time in the following slides:

Consistent 13-Capacity Truck Fleet: Cost Optimization with Variable Usage



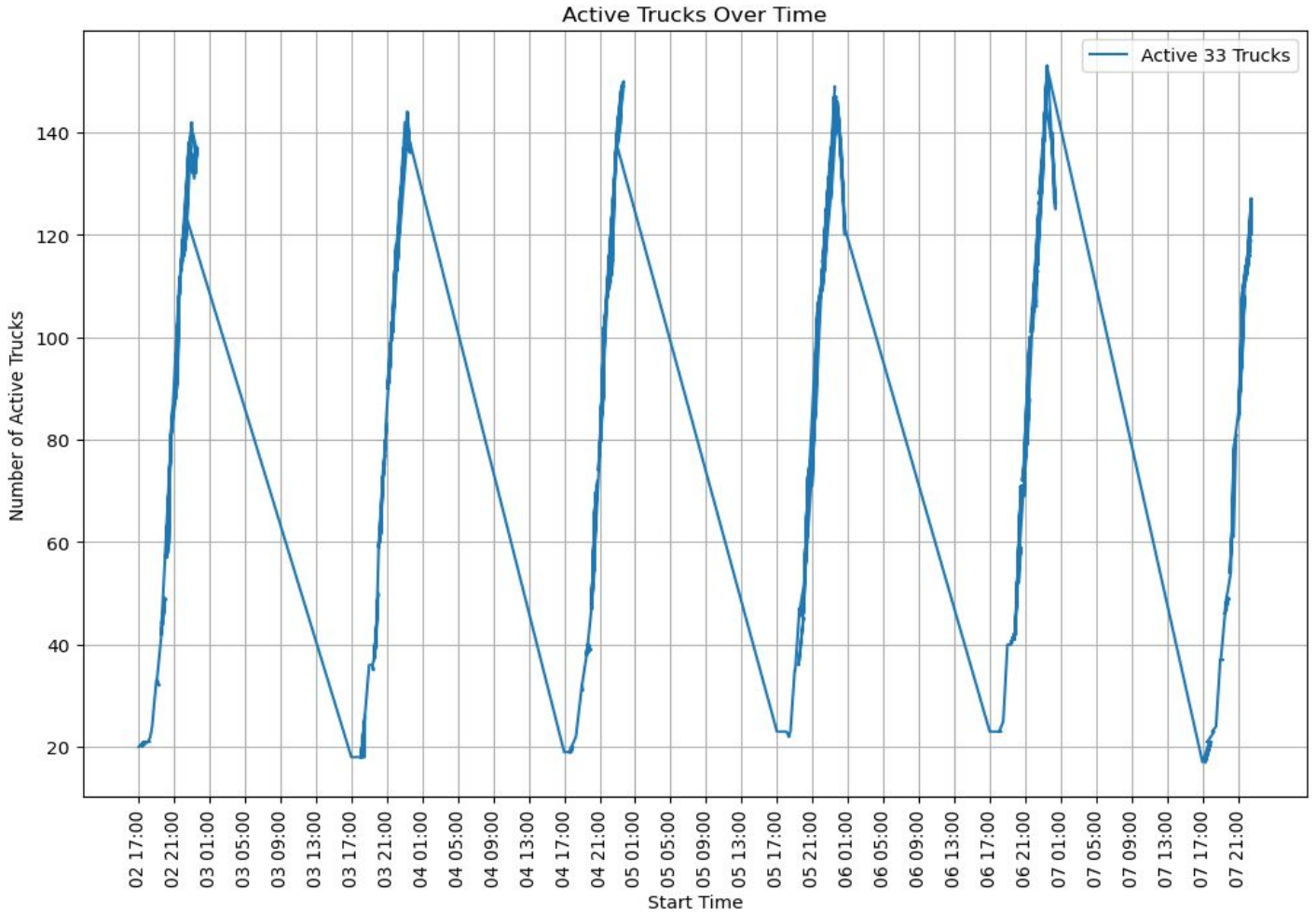
Each day the **fleet size** is quite **consistent** but **slight variation** might mean you can **save money** on **days** where **fewer trucks** are **needed.**

Flexible 24-Capacity Truck Fleet: Cost-Efficient Variability in Usage



Each day the **fleet size varies more than** for **other trucks**, this **variation means** you can **save money** on **days** where **fewer trucks** are **needed**

Stable 33-Capacity Truck Fleet: Cost Analysis in Minimal Variance



Each day the **fleet size varies** very **little**, this likely **means** that **reducing** the **fleet size** will **increase costs**

Cost Efficiency Through Fleet Size Reduction

If you **reduce fleet size**…

…you **save** on **vehicle costs**

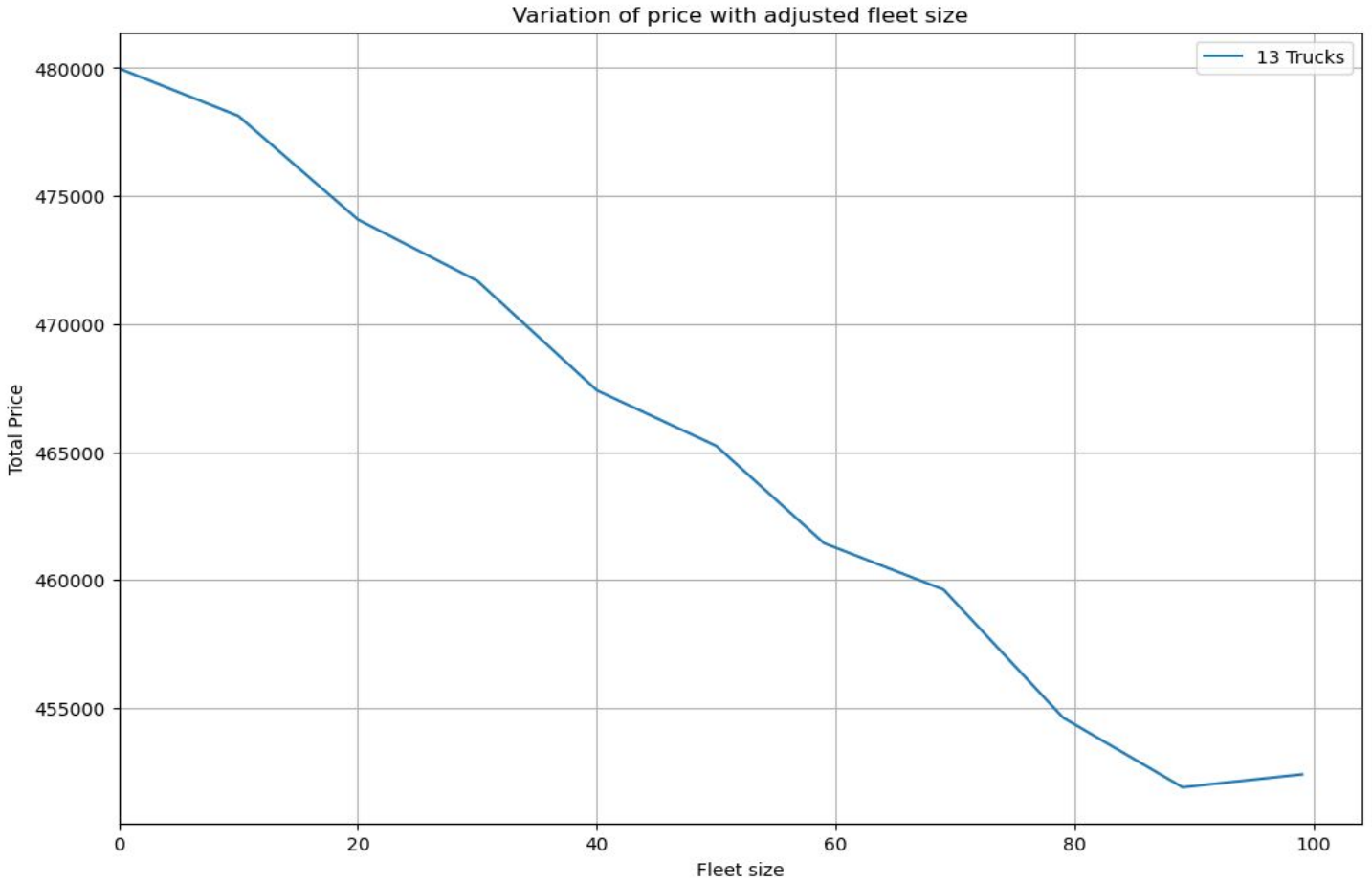
BUT

… you **pay 30% more** any time that **more trucks** are **needed**

The costs resulting from this code are plotted in the following slides:

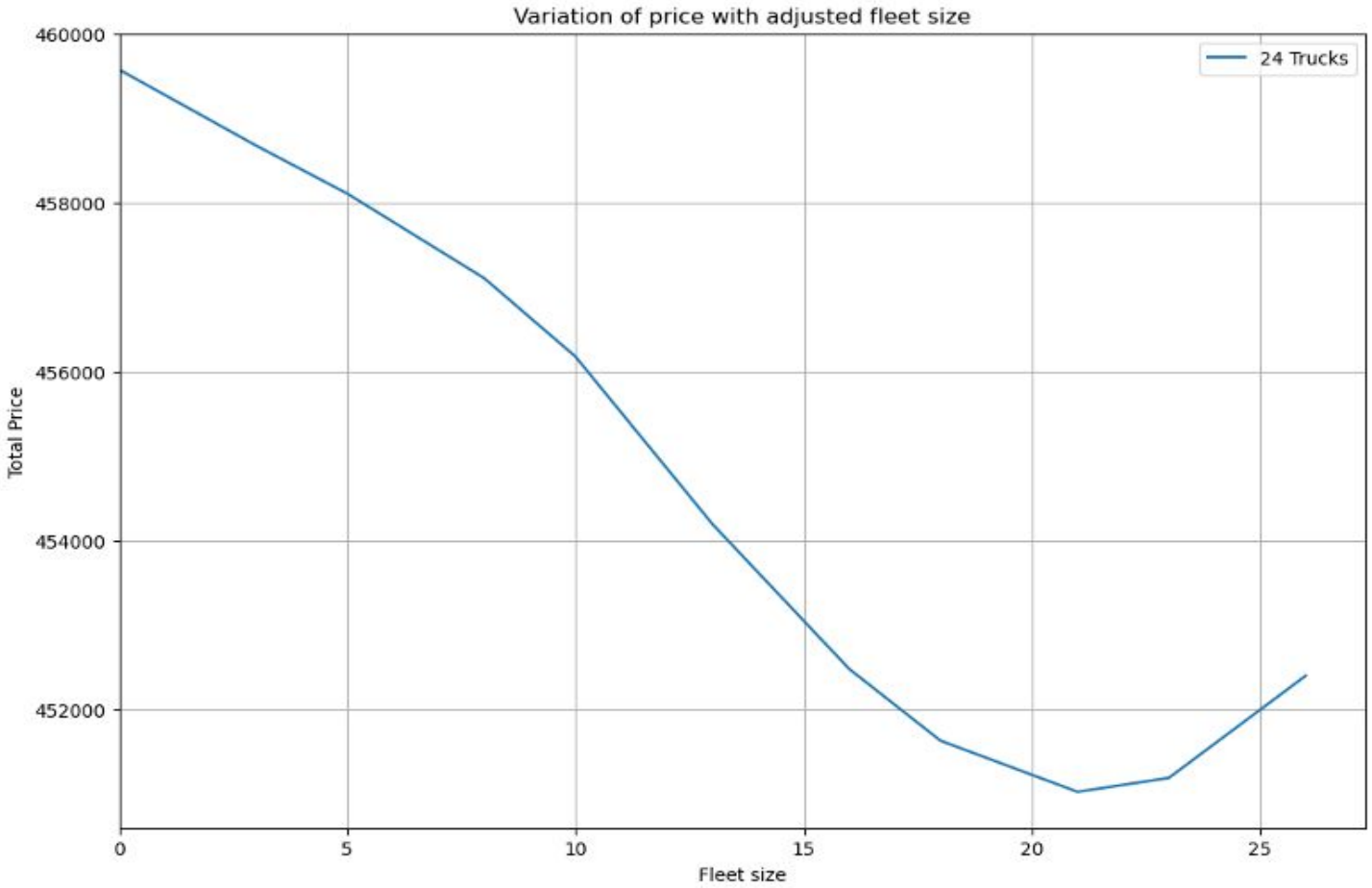
```python
def getFleetAdjustedCosts(plan, f1, f2, f3):
    #make each fleet size a fraction of the maximum active trucks value
    fleet_size_13 = round(plan['active_13trucks'].max() * f1)
    fleet_size_24 = round(plan['active_24trucks'].max() * f2)
    fleet_size_33 = round(plan['active_33trucks'].max() * f3)


    # Fleet cost without 30% increase
    base_vehicle_cost = (fleet_size_13 * 95 + fleet_size_24 * 110 + fleet_size_33 * 110) * len(delivery_days)

    #make a loop, it repeats everything separately for each day
    for day in delivery_days:
        day = pd.to_datetime(day + " 2023")
        # Filter the DataFrame to select rows where 'day' matches the desired delivery day
        day_plan = plan[plan['day'] == day].copy()

        # Calculate the additional cost for exceeding the fleet size with 30% increase
        additional_fleet_cost = 1.3 * (max(0, day_plan['active_13trucks'].max() - fleet_size_13)) * 95
        additional_fleet_cost += 1.3 * (max(0, day_plan['active_24trucks'].max() - fleet_size_24)) * 110
        additional_fleet_cost += 1.3 * (max(0, day_plan['active_33trucks'].max() - fleet_size_33)) * 110

    #total fuel cost is the sum of the fuel cost of each trip
    base_fuel_costs = plan['Fuel_cost'].sum()
    excess13 = plan[plan['active_13trucks'] > fleet_size_13]
    additional_fuel_cost = excess13[excess13['v_capacity'] == 13]['Fuel_cost'].sum() * 0.3

    excess24 = plan[plan['active_24trucks'] > fleet_size_24]
    additional_fuel_cost += excess24[excess24['v_capacity'] == 24]['Fuel_cost'].sum() * 0.3

    excess33 = plan[plan['active_33trucks'] > fleet_size_33]
    additional_fuel_cost += excess33[excess33['v_capacity'] == 33]['Fuel_cost'].sum() * 0.3


    #total driver cost is the sum of the driver cost of each trip
    base_driver_costs = plan['Driver_cost'].sum()
    excess13 = plan[plan['active_13trucks'] > fleet_size_13]
    additional_driver_cost = excess13[excess13['v_capacity'] == 13]['Driver_cost'].sum() * 0.3

    excess24 = plan[plan['active_24trucks'] > fleet_size_24]
    additional_driver_cost += excess24[excess24['v_capacity'] == 24]['Driver_cost'].sum() * 0.3

    excess33 = plan[plan['active_33trucks'] > fleet_size_33]
    additional_driver_cost += excess33[excess33['v_capacity'] == 33]['Driver_cost'].sum() * 0.3

    #total costs is the sum of all costs for that plan
    total_costs = base_vehicle_cost + base_fuel_costs + base_driver_costs + additional_fleet_cost + additional_fuel_cost + additional_driver_cost
    #print(additional_fleet_cost, additional_fuel_cost, additional_driver_cost)
    return [fleet_size_13, fleet_size_24, fleet_size_33, total_costs]
```

Cost Efficiency Through Fleet Size Reduction – 13 Trucks



Other fleet sizes kept at maximum

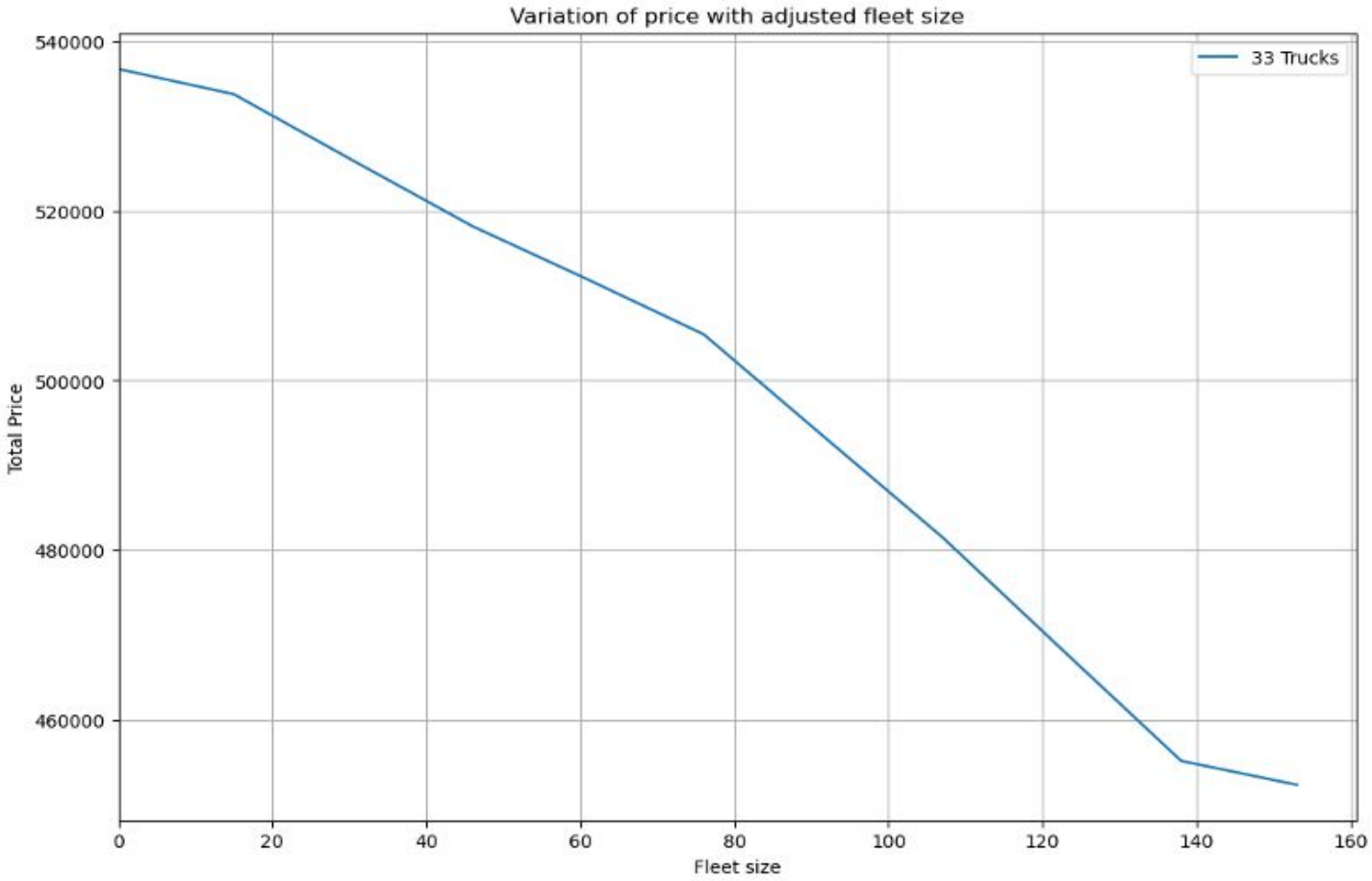
Cost Efficiency Through Fleet Size Reduction – 24 Trucks



ⓘ

Other fleet sizes kept at maximum

Cost Efficiency Through Fleet Size Reduction – 33 Trucks



Other fleet sizes kept at maximum

Outcome Overview: Analyzing Results

It's **heavily dependent** on **how frequently** the **truck type** is **used**, **33** size trucks are the **most regularly used** so there is more clarity for their optimal fleet size.

Optimal Planning Guidance:

Assuming that our planning is optimal and each week is similar in orders we **advise** to **utilize** a **slightly smaller fleet size** for **13** and **24 size trucks** for **lower costs**, as shown in the table below:

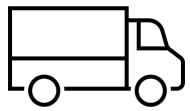
	Fleet 13	Fleet 24	Fleet 33	Cost (euro)
Max	**99**	**26**	**153**	**452405.24**
Optimal	**89**	**21**	**153**	**450527.33**

6. Solution Evaluation

MC Sonae

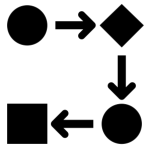
Comprehensive Solutions and Recommendations

Network



- ✔ There are no savings on journeys from the cross-docking station, so we recommend working without it.
- ✔ Utilize a slightly smaller fleet size for 13 and 24 size trucks for lower costs.

Process



- ✔ Mix dry with fresh goods after 19:00 if a fresh delivery is planned to the same store.

Sustainability

- ✔ Transition to electric vehicles could be a strategic move and brings tangible benefits in terms of cost, reputation, and long-term operational resilience.

How to improve?

- When using 2 drivers we should consider if a trip of 11 hours is worth 2 drivers, this option may not be explored enough.

- This is a bottom up approach, there may be better solutions than those generated by our code. However, the produced plans meet the success criteria.

- Costs and savings of cross-docking are just estimates, research into this could be deepened to find the exact impacts.

Appendix

MC Sonae

Sustainability Strategies - Practical Advice

Electric Vehicles (EVs):

An adoption of electric vehicles (EVs) in distribution offers a many advantages, strategically positioning the company for sustainable growth and operational efficiency. Some of the advantages:

1. Environmental Responsibility: By reducing emissions, Sonae's use of EVs demonstrates a commitment to environmental responsibility, aligning with global sustainability goals such as the UN:s 2030 Agenda and ensuring compliance with regulatory standards.

2. Economical Advantages: Electric vehicles contribute to financial efficiency by offering lower operating and maintenance costs. The initial investment is offset by reduced fuel expenses and maintenance requirements, resulting in long-term financial benefits.

3. Government Incentives: Sonae can take advantage of subsidies and incentives provided by the Portuguese government to promote and facilitate the adoption of electric vehicles, further optimizing the economic aspects of the transition.

4. Brand Image Enhancement: Operating a fleet of electric vehicles enhances Sonae's brand image. The company becomes synonymous with eco-friendly practices, appealing to environmentally conscious consumers and fostering a positive reputation in the market.

In summary, Sonae's transition to electric vehicles is a strategic move that not only addresses environmental concerns but also brings about tangible benefits in terms of cost, reputation, and long-term operational resilience.