"Planning Production and Transportation Activities at ABSA Oil Refineries"

EC6055: Prescriptive Analytics Group Project

MSc in Business Analytics 2023-24

Group Number: 17
Eoin Holly (114417398)
Karan Shankar (123114976)
Manish Kamble (123114058)
Rashmi Singh (123111338)
Sanyam Singh Chauhan (123114795).
Shabbir Yusufbhai Motorwala (123105671)

Submitted on: ***12/04/2024.***

Table of Contents

Executive Summary **3**

Problem Statement **4**

Core Assumptions **5**

Core Assumption 1: 5

Core Assumption 2: 5

Core Assumption 3: 7

Project Approach **8**

ABSA Supply Chain 9

Profit Maximization Model **11**

Output Analysis: 14

Minimum Profit Model **15**

Output Analysis: 16

Minimizing Shipping Cost Model **18**

Output Analysis: 27

Conclusion and Recommendations **32**

References **33**

Executive Summary

As operations research analysts at ABSA Oil, our primary responsibility was to organize and optimize the company's purchasing, shipping, and production procedures. In this work, we give a summary of ABSA's planning problem and detail the evaluations that were done as well as the models developed to address these challenges.

This paper aims to provide readers with an in-depth understanding of ABSA's planning challenge.

- To explain the analysis's models and methods.
- To highlight the inadequacies of the models and provide clarification on the assumptions made during the analytical process.
- To offer helpful advice supported by visually striking images.
- To discuss potential extensions of the analysis and identify more data required for next optimization efforts.

The analyses that were conducted focused on three primary models:

- Maximizing Profit Model: Designed to optimize production and procurement choices to maximize profitability while meeting demand constraints.
- Minimized Profit: Designed to optimize production and procurement choices to achieve minimum profitability while meeting demand constraints.
- Minimizing Shipping Cost Model: Shipment scheduling and routing were optimized to save money on transportation.

Each model is developed utilizing a rigorous analytical approach that integrates the necessary assumptions and uses the data that is currently available to provide meaningful insights for ABSA's management.

The suggestions in this paper are intended to assist ABSA in making data-driven decisions that would increase profitability and operational efficiency. These recommendations are based on the outcomes of these models.

The study also looks at potential areas for more examination, stressing the need for additional data to support these advancements as well as the potential for extending the current models.

ABSA will be able to better understand its business operations and apply that understanding to support long-term, sustainable growth with the aid of this research.

Let's analyze the research and recommendations in detail and have a closer look at ABSA's planning problem.

Problem Statement

The optimization of the ABSA Oil Distribution Network presents a number of challenges. The problem statement is as follows: Within the context of its global distribution network, ABSA Oil is confronted with a difficult optimization task. In order to achieve the goal of maximizing overall profit while maintaining a balance between various important factors:

The process of strategic purchasing of crude oil requires us to ascertain the best quantities of crude oil to acquire from a variety of providers (BP, Chevron, and others) at prices that are subject to fluctuation in the market. Optimization of production planning is required for the four European refineries (Greece, Poland, Spain, and the United Kingdom) in order to achieve efficiency in refining. We need to determine the optimal production levels for each of the six petroleum products (three gasolines, jet fuel, diesel fuel, and heating oil), taking into account the constraints imposed by the refinery's capacity and aiming to maximize profit margins through this process. Cost-Effective Transportation: It is essential to ensure that oil products that are produced are transported to regional demands in an efficient manner. We need to find the most cost-effective method of allocating and delivering products while simultaneously reducing the amount of money spent on transportation.

Core Assumptions

Core Assumption 1:

Each type of Crude Oil must be used to produce any refined product.

We are working off the assumption that we must blend all 4 types of crude oil in order to make any equal amount of product. We developed this assumption because it is clearly stated in the problem statement, pictured below.

> Each product is produced by blending 4 types of crude oils. These oils are supplied by:
>
> - BP
> - Chevron
> - Petróleos de Venezuela
> - Statoil

We are also given the different characteristics of the 4 types of crude oil, indicating that each oil is unique and therefore classed as a different "type" of crude oil based on its API gravity and Sulphur content.

Product Name	API gravity	Sulphur Content
Azeri BTC	36.1	0.14%
Poseidon Streams	29.6	1.97%
Laguna	10.9	5.4%
Snøhvit Condensate	60.1	0.019%

Core Assumption 2:

4 Barrels of Crude Oil = 4 Barrels of Refined Product

We haven't been given a ratio explaining what percentage of each crude oil is needed to produce each of our refined products, therefore we are assuming they must be blended in equal measure. If we must blend all 4 types of crude oil in equal measure to make our refined product, then our output will have to be 4 barrels of 1 specific refined product. The reasoning behind this assumption is an article we found when researching crude oil refining which indicates that

scientific processes such as hydrotreating can be used to manipulate, and indeed lower, the Sulphur content of refined products. (Sulfur (content), no date)

This assumption lets us disregard the API gravity and Sulphur contents of each product and gives us an idea of how many barrels we will need for purchasing. Since we need equal amounts of each product, our next step was to look at our monthly supply quotas.

Product Name	Price per Barrel	Monthly supply quota (barrels)
Azeri BTC	$57	645,000
Poseidon Streams	$48	575,000
Laguna	$35	550,000
Snøhvit Condensate	$71	645,000

As seen above, our lowest monthly supply quota is 550,000 barrels of Laguna. This means that to maximize our profits, we need 550,000 barrels of each product, coming to 2,200,000 barrels total. However, we needed to ensure we had the production capacity across all our plants to produce this amount. The actual plant where our refined product was produced did not matter, as the production cost was a flat $19 per barrel. The only relevant information here was that we had to meet our production quota in each refinery.

Oil	Greece	Poland	Spain	UK
Gasoline-87	35,000	22,000	76,000	98,000
Gasoline-89	45,000	38,000	103,000	52,000
Gasoline-92	50,000	60,000	83,000	223,000
Jet fuel	20,000	25,000	47,000	127,000
Diesel fuel	75,000	35,000	125,000	87,000
Heating oil	25,000	205,000	30,000	13,000

Any excess production can be sold with a 7% discount on the sales prices.

	Greece	Poland	Spain	UK
Oil Capacity	400,000	540,000	625,000	735,000

When we added up the total oil capacity of all of our refineries, this gave us a figure of 2,300,000, which meant that we had adequate capacity to maximize our output given our above assumption. The only other relevant piece of information here is that any excess production is sold with a 7% discount, which doesn't have any real impacts on us. It will only affect our overall product. Since we are working off the assumption that all 4 oils must be blended in equal measure, then our shipping costs (I will cover in the next assumption) must be incurred regardless, meaning that the optimal way for us to maximize our profit is to maximize production. That way, even if our assumption about API gravity and sulphur manipulation is off, it

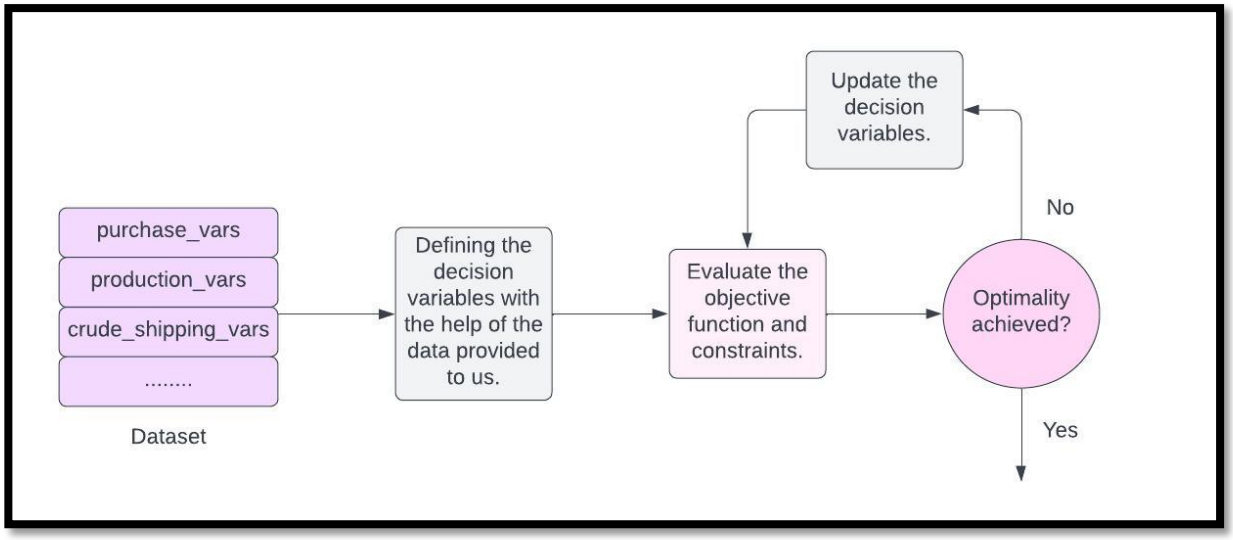
won't matter because we will still have enough raw crude oil at each location to be blended in whatever ratios are needed.

Core Assumption 3:

All Classes of Boats Can Be Considered

This part of the question threw us a little. We began by only considering LR1 and LR2 boats because the AFRA Scale categorizes these boats as "crude oil," however since our assumption requires all 4 types of crude oils to be shipped to all 4 locations this meant that we needed 16 boats. LR1 and LR2 only come to 8 boats total, so we decided to consider all boats and assume the "crude oil" tag was just a weight capacity classification as it is the heaviest type of oil and these boats had the heaviest capacities available. As a result, we considered all boats and all costs associated with these boats, from fuel costs, rates and port charges & commissions to the deadweight tonnage limitations. We then built a model to pick us the cheapest 16 boats after factoring in every variable. At this point, we still weren't sure how best to spread out the excess, as max capacities in Greece & Poland were lower than the base 550,000 figure we were considering. After much deliberation, we discovered that our optimal production would be if we filled our refineries in the opposite direction than we were given UK-Spain-Poland-Greece. This way we were able to hit our maximum production target of 2,200,000 while still at **least** meeting demand across all our refineries. This also opened the cheapest possible GPT boats to Greece, as their deadweight tonnage was less than the number of barrels we were producing there (84,000>75,000). If we maxed out production in Greece, we would have had to take a more expensive boat and then we would have needed less room in boats to the UK so we would not have been optimizing our supply in relation to our shipping costs. By taking the approach we have, we have assured that every factor from purchasing to shipping and production is optimized.

Project Approach



The above flowchart outlines a systematic approach used in optimization problem.

1. Data Acquisition: The process begins by collection of relevant datasets containing raw information about the problem at hand, such as purchase data, production metrics, or shipping details i.e purchase_vars, production_vars, crude_shipping_vars.

2. Identifying Variables: Next, we pinpoint the key factors, known as decision variables, that we can control within the problem's boundaries to achieve the best possible outcome. These variables are typically informed by the data we've gathered.

3. Objective and Constraints: We then evaluate a mathematical expression, called the objective function, which defines what we're trying to maximize or minimize. Alongside this, we consider any constraints that limit our options, like budgetary constraints or production capacity.

4. Optimality Check: We assess whether our current solution is the best one possible, given the defined objective and constraints. If it's not optimal, we move to the next step.

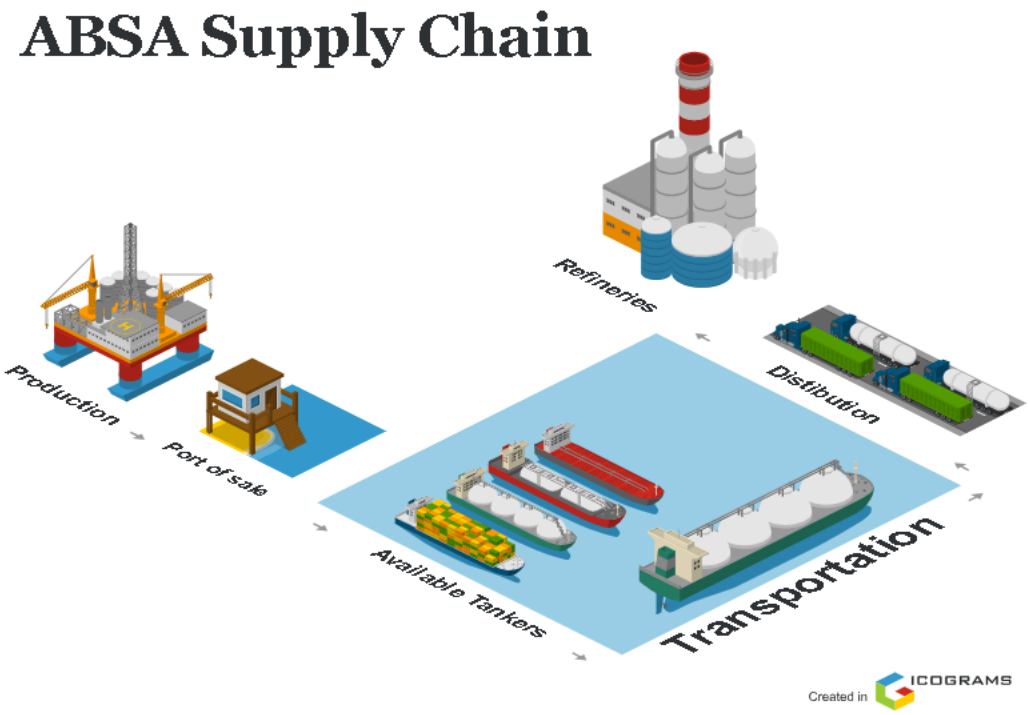
5. Adjusting Variables: Here, we tweak our decision variables based on various optimization techniques. This might involve methods like adjusting production levels, changing shipment routes, or altering resource allocations.

6. Iteration: The process of evaluating, adjusting, and reassessing continues in a loop until we reach an optimal solution.

7. Achieving Optimal Solution: Finally, we stop when we've achieved the best possible outcome according to our objective function, within the constraints we've set.

This methodical approach ensures that businesses make informed decisions to maximize efficiency, minimize costs, and optimize their operations for better overall performance.

ABSA Supply Chain



The chart above depicts a simplified isometric chart of the assumed supply chain management for ABSA refineries. We can understand it in the form of steps from start to end

1. Production - Crude oil is obtained through the process of drilling deep wells. After extraction, the crude oil is transferred to refineries, which are large-scale facilities that separate several products from the crude oil, including gasoline, diesel fuel, and heating oil.

2. Port of Sale: The port of sale is the last destination of the production part, where the oil is discharged, and ownership is officially transferred.Consider it as a refueling station for substantial oil deliveries. The seller transports the oil to the port, at which point the buyer assumes ownership and accountability for it.

3. Transportation: Oil transportation plays a vital role in the oil and gas supply chain, facilitating the movement of crude oil from extraction sites to refineries and delivering refined products to end users. Below is an analysis of the formal elements and how the names of the tankers that are provided connect with them:

   Modes of transportation:
   Sea Shipment- This is the primary method for transporting oil worldwide, using big tankers of different capacities. Some examples of tankers are Gudrun, Pretty World, and Ismini, which had been mentioned.
   Pipelines provide a cost- efficient and uninterrupted transportation method for oil across extensive distances, especially for refineries located in areas without access to the sea or for delivering oil within a certain geographical area.

4. Classification of Tankers: Tankers are classified according to their Deadweight Tonnage (DWT), which refers to the maximum weight of cargo, specifically oil, that they are capable of carrying. Below is an analysis of some typical tanker classifications: Very Large Crude Carriers (VLCC) are the largest tankers, with a Deadweight Tonnage (DWT) that surpasses 200,000 tons. They are well-suited for conveying substantial quantities of unrefined petroleum over extensive distances, such as Gudrun.
   Aframax tankers are classified as medium-sized vessels with a deadweight tonnage (DWT) ranging from 80,000 to 120,000 tons. They provide versatility for different routes and ports, and Ismini may potentially fall into this category.

5. Refineries: Finally the crude oil arrives at the refineries, these are the places where the crude oil are refined. They make fuels and other things from crude oil that can't be used directly. ABSA has such refineries in 4 Countries mentioned Greece, UK, Poland and Spain

Profit Maximization Model

This is a model we did in Python which IBM Decision Optimization CPLEX Modelling from the docplex library. To start with the model, we first import the libraries required for us to run the model.

```
pip install docplex
```

```
Requirement already satisfied: docplex in c:\users\karan\anaconda3\lib\site-packages (2.27.239)Note: you may need to restart th
e kernel to use updated packages.

Requirement already satisfied: six in c:\users\karan\anaconda3\lib\site-packages (from docplex) (1.16.0)
```

```
pip install cplex
```

```
Requirement already satisfied: cplex in c:\users\karan\anaconda3\lib\site-packages (22.1.1.2)
Note: you may need to restart the kernel to use updated packages.
```

The document has various variables to be defined at first which can be listed below-

1. **Costs**: Details on the expenses incurred in processing crude oil and the cost of purchasing different types of crude oil.
2. **Prices**: The selling prices of various products derived from crude oil.
3. **Capacities**: Maximum production limits at each refinery location, providing insight into the scale of operations.
4. **Monthly Supply Quotas**: Limits on how much of each type of crude oil can be purchased in a month, ensuring balanced procurement.
5. **Demands**: Quantifies the demand for each product across different regions, indicating market needs.

Our project's objective is to put this data to create an approach which will optimize profit while supplying demand, sustaining capacity for production, and upholding purchasing limits. Making judgements involve calculating exactly the most appropriate quantities of crude oil to purchase and the amount of each product each refinery can generate.

```
costs = {
    "processing": 19,
    "crude": {"Azeri BTC": 57, "Poseidon Streams": 48, "Laguna": 35, "Snøhvit Condensate": 71},
    }

prices = {
    "Gasoline-87": 90.45,
    "Gasoline-89": 93.66,
    "Gasoline-92": 95.50,
    "Jet fuel": 61.25,
    "Diesel fuel": 101.64,
    "Heating oil": 66.36
    }

capacities = {
    "Greece": 400000,
    "Poland": 540000,
    "Spain": 625000,
    "UK": 735000

    }

monthly_supply_quotas = {
    "Azeri BTC": 645000,         # The monthly supply quota for Azeri BTC
    "Poseidon Streams": 575000, # The monthly supply quota for Poseidon Streams
    "Laguna": 550000,            # The monthly supply quota for Laguna
    "Snøhvit Condensate": 645000 # The monthly supply quota for Snøhvit Condensate
    }
```

```
demands = {
    "Gasoline-87": {"Greece": 35000, "Poland": 22000, "Spain": 76000, "UK": 98000},
    "Gasoline-89": {"Greece": 45000, "Poland": 38000, "Spain": 103000, "UK": 52000},
    "Gasoline-92": {"Greece": 50000, "Poland": 60000, "Spain": 83000, "UK": 223000},
    "Jet fuel": {"Greece": 20000, "Poland": 25000, "Spain": 47000, "UK": 127000},
    "Diesel fuel": {"Greece": 75000, "Poland": 35000, "Spain": 125000, "UK": 87000},
    "Heating oil": {"Greece": 25000, "Poland": 205000, "Spain": 30000, "UK": 13000}
    }
```

Our purchase and manufacturing approaches to crude oil products are heavily influenced by the selection variables in our optimisation model. The first collection of variables are the purchase variables, which represent the different quantities of crude oil forms we are able to purchase. Since these variables are adaptable, we can change our purchasing approach depending on to demand and cost considerations. Then there is the production variable, which shows us how much we may produce at every refinery location of each product. Considering them to be our levers of modifying production to limitations in capacity and market demand.

```
# Decision Variables
purchase_vars = mdl.continuous_var_dict(costs["crude"].keys(), name="purchase")
production_vars = mdl.continuous_var_matrix(prices.keys(), capacities.keys(), name="produce")
```

Our guiding principle in our optimization model is to maximize profit, which can be expressed by the objective function. For this, we took into account a number of factors. First, we multiply the cost of each product by the number of units that is produced at each refinery to figure out the revenue which generates from exporting it. This gives us with a projection of the profits from sales. We next calculate for all the associated costs. We estimate the total expense of purchasing crude oil through multiplying the amount purchased by the cost per unit of each type of crude oil. In addition, we multiply the processing cost per unit by the production volume to figure out the cost of production spent at every refinery.

```
def discounted_revenue(production, demand, price):
    # Calculate excess production
    excess = mdl.max(production - demand, 0)
    # Calculate revenue for production up to demand
    regular_revenue = mdl.min(production, demand) * price
    # Calculate revenue for excess production at a discounted rate
    excess_revenue = excess * price * 0.93
    return regular_revenue + excess_revenue
```

```
revenue = mdl.sum(discounted_revenue(production_vars[prod, ref], demands[prod].get(ref, 0), prices[prod])
                  for prod in prices for ref in capacities)
crude_cost = mdl.sum(costs["crude"][crude] * purchase_vars[crude] for crude in costs["crude"])
production_cost = mdl.sum(costs["processing"] * production_vars[prod, ref] for prod in prices for ref in capacities)
mdl.maximize(revenue - crude_cost - production_cost)
```

Now, we defined the constraints for our model which can be explained below-

1. We made sure that each product's production either meets or exceeds local demand. It also means that each product should be produced in sufficient amounts in each region to satisfy the demand that is specific to that region.
2. We make sure that no refinery's output exceeds its capacity. We can't afford to overburden any refinery beyond what is technically reasonable under this constraint.
3. We enforce a 1:1 ratio among the cost of purchasing and consumption of crude oil. This ensures that the total quantity of crude oil used for all products and regions is equivalent to the quantity of crude oil purchased.

4. We meet every variety of crude oil's monthly supply quota. This limit ensures that we avoid purchasing more crude oil than we must purchase each month so as to meet our set supply quota.

```
# 1. Production meets or exceeds demand in each region for each product
for prod, regions in demands.items():
    for region, demand in regions.items():
        mdl.add_constraint(production_vars[prod, region] >= demand)

# 2. Refinery capacities are not exceeded
for ref, capacity in capacities.items():
    mdl.add_constraint(mdl.sum(production_vars[prod, ref] for prod in prices) <= capacity)

# 3. Crude oil purchases and usage with a 1:1 ratio
for crude in costs["crude"]:
    total_crude_usage = mdl.sum(production_vars[prod, ref] for prod in prices for ref in capacities) / len(costs["crude"])
    mdl.add_constraint(purchase_vars[crude] == total_crude_usage)

# 4. Respecting the monthly supply quota for each crude type
for crude, quota in monthly_supply_quotas.items():
    mdl.add_constraint(purchase_vars[crude] <= quota)
```

This part of code develops and resolves the optimization model, subsequently evaluates that a solution was found. It gets and prints the responses if there is an answer. The profit achieved is expressed by the objective value, that is shown. The quantity purchased is then printed for each type of oil that was purchased. It additionally displays the overall amount of every product generated at every refinery. The values provided show the ideal production and purchase schedule and are discovered by the optimization model. If, on the other hand, no viable solution is found, it merely produces a message to that extent. By this procedure, we are able to evaluate and understand the best choices given by the optimization model, providing insightful information for making decisions regarding the production and processing of oil.

```
# Solve the model
solution = mdl.solve()

# Print the solution
if solution:
    print("The objective value (Profit) is: ", mdl.objective_value)
    for crude in costs["crude"]:
        print(f"Purchased {purchase_vars[crude].solution_value} barrels of {crude} crude oil.")
    for prod in prices:
        for ref in capacities:
            print(f"Produced {production_vars[prod, ref].solution_value} barrels of {prod} in {ref} refinery.")
else:
    print("No solution found")
```

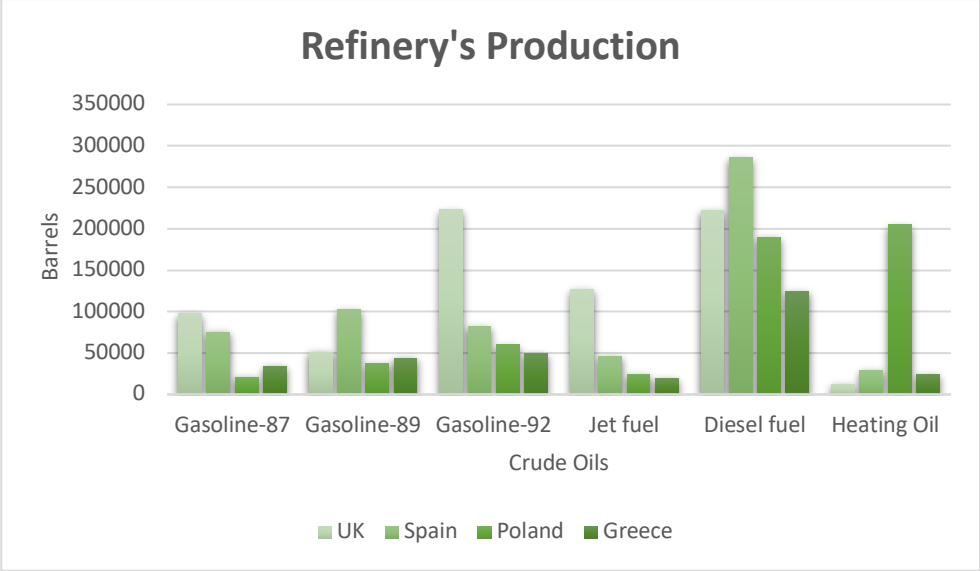
The output would be as follows-

	UK	Spain	Poland	Greece	Total
Gasoline-87	98000	76000	22000	35000	231000
Gasoline-89	52000	103000	38000	45000	238000
Gasoline-92	223000	83000	60000	50000	416000
Jet fuel	127000	47000	25000	20000	219000
Diesel fuel	222000	286000	190000	125000	823000
Heating Oil	13000	30000	205000	25000	273000
Total	735000	625000	540000	300000	2200000

```
The objective value (Profit) is:  36678265.2
Total Revenue:  198092780.0
Total Crude Cost:  116050000.0
Total Production Cost:  41800000.0
Purchased 550000.0 barrels of Azeri BTC crude oil.
Purchased 550000.0 barrels of Poseidon Streams crude oil.
Purchased 550000.0 barrels of Laguna crude oil.
Purchased 550000.0 barrels of Snøhvit Condensate crude oil.
Produced 98000.0 barrels of Gasoline-87 in UK refinery.
Produced 76000.0 barrels of Gasoline-87 in Spain refinery.
Produced 22000.0 barrels of Gasoline-87 in Poland refinery.
Produced 35000.0 barrels of Gasoline-87 in Greece refinery.
Produced 52000.0 barrels of Gasoline-89 in UK refinery.
Produced 103000.0 barrels of Gasoline-89 in Spain refinery.
Produced 38000.0 barrels of Gasoline-89 in Poland refinery.
Produced 45000.0 barrels of Gasoline-89 in Greece refinery.
Produced 223000.0 barrels of Gasoline-92 in UK refinery.
Produced 83000.0 barrels of Gasoline-92 in Spain refinery.
Produced 60000.0 barrels of Gasoline-92 in Poland refinery.
Produced 50000.0 barrels of Gasoline-92 in Greece refinery.
Produced 127000.0 barrels of Jet fuel in UK refinery.
Produced 47000.0 barrels of Jet fuel in Spain refinery.
Produced 25000.0 barrels of Jet fuel in Poland refinery.
Produced 20000.0 barrels of Jet fuel in Greece refinery.
Produced 222000.0 barrels of Diesel fuel in UK refinery.
Produced 286000.0 barrels of Diesel fuel in Spain refinery.
Produced 190000.0 barrels of Diesel fuel in Poland refinery.
Produced 125000.0 barrels of Diesel fuel in Greece refinery.
Produced 13000.0 barrels of Heating oil in UK refinery.
Produced 30000.0 barrels of Heating oil in Spain refinery.
Produced 205000.0 barrels of Heating oil in Poland refinery.
Produced 25000.0 barrels of Heating oil in Greece refinery.
Crude oil transported from each port to UK refinery: 183750.0
Crude oil transported from each port to Spain refinery: 156250.0
Crude oil transported from each port to Poland refinery: 135000.0
Crude oil transported from each port to Greece refinery: 75000.0
PS A:\ACADS\IS6055\Code>
```

Output Analysis:

An analysis of the refinery output reveals several interesting aspects. Firstly, refineries in the UK and Spain boast the highest production capacity, as evidenced by their maximum barrel output. From a profit perspective, diesel appears to be the most produced oil type, potentially signifying greater profitability, followed closely by Gasoline-92. Furthermore, three refineries are operating at full capacity, demonstrating efficient utilization of resources. However, Greece's production falls short of the maximum capacity by 100,000 barrels, suggesting under-utilization or potential constraints. The absence of excess production also implies effective inventory management, as the 7% discount scenario wouldn't be applicable.



Minimum Profit Model

This model being almost like the previous model is just the minimum profit model to find out the min profit we can get from the production of these barrels. The code being almost similar just has a minute change in the objective function-

```
def discounted_revenue(production, demand, price):
    # Calculate excess production
    excess = mdl.max(production - demand, 0)
    # Calculate revenue for production up to demand
    regular_revenue = mdl.min(production, demand) * price
    # Calculate revenue for excess production at a discounted rate
    excess_revenue = excess * price * 0.93
    return regular_revenue + excess_revenue

# Objective Function: Maximize Profit
revenue = mdl.sum(discounted_revenue(production_vars[prod, ref], demands[prod].get(ref, 0), prices[prod])
                  for prod in prices for ref in capacities)
crude_cost = mdl.sum(costs["crude"][crude] * purchase_vars[crude] for crude in costs["crude"])
production_cost = mdl.sum(costs["processing"] * production_vars[prod, ref] for prod in prices for ref in capacities)
mdl.minimize(revenue - crude_cost - production_cost)
```

The objective function is composed of three main components: revenue, crude cost, and production cost. Revenue is calculated by summing the product of the prices for each product and the corresponding production variable for each product and capacity combination. This represents the total revenue generated from selling the produced goods. Crude cost is computed by summing the product of costs per unit of crude and the purchase variable for each type of crude. This accounts for the total cost incurred in purchasing the required crude materials. Production cost is determined by summing the product of processing costs and the production variables for each product and capacity combination. This reflects the total cost of processing and manufacturing the products. The objective function minimizes the difference between revenue and the total costs (crude cost + production cost).

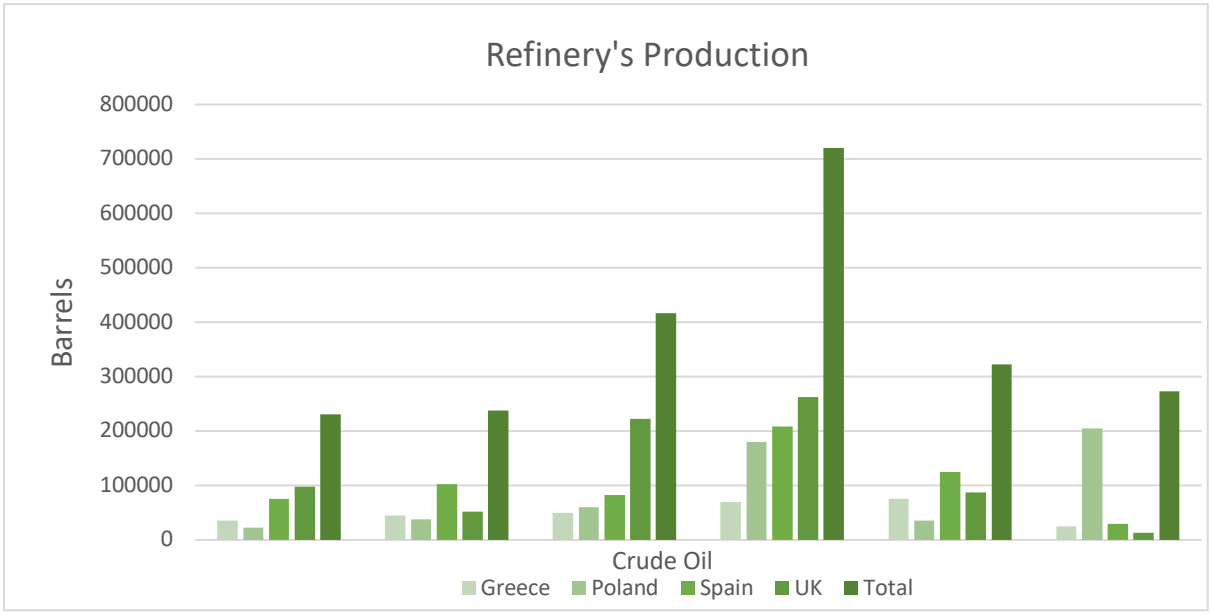
The output would be as follows-

```
The objective value (Profit) is:  17859352.5
Total Revenue:  177857390.0
Total Crude Cost:  116050000.0
Total Production Cost:  41800000.0
Purchased 550000.0 barrels of Azeri BTC crude oil.
Purchased 550000.0 barrels of Poseidon Streams crude oil.
Purchased 550000.0 barrels of Laguna crude oil.
Purchased 550000.0 barrels of Snøhvit Condensate crude oil.
Produced 98000.0 barrels of Gasoline-87 in UK refinery.
Produced 76000.0 barrels of Gasoline-87 in Spain refinery.
Produced 22000.0 barrels of Gasoline-87 in Poland refinery.
Produced 35000.0 barrels of Gasoline-87 in Greece refinery.
Produced 52000.0 barrels of Gasoline-89 in UK refinery.
Produced 103000.0 barrels of Gasoline-89 in Spain refinery.
Produced 38000.0 barrels of Gasoline-89 in Poland refinery.
Produced 45000.0 barrels of Gasoline-89 in Greece refinery.
Produced 223000.0 barrels of Gasoline-92 in UK refinery.
Produced 83000.0 barrels of Gasoline-92 in Spain refinery.
Produced 60000.0 barrels of Gasoline-92 in Poland refinery.
Produced 50000.0 barrels of Gasoline-92 in Greece refinery.
Produced 262000.0 barrels of Jet fuel in UK refinery.
Produced 208000.0 barrels of Jet fuel in Spain refinery.
Produced 180000.0 barrels of Jet fuel in Poland refinery.
Produced 70000.0 barrels of Jet fuel in Greece refinery.
Produced 87000.0 barrels of Diesel fuel in UK refinery.
Produced 125000.0 barrels of Diesel fuel in Spain refinery.
Produced 35000.0 barrels of Diesel fuel in Poland refinery.
Produced 75000.0 barrels of Diesel fuel in Greece refinery.
Produced 13000.0 barrels of Heating oil in UK refinery.
Produced 30000.0 barrels of Heating oil in Spain refinery.
Produced 205000.0 barrels of Heating oil in Poland refinery.
Produced 25000.0 barrels of Heating oil in Greece refinery.
Crude oil transported from each port to UK refinery: 183750.0
Crude oil transported from each port to Spain refinery: 156250.0
Crude oil transported from each port to Poland refinery: 135000.0
Crude oil transported from each port to Greece refinery: 75000.0
PS A:\ACADS\IS6055\Code>
```

Output Analysis:

The model aims at minimizing the profit to find the range of the profits the company can incur with the same production capacity but while increasing the production of jet fuel and decreasing the diesel fuel. As we can see the total minimum profit is coming out to be $17,859,352. We are still not overproducing so there is efficient resource management.

	UK	Spain	Poland	Greece	Total
Gasoline-87	98000	76000	22000	35000	231000
Gasoline-89	52000	103000	38000	45000	238000
Gasoline-92	223000	83000	60000	50000	416000
Jet fuel	262000	208000	180000	70000	720000
Diesel fuel	87000	125000	35000	75000	322000
Heating Oil	13000	30000	205000	25000	273000
Total	735000	625000	540000	300000	2200000



Minimizing Shipping Cost Model

```
import cplex
import sys
import itertools
from cplex.exceptions import CplexError
from cplex import SparsePair
from cplex.exceptions import CplexError, CplexSolverError
```

Importing Libraries

1. import cplex: This imports the CPLEX optimization library into the Python script, allowing access to its functionality for solving linear programming (LP), mixed-integer programming (MIP), and quadratic programming (QP) problems.

2. import sys: This imports the "sys" module, which provides access to some variables used or maintained by the Python interpreter and to functions that interact strongly with the interpreter.

3. import itertools: This imports the "itertools" module, which provides various functions that work on iterators to produce complex iterators.

4. from cplex.exceptions import CplexError: This imports the "CplexError" exception class from the "cplex.exceptions" module. This exception class is used to handle errors specific to the CPLEX library.

5. from cplex import SparsePair: This imports the "SparsePair" class from the "cplex" module. A "SparsePair" object represents a sparse vector, which can be used in various places within the CPLEX library, such as when specifying constraints or objectives.

6. from cplex.exceptions import CplexError, CplexSolverError: This imports both "CplexError" and "CplexSolverError" exception classes from the "cplex.exceptions" module. "CplexError" is used to handle general errors raised by the CPLEX library, while "CplexSolverError" specifically handles errors related to solving optimization problems using CPLEX.

```
# Example crude oil costs per barrel (you should replace these with your actual costs)
crude_oil_costs_per_barrel = {
    "Azeri BTC": 57,
    "Poseidon Streams": 48,
    "Laguna": 35,
    "Snøhvit Condensate": 71
}
```

We are creating a dictionary named crude_oil_costs_per_barrel which contains information about costs of different types of crude oil per barrel in key value format within the dictionary.

```
#Additional data for transportation
tanker_rates = {
    'GPT': {
        'Gudrun': 13000, 'Ingeborg': 22000, 'Valborg': 20000, 'Estrid': 15000, 'Rose': 14000,
        'Cork Cat': 23000, 'Guam': 21000, 'Chance': 16000
    },
    'MRT': {
        'Ismine': 25000, 'Signe': 27000, 'Venture': 23000, 'Pretty World': 25000, 'Viking': 26000,
        'Limerick': 28000, 'York Gulls': 25000, 'Lancaster': 26000
    },
    'LR1': {
        'PTI Volans': 30000, 'Trinity': 32000, 'Galway': 31000, 'Glasgow': 33000
    },
    'LR2': {
        'Garonne': 41000, 'Torm Rhone': 44000, 'Thorpe': 51000, 'Venus': 56000
    }
}
```

This code snippet defines a nested dictionary named **tanker_rates**, which contains information about the rates of different types of tankers for transporting crude oil. We have used key value pair for all the classes.

```
tankers_capicity_in_barrels = {'GPT': {'Gudrun': 84500,
   'Ingeborg': 205334,
   'Valborg': 173224,
   'Estrid': 141115,
   'Rose': 101399,
   'Cork Cat': 211249,
   'Guam': 188857,
   'Chance': 146523},
  'MRT': {'Ismine': 270400,
   'Signe': 358279,
   'Venture': 226459,
   'Pretty World': 316452,
   'Viking': 287300,
   'Limerick': 377714,
   'York Gulls': 242092,
   'Lancaster': 332211},
  'LR1': {'PTI Volans': 515449,
   'Trinity': 574600,
   'Galway': 557700,
   'Glasgow': 650650},
  'LR2': {'Garonne': 929499,
   'Torm Rhone': 1153425,
   'Thorpe': 1309750,
   'Venus': 1352000}}
```

The above code we are making nested dictionary named “tankers_capacity_in_barrels” consisting of capacity of barrels for each tanker.

```
port_charges = {
    'Ceyhan': {
        'GPT': 109000, 'MRT': 112000, 'LR1': 124000, 'LR2': 135000
    },
    'Houma': {
        'GPT': 111000, 'MRT': 114000, 'LR1': 138000, 'LR2': 159000
    },
    'Puerto Miranda': {
        'GPT': 135000, 'MRT': 147000, 'LR1': 158000, 'LR2': 169000
    },
    'Melkoya': {
        'GPT': 136000, 'MRT': 147000, 'LR1': 156000, 'LR2': 177000
    }
}
```

The above snippet is about dictionary port_charges which consists of all the port charges.

```
# Cost per hour
fuel_costs = {
    "GPT": 2500,
    "MRT": 2750,
    'LR1': 3000,
    'LR2': 3250,
}
```

This snippet explains the cost per hour for all the classes.

```
# Times in days from port to refinery
shipping_times_in_days = {
    'Ceyhan': {'Greece': 2, 'Poland': 15, 'Spain': 8, 'UK': 12},
    'Houma': {'Greece': 20, 'Poland': 18, 'Spain': 16, 'UK': 15},
    'Puerto Miranda': {'Greece': 19, 'Poland': 20, 'Spain': 14, 'UK': 15},
    'Melkoya': {'Greece': 11, 'Poland': 3, 'Spain': 4, 'UK': 3}
}
```

We made nested dictionary named shipping_times_in_days which contains key value of all the ports and travel times.

```
# Hypothetical mapping from crude oil types to ports
crude_to_port = {
    "Azeri BTC": "Ceyhan",
    "Poseidon Streams": "Houma",
    "Laguna": "Puerto Miranda",
    "Snøhvit Condensate": "Melkoya"
}
crude_to_tanker_type = {
    "Azeri BTC": "GPT",
    "Poseidon Streams": "MRT",
    "Laguna": "LR1",
    "Snøhvit Condensate": "LR2"
}
```

These dictionaries facilitate easy lookup of the port and tanker type associated with a specific type of crude oil. They provide a convenient way to organize and access this mapping information within the code.

```
# New quantities per destination
quantities_per_destination = {
    "UK": 183750,
    "Spain": 156250,
    "Poland": 135000,
    "Greece": 75000
}

# Invert the crude_to_tanker_type dictionary
tanker_type_to_crude = {v: k for k, v in crude_to_tanker_type.items()}
```

This dictionary, quantities_per_destination, specifies the quantities of crude oil that need to be transported to different destinations. Each key represents a destination (e.g., "UK", "Spain", "Poland", "Greece"), and the corresponding value represents the quantity of crude oil (in

barrels) that needs to be transported to that destination. The subsequent code creates a new dictionary tanker_type_to_crude where the keys are tanker types and the values are the corresponding crude oil types.

```
# Create a list of ports, tankers, and destinations
ports = list(port_charges.keys())
tankers = [tanker for sublist in tankers_capicity_in_barrels.values() for tanker in sublist]
destinations = ["Greece", "Poland", "Spain", "UK"]
```

Using above code we are creating list, these lists are created to provide easy access to the names of ports, tankers, and destinations, which may be useful for various calculations or operations within the code. Firstly, it retrieves the names of ports by extracting the keys from the dictionary "port_charges" using the "keys()" method and converting them into a list. Secondly, it compiles a list of tankers by iterating over the values of the dictionary "tankers_capicity_in_barrels" and appending each tanker to the list. Lastly, it directly creates a list called "destinations", comprising the names of four predetermined destinations: Greece, Poland, Spain, and the UK. These lists are intended to facilitate easy access to relevant information for subsequent calculations or operations within the code.

```
# Correct instantiation of a Cplex object
model = cplex.Cplex()
```

This line of code correctly instantiates a Cplex object named "model". The "cplex.Cplex()" function call creates a new instance of a Cplex object, which can then be used to define and solve optimization problems using the CPLEX optimization library in Python. However, typically you would specify additional parameters or options when creating the Cplex object to customize its behavior according to your specific optimization problem.

```
# Variables: binary decision for each port, tanker, destination
# Format: "Port_Tanker_Destination"
variables = ["{}_{}_{}".format(port, tanker, destination)
             for port, tanker, destination in itertools.product(ports, tankers, destinations)]
```

This code snippet utilizes list comprehension and the itertools module to construct a list of variable names representing binary decision variables for every conceivable combination of port, tanker, and destination. It iterates over each combination of elements from the lists of ports, tankers, and destinations, formatting a string representation for each combination. These formatted strings, representing variable names in the format "Port_Tanker_Destination", are then appended to the list of variables. This approach provides a concise and efficient way to generate the necessary variables for formulating optimization problems involving the transportation of crude oil.

```
# Add variables to the model
model.variables.add(names=variables, types=["B"] * len(variables))
```

This code snippet enhances the previously instantiated Cplex model by introducing variables into it. It employs the “variables.add()” method to achieve this. Specifically, the “names” parameter specifies the variable names derived earlier, while the “types” parameter assigns binary type ("B") to each variable, indicating they are binary decision variables. The multiplication “* len(variables)” ensures that each variable is matched with its corresponding type. Essentially, this line of code ensures the incorporation of essential binary decision variables into the model, a pivotal step in formulating and solving optimization problems using the CPLEX optimization library.

```
# Function to create consistent variable names
def create_var_name(port, tanker, destination):
    return f"{port}_{tanker}_{destination}"
```

This function, named “create_var_name”, is designed to produce uniform variable names using input parameters representing a combination of port, tanker, and destination. The function accepts three parameters: “port”, “tanker”, and “destination”, which correspond to the components necessary for forming a variable name. Inside the function, an f-string is utilized to concatenate these parameters with underscores (_) as separators, resulting in a consistent variable name format. This function offers a concise and reusable approach to generating variable names, ensuring consistency across the naming convention, particularly beneficial when handling extensive sets of variables within optimization problems.

```
# Adjust the objective function calculation
objective = []
for tanker_type in tankers_capicity_in_barrels:
    crude_type = tanker_type_to_crude[tanker_type]  # Get the crude type for this tanker type
    port = crude_to_port[crude_type]  # Get the port for this crude type
    for tanker in tankers_capicity_in_barrels[tanker_type]:
        for destination in destinations:
            quantity = quantities_per_destination[destination]
            var_name = create_var_name(port, tanker, destination)
            rate = tanker_rates[tanker_type].get(tanker, 0)
            port_charge = port_charges[port].get(tanker_type, 0)
            fuel_cost_per_hour = fuel_costs[tanker_type]
            shipping_days = shipping_times_in_days[port].get(destination, 0)
            fuel_cost = fuel_cost_per_hour * shipping_days * 24
            crude_oil_cost = crude_oil_costs_per_barrel[crude_type] * quantity
            total_cost = rate + port_charge + fuel_cost + crude_oil_cost
            objective.append((var_name, total_cost))

model.objective.set_sense(model.objective.sense.minimize)
model.objective.set_linear(objective)
```

This code constructs the objective function for an optimization model by iterating over combinations of port, tanker, and destination. It initializes an empty list named “objective” to store tuples containing variable names and their corresponding total costs. It iterates over tanker types and, for each type, retrieves associated crude types and ports. Then, it calculates the total cost for each combination of port, tanker, and destination, considering factors like tanker rates, port charges, fuel costs, and crude oil costs. These total costs are appended to the “objective”

list. Finally, the code sets the objective function sense to minimize and defines the linear part of the objective function using the list of tuples stored in "objective". Overall, this adjustment ensures that the optimization model aims to minimize the total cost associated with transporting crude oil to different destinations using various tankers and ports.

```
# Step 1: Create a dictionary for costs and capacities
costs_and_capacities = {}
for tanker_type in tankers_capicity_in_barrels:
    for tanker in tankers_capicity_in_barrels[tanker_type]:
        tanker_capacity = tankers_capicity_in_barrels[tanker_type][tanker]
        for port in ports:
            for destination in destinations:
                var_name = "{}_{}_{}".format(port, tanker, destination)
                rate = tanker_rates[tanker_type].get(tanker, 0)
                port_charge = port_charges[port].get(tanker_type, 0)
                fuel_cost_per_hour = fuel_costs[tanker_type]
                shipping_days = shipping_times_in_days[port].get(destination, 0)
                fuel_cost = fuel_cost_per_hour * shipping_days * 24
                total_cost = rate + port_charge + fuel_cost
                costs_and_capacities[var_name] = (total_cost, tanker_capacity)
```

This code computes the costs and capacities associated with transporting fuel using different types of tankers from various ports to diverse destinations. It begins by initializing an empty dictionary called "costs_and_capacities", which will store the calculated total costs and tanker capacities for each unique combination of port, tanker, and destination. Through nested loops, it iterates over each tanker type and then each specific tanker within that type, along with all available ports and destinations. Within these iterations, it dynamically generates a variable name representing the combination of port, tanker, and destination. The code then calculates the total cost of transportation considering factors such as the tanker's rate, port charges, fuel costs, and shipping times. Finally, it stores the computed total cost and tanker capacity in the "costs_and_capacities" dictionary using the generated variable name as the key. This systematic approach enables efficient computation and storage of essential data for fuel transportation logistics.

```
allocated_boats = set()

sorted_boats_by_route = {}
for port in ports:
    for destination in destinations:
        required_quantity = quantities_per_destination[destination]

        # Filter boats that are not already allocated and have sufficient capacity for the required quantity
        eligible_boats = [(var_name, cost) for var_name, (cost, capacity) in costs_and_capacities.items()
                          if var_name.split('_')[1] not in allocated_boats and capacity >= required_quantity and var_name.startswith(port +

        if eligible_boats:
            # Sort by cost and select the cheapest boat
            sorted_boats = sorted(eligible_boats, key=lambda x: x[1])
            cheapest_boat = sorted_boats[0][0]
            sorted_boats_by_route[(port, destination)] = [cheapest_boat]

            # Add the selected boat to the allocated set
            allocated_boats.add(cheapest_boat.split('_')[1])
        else:
            # No eligible boats available for this route
            sorted_boats_by_route[(port, destination)] = []
```

This section of the code is responsible for the allocation of boats to transportation routes based on considerations such as cost and capacity. It begins by initializing two data structures: a set named "allocated_boats", which tracks boats that have already been assigned to routes, and a dictionary called "sorted_boats_by_route", which will contain sorted lists of boats for each route. Through nested loops iterating over the available ports and destinations, the code

determines the quantity of fuel required for each destination. It then filters out boats that are eligible for allocation, considering criteria such as whether the boat has already been assigned (not in allocated_boats), if it has the necessary capacity (capacity >= required_quantity), and if it is available at the current port. Eligible boats are sorted by cost, and the cheapest boat is selected for each route. This information is stored in the "sorted_boats_by_route" dictionary. Additionally, the selected boat is added to the "allocated_boats" set to mark it as allocated for future reference. If no eligible boats are found for a particular route, an empty list is assigned to indicate the unavailability of boats. This allocation process optimizes fuel transportation logistics by efficiently assigning boats to routes based on cost, capacity, and availability constraints.

```
# Constraint: Each boat can only be assigned to one route
for tanker in tankers:
    vars_for_tanker = [create_var_name(port, tanker, destination) for port in ports for destination in destinations]
    model.linear_constraints.add(
        lin_expr=[ [vars_for_tanker, [1] * len(vars_for_tanker)] ],
        senses=["L"],
        rhs=[1]
    )

for port in ports:
    for destination in destinations:
        quantity = quantities_per_destination[destination]
        vars_for_port_destination = []

        # Capacity coefficients for each variable
        capacity_coefficients = []

        for tanker in tankers:
            var_name = create_var_name(port, tanker, destination)
            vars_for_port_destination.append(var_name)

            # Determine the tanker type based on the tanker's name
            for tanker_type, tankers_list in tankers_capicity_in_barrels.items():
                if tanker in tankers_list:
                    # Get the capacity of the tanker
                    capacity = tankers_capicity_in_barrels[tanker_type][tanker]
                    break

            capacity_coefficients.append(capacity)

        # Adjust the right-hand side of the constraint to match the new quantity
        model.linear_constraints.add(
            lin_expr=[ [vars_for_port_destination, capacity_coefficients] ],
            senses=["G"],  # Greater than or equal to, to ensure enough capacity
            rhs=[quantity]
        )
```

This code segment operates within a broader optimization model and is tasked with setting up constraints related to tanker capacity. It iterates over each tanker and generates variable names for each combination of ports and destinations using the "create_var_name" function. These variable names represent the amount of fuel transported by each tanker from a particular port to a specific destination. Subsequently, the code adds linear constraints to the optimization model using the "linear_constraints.add" method.

For each tanker, it adds a constraint ensuring that only one route is selected for transportation, indicating that each tanker can only be assigned to one route. This constraint is enforced by setting the sum of variables associated with each tanker to be less than or equal to 1.

Next, for each combination of ports and destinations, the code calculates the required quantity of fuel based on the destination's demand. It then constructs constraints to ensure that the total

capacity of tankers assigned to transport fuel from a particular port to a specific destination is sufficient to meet the demand. The capacities of the tankers are used as coefficients in these constraints to reflect the available capacity for each route. These constraints are added to the model with the sense set to "greater than or equal to," ensuring that the total capacity of assigned tankers is at least equal to the required quantity of fuel for each destination.

```
# Additional Constraint: Each port-destination pair should have exactly one tanker assigned
for port in ports:
    for destination in destinations:
        vars_for_port_destination = [create_var_name(port, tanker, destination) for tanker in tankers]
        # Add constraint for each port-destination pair
        model.linear_constraints.add(
            lin_expr=[[vars_for_port_destination, [1] * len(vars_for_port_destination)]],
            senses=["E"],  # "E" stands for equality
            rhs=[1]
        )
```

Then we added additional constraint that iterates through pairs of ports and destinations to assign exactly one tanker to each pair. For each combination, it generates a constraint ensuring that the sum of assignment variables for each port-destination pair equals one, meaning that exactly one tanker is assigned to that route in the solution.

```
# Solve the model
try:
    model.solve()
    print("Model solved successfully.")
except CplexError as exc:
    print("Error solving model:", exc)
    sys.exit(1)  # Exit if model couldn't be solved

# Initialize total cost
total_cost = 0

# Retrieve and print the solution
solution_values = model.solution.get_values()
for var_name, value in zip(variables, solution_values):
    if value > 0.5:  # If the variable is part of the solution
        route_cost = costs_and_capacities[var_name][0]
        total_cost += route_cost  # Add the cost of this route to the total cost
        # print(f"{var_name} selected with a cost of {route_cost}")

# Print the total cost
print(f"Total cost of the solution: {total_cost}")
```

Following the optimization of the model, the code attempts to solve it using the CPLEX solver by calling “model.solve()”. Upon successful completion of the solving process, it prints a message confirming the successful resolution of the model. In case of any errors during the solving process, it catches the “CplexError”, prints an error message indicating the issue, and exits the program with a status code of 1.

After solving the model, the code initializes the variable “total_cost” to track the cumulative cost of the solution. It then retrieves the solution values using “model.solution.get_values()”. For each variable in the solution, if its value is greater than 0.5, indicating it is part of the solution, the code calculates the cost associated with the corresponding route and adds it to the “total_cost”. Finally, it prints out the total cost of the solution.

This segment of the code effectively handles the solving of the optimization model, retrieves the solution, calculates the total cost, and prints it out for analysis and further action.

```python
# Check the solution status
solution = model.solution
if solution.is_primal_feasible():
    print("Solution status = ", solution.get_status())
    found_solution = False
    for tanker_type in tankers_capicity_in_barrels:
        for tanker in tankers_capicity_in_barrels[tanker_type]:
            for port in crude_to_port.values():
                for destination in destinations:
                    var_name = create_var_name(port, tanker, destination)
                    try:
                        if solution.get_values(var_name) > 0.5:
                            # Quantity required for the destination
                            quantity = quantities_per_destination[destination]
                            route_cost = costs_and_capacities[var_name][0]  # Get the route cost from the dictionary
                            print(f"From {port}, Tanker {tanker} to {destination}, transporting {quantity} barrels. Cost: {route_cost}")
                            found_solution = True
                    except CplexSolverError as e:
                        print(f"Error accessing variable '{var_name}': {e}")
    if not found_solution:
        print("No routes selected in the solution.")
else:
    print("No solution available.")
```

This section of the code evaluates the feasibility of the obtained solution and provides information about the selected routes if a feasible solution is found. It begins by checking if the solution is feasible using “solution.is_primal_feasible()”. If the solution is feasible, it prints the status of the solution obtained from “solution.get_status()”. Then, it iterates over various combinations of tankers, ports, and destinations to identify the selected routes.

For each combination, it constructs the variable name representing the route using “create_var_name” and attempts to retrieve the value of the variable from the solution. If the value is greater than 0.5, indicating that the route is part of the solution, it retrieves the required quantity of fuel for the destination and the cost of the route from the “quantities_per_destination” and “costs_and_capacities” dictionaries, respectively. It then prints information about the selected route, including the departure port, tanker type, destination, quantity of fuel transported, and the associated cost.

In case of any errors encountered during the retrieval of variable values, it catches the “CplexSolverError” and prints an error message. If no solution is found or no routes are selected in the solution, appropriate messages are printed to indicate the absence of a valid solution or the absence of selected routes.

```
Root node processing (before b&c):
  Real time             =    0.05 sec. (3.23 ticks)
Parallel b&c, 8 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.05 sec. (3.23 ticks)
Model solved successfully.
Total cost of the solution: 16029000
Solution status =  101
From Puerto Miranda, Tanker Gudrun of type GPT going to Greece, transporting 75000 barrels with shipping cost: 1288000
From Puerto Miranda, Tanker Ingeborg of type GPT going to Spain, transporting 156250 barrels with shipping cost: 997000
From Melkoya, Tanker Cork Cat of type GPT going to Spain, transporting 156250 barrels with shipping cost: 399000
From Houma, Tanker Guam of type GPT going to Greece, transporting 75000 barrels with shipping cost: 1332000
From Puerto Miranda, Tanker Venture of type MRT going to UK, transporting 183750 barrels with shipping cost: 1160000
From Puerto Miranda, Tanker Pretty World of type MRT going to Poland, transporting 135000 barrels with shipping cost: 1492000
From Melkoya, Tanker Limerick of type MRT going to UK, transporting 183750 barrels with shipping cost: 373000
From Melkoya, Tanker York Gulls of type MRT going to Greece, transporting 75000 barrels with shipping cost: 898000
From Ceyhan, Tanker Lancaster of type MRT going to Greece, transporting 75000 barrels with shipping cost: 270000
From Melkoya, Tanker PTI Volans of type LR1 going to Poland, transporting 135000 barrels with shipping cost: 402000
From Houma, Tanker Trinity of type LR1 going to UK, transporting 183750 barrels with shipping cost: 1250000
From Ceyhan, Tanker Galway of type LR1 going to UK, transporting 183750 barrels with shipping cost: 1019000
From Ceyhan, Tanker Glasgow of type LR1 going to Spain, transporting 156250 barrels with shipping cost: 733000
From Houma, Tanker Garonne of type LR2 going to Poland, transporting 135000 barrels with shipping cost: 1604000
From Houma, Tanker Torm Rhone of type LR2 going to Spain, transporting 156250 barrels with shipping cost: 1451000
From Ceyhan, Tanker Venus of type LR2 going to Poland, transporting 135000 barrels with shipping cost: 1361000
PS A:\ACADS\IS6055\Code>
```

Output Analysis:

Port Of Shipping (From)	Tanker Type	Destination (To)	No. of Barrels	Shipping Cost($)
Puerto Miranda	Gudrun	Greece	75000	1288000
Puerto Miranda	Ingeborg	Spain	156250	997000
Melkoya	Cork Cat	Spain	156250	399000
Houma	Guam	Greece	75000	1332000
Puerto Miranda	Venture	UK	183750	1160000
Puerto Miranda	Pretty World	Poland	135000	1492000
Melkoya	Limerick	UK	183750	373000
Melkoya	York Gulls	Greece	75000	898000
Ceyhan	Lancaster	Greece	75000	270000
Melkoya	PTI Volans	Poland	135000	402000
Houma	Trinity	UK	183750	1250000
Ceyhan	Galway	UK	183750	1019000
Ceyhan	Glasgow	Spain	156250	733000
Houma	Garonne	Poland	135000	1604000
Houma	Torm Rhone	Spain	156250	1451000
Ceyhan	Venus	Poland	135000	1361000
Total			2200000	16029000

The provided output lists the assignments of various tankers to transport crude oil, as determined by the optimization model. The model has calculated the most cost-effective way to ship specific quantities of crude oil to different destinations, while considering various constraints such as shipping times, costs, and tanker capacities. Here's what the information in the output signifies:

Total cost of the solution: $16,029,000: This is the sum of the costs for all the shipping assignments made by the model. It's the minimized figure the model has computed based on the parameters and constraints fed into it.

From [Port], Tanker [Name] of type [Type] going to [Destination]: This describes the route that each tanker will take:

- [Port] is the starting point where the tanker will load the crude oil.
- [Name] is the name of the tanker that will be used for the shipment.
- [Type] is the category or class of the tanker, which can affect its rate and capacity.
- [Destination] is the location where the tanker will unload the crude oil.
- Transporting [Quantity] barrels: This is the amount of crude oil, in barrels, that the tanker will carry on the specified route.
- With shipping cost: [Cost]: This is the total cost of shipping the specified quantity of crude oil on that route with the assigned tanker. It likely includes the costs of fuel, port charges, and other expenses.

For example, the first line of the detailed output reads:

From Puerto Miranda, Tanker Gudrun of type GPT going to Greece, transporting 75,000 barrels with shipping cost: $1,288,000.

This tells us that the tanker named Gudrun, which is a General Purpose Tanker (GPT), is assigned to transport 75,000 barrels of crude oil from Puerto Miranda to Greece. The total cost for this particular shipment is $1,288,000.

Each line in the output follows a similar pattern, detailing the logistics and costs for each tanker's route. The list as a whole represents the complete set of shipping assignments that make up the optimal solution for the given set of shipping requirements and constraints.

In the tables provided below, we have computed the cost per trip for transporting petroleum products using different ships.

For instance, when transporting from port Ceyhan to Greece using the GPT ship, the trip duration is 2 days.

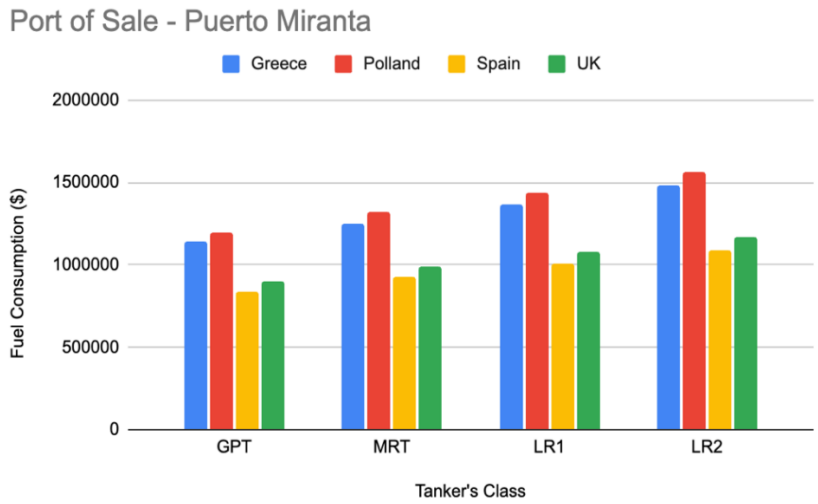
The cost is calculated as follows: 2 days * 2500 $/hr (fuel consumption) * 24 hours.

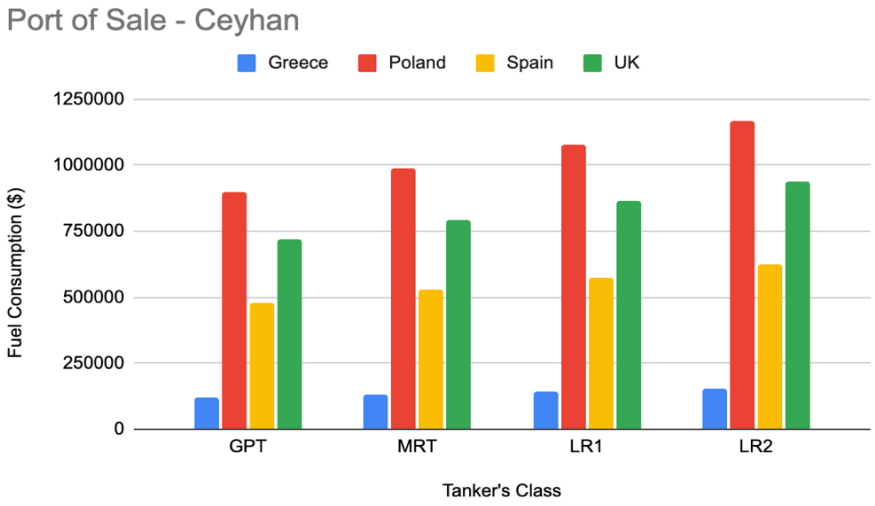
Ceyhan	Greece	Poland	Spain	UK
GPT	120000	900000	480000	720000
MRT	132000	990000	528000	792000
LR1	144000	1080000	576000	864000
LR2	156000	1170000	624000	936000

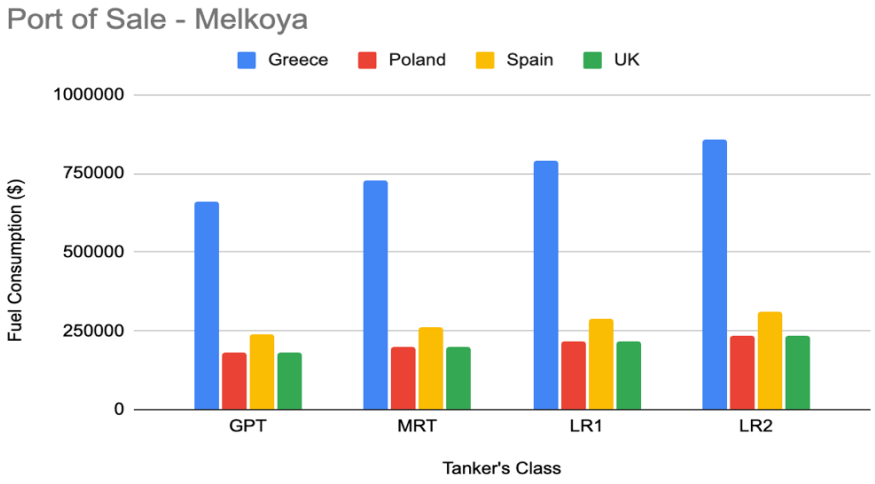
Houma	Greece	Poland	Spain	UK
GPT	1200000	1080000	960000	900000
MRT	1320000	1188000	1056000	990000
LR1	1440000	1296000	1152000	1080000
LR2	1560000	1404000	1248000	1170000

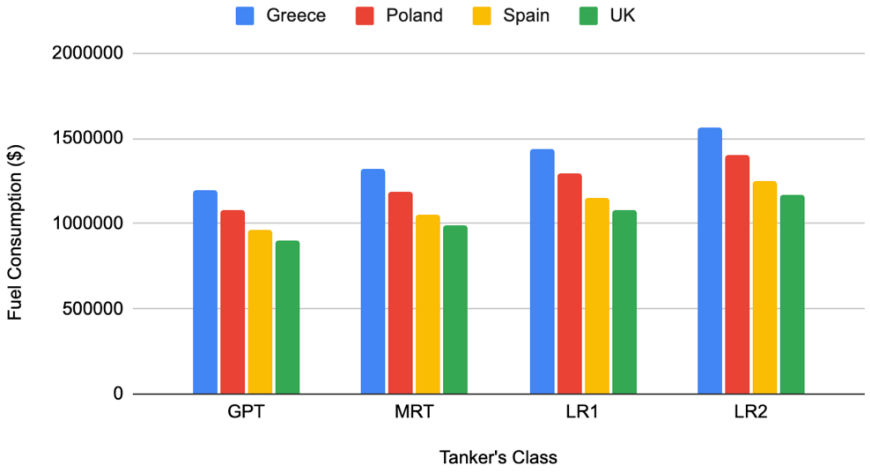
Puerto Miranta	Greece	Poland	Spain	UK
GPT	1140000	1200000	840000	900000
MRT	1254000	1320000	924000	990000
LR1	1368000	1440000	1008000	1080000
LR2	1482000	1560000	1090000	1170000

Melkoya	Greece	Poland	Spain	UK
GPT	660000	180000	240000	180000
MRT	726000	198000	264000	198000
LR1	792000	216000	288000	216000
LR2	858000	234000	312000	234000




Port of Sale - Ceyhan
Greece
Poland
Spain
UK
1250000
1000000
750000
500000
250000
0
Fuel Consumption ($)
GPT
MRT
LR1
LR2
Tanker's Class


Port of Sale - Melkoya
Greece
Poland
Spain
UK
1000000
750000
500000
250000
0
Fuel Consumption ($)
GPT
MRT
LR1
LR2
Tanker's Class


Port of Sale - Houma
Greece
Poland
Spain
UK
2000000
1500000
1000000
500000
0
Fuel Consumption ($)
GPT
MRT
LR1
LR2
Tanker's Class

These graph dives deeper into the breakdown of transportation costs per ship for a single trip originating from different ports. While the table represents the total cost encompassing all aspects, this graph focuses specifically on the fuel consumption expenses for voyages to various destinations – Greece, Poland, Spain, and the UK.

The horizontal axis, also known as the x-axis, categorizes different tanker classes used for these trips. These classes are identified as GPT, MRT, LR1, and LR2. By analysing the graph, we can identify which tanker class is the most fuel-efficient for each destination.

Here's a breakdown of what the graph reveals:

Fuel Efficiency: Looking across the destinations, it appears the GPT tanker class offers the most fuel-efficient option. This means for a single trip to any of the destinations, using an GPT tanker would result in the lowest fuel cost compared to the other classes (MRT, LR1, LR2). Conversely, the LR2 tanker class seems to be the least fuel-efficient option across the board.

Factors Affecting Efficiency: Several factors might influence this difference in fuel efficiency. The size and overall capacity of each tanker class could be a key factor. Larger tankers (like LR2) might be able to carry more fuel while expending less fuel per unit transported compared to smaller tankers (like GPT). Additionally, the distance to each destination could play a role. Longer journeys might inherently require more fuel regardless of the tanker class. Finally, the type of fuel being transported might also influence efficiency – heavier or denser fuels might demand adjustments in fuel consumption.

Remember, this graph only presents the fuel consumption cost. The total transportation expense per trip, as mentioned in the table, would likely encompass additional factors beyond fuel.

Conclusion and Recommendations

In conclusion, through our findings we have optimized all possible areas identified in our brief: planning purchasing, production, and transportation activities and we have given data-driven recommendations to ensure profits, maximizing them where possible and minimizing costs where possible. Our first 2 models measured our maximum and minimum profit, based off selling only Diesel Fuel and Jet Fuel as our excesses. We built these models disregarding shipping & transportation costs. These models factored in the cost price of each of our crude oils multiplied by the amount of barrels of each we were taking to give us our total cost. They then calculated the profit we got from each individual product by meeting our quotas and then calculated our excess production based on our lowest and highest refined product sales price, Jet Fuel ($61.25) and Diesel Fuel ($101.64), sold at a 7% discount for our excess production amount of 501,000.

Next, we looked at our minimum and maximum profit figures and we took the optimal shipping costs that we got from our third model and subtracted that from our profits. We found that no matter what refined product we produced, we were guaranteed to be in profit so long as we hit our production figure of 2,200,000. This gave us justification for disregarding API gravity and Sulphur content. We proved that if we only produced our cheapest product, with the method we have chosen we would still be turning a profit after meeting our demands so it would make sense. No matter the restraints API gravity and sulphur content would have on what we could produce, we can guarantee that we turn a profit of minimum $1.8m and maximum $20m, as indicated by the table below.

	Profit	**Shipping Cost**	**Total Profit**
Minimum	17,859,352	16,029,000	1,830,352
Maximum	36,678,265	16,029,000	20,649,265.00

References

Sulfur (content) (no date). https://www.mckinseyenergyinsights.com/resources/refinery-reference-desk/sulfur-content/#:~:text=Most%20of%20the%20sulfur%20in,are%20cracked%20and%20form%20H2S.