Gayathri Mude

It implements two quantitative techniques covered in the course:

One major optimization model and one major forecasting model.

The Major Optimization Model

1. Introduction

This report presents a Python-based Supply Chain Optimization model. The objective is to assign demand points to facilities while minimizing a weighted sum of cost and distance. The model implements multi-objective optimization using the PuLP library.

2. Python Code

Below is the full code combining the model class and example usage:

PROJECT.py 3 ×

C: > Users > ADMIN > OneDrive > Desktop > PROJECT.py > SupplyChainOptimizer > solve

```python
import subprocess
import sys

# Step 1: Ensure PuLP is installed
try:
    import pulp
except ImportError:
    print("PuLP not found. Installing PuLP...")
    subprocess.check_call([sys.executable, "-m", "pip", "install", "pulp"])
    import pulp

# Step 2: Define the SupplyChainOptimizer class
from pulp import LpProblem, LpVariable, lpSum, LpMinimize, PULP_CBC_CMD

class SupplyChainOptimizer:
    def __init__(self, costs, distances, facilities, demand_points):
        """
        Initializes the SupplyChainOptimizer.

        Parameters:
        - costs: dict of dicts, e.g., costs[facility][demand_point]
        - distances: dict of dicts, e.g., distances[facility][demand_point]
        - facilities: list of facility names
        - demand_points: list of demand point names
        """
        self.costs = costs
        self.distances = distances
        self.facilities = facilities
        self.demand_points = demand_points
```

C: > Users > ADMIN > OneDrive > Desktop > PROJECT.py > SupplyChainOptimizer > solve

```python
class SupplyChainOptimizer:

    def solve(self, weight_cost=0.5, weight_distance=0.5):
        """
        Solves the optimization problem using a weighted objective function.

        Parameters:
        - weight_cost: weight for cost in the objective (0 to 1)
        - weight_distance: weight for distance in the objective (0 to 1)

        Returns:
        - solution: dictionary of assigned facility-demand pairs
        """
        # Create a minimization problem
        model = LpProblem("MultiObjectiveFacilityLocation", LpMinimize)

        # Decision variables: x[(facility, demand_point)] ∈ {0, 1}
        x = {
            (f, d): LpVariable(f"x_{f}_{d}", cat="Binary")
            for f in self.facilities
            for d in self.demand_points
        }

        # Objective: weighted sum of cost and distance
        model += lpSum([
            weight_cost * self.costs[f][d] * x[(f, d)] +
            weight_distance * self.distances[f][d] * x[(f, d)]
            for f in self.facilities
            for d in self.demand_points
        ])

        # Constraint: each demand point is served by exactly one facility
        for d in self.demand_points:
            model += lpSum([x[(f, d)] for f in self.facilities]) == 1, f"Assign_{d}"

```

PROJECT.py 3 ×

C: > Users > ADMIN > OneDrive > Desktop > PROJECT.py > SupplyChainOptimizer > solve

```python
class SupplyChainOptimizer:
    def solve(self, weight_cost=0.5, weight_distance=0.5):

        # Solve the model
        model.solve(PULP_CBC_CMD(msg=False))

        # Extract solution
        solution = {
            (f, d): x[(f, d)].varValue
            for f in self.facilities
            for d in self.demand_points
            if x[(f, d)].varValue == 1
        }

        return solution

```

```python
# Step 3: Sample usage
if __name__ == "__main__":
    # Sample data
    facilities = ['F1', 'F2']
    demand_points = ['D1', 'D2', 'D3']
    costs = {
        'F1': {'D1': 10, 'D2': 20, 'D3': 15},
        'F2': {'D1': 12, 'D2': 18, 'D3': 25}
    }
    distances = {
        'F1': {'D1': 5, 'D2': 10, 'D3': 6},
        'F2': {'D1': 6, 'D2': 7, 'D3': 9}
    }

    # Initialize optimizer
    optimizer = SupplyChainOptimizer(costs, distances, facilities, demand_points)

    # Solve with specified weights
    solution = optimizer.solve(weight_cost=0.6, weight_distance=0.4)

    # Display the solution
    print("Optimal Assignments:")
    for (facility, demand_point), value in solution.items():
        print(f"Facility {facility} serves Demand Point {demand_point}")
```

3. Output Screenshot

The below screenshot shows the output of the optimization model execution:

```
PS C:\Users\ADMIN> & C:/Users/ADMIN/.julia/conda/3/x86_64/python.exe c:/Users/ADMIN/OneDrive/Desk
Optimal Assignments:
Facility F1 serves Demand Point D1
Facility F1 serves Demand Point D3
Facility F2 serves Demand Point D2
PS C:\Users\ADMIN>
```

Forecasting Model : Simple Exponential Smoothing (SES)

1. Introduction

This report explains the implementation of a forecasting model using Simple Exponential Smoothing (SES). SES is a time series technique that gives more importance to recent data and less to older values. It is widely used in inventory planning and demand forecasting when the data has no strong trend or seasonality.

2. Python Code

Below is the full code including the SES model class and an example usage:

ses_model_and_example_combined.py ×

C: > Users > ADMIN > Downloads > ses_model_and_example_combined.py > ...

```python

# Simple Exponential Smoothing (SES) Model with Full Example

class SimpleExponentialSmoothing:
    def __init__(self, alpha):
        """
        Initialize the SES model with smoothing parameter alpha.
        Alpha must be between 0 and 1.
        """
        if not (0 < alpha <= 1):
            raise ValueError("Alpha must be between 0 and 1.")
        self.alpha = alpha

    def forecast(self, demand_series):
        """
        Forecast the next demand value using Simple Exponential Smoothing.

        Parameters:
        - demand_series: list of historical demand values

        Returns:
        - forecast value for the next period
        """
        if not demand_series:
            raise ValueError("Demand series cannot be empty.")

        forecast = demand_series[0]
        for actual in demand_series[1:]:
            forecast = self.alpha * actual + (1 - self.alpha) * forecast
        return forecast
```

ses_model_and_example_combined.py ×

C: > Users > ADMIN > Downloads > ses_model_and_example_combined.py > ...

```python
        return forecast

# Full Example Usage
if __name__ == "__main__":
    # 12-month historical demand data
    monthly_demand = [120, 135, 150, 145, 160, 155, 170, 165, 180, 175, 190, 185]
    alpha = 0.5

    print("Historical Demand Data:", monthly_demand)
    print(f"Using alpha = {alpha}")

    # Create model and forecast next month
    model = SimpleExponentialSmoothing(alpha=alpha)
    forecast_next = model.forecast(monthly_demand)

    print(f"Forecast for next month (Month 13): {forecast_next:.2f}")

```

3. Output Explanation

The model processes the 12 months of historical demand data and calculates a forecast for month 13 using alpha = 0.5. The alpha value controls how quickly the model adapts to new data.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Forecast for next month (Month 13): 183.33
PS C:\Users\ADMIN> & C:/Users/ADMIN/.julia/conda/3/x86_64/python.exe c:/Users/ADMIN/Downloads/ses_model_and_example_combined.py
Historical Demand Data: [120, 135, 150, 145, 160, 155, 170, 165, 180, 175, 190, 185]
Using alpha = 0.5
Forecast for next month (Month 13): 183.33
PS C:\Users\ADMIN>
```