Pricing with Learning of Demand Curves under Inventory Constraint

Primal - Dual Algorithm for 2 Customer types

Main loop 1 and User Inputs - Code Appendix or Variable description:

- **c** = Current total inventory remaining.
- **c0** = Initial inventory, before or at start of booking period
- **customer_data** = List to store customer data
- **customer_id** = 1 = Serial number given to each customer as they aproach irerespective of bought or not. This is new even if a customer approaches more than one time.
- **customers_per_price** = number of customers approaching per price
- **demand_history** = list to store demand values with respect to price for plotting price vs demand for Phase 1.
- **delta** = (δ) parameter used in exploration for narrowing price intervals near optimal price.
- **df** = data frame for demand generated in 1 subphase in Exploration (Phase 1).
- **df_1** = data frame for complete Exploration (Phase 1).
- **df_1** = dataframe to store Phase 1 demand simulation data
- **df_1_type_1** = dataframe to store Phase 1 demand simulation data only for customer type 1.
- **df_1_type_2** = dataframe to store Phase 1 demand simulation data only for customer type 2.
- **K** = Total subphases in phase 1 (K x t = P1)
- **ki** = 1 Subphase in Phase 1 or 1 element in K range
- **num_steps** = Total number of Price points wanted
- **p** = Current price list for type 1 (Reserve Custopmers), initially created after inputs of Highest and Lowest price taken form user.
- **p_1_initial** = Initial price list for customer type 1, this does not change.
- **p_2_initial** = Initial price list for customer type 2, this does not change.

- **p2** = Current price list for type 2 (Preemptive Custopmers), initially created after inputs of Highest and Lowest price taken form user.
- **p_low** = lower price limit when we narrow price intervals towards optimal price.
- **p_opt_1** = p*, optimal price for 1st type (Reserve Customers) in Exploration.
- **p_opt_2** = p*, optimal price for 2nd type (Preemptive Customers) in Exploration.
- **p_opt_type1** = list for collecting optimal prices of type 1 customer in phase 1
- **p_opt_type2** = list for collecting optimal prices of type 2 customer in phase 1
- **p_upp** = upper price limit when we narrow price intervals towards optimal price.
- **pt** = price for a particular instance, pt is in range p.
- **P1** = total booking time of phase 1.
- **P2** = total percbooking time of phase 2, P1+P2 = T, a phase can have many subphases.
- **T** = current total booking tim remaining
- **T0** = Initial booking peroid
- **t** = time instances to divide booking time.
- **total_customers** = 2 x customers_per_price x num_steps (total price points), total customers approaching for a subphase ki in Phase 1. 2 is for 2 customer types.
- **unit_T** = unit of booking peroid (Usually days)
- **xt** = Unit of Booking time T one instance has.
- **z** = dual variable at beginning, can be zero at 1st instance.
- **z_opt** = z*, optimal dual variable for both types (Reserved and Preemptive) customers.

Exploration function (Phase 1) - Code Appendix or Variables description:

- **customer_data** = List to store demand simulation data as customer approaches
- **customer_id** = an serial number order number given to all cusotmers starting from 1 irerespective of bought or not.
- **demand_probability_1** = demand probability calculation for customer type 1 after demand simulation and recording.
- **demand_probability_2** = demand probability calculation for customer type 2 after demand simulation and recording.

- **demand_history** = list to store demand values with respect to price for plotting price vs demand for Phase 1.
- **df** = dataframe inside exploration funciton consisting data for demand simulation for that iterating subphase in phase 1 (exploration)
- **new_price_list_1** = New price list for customer type 1 found after narrowing price intervals via delta
- **new_price_list_2** = New price list for customer type 2 found after narrowing price intervals via delta
- **p_low_1** = new lower price interval found after narrowing with respect to p_opt_1
- **p_low_2** = new lower price interval found after narrowing with respect to p_opt_2
- **p_opt_1** = Optimal price for customer type 1
- **p_opt_2** = Optimal price for customer type 2
- **p_upp_1** = new upper price interval found after narrowing with respect to p_opt_1
- **p_upp_2** = new upper price interval found after narrowing with respect to p_opt_2
- **revenue_per_price_1** = revenue per price for customer type 1
- **revenue_per_price_2** = revenue per price for customer type 2
- **total_revenue** = revenue for both prices. here, revenue is normal revenue i.e. price x units sold.
- **total_revenue_1** = total revenue for customer type 1, here, revenue is normal revenue i.e. price x units sold.
- **total_revenue_2** = total revenue for customer type 2, here, revenue is normal revenue i.e. price x units sold.
- **units_sold** = inventory sold in one subphase of phase one.
- **z_opt** = Combined dual variable value for both customers types
- **z_opt_1** = Dual variable, Optimal unit value of inventory obtained from demand data of customer type 1
- **z_opt_2** = Dual variable, Optimal unit value of inventory obtained from demand data of customer type 2

Main loop 2 - Code Appendix or Variable description:

- **df_2** = dataframe to store Phase 2 demand simulation data.
- **df_2_type_1** = dataframe to store Phase 2 demand simulation data for customer type 1.
- **df_2_type_2** = dataframe to store Phase 2 demand simulation data for customer type 2.

- **safe_T** = Safe booking time limit, T > safe_T ensures booking peroid does not become negative while conducting exploitation's last iteration
- **safe_c** = Safe Inventory, to avoid inventory going (-ve) negative
- **ti** = iteration variale which will be used to see what iteration we are in while doing exploitation phase.

Exploitation function (Phase 2) - Code Appendix or Variables description:

- **alpha_1** = Alpha (bound length for a optimal price) for customer type 1
- **alpha_2** = Alpha (bound length for a optimal price) for customer type 2
- **D_lower_1** = demand probability calculated from df_temp_lower for customer type 1
- **D_lower_2** = demand probability calculated from df_temp_lower for customer type 2
- **D_upper_1** = demand probability calculated from df_temp_upper for customer type 1
- **D_upper_2** = demand probability calculated from df_temp_upper for customer type 2
- **demand_estimate_1** = Estimated demand for optimal price for customer type 1 via llightgbm_buying_probability function.
- **demand_estimate_2** = Estimated demand for optimal price for customer type 2 via llightgbm_buying_probability function.
- **df_exploitation** = data frame to store demand simulation for complete 1 iteration of exploitation.
- **df_temp_lower** = = temporary data frame in exploitation function for storing demand simulation for half instance of lower bound on both customer types.
- **df_temp_upper** = temporary data frame in exploitation function for storing demand simulation for half instance of upper bound on both customer types.
- **p_opt_1_high** = higher bound for Optimal price for customer type 1, p_opt_1 + alpha_1
- **p_opt_1_low** = Lower bound for Optimal price for customer type 1, p_opt_1 - alpha_1
- **p_opt_2_high** = higher bound for Optimal price for customer type 2, p_opt_2 + alpha_2
- **p_opt_2_low** = Lower bound for Optimal price for customer type 2, p_opt_2 - alpha_2
- **theta_1** = Time allocation metric for price bounds for customer type 1.
- **theta_1_lower_time** = time allocated to lower bound of price for customer type 1

- **theta_1_upper_time** = time allocated to upper bound of price for customer type 1
- **theta_2** = = Time allocation metric for price bounds for customer type 2.
- **theta_2_lower_time** = time allocated to lower bound of price for customer type 2
- **theta_2_upper_time** = time allocated to upper bound of price for customer type 2
- **total_estimated_demand** = total of demand_estimate_1 and demand_estimate_2
- **total_future_demand** = Total future demand estimated based on current demand according to current optimal prices.

0. Importing Libraries:

In [145…

```
# importing libraries
import numpy as np
import pandas as pd
from scipy.optimize import minimize_scalar
from scipy.optimize import minimize
import math
import matplotlib.pyplot as plt
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, GridSearchCV
```

1. Phase 1 - Exploration Start:

Below defined are functions used in phase 1's main function i.e. Exploration function, which is defined below these functions.

Functions are defined according to sequence in algorithm flow in phase 1.

1.1 Demand Function:

Returns base probability of purchase:

In [146…

```
##########################################################################
#""" Demand Function - Returns the base probability of purchase at price p "

# (Logistic) - Demand function - type  1: Reserve
def logistic_demand_reserved(pt, p, p0=0,L=1, k=0.001 ):
```

```python
        p0=(p.min()+p.max())/2
        return L / (1 + np.exp(k * (pt - p0)))


# (Logistic) - Demand function - type  2: Preemtive
def logistic_demand_preemptive(pt, p, p0=0,L=1, k=0.04 ):
        p0=(p.min()+p.max())/2
        return L / (1 + np.exp(k * (pt - p0)))




# (Poisson method) - Demand function for Simulation for Exploration (P1)

# Poisson-style Demand for Reserved
def poisson_demand_reserved(pt,  base_lambda=8.0, price_sensitivity=0.0025):
    lambda_ = base_lambda * np.exp(-price_sensitivity * pt)
    prob = 1 - np.exp(-lambda_)
    return np.clip(prob, 0.01, 0.99)

# Poisson-style Demand for Preemptive
def poisson_demand_preemptive(pt, base_lambda=10.0, price_sensitivity=0.009)
    lambda_ = base_lambda * np.exp(-price_sensitivity * pt)
    prob = 1 - np.exp(-lambda_)
    return np.clip(prob, 0.01, 0.99)
```

1.2 Demand Probability function:

Computes probability of buying from demand generated

In [147…

```python
##########################################################################
#""" Compute Demand Probability Function for a price pt -  Dx(p) in Paper -

def compute_demand_probability(df, p):
    """ Calculates demand probabilities from exploration data.
    Returns a local dictionary mapping price to demand probability.
    """

    # List to store demand for different prices
    demand_probability = {}

    for pt in p:
        # Use tolerance to avoid floating point mismatch
        subset = df[np.isclose(df['price'], pt, atol=0.01)]
        if not subset.empty:
            demand_prob = subset['buy'].sum() / subset['buy'].count()
            demand_probability[pt] = demand_prob
        else:
            demand_probability[pt] = 0.0  # fallback if no data at this pric

    # Print demand probabilities for debugging
    for pt in p:
        print(f"Price: {pt:.2f}, Demand Probability: {demand_probability[pt]

    return demand_probability
```

1.3 Revenue Calculation (Normal):

(Normal ***without*** considering ***z_opt***, unit value of inventiory)

normal revenue = price x units sold

In [148…

```
##########################################################################
#""" Compute Revenue for each price for Exploration (P1) """

def calculate_revenue(df):

    """
    Calculates total revenue from exploration sales data
    :param df: DataFrame containing sales data (must have 'price' and 'buy'
    :return: Total revenue, Revenue per price point dictionary
    """

    # Calculate revenue for each transaction
    df['revenue'] = df['price'] * df['buy']

    # Calculate total revenue
    total_revenue = df['revenue'].sum()

    # Calculate revenue per price point
    revenue_per_price = df.groupby('price')['revenue'].sum().to_dict()

    return total_revenue, revenue_per_price
```

1.4 Dual Variable: '***z_opt***' calculation function:

Gives unit value of inventory, after taking into consideration Inventory change, time, prices and demand.

In [149…

```
##########################################################################
# # Dual optimization: Find optimal value i.e. - z_opt in code and z* in pap

"""For bounds from 0 to maximum price"""
""" Minimize_scalar used to optimize"""

def dual_objective(z, demand_probability, p, c):

    # Finding order of magnitude of optimal price respective to the customer
    # Order of Magnitude is needed as the dual variable z* value comes in ve
    # and to make sense of dual variable we multiply it with magnitude of op
    magnitude_p_opt = len(str(int(abs(min(p)))))
    magnitude_order_p_opt=(10**(magnitude_p_opt-1))

    def objective(z):
        return max((pt - z) * demand_probability[pt] for pt in p) + z * c
    z_opt = minimize_scalar(objective, bounds=(0, max(p)), method='bounded')
    return z+((z_opt.x) *(10**5)*(magnitude_order_p_opt))
```

1.5 Optimal Price calculation function:

Calculates optimal price, takes into consideration of unit value of inventory and price at which highest adjusted revenue is generated.

Adjusted Profit Revenue: (price - z_opt) x demand

here, z_opt is unit value of inventory

and demand is probability of buying into customers approached or simply units sold at that price.

In [150…

```
##########################################################################
# Calculate and find optimal price - p* in paper for Exploration (P1)

def exploration_optimal_price (z_opt, demand_probability, p,customers_per_pr

    """
    Finds optimal price p* that maximizes (p - z*) * D_prob.(pt) (demand_pro
    where D_prob.(pt) is the empirical demand probability at price p.

    Parameters:
    - z_opt: optimal dual value (float)
    - demand_probability: dict mapping price -> demand probability
    - p: list or array of price points

    Returns:
    - p_opt: optimal price
    """

    optimal_price = None
    best_value = float('-inf')

    for pt in p:
        demand = demand_probability.get(pt, 0)
        value = (pt - float(z_opt)) * demand
        if value > best_value:
            best_value = value
            optimal_price = pt

    #print(f"*** Optimal Price Calculation ***")
    print(f"Optimal Price (p*): {optimal_price:.2f},\nFor Highest Revenue Va
    return optimal_price
```

1.6 Narrow Price Interval Calculation function:

Narrows down price interval towards optimal price as per delta.

In [151…

```
##########################################################################
# Narrow Price Interval Function for Exploration

def narrow_price_interval(p_opt, n, k, epsilon, p, num_steps, global_min,glo
```

```
    """
    Narrows the price interval around the estimated optimal price using
    the theoretical shrinking factor from the paper.

    Parameters:
    - p_opt: Optimal price estimate from current subphase
    - n: Number of total customers observed so far
    - k: ki = Current subphase index (starting from 0)
    - epsilon: Small constant (e.g., 0.01)
    - p: Current price list (to extract full range)
    - num_steps: Number of price points to generate in next subphase

    Returns:
    - (p_low, p_upp, new_price_list): narrowed interval and refined grid
    """

    # Finding order of magnitude of optimal price respective to the customer
    # Order of Magnitude is needed as the delta value comes to in very small
    # and to make sense of delta we multiply it with order of magnitude of c

    magnitude_p_opt = len(str(int(abs(min(p)))))
    magnitude_order_p_opt=(10**(magnitude_p_opt-1))


    # Compute interval shrinking factor - parameter δ - Delta
    log_n = np.log(n)
    delta = ((n ** ((-0.25) * (1 - ((3 / 5) ** k)))) * (log_n ** (-3 * epsil


    # Generate new interval around p_opt
    p_low = max(min(p), p_opt - delta)
    p_upp = min(max(p), p_opt + delta)

    # Ensure Interval does not Collapse*
    if p_low >= p_upp:
        # Fallback: Use 10% of current interval width
        width = (max(p) - min(p)) * 0.1
        p_low = max(global_min, p_opt - width)
        p_upp = min(global_max, p_opt + width)

    # New refined price grid
    new_price_list = np.linspace(p_low, p_upp, num_steps)

    #print(f"*** Subphase {k+1} Price Interval Update ***")
    print(f"delta: {delta:.5f}")
    print(f"New Interval: [{p_low:.2f}, {p_upp:.2f}]")
    #print(f"New Price List: {new_price_list}")
    #or
    print(f"New Price List: {[int(x) for x in new_price_list]}")


    return p_low, p_upp, new_price_list
```

1.7 Plotting function for Phase 1 i.e. Exploration: ***Price vs Demand***

Plots - Price vs Demand.

For both customer types.

In [152…

```
##########################################################################
# Plot Exploration demand curve

def plot_exploration_demand_curves(history):

    plt.figure(figsize=(24, 6))
    colors = plt.cm.viridis(np.linspace(0, 1, len(history)))  # Color gradie

    # Type 1 Plot
    plt.subplot(1, 2, 1)
    for i, data in enumerate(history):
        plt.plot(data['type1'][0], data['type1'][1],
                 'o-', color=colors[i],
                 label=f'Subphase {data["subphase"]}',
                 alpha=0.8)
    plt.title('Exploration: Reserved Customers (Type 1)\nPrice vs. Demand Pr
    plt.xlabel('Price ($)')
    plt.ylabel('Demand')
    plt.grid(alpha=0.2)
    plt.legend()

    # Type 2 Plot
    plt.subplot(1, 2, 2)
    for i, data in enumerate(history):
        plt.plot(data['type2'][0], data['type2'][1],
                 's--', color=colors[i],
                 label=f'Subphase {data["subphase"]}',
                 alpha=0.8)
    plt.title('Exploration: Preemptive Customers (Type 2)\nPrice vs. Demand 
    plt.xlabel('Price ($)')
    plt.ylabel('Demand')
    plt.grid(alpha=0.2)
    plt.legend()

    plt.tight_layout()
    plt.show()




def plot_demand_curves_from_df(df_type1, df_type2):
    plt.figure(figsize=(24, 6))

    # Helper function to compute demand per price
    def compute_demand(df):
        grouped = df.groupby('price')
        prices = []
```

```
        demands = []
        for price, group in grouped:
            total = len(group)
            bought = group['buy'].sum()
            demand = bought / total if total > 0 else 0
            prices.append(price)
            demands.append(round(demand, 4))
        return prices, demands

    # Type 1 Plot
    prices_1, demands_1 = compute_demand(df_type1)
    plt.subplot(1, 2, 1)
    plt.plot(prices_1, demands_1, 'o-', color='teal', label='Customer Type 1
    plt.title('Exploration: Reserved Customers (Type 1)\nPrice vs. Demand Pr
    plt.xlabel('Price ($)')
    plt.ylabel('Demand')
    plt.grid(alpha=0.2)
    plt.legend()

    # Type 2 Plot
    prices_2, demands_2 = compute_demand(df_type2)
    plt.subplot(1, 2, 2)
    plt.plot(prices_2, demands_2, 's--', color='darkorange', label='Customer
    plt.title('Exploration: Preemptive Customers (Type 2)\nPrice vs. Demand
    plt.xlabel('Price ($)')
    plt.ylabel('Demand')
    plt.grid(alpha=0.2)
    plt.legend()

    plt.tight_layout()
    plt.show()
```

1.8 Exploration Function:

Conducts Phase 1 of Primal - Dual algorithm.

In [153…

```
############################################################################
# Exploration (Phase - 1) Function:

def exploration(c,c0, T, unit_T, t, xt, ki, K, P1, p, p2, z, customers_per_p

    """ Exploration function for performing Phase 1 of Primal - Dual Algorit

    # list to store customer data for a subphase in exploration
    customer_data = []


    # Serial number given tocustomer as they aproach
    if ki==0:
        customer_id = 1
        demand_history.clear()  # Reset history at the start of Phase 1
    else:
```

```
        customer_id = customer_id



    for _ in range(total_customers):
        # Randomly assign customer type: 1 (Reserved) or 2 (Preemptive)
        #currently assigned 50% - 50% for both categories
        cust_type = np.random.choice([1, 2], p=[0.5, 0.5])  # 50% Reserved,

        # Picks price pt **randomly** from appropriate price list p_1_initia
        # Add noise to demand probability
        # buying probability set according to Logistic funciton
        if cust_type == 1:
            pt = np.random.choice(p)
            base_prob = logistic_demand_reserved(pt, p_1_initial)
            noise = np.random.normal(0, 0.06)  # More stable behavior

        else:
            pt = np.random.choice(p2)
            base_prob = logistic_demand_preemptive(pt, p_2_initial)
            noise = np.random.normal(0, 0.09)  # More volatile behavior


        # Use either of two If Else statement pair, Upper^^^ is for Logistic


        # Picks price pt **randomly** from appropriate price list p or p2
        # Add noise to demand probability
        # buying probability set according to Poisson Distribution

        # if cust_type == 1:
        #     pt = np.random.choice(p)
        #     base_prob = poisson_demand_reserved(pt)
        #     noise = np.random.normal(0, 0.025)
        # else:
        #     pt = np.random.choice(p2)
        #     base_prob = poisson_demand_preemptive(pt)
        #     noise = np.random.normal(0, 0.15)


        prob = np.clip(base_prob + noise, 0.01, 0.99)
        buy = int(np.random.rand() < prob)

        # Store data: [customer_id, price offered, bought?, customer_type]
        customer_data.append([customer_id, pt, buy, cust_type])
        customer_id += 1


    # Create DataFrame including customer_type
    df = pd.DataFrame(customer_data, columns=["customer_id", "price", "buy",
    # price rounded to two places
    df["price"] = df["price"].round(2)
    # print(df.head())
    # print(df)
```

```
    # Compute demand probability separately per customer type
    print("\n* Demand Prob. - Type 1: Reserved *")
    demand_probability_1 = compute_demand_probability(df[df['customer_type']

    print("\n* Demand Prob. - Type 2: Preemtive *")
    demand_probability_2 = compute_demand_probability(df[df['customer_type']



    # Revenue calculation separately per customer type
    total_revenue_1, revenue_per_price_1 = calculate_revenue(df[df['customer
    total_revenue_2, revenue_per_price_2 = calculate_revenue(df[df['customer
    total_revenue = total_revenue_1 + total_revenue_2

    print("\n* Revenue Calculation - Customer Type 1 (Reserve) *")
    for pt in p:
        print(f"Price: {pt:.2f}, Revenue: {revenue_per_price_1.get(pt, 0):.2
    print(f"Total Revenue- type 1: {total_revenue_1}")
    print("\n* Revenue Calculation - Customer Type 2 (Preemptive) *")
    for pt in p2:
        print(f"Price: {pt:.2f}, Revenue: {revenue_per_price_2.get(pt, 0):.2
    print(f"Total Revenue- type 2: {total_revenue_2:.2f}")
    print(f"\nTotal Revenue (Both Types): {total_revenue:.2f}")



    # Dual optimization - now needs to consider both customer types separate
    # You can do separate dual variable calculations or a weighted combined
    # For simplicity, run separately and sum the dual objectives or take wei
    z_opt_1 = dual_objective(float(z), demand_probability_1, p, float(c))
    z_opt_2 = dual_objective(float(z), demand_probability_2, p2, float(c))
    # Combine dual variables with weighted average by revenue or units sold;
    z_opt = (z_opt_1 + z_opt_2) / 2
    z_opt = round(z_opt, 2) # rounding to two places
    print(f"\n*** Optimal dual variable (z*) combined: {z_opt:.2f}")




    # Optimal price calculation separately
    print("\n* Type 1: Reserved *")
    p_opt_1 = exploration_optimal_price(z_opt_1, demand_probability_1, p, cu
    p_opt_1 = round(p_opt_1, 2) # rounding price to two places

    print("\n* Type 2: Preemtive *")
    p_opt_2 = exploration_optimal_price(z_opt_2, demand_probability_2, p2, c
    p_opt_2 = round(p_opt_2, 2) # rounding price to two places




    # Narrow price intervals separately
    print("\n\n* Narrowed Price Interval - Type 1: Reserved *")
    p_low_1, p_upp_1, new_price_list_1 = narrow_price_interval(p_opt_1,
                                                              n=len(df[df['
                                                               k=ki, epsilo
```

```
                                                            global_min=m
                                                            global_max=m
    # rounding to two places
    p_low_1, p_upp_1, new_price_list_1 = round(p_low_1, 2), round(p_upp_1, 2



    print("\n* Narrowed Price Interval - Type 2: Preemtive *")
    p_low_2, p_upp_2, new_price_list_2 = narrow_price_interval(p_opt_2,
                                                           n=len(df[df['
                                                            k=ki, epsilo
                                                            global_min=m
                                                            global_max=m
    # rounding to two places
    p_low_2, p_upp_2, new_price_list_2 = round(p_low_2, 2), round(p_upp_2, 2



    # Calculate total units sold from both types
    units_sold = df['buy'].sum()
    c = float(c) - units_sold
    print("\nUnits sold:         ",units_sold)
    print("Remaining inventory:",c)



    # Update Booking peroid T and total instances t
    T = T - (num_steps * xt)
    print("\nRemaining booking period 'T' = ", T, unit_T,)
    t = int(T/xt)
    print("\n-----------------------------------------------------------------------




    # Store complete subphase data (appended at each call)
    demand_history.append({
        'subphase': ki + 1,
        'type1': (p.copy(), list(demand_probability_1.values())),
        'type2': (p2.copy(), list(demand_probability_2.values()))
    })



    if ki == K - 1:
        print("\n\n-------------------------------------------------------------------
        print("\n***** Phase 1 - Exploration finished *****\n")
        print('Final values after Exploration:\n')

        print(f"\nOptimal dual variable:")
        print(f"(z*, unit value of inventory) Combined (Cust type 1 & 2): {z

        print(f"\nOptimal Prices:")
        print(f"Customer type 1 - Reserved =    {p_opt_1:.2f}")
        print(f"Customer type 2 - Preemptive =  {p_opt_2:.2f}")
```

```
        print(f"\nInventory used in phase 1:")
        print(f"Percentage of inventory used: {(((c0-c)/c0)*100):.2f}%")
        print(f"Remaining inventory after sales: {c}, out of total inventory
        print(f"Percentage of inventory remaining: {((c/c0)*100):.2f}%")

        print(f"\nBooking peroid Remaining after phase 1:")
        print(f"Booking peroid remaining: {T:.2f} out of total of {T0} {unit
        print(f"Percentage of booking peroid remaining: {((T/T0)*100):.2f}%"

        print("\n-----------------------------------------------------------
        plot_exploration_demand_curves(demand_history)
        print(f"\nNumber of subphases recorded: {len(demand_history)}\n")
        print("-------------------------------------------------------------


    return df, z_opt, (p_opt_1, p_opt_2), c, T, t, (p_low_1, p_low_2), (p_up
```

1.9 Price List Generation function:

Generates price list according to user input.

In [154…

```
###########################################################################
# Generate price ranges step wise as mentioned by user

# Generate price ranges step wise as mentioned by user for custoemr type 1
def generate_price_list_1(num_steps):
    min_price_1 = float(3000)  #float(input("Enter the minimum price: "))
    max_price_1 = float(7000)  #float(input("Enter the maximum price: "))
    num_steps = num_steps
    price_list = (np.linspace(min_price_1, max_price_1, num_steps))
    return price_list


# Generate price ranges step wise as mentioned by user for custoemr type 2
def generate_price_list_2(num_steps):
    min_price_2 = float(1000)  #float(input("Enter the minimum price: "))
    max_price_2 = float(5000)  #float(input("Enter the maximum price: "))
    num_steps = num_steps
    price_list = (np.linspace(min_price_2, max_price_2, num_steps))
    return price_list
```

1.10 Time allocation function to Phase 1 i.e. Exploration:

Allocates time to phase 1 from total booking time peroid T.

In [155…

```
###########################################################################
# Calculate how much unit of time is allocated to Phase 1

def phase_one_days_calc(num_steps,xt,T):
    xp=int(33)    #int(input("Enter % of Phase 1, between 0 to 100:"))

    if (T*(xp/100)) % (num_steps*xt) == 0:
```

```
        phase = T*(xp/100)
    else:
        phase = (T*(xp/100)) - ((T*(xp/100))% (num_steps*xt))
    return phase
```

1.11 User Input:

Takes input from user about basic given data like: Total inventory c, Total Booking peroid T, etc.

In [156…

```
################################################################################
# Take input from the user
# or
# Give given data to code basically

# HERE some values which are meant to be integer are taken in float because
# BUT they are not trested as float, for example a unit of inventry is not i

c      = float(10000) #float(input("Total Inventory 'c' :   "))
T      = float(100)  #float(input("Total booking peroid 'T' :  "))
unit_T= "days" #input("Enter unit of booking peroid (Usually days):")
t      = float(100)  #float(input("Instances 't' to divide Time peroid T into
xt     = T/t
num_steps = int(5)   #int(input("Enter how many price points you want: "))
p      = generate_price_list_1(num_steps)
p_1_initial = p
p2     = generate_price_list_2(num_steps)
p_2_initial = p2
P1     = phase_one_days_calc(num_steps,xt,T) # here inside fucntion: "phase_o
K      = int (P1/(num_steps*xt))
z      = 00   #input("Enter imitial value of 1 unit of inventory (Can be zero
customers_per_price = 100    #int(input("Enter number of customer approachin
total_customers = 2*customers_per_price*num_steps   #2 for 2 customer types,
c0     = c   # Initial Inventory declared before or at start of booking perio
T0     = T   # Booking peroid set by user, again variable made as T original

customer_data = [] # list to store customer data into data frame df_1 from e
df_1 = pd.DataFrame(customer_data, columns=["customer_id", "price", "buy", "

demand_history = [] # list to store demand values with respect to price for

p_opt_type1 = [] #list for collecting optimal prices of type 1 customer
p_opt_type2 = [] #list for collecting optimal prices of type 2 customer

# Serial number given to each customer as they approach
customer_id = 1

print("*** USER INPUTS ***")
print("\nTotal Inventory c:", c)
print("\nTotal booking peroid T:", T,unit_T)
print("\nTotal Instances t:", t)
```

```python
print(f"\n1 time Instance from t: xt = {xt:.2f} {unit_T}")
print("\nGenerated prices for Reserved Cust (Type 1):", p)
print("\nGenerated prices for Preemptive Cust (Type 2):", p2)
print(f"\nTotal time allocated to Phase 1: {P1:.2f}")
print("\nTotal subphases 'K' in Phase 1: ",K, '\n')
```

```
*** USER INPUTS ***

Total Inventory c: 10000.0

Total booking peroid T: 100.0 days

Total Instances t: 100.0

1 time Instance from t: xt = 1.00 days

Generated prices for Reserved Cust (Type 1): [3000. 4000. 5000. 6000. 7000.]

Generated prices for Preemptive Cust (Type 2): [1000. 2000. 3000. 4000. 500
0.]

Total time allocated to Phase 1: 30.00

Total subphases 'K' in Phase 1:  6
```

1.12 Main Loop 1: Conduct Exploration

This loop runs until all subphases in phase 1 are completed.

It calls exploration funciton.

There are two main loops, first is for conducting phase 1 i.e. exploration and second is for conducting phase 2 i.e. exploitation.

Loop for exploitation i.e. phase 2 is way below in phase 2 part of notebook after exploitation function.

In [157…

```python
##########################################################################
# Conduct Phase 1   i.e.   ***Exploration***: Call the function with user inp
# Or
# Phase 1 - Loop

print("\n","***Expoloration Start - Detailed Data:***")


# initializing variable ki to loop in total subphases in Phase 1
ki = 0

for ki in range(K):
```

```
    if c>0: # Inventory should exist  before sale

        print("\n","Phase 1 (P1), Subphase (ki): ", ki+1,"\n")

        df, z_opt, p_opt_tuple, c, T, t, p_low_tuple, p_upp_tuple, updated_p











        # Dataframe update for exploration - Phase 1
        df_1 = pd.concat([df_1, df], ignore_index=True)
        df_1["price"] = df_1["price"].round(2) # rounding price


        # Update price lists for next iteration
        p = updated_p_tuple[0]
        p2 = updated_p_tuple[1]

        # Seprating and storing Optimal prices in a list
        p_opt_type1.append(p_opt_tuple[0])
        p_opt_type2.append(p_opt_tuple[1])

    else:
        print("*** INSUFFICIENT INVENTORY ***")



# Save the Phase 1 dataframe to a CSV file
df_1.to_csv("all_output.csv", index=False)
print("Data from df_1 (Phase 1 - Exploration) appended and saved to all_outp
print("---------------------------------------------------------------------------


# Seprate Data frames for different customer types for phase 1
# Will be used later in the later stages like while estimating demand fucnti
df_1_type_1 = df_1[df_1['customer_type'] == 1].copy()
df_1_type_2 = df_1[df_1['customer_type'] == 2].copy()

# Plotting Price vs demand for Phase 1 (Exploration) for both customer types
plot_demand_curves_from_df(df_1_type_1, df_1_type_2)

###########################################################################
```

```
 ***Expoloration Start - Detailed Data:***

 Phase 1 (P1), Subphase (ki):  1


* Demand Prob. - Type 1: Reserved *
Price: 3000.00, Demand Probability: 0.86
Price: 4000.00, Demand Probability: 0.73
Price: 5000.00, Demand Probability: 0.53
Price: 6000.00, Demand Probability: 0.28
Price: 7000.00, Demand Probability: 0.11

* Demand Prob. - Type 2: Preemtive *
Price: 1000.00, Demand Probability: 0.94
Price: 2000.00, Demand Probability: 0.95
Price: 3000.00, Demand Probability: 0.53
Price: 4000.00, Demand Probability: 0.04
Price: 5000.00, Demand Probability: 0.03

* Revenue Calculation - Customer Type 1 (Reserve) *
Price: 3000.00, Revenue: 252000.00
Price: 4000.00, Revenue: 244000.00
Price: 5000.00, Revenue: 270000.00
Price: 6000.00, Revenue: 180000.00
Price: 7000.00, Revenue: 70000.00
Total Revenue- type 1: 1016000.0

* Revenue Calculation - Customer Type 2 (Preemptive) *
Price: 1000.00, Revenue: 96000.00
Price: 2000.00, Revenue: 192000.00
Price: 3000.00, Revenue: 171000.00
Price: 4000.00, Revenue: 16000.00
Price: 5000.00, Revenue: 15000.00
Total Revenue- type 2: 490000.00

Total Revenue (Both Types): 1506000.00

*** Optimal dual variable (z*) combined: 445.05

* Type 1: Reserved *
Optimal Price (p*): 4000.00,
For Highest Revenue Value: 262240.44

* Type 2: Preemtive *
Optimal Price (p*): 2000.00,
For Highest Revenue Value: 142451.90


* Narrowed Price Interval - Type 1: Reserved *
delta: 946.81679
New Interval: [3053.18, 4946.82]
New Price List: [3053, 3526, 4000, 4473, 4946]

* Narrowed Price Interval - Type 2: Preemtive *
delta: 946.52419
New Interval: [1053.48, 2946.52]
```

```
New Price List: [1053, 1526, 2000, 2473, 2946]

Units sold:          495
Remaining inventory: 9505.0

Remaining booking period 'T' =  95.0 days

----------------------------------------------------------------------------
-

 Phase 1 (P1), Subphase (ki):  2


* Demand Prob. - Type 1: Reserved *
Price: 3053.18, Demand Probability: 0.85
Price: 3526.59, Demand Probability: 0.83
Price: 4000.00, Demand Probability: 0.67
Price: 4473.41, Demand Probability: 0.69
Price: 4946.82, Demand Probability: 0.54

* Demand Prob. - Type 2: Preemtive *
Price: 1053.48, Demand Probability: 0.98
Price: 1526.74, Demand Probability: 0.95
Price: 2000.00, Demand Probability: 0.98
Price: 2473.26, Demand Probability: 0.97
Price: 2946.52, Demand Probability: 0.89

* Revenue Calculation - Customer Type 1 (Reserve) *
Price: 3053.18, Revenue: 262573.48
Price: 3526.59, Revenue: 296233.56
Price: 4000.00, Revenue: 240000.00
Price: 4473.41, Revenue: 339979.16
Price: 4946.82, Revenue: 267128.28
Total Revenue- type 1: 1405914.48

* Revenue Calculation - Customer Type 2 (Preemptive) *
Price: 1053.48, Revenue: 106401.48
Price: 1526.74, Revenue: 143513.56
Price: 2000.00, Revenue: 194000.00
Price: 2473.26, Revenue: 237432.96
Price: 2946.52, Revenue: 256347.24
Total Revenue- type 2: 937695.24

Total Revenue (Both Types): 2343609.72

*** Optimal dual variable (z*) combined: 477.45

* Type 1: Reserved *
Optimal Price (p*): 4473.41,
For Highest Revenue Value: 275050.89

* Type 2: Preemtive *
Optimal Price (p*): 2946.52,
For Highest Revenue Value: 220521.12
```

```
* Narrowed Price Interval - Type 1: Reserved *
delta: 508.29852
New Interval: [3965.11, 4946.82]
New Price List: [3965, 4210, 4455, 4701, 4946]

* Narrowed Price Interval - Type 2: Preemtive *
delta: 508.72497
New Interval: [2437.80, 2946.52]
New Price List: [2437, 2564, 2692, 2819, 2946]

Units sold:          835
Remaining inventory: 8670.0

Remaining booking period 'T' =  90.0 days

----------------------------------------------------------------------------
-

 Phase 1 (P1), Subphase (ki):  3
```

```
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2538179192.py:32: FutureWa
rning: The behavior of DataFrame concatenation with empty or all-NA entries
is deprecated. In a future version, this will no longer exclude empty or all
-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  df_1 = pd.concat([df_1, df], ignore_index=True)
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
```

```
* Demand Prob. - Type 1: Reserved *
Price: 3965.11, Demand Probability: 0.73
Price: 4210.54, Demand Probability: 0.70
Price: 4455.97, Demand Probability: 0.65
Price: 4701.39, Demand Probability: 0.62
Price: 4946.82, Demand Probability: 0.49

* Demand Prob. - Type 2: Preemtive *
Price: 2437.80, Demand Probability: 0.94
Price: 2564.98, Demand Probability: 0.98
Price: 2692.16, Demand Probability: 0.95
Price: 2819.34, Demand Probability: 0.96
Price: 2946.52, Demand Probability: 0.87

* Revenue Calculation - Customer Type 1 (Reserve) *
Price: 3965.11, Revenue: 222046.16
Price: 4210.54, Revenue: 311579.96
Price: 4455.97, Revenue: 303005.96
Price: 4701.39, Revenue: 258576.45
Price: 4946.82, Revenue: 252287.82
Total Revenue- type 1: 1347496.35

* Revenue Calculation - Customer Type 2 (Preemptive) *
Price: 2437.80, Revenue: 241342.20
Price: 2564.98, Revenue: 235978.16
Price: 2692.16, Revenue: 285368.96
Price: 2819.34, Revenue: 279114.66
Price: 2946.52, Revenue: 268133.32
Total Revenue- type 2: 1309937.30

Total Revenue (Both Types): 2657433.65

*** Optimal dual variable (z*) combined: 477.45

* Type 1: Reserved *
Optimal Price (p*): 4701.39,
For Highest Revenue Value: 260105.49

* Type 2: Preemtive *
Optimal Price (p*): 2819.34,
For Highest Revenue Value: 226532.28


* Narrowed Price Interval - Type 1: Reserved *
delta: 352.36045
New Interval: [4349.03, 4946.82]
New Price List: [4349, 4498, 4647, 4797, 4946]

* Narrowed Price Interval - Type 2: Preemtive *
delta: 348.20170
New Interval: [2471.14, 2946.52]
New Price List: [2471, 2589, 2708, 2827, 2946]

Units sold:          791
Remaining inventory: 7879.0
```

```
Remaining booking period 'T' =  85.0 days

---------------------------------------------------------------------------
-

 Phase 1 (P1), Subphase (ki):  4


* Demand Prob. - Type 1: Reserved *
Price: 4349.03, Demand Probability: 0.62
Price: 4498.48, Demand Probability: 0.65
Price: 4647.92, Demand Probability: 0.55
Price: 4797.37, Demand Probability: 0.44
Price: 4946.82, Demand Probability: 0.49

* Demand Prob. - Type 2: Preemtive *
Price: 2471.14, Demand Probability: 0.96
Price: 2589.98, Demand Probability: 0.93
Price: 2708.83, Demand Probability: 1.00
Price: 2827.67, Demand Probability: 0.97
Price: 2946.52, Demand Probability: 0.92

* Revenue Calculation - Customer Type 1 (Reserve) *
Price: 4349.03, Revenue: 260941.80
Price: 4498.48, Revenue: 274407.28
Price: 4647.92, Revenue: 264931.44
Price: 4797.37, Revenue: 182300.06
Price: 4946.82, Revenue: 262181.46
Total Revenue- type 1: 1244762.04

* Revenue Calculation - Customer Type 2 (Preemptive) *
Price: 2471.14, Revenue: 224873.74
Price: 2589.98, Revenue: 256408.02
Price: 2708.83, Revenue: 276300.66
Price: 2827.67, Revenue: 294077.68
Price: 2946.52, Revenue: 276972.88
Total Revenue- type 2: 1328632.98

Total Revenue (Both Types): 2573395.02

*** Optimal dual variable (z*) combined: 477.45

* Type 1: Reserved *
Optimal Price (p*): 4498.48,
For Highest Revenue Value: 259968.36

* Type 2: Preemtive *
Optimal Price (p*): 2827.67,
For Highest Revenue Value: 229886.75


* Narrowed Price Interval - Type 1: Reserved *
delta: 281.39627
New Interval: [4349.03, 4779.88]
New Price List: [4349, 4456, 4564, 4672, 4779]
```

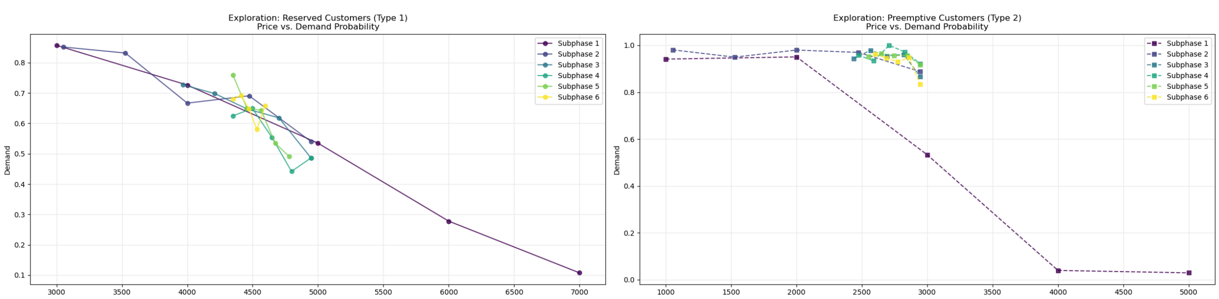
```
* Narrowed Price Interval - Type 2: Preemtive *
delta: 278.69620
New Interval: [2548.97, 2946.52]
New Price List: [2548, 2648, 2747, 2847, 2946]

Units sold:          759
Remaining inventory: 7120.0

Remaining booking period 'T' =  80.0 days

-----------------------------------------------------------------------------
-

 Phase 1 (P1), Subphase (ki):  5


* Demand Prob. - Type 1: Reserved *
Price: 4349.03, Demand Probability: 0.76
Price: 4456.74, Demand Probability: 0.65
Price: 4564.45, Demand Probability: 0.64
Price: 4672.16, Demand Probability: 0.53
Price: 4779.88, Demand Probability: 0.49

* Demand Prob. - Type 2: Preemtive *
Price: 2548.97, Demand Probability: 0.95
Price: 2648.36, Demand Probability: 0.96
Price: 2747.75, Demand Probability: 0.96
Price: 2847.13, Demand Probability: 0.96
Price: 2946.52, Demand Probability: 0.92

* Revenue Calculation - Customer Type 1 (Reserve) *
Price: 4349.03, Revenue: 356620.46
Price: 4456.74, Revenue: 280774.62
Price: 4564.45, Revenue: 296689.25
Price: 4672.16, Revenue: 214919.36
Price: 4779.88, Revenue: 248553.76
Total Revenue- type 1: 1397557.45

* Revenue Calculation - Customer Type 2 (Preemptive) *
Price: 2548.97, Revenue: 203917.60
Price: 2648.36, Revenue: 291319.60
Price: 2747.75, Revenue: 241802.00
Price: 2847.13, Revenue: 316031.43
Price: 2946.52, Revenue: 259293.76
Total Revenue- type 2: 1312364.39

Total Revenue (Both Types): 2709921.84

*** Optimal dual variable (z*) combined: 463.52

* Type 1: Reserved *
Optimal Price (p*): 4349.03,
For Highest Revenue Value: 294933.20

* Type 2: Preemtive *
Optimal Price (p*): 2847.13,
```

```
For Highest Revenue Value: 228185.47


* Narrowed Price Interval - Type 1: Reserved *
delta: 245.06929
New Interval: [4349.03, 4594.10]
New Price List: [4349, 4410, 4471, 4532, 4594]

* Narrowed Price Interval - Type 2: Preemtive *
delta: 244.63359
New Interval: [2602.50, 2946.52]
New Price List: [2602, 2688, 2774, 2860, 2946]

Units sold:          785
Remaining inventory: 6335.0

Remaining booking period 'T' =  75.0 days

----------------------------------------------------------------------------
-

 Phase 1 (P1), Subphase (ki):  6
```

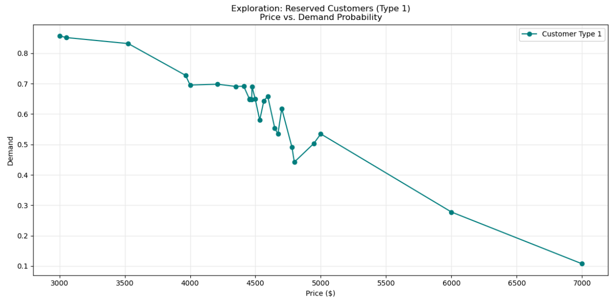
```
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\2180988554.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['revenue'] = df['price'] * df['buy']
```

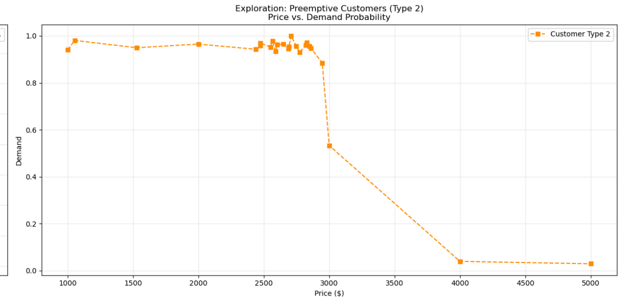
```
* Demand Prob. - Type 1: Reserved *
Price: 4349.03, Demand Probability: 0.68
Price: 4410.30, Demand Probability: 0.69
Price: 4471.56, Demand Probability: 0.65
Price: 4532.83, Demand Probability: 0.58
Price: 4594.10, Demand Probability: 0.66

* Demand Prob. - Type 2: Preemtive *
Price: 2602.50, Demand Probability: 0.96
Price: 2688.50, Demand Probability: 0.94
Price: 2774.51, Demand Probability: 0.93
Price: 2860.51, Demand Probability: 0.95
Price: 2946.52, Demand Probability: 0.83

* Revenue Calculation - Customer Type 1 (Reserve) *
Price: 4349.03, Revenue: 295734.04
Price: 4410.30, Revenue: 286669.50
Price: 4471.56, Revenue: 254878.92
Price: 4532.83, Revenue: 262904.14
Price: 4594.10, Revenue: 257269.60
Total Revenue- type 1: 1357456.2

* Revenue Calculation - Customer Type 2 (Preemptive) *
Price: 2602.50, Revenue: 270660.00
Price: 2688.50, Revenue: 276915.50
Price: 2774.51, Revenue: 258029.43
Price: 2860.51, Revenue: 308935.08
Price: 2946.52, Revenue: 250454.20
Total Revenue- type 2: 1364994.21

Total Revenue (Both Types): 2722450.41

*** Optimal dual variable (z*) combined: 448.01

* Type 1: Reserved *
Optimal Price (p*): 4410.30,
For Highest Revenue Value: 274989.22

* Type 2: Preemtive *
Optimal Price (p*): 2860.51,
For Highest Revenue Value: 227180.93


* Narrowed Price Interval - Type 1: Reserved *
delta: 229.56402
New Interval: [4349.03, 4594.10]
New Price List: [4349, 4410, 4471, 4532, 4594]

* Narrowed Price Interval - Type 2: Preemtive *
delta: 222.53076
New Interval: [2637.98, 2946.52]
New Price List: [2637, 2715, 2792, 2869, 2946]

Units sold:          797
Remaining inventory: 5538.0
```

```
Remaining booking period 'T' =  70.0 days

----------------------------------------------------------------------------
-


----------------------------------------------------------------------------
-

***** Phase 1 - Exploration finished *****

Final values after Exploration:


Optimal dual variable:
(z*, unit value of inventory) Combined (Cust type 1 & 2): 448.01

Optimal Prices:
Customer type 1 - Reserved =    4410.30
Customer type 2 - Preemptive =  2860.51

Inventory used in phase 1:
Percentage of inventory used: 44.62%
Remaining inventory after sales: 5538.0, out of total inventory 10000.0 unit
s.
Percentage of inventory remaining: 55.38%

Booking peroid Remaining after phase 1:
Booking peroid remaining: 70.00 out of total of 100.0 days
Percentage of booking peroid remaining: 70.00%

----------------------------------------------------------------------------
-
```




```
Number of subphases recorded: 6

----------------------------------------------------------------------------
------
Data from df_1 (Phase 1 - Exploration) appended and saved to all_output.csv
----------------------------------------------------------------------------
-------------------
```





2.0 Phase 2 - Exploitation Start:

2.1 Demand Estimation fucntion:

Estimates the demand for input price to function, used in exploitation function to compare demand and remaining inventory.

Exploitation function is next after this.

There are two models Random forests and Lightgmb.

These models can estimate demand for a price even with unknown relationship for a particular price and it' demand.

Known relationships example is Linear, polynomial, logistic, etc. But in our case we don't know relationship.

Lightgmb is way more fast computational efficient than random forests,

Both models give a avg. accuracy around 70%, which is not good, but when we can increase more features (dependent variables), we can get more accuracy, features like remaining time peroid and remaining inventory can be added.

Random forests takes around 10 to 12 minutes to run.

Where as Light gmb only takes a few seconds to run (Less than a minute).

We have kept both models and currently using Lightgmb by calling in Exploitation function.

Both models are defined below.

In [158…

```
#########################################################################
# Calculate buying probability based on demand generated in exploration
# there are two models Random forests and Lightgmb
# Light gmb is way more computational efficient than random forests,
# both models give a accuracy of 70%, which is not good, but when we increas
# features like remaining time peroid and remaining inventory can be added

# Random forests takes around 12 minutes to run,
# Where as Light gmb only takes a few seconds to run.
# We have introduced both models and currently using Light gmb in Exploitati
# Both models are defined below
#
#

#########################################################################
### This demand estimation is done by random forest
def random_forest_buying_probability(df, price_value):
    """
    Trains a Random Forest classifier and predicts buying probability for a
    Prints model accuracy on training data.

    Parameters:
    df (DataFrame): Filtered DataFrame for customer_type == 1
    price_value (int or float): Target price for prediction

    Returns:
    float: Predicted probability of buying
    """
    df_clean = df.copy()
    X = df_clean[['price']]
    y = df_clean['buy'].astype(int)

    # Split your data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

    # Define hyperparameter grid
    param_grid = {
        'n_estimators': [100, 200, 500],
        'max_depth': [None, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'max_features': ['sqrt', 'log2']
    }

    # Set up GridSearchCV
    grid_search = GridSearchCV(
        RandomForestClassifier(random_state=42),
        param_grid,
        cv=5,                     # 5-fold cross-validation
        scoring='accuracy',
        n_jobs=-1
    )

    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_
```

```
    # Evaluate accuracy on test set
    test_predictions = best_model.predict(X_test)
    test_acc = accuracy_score(y_test, test_predictions)
    print(f"\n############## Tuned Random Forest test accuracy: {round(test
    # print("Best parameters found:", grid_search.best_params_)

    # Predict probability for input price
    predicted_prob = best_model.predict_proba([[price_value]])[0][1]

    #safeguard to ensure demand probability stays between 0 and 1
    predicted_prob = max(0, min(1, predicted_prob))

    return round(predicted_prob, 2)




##############################################################################
### This demand estimation is done by lightgbm / xgboost
### More efficient than random forest
def lightgbm_buying_probability(df, price_value):
    df_clean = df.copy()
    X = df_clean[['price']]
    y = df_clean['buy'].astype(int)

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

    # Create LightGBM datasets
    train_data = lgb.Dataset(X_train, label=y_train)
    test_data = lgb.Dataset(X_test, label=y_test, reference=train_data)

    # Set parameters
    params = {
        'objective': 'binary',
        'metric': 'binary_logloss',
        'verbosity': -1,
        'boosting_type': 'gbdt',
        'learning_rate': 0.1,
        'num_leaves': 15,
        'max_depth': 5,
        'seed': 42
    }

    # Train with early stopping
    model = lgb.train(
        params,
        train_data,
        num_boost_round=100,
        valid_sets=[test_data],
        valid_names=['valid'],
        callbacks=[lgb.early_stopping(stopping_rounds=10)]
    )

    # Predict on test set for accuracy
```

```
    test_preds = model.predict(X_test)
    test_preds_binary = [1 if p > 0.5 else 0 for p in test_preds]
    test_acc = accuracy_score(y_test, test_preds_binary)
    print(f"\n############### Tuned Light Gmb test accuracy: {round(test_acc

    # Predict probability for input price
    predicted_prob = model.predict([[price_value]])[0]

    #safeguard to ensure demand probability stays between 0 and 1
    predicted_prob = max(0, min(1, predicted_prob))

    return round(predicted_prob, 2)




##########################################################################
    # after adding more features like remaining inventory and time peroid, y
    # this way, by changing dependent variables in model above.
    #
    # X = df_clean[['price', 'customer_type', 'remaining_time', 'remaining_i
    # df_clean['price_squared'] = df_clean['price'] ** 2
    # df_clean['price_cust_interaction'] = df_clean['price'] * df_clean['cus
```

2.2 Alpha (α) and Price bounds calculation function:

Calculates alpha needed for calculating lower and upper bounds.

Lower bound = Optimal price - alpha

Upper bound = Optimal price + alpha

In [159…
```
def calculate_alpha_and_bounds (p_opt,p):

        # defying variables for alpha, according to which prices are changed
        n = customers_per_price
        log_n = np.log(n)
        epsilon = 0.01


        # Calculating alpha according to price order of magnitude to make se
        magnitude_p_opt = len(str(int(abs(p_opt))))
        magnitude_order_p_opt = (10**(magnitude_p_opt-2))

        alpha = (log_n**(1+(9*epsilon)))*(n**(-1/4))*magnitude_order_p_opt
        alpha = round(alpha, 2)


        # Calculate Lower and upper bounds of prices for Cust type 1. accord
        p_opt_low =   max(min(p), p_opt - alpha)
        p_opt_high =  min(max(p), p_opt + alpha)
```

```
        return  alpha, p_opt_low, p_opt_high
```

2.3 Time Allocation, Theta (θ) function:

When Inventory is insufficient compared to estiamted future demand based on current optimal price in exploitatoin,

Theta is calculated based on demand recorded on lower and upper bound of price, remaining inventory and remaining booking peroid.

Theta ranges from [0,1], which tells us how much time to give to lower bound price and upper bound price from next instance.

In [160…

```
def compute_theta(c, T, D_lower, D_upper, p_opt):

    # Avoid division by zero with a small epsilon
    epsilon = 1e-8
    denominator = D_lower - D_upper

    # If demand for upper and lower bounds is nearly equal, assume theta as
    if abs(denominator) < epsilon:
        return 0.5

    # we multiply theta by price magnitude, this is done after thorough obse
    magnitude_c = len(str(int(abs(c))))
    magnitude_order_c = (10**(magnitude_c))

    raw_theta = (((c / T) - D_upper) / (D_lower - D_upper)) / magnitude_orde
    print(f"raw_theta: {raw_theta:.2f}")

    # If theta is Infinity:
    if math.isinf(raw_theta):
        if raw_theta > 0:
            theta = 1.0
        else:
            theta = 0.0

    # If theta is Nan:
    elif math.isnan(raw_theta):
        theta = 0.0

    # Safe guard Clamp and abs, ensures theta styas between 0 and 1.
    theta = max(0.0, min(1.0, abs(raw_theta)))


    return round(theta, 2)
```

2.4 Exploitation function:

Conducts Phase 2 i.e. Exploitation of Primal dual algorithm.

In [161…

```
##########################################################################
# Exploitation Function:

def exploitation(c,T, p_opt_1, p_opt_2, z_opt,p,p2, customer_id, xt, p_1_ini


    # list to store customer data for a subphase in exploration
    customer_data = []
    # Serial number given tocustomer as they aproach
    customer_id = customer_id




    # Estimating demand for each customer type whcih will be summed up and u

    demand_estimate_1 = lightgbm_buying_probability(df_1_type_1, p_opt_1)
    demand_estimate_2 = lightgbm_buying_probability(df_1_type_2, p_opt_2)
    total_estimated_demand =  demand_estimate_1 + demand_estimate_2

    total_future_demand = total_estimated_demand*customers_per_price*(T*xt)
    # Here,
    # we are multiplying demand for that price on that day into total remair




    # Raising iteration number by one from previous count for this iteratior
    ti = ti+1

    # Tells user in Output, exploitation for what time is carried out of tot
    print ("\n##############################################################
    print (ti,"(Tth) Iteration running in Exploitation:")
    print (f"Booking time remaining: {T:.2f} {unit_T} out of {T0} {unit_T} r

    print(f"\nCust type 1 (Reserved) Estimated demand at price {p_opt_1:.2f}
    print(f"Cust type 2 (Preemptive) Estimated demand at price {p_opt_2:.2f}
    print(f"Total Future demand: {total_future_demand:.2f}")
    print(f"total Inventory Remaining: {c:.0f}")




    ######################################################################
    # Case 1 - Sufficient Inventory
    if total_future_demand < c or total_future_demand == c :

        # Print in output to know while checking that we are in Case 1: Suff
        print("\nEntered Into Case 1: Sufficient Inventory")

        p_opt_1 = p_opt_1
```

```
        p_opt_2 = p_opt_2


        for _ in range(2*customers_per_price):
            # Randomly assign customer type: 1 (Reserved) or 2 (Preemptive)
            cust_type = np.random.choice([1, 2], p=[0.5, 0.5])  # 50% Reserv

            # buying probability set according to Logistic funciton
            if cust_type == 1:
                base_prob = logistic_demand_reserved(p_opt_1, p_1_initial)
                noise = np.random.normal(0, 0.0258)  # More stable behavior
                p_opt = p_opt_1
            else:
                base_prob = logistic_demand_preemptive(p_opt_2, p_2_initial)
                noise = np.random.normal(0, 0.258)  # More volatile behavior
                p_opt = p_opt_2

            prob = np.clip(base_prob + noise, 0.01, 0.99)
            buy = int(np.random.rand() < prob)

            # Store data: [customer_id, price offered, bought?, customer_typ
            customer_data.append([customer_id, p_opt, buy, cust_type])
            customer_id += 1

        # update Booking peroid
        T= T-(xt)

        print(f"\nOptimal Price not updated - Customer type 1 (Reserved):
        print(f"Optimal Price not updated - Customer type 2 (Preemptive): {p




    ########################################################################
    # Case 2 - Insufficient Inventory
    else:

        # Print in output to know while checking that we are in Case2: Insuf
        print("\nEntered Into Case 2: Insufficient Inventory")



        # Caculate Alpha, and Lower and upper bounds for prices

        alpha_1,p_opt_1_low, p_opt_1_high = calculate_alpha_and_bounds(p_opt
        print('\nAlpha for cust. type 1:',alpha_1)

        alpha_2,p_opt_2_low, p_opt_2_high = calculate_alpha_and_bounds(p_opt
        print('Alpha for cust. type 2:',alpha_2,'\n')



        ########################################################################
        # Sell for 1st half of instance with upper bound of prices for both

        for _ in range(customers_per_price):
```

```
        # Randomly assign customer type: 1 (Reserved) or 2 (Preemptive)
        cust_type = np.random.choice([1, 2], p=[0.5, 0.5])  # 50% Reserv

        # buying probability set according to Logistic funciton
        if cust_type == 1:
            base_prob = logistic_demand_reserved(p_opt_1_high, p_1_initi
            noise = np.random.normal(0, 0.0258)  # More stable behavior
            p_opt = p_opt_1_high
        else:
            base_prob = logistic_demand_preemptive(p_opt_2_high, p_2_ini
            noise = np.random.normal(0, 0.258)  # More volatile behavior
            p_opt = p_opt_2_high

        prob = np.clip(base_prob + noise, 0.01, 0.99)
        buy = int(np.random.rand() < prob)

        # Store data: [customer_id, price offered, bought?, customer_typ
        customer_data.append([customer_id, p_opt, buy, cust_type])
        customer_id += 1

    # Recording temporary data for demand calculation for upper bound
    df_temp_upper = pd.DataFrame(customer_data, columns=["customer_id",

    # Filter each customer type
    df_upper_1 = df_temp_upper[df_temp_upper['customer_type'] == 1]
    df_upper_2 = df_temp_upper[df_temp_upper['customer_type'] == 2]

    # Compute demand as average buy rate, used later for calculating The
    D_upper_1 = df_upper_1['buy'].mean()
    D_upper_2 = df_upper_2['buy'].mean()

    # Round if needed
    D_upper_1 = round(D_upper_1, 2)
    D_upper_2 = round(D_upper_2, 2)

    # update Booking peroid
    T= T-(xt/2)



    ####################################################################
    # Sell for 2nd half of instance with lower  bound of prices for both

    for _ in range(customers_per_price):
        # Randomly assign customer type: 1 (Reserved) or 2 (Preemptive)
        cust_type = np.random.choice([1, 2], p=[0.5, 0.5])  # 50% Reserv

        # buying probability set according to Logistic funciton
        if cust_type == 1:
            base_prob = logistic_demand_reserved(p_opt_1_low, p_1_initia
            noise = np.random.normal(0, 0.0258)  # More stable behavior
            p_opt = p_opt_1_low
        else:
            base_prob = logistic_demand_preemptive(p_opt_2_low, p_2_init
            noise = np.random.normal(0, 0.258)  # More volatile behavior
            p_opt = p_opt_2_low
```

```
            prob = np.clip(base_prob + noise, 0.01, 0.99)
            buy = int(np.random.rand() < prob)

            # Store data: [customer_id, price offered, bought?, customer_typ
            customer_data.append([customer_id, p_opt, buy, cust_type])
            customer_id += 1

        # Recording temporary data for demand calculation for lower bound
        df_temp_lower = pd.DataFrame(customer_data, columns=["customer_id",

        # Filter each customer type
        df_lower_1 = df_temp_lower[df_temp_lower['customer_type'] == 1]
        df_lower_2 = df_temp_lower[df_temp_lower['customer_type'] == 2]

        # Compute demand as average buy rate, used later for calculating The
        D_lower_1 = df_lower_1['buy'].mean()
        D_lower_2 = df_lower_2['buy'].mean()

        # Round if needed
        D_lower_1 = round(D_lower_1, 2)
        D_lower_2 = round(D_lower_2, 2)

        # update Booking peroid
        T= T-(xt/2)


        ##################################################################
        # this block only for debugging, to check the demand comparison of b

        ## Cust type 1 - Reserved:
        if D_upper_1 > D_lower_1:
            print ("Cust type 1 - Reserved: Upper bound price has High demar
        elif D_upper_1 < D_lower_1:
            print ("Cust type 1 - Reserved: Lower bound price has High demar
        else:
            print("Cust type 1 - Reserved: Both upper and lower bound demanc


        ## Cust type 2 - Preemptive:
        if D_upper_2 > D_lower_2:
            print ("Cust type 2 - Preemptive: Upper bound price has High dem
        elif D_upper_2 < D_lower_2:
            print ("Cust type 2 - Preemptive: Lower bound price has High dem
        else:
            print("Cust type 2 - Preemptive: Both upper and lower bound dema


        ##################################################################
        # Calculate Theta (θ) - Time allocation for next instance for Upper
        # And allocating time to upper and lower bound for next 2 instances


        ## for Cust. type 1 - Reserved ###################################
        theta_1 = compute_theta(c, T, D_lower_1, D_upper_1, p_opt_1)
        print("Theta cust type 1 - Reserved:    ", theta_1)
```

```
        # allocating time according to calculated theta to upper and lower b
        # here data type needs to be int as customers are Whole numbers, car
        theta_1_lower_time = int (theta_1*customers_per_price)
        theta_1_upper_time = int ((1-theta_1)*customers_per_price)



        ## for Cust. type 2 - Preemptive ####################################
        theta_2 = compute_theta(c, T, D_lower_2, D_upper_2, p_opt_2)
        print("Theta cust type 2 - Preemptive:  ", theta_2)

        # allocating time according to calculated theta to upper and lower b
        # here data type needs to be int as customers are Whole numbers, car
        theta_2_lower_time = int (theta_2*customers_per_price)
        theta_2_upper_time = int ((1-theta_2)*customers_per_price)
        ####################################################################



        ####################################################################
        # Selling for cust type 1 according to theta 1 for upper bound
        for _ in range(theta_1_upper_time):
            cust_type = 1
            base_prob = logistic_demand_reserved(p_opt_1_high, p_1_initial)
            noise = np.random.normal(0, 0.0258)  # More stable behavior
            p_opt = p_opt_1_high

            prob = np.clip(base_prob + noise, 0.01, 0.99)
            buy = int(np.random.rand() < prob)

            # Store data: [customer_id, price offered, bought?, customer_typ
            customer_data.append([customer_id, p_opt, buy, cust_type])
            customer_id += 1

        # update Booking peroid
        T= T-(xt/2)

        ####################################################################
        # Selling for cust type 1 according to theta 1 for lower bound
        for _ in range(theta_1_lower_time):
            cust_type = 1
            base_prob = logistic_demand_reserved(p_opt_1_low, p_1_initial)
            noise = np.random.normal(0, 0.0258)  # More stable behavior
            p_opt = p_opt_1_low

            prob = np.clip(base_prob + noise, 0.01, 0.99)
            buy = int(np.random.rand() < prob)

            # Store data: [customer_id, price offered, bought?, customer_typ
            customer_data.append([customer_id, p_opt, buy, cust_type])
            customer_id += 1

        # update Booking peroid
        T= T-(xt/2)
```

```
    ##################################################################
    # Selling for cust type 2 according to theta 2 for upper bound
    for _ in range(theta_2_upper_time):
        # Randomly assign customer type: 1 (Reserved) or 2 (Preemptive)
        cust_type = 2
        base_prob = logistic_demand_preemptive(p_opt_2_high, p_2_initial
        noise = np.random.normal(0, 0.258)  # More stable behavior
        p_opt = p_opt_2_high

        prob = np.clip(base_prob + noise, 0.01, 0.99)
        buy = int(np.random.rand() < prob)

        # Store data: [customer_id, price offered, bought?, customer_typ
        customer_data.append([customer_id, p_opt, buy, cust_type])
        customer_id += 1

    # update Booking peroid
    T= T-(xt/2)

    ##################################################################
    # Selling for cust type 2 according to theta 2 for lower bound
    for _ in range(theta_2_lower_time):
        cust_type = 2
        base_prob = logistic_demand_preemptive(p_opt_2_low, p_2_initial)
        noise = np.random.normal(0, 0.258)  # More stable behavior
        p_opt = p_opt_2_low

        prob = np.clip(base_prob + noise, 0.01, 0.99)
        buy = int(np.random.rand() < prob)

        # Store data: [customer_id, price offered, bought?, customer_typ
        customer_data.append([customer_id, p_opt, buy, cust_type])
        customer_id += 1

    # update Booking peroid
    T= T-(xt/2)

    ##################################################################
    # Changing inital optimal prices to current,
    # Meaning whichever Price, upper or lower bound has more time alloca
    # Seprate for different customer types


    # Optimal price update - customer type 1 (Reserved)
    if theta_1_upper_time > theta_1_lower_time:
        p_opt_1 = p_opt_1_high

    elif theta_1_upper_time < theta_1_lower_time:
        p_opt_1 = p_opt_1_low

    else:
        p_opt_1 = p_opt_1

    print(f"\nOptimal Price updated - Customer type 1 (Reserved):   {p_o
```

```
        # Optimal price - customer type 2 (Preemptive)
        if theta_2_upper_time > theta_2_lower_time:
            p_opt_2 = p_opt_2_high

        elif theta_2_upper_time < theta_2_lower_time:
            p_opt_2 = p_opt_2_low

        else:
            p_opt_2 = p_opt_2

        print(f"Optimal Price updated - Customer type 2 (Preemptive): {p_opt

        # These Optimal prices are now returned and for next iteration, So C




    ##########################################################################
    # Create DataFrame for storing complete exploitation one iteration data.
    df_exploitation = pd.DataFrame(customer_data, columns=["customer_id", "p
    #print(df.head())
    #print(df2)

    # Update Inventory for this one exploitation iteration
    units_sold = df_exploitation['buy'].sum()
    c = float(c) - units_sold

    print(f"\nTotal remaining Inventory:      {c:.2f}")
    print(f"\nTotal remaining Booking peroid: {T:.2f}")

    # for output representation purpose
    print("\n####################################################################

    return T, c, p_opt_1, p_opt_2, customer_id, df_exploitation, p_opt_1, p_
```

2.6 Main Loop 2 - Conduct Phase 2 i.e. Exploitation:

This is the second and final main loop in the notebook.

Runs until booking peroid or inventory is finished whichever is first.

It calls exploitation funciton.

In [162…

```
##############################################################################
# Conduct Phase 2   i.e.   ***Exploitation***: Call the function with results
# Or
# Phase 2 - Loop

print("\n","*** Phase 2, Exploitation Start - Detailed Input Data from Explo
```

```
p_opt_1 = p_opt_tuple[0]
p_opt_1 = round(p_opt_1, 2)

p_opt_2 = p_opt_tuple[1]
p_opt_2 = round(p_opt_2, 2)


print("\nRemaining Inventory: ",c)
print("\nRemaining Booking peroid: ",T)
print(f"\nOptimal Price - Reserved, p_opt_1: {p_opt_1}")
print('\nPrice List - Reserved: ', p)
print(f"\nOptimal Price - Preemptive, p_opt_2: {p_opt_2}")
print('\nPrice List - Preemptive: ', p2)
print(f"\nUnit value of inventory - Dual variable,  z*: {z_opt:.2f}\n\n\n")




customer_data = [] # list to store customer data into data frame df_2 from e
df_2 = pd.DataFrame(customer_data, columns=["customer_id", "price", "buy", "

# ensuring Booking peroid is not in decimal, even if it is then converting t
T=int(T)

# Intorducing ti as iteration variale which will be used to see what iterati
ti = 0

# Safe Inventory - variable used in Inventory interlock, to avoid inventory
safe_c = 3*2*customers_per_price*0.6
# here,
# 3 = maximum instances used in exploitation at once when we change prices w
# if case of sufficient is there, demand cannot pass insufficient inventory
# so demand for 3 instances is more than 1 and thus, safety of assuming 3 in
# 2 = customer types
# 0.6 = rough probability that people can buy at that instance, decided afte
# customers_per_price = customerr approaching per price per type.
# So, basically we are ensuring that Inventory doesn't go negative and we ar
# then we are ensuring to run code only when inventory is above safety inven


# Safe booking time limit
# This ensures booking peroid does not cross zero and become negative
safe_T = xt*3
# here, 3 is the maximum instances exploitation takes in one iteration
# xt is the unit time a instance has. xt = T/t
# here, t is total instances


while  T > safe_T :

    if c > safe_c:
        T, c, p_opt_1, p_opt_2, customer_id, df_exploitation, p_opt_1, p_opt
```

```
        # Dataframe update for exploitation - Phase 2
        # Complete data of both phases in df_2
        df_2 = pd.concat([df_2, df_exploitation], ignore_index=True)


    else:
        print('Insufficient Inventory')
        break
        # or - if you want to observe how many more times loop executes unco
        # T -= 1



# Save the final dataframe to a CSV file
df_2['price'] = df_2['price'].round(2)  # rounds to 2 decimal places
df_2.to_csv("all_output.csv", mode='a', header=False, index=False)
print("\nData from df_2 appended to all_output.csv")
print("---------------------------------------------------------------------



print("\n\n-------------------------------------------------------------------
print("\n****** Phase 2 - Exploitation has Ended *******\n")

print("\nInventory details:")
print("Total remaining Inventory after all sales (booking peroid finish):",c
print(f"Total Inventory utilized: {c0-c} out of {c0}   i.e. {(((c0-c)/c0)*10

print("\nBooking peroid details:")
print("Booking peroid remaining after all Sales and all Phases completion:",
print(f"Total Booking peroid Utilized: {T0-T} out of {T0}   i.e.  {(((T0-T)/

print("\nOptimal price obtained at End:")
print(f"Cust type 1 (Reserved):   {p_opt_1}")
print(f"Cust type 2 (Preemptive): {p_opt_2}")

print("\n---------------------------------------------------------------------
print("######################################################################


# Seprate Data frames for different customer types for phase 2
# Just made in case Maybe used later for other different operations like plo
df_2_type_1 = df_2[df_2['customer_type'] == 1].copy()
df_2_type_2 = df_2[df_2['customer_type'] == 2].copy()


#########################################################################
```

```
 *** Phase 2, Exploitation Start - Detailed Input Data from Exploration Resu
lts: ***

Remaining Inventory:  5538.0

Remaining Booking peroid:  70.0

Optimal Price - Reserved, p_opt_1: 4410.3

Price List - Reserved:  [4349.03 4410.3  4471.57 4532.83 4594.1 ]

Optimal Price - Preemptive, p_opt_2: 2860.51

Price List - Preemptive:  [2637.98 2715.11 2792.25 2869.38 2946.52]

Unit value of inventory - Dual variable,  z*: 448.01



Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
1 (Tth) Iteration running in Exploitation:
Booking time remaining: 70.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4410.30 = 0.72
Cust type 2 (Preemptive) Estimated demand at price 2860.51 = 0.96
Total Future demand: 11760.00
total Inventory Remaining: 5538

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Upper bound price has High demand - D_upper_1 =  0.7
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.84
raw_theta: -0.40
Theta cust type 1 - Reserved:     0.4
raw_theta: 0.26
Theta cust type 2 - Preemptive:   0.26

Optimal Price updated - Customer type 1 (Reserved):   4577.39
```

```
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      5226.00

Total remaining Booking peroid: 67.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
2 (Tth) Iteration running in Exploitation:
Booking time remaining: 67.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4577.39 = 0.66
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 10251.00
total Inventory Remaining: 5226

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Upper bound price has High demand - D_upper_1 =  0.7
5
Cust type 2 - Preemptive: Both upper and lower bound demand is same
raw_theta: -0.10
Theta cust type 1 - Reserved:     0.1
Theta cust type 2 - Preemptive:   0.5

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      4921.00

Total remaining Booking peroid: 64.00

############################################################################
#################
```

```
Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
3 (Tth) Iteration running in Exploitation:
Booking time remaining: 64.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 9984.00
total Inventory Remaining: 4921

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09
```

```
C:\Users\vijay\AppData\Local\Temp\ipykernel_14996\3813607715.py:73: FutureWa
rning: The behavior of DataFrame concatenation with empty or all-NA entries
is deprecated. In a future version, this will no longer exclude empty or all
-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  df_2 = pd.concat([df_2, df_exploitation], ignore_index=True)
```

```
Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.6
3
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.84
raw_theta: 0.13
Theta cust type 1 - Reserved:     0.13
raw_theta: 0.11
Theta cust type 2 - Preemptive:   0.11

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      4625.00

Total remaining Booking peroid: 61.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
4 (Tth) Iteration running in Exploitation:
Booking time remaining: 61.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 9516.00
total Inventory Remaining: 4625

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.7
3
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.91
raw_theta: 0.76
Theta cust type 1 - Reserved:     0.76
raw_theta: 0.10
Theta cust type 2 - Preemptive:   0.1
```

```
Optimal Price updated - Customer type 1 (Reserved):   4427.01
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      4308.00

Total remaining Booking peroid: 58.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
5 (Tth) Iteration running in Exploitation:
Booking time remaining: 58.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4427.01 = 0.72
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 9222.00
total Inventory Remaining: 4308

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.7
1
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.85
raw_theta: 0.15
Theta cust type 1 - Reserved:     0.15
raw_theta: 0.12
Theta cust type 2 - Preemptive:   0.12

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      4006.00

Total remaining Booking peroid: 55.00
```

```
############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]	valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]	valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
6 (Tth) Iteration running in Exploitation:
Booking time remaining: 55.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 8580.00
total Inventory Remaining: 4006

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.6
6
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.9
raw_theta: 0.37
Theta cust type 1 - Reserved:     0.37
raw_theta: 0.24
Theta cust type 2 - Preemptive:   0.24

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      3698.00

Total remaining Booking peroid: 52.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]	valid's binary_logloss: 0.636184
```

```
############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
7 (Tth) Iteration running in Exploitation:
Booking time remaining: 52.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 8112.00
total Inventory Remaining: 3698

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Upper bound price has High demand - D_upper_1 =  0.6
8
Cust type 2 - Preemptive: Upper bound price has High demand - D_upper_2 =
0.87
raw_theta: -0.72
Theta cust type 1 - Reserved:     0.72
raw_theta: -0.18
Theta cust type 2 - Preemptive:   0.18

Optimal Price updated - Customer type 1 (Reserved):   4427.01
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      3393.00

Total remaining Booking peroid: 49.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################
```

```
############################################################################
#################
8 (Tth) Iteration running in Exploitation:
Booking time remaining: 49.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4427.01 = 0.72
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 7791.00
total Inventory Remaining: 3393

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.6
7
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.85
raw_theta: 0.14
Theta cust type 1 - Reserved:     0.14
raw_theta: 0.12
Theta cust type 2 - Preemptive:   0.12

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      3085.00

Total remaining Booking peroid: 46.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
9 (Tth) Iteration running in Exploitation:
Booking time remaining: 46.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
```

```
Total Future demand: 7176.00
total Inventory Remaining: 3085

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.6
4
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.93
raw_theta: 0.14
Theta cust type 1 - Reserved:     0.14
raw_theta: 0.17
Theta cust type 2 - Preemptive:   0.17

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      2795.00

Total remaining Booking peroid: 43.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
10 (Tth) Iteration running in Exploitation:
Booking time remaining: 43.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 6708.00
total Inventory Remaining: 2795

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09
```

```
Cust type 1 - Reserved: Upper bound price has High demand - D_upper_1 =  0.6
2
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.89
raw_theta: -0.16
Theta cust type 1 - Reserved:     0.16
raw_theta: 0.22
Theta cust type 2 - Preemptive:   0.22

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      2504.00

Total remaining Booking peroid: 40.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
11 (Tth) Iteration running in Exploitation:
Booking time remaining: 40.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 6240.00
total Inventory Remaining: 2504

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Upper bound price has High demand - D_upper_1 =  0.6
7
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.88
raw_theta: -0.32
Theta cust type 1 - Reserved:     0.32
raw_theta: 0.63
Theta cust type 2 - Preemptive:   0.63
```

```
Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2779.43

Total remaining Inventory:      2203.00

Total remaining Booking peroid: 37.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
12 (Tth) Iteration running in Exploitation:
Booking time remaining: 37.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2779.43 = 0.96
Total Future demand: 6105.00
total Inventory Remaining: 2203

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.6
5
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.89
raw_theta: 0.20
Theta cust type 1 - Reserved:     0.2
raw_theta: 0.30
Theta cust type 2 - Preemptive:   0.3

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      1909.00

Total remaining Booking peroid: 34.00
```

```
################################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]	valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]	valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


################################################################################
#################
13 (Tth) Iteration running in Exploitation:
Booking time remaining: 34.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 5304.00
total Inventory Remaining: 1909

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.7
3
Cust type 2 - Preemptive: Upper bound price has High demand - D_upper_2 =
0.84
raw_theta: 0.14
Theta cust type 1 - Reserved:     0.14
raw_theta: -0.57
Theta cust type 2 - Preemptive:   0.57

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2779.43

Total remaining Inventory:      1603.00

Total remaining Booking peroid: 31.00

################################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]	valid's binary_logloss: 0.636184
```

```
############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
14 (Tth) Iteration running in Exploitation:
Booking time remaining: 31.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2779.43 = 0.96
Total Future demand: 5115.00
total Inventory Remaining: 1603

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.6
2
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.86
raw_theta: 0.05
Theta cust type 1 - Reserved:     0.05
raw_theta: 0.06
Theta cust type 2 - Preemptive:   0.06

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      1303.00

Total remaining Booking peroid: 28.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################
```

```
############################################################################
#################
15 (Tth) Iteration running in Exploitation:
Booking time remaining: 28.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 4368.00
total Inventory Remaining: 1303

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.6
4
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.87
raw_theta: 0.12
Theta cust type 1 - Reserved:     0.12
raw_theta: 0.12
Theta cust type 2 - Preemptive:   0.12

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      1006.00

Total remaining Booking peroid: 25.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
16 (Tth) Iteration running in Exploitation:
Booking time remaining: 25.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
```

```
Total Future demand: 3900.00
total Inventory Remaining: 1006

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Lower bound price has High demand - D_lower_1 =  0.7
1
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.89
raw_theta: 0.41
Theta cust type 1 - Reserved:     0.41
raw_theta: 0.10
Theta cust type 2 - Preemptive:   0.1

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2946.52

Total remaining Inventory:      700.00

Total remaining Booking peroid: 22.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
17 (Tth) Iteration running in Exploitation:
Booking time remaining: 22.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2946.52 = 0.87
Total Future demand: 3432.00
total Inventory Remaining: 700

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09
```

```
Cust type 1 - Reserved: Upper bound price has High demand - D_upper_1 =  0.7
4
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.89
raw_theta: -0.30
Theta cust type 1 - Reserved:     0.3
raw_theta: 0.81
Theta cust type 2 - Preemptive:   0.81

Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2779.43

Total remaining Inventory:      404.00

Total remaining Booking peroid: 19.00

############################################################################
#################


Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[24]    valid's binary_logloss: 0.636184

############### Tuned Light Gmb test accuracy: 0.62, ################

Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[31]    valid's binary_logloss: 0.237325

############### Tuned Light Gmb test accuracy: 0.93, ################


############################################################################
#################
18 (Tth) Iteration running in Exploitation:
Booking time remaining: 19.00 days out of 100.0 days remaining.

Cust type 1 (Reserved) Estimated demand at price 4594.10 = 0.69
Cust type 2 (Preemptive) Estimated demand at price 2779.43 = 0.96
Total Future demand: 3135.00
total Inventory Remaining: 404

Entered Into Case 2: Insufficient Inventory

Alpha for cust. type 1: 167.09
Alpha for cust. type 2: 167.09

Cust type 1 - Reserved: Upper bound price has High demand - D_upper_1 =  0.6
1
Cust type 2 - Preemptive: Lower bound price has High demand - D_lower_2 =
0.88
raw_theta: -0.36
Theta cust type 1 - Reserved:     0.36
raw_theta: 0.54
Theta cust type 2 - Preemptive:   0.54
```

```
Optimal Price updated - Customer type 1 (Reserved):   4594.10
Optimal Price updated - Customer type 2 (Preemptive): 2637.98

Total remaining Inventory:      113.00

Total remaining Booking peroid: 16.00

############################################################################
#################


Insufficient Inventory

Data from df_2 appended to all_output.csv
----------------------------------------------------------------------------
---------------------------------



----------------------------------------------------------------------------
---------------------------------

****** Phase 2 - Exploitation has Ended *******


Inventory details:
Total remaining Inventory after all sales (booking peroid finish): 113.0
Total Inventory utilized: 9887.0 out of 10000.0   i.e. 98.87%


Booking peroid details:
Booking peroid remaining after all Sales and all Phases completion: 16.0 day
s
Total Booking peroid Utilized: 84.0 out of 100.0   i.e.  84.00%


Optimal price obtained at End:
Cust type 1 (Reserved):   4594.1
Cust type 2 (Preemptive): 2637.98

----------------------------------------------------------------------------
---------------------------------



############################################################################
##########################################
```

3.0 Data Management and Data Preparation:

Data preparation for results and plotting used further.

3.1 **Revenue** and **Demand** Calculation: **Final** for **Both Phases**

Prices Order - Last occured (Prices not sorted in ascending order)

Prices are grouped by and sorted according to latest occurence of price in price column.

Used in Regret calculation, plotting and storing data into CSVs repective to customer types.

3.1.1 Revenue and Demand Calculation function --> (***With*** considering **z_opt** (unit value of inventory)):

z_opt = z_opt

This is used generally, but sometimes according to business calculations which might not consider Dual variable i.e. z_opt.

The function can be used in both ways with and without z_opt only by changing z_opt value.

And

3.1.2 Revenue and Demand Calculation function --> (***Without*** considering **z_opt** (unit value of inventory)):

z_opt = 0

In real life there is no algorithm's dual variable assumed and one might calculate revenue naturally.

i.e. just pure revenue = price x units sold.

In function below written as: revenue = (price) * customers_bought

So, z_opt = 0, its kept as 0 in function attribute input while calling funtion.

In [163…

```
# sorts the prices and customer type because of group by , by defaul sorting
def compute_final_revenue_demand(df, z_opt):
    # Group by both price and customer_type
    grouped = df.groupby(['price', 'customer_type'])

    data = []
    for (price, customer_type), group in grouped:
        total_customers = len(group)
        customers_bought = group['buy'].sum()
        revenue = (price - z_opt) * customers_bought
        demand = customers_bought / total_customers if total_customers > 0 e
        data.append((price, round(revenue, 2), round(demand, 4), customer_ty

    # Create final DataFrame
    return pd.DataFrame(data, columns=['price', 'revenue', 'demand', 'custom





# Does not sort prices, instead puts the price index where its found in late
# but still groups by prices and ensures to calculate revenue and demand acc
# Used on dataframes - seprated on basis of customer type from final datafra

def compute_final_revenue_demand_natural_order(df, z_opt):

    """ # --- Step 1: Determine unique prices based on their last appearance
        # We reverse the DataFrame to prioritize the last occurrence of each
        # Then drop duplicates to get one entry per price (based on last app
        # Finally, reverse again to restore the original row order (but now
    """

    last_occurrence_order = df[::-1].drop_duplicates('price')[::-1]['price']

    data = []
    # Assumes the entire DataFrame is for a single customer type
    customer_type = df['customer_type'].iloc[0]

    # --- Step 2: Loop through each price in the custom order ---
    for price in last_occurrence_order:
        # Filter rows for the current price
        group = df[df['price'] == price]

        # --- Step 3: Compute metrics ---
        total_customers = len(group)                    # Total number of c
        customers_bought = group['buy'].sum()           # Number of custome
        revenue = (price - z_opt) * customers_bought    # Revenue after sub
        demand = customers_bought / total_customers if total_customers > 0 e
```

```
        # Append results as a tuple
        data.append((price, round(revenue, 2), round(demand, 4), customer_ty

    # --- Step 4: Convert results to a DataFrame ---
    return pd.DataFrame(data, columns=['price', 'revenue', 'demand', 'custom
```

3.2 Data Preperation:

Data Frames Sepration for Data Preperation for Results and Plotting:

In [164…

```
# Final data frame which stores data from both phase 1 & 2:
df_final = df_final = pd.concat([df_1, df_2], ignore_index=True)

# Seprate Data frames for different customer types for total data i.e. from
df_final_type_1 = df_final[df_final['customer_type'] == 1].copy()
df_final_type_2 = df_final[df_final['customer_type'] == 2].copy()

# Counting the total rows in buy column for different customer types, used i
# This tells us how many customers approached in total for a particular cust
buy_count_final_type_1 = df_final_type_1['buy'].count()
buy_count_final_type_2 = df_final_type_2['buy'].count()
```

3.2.1 Data Preparation for Regret Calculation, Plotting Results ----> ***With*** considering ***z_opt***

And

Data Preparation for Showcasing and Storing Revenue and Demand Data in diff. CSVs:

Note - This is for whole booking peroid combining both phases - 1 & 2 (Exploration and Exploitation), but also differentiated according to customer types

****With*** considering ***z_opt*** for both customer types in revenue calculation.

In [165…

```
################################################################################

# Calculating and storing results (Revenue and Demand ***WITH*** considering

################################################################################
# Calculating results (both cust. types): Revenue and demand per price and p
df_final_results_with_z_opt = compute_final_revenue_demand(df_final, z_opt)

# Printing Results data frame
print("Results (Revenue, Demand acc. to price) for both combined customer ty

# Storing to file results_with_z_opt.csv
df_final_results_with_z_opt.to_csv("results_with_z_opt.csv", mode='w', heade
print("Results stored to results_with_z_opt.csv\n")
```

```
##############################################################################
# Calculating results (Cust. type 1 (Reserved)): Revenue and demand per pric
df_final_results_type_1_with_z_opt = compute_final_revenue_demand_natural_or

# Printing Results data frame
print("Results (Revenue, Demand acc. to price) for Cust. type 1 (Reserved)\n

# Storing to file results_type_1_with_z_opt.csv
df_final_results_type_1_with_z_opt.to_csv("results_type_1_with_z_opt.csv", m
print("Results stored to results_type_1_with_z_opt.csv\n")


##############################################################################
# Calculating results (Cust. type 2 (Preemptive)): Revenue and demand per pr
df_final_results_type_2_with_z_opt = compute_final_revenue_demand_natural_or

# Printing Results data frame
print("\n\nResults (Revenue, Demand acc. to price) for Cust. type 2 (Preempt

# Storing to file results_type_2_with_z_opt.csv
df_final_results_type_2_with_z_opt.to_csv("results_type_2_with_z_opt.csv", m
print("Results stored to results_type_2_with_z_opt.csv\n")
```

```
Results (Revenue, Demand acc. to price) for both combined customer types:
      price    revenue  demand  customer_type
0  1000.00   52991.04  0.9412              2
1  1053.48   61152.47  0.9806              2
2  1526.74  101400.62  0.9495              2
3  2000.00  299534.07  0.9650              2
4  2437.80  196989.21  0.9429              2
Results stored to results_with_z_opt.csv

Results (Revenue, Demand acc. to price) for Cust. type 1 (Reserved)
      price    revenue  demand  customer_type
0  3000.00  214367.16  0.8571              1
1  6000.00  166559.70  0.2778              1
2  5000.00  245807.46  0.5347              1
3  7000.00   65519.90  0.1075              1
4  3053.18  224044.62  0.8515              1
Results stored to results_type_1_with_z_opt.csv



Results (Revenue, Demand acc. to price) for Cust. type 2 (Preemptive)
     price    revenue  demand  customer_type
0  5000.0   13655.97  0.0288              2
1  4000.0   14207.96  0.0392              2
2  3000.0  145463.43  0.5327              2
3  1000.0   52991.04  0.9412              2
4  2000.0  299534.07  0.9650              2
Results stored to results_type_2_with_z_opt.csv
```

3.2.2 Data Preparation for Regret Calculation, Plotting Results ----> ***Without*** considering ***z_opt***

And

Data Preparation for Showcasing and Storing Revenue and Demand Data in diff. CSVs:

Note - This is for whole booking peroid combining both phases - 1 & 2 (Exploration and Exploitation),

****Without*** considering ***z_opt*** for both customer types.

In [166…

```
###########################################################################

# Calculating and storing results (Revenue and Demand ***WITHOUT*** consider

###########################################################################
# Calculating results (both cust. types): Revenue and demand per price and p
df_final_results = compute_final_revenue_demand(df_final, z_opt = 0)

# Printing Results data frame
print("Results (Revenue, Demand acc. to price) for both combined customer ty

# Storing to file results.csv
df_final_results.to_csv("results.csv", mode='w', header=True, index=False)
print("Results stored to results.csv\n")


###########################################################################
# Calculating results (Cust. type 1 (Reserved)): Revenue and demand per pric
df_final_results_type_1 = compute_final_revenue_demand_natural_order(df_fina

# Printing Results data frame
print("Results (Revenue, Demand acc. to price) for Cust. type 1 (Reserved)\n

# Storing to file results_type_1_with_z_opt.csv
df_final_results_type_1.to_csv("results_type_1.csv", mode='w', header=True,
print("Results stored to results_type_1.csv\n")


###########################################################################
# Calculating results (Cust. type 2 (Preemptive)): Revenue and demand per pr
df_final_results_type_2 = compute_final_revenue_demand_natural_order(df_fina

# Printing Results data frame
print("\n\nResults (Revenue, Demand acc. to price) for Cust. type 2 (Preempt

# Storing to file results_type_2.csv
df_final_results_type_2.to_csv("results_type_2.csv", mode='w', header=True,
print("Results stored to results_type_2.csv\n")
```

```
Results (Revenue, Demand acc. to price) for both combined customer types:
      price    revenue  demand  customer_type
0  1000.00   96000.00  0.9412              2
1  1053.48  106401.48  0.9806              2
2  1526.74  143513.56  0.9495              2
3  2000.00  386000.00  0.9650              2
4  2437.80  241342.20  0.9429              2
Results stored to results.csv

Results (Revenue, Demand acc. to price) for Cust. type 1 (Reserved)
      price    revenue  demand  customer_type
0  3000.00  252000.00  0.8571              1
1  6000.00  180000.00  0.2778              1
2  5000.00  270000.00  0.5347              1
3  7000.00   70000.00  0.1075              1
4  3053.18  262573.48  0.8515              1
Results stored to results_type_1.csv



Results (Revenue, Demand acc. to price) for Cust. type 2 (Preemptive)
     price   revenue  demand  customer_type
0  5000.0   15000.0  0.0288              2
1  4000.0   16000.0  0.0392              2
2  3000.0  171000.0  0.5327              2
3  1000.0   96000.0  0.9412              2
4  2000.0  386000.0  0.9650              2
Results stored to results_type_2.csv
```

3.2.3 Creating Dataframes with Ascending (Sortred) Prices order

First we created non sorted or ordered purposely to use in the plots below.

Now we create an ascending prices sorted dataframes and CSVs accordingly.

In [167…

```
# Sorting each DataFrame by ascending price for different plotting.
# This step is crucial for generating visually coherent demand and revenue c
# Without sorting, price points may appear out of order, resulting in mislea

# we are going to plot both sorted and non sorted.


## With considering z_opt
df_final_results_type_1_with_z_opt_sorted = df_final_results_type_1_with_z_c
df_final_results_type_2_with_z_opt_sorted = df_final_results_type_2_with_z_c


## Without considering z_opt
df_final_results_type_1_sorted = df_final_results_type_1.sort_values(by='pri
df_final_results_type_2_sorted = df_final_results_type_2.sort_values(by='pri
```

4.0 **Results:**

4.1 Total Revenue Calculation function: for ***whole booking peroid***

In [168…
```
def calculate_total_revenue(df):
    return df['revenue'].sum()
```

4.1.1 Calculating different Revenues: ***With considering z_opt*** (Unit value of inventory)

Here, revenue is Adjusted Revenue = (Price - z_opt) x Units sold

z_opt = unit price of inventory learned by Dual variable throughout the algorithm iterations.

In [169…
```
total_revenue_with_z_opt = round(calculate_total_revenue(df_final_results_wi
total_revenue_type_1_with_z_opt = round(calculate_total_revenue(df_final_res
total_revenue_type_2_with_z_opt = round(calculate_total_revenue(df_final_res

print("\nRevenues With considering Unit value of inventory (z_opt):\n")
print("Total Revenue:                     ", total_revenue_with_z_opt)
print("\nTotal Revenue for Customer Type 1: ", total_revenue_type_1_with_z_o
print("\nTotal Revenue for Customer Type 2: ", total_revenue_type_2_with_z_o

print("\n\nRevenues calculated from: \n")

print("Price list given by user, for Customer type 1 (Reserved):    ",p_1_ir
print("Price list given by user, for Customer type 2 (Preemptive):  ",p_2_ir

print("\nSelling inventory of:    ",c0-c,"units.")
print("From total inventory of: ",c0,"units.")

print("\nIn time peroid of:            ",T0-T,unit_T)
print("From total booking peroid of:",T0,unit_T,)

print("\nFinal Optimal Price for Customer type 1 (Reserved):      ",p_opt_1,
print("Final Optimal Price for Customer type 2 (Preemptive):    ",p_opt_2,"\
```

```
Revenues With considering Unit value of inventory (z_opt):

Total Revenue:                      29427201.4

Total Revenue for Customer Type 1:  16305867.92

Total Revenue for Customer Type 2:  13121333.48


Revenues calculated from:

Price list given by user, for Customer type 1 (Reserved):     [3000. 4000. 5
000. 6000. 7000.]
Price list given by user, for Customer type 2 (Preemptive):   [1000. 2000. 3
000. 4000. 5000.]


Selling inventory of:     9887.0 units.
From total inventory of:  10000.0 units.

In time peroid of:             84.0 days
From total booking peroid of: 100.0 days

Final Optimal Price for Customer type 1 (Reserved):       4594.1
Final Optimal Price for Customer type 2 (Preemptive):     2637.98
```

4.1.2 Calculating different Revenues: ***Without considering z_opt*** (Unit value of inventory)

Here, revenue is normal Revenue = Price x Units sold

In [170…

```
total_revenue = round(calculate_total_revenue(df_final_results), 2)
total_revenue_type_1 = round(calculate_total_revenue(df_final_results_type_1
total_revenue_type_2 = round(calculate_total_revenue(df_final_results_type_2

print("\nRevenues Without considering Unit value of inventory (z_opt):\n")
print("Total Revenue:                     ", total_revenue)
print("\nTotal Revenue for Customer Type 1: ", total_revenue_type_1)
print("\nTotal Revenue for Customer Type 2: ", total_revenue_type_2)

print("\n\nRevenues calculated from: \n")

print("Price list given by user, for Customer type 1 (Reserved):    ",p_1_ir
print("Price list given by user, for Customer type 2 (Preemptive):  ",p_2_ir

print("\nSelling inventory of:    ",c0-c,"units.")
print("From total inventory of: ",c0,"units.")

print("\nIn time peroid of:            ",T0-T,unit_T)
print("From total booking peroid of:",T0,unit_T,)

print("\nFinal Optimal Price for Customer type 1 (Reserved):      ",p_opt_1,
print("Final Optimal Price for Customer type 2 (Preemptive):    ",p_opt_2,"\
```

```
Revenues Without considering Unit value of inventory (z_opt):

Total Revenue:                        33856676.27

Total Revenue for Customer Type 1:   18131956.68

Total Revenue for Customer Type 2:   15724719.59


Revenues calculated from:

Price list given by user, for Customer type 1 (Reserved):     [3000. 4000. 5
000. 6000. 7000.]
Price list given by user, for Customer type 2 (Preemptive):   [1000. 2000. 3
000. 4000. 5000.]


Selling inventory of:     9887.0 units.
From total inventory of:  10000.0 units.

In time peroid of:             84.0 days
From total booking peroid of: 100.0 days

Final Optimal Price for Customer type 1 (Reserved):       4594.1
Final Optimal Price for Customer type 2 (Preemptive):     2637.98
```

4.2 Calculating ***REGRET*** i.e. performance of algorithm:

Note - Please look below (after regret calculation) regret info for which is good regret and which is not.

4.2.1 Regret function:

In [171…

```python
def calculate_regret(df, buy_count, z_opt):
    # Extract actual revenue
    actual_revenue = df['revenue'].sum()
    # print(actual_revenue)

    # Get the row with maximum revenue
    max_revenue_row = df.loc[df['revenue'].idxmax()]
    # print(max_revenue_row)

    # Extract price and demand from that row
    optimal_price = max_revenue_row['price']
    optimal_demand = max_revenue_row['demand']
    # print(optimal_price,optimal_price)
```

```
    # Calculate optimal revenue
    optimal_revenue = (optimal_price - z_opt) * optimal_demand * buy_count
    # print(optimal_revenue)

    # Avoid division by zero
    if optimal_revenue == 0:
        return 0.0

    # Calculate regret
    regret = ((optimal_revenue - actual_revenue) / optimal_revenue) * 100
    return round(regret, 2)
```

4.2.2 Calculating Regret for both customer types on complete data of both phase 1 and 2:

With considering ***z_opt***, unit value of inventory:

In [172…

```
regret_type_1_z_opt = calculate_regret(df_final_results_type_1_with_z_opt, b
print("Regret for customer type 1 with considering unit value of inventory (

regret_type_2_z_opt = calculate_regret(df_final_results_type_2_with_z_opt, b
print("\nRegret for customer type 2 with considering unit value of inventory
```

```
Regret for customer type 1 with considering unit value of inventory (z_opt):
3.8 % ***

Regret for customer type 2 with considering unit value of inventory (z_opt):
7.6 % ***
```

4.2.3 Calculating Regret for both customer types on complete data of both phase 1 and 2:

Without considering ***z_opt***, unit value of inventory:

In [173…

```
regret_type_1 = calculate_regret(df_final_results_type_1, buy_count_final_ty
print("Regret for customer type 1 without considering unit value of inventor

regret_type_2 = calculate_regret(df_final_results_type_2, buy_count_final_ty
print("\nRegret for customer type 2 without considering unit value of invent
```

```
Regret for customer type 1 without considering unit value of inventory (z_op
t): 3.45 % ***

Regret for customer type 2 without considering unit value of inventory (z_op
t): 6.11 % ***
```

*****Regarding Regret Results**:

Calculates performance of algorithm.

Basically compares actual revenue to optimal revenue which means if actual revenue is near optimal revenue then algorithm is working well otherwise not.

According to the Main paper, regret should be **between 0 and 15 %**, if the value crosses 15 then it is not good.

If Regret is below 0 i.e. -ve than it means, actual revenue is better than optimal revenue.

5. **Plotting Results:**

5.1 Plotting function: ***Price vs Customer id***

Sequentially plotting all prices to see price change behaviour (how price changes thorughout the run of algorithm)

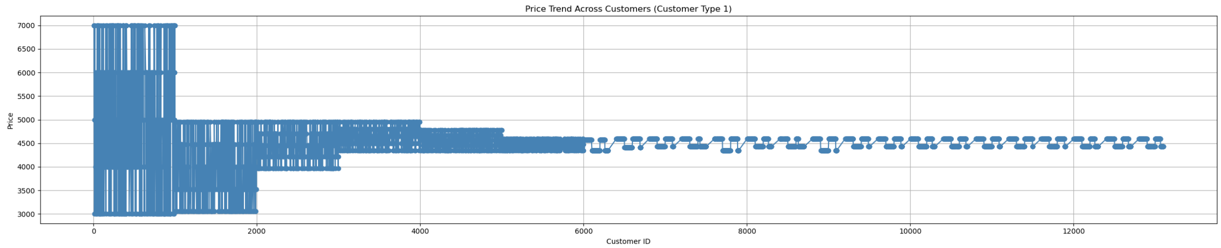
In [174…
```
def plot_customer_price(df):
    plt.figure(figsize=(24, 5))
    plt.plot(df['customer_id'], df['price'], marker='o', linestyle='-', colo

    plt.title(f"Price Trend Across Customers (Customer Type {df['customer_ty
    plt.xlabel("Customer ID")
    plt.ylabel("Price")
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

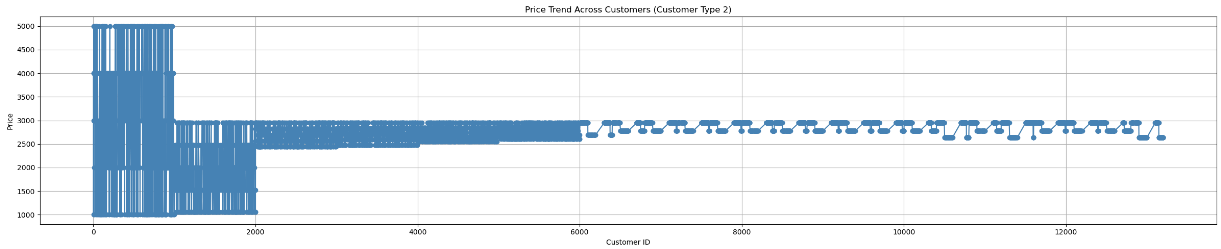
5.1.1 Plotting: ***Price vs customer***

Simply plotting all the price record to visualize the price movement throughout the run of algorithm.

Plotting only seprated for different customer types.

In [175…
```
# Plot Price vs Customer_id for customer type 1 (Reserved)
plot_customer_price(df_final_type_1)

# Plot Price vs Customer_id for customer type 2 (Preemptive)
plot_customer_price(df_final_type_2)
```





5.2 Plotting function: ***Price vs Demand and Price vs Revenue***

Plots:

Price vs Demand

and

Price vs Revenue

We have done plotting on two types of data. sorted and non sorted.

Sorted has data according to sorted prices, which shows results very clearly.

Non - Sorted visualizes true nature of price flow (price history), but very unclear to see.

But for user's requirment we have kept all the plots.

In [176…

```
# ascending price with z_opt
def plot_results_ascending_z_opt(df):

    plt.figure(figsize=(24, 4))
    # Plot Price vs Demand
    plt.subplot(1, 2, 1)
    plt.plot(df['price'], df['demand'], marker='o', color='teal')
    plt.title(f"(Sorted - Ascending)  Price vs Demand - (Customer Type {df['
    plt.xlabel("Price")
    plt.ylabel("Demand")
    plt.grid(True)
    plt.tight_layout()
    plt.show()


    plt.figure(figsize=(24, 4))
    # Plot Price vs Revenue
```

```
    plt.subplot(1, 2, 2)
    plt.plot(df['price'], df['revenue'], marker='o', color='darkorange')
    plt.title(f"(Sorted - Ascending)  Price vs Revenue - (Customer Type {df[
    plt.xlabel("Price")
    plt.ylabel("Revenue")
    plt.grid(True)
    plt.tight_layout()
    plt.show()



# ascending price without z_opt
def plot_results_ascending(df):

    plt.figure(figsize=(24, 4))
    # Plot Price vs Demand
    plt.subplot(1, 2, 1)
    plt.plot(df['price'], df['demand'], marker='o', color='teal')
    plt.title(f"(Sorted - Ascending)  Price vs Demand - (Customer Type {df['
    plt.xlabel("Price")
    plt.ylabel("Demand")
    plt.grid(True)
    plt.tight_layout()
    plt.show()


    plt.figure(figsize=(24, 4))
    # Plot Price vs Revenue
    plt.subplot(1, 2, 2)
    plt.plot(df['price'], df['revenue'], marker='o', color='darkorange')
    plt.title(f"(Sorted - Ascending)  Price vs Revenue - (Customer Type {df[
    plt.xlabel("Price")
    plt.ylabel("Revenue")
    plt.grid(True)
    plt.tight_layout()
    plt.show()


# Non- sorted price with z_opt   # Here only Title changes otherwise both no
def plot_results_z_opt(df):

    x = range(len(df))  # 0, 1, 2, ..., n
    price_labels = df['price'].astype(str)  # Convert to string for cleaner

    plt.figure(figsize=(24, 4))
    # Plot Price vs Demand
    plt.subplot(1, 2, 1)
    plt.plot(x, df['demand'], marker='o', color='teal')
    plt.title(f"(Non-Sorted - Original)  Price vs Demand - (Customer Type {c
    plt.xlabel("Price")
    plt.ylabel("Demand")
    plt.xticks(ticks=x, labels=price_labels, rotation=45)
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

```python
    plt.figure(figsize=(24, 4))
    # Plot Price vs Revenue
    plt.subplot(1, 2, 2)
    plt.plot(x, df['revenue'], marker='o', color='darkorange')
    plt.title(f"(Non-Sorted - Original)  Price vs Revenue - (Customer Type {
    plt.xlabel("Price")
    plt.ylabel("Revenue")
    plt.xticks(ticks=x, labels=price_labels, rotation=45)
    plt.grid(True)
    plt.tight_layout()
    plt.show()




# Non- sorted price without z_opt   # Here only Title changes otherwise both
def plot_results(df):

    x = range(len(df))  # 0, 1, 2, ..., n
    price_labels = df['price'].astype(str)  # Convert to string for cleaner

    plt.figure(figsize=(24, 4))
    # Plot Price vs Demand
    plt.subplot(1, 2, 1)
    plt.plot(x, df['demand'], marker='o', color='teal')
    plt.title(f"(Non-Sorted - Original)  Price vs Demand - (Customer Type {c
    plt.xlabel("Price")
    plt.ylabel("Demand")
    plt.xticks(ticks=x, labels=price_labels, rotation=45)
    plt.grid(True)
    plt.tight_layout()
    plt.show()


    plt.figure(figsize=(24, 4))
    # Plot Price vs Revenue
    plt.subplot(1, 2, 2)
    plt.plot(x, df['revenue'], marker='o', color='darkorange')
    plt.title(f"(Non-Sorted - Original)  Price vs Revenue - (Customer Type {
    plt.xlabel("Price")
    plt.ylabel("Revenue")
    plt.xticks(ticks=x, labels=price_labels, rotation=45)
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

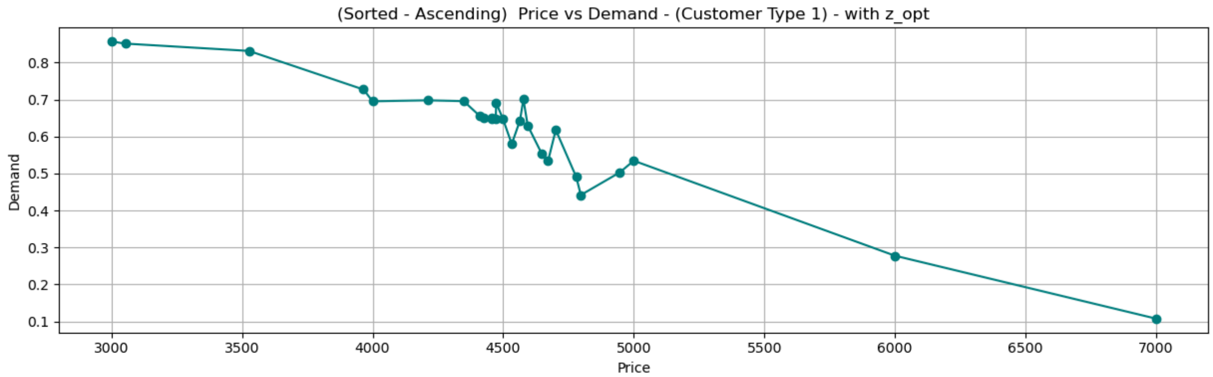
Plotting ***Price (ascending) vs Demand*** and ***Price (ascending) vs Revenue***:

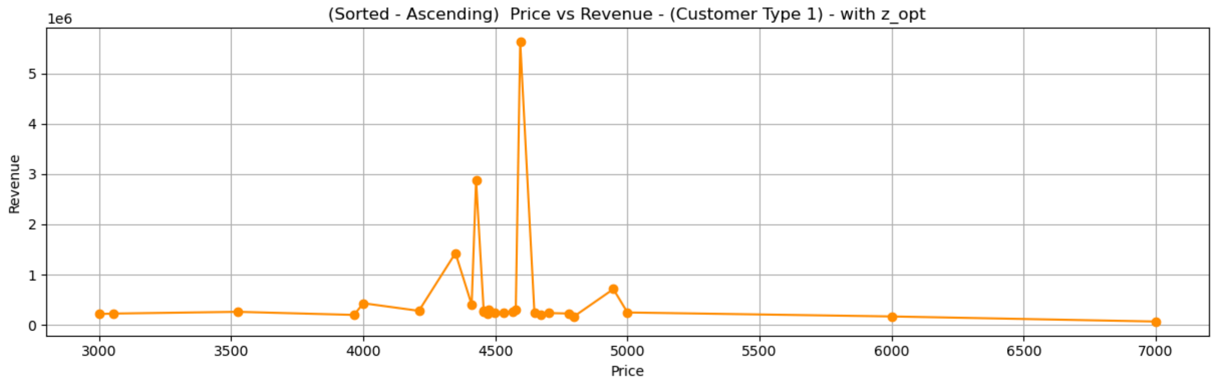
With considering ***z_opt***, Unit value of inventory, while revenue calculation.

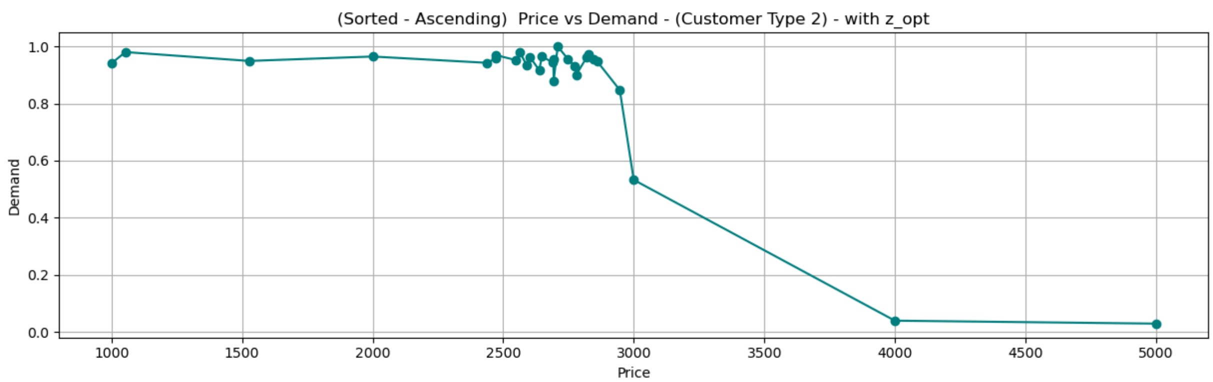
Calling plotting function with dataframes which have results (revenue and demand) taken into consideration with z_opt.

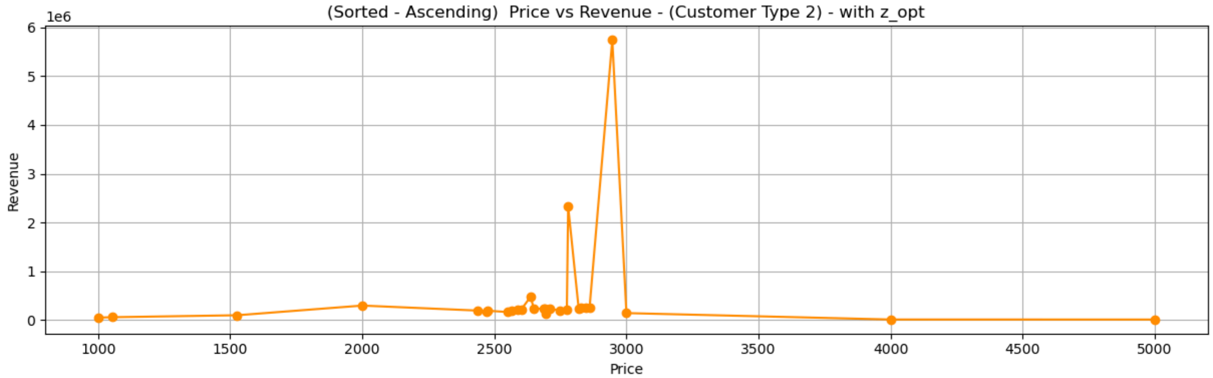
In [177…

```
# Plot ascending price results for Customer type 1 considering revenue calcu
plot_results_ascending_z_opt(df_final_results_type_1_with_z_opt_sorted)

# Plot ascending price results for Customer type 2 considering revenue calcu
plot_results_ascending_z_opt(df_final_results_type_2_with_z_opt_sorted)
```









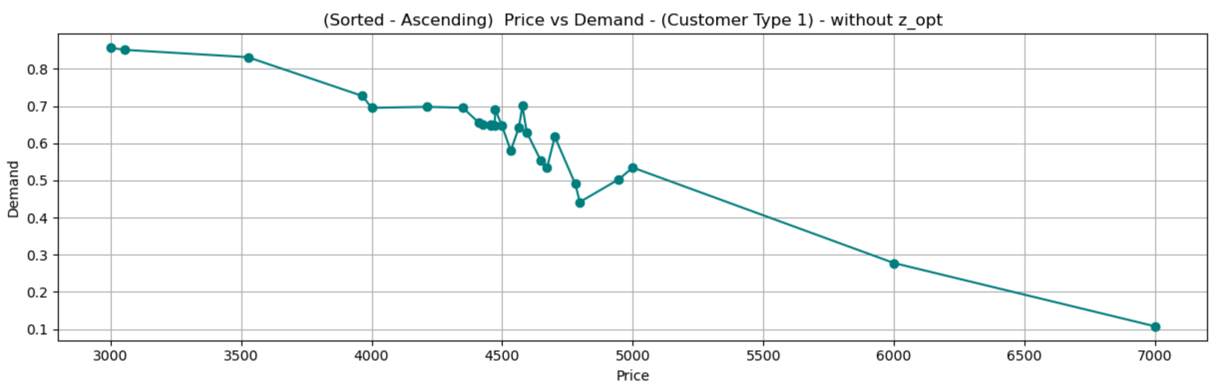
Plotting ***Price (ascending) vs Demand*** and ***Price (ascending) vs Revenue***:

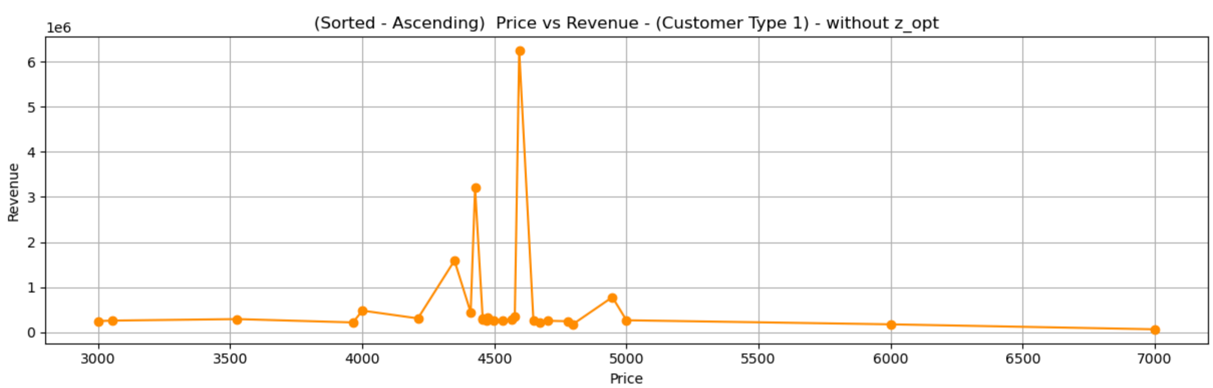
Without considering ***z_opt***, Unit value of inventory, while revenue calculation.

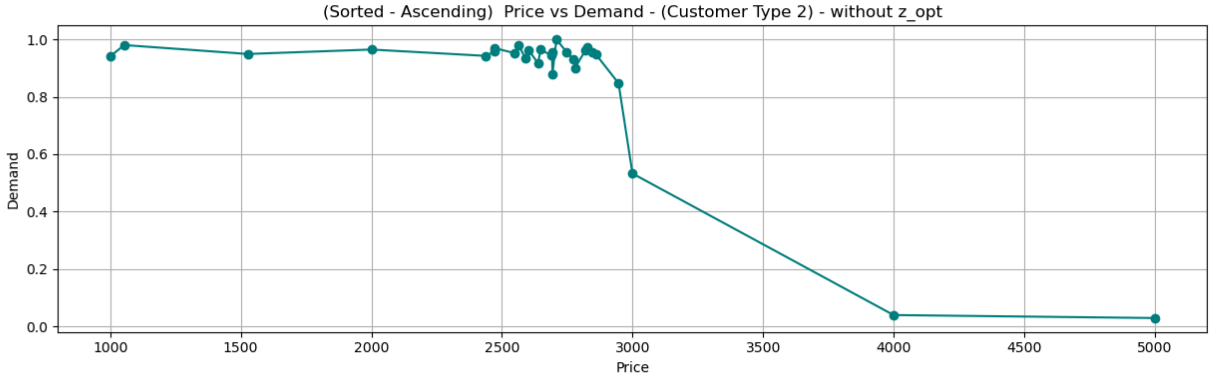
Calling plotting function with dataframes which have results (revenue and demand) taken without considering z_opt.

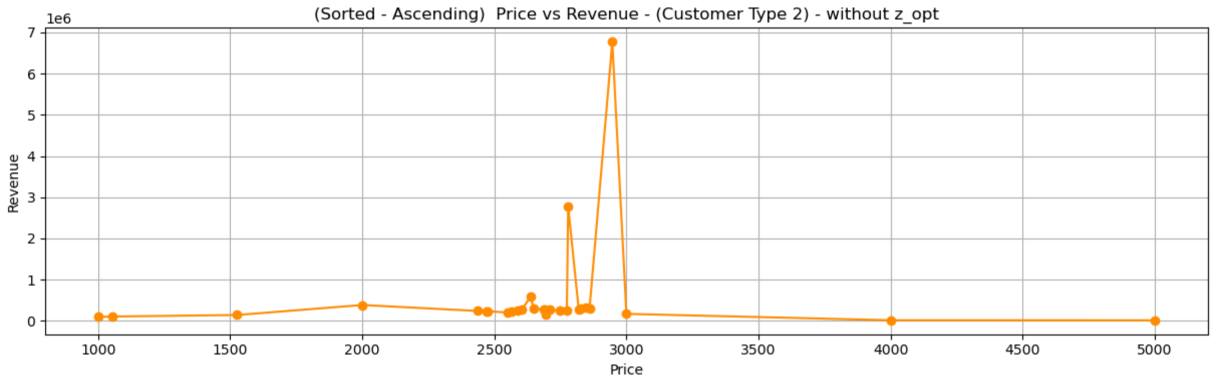
In [178…

```
# Plot ascending price results for Customer type 1 considering revenue calcu
plot_results_ascending(df_final_results_type_1_sorted)

# Plot ascending price results for Customer type 2 considering revenue calcu
plot_results_ascending(df_final_results_type_2_sorted)
```









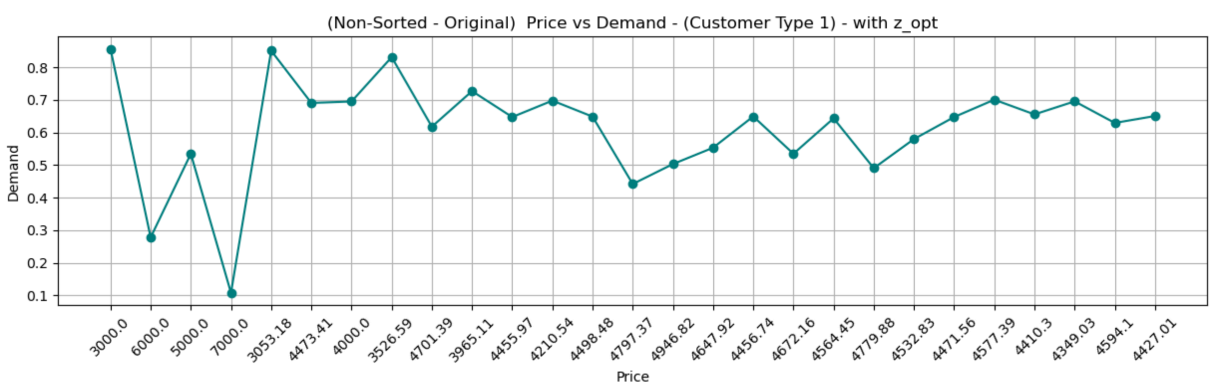
Plotting ***Price (Non Sorted) vs Demand*** and ***Price (Non Sorted) vs Revenue***:

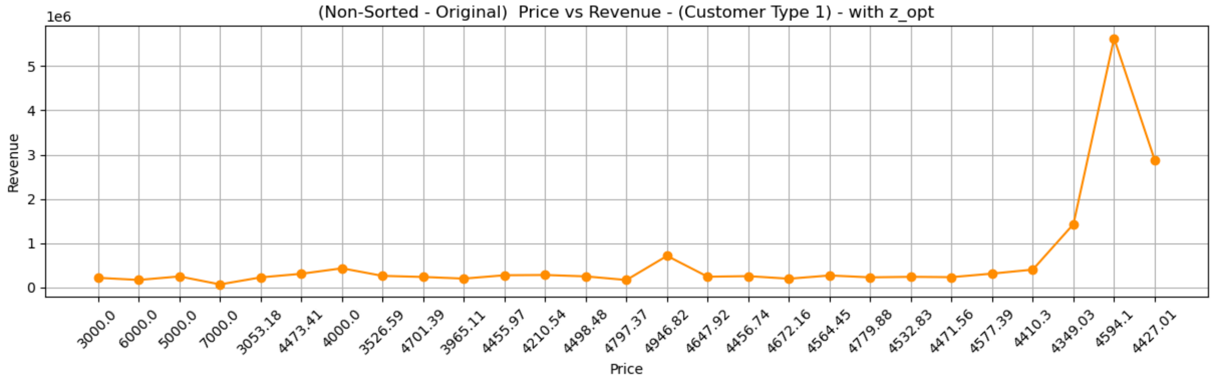
With considering ***z_opt***, Unit value of inventory, while revenue calculation.

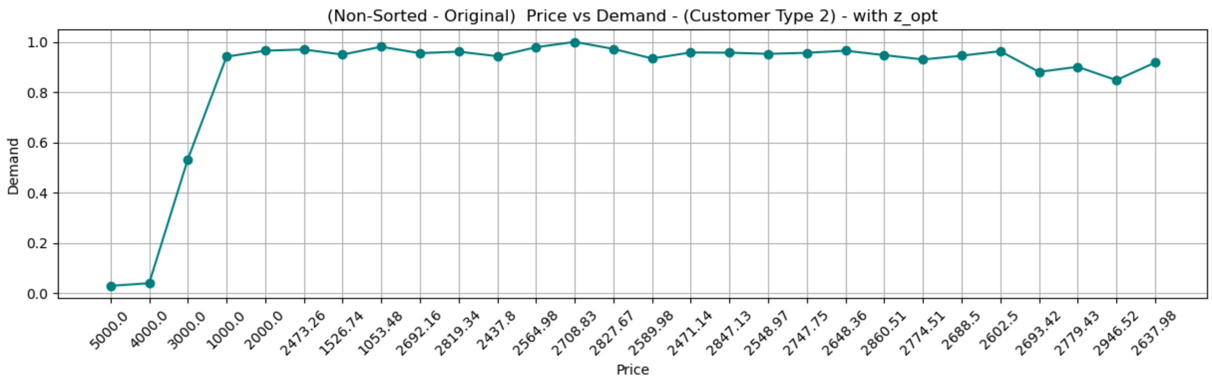
Calling plotting function with dataframes which have results (revenue and demand) taken into consideration with z_opt.

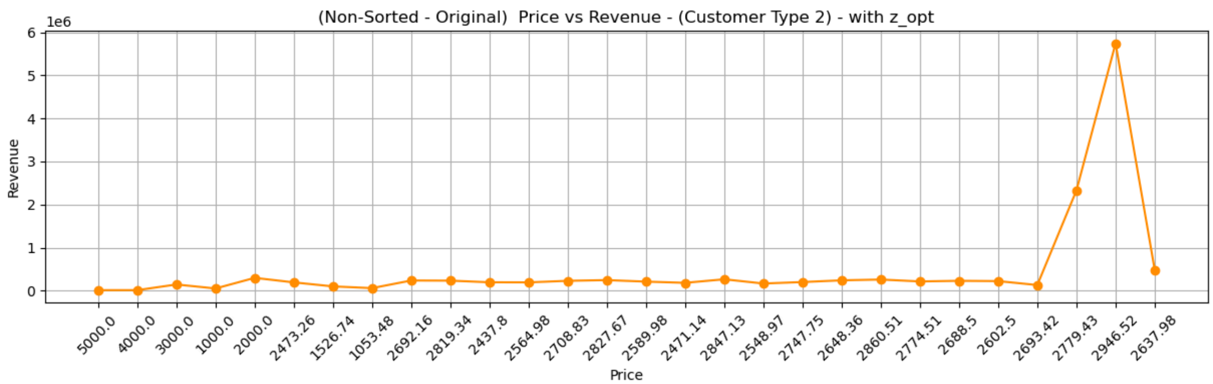
In [179…

```
# Plot non sorted prices results for Customer type 1 considering revenue cal
plot_results_z_opt(df_final_results_type_1_with_z_opt)

# Plot non sorted prices results for Customer type 2 considering revenue cal
plot_results_z_opt(df_final_results_type_2_with_z_opt)
```









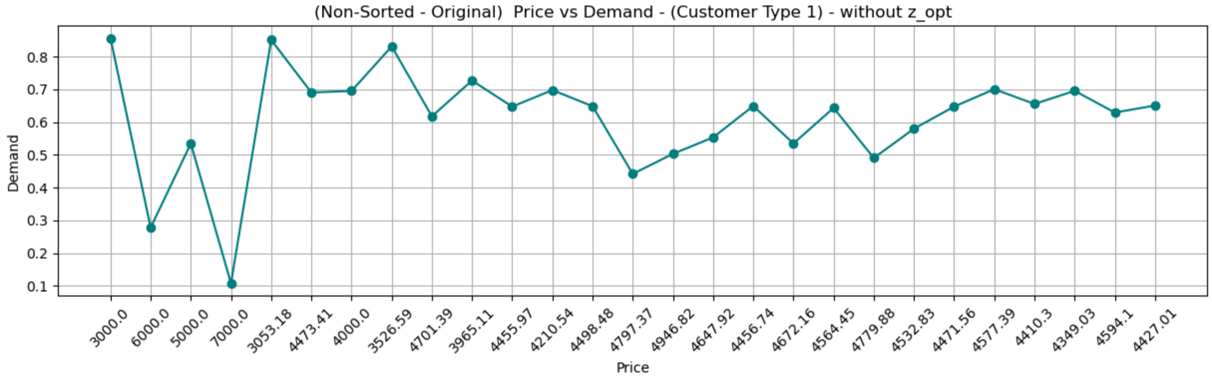
Plotting ***Price (Non Sorted) vs Demand*** and ***Price (Non Sorted) vs Revenue***:

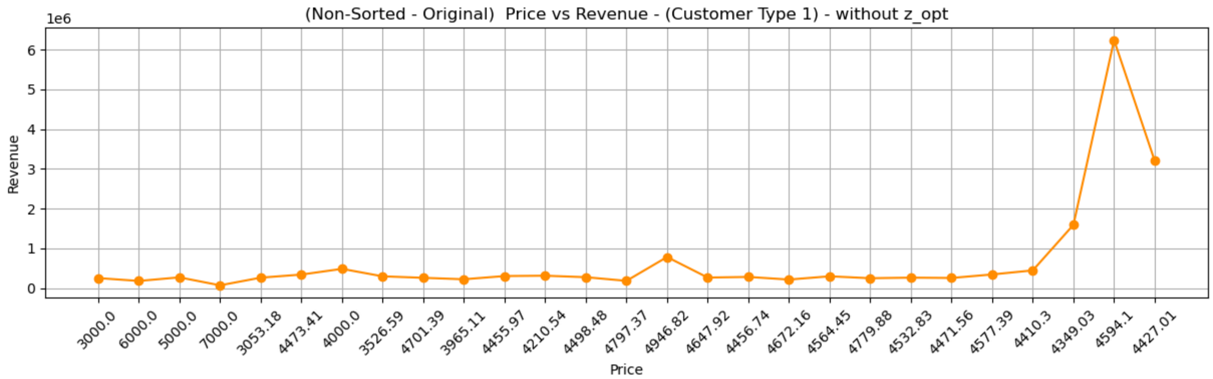
Without considering ***z_opt***, Unit value of inventory, while revenue calculation.

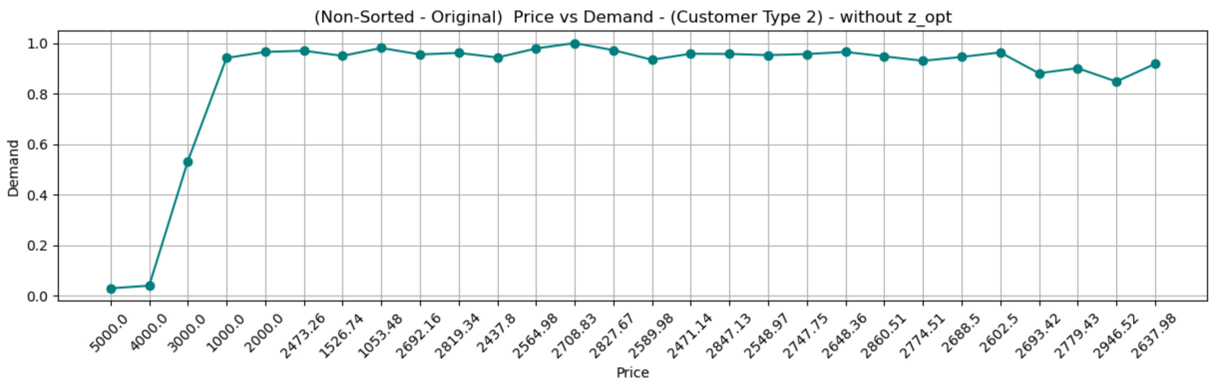
Calling plotting function with dataframes which have results (revenue and demand) taken into consideration with z_opt.

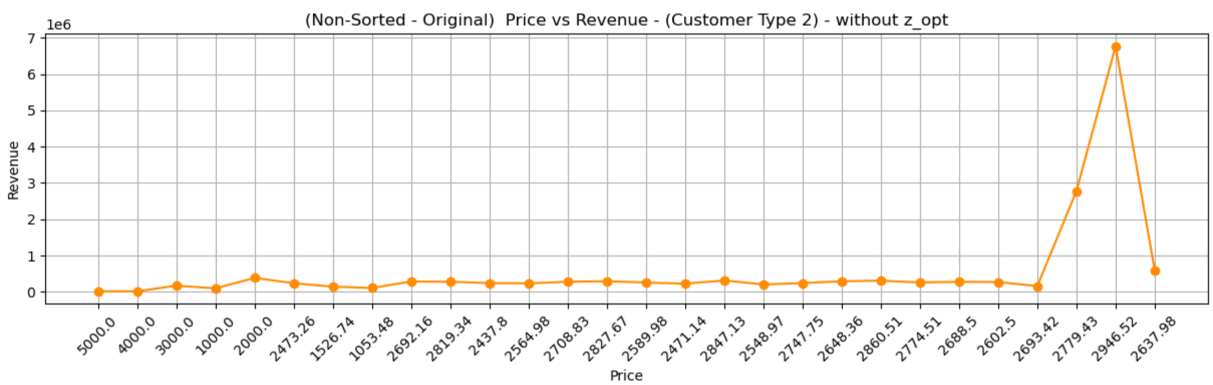
In [180…

```
# Plot non sorted prices results for Customer type 1 considering revenue cal
plot_results(df_final_results_type_1)

# Plot non sorted prices results for Customer type 2 considering revenue cal
plot_results(df_final_results_type_2)
```









END

Thank you!!