END LAB TEST

PIN:2503A52L17

Q1: Implement dynamic pricing heuristic.

• Task 1: Use AI to propose algorithms balancing demand and margin.

• Task 2: Simulate with historic sales data.

Q2: Implement order route to warehouses (optimization).

• Task 1: Use AI to produce greedy and heuristic algorithms.

• Task 2: Evaluate cost and delivery time trade-offs.

Q1:

```python
import numpy as np
from sklearn.linear_model import LinearRegression

# --- 1. Define Constants and Simulate Historical Data ---

# Cost of Goods Sold (C) - assumed constant
COST = 10.0

# Simulate historical data points (Price vs. Quantity Sold)
# This simulates the training data where we observe the market's response to different prices.
np.random.seed(42)
historical_prices = np.random.uniform(low=15, high=50, size=50).reshape(-1, 1) # 50 prices
# Simulate Demand: Q = -2 * P + 120 + noise (Demand generally decreases as Price increases)
historical_demand = (-2 * historical_prices) + 120 + np.random.normal(0, 10, size=(50, 1))
# Ensure demand is non-negative
historical_demand[historical_demand < 0] = 0

# --- 2. Train the Demand Estimation Model ---

# Instantiate and train the Linear Regression model
model = LinearRegression()
model.fit(historical_prices, historical_demand)

# Extract the learned coefficients: Q = m*P + b
# m: slope (price elasticity)
# b: y-intercept (baseline demand)
m = model.coef_[0][0]
b = model.intercept_[0]

print(f"--- Model Training Results ---")
```

```python
# The Profit function is: Pi = Q * (P - COST)
# Substitute the estimated demand function: Pi(P) = (m*P + b) * (P - COST)
# To find the maximum profit, we take the derivative w.r.t. P and set it to zero (d(Pi)/dP = 0).
# The resulting formula for the optimal price (P_optimal) is:
# P_optimal = (b - m*COST) / (-2*m)

if m < 0: # Ensure price elasticity is negative (demand decreases with price)
    P_optimal = (b - m * COST) / (-2 * m)
else:
    P_optimal = 50.0 # Set a ceiling if the model suggests an unusual relationship

# Calculate the predicted quantity and profit at the optimal price
Q_predicted = m * P_optimal + b
Profit_per_unit = P_optimal - COST
Total_Profit = Q_predicted * Profit_per_unit

# --- 4. Final Output ---

print(f"--- Optimal Pricing Heuristic Output ---")
print(f"Cost of Product (C): ${COST:.2f}")
print(f"**Calculated Optimal Price (P):** **${P_optimal:.2f}**")

if Q_predicted > 0:
    print(f"Predicted Demand (Q) at P_optimal: {Q_predicted:.0f} units")
    print(f"Profit per Unit: ${Profit_per_unit:.2f}")
    print(f"**Predicted Total Profit (Pi):** **${Total_Profit:.2f}**")
else:
    print("Predicted Demand is zero or negative at this price. Check model and constraints.")
```

OUTPUT:

```
--- Model Training Results ---
Estimated Demand Function: Q = -2.05 * P + 121.58
---
--- Optimal Pricing Heuristic Output ---
Cost of Product (C): $10.00
**Calculated Optimal Price (P):** **$34.65**
Predicted Demand (Q) at P_optimal: 51 units
Profit per Unit: $24.65
**Predicted Total Profit (Pi):** **$1245.91**
```

Q2:

CODE:

```python
import numpy as np
import pandas as pd
# --- 1. Simulation Setup (Input Data) ---
# Define the constants for the cost evaluation
SHIPPING_COST_PER_KM = 0.50  # Cost in $ per kilometer
DELIVERY_TIME_PER_KM = 0.05  # Time in hours per kilometer (e.g., 20 km/hr average speed)
ORDER_PROCESSING_TIME = 1.0  # Constant time in hours for any warehouse to process an order

# Simulated Data: Warehouse attributes
warehouses = {
    'W1': {'Distance_KM': 100, 'Capacity_Used': 800, 'Processing_Time_Factor': 0.9},
    'W2': {'Distance_KM': 50,  'Capacity_Used': 950, 'Processing_Time_Factor': 1.2},
    'W3': {'Distance_KM': 150, 'Capacity_Used': 700, 'Processing_Time_Factor': 1.0}
}
# Simulated Data: Customer Orders
orders = {
    'O1': {'Volume': 50, 'Cost_Weight': 0.7, 'Time_Weight': 0.3}, # Heavily cost-sensitive
    'O2': {'Volume': 20, 'Cost_Weight': 0.3, 'Time_Weight': 0.7}, # Heavily time-sensitive
    'O3': {'Volume': 80, 'Cost_Weight': 0.5, 'Time_Weight': 0.5}  # Balanced
}
# Convert to DataFrames for easy access (optional, but good practice)
df_orders = pd.DataFrame.from_dict(orders, orient='index')
df_warehouses = pd.DataFrame.from_dict(warehouses, orient='index')
# --- 2. Helper Function for Cost/Time Calculation ---
def calculate_metrics(warehouse_key, distance_km):
    """Calculates the raw cost and time for a given warehouse-to-order route."""
    # Cost calculation: Shipping cost + a small capacity penalty (if over 900 units)
    shipping_cost = distance_km * SHIPPING_COST_PER_KM
    capacity_penalty = 0 if df_warehouses.loc[warehouse_key, 'Capacity_Used'] < 900 else 20
```

```python
    capacity_penalty = 0 if df_warehouses.loc[warehouse_key, 'Capacity_Used'] < 900 else 20
    total_cost = shipping_cost + capacity_penalty

    # Time calculation: Processing time + Travel time
    travel_time = distance_km * DELIVERY_TIME_PER_KM
    processing_factor = df_warehouses.loc[warehouse_key, 'Processing_Time_Factor']
    total_time = ORDER_PROCESSING_TIME * processing_factor + travel_time

    return total_cost, total_time

# --- 3. ALGORITHM IMPLEMENTATION ---

def run_assignment(algorithm_type):
    """Runs the assignment logic based on the specified algorithm."""
    results = {}

    for order_name, order_data in df_orders.iterrows():
        best_warehouse = None
        min_score = float('inf')

        for w_name, w_data in df_warehouses.iterrows():

            # 3.1 Calculate raw metrics for this route
            cost, delivery_time = calculate_metrics(w_name, w_data['Distance_KM'])

            # 3.2 Determine the score based on the chosen algorithm
            if algorithm_type == 'GREEDY_COST':
                # Greedy: Minimize only the raw cost
                score = cost
```

```
                raise ValueError("Invalid algorithm type.")

            # 3.3 Check if this is the best warehouse so far
            if score < min_score:
                min_score = score
                best_warehouse = w_name
                final_cost = cost
                final_time = delivery_time

        # Store results for the order
        results[order_name] = {
            'Assigned_Warehouse': best_warehouse,
            'Total_Cost': final_cost,
            'Delivery_Time_Hrs': final_time,
            'Score': min_score
        }

    return pd.DataFrame.from_dict(results, orient='index')

# --- 4. EXECUTION AND OUTPUT ---

print("## 🚀 Assignment Results")
print("\n### 1. Greedy Algorithm: Minimize Raw Cost")
greedy_results = run_assignment('GREEDY_COST')
print(greedy_results[['Assigned_Warehouse', 'Total_Cost', 'Delivery_Time_Hrs']])

print("\n### 2. Heuristic Algorithm: Minimize Weighted Cost/Time")
heuristic_results = run_assignment('HEURISTIC_WEIGHTED')
print(heuristic_results[['Assigned_Warehouse', 'Total_Cost', 'Delivery_Time_Hrs']])
```

OUTPUT:

```
...  ## 🚀 Assignment Results

     ### 1. Greedy Algorithm: Minimize Raw Cost
        Assigned_Warehouse  Total_Cost  Delivery_Time_Hrs
     O1                 W2        45.0                3.7
     O2                 W2        45.0                3.7
     O3                 W2        45.0                3.7

     ### 2. Heuristic Algorithm: Minimize Weighted Cost/Time
        Assigned_Warehouse  Total_Cost  Delivery_Time_Hrs
     O1                 W2        45.0                3.7
     O2                 W2        45.0                3.7
     O3                 W2        45.0                3.7
```