In [1]:

```
#Task 1: Sales Data Summary
#Objective: XYZ Retail wants to automate the process of calculating basic sales met


   #- Assign the number of units sold in two categories, `Category A` and `Category
  # - Calculate the total units sold, the difference between the categories, and th
   #- Print these results clearly for the management team.



def calculate_sales_metrics(category_a_sales: int, category_b_sales: int) -> dict:
    """
    Calculate basic sales metrics for two product categories.

    Args:
        category_a_sales: Number of units sold in Category A
        category_b_sales: Number of units sold in Category B

    Returns:
        Dictionary containing calculated metrics
    """
    # Calculate metrics
    total_units = category_a_sales + category_b_sales
    sales_difference = abs(category_a_sales - category_b_sales)
    sales_ratio = category_a_sales / category_b_sales if category_b_sales != 0 else

    # Store results in dictionary
    metrics = {
        'total_units': total_units,
        'sales_difference': sales_difference,
        'sales_ratio': round(sales_ratio, 2)
    }

    return metrics

def print_sales_report(metrics: dict, category_a_sales: int, category_b_sales: int)
    """Print formatted sales report."""
    print("\nXYZ Retail Sales Report")
    print("-" * 30)
    print(f"Category A Sales: {category_a_sales}")
    print(f"Category B Sales: {category_b_sales}")
    print(f"Total Units Sold: {metrics['total_units']}")
    print(f"Sales Difference: {metrics['sales_difference']}")
    print(f"Sales Ratio (A/B): {metrics['sales_ratio']}")

if __name__ == "__main__":
    category_a_sales = 1500
    category_b_sales = 1200

    metrics = calculate_sales_metrics(category_a_sales, category_b_sales)
    print_sales_report(metrics, category_a_sales, category_b_sales)
```

```
XYZ Retail Sales Report
------------------------------
Category A Sales: 1500
Category B Sales: 1200
Total Units Sold: 2700
Sales Difference: 300
Sales Ratio (A/B): 1.25
```

In [14]:
```python
# Basic Sales Metrics

# Assign sales data to variables
category_a_sales = 1500  # Number of units sold in Category A
category_b_sales = 1200  # Number of units sold in Category B

# Print the sales data
print(f"Category A Sales: {category_a_sales}")
print(f"Category B Sales: {category_b_sales}")
```

```
Category A Sales: 1500
Category B Sales: 1200
```

In [16]:
```python
# Assign sales data to variables
category_a_sales = 1500  # Number of units sold in Category A
category_b_sales = 1200  # Number of units sold in Category B

# Calculate metrics
total_units = category_a_sales + category_b_sales
sales_difference = abs(category_a_sales - category_b_sales)
sales_ratio = category_a_sales / category_b_sales if category_b_sales != 0 else 0

# Print results
print(f"Total Units Sold: {total_units}")
print(f"Difference between Categories: {sales_difference}")
print(f"Sales Ratio (A/B): {sales_ratio:.2f}")
```

```
Total Units Sold: 2700
Difference between Categories: 300
Sales Ratio (A/B): 1.25
```

In [104…
```python
#Task 2: Customer Age Data
#Objective: Understanding the age distribution of customers is crucial for marketin


   #- Store a customer's name and age.
   #- Convert the age into a string and create a personalized marketing message lik
  # - Print the message for use in email campaigns.


def create_marketing_message(name: str, age: int) -> str:
    """Generate personalized marketing message based on customer age."""

    if age >= 60:
        program = "senior rewards program"
    elif age >= 30:
        program = "premium loyalty program"
    else:
        program = "young adult rewards program"
```

```
    message = f"Dear {name}, at {age}, you're eligible for our {program}."
    return message


customer_name = "John Doe"
customer_age = 30

# Generate and print marketing message
message = create_marketing_message(customer_name, customer_age)
print(message)
```

```
Dear John Doe, at 30, you're eligible for our premium loyalty program.
```

In [106…

```
#Task 3: Product List Management
#Objective: Efficient management of the product list is essential for inventory con


   #- Given a list of product prices, extract the highest and lowest prices.
  # - Create a new list with the mid-range products.
   #- Add a new premium product price to the list and print the updated list for th


def analyze_product_prices(prices: list, new_premium_price: float, mid_range_buffer
    """Analyze and update product prices."""

    # Find highest and lowest prices
    min_price = min(prices)
    max_price = max(prices)

    # Calculate mid-range products (within buffer of average price)
    avg_price = sum(prices) / len(prices)
    mid_range = [p for p in prices if (avg_price - mid_range_buffer) <= p <= (avg_p

    # Add new premium product
    updated_prices = prices + [new_premium_price]

    return min_price, max_price, mid_range, updated_prices


product_prices = [99.99, 149.99, 199.99, 249.99, 299.99]
new_premium_price = 399.99

min_price, max_price, mid_range, updated_prices = analyze_product_prices(product_pr

# Print results
print(f"Lowest Price: ${min_price}")
print(f"Highest Price: ${max_price}")
print(f"Mid-Range Products: ${mid_range}")
print(f"Updated Product List: ${updated_prices}")
```

```
Lowest Price: $99.99
Highest Price: $299.99
Mid-Range Products: $[149.99, 199.99, 249.99]
Updated Product List: $[99.99, 149.99, 199.99, 249.99, 299.99, 399.99]
```

In [108…

```python
#Task 4: Inventory Lookup
#Objective: Quick access to product details is important for customer service repre


   #- Create a dictionary storing key information about a product (e.g., `product_n
  # - Print the product name and SKU when queried by a customer service representat


def create_product_database():
    """Create a dictionary of product information."""
    return {
        "LAPTOP001": {
            "product_name": "Ultra Slim Laptop",
            "SKU": "LAPTOP001",
            "price": 999.99,
            "category": "Electronics"
        },
        "PHONE002": {
            "product_name": "SmartPhone Pro",
            "SKU": "PHONE002",
            "price": 799.99,
            "category": "Mobile Devices"
        }
    }

def lookup_product(sku: str, products: dict):
    """Look up product information by SKU."""
    if sku in products:
        product = products[sku]
        print(f"Product Name: {product['product_name']}")
        print(f"SKU: {product['SKU']}")
    else:
        print(f"Product with SKU {sku} not found.")


products = create_product_database()
lookup_product("LAPTOP001", products)
```

```
Product Name: Ultra Slim Laptop
SKU: LAPTOP001
```

In [110…

```python
#Task 5: Stock Level Alert System
#Objective: Ensuring that stock levels are maintained is critical to avoid stockout


   #- Write the code that takes the stock level as input.
  # - If stock is below a certain threshold, print a "Reorder Now" alert. If stock


def check_stock_level(current_stock: int, reorder_threshold: int = 100):
    """
    Monitor stock levels and generate alerts.

    Args:
        current_stock: Current quantity in stock
        reorder_threshold: Minimum stock level before reorder
```

```
    """
    if current_stock <= reorder_threshold:
        print(f"ALERT: Reorder Now! Current stock ({current_stock}) is below thresh
    else:
        print(f"Stock is sufficient. Current level: {current_stock}")


current_stock = 75
check_stock_level(current_stock)

# Test with different stock levels
test_stocks = [150, 50, 100]
for stock in test_stocks:
    check_stock_level(stock)
```

```
ALERT: Reorder Now! Current stock (75) is below threshold (100)
Stock is sufficient. Current level: 150
ALERT: Reorder Now! Current stock (50) is below threshold (100)
ALERT: Reorder Now! Current stock (100) is below threshold (100)
```

In [112…

```
#Task 6: Sales Report Formatting
#Objective: Formatting the sales data for management reports is crucial.


   #- Given a list of products sold, print each product name in uppercase for bette
  # - Implement both a `for` loop and a `while` loop for this task to ensure code f


def format_sales_data(products: list):
    """Format product names using for and while loops."""

    # Using for loop
    print("Using For Loop:")
    for product in products:
        print(product.upper())

    # Using while loop
    print("\nUsing While Loop:")
    i = 0
    while i < len(products):
        print(products[i].upper())
        i += 1


products_sold = ["laptop", "smartphone", "tablet", "smartwatch"]
format_sales_data(products_sold)
```

```
Using For Loop:
LAPTOP
SMARTPHONE
TABLET
SMARTWATCH

Using While Loop:
LAPTOP
SMARTPHONE
TABLET
SMARTWATCH
```

In [114…

```python
#Task 7: Area Calculation for Store Layout
#Objective: Accurate area calculations are needed to plan new store layouts.


   #- Create a function that calculates the area of a section of the store based on
   #- Use this function to calculate and print the area of several store sections.


def calculate_section_area(length: float, width: float) -> float:
    """Calculate area of store section."""
    return length * width

def print_section_areas(sections: dict):
    """Print areas for multiple store sections."""
    for section, dimensions in sections.items():
        area = calculate_section_area(dimensions['length'], dimensions['width'])
        print(f"{section} Area: {area} square feet")

store_sections = {
    'Electronics': {'length': 30, 'width': 20},
    'Clothing': {'length': 40, 'width': 35},
    'Groceries': {'length': 50, 'width': 30}
}

print_section_areas(store_sections)
```

```
Electronics Area: 600 square feet
Clothing Area: 1400 square feet
Groceries Area: 1500 square feet
```

In [116…

```python
#Task 8: Customer Feedback Analysis
#Objective: Analyzing customer feedback is vital to improving service.


   #- Write the code to count the number of vowels in a customer feedback message.
   #- Also, reverse the feedback message for a unique data presentation in reports.


def analyze_feedback(feedback: str):
    """Analyze customer feedback for vowels and create reverse message."""


    vowels = 'aeiouAEIOU'
    vowel_count = sum(1 for char in feedback if char in vowels)
```

```
    reversed_feedback = feedback[::-1]

    return vowel_count, reversed_feedback


feedback = "Great customer service and products!"
vowels, reversed_msg = analyze_feedback(feedback)

print(f"Vowel Count: {vowels}")
print(f"Reversed Message: {reversed_msg}")
```

```
Vowel Count: 11
Reversed Message: !stcudorp dna ecivres remotsuc taerG
```

In [118…

```
#Task 9: Price Filtering Tool
#Objective: Filtering product prices helps in creating targeted discounts.


  #- Use list comprehension to filter out products priced below a certain threshold
   #- Print the list of eligible products for a discount campaign.


def filter_products(products: list, min_price: float):
    """Filter products above price threshold."""


    product_list = [
         {"name": "Laptop", "price": 999},
        {"name": "Phone", "price": 499},
        {"name": "Tablet", "price": 700},
        {"name": "Smartwatch", "price": 199}
    ]


    eligible_products = [product for product in product_list if product["price"] >=


    print(f"Products eligible for discount (Min price: ${min_price}):")
    for product in eligible_products:
        print(f"{product['name']}: ${product['price']}")


filter_products(products=None, min_price=400.00)
```

```
Products eligible for discount (Min price: $400.0):
Laptop: $999
Phone: $499
Tablet: $700
```

In [120…

```
#Task 10: Sales Log File Management
#Objective: Proper management of sales log files is necessary for compliance and au


   #- Create a text file named `sales_log.txt` to store daily sales summaries.
   #-write two lines summarizing the daily sales performance.
```

```
   #- Read and print the content of the file to ensure data integrity.


def manage_sales_log():
    """Create, write to, and read from sales log file."""

    # Write to file
    with open('sales_log.txt', 'w') as file:
        file.write("Daily Sales Summary - 2024-12-04\n")
        file.write("Total Sales: $15,750 | Units Sold: 125 | Average Transaction: $

    # Read and print file contents
    with open('sales_log.txt', 'r') as file:
        content = file.read()
        print("File Contents:")
        print(content)


manage_sales_log()
```

```
File Contents:
Daily Sales Summary - 2024-12-04
Total Sales: $15,750 | Units Sold: 125 | Average Transaction: $126
```

In [122…

```
# Task 11: Daily Sales Average
#Objective: Calculate the average daily sales for the past week.
    #Given a list of sales figures for the last 7 days, calculate the average sales
    #Print the average sales to help the finance team understand the weekly perform

daily_sales = [1200.50, 950.75, 1500.25, 1100.00, 1350.50, 800.25, 1425.75]

average_sales = sum(daily_sales) / len(daily_sales)

print(f"Daily sales for the past week: ${daily_sales}")
print(f"Average daily sales: ${average_sales:.2f}")
```

```
Daily sales for the past week: $[1200.5, 950.75, 1500.25, 1100.0, 1350.5, 800.25, 14
25.75]
Average daily sales: $1189.71
```

In [128…

```
#Task 12: Customer Segmentation
#Objective: Categorize customers based on their total spending.
    #Create a list of customer spending amounts.
    #Use a loop to categorize customers as "Low", "Medium", or "High" spenders base
    #Print the categorized results to assist in targeted marketing.

customer_spending = [
    {"customer_id": 1, "spending": 150.75},
    {"customer_id": 2, "spending": 850.25},
    {"customer_id": 3, "spending": 1500.50},
    {"customer_id": 4, "spending": 300.00},
    {"customer_id": 5, "spending": 2200.75},
    {"customer_id": 6, "spending": 450.25}
]

LOW_THRESHOLD = 500
```

```
HIGH_THRESHOLD = 1000

customer_categories = {
    "Low": [],
    "Medium": [],
    "High": []
}

for customer in customer_spending:
    spending = customer["spending"]
    customer_id = customer["customer_id"]

    if spending < LOW_THRESHOLD:
        customer_categories["Low"].append(customer_id)
    elif spending < HIGH_THRESHOLD:
        customer_categories["Medium"].append(customer_id)
    else:
        customer_categories["High"].append(customer_id)


print("Customer Spending Categories:")
print("-" * 30)
for category, customers in customer_categories.items():
    print(f"{category} Spenders (Customer IDs): {customers}")


print("\nDetailed Customer Breakdown:")
print("-" * 30)
for customer in customer_spending:
    spending = customer["spending"]
    category = "Low" if spending < LOW_THRESHOLD else "Medium" if spending < HIGH_T
    print(f"Customer {customer['customer_id']}: ${spending:.2f} - {category} Spende
```

```
Customer Spending Categories:
------------------------------
Low Spenders (Customer IDs): [1, 4, 6]
Medium Spenders (Customer IDs): [2]
High Spenders (Customer IDs): [3, 5]

Detailed Customer Breakdown:
------------------------------
Customer 1: $150.75 - Low Spender
Customer 2: $850.25 - Medium Spender
Customer 3: $1500.50 - High Spender
Customer 4: $300.00 - Low Spender
Customer 5: $2200.75 - High Spender
Customer 6: $450.25 - Low Spender
```

In [126…

```
#Task 13: Discount Calculation
#Objective: Automate the calculation of discounts for a promotional campaign.
     #Write a code that calculates the final price after applying a discount percen
     #Test this function on a list of products with different discounts and print t


def calculate_discount(original_price, discount_percentage):
    """
```

```
    Calculate the final price after applying a discount

    Args:
        original_price (float): The original price of the product
        discount_percentage (float): The discount percentage (0-100)

    Returns:
        float: The final price after discount
    """
    if not 0 <= discount_percentage <= 100:
        raise ValueError("Discount percentage must be between 0 and 100")

    discount_amount = original_price * (discount_percentage / 100)
    final_price = original_price - discount_amount
    return round(final_price, 2)

# List of products with their original prices and discount percentages
products = [
    {"name": "Laptop", "price": 999.99, "discount": 15},
    {"name": "Headphones", "price": 149.99, "discount": 25},
    {"name": "Mouse", "price": 49.99, "discount": 10},
    {"name": "Keyboard", "price": 89.99, "discount": 20},
    {"name": "Monitor", "price": 299.99, "discount": 30}
]

# Calculate and display discounted prices for each product
print("Promotional Campaign Discounts")
print("-" * 50)
print(f"{'Product':<15} {'Original':<10} {'Discount':<10} {'Final Price':<10}")
print("-" * 50)

for product in products:
    name = product["name"]
    original_price = product["price"]
    discount = product["discount"]

    try:
        final_price = calculate_discount(original_price, discount)
        print(f"{name:<15} ${original_price:<9.2f} {discount:>3}%       ${final_pri
    except ValueError as e:
        print(f"Error processing {name}: {e}")

print("-" * 50)
```

```
Promotional Campaign Discounts
--------------------------------------------------
Product         Original   Discount   Final Price
--------------------------------------------------
Laptop          $999.99     15%       $849.99
Headphones      $149.99     25%       $112.49
Mouse           $49.99      10%       $44.99
Keyboard        $89.99      20%       $71.99
Monitor         $299.99     30%       $209.99
--------------------------------------------------
```

In [130…

```python
#Task 14: Customer Feedback Sentiment Analysis
#Objective: Basic sentiment analysis of customer feedback.
    #Write the Python code that checks if certain positive or negative words (e.g.,
    #Print "Positive" or "Negative" based on the words found in the feedback.


def analyze_sentiment(feedback):
    """
    Analyze the sentiment of customer feedback based on keyword matching

    Args:
        feedback (str): The customer feedback text

    Returns:
        str: The sentiment classification ('Positive', 'Negative', or 'Neutral')
    """
    # Convert feedback to lowercase for case-insensitive matching
    feedback = feedback.lower()


    positive_words = {
        'good', 'great', 'excellent', 'amazing', 'wonderful', 'happy',
        'satisfied', 'love', 'perfect', 'fantastic', 'awesome',
        'helpful', 'recommended', 'best', 'impressed'
    }

    negative_words = {
        'bad', 'poor', 'terrible', 'awful', 'disappointed', 'unhappy',
        'dissatisfied', 'hate', 'worst', 'horrible', 'useless',
        'frustrating', 'waste', 'annoying', 'refund'
    }


    positive_count = sum(1 for word in positive_words if word in feedback)
    negative_count = sum(1 for word in negative_words if word in feedback)


    if positive_count > negative_count:
        return 'Positive'
    elif negative_count > positive_count:
        return 'Negative'
    else:
        return 'Neutral'


customer_feedback = [
    "The product is amazing and I'm very satisfied with my purchase!",
    "This is the worst experience ever, I want a refund.",
    "The product is okay, nothing special about it.",
    "I love how helpful the customer service team was!",
    "Disappointed with the quality, wouldn't recommend."
]


print("Customer Feedback Sentiment Analysis")
```

```
print("-" * 50)

for i, feedback in enumerate(customer_feedback, 1):
    sentiment = analyze_sentiment(feedback)
    print(f"\nFeedback {i}:")
    print(f"Text: {feedback}")
    print(f"Sentiment: {sentiment}")
```

```
Customer Feedback Sentiment Analysis
--------------------------------------------------

Feedback 1:
Text: The product is amazing and I'm very satisfied with my purchase!
Sentiment: Positive

Feedback 2:
Text: This is the worst experience ever, I want a refund.
Sentiment: Negative

Feedback 3:
Text: The product is okay, nothing special about it.
Sentiment: Neutral

Feedback 4:
Text: I love how helpful the customer service team was!
Sentiment: Positive

Feedback 5:
Text: Disappointed with the quality, wouldn't recommend.
Sentiment: Negative
```

In [67]:
```
#Task 15: Employee Salary Increment Calculator
#Objective: Calculate the salary increment for employees based on their performance
    #Create a dictionary that stores employee names and their performance ratings.
    #Write the code that applies a different increment percentage based on the rati
    #Print the updated salary for each employee.


def calculate_increment(current_salary, performance_rating):
    """
    Calculate salary increment based on performance rating

    Args:
        current_salary (float): Current salary of the employee
        performance_rating (str): Performance rating (A, B, C, D)

    Returns:
        float: New salary after increment
    """
    increment_rates = {
        'A': 15,  # 15% increment for outstanding performance
        'B': 10,  # 10% increment for good performance
        'C': 5,   # 5% increment for average performance
        'D': 0    # No increment for below average performance
    }
```

```
    if performance_rating not in increment_rates:
        raise ValueError(f"Invalid performance rating: {performance_rating}")

    increment_percentage = increment_rates[performance_rating]
    increment_amount = current_salary * (increment_percentage / 100)
    new_salary = current_salary + increment_amount

    return round(new_salary, 2)


employees = {
    'John Smith': {'salary': 50000, 'rating': 'A'},
    'Emma Wilson': {'salary': 45000, 'rating': 'B'},
    'Michael Brown': {'salary': 55000, 'rating': 'A'},
    'Sarah Davis': {'salary': 48000, 'rating': 'C'},
    'James Johnson': {'salary': 52000, 'rating': 'B'},
    'Lisa Anderson': {'salary': 47000, 'rating': 'D'}
}


print("Employee Salary Increment Report")
print("-" * 65)
print(f"{'Employee Name':<20} {'Current':<12} {'Rating':<8} {'Increment':<10} {'New
print("-" * 65)

for name, data in employees.items():
    current_salary = data['salary']
    rating = data['rating']

    try:
        new_salary = calculate_increment(current_salary, rating)
        increment_amount = new_salary - current_salary

        print(f"{name:<20} ${current_salary:<11,} {rating:<8} ${increment_amount:<9
    except ValueError as e:
        print(f"Error processing {name}: {e}")

print("-" * 65)
```

```
Employee Salary Increment Report
-----------------------------------------------------------------
Employee Name        Current      Rating   Increment  New Salary
-----------------------------------------------------------------
John Smith           $50,000      A        $7,500.0   $57,500.0
Emma Wilson          $45,000      B        $4,500.0   $49,500.0
Michael Brown        $55,000      A        $8,250.0   $63,250.0
Sarah Davis          $48,000      C        $2,400.0   $50,400.0
James Johnson        $52,000      B        $5,200.0   $57,200.0
Lisa Anderson        $47,000      D        $0.0       $47,000.0
-----------------------------------------------------------------
```

In [132…

```
#Task 16: Monthly Sales Report Generator
#Objective: Generate a simple text-based monthly sales report.
    #Create a list of daily sales figures for a month.
    #Calculate the total and average sales for the month.
    #Write these statistics to a text file named monthly_report.txt.
```

```
from datetime import datetime
import random  # Used to generate sample data

def generate_monthly_report(sales_data):
    """
    Generate a monthly sales report from daily sales data

    Args:
        sales_data (dict): Dictionary with dates and sales figures

    Returns:
        tuple: Total sales, average sales, highest day, lowest day
    """
    total_sales = sum(sales_data.values())
    average_sales = total_sales / len(sales_data)
    highest_day = max(sales_data.items(), key=lambda x: x[1])
    lowest_day = min(sales_data.items(), key=lambda x: x[1])

    return total_sales, average_sales, highest_day, lowest_day


current_date = datetime.now()
month_name = current_date.strftime("%B %Y")


daily_sales = {
    f"{current_date.replace(day=day).strftime('%Y-%m-%d')}":
    round(random.uniform(500, 2000), 2)
    for day in range(1, 31)
}


total_sales, average_sales, highest_day, lowest_day = generate_monthly_report(daily


report_content = f"""
Monthly Sales Report - {month_name}
{'='* 50}

Summary Statistics:
-----------------
Total Monthly Sales: ${total_sales:,.2f}
Average Daily Sales: ${average_sales:,.2f}
Best Performing Day: {highest_day[0]} (${highest_day[1]:,.2f})
Lowest Performing Day: {lowest_day[0]} (${lowest_day[1]:,.2f})

Daily Sales Breakdown:
-------------------"""


for date, amount in daily_sales.items():
    report_content += f"\n{date}: ${amount:,.2f}"
```

```
report_filename = "monthly_report.txt"
try:
    with open(report_filename, 'w') as file:
        file.write(report_content)
    print(f"Report has been generated successfully: {report_filename}")


    print("\nReport Preview:")
    print(report_content)

except IOError as e:
    print(f"Error writing report to file: {e}")
```

```
Report has been generated successfully: monthly_report.txt

Report Preview:

Monthly Sales Report - December 2024
==================================================

Summary Statistics:
-----------------
Total Monthly Sales: $36,998.67
Average Daily Sales: $1,233.29
Best Performing Day: 2024-12-07 ($1,945.60)
Lowest Performing Day: 2024-12-26 ($534.83)

Daily Sales Breakdown:
-------------------
2024-12-01: $1,082.00
2024-12-02: $1,382.16
2024-12-03: $1,181.79
2024-12-04: $558.06
2024-12-05: $692.88
2024-12-06: $1,933.48
2024-12-07: $1,945.60
2024-12-08: $1,061.63
2024-12-09: $1,622.28
2024-12-10: $543.95
2024-12-11: $901.92
2024-12-12: $1,389.66
2024-12-13: $1,634.22
2024-12-14: $1,342.64
2024-12-15: $1,476.81
2024-12-16: $1,589.22
2024-12-17: $1,261.77
2024-12-18: $1,516.25
2024-12-19: $572.85
2024-12-20: $1,074.49
2024-12-21: $1,138.47
2024-12-22: $731.15
2024-12-23: $1,655.83
2024-12-24: $1,859.03
2024-12-25: $1,382.13
2024-12-26: $534.83
2024-12-27: $1,244.16
2024-12-28: $1,381.29
2024-12-29: $1,557.78
2024-12-30: $750.34
```

In [134…

```python
#Task 17: Stock Replenishment Planning
#Objective: Determine which products need replenishment based on sales data.
    #Given a list of products and their current stock levels, compare these against
    #Print a list of products that need to be reordered to maintain adequate stock


def check_inventory_levels(inventory, min_threshold, reorder_quantity):
    """
    Check inventory levels and determine which products need replenishment
```

```
    Args:
        inventory (dict): Dictionary of products with their current stock levels
        min_threshold (dict): Minimum stock levels for each product
        reorder_quantity (dict): Recommended reorder quantity for each product

    Returns:
        list: Products that need replenishment
    """
    products_to_reorder = []

    for product, stock in inventory.items():
        if stock <= min_threshold.get(product, 0):
            products_to_reorder.append({
                'product': product,
                'current_stock': stock,
                'threshold': min_threshold.get(product, 0),
                'reorder_quantity': reorder_quantity.get(product, 0)
            })

    return products_to_reorder


current_inventory = {
    'Laptop': 12,
    'Smartphone': 8,
    'Tablet': 5,
    'Headphones': 15,
    'Mouse': 3,
    'Keyboard': 6,
    'Monitor': 4
}


minimum_threshold = {
    'Laptop': 10,
    'Smartphone': 15,
    'Tablet': 8,
    'Headphones': 20,
    'Mouse': 5,
    'Keyboard': 10,
    'Monitor': 5
}


reorder_quantities = {
    'Laptop': 15,
    'Smartphone': 25,
    'Tablet': 12,
    'Headphones': 30,
    'Mouse': 20,
    'Keyboard': 15,
    'Monitor': 8
}
```

```
products_needed = check_inventory_levels(
    current_inventory,
    minimum_threshold,
    reorder_quantities
)


print("Inventory Replenishment Report")
print("=" * 70)

if products_needed:
    print("\nProducts that need reordering:")
    print("-" * 70)
    print(f"{'Product':<15} {'Current Stock':<15} {'Min Threshold':<15} {'Reorder Q
    print("-" * 70)

    for item in products_needed:
        print(f"{item['product']:<15} {item['current_stock']:<15} "
              f"{item['threshold']:<15} {item['reorder_quantity']:<15}")

    print("\nTotal products to reorder:", len(products_needed))
else:
    print("\nAll products are above minimum threshold levels. No reordering needed.


total_items_to_order = sum(item['reorder_quantity'] for item in products_needed)
print(f"\nTotal units to be ordered: {total_items_to_order}")
```

```
Inventory Replenishment Report
======================================================================

Products that need reordering:
----------------------------------------------------------------------
Product         Current Stock   Min Threshold   Reorder Qty
----------------------------------------------------------------------
Smartphone      8               15              25
Tablet          5               8               12
Headphones      15              20              30
Mouse           3               5               20
Keyboard        6               10              15
Monitor         4               5               8

Total products to reorder: 6

Total units to be ordered: 110
```

In [136…

```
print("Good Morning")
```

```
Good Morning
```

In [138…

```
#Task 18: Data Cleaning Utility
#Objective: Create a utility to clean customer names for better data consistency.
    #Write the code that takes a list of customer names with extra spaces and incon
    #Clean the names by trimming spaces and standardizing the capitalization (e.g.,
    #Print the cleaned names for database entry.

def clean_customer_name(name):
```

```
    """
    Clean and standardize a customer name

    Args:
        name (str): Raw customer name

    Returns:
        str: Cleaned and standardized name
    """
    if not isinstance(name, str):
        return "Invalid Name"


    cleaned_name = " ".join(name.split()).title()


    special_cases = {
        "Mc": lambda x: "Mc" + x[2:].capitalize(),  # McDonald -> McDonald
        "Mac": lambda x: "Mac" + x[3:].capitalize(), # MacArthur -> MacArthur
        "O'": lambda x: "O'" + x[2:].capitalize()    # O'BRIEN -> O'Brien
    }

    for prefix, handler in special_cases.items():
        if cleaned_name.startswith(prefix):
            cleaned_name = handler(cleaned_name)

    return cleaned_name


customer_names = [
    "   john    smith   ",
    "MARY JONES",
    "bob     WILSON",
    "Sarah   O'CONNOR",
    "  TOM    McDonald  ",
    "alice      MacARTHUR",
    "PETER        BROWN   ",
    "  susan     O'BRIEN  ",
    "   mike  johnson    "
]


print("Customer Name Standardization Report")
print("-" * 50)
print(f"{'Original Name':<30} {'Cleaned Name':<30}")
print("-" * 50)

cleaned_names = []
for name in customer_names:
    cleaned = clean_customer_name(name)
    cleaned_names.append(cleaned)
    print(f"{name:<30} {cleaned:<30}")

print("\nSummary:")
print("-" * 50)
print(f"Total names processed: {len(customer_names)}")
```

```
print(f"Unique names after cleaning: {len(set(cleaned_names))}")


print("\nFormatted for Database Entry:")
print("-" * 50)
for i, name in enumerate(cleaned_names, 1):
    print(f"INSERT INTO customers (customer_name) VALUES ('{name}');")
```

```
Customer Name Standardization Report
--------------------------------------------------
Original Name                  Cleaned Name
--------------------------------------------------
   john    smith               John Smith
MARY JONES                     Mary Jones
bob     WILSON                 Bob Wilson
Sarah   O'CONNOR               Sarah O'Connor
  TOM    McDonald              Tom Mcdonald
alice      MacARTHUR           Alice Macarthur
PETER        BROWN             Peter Brown
  susan     O'BRIEN            Susan O'Brien
   mike  johnson                Mike Johnson

Summary:
--------------------------------------------------
Total names processed: 9
Unique names after cleaning: 9

Formatted for Database Entry:
--------------------------------------------------
INSERT INTO customers (customer_name) VALUES ('John Smith');
INSERT INTO customers (customer_name) VALUES ('Mary Jones');
INSERT INTO customers (customer_name) VALUES ('Bob Wilson');
INSERT INTO customers (customer_name) VALUES ('Sarah O'Connor');
INSERT INTO customers (customer_name) VALUES ('Tom Mcdonald');
INSERT INTO customers (customer_name) VALUES ('Alice Macarthur');
INSERT INTO customers (customer_name) VALUES ('Peter Brown');
INSERT INTO customers (customer_name) VALUES ('Susan O'Brien');
INSERT INTO customers (customer_name) VALUES ('Mike Johnson');
```

In [140…

```
#Task 19: Simple Sales Forecasting
#Objective: Implement a basic forecasting model for next month’s sales.
    #Based on the average sales of the last 3 months, predict next month’s sales us
    #Print the forecasted sales figures for budget planning.


from datetime import datetime, timedelta
import calendar

def calculate_sales_forecast(monthly_sales, months_to_consider=3):
    """
    Calculate sales forecast based on previous months' data

    Args:
        monthly_sales (dict): Dictionary of monthly sales data
        months_to_consider (int): Number of months to use for prediction
```

```
    Returns:
        float: Forecasted sales for next month
    """

    recent_sales = list(monthly_sales.values())[-months_to_consider:]


    forecast = sum(recent_sales) / len(recent_sales)


    if len(recent_sales) >= 2:
        trend = (recent_sales[-1] - recent_sales[0]) / (len(recent_sales) - 1)
        forecast += trend  # Adjust forecast based on trend

    return round(forecast, 2)


monthly_sales = {
    "2024-01": 45250.75,
    "2024-02": 48750.25,
    "2024-03": 52500.50
}


forecast = calculate_sales_forecast(monthly_sales)


print("Sales Forecast Report")
print("=" * 50)


print("\nHistorical Sales Data:")
print("-" * 50)
print(f"{'Month':<15} {'Sales':<15}")
print("-" * 50)

for month, sales in monthly_sales.items():
    print(f"{month:<15} ${sales:,.2f}")


avg_sales = sum(monthly_sales.values()) / len(monthly_sales)
min_sales = min(monthly_sales.values())
max_sales = max(monthly_sales.values())


last_month = max(monthly_sales.keys())
year, month = map(int, last_month.split("-"))
next_month = datetime(year, month, 1) + timedelta(days=32)
next_month_str = next_month.strftime("%Y-%m")


print("\nForecast Analysis")
print("-" * 50)
print(f"Average Historical Sales: ${avg_sales:,.2f}")
print(f"Minimum Monthly Sales:   ${min_sales:,.2f}")
print(f"Maximum Monthly Sales:   ${max_sales:,.2f}")
```

```
print(f"\nForecast for {next_month_str}:  ${forecast:,.2f}")


variance = max_sales - min_sales
confidence_range = (forecast - variance/2, forecast + variance/2)

print("\nConfidence Range:")
print("-" * 50)
print(f"Lower Bound: ${confidence_range[0]:,.2f}")
print(f"Upper Bound: ${confidence_range[1]:,.2f}")


print("\nInsights:")
print("-" * 50)
if forecast > avg_sales:
    print("- Forecast indicates potential growth compared to historical average")
else:
    print("- Forecast indicates potential decline compared to historical average")

if forecast > max_sales:
    print("- Forecasted sales exceed historical maximum - consider validating forec
elif forecast < min_sales:
    print("- Forecasted sales below historical minimum - consider validating foreca
```

```
Sales Forecast Report
==================================================

Historical Sales Data:
--------------------------------------------------
Month           Sales
--------------------------------------------------
2024-01         $45,250.75
2024-02         $48,750.25
2024-03         $52,500.50

Forecast Analysis
--------------------------------------------------
Average Historical Sales: $48,833.83
Minimum Monthly Sales:   $45,250.75
Maximum Monthly Sales:   $52,500.50

Forecast for 2024-04:  $52,458.71

Confidence Range:
--------------------------------------------------
Lower Bound: $48,833.83
Upper Bound: $56,083.58

Insights:
--------------------------------------------------
- Forecast indicates potential growth compared to historical average
```

In [142…

```
#Task 20: Customer Loyalty Points Calculator
#Objective: Calculate loyalty points for customers based on their purchases.
    #Write a code that assigns loyalty points to customers based on their total pur
    #Implement a tiered system where different spending levels earn different point
```

```
    #Print the loyalty points for a list of customers.


def calculate_loyalty_points(purchase_amount):
    """
    Calculate loyalty points based on purchase amount using a tiered system

    Args:
        purchase_amount (float): Total purchase amount

    Returns:
        int: Loyalty points earned
    """
    tier_system = [
        {'min_spend': 1000, 'multiplier': 3.0, 'name': 'Platinum'},
        {'min_spend': 500, 'multiplier': 2.0, 'name': 'Gold'},
        {'min_spend': 100, 'multiplier': 1.5, 'name': 'Silver'},
        {'min_spend': 0, 'multiplier': 1.0, 'name': 'Bronze'}
    ]


    base_points = purchase_amount


    for tier in tier_system:
        if purchase_amount >= tier['min_spend']:
            points = int(base_points * tier['multiplier'])
            return points, tier['name']

    return int(base_points), 'Bronze'


customers = [
    {'id': '001', 'name': 'John Smith', 'purchases': 750.50},
    {'id': '002', 'name': 'Emma Wilson', 'purchases': 1250.75},
    {'id': '003', 'name': 'Michael Brown', 'purchases': 350.25},
    {'id': '004', 'name': 'Sarah Davis', 'purchases': 950.00},
    {'id': '005', 'name': 'James Johnson', 'purchases': 125.50}
]


print("Customer Loyalty Points Report")
print("=" * 75)
print(f"{'ID':<6} {'Name':<20} {'Purchases':<12} {'Tier':<10} {'Points Earned':<15}
print("-" * 75)

total_points = 0
for customer in customers:
    points, tier = calculate_loyalty_points(customer['purchases'])
    total_points += points

    print(f"{customer['id']:<6} {customer['name']:<20} "
          f"${customer['purchases']:>9,.2f} {tier:<10} {points:>12,}")


print("\nLoyalty Program Summary")
```

```
print("-" * 75)
print(f"Total Customers: {len(customers)}")
print(f"Total Points Awarded: {total_points:,}")
print(f"Average Points per Customer: {total_points // len(customers):,}")

print("\nLoyalty Tier System")
print("-" * 75)
print("Platinum Tier: Spend $1,000+ and earn 3x points")
print("Gold Tier: Spend $500+ and earn 2x points")
print("Silver Tier: Spend $100+ and earn 1.5x points")
print("Bronze Tier: All other purchases earn 1x points")
```

```
Customer Loyalty Points Report
============================================================================
ID     Name                 Purchases    Tier       Points Earned
----------------------------------------------------------------------------
001    John Smith           $   750.50 Gold              1,501
002    Emma Wilson          $ 1,250.75 Platinum          3,752
003    Michael Brown        $   350.25 Silver              525
004    Sarah Davis          $   950.00 Gold              1,900
005    James Johnson        $   125.50 Silver              188

Loyalty Program Summary
----------------------------------------------------------------------------
Total Customers: 5
Total Points Awarded: 7,866
Average Points per Customer: 1,573

Loyalty Tier System
----------------------------------------------------------------------------
Platinum Tier: Spend $1,000+ and earn 3x points
Gold Tier: Spend $500+ and earn 2x points
Silver Tier: Spend $100+ and earn 1.5x points
Bronze Tier: All other purchases earn 1x points
```

In [ ]: