**Task 1: Sales Data Summary**

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 0

# 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)

Category A Sales: 1500

Category B Sales: 1200

Total Units Sold: 2700

Sales Difference: 300

Sales Ratio (A/B): 1.25

**Task 2: Customer Age Data**

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)

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

**Task 3: Product List Management**

def analyze\_product\_prices(prices: list, new\_premium\_price: float, mid\_range\_buffer: float = 50.0):

"""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\_price + mid\_range\_buffer)]

# 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\_prices, new\_premium\_price)

# 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}")

Result:

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]

**Task 4: Inventory Lookup**

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)

Result:

Product Name: Ultra Slim Laptop

SKU: LAPTOP001

**Task 5: Stock Level Alert System**

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 threshold ({reorder\_threshold})")

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)

Result:

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)

**Task 6: Sales Report Formatting**

# Sales Data Formatter

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)

Result:

Using For Loop:

LAPTOP

SMARTPHONE

TABLET

SMARTWATCH

Using While Loop:

LAPTOP

SMARTPHONE

TABLET

SMARTWATCH

**Task 7: Area Calculation for Store Layout**

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)

Result:

Electronics Area: 600 square feet

Clothing Area: 1400 square feet

Groceries Area: 1500 square feet

**Task 8: Customer Feedback Analysis**

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}")

Result: Vowel Count: 11

Reversed Message: !stcudorp dna ecivres remotsuc taerG

**Task 9: Price Filtering Tool**

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"] >= min\_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)

Result:

Products eligible for discount (Min price: $400.0):

Laptop: $999

Phone: $499

Tablet: $700

**Task 10: Sales Log File Management**

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: $126\n")

# Read and print file contents

with open('sales\_log.txt', 'r') as file:

content = file.read()

print("File Contents:")

print(content)

manage\_sales\_log()

Result:

File Contents:

Daily Sales Summary - 2024-12-04

Total Sales: $15,750 | Units Sold: 125 | Average Transaction: $126

**Task 11: Daily Sales Average**

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}")

Result:

Daily sales for the past week: $[1200.5, 950.75, 1500.25, 1100.0, 1350.5, 800.25, 1425.75]

Average daily sales: $1189.71

Task 12: Customer Segmentation

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\_THRESHOLD else "High"

print(f"Customer {customer['customer\_id']}: ${spending:.2f} - {category} Spender")

Result:

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

Task 13: Discount Calculation

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\_price:<9.2f}")

except ValueError as e:

print(f"Error processing {name}: {e}")

print("-" \* 50)

Result:

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

Task 14: Customer Feedback Sentiment Analysis

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}")

Result:

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

Task 15: Employee Salary Increment Calculator

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 Salary':<12}")

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,} ${new\_salary:<11,}")

except ValueError as e:

print(f"Error processing {name}: {e}")

print("-" \* 65)

Result:

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

Task 16: Monthly Sales Report Generator

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\_sales)

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}")

Result:

Report has been generated successfully: monthly\_report.txt

Report Preview:

Monthly Sales Report - December 2024

==================================================

Summary Statistics:

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

Total Monthly Sales: $40,212.58

Average Daily Sales: $1,340.42

Best Performing Day: 2024-12-05 ($1,901.92)

Lowest Performing Day: 2024-12-15 ($652.41)

Daily Sales Breakdown:

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

2024-12-01: $1,449.42

2024-12-02: $1,433.81

2024-12-03: $1,287.72

2024-12-04: $841.98

2024-12-05: $1,901.92

2024-12-06: $961.39

2024-12-07: $1,786.83

2024-12-08: $1,850.41

2024-12-09: $1,058.93

2024-12-10: $1,798.46

2024-12-11: $1,443.77

2024-12-12: $1,130.77

2024-12-13: $766.30

2024-12-14: $887.35

2024-12-15: $652.41

2024-12-16: $1,855.67

2024-12-17: $803.80

2024-12-18: $1,535.30

2024-12-19: $1,656.50

2024-12-20: $1,697.62

2024-12-21: $1,511.96

2024-12-22: $1,190.67

2024-12-23: $945.97

2024-12-24: $1,123.19

2024-12-25: $724.04

2024-12-26: $1,886.70

2024-12-27: $1,711.87

2024-12-28: $1,224.47

2024-12-29: $1,272.16

2024-12-30: $1,821.19

Task 17: Stock Replenishment Planning

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 Qty':<15}")

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}")

Result:

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

Task 18: Data Cleaning Utility

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}');")

Results:

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');

Task 19: Simple Sales Forecasting

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 forecast")

elif forecast < min\_sales:

print("- Forecasted sales below historical minimum - consider validating forecast")

Results:

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

Task 20: Customer Loyalty Points Calculator

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")

Results:

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