Name	**SHANTANU KAUSHIK [ABADS BATCH 16]**
Email	**shankshk@gmail.com**
Phone	**9887779477**
Project Overview:	To build an ETL (Extract, Transform, Load) pipeline to process and analyze sales data for a small e-commerce company. The company stores its data in separate CSV files and requires a system to integrate, clean, and analyze this data to generate valuable business insights. • Extract data from four CSV files: sales.csv, products.csv, customers.csv, and employees.csv. • Transform the data by cleaning, integrating, and enriching it. • Load the processed data into new CSV files for analysis and reporting. • Perform basic analysis on the transformed data to generate business insights.
Task 1	**Data Extraction:** • Read data from all four input CSV files. • Handle potential errors such as missing files or corrupt data.
Solution Task 1	[51]: ``` import csv from datetime import datetime # Helper function to load CSV files into lists of dictionaries def load_csv(filepath): with open(filepath, mode='r') as file: reader = csv.DictReader(file) return list(reader) # Load CSV files customers = load_csv(r'C:\Users\shantanu.kaushik\Data Analysis Course\Assignments\ELT\CSV\Customer.csv') employees = load_csv(r'C:\Users\shantanu.kaushik\Data Analysis Course\Assignments\ELT\CSV\employee.csv') products = load_csv(r'C:\Users\shantanu.kaushik\Data Analysis Course\Assignments\ELT\CSV\Product.csv') sales = load_csv(r'C:\Users\shantanu.kaushik\Data Analysis Course\Assignments\ELT\CSV\sales.csv') ```
Task 2	**Data Transformation:** • Clean the data by addressing missing values and correcting data types. • Integrate sales data with product, customer, and employee information. • Calculate derived fields such as total sale amount and profit margin. • Create a date dimension with additional time-based attributes (e.g., month, quarter, year).

Solution Task 2	

```python
# Clean data by addressing missing values and correcting data types
def clean_data(data, default_values):
    cleaned_data = []
    for row in data:
        cleaned_row = {}
        for key, value in row.items():
            # Handle missing values and assign default values where necessary
            if value == '' or value is None:
                cleaned_row[key] = default_values.get(key, None)
            else:
                # Convert data types based on expected type
                try:
                    # Assuming prices and quantities are numeric
                    if key.lower().endswith(('price', 'cost_price', 'quantity')):
                        cleaned_row[key] = float(value)
                    elif key.lower().endswith(('date', 'time')):
                        cleaned_row[key] = datetime.strptime(value, '%Y-%m-%d')  # Convert to date
                    else:
                        cleaned_row[key] = value
                except ValueError:
                    cleaned_row[key] = value
        cleaned_data.append(cleaned_row)
    return cleaned_data

# Example default values for missing data
default_values = {
    'price': 0,
    'quantity': 0,
    'cost_price': 0,
    'date': '1970-01-01'
}
```

```python
# Clean each dataset
cleaned_customers = clean_data(customers, default_values)
cleaned_employees = clean_data(employees, default_values)
cleaned_products = clean_data(products, default_values)
cleaned_sales = clean_data(sales, default_values)

# Integrate sales data with product, customer, and employee data
def integrate_data(sales, products, customers, employees):
    integrated_data = []

    for sale in sales:
        # Find related product, customer, and employee
        product = next((p for p in products if p['product_id'] == sale['product_id']), {})
        customer = next((c for c in customers if c['customer_id'] == sale['customer_id']), {})


        # Combine the data
        integrated_record = {**sale, **product, **customer}
        integrated_data.append(integrated_record)

    return integrated_data

# Integrated data set
integrated_sales = integrate_data(cleaned_sales, cleaned_products, cleaned_customers, cleaned_employees)

# Calculate derived fields: total sale amount and profit margin
def calculate_derived_fields(data):
    for row in data:
        row['total_sale_amount'] = float(row.get('price', 0)) * int(row.get('quantity', 0))
        row['profit_margin'] = row['total_sale_amount'] - float(row.get('cost', 0))
    return data

# Apply derived field calculations
integrated_sales = calculate_derived_fields(integrated_sales)
```

```python
# Create a date dimension with month, quarter, and year
def create_date_dimension(data):
    for row in data:
        sale_date = row.get('date')
        # Ensure the date is a datetime object, convert if necessary
        if isinstance(sale_date, str):
            try:
                sale_date = datetime.strptime(sale_date, '%Y-%m-%d')
                row['date'] = sale_date  # Update the row with the proper datetime object
            except ValueError:
                continue  # Skip rows with invalid date formats

        if isinstance(sale_date, datetime):
            row['year'] = sale_date.year
            row['month'] = sale_date.month
            row['quarter'] = (sale_date.month - 1) // 3 + 1
            row['day'] = sale_date.day
    return data


# Apply date dimension creation
integrated_sales = create_date_dimension(integrated_sales)

# Display the integrated and cleaned data with derived fields
for sale in integrated_sales[:5]:  # Display only the first 5 records for simplicity
    print(sale)
```

Task 3	**Data Loading:** • Generate at least two output CSV files: a. A comprehensive sales report combining information from all sources. b. A summary report with aggregated sales data.
Solution Task 3	``` # Write comprehensive sales report to CSV def write_comprehensive_report(data, filepath): if data: fieldnames = data[0].keys() with open(filepath, mode='w', newline='') as file: writer = csv.DictWriter(file, fieldnames=fieldnames) writer.writeheader() writer.writerows(data) # Save comprehensive report write_comprehensive_report(integrated_sales, r'C:\Users\shantanu.kaushik\Data Analysis Course\Assignments\ELT\CSV\integrated_sales.csv') ```

```python
# Generate summary report by aggregating sales data by product
def generate_summary_report(data):
    summary = {}

    for row in data:
        product_id = row.get('product_id')
        if product_id in summary:
            summary[product_id]['total_sales_amount'] += row['total_sale_amount']
            summary[product_id]['total_profit'] += row['profit_margin']
            summary[product_id]['total_quantity'] += int(row['quantity'])
        else:
            summary[product_id] = {
                'product_id': product_id,
                'product_name': row.get('product_name'),
                'total_sales_amount': row['total_sale_amount'],
                'total_profit': row['profit_margin'],
                'total_quantity': int(row['quantity']),
            }

    return list(summary.values())

# Generate summary report
summary_report = generate_summary_report(integrated_sales)

# Write summary report to CSV
def write_summary_report(data, filepath):
    if data:
        fieldnames = data[0].keys()
        with open(filepath, mode='w', newline='') as file:
            writer = csv.DictWriter(file, fieldnames=fieldnames)
            writer.writeheader()
            writer.writerows(data)

# Save summary report
write_summary_report(summary_report, r'C:\Users\shantanu.kaushik\Data Analysis Course\Assignments\ELT\CSV\summary_report.csv')
```

Task 4

Data Analysis:

- Calculate total sales and profit by product category.
- Identify top-selling products and key customers.
- Analyze sales trends over time (daily, monthly, quarterly).

Solution Task 4	```python ### Code to Calculate Total Sales and Profit by Product Category def sales_by_category(data): category_sales = {} for row in data: category = row.get('category') if category in category_sales: category_sales[category]['total_sales'] += row['total_sale_amount'] category_sales[category]['total_profit'] += row['profit_margin'] else: category_sales[category] = { 'category': category, 'total_sales': row['total_sale_amount'], 'total_profit': row['profit_margin'] } return list(category_sales.values()) # Calculate total sales and profit by product category category_sales_report = sales_by_category(integrated_sales) write_summary_report(category_sales_report, r'C:\Users\shantanu.kaushik\Data Analysis Course\Assignments\ELT\CSV\category_sales_report.csv') ###Code to Identify Top-Selling Products and Key Customers def top_selling_products(data, top_n=5): product_sales = {} for row in data: product_id = row.get('product_id') if product_id in product_sales: product_sales[product_id]['total_sales'] += row['total_sale_amount'] else: product_sales[product_id] = { 'product_id': product_id, 'product_name': row.get('product_name'), 'total_sales': row['total_sale_amount'], } sorted_products = sorted(product_sales.values(), key=lambda x: x['total_sales'], reverse=True) return sorted_products[:top_n] top_products = top_selling_products(integrated_sales) write_summary_report(top_products, r'C:\Users\shantanu.kaushik\Data Analysis Course\Assignments\ELT\CSV\top_products.csv') ```
Deliverables	**A Jupyter Notebook containing the complete ETL pipeline and analysis. - Attached** **Output CSV files with the processed and analyzed data. - Attached** **A brief report summarizing the insights derived from the data analysis. - Attached** **Share the repository link by including it in a text, Word, or PDF file format. –** **https://github.com/Shantanuneo/gradedETLproject.git**