LAB TEST-03

Name : P.Reshmitha Reddy

Rollno :2403A510A7

Batch :05

Course :AI Assisted Coding

Set-E1

Q1:

Scenario: In the Hospitality sector, a company faces a challenge related to code refactoring.

Task: Use AI-assisted tools to solve a problem involving code refactoring in this context.

Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

Prompt:# Use AI-assisted tools to solve a problem involving code refactoring in this context.

Code :

```python
"""
Refactored Hospitality Data Processor

- Cleans reservation and room-inventory CSVs
- Handles missing/invalid values
- Detects overbookings and anomalous revenue entries
- Aggregates occupancy and revenue by room type and date
- Small demo when run directly
"""
from __future__ import annotations
from dataclasses import dataclass
from typing import List, Dict, Optional, Tuple
import csv
import statistics
import pathlib
import random
from datetime import datetime, timedelta


@dataclass
class Reservation:
    reservation_id: str
    hotel_id: str
    guest_name: str
    room_type: str
    check_in: Optional[datetime]
    check_out: Optional[datetime]
    price_per_night: float
    nights: int
    total_price: float
    status: str  # e.g., booked, checked_out, cancelled


class HotelProcessor:
    """Processor for hotel reservations and room inventory."""

    def __init__(self, reservations: Optional[List[Reservation]] = None, capacity_by_type: Optional[Dict[str, int]] = None):
        self.reservations: List[Reservation] = reservations or []
        # capacity_by_type maps room_type -> number of rooms
        self.capacity_by_type: Dict[str, int] = capacity_by_type or {}

    @staticmethod
    def from_csv(path: str, capacity_by_type: Optional[Dict[str, int]] = None) -> "HotelProcessor":
        """Load CSV with headers: reservation_id,hotel_id,guest_name,room_type,check_in,check_out,price_per_night,status"""
        reservations: List[Reservation] = []
        with open(path, newline="", encoding="utf-8") as f:
            reader = csv.DictReader(f)
            for r in reader:
                try:
                    ci = HotelProcessor._parse_dt(r.get("check_in"))
                    co = HotelProcessor._parse_dt(r.get("check_out"))
                    price = float(r.get("price_per_night") or 0.0)
                    nights = HotelProcessor._compute_nights(ci, co)
                    total = float(r.get("total_price") or (price * nights))
                    reservations.append(
                        Reservation(
                            reservation_id=str(r.get("reservation_id", "")).strip(),
                            hotel_id=str(r.get("hotel_id", "")).strip(),
                            guest_name=str(r.get("guest_name", "")).strip(),
                            room_type=str(r.get("room_type", "standard")).strip(),
                            check_in=ci,
                            check_out=co,
                            price_per_night=price,
                            nights=nights,
                            total_price=total,
                            status=str(r.get("status", "booked")).strip(),
                        )
                    )
                except Exception:
                    continue
        return HotelProcessor(reservations, capacity_by_type)
```

```python
    def _parse_dt(val: Optional[str]) -> Optional[datetime]:
        if not val or str(val).strip() == "":
            return None
        for fmt in ("%Y-%m-%d", "%Y-%m-%d %H:%M:%S", "%d/%m/%Y"):
            try:
                return datetime.strptime(val, fmt)
            except Exception:
                continue
        try:
            # fallback to ISO parse
            return datetime.fromisoformat(val)
        except Exception:
            return None

    @staticmethod
    def _compute_nights(ci: Optional[datetime], co: Optional[datetime]) -> int:
        if not ci or not co:
            return 0
        delta = (co.date() - ci.date()).days
        return max(0, delta)

    @staticmethod
    def sample_data(n: int = 40) -> "HotelProcessor":
        """Generate sample reservations and a small capacity map for demo/testing."""
        room_types = ["standard", "deluxe", "suite"]
        cap = {"standard": 10, "deluxe": 5, "suite": 2}
        recs: List[Reservation] = []
        base_date = datetime(2025, 6, 1)
        for i in range(n):
            rt = random.choice(room_types)
            start = base_date + timedelta(days=random.randint(0, 14))
            nights = random.randint(1, 5)
            end = start + timedelta(days=nights)
            price = {"standard": 80.0, "deluxe": 150.0, "suite": 300.0}[rt] * random.uniform(0.8, 1.2)
            # inject occasional missing price or bad dates
            if random.random() < 0.05:
```

```
                price = 0.0
            if random.random() < 0.03:
                end = start  # zero nights (invalid)
            total = round(price * nights, 2)
            recs.append(
                Reservation(
                    reservation_id=f"R{i+1}",
                    hotel_id="H1",
                    guest_name=f"Guest{i+1}",
                    room_type=rt,
                    check_in=start,
                    check_out=end,
                    price_per_night=round(price, 2),
                    nights=nights,
                    total_price=total,
                    status="booked",
                )
            )
        return HotelProcessor(recs, cap)

    def clean(self) -> None:
        """Clean reservations in-place:
        - remove records with missing or invalid dates
        - fill missing prices with median per room type
        - compute nights/total_price where missing
        - remove exact duplicates (reservation_id)
        """
        # drop reservations without valid check_in/check_out or zero nights
        valid = [r for r in self.reservations if r.check_in and r.check_out and r.nights > 0]
        self.reservations = valid

        # median price per room type (ignore zero or missing)
        prices: Dict[str, List[float]] = {}
        for r in self.reservations:
```

```
            if r.price_per_night and r.price_per_night > 0:
                prices.setdefault(r.room_type, []).append(r.price_per_night)
        medians = {rt: (statistics.median(vals) if vals else 0.0) for rt, vals in prices.items()}

        # fill missing prices, recompute totals, deduplicate by reservation_id
        seen_ids = set()
        cleaned: List[Reservation] = []
        for r in self.reservations:
            if r.reservation_id in seen_ids:
                continue
            seen_ids.add(r.reservation_id)
            if r.price_per_night <= 0:
                r.price_per_night = medians.get(r.room_type, 0.0)
            if r.nights == 0:
                r.nights = self._compute_nights(r.check_in, r.check_out)
            if r.total_price <= 0:
                r.total_price = round(r.price_per_night * r.nights, 2)
            cleaned.append(r)
        self.reservations = cleaned

    def occupancy_by_date_and_type(self) -> Dict[Tuple[datetime.date, str], int]:
        """Compute number of occupied rooms for each date and room_type."""
        occ: Dict[Tuple[datetime.date, str], int] = {}
        for r in self.reservations:
            for d in range(r.nights):
                day = (r.check_in + timedelta(days=d)).date()
                k = (day, r.room_type)
                occ[k] = occ.get(k, 0) + 1
        return occ

    def detect_overbookings(self) -> List[Tuple[datetime.date, str, int, int]]:
        """Return list of (date, room_type, occupied, capacity) where occupied > capacity."""
        issues = []
        occ = self.occupancy_by_date_and_type()
```

```
        for (day, rt), occupied in occ.items():
            cap = self.capacity_by_type.get(rt, 0)
            if cap and occupied > cap:
                issues.append((day, rt, occupied, cap))
        return issues

    def detect_revenue_anomalies(self, z_threshold: float = 3.0) -> List[Reservation]:
        """Detect reservations where total_price per night is an outlier for the room type."""
        per_night = {}
        for r in self.reservations:
            if r.nights > 0:
                v = round(r.total_price / r.nights, 2)
                per_night.setdefault(r.room_type, []).append((r, v))
        anomalies: List[Reservation] = []
        for rt, items in per_night.items():
            vals = [v for (_r, v) in items]
            if len(vals) < 2:
                continue
            mean = statistics.mean(vals)
            stdev = statistics.pstdev(vals) or 0.0
            for r, v in items:
                z = 0.0 if stdev == 0 else abs((v - mean) / stdev)
                if z > z_threshold:
                    anomalies.append(r)
        return anomalies

    def aggregate_by_room_type(self) -> Dict[str, Dict[str, float]]:
        """Aggregate occupancy nights and revenue by room type."""
        agg: Dict[str, Dict[str, float]] = {}
        for r in self.reservations:
            a = agg.setdefault(r.room_type, {"nights": 0.0, "revenue": 0.0, "bookings": 0})
            a["nights"] += r.nights
            a["revenue"] += r.total_price
            a["bookings"] += 1
        for rt, v in agg.items():
```

```
            v["rev_per_night"] = v["revenue"] / v["nights"] if v["nights"] else 0.0
        return agg

    def summary(self) -> str:
        agg = self.aggregate_by_room_type()
        lines = [f"Reservations: {len(self.reservations)}"]
        for rt, v in agg.items():
            lines.append(
                f"{rt}: bookings={v['bookings']}, nights={v['nights']:.0f}, revenue={v['revenue']:.2f}, rev/night={v['rev_per_night']:.2f}"
            )
        over = self.detect_overbookings()
        lines.append(f"Overbookings: {len(over)}")
        return "\n".join(lines)

    def save_csv(self, path: str) -> None:
        p = pathlib.Path(path)
        p.parent.mkdir(parents=True, exist_ok=True)
        with open(p, "w", newline="", encoding="utf-8") as f:
            writer = csv.writer(f)
            writer.writerow(
                [
                    "reservation_id",
                    "hotel_id",
                    "guest_name",
                    "room_type",
                    "check_in",
                    "check_out",
                    "price_per_night",
                    "nights",
                    "total_price",
                    "status",
                ]
            )
            for r in self.reservations:
                writer.writerow(
```

```
                    [
                        r.reservation_id,
                        r.hotel_id,
                        r.guest_name,
                        r.room_type,
                        r.check_in.isoformat() if r.check_in else "",
                        r.check_out.isoformat() if r.check_out else "",
                        f"{r.price_per_night:.2f}",
                        r.nights,
                        f"{r.total_price:.2f}",
                        r.status,
                    ]
                )


def demo_run():
    proc = HotelProcessor.sample_data(60)
    print("=== Raw sample statistics ===")
    print(f"Raw reservations: {len(proc.reservations)}")
    anomalies_before = proc.detect_revenue_anomalies()
    print(f"Revenue anomalies before cleaning: {len(anomalies_before)}")

    proc.clean()
    print("\n=== After cleaning ===")
    print(proc.summary())

    anomalies_after = proc.detect_revenue_anomalies()
    print(f"\nRevenue anomalies after cleaning: {len(anomalies_after)}")
    overbookings = proc.detect_overbookings()
    if overbookings:
        print("\nExample overbooking:", overbookings[0])

    proc.save_csv("output/cleaned_hotel_reservations.csv")
    print("\nSaved cleaned CSV -> output/cleaned_hotel_reservations.csv")
```

```


if __name__ == "__main__":
    demo_run()
```

Output :

```
PS C:\Users\Reshm\Desktop\AIAC> & C:/Users/Reshm/anaconda3/python.exe c:/Users/Reshm/Desktop/AIAC/test-03.py
=== Raw sample statistics ===
Raw reservations: 60
Revenue anomalies before cleaning: 1

=== After cleaning ===
Reservations: 60
suite: bookings=20, nights=69, revenue=21250.85, rev/night=307.98
deluxe: bookings=22, nights=65, revenue=9682.62, rev/night=148.96
standard: bookings=18, nights=43, revenue=3434.83, rev/night=79.88
Overbookings: 16

Revenue anomalies after cleaning: 0

Example overbooking: (datetime.date(2025, 6, 11), 'suite', 7, 2)

Saved cleaned CSV -> output/cleaned_hotel_reservations.csv
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

- Purpose: refactors hotel reservation processing — parses CSV, cleans data, detects overbookings and revenue anomalies, aggregates occupancy/revenue, and saves cleaned CSV.
- Strengths: modular, typed, handles missing prices via median fill, computes nights/totals, provides demo data and CSV I/O.
- Output: textual summary, CSV "output/cleaned_hotel_reservations.csv", and lists of anomalies/overbookings from demo_run().

Q2:

Scenario: In the E-commerce sector, a company faces a challenge related to code refactoring.

Task: Use AI-assisted tools to solve a problem involving code

refactoring in this context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output

Prompt :# Use AI tools to refactor the given code to improve efficiency and readability. Provide the updated source code, explain how AI assisted the changes, and include sample output.

Code :

```python
"""
Refactored E-commerce Data Processor

- Cleans orders / product / customer CSVs
- Handles missing/invalid values
- Detects potential fraud/anomalies and inventory issues
- Aggregates sales by category and top customers
- Demo when run directly
"""
from __future__ import annotations
from dataclasses import dataclass
from typing import List, Dict, Optional, Tuple
import csv
import statistics
import pathlib
import random
from datetime import datetime, timedelta


@dataclass
class Order:
    order_id: str
    customer_id: str
    product_id: str
    category: str
    quantity: int
    price: float
    order_ts: Optional[datetime]
    status: str  # e.g., completed, refunded, cancelled
    total: float


class ECommerceProcessor:
    """Processor for e-commerce orders, products and customers."""

    def __init__(self, orders: Optional[List[Order]] = None, product_stock: Optional[Dict[str, int]] = None):
```

```python
        self.orders: List[Order] = orders or []
        self.product_stock: Dict[str, int] = product_stock or {}

    @staticmethod
    def _parse_dt(val: Optional[str]) -> Optional[datetime]:
        if not val:
            return None
        for fmt in ("%Y-%m-%d %H:%M:%S", "%Y-%m-%d", "%d/%m/%Y %H:%M:%S"):
            try:
                return datetime.strptime(val, fmt)
            except Exception:
                continue
        try:
            return datetime.fromisoformat(val)
        except Exception:
            return None

    @staticmethod
    def from_csv(path: str, product_stock: Optional[Dict[str, int]] = None) -> "ECommerceProcessor":
        """Load orders CSV with headers: order_id,customer_id,product_id,category,quantity,price,order_ts,status,total
        orders: List[Order] = []
        with open(path, newline="", encoding="utf-8") as f:
            reader = csv.DictReader(f)
            for r in reader:
                try:
                    ts = ECommerceProcessor._parse_dt(r.get("order_ts"))
                    qty = int(float(r.get("quantity") or 0))
                    price = float(r.get("price") or 0.0)
                    total = float(r.get("total") or (price * qty))
                    orders.append(
                        Order(
                            order_id=str(r.get("order_id", "")).strip(),
                            customer_id=str(r.get("customer_id", "")).strip(),
                            product_id=str(r.get("product_id", "")).strip(),
                            category=str(r.get("category", "uncategorized")).strip(),
```

```
                            quantity=qty,
                            price=price,
                            order_ts=ts,
                            status=str(r.get("status", "completed")).strip(),
                            total=total,
                        )
                    )
                except Exception:
                    continue
        return ECommerceProcessor(orders, product_stock or {})

    @staticmethod
    def sample_data(n: int = 80) -> "ECommerceProcessor":
        """Generate sample orders and product stock for demo/testing."""
        categories = ["electronics", "books", "apparel", "home"]
        product_stock = {f"P{i}": random.randint(0, 50) for i in range(1, 21)}
        orders = []
        now = datetime(2025, 7, 1, 10, 0, 0)
        for i in range(n):
            pid = random.choice(list(product_stock.keys()))
            cat = random.choice(categories)
            cid = f"C{random.randint(1,20)}"
            qty = random.randint(1, 5)
            price = round(random.uniform(5, 500), 2)
            ts = now - timedelta(minutes=random.randint(0, 60*24*30))
            status = "completed"
            # inject some refunds and suspicious behavior
            if random.random() < 0.08:
                status = "refunded"
            # inject extreme totals
            total = round(price * qty, 2)
            if random.random() < 0.03:
                total *= 10
            orders.append(Order(f"O{i+1}", cid, pid, cat, qty, price, ts, status, total))
        return ECommerceProcessor(orders, product_stock)
```

```
    def clean(self) -> None:
        """Clean orders in-place:
         - remove malformed orders (missing ids, non-positive qty)
         - fill missing totals/prices with sane defaults (median price per category)
         - update product_stock (deduct completed orders)
         - deduplicate by order_id
        """
        # drop malformed
        self.orders = [o for o in self.orders if o.order_id and o.customer_id and o.quantity > 0]

        # median price per category
        cat_prices: Dict[str, List[float]] = {}
        for o in self.orders:
            if o.price > 0:
                cat_prices.setdefault(o.category, []).append(o.price)
        medians = {c: (statistics.median(p) if p else 0.0) for c, p in cat_prices.items()}

        seen = set()
        cleaned = []
        for o in self.orders:
            if o.order_id in seen:
                continue
            seen.add(o.order_id)
            if o.price <= 0:
                o.price = medians.get(o.category, 0.0)
            if o.total <= 0:
                o.total = round(o.price * o.quantity, 2)
            cleaned.append(o)
        self.orders = cleaned

        # update stock for completed orders
        for o in self.orders:
            if o.status == "completed":
                self.product_stock[o.product_id] = self.product_stock.get(o.product_id, 0) - o.quantity
```

```
    def aggregate_sales_by_category(self) -> Dict[str, Dict[str, float]]:
        """Aggregate sales and quantities by category."""
        agg: Dict[str, Dict[str, float]] = {}
        for o in self.orders:
            a = agg.setdefault(o.category, {"revenue": 0.0, "quantity": 0, "orders": 0})
            a["revenue"] += o.total if o.status == "completed" else 0.0
            a["quantity"] += o.quantity if o.status == "completed" else 0
            a["orders"] += 1
        return agg

    def top_customers(self, top_n: int = 5) -> List[Tuple[str, float]]:
        """Return top customers by revenue."""
        rev: Dict[str, float] = {}
        for o in self.orders:
            if o.status == "completed":
                rev[o.customer_id] = rev.get(o.customer_id, 0.0) + o.total
        return sorted(rev.items(), key=lambda x: x[1], reverse=True)[:top_n]

    def detect_fraud(self, refund_rate_threshold: float = 0.3, large_order_multiplier: float = 5.0) -> Dict[str, List]:
        """
        Simple heuristics:
         - customers with refund rate above threshold
         - orders with total >> median total for category (multiplier)
         - many orders from same customer within short period (burst)
        """
        fraud = {"high_refund_customers": [], "large_orders": [], "bursty_customers": []}

        # refund rates per customer
        cust_counts: Dict[str, Dict[str, int]] = {}
        for o in self.orders:
            c = cust_counts.setdefault(o.customer_id, {"total": 0, "refunded": 0})
            c["total"] += 1
```

```
            if o.status == "refunded":
                c["refunded"] += 1
        for cid, counts in cust_counts.items():
            if counts["total"] > 0 and (counts["refunded"] / counts["total"]) >= refund_rate_threshold:
                fraud["high_refund_customers"].append(cid)

        # large orders compared to category median total
        cat_totals: Dict[str, List[float]] = {}
        for o in self.orders:
            cat_totals.setdefault(o.category, []).append(o.total)
        cat_medians = {c: (statistics.median(v) if v else 0.0) for c, v in cat_totals.items()}
        for o in self.orders:
            med = cat_medians.get(o.category, 0.0)
            if med > 0 and o.total > med * large_order_multiplier:
                fraud["large_orders"].append(o.order_id)

        # bursty customers: many orders within 1 hour
        orders_by_customer: Dict[str, List[datetime]] = {}
        for o in self.orders:
            if o.order_ts:
                orders_by_customer.setdefault(o.customer_id, []).append(o.order_ts)
        for cid, ts_list in orders_by_customer.items():
            ts_list.sort()
            # sliding window
            for i in range(len(ts_list)):
                j = i
                while j < len(ts_list) and (ts_list[j] - ts_list[i]).total_seconds() <= 3600:
                    j += 1
                if (j - i) >= 5:  # 5+ orders within 1 hour
                    fraud["bursty_customers"].append(cid)
                    break

        return fraud

    def detect_inventory_issues(self) -> List[Tuple[str, int]]:
```

```
        """Return products with negative stock."""
        return [(pid, stk) for pid, stk in self.product_stock.items() if stk < 0]

    def summary(self) -> str:
        agg = self.aggregate_sales_by_category()
        lines = [f"Orders: {len(self.orders)}"]
        for c, v in agg.items():
            lines.append(f"{c}: orders={v['orders']}, qty={v['quantity']}, revenue={v['revenue']:.2f}")
        inv_issues = self.detect_inventory_issues()
        lines.append(f"Inventory issues: {len(inv_issues)}")
        return "\n".join(lines)

    def save_csv(self, path: str) -> None:
        p = pathlib.Path(path)
        p.parent.mkdir(parents=True, exist_ok=True)
        with open(p, "w", newline="", encoding="utf-8") as f:
            writer = csv.writer(f)
            writer.writerow(["order_id", "customer_id", "product_id", "category", "quantity", "price", "order_ts", "status", "total"])
            for o in self.orders:
                writer.writerow([o.order_id, o.customer_id, o.product_id, o.category, o.quantity, f"{o.price:.2f}", o.order_ts.isoformat()


def demo_run():
    proc = ECommerceProcessor.sample_data(120)
    print("=== Raw sample statistics ===")
    print(f"Raw orders: {len(proc.orders)}")

    proc.clean()
    print("\n=== After cleaning ===")
    print(proc.summary())

    fraud = proc.detect_fraud()
    print(f"\nFraud findings: { {k: len(v) for k, v in fraud.items()} }")
    inv_issues = proc.detect_inventory_issues()
    if inv_issues:
```

```
245         print("\nExample inventory issue:", inv_issues[0])
246
247     proc.save_csv("output/cleaned_ecommerce_orders.csv")
248     print("\nSaved cleaned CSV -> output/cleaned_ecommerce_orders.csv")
249
250
251 if __name__ == "__main__":
252     demo_run()
```

Output :

```
PS C:\Users\Reshm\Desktop\AIAC> & C:/Users/Reshm/anaconda3/python.exe c:/Users/Reshm/Desktop/AIAC/test3q2.py
=== Raw sample statistics ===
Raw orders: 120

=== After cleaning ===
Orders: 120
books: orders=36, qty=118, revenue=37040.58
electronics: orders=29, qty=76, revenue=23589.91
home: orders=24, qty=68, revenue=17166.37
apparel: orders=31, qty=89, revenue=25032.62
Inventory issues: 8

Fraud findings: {'high_refund_customers': 1, 'large_orders': 3, 'bursty_customers': 0}

Example inventory issue: ('P5', -13)

Saved cleaned CSV -> output/cleaned_ecommerce_orders.csv
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

- Purpose: processes e-commerce orders — loads/creates orders, cleans data, updates product stock, aggregates sales, detects fraud and inventory issues, and can save cleaned CSV.
- Structure: Order dataclass + ECommerceProcessor with methods for parsing, sample data, clean, aggregate, top_customers, detect_fraud, detect_inventory_issues, summary, save_csv, and demo_run.
- Strengths: modular design, type hints, robust parsing with fallbacks, median-fill for missing prices/totals,

simple fraud heuristics (refund rate, large orders, bursts), and demo/sample-data support.

- Output: textual summary, fraud/inventory lists, and "output/cleaned_ecommerce_orders.csv" from demo_run().
- Caveats: heuristics are simple and may produce false positives; timestamp parsing and random sample data may not reflect production; performance may need improvements for very large datasets.