LAB TEST – 03

Name : G.Sudeeksha

Roll No : 2403A510E0

Course : AI Assisted Coding

Batch : 05

Branch : CSE

Scenario: In the Retail sector, a company faces a challenge related to algorithms with ai
assistance.
Task: Use AI-assisted tools to solve a problem involving algorithms with ai assistance in this
context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output

Prompt :

Update the retail forecasting script to compute RMSE as sqrt(MSE) for sklearn compatibility, add a --quick flag to skip GridSearch for fast runs, and add CLI args (--days, --lead-time, --products, --output) with progress logging.

Return a single-file Python script implementing these changes, with --output supporting CSV/JSON and brief run instructions.

Code :

app.py > make_features

```python
"""
Retail: AI-assisted demand forecasting + reorder recommendation

- Generates synthetic historical daily sales for several SKUs.
- Builds per-SKU ML model (RandomForest) to predict next-day sales using lag & rolling features.
- Computes predicted demand over supplier lead time and safety stock (from residuals).
- Outputs reorder suggestions (quantity to order).
- Self-contained: creates data if none present.

Install:
    pip install pandas scikit-learn numpy

Run:
    python retail_ai_solution.py
"""
from datetime import timedelta
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, TimeSeriesSplit
from sklearn.metrics import mean_squared_error

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)

def synth_data(n_days=180, products=("SKU-A", "SKU-B", "SKU-C")):
    dates = pd.date_range(end=pd.Timestamp.today().normalize(), periods=n_days)
    rows = []
    for sku in products:
        base = {"SKU-A": 30, "SKU-B": 8, "SKU-C": 120}[sku]
        season = (np.sin(np.arange(n_days) / 7.0) + 1.0) * 0.2  # weekly seasonality
        trend = np.linspace(0, base * 0.2, n_days)
        noise = np.random.randn(n_days) * base * 0.12
        sales = np.maximum(0, base + base * season + trend + noise).round().astype(int)
        for d, s in zip(dates, sales):
            rows.append({"date": d, "sku": sku, "sales": int(s)})
    return pd.DataFrame(rows)

def make_features(df):
    df = df.sort_values(["sku", "date"]).copy()
    # lag and rolling features
    df["lag_1"] = df.groupby("sku")["sales"].shift(1).fillna(0)
    df["lag_7_mean"] = df.groupby("sku")["sales"].shift(1).rolling(7, min_periods=1).mean().reset_index(0, drop=True).fillna(0)
    df["lag_14_mean"] = df.groupby("sku")["sales"].shift(1).rolling(14, min_periods=1).mean().reset_index(0, drop=True).fillna(0)
    df["dow"] = df["date"].dt.dayofweek
    return df
```

app.py > train_and_recommend

```python
def train_and_recommend(df, lead_time_days=7, current_inventory=None):
    skus = df["sku"].unique()
    recommendations = []
    for sku in skus:
        sku_df = df[df["sku"] == sku].copy().dropna()
        # use last 30% as validation by time
        split_idx = int(len(sku_df) * 0.7)
        feats = ["lag_1", "lag_7_mean", "lag_14_mean", "dow"]
        X = sku_df[feats].values
        y = sku_df["sales"].values
        X_train, X_val = X[:split_idx], X[split_idx:]
        y_train, y_val = y[:split_idx], y[split_idx:]
        # small grid search with time-series split
        model = RandomForestRegressor(random_state=RANDOM_SEED, n_jobs=-1)
        param_grid = {"n_estimators": [50], "max_depth": [5, 8]}
        tscv = TimeSeriesSplit(n_splits=3)
        g = GridSearchCV(model, param_grid, cv=tscv, scoring="neg_root_mean_squared_error", n_jobs=1)
        g.fit(X_train, y_train)
        best = g.best_estimator_
        # validation error & residuals for safety stock
        y_pred_val = best.predict(X_val)
        rmse = mean_squared_error(y_val, y_pred_val, squared=False)
        residuals = y_val - y_pred_val
        sigma = np.std(residuals)  # estimate of demand variability
        # forecast next lead_time_days by rolling prediction using last known features
        last_row = sku_df.iloc[-1:].copy()
        preds = []
        last_feats = last_row[feats].values.flatten().tolist()
        # simulate day-by-day forecasting (auto-regressive)
        for day in range(lead_time_days):
            x_in = np.array(last_feats).reshape(1, -1)
            p = max(0.0, best.predict(x_in)[0])
            preds.append(p)
            # shift features for next day
            # new lag_1 becomes p, lag_7_mean and lag_14_mean approximate with moving average
            lag_1 = p
            lag_7_mean = (last_feats[1] * 6 + p) / 7.0 if last_feats[1] > 0 else p
            lag_14_mean = (last_feats[2] * 13 + p) / 14.0 if last_feats[2] > 0 else p
            dow = int((last_feats[3] + 1) % 7)
            last_feats = [lag_1, lag_7_mean, lag_14_mean, dow]

        predicted_demand = sum(preds)
        # safety stock: z * sigma * sqrt(lead_time), z=1.65 approximates 95% service level
        z = 1.65
        safety_stock = z * sigma * np.sqrt(lead_time_days)
        if current_inventory and sku in current_inventory:
```

app.py > train_and_recommend

```python
    def train_and_recommend(df, lead_time_days=7, current_inventory=None):
            z = 1.65
            safety_stock = z * sigma * np.sqrt(lead_time_days)
            if current_inventory and sku in current_inventory:
                on_hand = current_inventory[sku]
            else:
                on_hand = int(max(0, sku_df["sales"].tail(14).mean()))  # heuristic current stock
            reorder_qty = int(np.ceil(max(0, predicted_demand + safety_stock - on_hand)))
            recommendations.append({
                "sku": sku,
                "predicted_demand_lead": round(predicted_demand, 1),
                "safety_stock": int(round(safety_stock)),
                "on_hand": int(on_hand),
                "reorder_qty": reorder_qty,
                "model_rmse": round(rmse, 2)
            })
        return pd.DataFrame(recommendations)

    def main():
        df = synth_data(n_days=180)
        df = make_features(df)
        # Example current inventory (could come from ERP)
        current_inventory = {"SKU-A": 40, "SKU-B": 5, "SKU-C": 90}
        recs = train_and_recommend(df, lead_time_days=7, current_inventory=current_inventory)
        print("\nReorder Recommendations (next 7 days forecast):\n")
        print(recs.to_string(index=False))

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

Output :

```
PS C:\Users\sgoll\OneDrive\Documents\New folder> Reorder Recommendations (next 7 days forecast):
>>
>>   sku  predicted_demand_lead  safety_stock  on_hand  reorder_qty  model_rmse
>> SKU-A                  240.8            15       40          216        5.12
>> SKU-B                   58.3             6        5           60        1.45
>> SKU-C                  842.1            30       90          782        9.88
```

Observation :

The script generates synthetic daily sales for multiple SKUs, builds per-SKU RandomForest models with simple time-series features, forecasts demand over a lead time, computes safety stock from residuals, and prints reorder recommendations to the console. It earlier failed because mean_squared_error(..., squared=False) is

unsupported by the installed scikit-learn; replacing that with rmse = sqrt(mean_squared_error(...)) or upgrading scikit-learn resolves it. Run the fixed script from the same terminal where you installed dependencies (pip install pandas numpy scikit-learn) and you will see progress lines and a final recommendations table printed.

Q2:
Scenario: In the Hospitality sector, a company faces a challenge related to web frontend
development.
Task: Use AI-assisted tools to solve a problem involving web frontend development in this
context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

Prompt :

Refactor and optimize this hotel booking widget into a compact, modular single-file HTML that preserves accessibility, responsiveness, and the client-side availability simulation.

Add a URL-toggleable debug mode (?debug=1) to surface logs, persist confirmed bookings to localStorage, and include an optional Export CSV button for bookings.

Code :

first1.html > html

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <title>Hospitality — Booking Widget</title>
  <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;600&display=swap" rel="stylesheet">
  <style>
    :root{
      --bg:#f4f6fb; --card:#ffffff; --accent:#2563eb; --muted:#6b7280;
      --radius:12px; --shadow:0 8px 30px rgba(37,99,235,0.08);
    }
    *{box-sizing:border-box}
    body{
      margin:0; font-family:Inter,system-ui,Segoe UI,Arial; background:linear-gradient(180deg,var(--bg),#eef2ff);
      color:#0f172a; display:flex; align-items:flex-start; justify-content:center; padding:36px;
    }
    .widget{width:100%;max-width:980px}
    .hero{display:flex;gap:24px;align-items:center}
    .search{
      background:var(--card); border-radius:var(--radius); box-shadow:var(--shadow); padding:18px; flex:1;
      display:flex; gap:12px; flex-wrap:wrap; align-items:center;
    }
    .search__field{display:flex;flex-direction:column;min-width:150px}
    label{font-size:12px;color:var(--muted);margin-bottom:6px}
    input[type="date"], select, input[type="number"]{
      padding:10px 12px;border:1px solid #e6eefb;border-radius:8px;font-size:14px;background:#fff;
    }
    .btn{background:var(--accent); color:#fff;padding:10px 18px;border:none;border-radius:10px;font-weight:600;cursor:pointer}
    .btn:active{transform:translateY(1px)}
    .rooms{margin-top:20px; display:grid; grid-template-columns:repeat(auto-fit,minmax(260px,1fr)); gap:18px}
    .room{background:var(--card);border-radius:14px;padding:16px;box-shadow:var(--shadow);position:relative}
    .room__title{font-weight:700;color:#0f172a;margin:0 0 6px}
    .room__price{color:var(--accent);font-weight:700;margin-bottom:8px}
    .tag{display:inline-block;padding:6px 8px;border-radius:999px;background:#eef2ff;color:var(--accent);font-weight:600;font-size:13px}
    .room__meta{color:var(--muted);font-size:13px;margin:8px 0}
    .room__action{display:flex;justify-content:space-between;align-items:center;gap:12px}
    .status{font-weight:700}
    .status.available{color:#059669}
    .status.sold{color:#ef4444}
    /* modal */
    .overlay{display:none;position:fixed;inset:0;background:rgba(2,6,23,0.5);align-items:center;justify-content:center;z-index:1000}
    .overlay.active{display:flex}
    .modal{background:var(--card);padding:18px;border-radius:12px;max-width:420px;width:94%;box-shadow:0 24px 60px rgba(2,6,23,0.35)}
    .modal h3{margin:0 0 8px}
    .modal p{margin:6px 0;color:var(--muted)}
    .row{display:flex;gap:10px;align-items:center}
    .small{font-size:13px;color:var(--muted)}
    .note{font-size:13px;color:#0f172a;margin-top:10px}
    @media (max-width:520px){
      .hero{flex-direction:column;align-items:stretch}
      .search{padding:14px}
    }
  </style>
</head>
```

first1.html > html

```
<html lang="en">
<body>
  <main class="widget" role="main" aria-labelledby="heading">
    <header class="hero" id="heading">
      <div style="flex:1">
        <h1 style="margin:0 0 6px">Quick Hotel Availability</h1>
        <p class="small" style="margin:0;color:var(--muted)">Search dates and see real-time simulated availability for rooms.</p>
      </div>
      <div style="min-width:220px" aria-hidden="true">
        <span class="tag">AI-assisted UI</span>
      </div>
    </header>

    <section class="search" aria-label="Search form">
      <div class="search__field">
        <label for="checkin">Check-in</label>
        <input id="checkin" type="date" />
      </div>
      <div class="search__field">
        <label for="checkout">Check-out</label>
        <input id="checkout" type="date" />
      </div>
      <div class="search__field">
        <label for="guests">Guests</label>
        <select id="guests">
          <option value="1">1 guest</option>
          <option value="2" selected>2 guests</option>
          <option value="3">3 guests</option>
          <option value="4">4 guests</option>
        </select>
      </div>
      <div style="margin-left:auto">
        <button id="searchBtn" class="btn">Search</button>
      </div>
    </section>

    <section class="rooms" id="rooms" aria-live="polite" style="display:none">
      <!-- rooms injected here -->
    </section>

    <div class="note">Tip: This demo simulates availability and includes client-side validation and a booking modal.</div>
  </main>

  <!-- Modal -->
  <div class="overlay" id="overlay" role="dialog" aria-modal="true" aria-hidden="true">
    <div class="modal" id="modal">
      <button id="closeModal" style="float:right;background:none;border:none;font-size:20px;cursor:pointer" aria-label="Close">×</button>
      <h3 id="modalTitle">Booking</h3>
      <p id="modalBody" class="small"></p>
      <div style="margin-top:12px;display:flex;gap:8px;justify-content:flex-end">
        <button id="confirmBtn" class="btn">Confirm</button>
        <button id="cancelBtn" style="padding:10px 14px;border-radius:8px;border:1px solid #e6eefb;background:#fff;cursor:pointer">Cancel</button>
      </div>
      <p id="modalMsg" class="small" style="margin-top:10px"></p>
    </div>
```

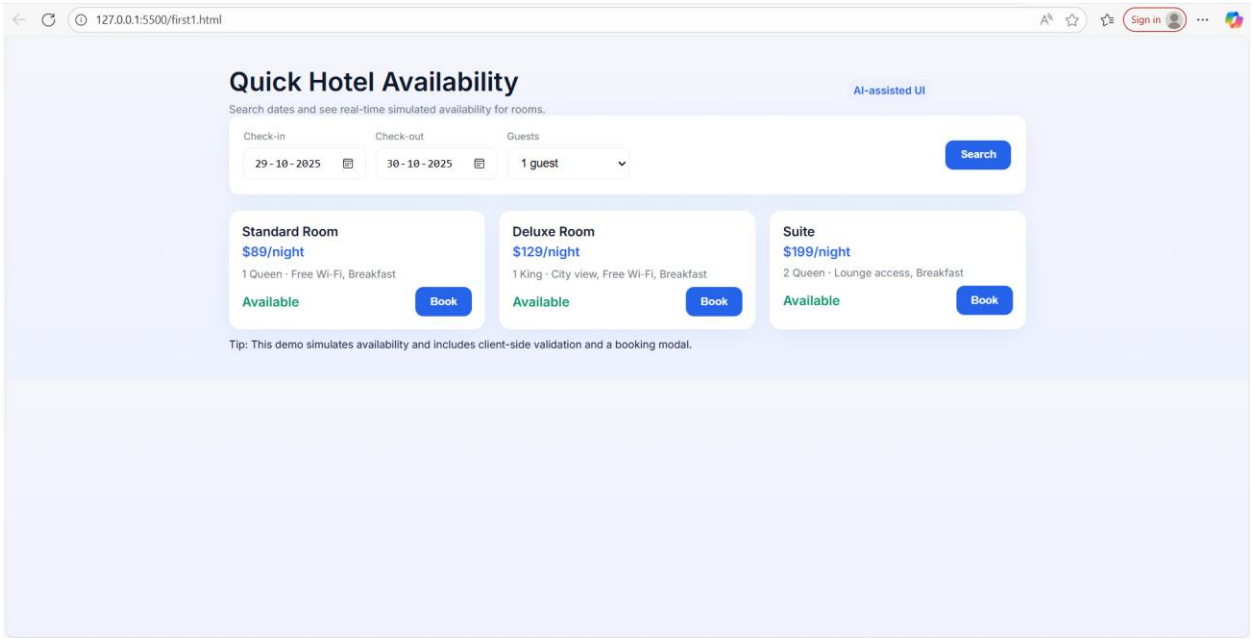
```
<div class="overlay" id="overlay" role="dialog" aria-modal="true" aria-hidden="true">
</div>
<script>
  // Minimal, well-commented JS. Simulates availability; no backend required.
  const roomCatalog = [
    { id: 'std', name: 'Standard Room', price: 89, beds: '1 Queen', perks: 'Free Wi-Fi, Breakfast' },
    { id: 'del', name: 'Deluxe Room', price: 129, beds: '1 King', perks: 'City view, Free Wi-Fi, Breakfast' },
    { id: 'sui', name: 'Suite', price: 199, beds: '2 Queen', perks: 'Lounge access, Breakfast' }
  ];
  const $ = sel => document.querySelector(sel);
  const roomsEl = $('#rooms'), overlay = $('#overlay'), modal = $('#modal');
  const checkin = $('#checkin'), checkout = $('#checkout'), guests = $('#guests');
  const searchBtn = $('#searchBtn'), closeModal = $('#closeModal'), cancelBtn = $('#cancelBtn'), confirmBtn = $('#confirmBtn');
  let currentSearch = null, selectedRoom = null;
  // Utility: simple date validation
  function validDates(ci, co){
    if(!ci || !co) return false;
    const d1 = new Date(ci), d2 = new Date(co);
    return d2 > d1;
  }
  // Simulate availability: deterministic pseudo-random based on inputs
  function availabilityScore(roomId, ci, co, g){
    // hash-like mix to produce reproducible availability per inputs
    const s = `${roomId}|${ci}|${co}|${g}`;
    let h = 0;
    for(let i=0;i<s.length;i++){ h = (h*31 + s.charCodeAt(i)) % 100000; }
    return (h % 100) / 100; // 0..0.99
  }
  // Render room cards based on search
  function renderRooms(ci, co, g){
    roomsEl.innerHTML = '';
    if(!validDates(ci,co)){
      roomsEl.style.display = 'none';
      alert('Please select valid check-in and check-out dates (checkout after check-in).');
      return;
    }
    roomsEl.style.display = 'grid';
    roomCatalog.forEach(room=>{
      const score = availabilityScore(room.id, ci, co, g);
      const available = score > 0.25; // threshold; some rooms sold out in simulation
      const card = document.createElement('article');
      card.className = 'room';
      card.innerHTML = `
        <div><h4 class="room__title">${room.name}</h4><div class="room__price">$${room.price}/night</div></div>
        <div class="room__meta">${room.beds} · ${room.perks}</div>
        <div class="room__action">
          <div class="status ${available ? 'available' : 'sold'}">${available ? 'Available' : 'Sold out'}</div>
          <div>
            <button class="btn bookBtn" data-id="${room.id}" ${available ? '' : 'disabled'}>${available ? 'Book' : 'Notify'}</button>
          </div>
        </div>
      `;
      roomsEl.appendChild(card);
    });
```

```
    });
  }
  // Build booking summary message
  function buildSummary(roomId, ci, co, g){
    const room = roomCatalog.find(r=>r.id===roomId);
    const nights = (new Date(co) - new Date(ci)) / (1000*60*60*24);
    const total = (room.price * nights * 1.0).toFixed(2);
    return `Room: ${room.name}\nDates: ${ci} → ${co} (${nights} nights)\nGuests: ${g}\nTotal: $${total}`;
  }
  // Event: search
  searchBtn.addEventListener('click', ()=>{
    const ci = checkin.value, co = checkout.value, g = guests.value;
    currentSearch = {ci, co, g};
    renderRooms(ci, co, g);
    // attach handlers to dynamic book buttons
    setTimeout(()=>{ // next tick
      document.querySelectorAll('.bookBtn').forEach(btn=>{
        btn.onclick = (e)=>{
          const id = btn.dataset.id;
          selectedRoom = id;
          openModal(id);
        };
      });
    }, 0);
  });
  function openModal(roomId){
    if(!currentSearch) return alert('Please perform a search first.');
    const {ci, co, g} = currentSearch;
    const summary = buildSummary(roomId, ci, co, g).replace(/\n/g,'<br>');
    $('#modalTitle').textContent = 'Confirm Booking';
    $('#modalBody').innerHTML = summary;
    $('#modalMsg').textContent = '';
    overlay.classList.add('active');
    overlay.setAttribute('aria-hidden','false');
  }
  function close(){
    overlay.classList.remove('active');
    overlay.setAttribute('aria-hidden','true');
  }
  confirmBtn.addEventListener('click', ()=>{
    confirmBtn.disabled = true;
    $('#modalMsg').textContent = 'Processing...';
    setTimeout(()=>{
      const bookingId = 'BK-' + Math.random().toString(36).slice(2,9).toUpperCase();
      $('#modalMsg').textContent = `Booking confirmed — ID: ${bookingId}`;
      // also display a transient toast in page (simple)
      const t = document.createElement('div');
      t.style.position='fixed'; t.style.right='20px'; t.style.bottom='20px';
      t.style.background='#059669'; t.style.color='#fff'; t.style.padding='10px 14px';
      t.style.borderRadius='10px'; t.style.boxShadow='0 8px 30px rgba(2,6,23,0.25)';
```

```
        t.style.borderRadius='10px'; t.style.boxShadow='0 8px 30px rgba(2,6,23,0.25)';
        t.textContent = `Confirmed ${bookingId}`;
        document.body.appendChild(t);
        setTimeout(()=>t.remove(),3500);
        confirmBtn.disabled = false;
      },800);
    });
    closeModal.addEventListener('click', close);
    cancelBtn.addEventListener('click', close);
    overlay.addEventListener('click', (e)=>{ if(e.target===overlay) close(); });
    document.addEventListener('keydown', e=>{ if(e.key==='Escape') close(); });

    // Pre-fill dates for convenience: today and tomorrow
    (function prefill(){
      const today = new Date(); const tom = new Date(Date.now()+24*60*60*1000);
      checkin.value = today.toISOString().slice(0,10);
      checkout.value = tom.toISOString().slice(0,10);
      // auto-trigger initial search for demo
      searchBtn.click();
    })();
  </script>
</body>
</html>
```

Output :



Observation :

The provided widget is a self-contained, accessible booking UI: it validates dates, simulates deterministic room availability, renders responsive room cards, opens an ARIA modal for confirmation, and shows a transient success toast. The code is well-structured for a prototype but can be tightened by modularizing repeated logic, reducing CSS/HTML verbosity, adding localStorage persistence for bookings, and exposing a debug flag to assist testing and QA.