```
* Recurring - apply monthly if missing for current month */
function applyRecurringIfDue() {
  const now = new Date();
  state.transactions.forEach(tx => {
    if (tx.recurring && tx.recurring.interval === 'monthly') {
      // if lastAppliedISO is not same month as now, create a copy for this month
      const last = tx.recurring.lastAppliedISO ? new Date(tx.recurring.lastAppliedISO) : null;
      const lastMonth = last ? (last.getFullYear() === now.getFullYear() && last.getMonth() === now.getMonth()) : false;
      // We should avoid duplicating today's recurring; only add if not applied this month
      if (!lastMonth) {
        const copy = { ...tx };
        delete copy.id; // new id
        copy.recurring = { ...tx.recurring, lastAppliedISO: nowISO() };
        // set date to start-of-month or today
        copy.dateISO = nowISO();
        addTransaction(copy, false); // don't save & render yet for each; will save after loop
        // update original recurring lastAppliedISO to now (to mark applied)
        tx.recurring.lastAppliedISO = nowISO();
      }
    }
  });
  saveState(); renderAll();
}

/* Aggregations */
function totalsForMonth(year, month) {
  // year, month are numbers (0-11)
  const byCategory = {};
  let income = 0, expense = 0;
  state.transactions.forEach(t => {
```

```
    const d = new Date(t.dateISO);
    if (d.getFullYear() === year && d.getMonth() === month) {
      if (t.type === 'income') income += Number(t.amount);
      else expense += Number(t.amount);
      const cat = (t.category || 'uncategorized').toLowerCase();
      byCategory[cat] = (byCategory[cat]||0) + Number(t.amount || 0);
    }
  });
  return { income, expense, byCategory };
}

function totalExpensesForCategoryThisMonth(category) {
  const now = new Date();
  const { byCategory } = totalsForMonth(now.getFullYear(), now.getMonth());
  return byCategory[category.toLowerCase()] || 0;
}

/* Alerts when nearing or exceeding budgets */
function checkBudgetAlert(category) {
  const cat = category.toLowerCase();
  const limit = state.budgets[cat];
  if (!limit) return;
  const spent = totalExpensesForCategoryThisMonth(cat);
  const ratio = spent / limit;
  if (ratio >= 1) {
    botSay(⚠ You have exceeded your budget for "${category}". Spent ${formatCurrency(spent)} / Limit ${formatCurrency(limit)}., true);
    try { navigator.vibrate && navigator.vibrate(200); } catch(e){}
  } else if (ratio >= 0.8) {
    botSay(⚠ You're nearing your budget for "${category}". Spent ${formatCurrency(spent)} / Limit ${formatCurrency(limit)}., true);
  } else {
```

```
    // no alert
  }
}

/* CSV Export */
function exportCSV() {
  if (!state.transactions.length) { botSay('No transactions to export.'); return; }
  const header = ['id','date','type','category','amount','note','recurring'];
  const rows = state.transactions.map(t => [
    t.id, t.dateISO, t.type, (t.category||''), t.amount, (t.note||''), JSON.stringify(t.recurring||'')
  ]);
  const csv = [header, ...rows].map(r => r.map(cell => {
    if (String(cell).includes(',') || String(cell).includes('"')) {
      return "${String(cell).replace(/"/g,'""')}";
    }
    return cell;
  }).join(',')).join('\n');
  const blob = new Blob([csv], { type:'text/csv' });
  const url = URL.createObjectURL(blob);
  const a = document.createElement('a'); a.href = url; a.download = 'transactions.csv'; a.click();
  URL.revokeObjectURL(url);
  botSay('✅ CSV exported.');
}
```