In [3]:
```
%pip install -U selenium
```

```
Defaulting to user installation because normal site-packages is not writeableNote: you may need to restart the kernel to use updated packages.

Requirement already satisfied: selenium in c:\users\jessi\appdata\roaming\python\python312\site-packages (4.39.0)
Requirement already satisfied: urllib3<3.0,>=2.5.0 in c:\users\jessi\appdata\roaming\python\python312\site-packages (from urllib3[socks]<3.0,>=2.5.0->selenium) (2.6.1)
Requirement already satisfied: trio<1.0,>=0.31.0 in c:\users\jessi\appdata\roaming\python\python312\site-packages (from selenium) (0.32.0)
Requirement already satisfied: trio-websocket<1.0,>=0.12.2 in c:\users\jessi\appdata\roaming\python\python312\site-packages (from selenium) (0.12.2)
Requirement already satisfied: certifi>=2025.10.5 in c:\users\jessi\appdata\roaming\python\python312\site-packages (from selenium) (2025.11.12)
Requirement already satisfied: typing_extensions<5.0,>=4.15.0 in c:\users\jessi\appdata\roaming\python\python312\site-packages (from selenium) (4.15.0)
Requirement already satisfied: websocket-client<2.0,>=1.8.0 in c:\programdata\anaconda3\lib\site-packages (from selenium) (1.8.0)
Requirement already satisfied: attrs>=23.2.0 in c:\users\jessi\appdata\roaming\python\python312\site-packages (from trio<1.0,>=0.31.0->selenium) (25.4.0)
Requirement already satisfied: sortedcontainers in c:\programdata\anaconda3\lib\site-packages (from trio<1.0,>=0.31.0->selenium) (2.4.0)
Requirement already satisfied: idna in c:\programdata\anaconda3\lib\site-packages (from trio<1.0,>=0.31.0->selenium) (3.7)
Requirement already satisfied: outcome in c:\users\jessi\appdata\roaming\python\python312\site-packages (from trio<1.0,>=0.31.0->selenium) (1.3.0.post0)
Requirement already satisfied: sniffio>=1.3.0 in c:\programdata\anaconda3\lib\site-packages (from trio<1.0,>=0.31.0->selenium) (1.3.0)
Requirement already satisfied: cffi>=1.14 in c:\programdata\anaconda3\lib\site-packages (from trio<1.0,>=0.31.0->selenium) (1.17.1)
Requirement already satisfied: wsproto>=0.14 in c:\users\jessi\appdata\roaming\python\python312\site-packages (from trio-websocket<1.0,>=0.12.2->selenium) (1.3.2)
Requirement already satisfied: pysocks!=1.5.7,<2.0,>=1.5.6 in c:\programdata\anaconda3\lib\site-packages (from urllib3[socks]<3.0,>=2.5.0->selenium) (1.7.1)
Requirement already satisfied: pycparser in c:\programdata\anaconda3\lib\site-packages (from cffi>=1.14->trio<1.0,>=0.31.0->selenium) (2.21)
Requirement already satisfied: h11<1,>=0.16.0 in c:\users\jessi\appdata\roaming\python\python312\site-packages (from wsproto>=0.14->trio-websocket<1.0,>=0.12.2->selenium) (0.16.0)
```

In [4]:
```
# ============================================================================
# IMPORTS AND SETUP
# ============================================================================

import time
import json
import re
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException, NoSuchElementException
from selenium.webdriver.common.action_chains import ActionChains
from urllib.parse import urljoin

ROOT_URL = "https://www.loblaws.ca/en"
MAX_WAIT = 60
TARGET_CATEGORIES = ["GROCERY", "HOME, BEAUTY & BABY"]


def get_driver(headless=False):
    """Create Chrome WebDriver."""
    opts = Options()
    if headless:
        opts.add_argument("--headless=new")
    opts.add_argument("--start-maximized")
    opts.add_argument("--disable-blink-features=AutomationControlled")
    return webdriver.Chrome(options=opts)


def dismiss_overlays(driver):
    """Close popups."""
    for selector in ['[data-testid="close-button"]', 'button[aria-label*="close" i]']:
        try:
            for el in driver.find_elements(By.CSS_SELECTOR, selector):
                if el.is_displayed():
                    el.click()
        except:
            pass
```

Extraction Functions

Extracting main categories from the menu bar

In [5]:
```
def extract_main_categories(driver, wait):
    """Extract main categories from navigation with aria-controls."""
    if driver.current_url != ROOT_URL:
        driver.get(ROOT_URL)
        time.sleep(2)
        dismiss_overlays(driver)

    # Wait for nav and then find L1_div using a direct path
    nav = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, 'nav[data-testid="iceberg-menu-bar"]')))

    # Try direct CSS path
    try:
        l1_div = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR,
            'nav[data-testid="iceberg-menu-bar"] > div > div[data-testid="iceberg-masthead-left"] > div[data-testid="iceberg-main-nav-l1"]')))
    except:
        # Fallback: navigate step by step
        first_div = nav.find_element(By.CSS_SELECTOR, 'div')
        masthead_left = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, 'div[data-testid="iceberg-masthead-left"]')))
        l1_div = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, 'div[data-testid="iceberg-main-nav-l1"]')))

    ul = l1_div.find_element(By.CSS_SELECTOR, 'ul')
    li_elements = ul.find_elements(By.CSS_SELECTOR, 'li')

    categories = []
    for li in li_elements:
        try:
            btn = li.find_element(By.CSS_SELECTOR, 'button[data-testid="iceberg-main-nav-l1-button"], a[data-testid="iceberg-main-nav-l1-button"]')
            span = btn.find_element(By.CSS_SELECTOR, 'span')
            text = span.text.strip()
            aria_controls = btn.get_attribute('aria-controls')

            if text:
                # Check if category already exists
                existing = next((c for c in categories if c['name'] == text), None)
                if not existing:
                    categories.append({"name": text, "aria-controls": aria_controls})
        except:
            continue

    return categories
```

Test the function

In [6]:
```
# Test extract_main_categories
driver = get_driver(headless=False)
wait = WebDriverWait(driver, MAX_WAIT)

try:
    print("Testing extract_main_categories()...")
    categories = extract_main_categories(driver, wait)

    print(f"\n✓ Extracted {len(categories)} categories:")
    for i, cat in enumerate(categories, 1):
        print(f"  {i}. {cat['name']} (aria-controls: {cat.get('aria-controls', 'N/A')})")

    print(f"\n✓ Test passed! Found {len(categories)} categories.")
except Exception as e:
    print(f"\n✗ Test failed: {e}")
    import traceback
    traceback.print_exc()
finally:
    driver.quit()
```

```
Testing extract_main_categories()...

✓ Extracted 7 categories:
  1. GROCERY (aria-controls: radix-:r14:)
  2. HOME, BEAUTY & BABY (aria-controls: radix-:r15:)
  3. NEW (aria-controls: radix-:r16:)
  4. FLYERS & DEALS (aria-controls: radix-:r17:)
  5. PHARMACY & SERVICES (aria-controls: radix-:r18:)
  6. MY SHOP (aria-controls: None)
  7. PC EXPRESS PASS (aria-controls: None)

✓ Test passed! Found 7 categories.
```

Extracting Sub Categories

In [7]:
```
def extract_subcategories(driver, wait, category_name, aria_controls=None):
    """Extract subcategories for a category using aria-controls."""
    if category_name.upper() not in [cat.upper() for cat in TARGET_CATEGORIES]:
        return []
```

```
    if aria_controls is None:
        return []

    if driver.current_url != ROOT_URL:
        driver.get(ROOT_URL)
        time.sleep(2)
        dismiss_overlays(driver)

    # Find the category button to hover over
    nav = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, 'nav[data-testid="iceberg-menu-bar"]')))
    try:
        l1_div = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR,
            'nav[data-testid="iceberg-menu-bar"] > div > div[data-testid="iceberg-masthead-left"] > div[data-testid="iceberg-main-nav-l1"]')))
    except:
        first_div = nav.find_element(By.CSS_SELECTOR, 'div')
        masthead_left = first_div.find_element(By.CSS_SELECTOR, 'div[data-testid="iceberg-masthead-left"]')
        l1_div = masthead_left.find_element(By.CSS_SELECTOR, 'div[data-testid="iceberg-main-nav-l1"]')

    # Find the category button by aria-controls (use XPath to handle colons in ID)
    cat_btn = l1_div.find_element(By.XPATH, f'.//button[@aria-controls="{aria_controls}"] | .//a[@aria-controls="{aria_controls}"]')

    # Try to click the chevron/arrow to open the dropdown
    try:
        chevron = cat_btn.find_element(By.CSS_SELECTOR, 'svg[data-testid="iceberg-main-nav-l1-button-chevron"]')
        driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", cat_btn)
        time.sleep(0.5)
        chevron.click()
    except:
        # Fallback: click the button itself or hover
        try:
            driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", cat_btn)
            time.sleep(0.5)
            cat_btn.click()
        except:
            # Last resort: hover
            ActionChains(driver).move_to_element(cat_btn).perform()

    time.sleep(1.5)

    # Find the popover content using the aria-controls id (use XPath to handle colons)
    popover_content = wait.until(EC.presence_of_element_located((By.XPATH, f'//*[@id="{aria_controls}"]')))

    # Navigate to the tablist inside the popover
    tabs_div = popover_content.find_element(By.CSS_SELECTOR, 'div[data-testid="iceberg-main-nav-l2-tabs"]')
    tablist = tabs_div.find_element(By.CSS_SELECTOR, 'div[role="tablist"][data-testid="iceberg-main-nav-l2-tabs-list"]')
    buttons = tablist.find_elements(By.CSS_SELECTOR, 'button[data-testid="iceberg-main-nav-l2-button"], a[data-testid="iceberg-main-nav-l2-button"]')

    subcategories = []
    for btn in buttons:
        try:
            # Find span inside a element
            a_elem = btn.find_element(By.CSS_SELECTOR, 'a')
            span = a_elem.find_element(By.CSS_SELECTOR, 'span')
            text = span.text.strip()

            if text and text not in [s['name'] for s in subcategories]:
                subcategories.append({"name": text})
        except:
            # Fallback: try direct span (for links that are already <a> elements)
            try:
                span = btn.find_element(By.CSS_SELECTOR, 'span')
                text = span.text.strip()

                if text and text not in [s['name'] for s in subcategories]:
                    subcategories.append({"name": text})
            except:
                continue

    return subcategories
```

Test the function

In [8]:

```
# Test extract_subcategories
driver = get_driver(headless=False)
wait = WebDriverWait(driver, MAX_WAIT)

try:
    print("Testing extract_subcategories()...")

    # First get categories with aria-controls
    categories = extract_main_categories(driver, wait)

    # Test with GROCERY category (one of the target categories)
    category_name = "GROCERY"
    print(f"Extracting subcategories for: {category_name}")

    # Find the category's aria-controls
    category_data = next((c for c in categories if category_name.upper() in c['name'].upper()), None)
    if not category_data or not category_data.get('aria-controls'):
        print(f"\n✗ Could not find {category_name} or aria-controls")
    else:
        aria_controls = category_data['aria-controls']
        print(f"Using aria-controls: {aria_controls}")

        subcategories = extract_subcategories(driver, wait, category_name, aria_controls)

        print(f"\n✓ Extracted {len(subcategories)} subcategories:")
        for i, subcat in enumerate(subcategories, 1):
            print(f"  {i}. {subcat['name']}")

        if len(subcategories) > 0:
            print(f"\n✓ Test passed! Found {len(subcategories)} subcategories for {category_name}.")
        else:
            print(f"\n⚠ Test completed but no subcategories found. This might be normal if the page structure is different.")

except Exception as e:
    print(f"\n✗ Test failed: {e}")
    import traceback
    traceback.print_exc()
finally:
    driver.quit()
```

```
Testing extract_subcategories()...
Extracting subcategories for: GROCERY
Using aria-controls: radix-:r13:

✓ Extracted 15 subcategories:
  1. Fruits & Vegetables
  2. Dairy & Eggs
  3. Meat
  4. Pantry
  5. International Foods
  6. Snacks, Chips & Candy
  7. Frozen Food
  8. Natural and Organic
  9. Bakery
  10. Prepared Meals
  11. Drinks
  12. Deli
  13. Fish & Seafood
  14. Beer & Wine
  15. Floral Shop

✓ Test passed! Found 15 subcategories for GROCERY.
```

Extracting subcategory2 from dropdown

In [9]:

```
def extract_subcategory2_from_dropdown(driver, wait, category_name, subcategory_name, category_aria_controls, popover_content=None):
    """
    Extract subcategory2 items from dropdown by clicking on subcategory button.

    Args:
        driver: Selenium WebDriver instance
        wait: WebDriverWait instance
        category_name: Name of the category
        subcategory_name: Name of the subcategory
        category_aria_controls: aria-controls value for the category
        popover_content: Optional pre-opened popover content element (for optimization)

    Returns:
        list: List of subcategory2 items with name and url
    """
    # Only navigate if not already on ROOT_URL or if popover_content not provided
    if popover_content is None:
        if driver.current_url != ROOT_URL:
            driver.get(ROOT_URL)
            time.sleep(1.5)  # Reduced from 2
```

```
            dismiss_overlays(driver)

        # Find the category button and open dropdown
        nav = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, 'nav[data-testid="iceberg-menu-bar"]')))
        try:
            l1_div = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR,
                'nav[data-testid="iceberg-menu-bar"] > div > div[data-testid="iceberg-masthead-left"] > div[data-testid="iceberg-main-nav-l1"]')))
        except:
            first_div = nav.find_element(By.CSS_SELECTOR, 'div')
            masthead_left = first_div.find_element(By.CSS_SELECTOR, 'div[data-testid="iceberg-masthead-left"]')
            l1_div = masthead_left.find_element(By.CSS_SELECTOR, 'div[data-testid="iceberg-main-nav-l1"]')

        # Find and click the category button
        cat_btn = l1_div.find_element(By.XPATH, f'.//button[@aria-controls="{category_aria_controls}"] | .//a[@aria-controls="{category_aria_controls}"]')

        try:
            chevron = cat_btn.find_element(By.CSS_SELECTOR, 'svg[data-testid="iceberg-main-nav-l1-button-chevron"]')
            driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", cat_btn)
            time.sleep(0.3)  # Reduced from 0.5
            chevron.click()
        except:
            try:
                driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", cat_btn)
                time.sleep(0.3)  # Reduced from 0.5
                cat_btn.click()
            except:
                ActionChains(driver).move_to_element(cat_btn).perform()

        time.sleep(1.2)  # Reduced from 1.5

        # Find the popover content
        popover_content = wait.until(EC.presence_of_element_located((By.XPATH, f'//*[@id="{category_aria_controls}"]')))
    else:
        # Popover already open, just need to find subcategory button
        pass

    # Find the subcategory button
    tabs_div = popover_content.find_element(By.CSS_SELECTOR, 'div[data-testid="iceberg-main-nav-l2-tabs"]')
    tablist = tabs_div.find_element(By.CSS_SELECTOR, 'div[role="tablist"][data-testid="iceberg-main-nav-l2-tabs-list"]')
    buttons = tablist.find_elements(By.CSS_SELECTOR, 'button[data-testid="iceberg-main-nav-l2-button"], a[data-testid="iceberg-main-nav-l2-button"]')

    subcategory_btn = None
    for btn in buttons:
        try:
            a_elem = btn.find_element(By.CSS_SELECTOR, 'a')
            span = a_elem.find_element(By.CSS_SELECTOR, 'span')
            if subcategory_name.upper() in span.text.upper():
                subcategory_btn = btn
                break
        except:
            try:
                span = btn.find_element(By.CSS_SELECTOR, 'span')
                if subcategory_name.upper() in span.text.upper():
                    subcategory_btn = btn
                    break
            except:
                continue

    if subcategory_btn is None:
        return []

    # Click the subcategory button to show its content
    driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", subcategory_btn)
    time.sleep(0.1)  # Reduced from 0.2
    subcategory_btn.click()
    time.sleep(0.3)  # Reduced from 0.5

    # Get the button's aria-controls to find the content panel
    btn_id = subcategory_btn.get_attribute('id')
    btn_aria_controls = subcategory_btn.get_attribute('aria-controls')

    if not btn_aria_controls and btn_id and '-trigger-' in btn_id:
        btn_aria_controls = btn_id.replace('-trigger-', '-content-')

    if not btn_aria_controls:
        return []

    # Find the content panel and extract subcategory2 items
    try:
        content_panel = popover_content.find_element(By.XPATH, f'.//*[@id="{btn_aria_controls}"]')
        content_list = content_panel.find_element(By.CSS_SELECTOR, 'ul[data-testid="iceberg-main-nav-l3-content-list"]')
        links = content_list.find_elements(By.CSS_SELECTOR, 'a[data-testid="iceberg-main-nav-l3-button"]')

        subcategory2_list = []
        for link in links:
            try:
                link_text = link.text.strip()
                link_href = link.get_attribute('href')
                # Skip "See ALL" links
                if link_text and link_href and not link_text.lower().startswith('see all'):
                    if not link_href.startswith('http'):
                        link_href = urljoin(ROOT_URL, link_href)
                    subcategory2_list.append({
                        "name": link_text,
                        "url": link_href
                    })
            except:
                continue

        return subcategory2_list
    except:
        return []
```

In [10]:

```
def extract_subcategory2_while_dropdown_open(driver, wait, category_name, subcategory_name, popover_content, all_subcategory_buttons):
    """
    Extract subcategory2 items for a specific subcategory while dropdown is already open.
    This is optimized for sequential processing within the same category.

    Args:
        driver: Selenium WebDriver instance
        wait: WebDriverWait instance
        category_name: Name of the category
        subcategory_name: Name of the subcategory
        popover_content: Already opened popover content element
        all_subcategory_buttons: List of all subcategory buttons (pre-fetched)

    Returns:
        list: List of subcategory2 items with name and url
    """
    subcategory2_list = []

    # Find the matching subcategory button
    subcategory_btn = None
    for btn in all_subcategory_buttons:
        try:
            a_elem = btn.find_element(By.CSS_SELECTOR, 'a')
            span = a_elem.find_element(By.CSS_SELECTOR, 'span')
            if subcategory_name.upper() in span.text.upper():
                subcategory_btn = btn
                break
        except:
            try:
                span = btn.find_element(By.CSS_SELECTOR, 'span')
                if subcategory_name.upper() in span.text.upper():
                    subcategory_btn = btn
                    break
            except:
                continue

    if subcategory_btn is None:
        return []

    # Click the subcategory button to show its content
    try:
        driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", subcategory_btn)
        time.sleep(0.1)
        subcategory_btn.click()
        time.sleep(0.3)

        # Get the button's aria-controls to find the content panel
        btn_id = subcategory_btn.get_attribute('id')
        btn_aria_controls = subcategory_btn.get_attribute('aria-controls')

        if not btn_aria_controls and btn_id and '-trigger-' in btn_id:
            btn_aria_controls = btn_id.replace('-trigger-', '-content-')

        if not btn_aria_controls:
            return []
```

```
        # Find the content panel and extract subcategory2 items
        try:
            content_panel = popover_content.find_element(By.XPATH, f'.//*[@id="{btn_aria_controls}"]')
            content_list = content_panel.find_element(By.CSS_SELECTOR, 'ul[data-testid="iceberg-main-nav-l3-content-list"]')
            links = content_list.find_elements(By.CSS_SELECTOR, 'a[data-testid="iceberg-main-nav-l3-button"]')

            for link in links:
                try:
                    link_text = link.text.strip()
                    link_href = link.get_attribute('href')
                    # Skip "See All" links
                    if link_text and link_href and not link_text.lower().startswith('see all'):
                        if not link_href.startswith('http'):
                            link_href = urljoin(ROOT_URL, link_href)
                        subcategory2_list.append({
                            "name": link_text,
                            "url": link_href
                        })
                except:
                    continue
        except:
            pass
    except:
        pass

    return subcategory2_list
```

In [11]:

```
def extract_all_subcategory2_for_category(driver, wait, category_name, subcategories, category_aria_controls):
    """
    Optimized: Extract all subcategory2 items for all subcategories in one dropdown session.
    This avoids repeatedly opening/closing the dropdown.

    Args:
        driver: Selenium WebDriver instance
        wait: WebDriverWait instance
        category_name: Name of the category
        subcategories: List of subcategory dicts with 'name' key
        category_aria_controls: aria-controls value for the category

    Returns:
        dict: Mapping of subcategory_name -> list of subcategory2 items
    """
    result = {}

    # Navigate to ROOT_URL and open dropdown once
    if driver.current_url != ROOT_URL:
        driver.get(ROOT_URL)
        time.sleep(1.5)
        dismiss_overlays(driver)

    # Find the category button and open dropdown
    nav = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, 'nav[data-testid="iceberg-menu-bar"]')))
    try:
        l1_div = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR,
            'nav[data-testid="iceberg-menu-bar"] > div > div[data-testid="iceberg-masthead-left"] > div[data-testid="iceberg-main-nav-l1"]')))
    except:
        first_div = nav.find_element(By.CSS_SELECTOR, 'div')
        masthead_left = first_div.find_element(By.CSS_SELECTOR, 'div[data-testid="iceberg-masthead-left"]')
        l1_div = masthead_left.find_element(By.CSS_SELECTOR, 'div[data-testid="iceberg-main-nav-l1"]')

    # Find and click the category button
    cat_btn = l1_div.find_element(By.XPATH, f'.//button[@aria-controls="{category_aria_controls}"] | .//a[@aria-controls="{category_aria_controls}"]')

    try:
        chevron = cat_btn.find_element(By.CSS_SELECTOR, 'svg[data-testid="iceberg-main-nav-l1-button-chevron"]')
        driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", cat_btn)
        time.sleep(0.3)
        chevron.click()
    except:
        try:
            driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", cat_btn)
            time.sleep(0.3)
            cat_btn.click()
        except:
            ActionChains(driver).move_to_element(cat_btn).perform()

    time.sleep(1.2)

    # Get the popover content once
    popover_content = wait.until(EC.presence_of_element_located((By.XPATH, f'//*[@id="{category_aria_controls}"]')))

    # Get all subcategory buttons
    tabs_div = popover_content.find_element(By.CSS_SELECTOR, 'div[data-testid="iceberg-main-nav-l2-tabs"]')
    tablist = tabs_div.find_element(By.CSS_SELECTOR, 'div[role="tablist"][data-testid="iceberg-main-nav-l2-tabs-list"]')
    all_subcategory_buttons = tablist.find_elements(By.CSS_SELECTOR, 'button[data-testid="iceberg-main-nav-l2-button"], a[data-testid="iceberg-main-nav-l2-button"]')

    # Process each subcategory
    for subcategory in subcategories:
        subcategory_name = subcategory['name']
        result[subcategory_name] = []

        # Find the matching subcategory button
        subcategory_btn = None
        for btn in all_subcategory_buttons:
            try:
                a_elem = btn.find_element(By.CSS_SELECTOR, 'a')
                span = a_elem.find_element(By.CSS_SELECTOR, 'span')
                if subcategory_name.upper() in span.text.upper():
                    subcategory_btn = btn
                    break
            except:
                try:
                    span = btn.find_element(By.CSS_SELECTOR, 'span')
                    if subcategory_name.upper() in span.text.upper():
                        subcategory_btn = btn
                        break
                except:
                    continue

        if subcategory_btn is None:
            continue

        # Click the subcategory button to show its content
        try:
            driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", subcategory_btn)
            time.sleep(0.1)
            subcategory_btn.click()
            time.sleep(0.3)

            # Get the button's aria-controls to find the content panel
            btn_id = subcategory_btn.get_attribute('id')
            btn_aria_controls = subcategory_btn.get_attribute('aria-controls')

            if not btn_aria_controls and btn_id and '-trigger-' in btn_id:
                btn_aria_controls = btn_id.replace('-trigger-', '-content-')

            if not btn_aria_controls:
                continue

            # Find the content panel and extract subcategory2 items
            try:
                content_panel = popover_content.find_element(By.XPATH, f'.//*[@id="{btn_aria_controls}"]')
                content_list = content_panel.find_element(By.CSS_SELECTOR, 'ul[data-testid="iceberg-main-nav-l3-content-list"]')
                links = content_list.find_elements(By.CSS_SELECTOR, 'a[data-testid="iceberg-main-nav-l3-button"]')

                for link in links:
                    try:
                        link_text = link.text.strip()
                        link_href = link.get_attribute('href')
                        # Skip "See All" links
                        if link_text and link_href and not link_text.lower().startswith('see all'):
                            if not link_href.startswith('http'):
                                link_href = urljoin(ROOT_URL, link_href)
                            result[subcategory_name].append({
                                "name": link_text,
                                "url": link_href
                            })
                    except:
                        continue
            except:
                pass
        except:
            continue

    return result
```

In [12]:

```
### Test extract subcategory2 from dropdown
```

In [12]:
```
# Test extract_subcategory2_from_dropdown
driver = get_driver(headless=False)
wait = WebDriverWait(driver, MAX_WAIT)

try:
    print("Testing extract_subcategory2_from_dropdown()...")

    # Get categories
    categories = extract_main_categories(driver, wait)
    category_name = "GROCERY"
    subcategory_name = "Dairy & Eggs"

    category_data = next((c for c in categories if category_name.upper() in c['name'].upper()), None)
    if not category_data or not category_data.get('aria-controls'):
        print(f"\nX Could not find {category_name} or aria-controls")
    else:
        aria_controls = category_data['aria-controls']
        print(f"Extracting subcategory2 for: {category_name} > {subcategory_name}")
        print(f"Using aria-controls: {aria_controls}")

        subcategory2_list = extract_subcategory2_from_dropdown(driver, wait, category_name, subcategory_name, aria_controls)

        print(f"\n✓ Extracted {len(subcategory2_list)} subcategory2 items:")
        for i, item in enumerate(subcategory2_list, 1):
            print(f"  {i}. {item['name']}")
            print(f"      url: {item['url']}")

        if len(subcategory2_list) > 0:
            print(f"\n✓ Test passed! Found {len(subcategory2_list)} subcategory2 items.")
        else:
            print(f"\n⚠ Test completed but no subcategory2 items found.")

except Exception as e:
    print(f"\nX Test failed: {e}")
    import traceback
    traceback.print_exc()
finally:
    driver.quit()
```

```
Testing extract_subcategory2_from_dropdown()...
Extracting subcategory2 for: GROCERY > Dairy & Eggs
Using aria-controls: radix-:r1e:

✓ Extracted 9 subcategory2 items:
  1. Milk & Cream
      url: https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?navid=flyout-L3-Milk-and-Cream
  2. Egg & Egg Substitutes
      url: https://www.loblaws.ca/en/food/dairy-eggs/egg-egg-substitutes/c/28222?navid=flyout-L3-Egg-and-Egg-Substitutes
  3. Butter & Spreads
      url: https://www.loblaws.ca/en/food/dairy-eggs/butter-spreads/c/28220?navid=flyout-L3-Butter-and-Spreads
  4. Cheese
      url: https://www.loblaws.ca/en/food/dairy-eggs/cheese/c/28225?navid=flyout-L3-Cheese
  5. Yogurt
      url: https://www.loblaws.ca/en/food/dairy-eggs/yogurt/c/28227?navid=flyout-L3-Yogurt
  6. Desserts & Doughs
      url: https://www.loblaws.ca/en/food/dairy-eggs/desserts-doughs/c/28221?navid=flyout-L3-Desserts-and-Doughs
  7. Sour Cream & Dips
      url: https://www.loblaws.ca/en/food/dairy-eggs/sour-cream-dips/c/28226?navid=flyout-L3-Sour-Cream-and-Dips
  8. Lactose Free
      url: https://www.loblaws.ca/en/food/dairy-eggs/lactose-free/c/28223?navid=flyout-L3-Lactose-Free
  9. Non-Dairy Milk Alternatives
      url: https://www.loblaws.ca/en/food/dairy-eggs/non-dairy-milk-alternatives/c/58904?navid=flyout-L3-Food-DairyandEggs-Non-DairyMilkAlternatives

✓ Test passed! Found 9 subcategory2 items.
```

Extract products

In [13]:
```
def extract_products_from_grid(driver, wait, see_all_true_url, debug=False):
    """Extract products from product grid."""
    if debug:
        print(f"\n[DEBUG] Navigating to: {see_all_true_url}")

    driver.get(see_all_true_url)
    time.sleep(2)  # Reduced from 3 - optimized for faster page loads
    dismiss_overlays(driver)

    # Wait for the listing page container to appear
    try:
        listing_container = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, '[data-testid="listing-page-container"]')))
        if debug:
            print(f"[DEBUG] ✓ Listing container found: [data-testid='listing-page-container']")
    except Exception as e:
        if debug:
            print(f"[DEBUG] X Listing container NOT found: [data-testid='listing-page-container'] - {e}")
        return []

    # Wait for the grid to appear within the listing container
    try:
        grid = listing_container.find_element(By.CSS_SELECTOR, '[data-testid="product-grid-component"]')
        if debug:
            print(f"[DEBUG] ✓ Grid found: [data-testid='product-grid-component']")
    except Exception as e:
        if debug:
            print(f"[DEBUG] X Grid NOT found: [data-testid='product-grid-component'] - {e}")
        return []

    # Wait for at least one product title to appear (indicates products are loaded)
    try:
        wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, 'h3[data-testid="product-title"]')))
        if debug:
            print(f"[DEBUG] ✓ At least one product title found")
    except Exception as e:
        if debug:
            print(f"[DEBUG] X No product titles found - {e}")

    # Scroll multiple times to load all products (lazy loading)
    last_height = driver.execute_script("return document.body.scrollHeight")
    scroll_attempts = 0
    max_scrolls = 5

    while scroll_attempts < max_scrolls:
        driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        time.sleep(1.5)  # Reduced from 2 - optimized for faster product loading
        new_height = driver.execute_script("return document.body.scrollHeight")
        if new_height == last_height:
            break
        last_height = new_height
        scroll_attempts += 1

    if debug:
        print(f"[DEBUG] Scrolled {scroll_attempts} times")

    # Scroll back to top to ensure all elements are accessible
    driver.execute_script("window.scrollTo(0, 0);")
    time.sleep(2)

    products = []
    seen_titles = set()  # To avoid duplicates across pages

    # Ensure re module is available
    try:
        import re
    except:
        pass

    # Check for pagination and find maximum page number
    max_page = 1
    try:
        # Scroll down to find pagination (it might be at the bottom)
        driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        time.sleep(1)

        pagination = listing_container.find_element(By.CSS_SELECTOR, '[data-testid="pagination"]')
        if debug:
            print(f"[DEBUG] ✓ Pagination container found")

        # Get all page links with aria-label starting with "Page"
        page_links = pagination.find_elements(By.CSS_SELECTOR, 'a[aria-label^="Page"]')
        if debug:
            print(f"[DEBUG] Found {len(page_links)} page links")

        # Extract all page numbers from href attributes
        page_numbers_found = []
        for link in page_links:
            try:
                href = link.get_attribute('href')
                aria_label = link.get_attribute('aria-label')
```

```
            if debug and len(page_numbers_found) < 5:  # Debug first few
                print(f"[DEBUG]   Link: aria-label='{aria_label}', href='{href}'")

            if href:
                # Extract page number from ?page=N or &page=N
                try:
                    match = re.search(r'[?&]page=(\d+)', href)
                    if match:
                        page_num = int(match.group(1))
                        if page_num not in page_numbers_found:
                            page_numbers_found.append(page_num)
                except Exception as e:
                    if debug:
                        print(f"[DEBUG]   Error extracting page number from '{href}': {e}")
        except Exception as e:
            if debug:
                print(f"[DEBUG]   Error processing link: {e}")

    if page_numbers_found:
        max_page = max(page_numbers_found)
        if debug:
            print(f"[DEBUG] ✓ Found pagination - Maximum page: {max_page} (from visible pages: {sorted(page_numbers_found)})")
    else:
        if debug:
            print(f"[DEBUG] ⚠ Found pagination but no page numbers extracted")
            # Try alternative: get all links in pagination
            all_links = pagination.find_elements(By.CSS_SELECTOR, 'a')
            if debug:
                print(f"[DEBUG]   Total links in pagination: {len(all_links)}")
                for i, link in enumerate(all_links[:10]):
                    print(f"[DEBUG]   Link {i+1}: aria-label='{link.get_attribute('aria-label')}', href='{link.get_attribute('href')}'")

    # Generate list of all page numbers from 1 to max_page
    page_numbers = list(range(1, max_page + 1))
    if debug:
        print(f"[DEBUG] Will process pages: {page_numbers}")

    # Scroll back to top after finding pagination
    driver.execute_script("window.scrollTo(0, 0);")
    time.sleep(1)

except Exception as e:
    if debug:
        print(f"[DEBUG] X No pagination found, extracting from single page: {e}")
        import traceback
        traceback.print_exc()
    page_numbers = [1]

# Extract products from each page
for page_num in page_numbers:
    if page_num > 1:
        # Build URL with page parameter
        # Check if URL already has query parameters
        if '?' in see_all_true_url:
            page_url = f"{see_all_true_url}&page={page_num}"
        else:
            page_url = f"{see_all_true_url}?page={page_num}"

        if debug:
            print(f"[DEBUG] Navigating to page {page_num}: {page_url}")

        try:
            driver.get(page_url)
            time.sleep(2)  # Reduced from 3 - optimized for faster navigation
            dismiss_overlays(driver)
            # Re-find containers after navigation
            listing_container = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, '[data-testid="listing-page-container"]')))
            grid = listing_container.find_element(By.CSS_SELECTOR, '[data-testid="product-grid-component"]')
            wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, 'h3[data-testid="product-title"]')))

            # Scroll to load products
            driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
            time.sleep(1.5)  # Reduced from 2
            driver.execute_script("window.scrollTo(0, 0);")
            time.sleep(0.8)  # Reduced from 1

            if debug:
                print(f"[DEBUG] ✓ Navigated to page {page_num}")
        except Exception as e:
            if debug:
                print(f"[DEBUG] X Could not navigate to page {page_num}: {e}")
            break

    if debug:
        print(f"\n[DEBUG] Extracting products from page {page_num}/{len(page_numbers)}")

    # Structure: listing-page-container > product-grid-component > div.css-0 (multiple) > div.css-yyn1h > product content
    # First find all div.css-0 elements within the grid
    css0_containers = grid.find_elements(By.CSS_SELECTOR, 'div.css-0')

    if debug:
        print(f"[DEBUG] Found {len(css0_containers)} containers with selector 'div.css-0' within grid")

    # Now find div.css-yyn1h within each div.css-0
    product_containers = []
    for css0 in css0_containers:
        try:
            css_yyn1h = css0.find_element(By.CSS_SELECTOR, 'div.css-yyn1h')
            product_containers.append(css_yyn1h)
        except:
            continue

    if debug:
        print(f"[DEBUG] Found {len(product_containers)} product containers (div.css-yyn1h) within div.css-0")

    # If still no containers, try finding by product-title directly
    if len(product_containers) == 0:
        if debug:
            print(f"[DEBUG] No containers found, trying to find by product titles...")
        product_titles = grid.find_elements(By.CSS_SELECTOR, 'h3[data-testid="product-title"]')
        if debug:
            print(f"[DEBUG] Found {len(product_titles)} product titles directly")

        for title in product_titles:
            try:
                # Find the css-yyn1h ancestor
                parent = title.find_element(By.XPATH, './ancestor::div[contains(@class, "css-yyn1h")]')
                if parent not in product_containers:
                    product_containers.append(parent)
            except:
                try:
                    # Fallback to chakra-linkbox
                    parent = title.find_element(By.XPATH, './ancestor::div[contains(@class, "chakra-linkbox")]')
                    if parent not in product_containers:
                        product_containers.append(parent)
                except:
                    pass

        if debug:
            print(f"[DEBUG] Found {len(product_containers)} containers via title ancestors")

    # Process containers for this page
    for idx, container in enumerate(product_containers):
        if debug and idx < 3:  # Debug first 3 containers per page
            print(f"\n[DEBUG] Processing container {idx + 1}/{len(product_containers)} on page {page_num}")

        try:
            product_data = {}

            # Title - h3 with data-testid="product-title" (inside the a.chakra-linkbox__overlay)
            try:
                title = container.find_element(By.CSS_SELECTOR, 'h3[data-testid="product-title"]')
                title_text = title.text.strip()
                if debug and idx < 3:
                    print(f"  [DEBUG] ✓ Title found: '{title_text[:50]}...'")
                if not title_text or title_text in seen_titles:
                    if debug and idx < 3:
                        print(f"  [DEBUG] ⚠ Skipping duplicate or empty title")
                    continue
                seen_titles.add(title_text)
                product_data["product-title"] = title_text
            except Exception as e:
                if debug and idx < 3:
                    print(f"  [DEBUG] X Title NOT found: {e}")
                continue  # Skip if no title found

            # Price - span with data-testid="regular-price" or "sale-price" (inside div[data-testid="price-product-tile"])
            try:
```

```
                    price_tile = container.find_element(By.CSS_SELECTOR, 'div[data-testid="price-product-tile"]')
                    if debug and idx < 3:
                        print(f"  [DEBUG] ✓ Price tile found")
                    # Try regular price first
                    try:
                        price = price_tile.find_element(By.CSS_SELECTOR, 'span[data-testid="regular-price"]')
                        price_text = price.text.strip()
                        # Clean up price text (remove "sale" prefix if present)
                        price_text = price_text.replace('sale', '').strip()
                        product_data["regular-price"] = price_text
                        if debug and idx < 3:
                            print(f"  [DEBUG] ✓ Regular price found: '{price_text}'")
                    except:
                        # Try sale price
                        try:
                            price = price_tile.find_element(By.CSS_SELECTOR, 'span[data-testid="sale-price"]')
                            price_text = price.text.strip()
                            price_text = price_text.replace('sale', '').strip()
                            product_data["regular-price"] = price_text
                            if debug and idx < 3:
                                print(f"  [DEBUG] ✓ Sale price found: '{price_text}'")
                        except:
                            if debug and idx < 3:
                                print(f"  [DEBUG] ✗ No price span found in price tile")
                except Exception as e:
                    if debug and idx < 3:
                        print(f"  [DEBUG] ✗ Price tile NOT found: {e}")

                # Size - p with data-testid="product-package-size" (inside the a.chakra-linkbox__overlay)
                try:
                    size = container.find_element(By.CSS_SELECTOR, 'p[data-testid="product-package-size"]')
                    product_data["product-package-size"] = size.text.strip()
                    if debug and idx < 3:
                        print(f"  [DEBUG] ✓ Size found: '{product_data['product-package-size']}'")
                except Exception as e:
                    if debug and idx < 3:
                        print(f"  [DEBUG] ✗ Size NOT found: {e}")

                # URL - try multiple selectors (prioritize div.css-qoklea > a)
                href = None
                # First try: a tag inside div.css-qoklea (as requested by user)
                try:
                    qoklea_div = container.find_element(By.CSS_SELECTOR, 'div.css-qoklea')
                    link = qoklea_div.find_element(By.CSS_SELECTOR, 'a[href]')
                    href = link.get_attribute('href')
                    if debug and idx < 3:
                        print(f"  [DEBUG]   Found link in div.css-qoklea, href='{href}'")
                except Exception as e:
                    if debug and idx < 3:
                        print(f"  [DEBUG]   div.css-qoklea > a not found: {e}")
                    # Second try: a tag with class chakra-linkbox__overlay
                    try:
                        link = container.find_element(By.CSS_SELECTOR, 'a.chakra-linkbox__overlay')
                        href = link.get_attribute('href')
                        if debug and idx < 3:
                            print(f"  [DEBUG]   Found link in chakra-linkbox__overlay, href='{href}'")
                    except:
                        # Third try: any a tag with href in the container
                        try:
                            link = container.find_element(By.CSS_SELECTOR, 'a[href]')
                            href = link.get_attribute('href')
                            if debug and idx < 3:
                                print(f"  [DEBUG]   Found any a[href] in container, href='{href}'")
                        except:
                            pass

                if href:
                    # Clean and normalize the URL
                    href = href.strip()
                    # If it's a relative URL, make it absolute
                    if not href.startswith('http'):
                        href = urljoin(ROOT_URL, href)
                    # Remove any fragments or ensure we have the full URL
                    if '#' in href:
                        href = href.split('#')[0]
                    product_data["product-url"] = href
                    if debug and idx < 3:
                        print(f"  [DEBUG] ✓ URL found: '{href}'")
                else:
                    if debug and idx < 3:
                        print(f"  [DEBUG] ✗ URL NOT found")

                # Note: extract_product_details is excluded from this pipeline

                # Only add if we have at least title
                if product_data.get("product-title"):
                    products.append(product_data)
                    if debug and idx < 3:
                        print(f"  [DEBUG] ✓ Product added to list")
            except Exception as e:
                if debug and idx < 3:
                    print(f"  [DEBUG] ✗ Error processing container: {e}")
                continue

        if debug:
            print(f"[DEBUG] Page {page_num}: Extracted {len([p for p in products if p.get('product-title')])} products so far")

    if debug:
        print(f"\n[DEBUG] Total products extracted: {len(products)}")

    return products
```

Test the function

In [14]:

```
### Test extract_products_from_grid - All Pages

# Test extract_products_from_grid with pagination
driver = get_driver(headless=False)
wait = WebDriverWait(driver, MAX_WAIT)

try:
    print("=" * 70)
    print("Testing extract_products_from_grid() - ALL PAGES")
    print("=" * 70)

    # Get a subcategory2 URL from the dropdown
    categories = extract_main_categories(driver, wait)
    category_name = "GROCERY"
    subcategory_name = "Dairy & Eggs"

    category_data = next((c for c in categories if category_name.upper() in c['name'].upper()), None)
    if not category_data or not category_data.get('aria-controls'):
        print(f"\n✗ Could not find {category_name} or aria-controls")
    else:
        aria_controls = category_data['aria-controls']

        # Get subcategory2 items
        subcategory2_list = extract_subcategory2_from_dropdown(driver, wait, category_name, subcategory_name, aria_controls)

        if not subcategory2_list:
            print(f"\n✗ No subcategory2 items found for {subcategory_name}")
        else:
            # Use the first subcategory2 URL
            test_url = subcategory2_list[0]['url']
            print(f"\n🎯 Testing URL: {test_url}")
            print(f"🎯 Category: {subcategory2_list[0]['name']}")
            print(f"\n{'=' * 70}")
            print("Starting extraction from ALL pages...")
            print(f"{'=' * 70}\n")

            # Test the function with debug enabled
            products = extract_products_from_grid(driver, wait, test_url, debug=True)

            print(f"\n{'=' * 70}")
            print(f"EXTRACTION SUMMARY")
            print(f"{'=' * 70}")
            print(f"✓ Total products extracted: {len(products)}")

            # Show sample products
            if len(products) > 0:
                print(f"\n📦 Sample products (first 10):")
                for i, product in enumerate(products[:10], 1):
                    print(f"  {i}. {product.get('product-title', 'N/A')}")
                    print(f"     Price: {product.get('regular-price', 'N/A')}")
                    print(f"     Size: {product.get('product-package-size', 'N/A')}")

                if len(products) > 10:
```

```
                print(f"\n  ... and {len(products) - 10} more products")

            # Count products with all fields
            complete_products = [p for p in products if p.get('product-title') and p.get('product-url')]
            print(f"\n📊 Statistics:")
            print(f"  - Products with title: {len([p for p in products if p.get('product-title')])}")
            print(f"  - Products with price: {len([p for p in products if p.get('regular-price')])}")
            print(f"  - Products with size: {len([p for p in products if p.get('product-package-size')])}")
            print(f"  - Products with URL: {len([p for p in products if p.get('product-url')])}")
            print(f"  - Complete products (title + URL): {len(complete_products)}")

            print(f"\n✓ Test PASSED! Successfully extracted {len(products)} products from all pages.")
        else:
            print(f"\n⚠ Test completed but no products found.")

except Exception as e:
    print(f"\n✗ Test FAILED: {e}")
    import traceback
    traceback.print_exc()
finally:
    driver.quit()
```

```
======================================================================
Testing extract_products_from_grid() - ALL PAGES
======================================================================

📍 Testing URL: https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?navid=flyout-L3-Milk-and-Cream
📍 Category: Milk & Cream

======================================================================
Starting extraction from ALL pages...
======================================================================


[DEBUG] Navigating to: https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?navid=flyout-L3-Milk-and-Cream
[DEBUG] ✓ Listing container found: [data-testid='listing-page-container']
[DEBUG] ✓ Grid found: [data-testid='product-grid-component']
[DEBUG] ✓ At least one product title found
[DEBUG] Scrolled 0 times
[DEBUG] ✓ Pagination container found
[DEBUG] Found 5 page links
[DEBUG]   Link: aria-label='Page 1', href='https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?page=1'
[DEBUG]   Link: aria-label='Page 2', href='https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?page=2'
[DEBUG]   Link: aria-label='Page 3', href='https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?page=3'
[DEBUG]   Link: aria-label='Page 4', href='https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?page=4'
[DEBUG]   Link: aria-label='Page 5', href='https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?page=5'
[DEBUG] ✓ Found pagination - Maximum page: 5 (from visible pages: [1, 2, 3, 4, 5])
[DEBUG] Will process pages: [1, 2, 3, 4, 5]

[DEBUG] Extracting products from page 1/5
[DEBUG] Found 96 containers with selector 'div.css-0' within grid
[DEBUG] Found 48 product containers (div.css-yyn1h) within div.css-0

[DEBUG] Processing container 1/48 on page 1
  [DEBUG] ✓ Title found: 'PürFiltre Milk 2% Partly Skimmed...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$6.00'
  [DEBUG] ✓ Size found: '2 l, $0.30/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/p-rfiltre-milk-2-partly-skimmed/p/21424991_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/p-rfiltre-milk-2-partly-skimmed/p/21424991_EA?source=nspt'
  [DEBUG] ✓ Product added to list

[DEBUG] Processing container 2/48 on page 1
  [DEBUG] ✓ Title found: 'Coffee Creamer, French Vanilla...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$7.00'
  [DEBUG] ✓ Size found: '946 ml, $0.74/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/coffee-creamer-french-vanilla/p/20895480002_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/coffee-creamer-french-vanilla/p/20895480002_EA?source=nspt'
  [DEBUG] ✓ Product added to list

[DEBUG] Processing container 3/48 on page 1
  [DEBUG] ✓ Title found: 'Almond for Coffee, Vanilla Flavour, Plant Based Da...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$5.29'
  [DEBUG] ✓ Size found: '890 ml, $0.59/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/almond-for-coffee-vanilla-flavour-plant-based-dair/p/21245360_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/almond-for-coffee-vanilla-flavour-plant-based-dair/p/21245360_EA?source=nspt'
  [DEBUG] ✓ Product added to list
[DEBUG] Page 1: Extracted 47 products so far
[DEBUG] Navigating to page 2: https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?navid=flyout-L3-Milk-and-Cream&page=2
[DEBUG] ✓ Navigated to page 2

[DEBUG] Extracting products from page 2/5
[DEBUG] Found 96 containers with selector 'div.css-0' within grid
[DEBUG] Found 48 product containers (div.css-yyn1h) within div.css-0

[DEBUG] Processing container 1/48 on page 2
  [DEBUG] ✓ Title found: '3.8% Milk...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$8.99'
  [DEBUG] ✓ Size found: '2 l, $0.45/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/3-8-milk/p/20117112_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/3-8-milk/p/20117112_EA?source=nspt'
  [DEBUG] ✓ Product added to list

[DEBUG] Processing container 2/48 on page 2
  [DEBUG] ✓ Title found: 'Oat Creamer, Original...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$6.99'
  [DEBUG] ✓ Size found: '946 ml, $0.74/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/oat-creamer-original/p/21576754_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/oat-creamer-original/p/21576754_EA?source=nspt'
  [DEBUG] ✓ Product added to list

[DEBUG] Processing container 3/48 on page 2
  [DEBUG] ✓ Title found: 'Goat Milk, 2% M.F...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$15.99'
  [DEBUG] ✓ Size found: '4 l, $0.40/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/goat-milk-2-m-f/p/20790474001_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/goat-milk-2-m-f/p/20790474001_EA?source=nspt'
  [DEBUG] ✓ Product added to list
[DEBUG] Page 2: Extracted 79 products so far
[DEBUG] Navigating to page 3: https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?navid=flyout-L3-Milk-and-Cream&page=3
[DEBUG] ✓ Navigated to page 3

[DEBUG] Extracting products from page 3/5
[DEBUG] Found 96 containers with selector 'div.css-0' within grid
[DEBUG] Found 48 product containers (div.css-yyn1h) within div.css-0

[DEBUG] Processing container 1/48 on page 3
  [DEBUG] ✓ Title found: '1% Strawberry Partly Skimmed Milk...'
  [DEBUG] ⚠ Skipping duplicate or empty title

[DEBUG] Processing container 2/48 on page 3
  [DEBUG] ✓ Title found: '1% Vanilla Partly Skimmed Milk...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$2.49'
  [DEBUG] ✓ Size found: '473 ml, $0.53/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/1-vanilla-partly-skimmed-milk/p/20895935005_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/1-vanilla-partly-skimmed-milk/p/20895935005_EA?source=nspt'
  [DEBUG] ✓ Product added to list

[DEBUG] Processing container 3/48 on page 3
  [DEBUG] ✓ Title found: '10% Cream...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$5.08'
  [DEBUG] ✓ Size found: '1 l, $0.51/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/10-cream/p/20067088_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/10-cream/p/20067088_EA?source=nspt'
  [DEBUG] ✓ Product added to list
[DEBUG] Page 3: Extracted 103 products so far
[DEBUG] Navigating to page 4: https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?navid=flyout-L3-Milk-and-Cream&page=4
[DEBUG] ✓ Navigated to page 4

[DEBUG] Extracting products from page 4/5
[DEBUG] Found 96 containers with selector 'div.css-0' within grid
[DEBUG] Found 48 product containers (div.css-yyn1h) within div.css-0

[DEBUG] Processing container 1/48 on page 4
  [DEBUG] ✓ Title found: 'Café Mocha Coffee & Hot Chocolate Mix Instant Coff...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$7.99'
  [DEBUG] ✓ Size found: '8 ea, $1.00/1ea'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/caf-mocha-coffee-hot-chocolate-mix-instant-coffee/p/21431715_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/caf-mocha-coffee-hot-chocolate-mix-instant-coffee/p/21431715_EA?source=nspt'
  [DEBUG] ✓ Product added to list
```

```
[DEBUG] Processing container 2/48 on page 4
  [DEBUG] ✓ Title found: 'Caramilk Chocolate Milkshake...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$2.99'
  [DEBUG] ✓ Size found: '310 ml, $0.96/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/caramilk-chocolate-milkshake/p/21040942_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/caramilk-chocolate-milkshake/p/21040942_EA?source=nspt'
  [DEBUG] ✓ Product added to list

[DEBUG] Processing container 3/48 on page 4
  [DEBUG] ✓ Title found: 'Chocolate High Protein Shake...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$3.69'
  [DEBUG] ✓ Size found: '460 ml, $0.80/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/chocolate-high-protein-shake/p/21345511_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/chocolate-high-protein-shake/p/21345511_EA?source=nspt'
  [DEBUG] ✓ Product added to list
[DEBUG] Page 4: Extracted 139 products so far
[DEBUG] Navigating to page 5: https://www.loblaws.ca/en/food/dairy-eggs/milk-cream/c/28224?navid=flyout-L3-Milk-and-Cream&page=5
[DEBUG] ✓ Navigated to page 5

[DEBUG] Extracting products from page 5/5
[DEBUG] Found 88 containers with selector 'div.css-0' within grid
[DEBUG] Found 44 product containers (div.css-yyn1h) within div.css-0

[DEBUG] Processing container 1/44 on page 5
  [DEBUG] ✓ Title found: 'Lactose Free 2% Milk...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$6.00'
  [DEBUG] ✓ Size found: '2 l, $0.30/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/lactose-free-2-milk/p/21441429_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/lactose-free-2-milk/p/21441429_EA?source=nspt'
  [DEBUG] ✓ Product added to list

[DEBUG] Processing container 2/44 on page 5
  [DEBUG] ✓ Title found: 'Lactose Free Chocolate Milk...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$7.48'
  [DEBUG] ✓ Size found: '2 l, $0.37/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/lactose-free-chocolate-milk/p/21040983_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/lactose-free-chocolate-milk/p/21040983_EA?source=nspt'
  [DEBUG] ✓ Product added to list

[DEBUG] Processing container 3/44 on page 5
  [DEBUG] ✓ Title found: 'Lactose Free Fat Free Skim 0% Dairy Product (2L)...'
  [DEBUG] ✓ Price tile found
  [DEBUG] ✓ Regular price found: '$7.14'
  [DEBUG] ✓ Size found: '2 l, $0.36/100ml'
  [DEBUG]   Found link in div.css-qoklea, href='https://www.loblaws.ca/en/lactose-free-fat-free-skim-0-dairy-product-2l/p/20077874004_EA?source=nspt'
  [DEBUG] ✓ URL found: 'https://www.loblaws.ca/en/lactose-free-fat-free-skim-0-dairy-product-2l/p/20077874004_EA?source=nspt'
  [DEBUG] ✓ Product added to list
[DEBUG] Page 5: Extracted 178 products so far

[DEBUG] Total products extracted: 178

======================================================================
EXTRACTION SUMMARY
======================================================================
✓ Total products extracted: 178

📦 Sample products (first 10):
  1. PürFiltre Milk 2% Partly Skimmed
      Price: $6.00
      Size: 2 l, $0.30/100ml
  2. Coffee Creamer, French Vanilla
      Price: $7.00
      Size: 946 ml, $0.74/100ml
  3. Almond for Coffee, Vanilla Flavour, Plant Based Dairy Free Coffee Creamer 890ml
      Price: $5.29
      Size: 890 ml, $0.59/100ml
  4. Coffee Creamer, Hazelnut
      Price: $7.00
      Size: 946 ml, $0.74/100ml
  5. Oat for Coffee, Vanilla, Plant Based, Dairy Free Coffee Creamer
      Price: $5.29
      Size: 890 ml, $0.59/100ml
  6. Oat Coffee Creamer, Peppermint Mocha, Limited Edition
      Price: $2.79
      Size: 450 ml, $0.62/100ml
  7. Vanilla Toffee Caramel Coffee Creamer, 63 Servings
      Price: $7.00
      Size: 946 ml, $0.74/100ml
  8. Chocolate Milk
      Price: $3.70
      Size: 1 l, $0.37/100ml
  9. Coffee Creamer, Southern Butter Pecan
      Price: $7.00
      Size: 946 ml, $0.74/100ml
  10. French Vanilla Flavoured Fat Free Coffee Creamer, 63 Servings
      Price: $7.00
      Size: 946 ml, $0.74/100ml

  ... and 168 more products

📊 Statistics:
  - Products with title: 178
  - Products with price: 178
  - Products with size: 178
  - Products with URL: 178
  - Complete products (title + URL): 178

✓ Test PASSED! Successfully extracted 178 products from all pages.
```

Extracting product details

In [15]:

```python
def extract_product_details(driver, wait, product_url):
    """Extract product details from product page."""
    driver.get(product_url)
    time.sleep(2)
    dismiss_overlays(driver)

    details = {"nutrition-info": "", "description": "", "legal-disclaimer": ""}

    try:
        container = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, '[data-testid="product-details-page-details"], [data-testid="product-details-accordion"]')))
    except:
        container = driver.find_element(By.CSS_SELECTOR, 'body')

    # Nutrition
    try:
        nutrition = container.find_element(By.CSS_SELECTOR, 'div.product-details-page-nutrition-info, [class*="nutrition"]')
        details["nutrition-info"] = nutrition.text.strip()
    except:
        pass

    # Description
    try:
        desc = container.find_element(By.CSS_SELECTOR, 'div.product-details-page-description, [class*="description"]')
        details["description"] = desc.text.strip()
    except:
        pass

    # Disclaimer
    try:
        disclaimer = container.find_element(By.CSS_SELECTOR, 'div.product-details-page-legal-disclaimer, [class*="disclaimer"]')
        details["legal-disclaimer"] = disclaimer.text.strip()
    except:
        pass

    return details
```

Test the function

Pipeline: Extract all products from GROCERY and HOME, BEAUTY & BABY

In [18]:

```python
### Improved JSON Export Function

def export_products_to_json(products, output_file, extraction_stats, target_categories):
    """
    Export products to a clean, well-structured JSON format matching the test output.

    Args:
        products: List of product dictionaries
        output_file: Path to output JSON file
        extraction_stats: Dictionary with extraction statistics
        target_categories: List of target categories processed
```

```
    Returns:
        dict: The output data structure
    """
    # Clean and structure the products data - match the test output format
    cleaned_products = []
    for product in products:
        cleaned_product = {
            "product-title": product.get("product-title", ""),
            "regular-price": product.get("regular-price", ""),
            "product-package-size": product.get("product-package-size", ""),
            "product-url": product.get("product-url", ""),
            "category": product.get("category", ""),
            "subcategory": product.get("subcategory", ""),
            "subcategory2": product.get("subcategory2", "")
        }
        # Only add products that have at least a title
        if cleaned_product["product-title"]:
            cleaned_products.append(cleaned_product)

    # Create well-structured JSON output matching test format
    output_data = {
        "extraction_date": time.strftime("%Y-%m-%d %H:%M:%S"),
        "extraction_summary": {
            "categories_processed": target_categories,
            "total_categories": extraction_stats.get("categories_processed", 0),
            "total_subcategories": extraction_stats.get("subcategories_processed", 0),
            "total_subcategory2": extraction_stats.get("subcategory2_processed", 0),
            "total_products": len(cleaned_products),
            "errors_count": len(extraction_stats.get("errors", []))
        },
        "statistics": {
            "products_with_title": len([p for p in cleaned_products if p.get("product-title")]),
            "products_with_price": len([p for p in cleaned_products if p.get("regular-price")]),
            "products_with_size": len([p for p in cleaned_products if p.get("product-package-size")]),
            "products_with_url": len([p for p in cleaned_products if p.get("product-url")]),
            "complete_products": len([p for p in cleaned_products if p.get("product-title") and p.get("product-url")])
        },
        "products": cleaned_products
    }

    # Add errors if any (limit to first 50 to keep file size manageable)
    if extraction_stats.get("errors"):
        output_data["errors"] = extraction_stats["errors"][:50]

    # Write to JSON file
    with open(output_file, 'w', encoding='utf-8') as f:
        json.dump(output_data, f, indent=2, ensure_ascii=False)

    print(f"✓ Data exported to: {output_file}")
    print(f"✓ Total products: {len(cleaned_products)}")
    print(f"✓ Products with title: {output_data['statistics']['products_with_title']}")
    print(f"✓ Products with price: {output_data['statistics']['products_with_price']}")
    print(f"✓ Products with size: {output_data['statistics']['products_with_size']}")
    print(f"✓ Products with URL: {output_data['statistics']['products_with_url']}")
    print(f"✓ Complete products: {output_data['statistics']['complete_products']}")

    return output_data
```

In [19]:

```
def extract_all_products_pipeline(driver, wait, output_file="loblaws_products.json", debug=False):
    """
    Pipeline to extract all products from GROCERY and HOME, BEAUTY & BABY categories.

    Args:
        driver: Selenium WebDriver instance
        wait: WebDriverWait instance
        output_file: Path to output JSON file
        debug: Enable debug output

    Returns:
        dict: Summary of extraction with total products and statistics
    """
    print("=" * 70)
    print("STARTING PRODUCT EXTRACTION PIPELINE")
    print("=" * 70)
    print(f"Target categories: {TARGET_CATEGORIES}")
    print(f"Output file: {output_file}")
    print("=" * 70)

    all_products = []
    extraction_stats = {
        "categories_processed": 0,
        "subcategories_processed": 0,
        "subcategory2_processed": 0,
        "total_products": 0,
        "errors": []
    }

    try:
        # Step 1: Get main categories
        if debug:
            print("\n[STEP 1] Extracting main categories...")
        categories = extract_main_categories(driver, wait)

        if debug:
            print(f"✓ Found {len(categories)} main categories")

        # Step 2: Filter for target categories
        target_category_data = []
        for cat in categories:
            for target in TARGET_CATEGORIES:
                if target.upper() in cat['name'].upper():
                    target_category_data.append(cat)
                    break

        if debug:
            print(f"✓ Found {len(target_category_data)} target categories:")
            for cat in target_category_data:
                print(f"  - {cat['name']}")

        if not target_category_data:
            print("\n✗ No target categories found!")
            return extraction_stats

        # Step 3: Process each target category
        for category_idx, category_data in enumerate(target_category_data, 1):
            category_name = category_data['name']
            aria_controls = category_data.get('aria-controls')

            if not aria_controls:
                error_msg = f"Category '{category_name}' has no aria-controls"
                extraction_stats["errors"].append(error_msg)
                if debug:
                    print(f"\n⚠ {error_msg}")
                continue

            print(f"\n{'=' * 70}")
            print(f"PROCESSING CATEGORY {category_idx}/{len(target_category_data)}: {category_name}")
            print(f"{'=' * 70}")

            extraction_stats["categories_processed"] += 1

            # Step 4: Get subcategories for this category
            if debug:
                print(f"\n[STEP 4] Extracting subcategories for '{category_name}'...")

            try:
                subcategories = extract_subcategories(driver, wait, category_name, aria_controls)

                if debug:
                    print(f"✓ Found {len(subcategories)} subcategories")

                if not subcategories:
                    if debug:
                        print(f"⚠ No subcategories found for '{category_name}'")
                    continue

                # Step 5: OPTIMIZED - Extract all subcategory2 items for all subcategories in one batch
                # This avoids repeatedly opening/closing the dropdown
                if debug:
                    print(f"\n[STEP 5] Batch extracting subcategory2 items for all {len(subcategories)} subcategories...")

                try:
                    subcategory2_map = extract_all_subcategory2_for_category(
                        driver, wait, category_name, subcategories, aria_controls
                    )

                    if debug:
```

```
                    total_subcat2 = sum(len(items) for items in subcategory2_map.values())
                    print(f"✓ Batch extraction complete: Found {total_subcat2} subcategory2 items across {len(subcategory2_map)} subcategories")

                # Step 6: Process all subcategory2 items and extract products
                # Now we can process all products sequentially without going back to dropdown
                total_subcat2_count = sum(len(items) for items in subcategory2_map.values())
                current_subcat2_count = 0

                for subcat_idx, subcategory in enumerate(subcategories, 1):
                    subcategory_name = subcategory['name']
                    subcategory2_list = subcategory2_map.get(subcategory_name, [])

                    if not subcategory2_list:
                        if debug:
                            print(f"\n  ⚠ No subcategory2 items found for '{subcategory_name}'")
                        extraction_stats["subcategories_processed"] += 1
                        continue

                    print(f"\n  Processing subcategory {subcat_idx}/{len(subcategories)}: {subcategory_name} ({len(subcategory2_list)} subcategory2 items)")
                    extraction_stats["subcategories_processed"] += 1

                    # Process each subcategory2 and extract products
                    for subcat2_idx, subcat2_item in enumerate(subcategory2_list, 1):
                        current_subcat2_count += 1
                        subcat2_name = subcat2_item['name']
                        subcat2_url = subcat2_item['url']

                        print(f"    [{current_subcat2_count}/{total_subcat2_count}] Extracting products from: {subcat2_name}")

                        extraction_stats["subcategory2_processed"] += 1

                        try:
                            # Extract products from all pages
                            products = extract_products_from_grid(driver, wait, subcat2_url, debug=False)

                            # Add metadata to each product
                            for product in products:
                                product["category"] = category_name
                                product["subcategory"] = subcategory_name
                                product["subcategory2"] = subcat2_name

                            all_products.extend(products)

                            print(f"      ✓ Extracted {len(products)} products")

                        except Exception as e:
                            error_msg = f"Error extracting products from '{subcat2_name}': {str(e)}"
                            extraction_stats["errors"].append(error_msg)
                            if debug:
                                print(f"      ✗ {error_msg}")
                                import traceback
                                traceback.print_exc()
                            else:
                                print(f"      ✗ Error: {str(e)[:100]}")
                            continue

            except Exception as e:
                error_msg = f"Error in batch extracting subcategory2 for '{category_name}': {str(e)}"
                extraction_stats["errors"].append(error_msg)
                if debug:
                    print(f"  ✗ {error_msg}")
                    import traceback
                    traceback.print_exc()
                # Fallback: try individual extraction for each subcategory
                print(f"  ⚠ Falling back to individual subcategory2 extraction...")
                for subcat_idx, subcategory in enumerate(subcategories, 1):
                    subcategory_name = subcategory['name']
                    extraction_stats["subcategories_processed"] += 1

                    try:
                        subcategory2_list = extract_subcategory2_from_dropdown(
                            driver, wait, category_name, subcategory_name, aria_controls
                        )

                        if not subcategory2_list:
                            continue

                        for subcat2_item in subcategory2_list:
                            subcat2_name = subcat2_item['name']
                            subcat2_url = subcat2_item['url']
                            extraction_stats["subcategory2_processed"] += 1

                            try:
                                products = extract_products_from_grid(driver, wait, subcat2_url, debug=False)
                                for product in products:
                                    product["category"] = category_name
                                    product["subcategory"] = subcategory_name
                                    product["subcategory2"] = subcat2_name
                                all_products.extend(products)
                                print(f"      ✓ Extracted {len(products)} products from {subcat2_name}")
                            except Exception as e:
                                extraction_stats["errors"].append(f"Error extracting products from '{subcat2_name}': {str(e)}")
                                continue
                    except:
                        continue

        except Exception as e:
            error_msg = f"Error extracting subcategories for '{category_name}': {str(e)}"
            extraction_stats["errors"].append(error_msg)
            if debug:
                print(f"  ✗ {error_msg}")
                import traceback
                traceback.print_exc()
            continue

    # Step 8: Export to JSON
    extraction_stats["total_products"] = len(all_products)

    print(f"\n{'=' * 70}")
    print("EXPORTING DATA TO JSON")
    print(f"{'=' * 70}")

    output_data = {
        "extraction_date": time.strftime("%Y-%m-%d %H:%M:%S"),
        "categories_processed": TARGET_CATEGORIES,
        "statistics": extraction_stats,
        "products": all_products
    }

    with open(output_file, 'w', encoding='utf-8') as f:
        json.dump(output_data, f, indent=2, ensure_ascii=False)

    print(f"✓ Data exported to: {output_file}")
    print(f"✓ Total products: {len(all_products)}")

    # Print summary
    print(f"\n{'=' * 70}")
    print("EXTRACTION SUMMARY")
    print(f"{'=' * 70}")
    print(f"Categories processed: {extraction_stats['categories_processed']}")
    print(f"Subcategories processed: {extraction_stats['subcategories_processed']}")
    print(f"Subcategory2 processed: {extraction_stats['subcategory2_processed']}")
    print(f"Total products extracted: {extraction_stats['total_products']}")
    print(f"Errors encountered: {len(extraction_stats['errors'])}")

    if extraction_stats['errors']:
        print(f"\n⚠ Errors encountered:")
        for i, error in enumerate(extraction_stats['errors'][:10], 1):
            print(f"  {i}. {error}")
        if len(extraction_stats['errors']) > 10:
            print(f"  ... and {len(extraction_stats['errors']) - 10} more errors")

    print(f"\n{'=' * 70}")
    print("PIPELINE COMPLETED SUCCESSFULLY")
    print(f"{'=' * 70}")

    return extraction_stats

except Exception as e:
    error_msg = f"Pipeline failed: {str(e)}"
    extraction_stats["errors"].append(error_msg)
    print(f"\n✗ {error_msg}")
    import traceback
    traceback.print_exc()
    return extraction_stats
```

In [20]:

```
### Run the pipeline
```

```python
# Initialize driver and wait
driver = get_driver(headless=False)
wait = WebDriverWait(driver, MAX_WAIT)

try:
    # Run the pipeline
    stats = extract_all_products_pipeline(
        driver,
        wait,
        output_file="loblaws_products.json",
        debug=False  # Set to True for detailed debug output
    )

    print("\n✓ Pipeline execution completed!")

except Exception as e:
    print(f"\n✗ Pipeline execution failed: {e}")
    import traceback
    traceback.print_exc()
finally:
    driver.quit()
    print("\n✓ Driver closed")
```

```
======================================================================
STARTING PRODUCT EXTRACTION PIPELINE
======================================================================
Target categories: ['GROCERY', 'HOME, BEAUTY & BABY']
Output file: loblaws_products.json
======================================================================

======================================================================
PROCESSING CATEGORY 1/2: GROCERY
======================================================================

  Processing subcategory 1/15: Fruits & Vegetables (7 subcategory2 items)
    [1/157] Extracting products from: Fresh Vegetables
      ✓ Extracted 317 products
    [2/157] Extracting products from: Fresh Fruits
      ✓ Extracted 175 products
    [3/157] Extracting products from: Packaged Salads & Dressing
      ✓ Extracted 120 products
    [4/157] Extracting products from: Herbs
      ✓ Extracted 0 products
    [5/157] Extracting products from: Fresh Cut Fruits & Vegetables
      ✓ Extracted 65 products
    [6/157] Extracting products from: Dried Fruits & Nuts
      ✓ Extracted 53 products
    [7/157] Extracting products from: Fresh Juice & Smoothies
      ✓ Extracted 80 products

  Processing subcategory 2/15: Dairy & Eggs (9 subcategory2 items)
    [8/157] Extracting products from: Milk & Cream
      ✓ Extracted 178 products
    [9/157] Extracting products from: Egg & Egg Substitutes
      ✓ Extracted 42 products
    [10/157] Extracting products from: Butter & Spreads
      ✓ Extracted 68 products
    [11/157] Extracting products from: Cheese
      ✓ Extracted 262 products
    [12/157] Extracting products from: Yogurt
      ✓ Extracted 339 products
    [13/157] Extracting products from: Desserts & Doughs
      ✓ Extracted 45 products
    [14/157] Extracting products from: Sour Cream & Dips
      ✓ Extracted 25 products
    [15/157] Extracting products from: Lactose Free
      ✓ Extracted 40 products
    [16/157] Extracting products from: Non-Dairy Milk Alternatives
      ✓ Extracted 102 products

  Processing subcategory 3/15: Meat (11 subcategory2 items)
    [17/157] Extracting products from: Chicken & Turkey
      ✓ Extracted 173 products
    [18/157] Extracting products from: Beef
      ✓ Extracted 129 products
    [19/157] Extracting products from: Sausages
      ✓ Extracted 65 products
    [20/157] Extracting products from: Bacon
      ✓ Extracted 38 products
    [21/157] Extracting products from: Hot Dogs
      ✓ Extracted 31 products
    [22/157] Extracting products from: Pork & Ham
      ✓ Extracted 70 products
    [23/157] Extracting products from: Lamb & Veal
      ✓ Extracted 35 products
    [24/157] Extracting products from: Kebabs & Marinated Meat
      ✓ Extracted 39 products
    [25/157] Extracting products from: Game Meat, Offals & Fowl
      ✓ Extracted 20 products
    [26/157] Extracting products from: Plant-Based Meat Alternatives
      ✓ Extracted 28 products
    [27/157] Extracting products from: Deli Meat
      ✓ Extracted 228 products

  Processing subcategory 4/15: Pantry (13 subcategory2 items)
    [28/157] Extracting products from: Canned & Pickled
      ✓ Extracted 759 products
    [29/157] Extracting products from: Baking Essentials
      ✓ Extracted 670 products
    [30/157] Extracting products from: Pasta & Pasta Sauce
      ✓ Extracted 618 products
    [31/157] Extracting products from: Easy Meals & Sides
      ✓ Extracted 280 products
    [32/157] Extracting products from: Cereal & Breakfast
      ✓ Extracted 235 products
    [33/157] Extracting products from: Honey, Syrups & Spreads
      ✓ Extracted 179 products
    [34/157] Extracting products from: Rice
      ✓ Extracted 158 products
    [35/157] Extracting products from: Oils & Vinegar
      ✓ Extracted 210 products
    [36/157] Extracting products from: Condiments
      ✓ Extracted 942 products
    [37/157] Extracting products from: Spices & Seasonings
      ✓ Extracted 435 products
    [38/157] Extracting products from: Dried Beans, Vegetables & Grains
      ✓ Extracted 54 products
    [39/157] Extracting products from: Bulk Nuts and Candy
      ✓ Extracted 261 products
    [40/157] Extracting products from: International Foods
      ✓ Extracted 261 products

  Processing subcategory 5/15: International Foods (47 subcategory2 items)
    [41/157] Extracting products from: East Asian Foods
      ✓ Extracted 620 products
    [42/157] Extracting products from: Rice & Grain
      ✓ Extracted 26 products
    [43/157] Extracting products from: Noodles & Noodle Soup
      ✓ Extracted 104 products
    [44/157] Extracting products from: Snacks, Cookies & Crackers
      ✓ Extracted 91 products
    [45/157] Extracting products from: Condiments, Sauces & Oil
      ✓ Extracted 95 products
    [46/157] Extracting products from: Shop More East Asian Foods
      ✓ Extracted 625 products
    [47/157] Extracting products from: South Asian Foods
      ✓ Extracted 0 products
    [48/157] Extracting products from: Rice, Lentils & Beans
      ✓ Extracted 35 products
    [49/157] Extracting products from: Snacks, Cookies & Crackers
      ✓ Extracted 43 products
    [50/157] Extracting products from: Bakery
      ✓ Extracted 19 products
    [51/157] Extracting products from: Spices & Seasonings
      ✓ Extracted 54 products
    [52/157] Extracting products from: Shop More South Asian Foods
      ✓ Extracted 325 products
    [53/157] Extracting products from: Caribbean Foods
      ✓ Extracted 112 products
    [54/157] Extracting products from: Rice, Lentils & Beans
      ✓ Extracted 4 products
    [55/157] Extracting products from: Snacks, Cookies & Crackers
      ✓ Extracted 4 products
    [56/157] Extracting products from: Baking & Flour
      ✓ Extracted 4 products
    [57/157] Extracting products from: Spices & Seasonings
      ✓ Extracted 7 products
    [58/157] Extracting products from: Shop More Caribbean Foods
      ✓ Extracted 112 products
```

```
  [59/157] Extracting products from: Halal Foods
    ✓ Extracted 73 products
  [60/157] Extracting products from: Meat
    ✓ Extracted 59 products
  [61/157] Extracting products from: Deli
    ✓ Extracted 8 products
  [62/157] Extracting products from: Dairy
    ✓ Extracted 11 products
  [63/157] Extracting products from: Frozen Entrees
    ✓ Extracted 0 products
  [64/157] Extracting products from: Middle Eastern Foods
    ✓ Extracted 145 products
  [65/157] Extracting products from: Canned, Pickled Food & Olives
    ✓ Extracted 24 products
  [66/157] Extracting products from: Snacks, Cookies & Crackers
    ✓ Extracted 30 products
  [67/157] Extracting products from: Frozen
    ✓ Extracted 0 products
  [68/157] Extracting products from: Shop More Middle Eastern Foods
    ✓ Extracted 145 products
  [69/157] Extracting products from: Filipino Foods
    ✓ Extracted 71 products
  [70/157] Extracting products from: Rice, Noodles & Noodle Soup
    ✓ Extracted 15 products
  [71/157] Extracting products from: Snacks, Cookies & Crackers
    ✓ Extracted 2 products
  [72/157] Extracting products from: Canned Foods
    ✓ Extracted 19 products
  [73/157] Extracting products from: Shop More Filipino Foods
    ✓ Extracted 71 products
  [74/157] Extracting products from: Latin American Foods
    ✓ Extracted 52 products
  [75/157] Extracting products from: Condiments & Salsa
    ✓ Extracted 30 products
  [76/157] Extracting products from: Hot & Cold Drinks
    ✓ Extracted 6 products
  [77/157] Extracting products from: Fruits & Vegetables
    ✓ Extracted 6 products
  [78/157] Extracting products from: Shop More Latin American Foods
    ✓ Extracted 52 products
  [79/157] Extracting products from: European Foods
    ✓ Extracted 179 products
  [80/157] Extracting products from: Canned Goods
    ✓ Extracted 17 products
  [81/157] Extracting products from: Bakery & Deli
    ✓ Extracted 29 products
  [82/157] Extracting products from: Snacks, Cookies & Crackers
    ✓ Extracted 18 products
  [83/157] Extracting products from: Shop More European Foods
    ✓ Extracted 179 products
  [84/157] Extracting products from: Kosher Foods
    ✓ Extracted 109 products
  [85/157] Extracting products from: Snacks, Cookies & Crackers
    ✓ Extracted 66 products
  [86/157] Extracting products from: Baking, Breakfast & Dressings
    ✓ Extracted 23 products
  [87/157] Extracting products from: Meat & Seafood
    ✓ Extracted 0 products

Processing subcategory 6/15: Snacks, Chips & Candy (5 subcategory2 items)
  [88/157] Extracting products from: Chips & Snacks
    ✓ Extracted 1076 products
  [89/157] Extracting products from: Candy & Chocolate
    ✓ Extracted 809 products
  [90/157] Extracting products from: Crackers & Cookies
    ✓ Extracted 639 products
  [91/157] Extracting products from: Snack Cakes
    ✓ Extracted 17 products
  [92/157] Extracting products from: Bulk Nuts and Candy
    ✓ Extracted 262 products

Processing subcategory 7/15: Frozen Food (9 subcategory2 items)
  [93/157] Extracting products from: Ice Cream & Desserts
    ✓ Extracted 454 products
  [94/157] Extracting products from: Frozen Fruit & Vegetables
    ✓ Extracted 147 products
  [95/157] Extracting products from: Frozen Meat and Seafood
    ✓ Extracted 326 products
  [96/157] Extracting products from: Frozen Meals, Entrees & Sides
    ✓ Extracted 384 products
  [97/157] Extracting products from: Frozen Pizza
    ✓ Extracted 100 products
  [98/157] Extracting products from: Frozen Appetizers & Snacks
    ✓ Extracted 83 products
  [99/157] Extracting products from: Frozen Bakery & Breakfast
    ✓ Extracted 112 products
  [100/157] Extracting products from: Frozen Beverages & Ice
    ✓ Extracted 13 products
  [101/157] Extracting products from: Frozen Meatless Alternatives
    ✓ Extracted 10 products

Processing subcategory 8/15: Natural and Organic (14 subcategory2 items)
  [102/157] Extracting products from: Cereals, Spreads & Syrups
    ✓ Extracted 214 products
  [103/157] Extracting products from: Bakery
    ✓ Extracted 32 products
  [104/157] Extracting products from: Condiments, Sauces and Oils
    ✓ Extracted 98 products
  [105/157] Extracting products from: Snacks, Chips & Candy
    ✓ Extracted 489 products
  [106/157] Extracting products from: Dairy and Eggs
    ✓ Extracted 256 products
  [107/157] Extracting products from: Drinks
    ✓ Extracted 317 products
  [108/157] Extracting products from: Frozen Foods
    ✓ Extracted 155 products
  [109/157] Extracting products from: Baking and Spices
    ✓ Extracted 204 products
  [110/157] Extracting products from: Canned
    ✓ Extracted 44 products
  [111/157] Extracting products from: Pasta and Side Dishes
    ✓ Extracted 208 products
  [112/157] Extracting products from: Bars and Protein
    ✓ Extracted 326 products
  [113/157] Extracting products from: Health & Beauty
    ✓ Extracted 242 products
  [114/157] Extracting products from: Household Supplies
    ✓ Extracted 46 products
  [115/157] Extracting products from: Vitamins, Minerals and Supplements
    ✓ Extracted 413 products

Processing subcategory 9/15: Bakery (6 subcategory2 items)
  [116/157] Extracting products from: Bread
    ✓ Extracted 219 products
  [117/157] Extracting products from: Buns & Rolls
    ✓ Extracted 68 products
  [118/157] Extracting products from: Cookies, Muffins & Desserts
    ✓ Extracted 186 products
  [119/157] Extracting products from: Bagels, Croissants & English Muffins
    ✓ Extracted 53 products
  [120/157] Extracting products from: Wraps, Flatbread & Pizza Crust
    ✓ Extracted 59 products
  [121/157] Extracting products from: Cakes
    ✓ Extracted 146 products

Processing subcategory 10/15: Prepared Meals (10 subcategory2 items)
  [122/157] Extracting products from: Rotisserie & Fried Chicken
    ✓ Extracted 36 products
  [123/157] Extracting products from: Fresh Pasta & Sauce
    ✓ Extracted 37 products
  [124/157] Extracting products from: Entrees & Appetizers
    ✓ Extracted 80 products
  [125/157] Extracting products from: Salads & Soups
    ✓ Extracted 61 products
  [126/157] Extracting products from: Sandwiches
    ✓ Extracted 36 products
  [127/157] Extracting products from: Sushi
    ✓ Extracted 45 products
  [128/157] Extracting products from: Fries & Sides
    ✓ Extracted 11 products
  [129/157] Extracting products from: Pizza
    ✓ Extracted 10 products
  [130/157] Extracting products from: Quiches & Pies
    ✓ Extracted 13 products
  [131/157] Extracting products from: Snacks & Dips
    ✓ Extracted 12 products

Processing subcategory 11/15: Drinks (9 subcategory2 items)
  [132/157] Extracting products from: Juice
```

```
  [132/157] Extracting products from: Juice
    ✓ Extracted 330 products
  [133/157] Extracting products from: Coffee
    ✓ Extracted 308 products
  [134/157] Extracting products from: Tea & Hot Drinks
    ✓ Extracted 313 products
  [135/157] Extracting products from: Soft Drinks
    ✓ Extracted 181 products
  [136/157] Extracting products from: Water
    ✓ Extracted 210 products
  [137/157] Extracting products from: Soy, Rice & Almond Drinks
    ✓ Extracted 49 products
  [138/157] Extracting products from: Sports & Energy
    ✓ Extracted 158 products
  [139/157] Extracting products from: Drink Mixes
    ✓ Extracted 130 products
  [140/157] Extracting products from: Non-alcoholic drinks
    ✓ Extracted 75 products

 Processing subcategory 12/15: Deli (7 subcategory2 items)
  [141/157] Extracting products from: Deli Meat
    ✓ Extracted 217 products
  [142/157] Extracting products from: Deli Cheese
    ✓ Extracted 359 products
  [143/157] Extracting products from: Antipasto, Dips & Spreads
    ✓ Extracted 89 products
  [144/157] Extracting products from: Crackers & Condiments
    ✓ Extracted 192 products
  [145/157] Extracting products from: Vegan & Vegetarian
    ✓ Extracted 56 products
  [146/157] Extracting products from: Lunch & Snack Kits
    ✓ Extracted 23 products
  [147/157] Extracting products from: Party Trays
    ✓ Extracted 21 products

 Processing subcategory 13/15: Fish & Seafood (7 subcategory2 items)
  [148/157] Extracting products from: Shrimp
    ✓ Extracted 36 products
  [149/157] Extracting products from: Salmon
    ✓ Extracted 30 products
  [150/157] Extracting products from: Fish
    ✓ Extracted 84 products
  [151/157] Extracting products from: Smoked Fish
    ✓ Extracted 20 products
  [152/157] Extracting products from: Seafood Appetizers
    ✓ Extracted 49 products
  [153/157] Extracting products from: Shellfish
    ✓ Extracted 61 products
  [154/157] Extracting products from: Squid & Octopus
    ✓ Extracted 8 products

 Processing subcategory 14/15: Beer & Wine (3 subcategory2 items)
  [155/157] Extracting products from: Beer
    ✓ Extracted 78 products
  [156/157] Extracting products from: Wine
    ✓ Extracted 0 products
  [157/157] Extracting products from: Coolers & Ciders
    ✓ Extracted 21 products

======================================================================
PROCESSING CATEGORY 2/2: HOME, BEAUTY & BABY
======================================================================

======================================================================
EXPORTING DATA TO JSON
======================================================================
✓ Data exported to: loblaws_products.json
✓ Total products: 23194

======================================================================
EXTRACTION SUMMARY
======================================================================
Categories processed: 2
Subcategories processed: 15
Subcategory2 processed: 157
Total products extracted: 23194
Errors encountered: 1

⚠ Errors encountered:
  1. Error extracting subcategories for 'HOME, BEAUTY & BABY': Message: no such element: Unable to locate element: {"method":"css selector","selector":"div[data-testid="iceberg-main-nav-l2-tabs"]"}
  (Session info: chrome=143.0.7499.41); For documentation on this error, please visit: https://www.selenium.dev/documentation/webdriver/troubleshooting/errors#nosuchelementexception
Stacktrace:
Symbols not available. Dumping unresolved backtrace:
        0x7ff6c30f8245
        0x7ff6c30f82a0
        0x7ff6c2ed165d
        0x7ff6c2f29a33
        0x7ff6c2f29d3c
        0x7ff6c2f1c88c
        0x7ff6c2f1c746
        0x7ff6c2f7ac97
        0x7ff6c2f1ac29
        0x7ff6c2f1ba93
        0x7ff6c340ffe0
        0x7ff6c340a920
        0x7ff6c3429086
        0x7ff6c3115744
        0x7ff6c311e6ec
        0x7ff6c3101964
        0x7ff6c3101b15
        0x7ff6c30e7842
        0x7ff81b6ae8d7
        0x7ff81d86c53c


======================================================================
PIPELINE COMPLETED SUCCESSFULLY
======================================================================

✓ Pipeline execution completed!

✓ Driver closed
```

Validation

In [ ]:

```python
# ======================================================================
# GROUND TRUTH VALIDATION - Structured by Category Hierarchy
# ======================================================================

import json
import random
from datetime import datetime
from collections import defaultdict

def create_ground_truth_by_hierarchy(extracted_file, output_file="ground_truth.json", products_per_subcategory2=10):
    """
    Create ground truth data with specified number of products per subcategory2.
    Products are grouped by category > subcategory > subcategory2.
    All products in each group have the same category, subcategory, and subcategory2.

    Args:
        extracted_file: Path to extracted products JSON file
        output_file: Path to output ground truth file
        products_per_subcategory2: Number of products to select per subcategory2 (default: 10)

    Returns:
        dict: Ground truth data structure
    """
    # Load extracted products
    with open(extracted_file, 'r', encoding='utf-8') as f:
        extracted_data = json.load(f)

    all_products = extracted_data.get('products', [])

    if not all_products:
        print("X No products found in extraction file")
        return None

    # Group products by category > subcategory > subcategory2
    grouped_products = defaultdict(lambda: defaultdict(lambda: defaultdict(list)))

    for product in all_products:
        category = product.get('category', '').strip()
        subcategory = product.get('subcategory', '').strip()
        subcategory2 = product.get('subcategory2', '').strip()

        if category and subcategory and subcategory2:
            grouped_products[category][subcategory][subcategory2].append(product)
```

```python
    # Create ground truth structure
    ground_truth = {
        "version": "1.0",
        "created_date": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "description": f"Ground truth data with {products_per_subcategory2} products per subcategory2",
        "products_per_subcategory2": products_per_subcategory2,
        "verification_status": "pending",
        "statistics": {
            "total_categories": 0,
            "total_subcategories": 0,
            "total_subcategory2": 0,
            "total_products": 0
        },
        "categories": []
    }

    # Process each category
    for category_name in sorted(grouped_products.keys()):
        category_data = {
            "category": category_name,
            "subcategories": []
        }

        # Process each subcategory
        for subcategory_name in sorted(grouped_products[category_name].keys()):
            subcategory_data = {
                "subcategory": subcategory_name,
                "subcategory2_items": []
            }

            # Process each subcategory2
            for subcategory2_name in sorted(grouped_products[category_name][subcategory_name].keys()):
                products_in_subcat2 = grouped_products[category_name][subcategory_name][subcategory2_name]

                # Select up to N products (or all if less than N)
                num_to_select = min(products_per_subcategory2, len(products_in_subcat2))
                selected_products = products_in_subcat2[:num_to_select]  # Take first N, or use random.sample() for random selection

                subcategory2_data = {
                    "subcategory2": subcategory2_name,
                    "expected_count": products_per_subcategory2,
                    "actual_count": num_to_select,
                    "products": []
                }

                # Add selected products
                for product in selected_products:
                    gt_product = {
                        "product-url": product.get("product-url", ""),
                        "product-title": product.get("product-title", ""),
                        "regular-price": product.get("regular-price", ""),
                        "product-package-size": product.get("product-package-size", ""),
                        "category": product.get("category", ""),
                        "subcategory": product.get("subcategory", ""),
                        "subcategory2": product.get("subcategory2", ""),
                        "verified": False,
                        "is_correct": None,
                        "notes": ""
                    }
                    subcategory2_data["products"].append(gt_product)

                subcategory_data["subcategory2_items"].append(subcategory2_data)
                ground_truth["statistics"]["total_subcategory2"] += 1
                ground_truth["statistics"]["total_products"] += len(subcategory2_data["products"])

            category_data["subcategories"].append(subcategory_data)
            ground_truth["statistics"]["total_subcategories"] += len(subcategory_data["subcategory2_items"])

        ground_truth["categories"].append(category_data)
        ground_truth["statistics"]["total_categories"] += 1

    # Save to file
    with open(output_file, 'w', encoding='utf-8') as f:
        json.dump(ground_truth, f, indent=2, ensure_ascii=False)

    # Print summary
    print("=" * 70)
    print("GROUND TRUTH CREATED")
    print("=" * 70)
    print(f"✓ Output file: {output_file}")
    print(f"✓ Products per subcategory2: {products_per_subcategory2}")
    print(f"\nStatistics:")
    print(f"  Total Categories: {ground_truth['statistics']['total_categories']}")
    print(f"  Total Subcategories: {ground_truth['statistics']['total_subcategories']}")
    print(f"  Total Subcategory2: {ground_truth['statistics']['total_subcategory2']}")
    print(f"  Total Products: {ground_truth['statistics']['total_products']}")

    print(f"\nBreakdown by Category:")
    for cat_data in ground_truth["categories"]:
        cat_name = cat_data["category"]
        total_subcat2 = sum(len(subcat["subcategory2_items"]) for subcat in cat_data["subcategories"])
        total_products = sum(
            len(subcat2["products"])
            for subcat in cat_data["subcategories"]
            for subcat2 in subcat["subcategory2_items"]
        )
        print(f"  {cat_name}: {total_subcat2} subcategory2 items, {total_products} products")

    print(f"\n⚠ Please manually verify each product:")
    print(f"  1. Open the product URL in a browser")
    print(f"  2. Verify each field (title, price, size, category hierarchy)")
    print(f"  3. Set 'verified': true and 'is_correct': true/false")
    print(f"  4. Add any notes in the 'notes' field")
    print("=" * 70)

    return ground_truth


def create_random_sample_for_verification(extracted_file, output_file="random_sample_verification.json", num_products=50):
    """
    Create a random sample of products from extraction for manual verification.

    Args:
        extracted_file: Path to extracted products JSON file
        output_file: Path to output sample file
        num_products: Number of products to randomly sample (default: 50)

    Returns:
        dict: Random sample data
    """
    # Load extracted products
    with open(extracted_file, 'r', encoding='utf-8') as f:
        extracted_data = json.load(f)

    products = extracted_data.get('products', [])

    if len(products) < num_products:
        print(f"⚠ Warning: Only {len(products)} products available, using all of them")
        num_products = len(products)

    # Randomly sample products
    random.seed()  # Use current time as seed
    selected_products = random.sample(products, num_products)

    # Create sample structure
    sample_data = {
        "version": "1.0",
        "created_date": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "description": f"Random sample of {num_products} products for manual verification",
        "total_products_in_extraction": len(products),
        "sample_size": num_products,
        "verification_status": "pending",
        "products": []
    }

    # Add products with indices for reference
    for idx, product in enumerate(selected_products, 1):
        sample_product = {
            "sample_index": idx,
            "product-url": product.get("product-url", ""),
            "product-title": product.get("product-title", ""),
            "regular-price": product.get("regular-price", ""),
            "product-package-size": product.get("product-package-size", ""),
            "category": product.get("category", ""),
            "subcategory": product.get("subcategory", ""),
            "subcategory2": product.get("subcategory2", ""),
            "verified": False,
            "is_correct": None,
```

```
            "is_correct": None,
            "notes": ""
        }
        sample_data["products"].append(sample_product)

    # Save to file
    with open(output_file, 'w', encoding='utf-8') as f:
        json.dump(sample_data, f, indent=2, ensure_ascii=False)

    print(f"✓ Random sample created: {output_file}")
    print(f"✓ Contains {len(sample_data['products'])} randomly selected products")
    print(f"\n⚠ Please manually verify each product and update the 'verified' and 'is_correct' fields")

    return sample_data


def compare_with_ground_truth(extracted_file, ground_truth_file):
    """
    Compare extracted products against ground truth.
    Works with the hierarchical ground truth structure.

    Args:
        extracted_file: Path to extracted products JSON file
        ground_truth_file: Path to ground truth JSON file

    Returns:
        dict: Comparison results
    """
    # Load files
    with open(extracted_file, 'r', encoding='utf-8') as f:
        extracted_data = json.load(f)

    with open(ground_truth_file, 'r', encoding='utf-8') as f:
        ground_truth_data = json.load(f)

    extracted_products = extracted_data.get('products', [])

    # Extract all ground truth products from hierarchical structure
    gt_products = []
    for category_data in ground_truth_data.get('categories', []):
        for subcategory_data in category_data.get('subcategories', []):
            for subcategory2_data in subcategory_data.get('subcategory2_items', []):
                for product in subcategory2_data.get('products', []):
                    gt_products.append(product)

    # Create a lookup dictionary by URL (most reliable identifier)
    extracted_by_url = {p.get('product-url', ''): p for p in extracted_products if p.get('product-url')}

    results = {
        "comparison_date": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "total_ground_truth": len(gt_products),
        "total_extracted": len(extracted_products),
        "matched_by_url": 0,
        "field_comparisons": {
            "product-title": {"exact_match": 0, "mismatch": 0, "missing_in_extracted": 0, "missing_in_ground_truth": 0},
            "regular-price": {"exact_match": 0, "mismatch": 0, "missing_in_extracted": 0, "missing_in_ground_truth": 0},
            "product-package-size": {"exact_match": 0, "mismatch": 0, "missing_in_extracted": 0, "missing_in_ground_truth": 0},
            "category": {"exact_match": 0, "mismatch": 0, "missing_in_extracted": 0, "missing_in_ground_truth": 0},
            "subcategory": {"exact_match": 0, "mismatch": 0, "missing_in_extracted": 0, "missing_in_ground_truth": 0},
            "subcategory2": {"exact_match": 0, "mismatch": 0, "missing_in_extracted": 0, "missing_in_ground_truth": 0}
        },
        "detailed_comparisons": [],
        "not_found_in_extraction": [],
        "by_category": defaultdict(lambda: {
            "total": 0,
            "matched": 0,
            "field_accuracy": defaultdict(lambda: {"exact_match": 0, "mismatch": 0, "total": 0})
        })
    }

    # Compare each ground truth product
    for gt_product in gt_products:
        gt_url = gt_product.get('product-url', '')
        gt_category = gt_product.get('category', '')

        results["by_category"][gt_category]["total"] += 1

        if gt_url in extracted_by_url:
            results["matched_by_url"] += 1
            results["by_category"][gt_category]["matched"] += 1
            extracted_product = extracted_by_url[gt_url]

            comparison = {
                "product-url": gt_url,
                "product-title": gt_product.get('product-title', ''),
                "category": gt_category,
                "fields": {}
            }

            # Compare each field
            for field in ["product-title", "regular-price", "product-package-size",
                          "category", "subcategory", "subcategory2"]:
                gt_value = gt_product.get(field, '').strip()
                ext_value = extracted_product.get(field, '').strip()

                if not gt_value and not ext_value:
                    status = "both_missing"
                elif not ext_value:
                    status = "missing_in_extracted"
                    results["field_comparisons"][field]["missing_in_extracted"] += 1
                elif not gt_value:
                    status = "missing_in_ground_truth"
                    results["field_comparisons"][field]["missing_in_ground_truth"] += 1
                elif gt_value.lower() == ext_value.lower():
                    status = "exact_match"
                    results["field_comparisons"][field]["exact_match"] += 1
                    results["by_category"][gt_category]["field_accuracy"][field]["exact_match"] += 1
                else:
                    status = "mismatch"
                    results["field_comparisons"][field]["mismatch"] += 1
                    results["by_category"][gt_category]["field_accuracy"][field]["mismatch"] += 1

                results["by_category"][gt_category]["field_accuracy"][field]["total"] += 1

                comparison["fields"][field] = {
                    "ground_truth": gt_value,
                    "extracted": ext_value,
                    "status": status
                }

            results["detailed_comparisons"].append(comparison)
        else:
            # Product not found in extraction
            results["not_found_in_extraction"].append({
                "product-url": gt_url,
                "product-title": gt_product.get('product-title', ''),
                "category": gt_category
            })

    # Calculate accuracy percentages
    total_gt = len(gt_products)
    if total_gt > 0:
        results["match_rate"] = (results["matched_by_url"] / total_gt) * 100

        results["field_accuracy"] = {}
        for field, stats in results["field_comparisons"].items():
            total = sum(stats.values())
            if total > 0:
                exact_pct = (stats["exact_match"] / total) * 100
                results["field_accuracy"][field] = {
                    "exact_match_percentage": f"{exact_pct:.2f}%",
                    "exact_matches": stats["exact_match"],
                    "mismatches": stats["mismatch"],
                    "missing_in_extracted": stats["missing_in_extracted"]
                }

        # Calculate per-category accuracy
        for category, cat_stats in results["by_category"].items():
            if cat_stats["total"] > 0:
                cat_stats["match_rate"] = (cat_stats["matched"] / cat_stats["total"]) * 100
                cat_stats["field_accuracy_percentages"] = {}
                for field, field_stats in cat_stats["field_accuracy"].items():
                    if field_stats["total"] > 0:
                        exact_pct = (field_stats["exact_match"] / field_stats["total"]) * 100
                        cat_stats["field_accuracy_percentages"][field] = f"{exact_pct:.2f}%"

    return results
```

```
def print_comparison_report(comparison_results):
    """Print a formatted comparison report."""
    print("=" * 70)
    print("GROUND TRUTH COMPARISON REPORT")
    print("=" * 70)

    print(f"\nOverall Statistics:")
    print(f"  Total Ground Truth Products: {comparison_results['total_ground_truth']}")
    print(f"  Total Extracted Products: {comparison_results['total_extracted']}")
    print(f"  Matched by URL: {comparison_results['matched_by_url']}")
    if comparison_results.get('match_rate'):
        print(f"  Match Rate: {comparison_results['match_rate']:.2f}%")

    print(f"\nField-Level Accuracy (Overall):")
    if comparison_results.get('field_accuracy'):
        for field, accuracy in comparison_results['field_accuracy'].items():
            print(f"  {field}:")
            print(f"    Exact Match: {accuracy['exact_match_percentage']} ({accuracy['exact_matches']} matches)")
            print(f"    Mismatches: {accuracy['mismatches']}")
            print(f"    Missing in Extraction: {accuracy['missing_in_extracted']}")

    # Per-category breakdown
    if comparison_results.get('by_category'):
        print(f"\nAccuracy by Category:")
        for category, cat_stats in comparison_results['by_category'].items():
            if cat_stats.get('total', 0) > 0:
                print(f"  {category}:")
                print(f"    Match Rate: {cat_stats.get('match_rate', 0):.2f}% ({cat_stats['matched']}/{cat_stats['total']})")
                if cat_stats.get('field_accuracy_percentages'):
                    print(f"    Field Accuracy:")
                    for field, pct in cat_stats['field_accuracy_percentages'].items():
                        print(f"      {field}: {pct}")

    if comparison_results['not_found_in_extraction']:
        print(f"\n⚠ Products in Ground Truth but NOT found in Extraction: {len(comparison_results['not_found_in_extraction'])}")
        print("  First 5 missing products:")
        for i, missing in enumerate(comparison_results['not_found_in_extraction'][:5], 1):
            print(f"    {i}. {missing.get('product-title', 'N/A')}")
            print(f"       Category: {missing.get('category', 'N/A')}")

    print("\n" + "=" * 70)


def save_comparison_report(comparison_results, output_file="comparison_report.json"):
    """Save comparison results to JSON file."""
    # Convert defaultdict to regular dict for JSON serialization
    def convert_defaultdict(obj):
        if isinstance(obj, defaultdict):
            return {k: convert_defaultdict(v) for k, v in obj.items()}
        return obj

    serializable_results = convert_defaultdict(comparison_results)

    with open(output_file, 'w', encoding='utf-8') as f:
        json.dump(serializable_results, f, indent=2, ensure_ascii=False)
    print(f"✓ Comparison report saved to: {output_file}")
```

Test validation

In [ ]:
```
# Create ground truth with 10 products per subcategory2
ground_truth = create_ground_truth_by_hierarchy(
    extracted_file="loblaws_products.json",
    output_file="ground_truth.json",
    products_per_subcategory2=10
)
```

In [ ]:
```
# Create random sample (50 products for random verification)
random_sample = create_random_sample_for_verification(
    extracted_file="loblaws_products.json",
    output_file="random_sample_verification.json",
    num_products=50
)
```

In [ ]:
```
# After manually verifying ground_truth.json, compare results
comparison_results = compare_with_ground_truth(
    extracted_file="loblaws_products.json",
    ground_truth_file="ground_truth.json"
)

# Print report
print_comparison_report(comparison_results)

# Save detailed report
save_comparison_report(comparison_results, "comparison_report.json")
```