Retail Promotions Analysis: Courts vs Harvey Norman

Introduction

This project focuses on web scraping product data from two of Singapore's most popular electronics and home appliance retailers: Courts and Harvey Norman. With the upcoming Hari Raya celebrations in 2025, both retailers have launched special promotional campaigns offering significant discounts across various product categories.

The primary objectives of this web scraping project are to:

1. Extract comprehensive product information from both Courts and Harvey Norman websites
2. Identify the top discounted products from each retailer's Hari Raya promotions
3. Compare identical or similar products across both retailers to find the most affordable options
4. Analyze pricing trends and discount patterns to help consumers make informed purchasing decisions

By systematically collecting and analyzing this data, the project aims to provide valuable insights for consumers looking to maximize savings during the Hari Raya promotional period, while also demonstrating effective web scraping techniques for retail price comparison.

Project Overview

The Harvey Norman Product Data Scraper is a Python-based web scraping tool that extracts detailed product information from Harvey Norman Singapore's website. It specifically targets product listings and extracts data such as:

- Product ID
- Product name
- Brand name
- Primary category
- Additional categories
- Price information

Requirements

- Python 3.7+
- Beautiful Soup 4
- Requests
- Pandas
- Regular expressions (re)

Methodology

This project employed a systematic approach to extract promotional product data from Courts and Harvey Norman websites. The methodology focused on efficiently gathering comprehensive product information while respecting website structures and ethical scraping practices.

Technology Stack

The web scraping implementation relied on the following technologies:

- **Python 3.7+**: Core programming language for the project
- **Beautiful Soup 4**: HTML parsing library to navigate and extract data from web page structures
- **Requests**: HTTP library to fetch web pages and handle sessions
- **Pandas**: Data manipulation library for organizing and analyzing the extracted data
- **Regular expressions (re)**: Pattern matching for extracting specific data formats and cleaning text

Scraping Approach

1. Initial Research and Website Analysis

Before writing any code, I conducted a thorough analysis of both websites:

- Manually navigated through the Hari Raya promotional pages on both Courts and Harvey Norman
- Identified common product information patterns and data structures
- Examined the HTML structure and JavaScript data layers using browser developer tools
- Checked for any rate limiting, anti-scraping measures, or robots.txt restrictions
- Documented the URL patterns for promotional pages and pagination systems

2. Courts Website Scraping Strategy

For the Courts website:

1. **Entry Point Identification**: Located the main Hari Raya promotion landing page and mapped the category structure
2. **Pagination Handling**: Implemented logic to navigate through multiple pages of product listings
3. **Data Extraction Method**:
    - Primary approach: Located embedded JavaScript objects (dataLayer or similar) containing structured product data
    - Secondary approach: Direct HTML parsing of product grid elements when JavaScript data wasn't available
4. **Session Management**: Maintained consistent session cookies to simulate normal browsing behavior

3. Harvey Norman Website Scraping Strategy

For the Harvey Norman website:

1. **Promotion Page Navigation**: Located and mapped the Hari Raya promotional sections
2. **JavaScript Data Extraction**:
    - Identified script tags containing product arrays with item details
    - Used regular expressions to extract and convert JavaScript objects to valid JSON
    - Parsed structured data to extract comprehensive product information
3. **HTML Parsing Fallback**: Implemented secondary extraction logic using BeautifulSoup to parse product elements directly from the DOM
4. **Category and Brand Extraction**: Special focus on extracting hierarchical category information and normalizing brand names

4. Data Extraction Workflow

The general extraction workflow for both websites followed these steps:

1. Send HTTP requests with appropriate headers (User-Agent, Accept, etc.) to avoid blocking
2. Handle response status codes and implement retry logic for failed requests
3. Parse the HTML content using BeautifulSoup
4. Extract product data using dual approaches:

    - Search for JavaScript data structures containing structured product information
    - Parse HTML elements containing product details as fallback
5. Extract key product attributes:
    - Product ID and SKU
    - Product name
    - Brand information
    - Price (original and discounted)
    - Discount percentage/amount
    - Product categories
    - Product specifications
6. Store extracted data in structured Python dictionaries
7. Implement appropriate delays between requests to respect server resources

5. Parallel Processing

To improve efficiency while maintaining ethical scraping practices:

- Implemented throttled parallel processing for handling multiple pages
- Used thread pools with controlled concurrency (max 3-5 concurrent requests)
- Maintained sufficient delays between requests to the same domain
- Implemented exponential backoff for retry attempts

Data Extraction

The extraction process focused on gathering comprehensive product information from both Courts and Harvey Norman websites during their Hari Raya 2025 promotional campaigns.

Courts Data Extraction

For the Courts website, I implemented a structured approach to navigate through their promotional pages:

1. **Initial Page Access**: Started with the main Hari Raya promotional landing page to access all featured products.

2. **Navigation Through Categories**: Systematically accessed each product category section to ensure comprehensive coverage of all promotional items.

3. **Product Information Extraction**: For each product listing, I extracted:

    - Full product name (which typically contained brand and model information)
    - Current promotional price
    - Original price (when available)
    - Product URL for reference

4. **Pagination Handling**: Implemented logic to navigate through multiple pages of results within each category section to capture all available products.

Harvey Norman Data Extraction

The Harvey Norman website required a different approach due to its unique structure:

JavaScript Data Targeting: Identified that product information was primarily stored in JavaScript variables within the page source. For example:

var items = [{ 'item_id': '85044', 'item_name': 'Samsung 530L 2 Door Fridge - Refined Inox (RT53DG7A6CS9SS)', 'item_brand': 'S', "item_category" : 'Fridges', ...}];

1.
2. **Regular Expression Extraction**: Used regex patterns to locate and extract these JavaScript data structures that contained rich product information.

3. **Data Conversion**: Transformed the extracted JavaScript objects into structured Python dictionaries for further processing.

4. **HTML Fallback Method**: In cases where JavaScript data wasn't available, implemented a secondary extraction method that directly parsed the HTML elements containing product information.

5. **Multi-Page Extraction**: Systematically worked through all pages of the Harvey Norman Hari Raya promotional section to ensure complete data collection.

Handling Website Variations

Both websites presented unique challenges that required adaptive extraction techniques:

- **Dynamic Content**: Some promotional sections loaded products dynamically with JavaScript, requiring session management and request headers that mimicked browser behavior.

- **Inconsistent Data Structures**: Product information wasn't always presented in a consistent format, requiring flexible parsing logic.

- **Rate Limiting Consideration**: Implemented appropriate delays between requests to respect server resources and avoid IP blocking.

Data Cleaning

After extraction, the raw data required several cleaning steps to make it suitable for analysis and comparison:

1. Brand Extraction

The combined product information often included brand names embedded within the full product name:

- **Brand Identification**: Extracted brand names from the beginning of product names where they typically appeared (e.g., "Samsung 530L 2 Door Fridge" → brand: "Samsung").

- **Brand Normalization**: Standardized brand representations across both retailers. For example, "S" was mapped to "Samsung" to ensure consistent comparison.

2. Price Cleaning

Price data required significant processing:

- **Currency Symbol Removal**: Removed the "$" sign from all price values to convert them to numeric format.

- **String to Numeric Conversion**: Transformed price strings into float values for mathematical operations.

- **Missing Price Handling**: Implemented logic to handle cases where original prices were not explicitly listed.

3. Product Type & Category Classification

Product categorization was derived from analyzing the full product names:

- **Keyword Extraction**: Identified key product type indicators in names (e.g., "Fridge", "TV", "Washing Machine").

- **Hierarchical Categorization**: Created a category hierarchy by grouping products into major categories (e.g., "Kitchen Appliances", "Home Entertainment").

- **Category Standardization**: Normalized category names across retailers to enable direct comparison of similar product types.

4. Discount Calculation

Created additional data points to facilitate deal comparison:

Percentage Discount: Calculated discount percentage using the formula:

```
discount_percentage = ((original_price - current_price) / original_price) * 100
```

-

Absolute Discount Amount: Calculated the actual dollar savings:

```
discount_amount = original_price - current_price
```

-

5. Data Structuring

The final cleaned data was structured into a comprehensive CSV file with the following columns:

- `retailer`: Source retailer (Courts or Harvey Norman)
- `product_name`: Full product name as displayed on the website
- `brand`: Extracted and normalized brand name
- `product_type`: Primary product type (e.g., TV, Refrigerator)
- `category`: Broader product category (e.g., Electronics, Home Appliances)
- `original_price`: Original price before discount
- `current_price`: Current promotional price
- `discount_amount`: Dollar amount saved
- `discount_percentage`: Percentage discount offered
- `url`: Direct link to the product page

This structured dataset provided the foundation for subsequent analysis to identify the best deals across both retailers during the Hari Raya promotional period.

==========

Harvey Norman Product Data Scraper

Introduction

This project extracts product information from Harvey Norman's website, focusing on product details including categories, brand names, and pricing. The scraper is designed to help analyze product offerings, track promotions, and gather structured data for further analysis.

Project Overview

The Harvey Norman Product Data Scraper is a Python-based web scraping tool that extracts detailed product information from Harvey Norman Singapore's website. It specifically targets product listings and extracts data such as:

- Product ID
- Product name
- Brand name
- Primary category
- Additional categories
- Price information

Requirements

- Python 3.7+
- Beautiful Soup 4
- Requests
- Pandas
- Regular expressions (re)

Installation

1. Clone this repository:

```
git clone https://github.com/yourusername/harvey-norman-scraper.git
cd harvey-norman-scraper
```

2. Create a virtual environment (optional but recommended):

```
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3. Install the required packages:

```
pip install -r requirements.txt
```

Usage

Basic Usage

```
from harvey_norman_scraper import extract_product_data
```

```
url = "https://www.harveynorman.com.sg/promotions-en/harvey-raya-sale-2025-en/page-64/"
products = extract_product_data(url)

# Save to CSV
import pandas as pd
df = pd.DataFrame(products)
df.to_csv("harvey_norman_products.csv", index=False)
```

Command Line Interface

```
python scraper.py --url "https://www.harveynorman.com.sg/promotions-en/harvey-raya-sale-2025-en/page-64/"
--output products.csv
```

Methodology

The scraper employs two main strategies to extract product data:

1. JavaScript Data Extraction

The primary method targets JavaScript data structures embedded in the webpage. It:

1. Searches for script tags containing product data arrays
2. Uses regular expressions to extract structured data
3. Converts JavaScript objects to valid JSON
4. Parses the JSON to extract product information

2. HTML Parsing Fallback

If JavaScript extraction fails, the scraper falls back to direct HTML parsing:

1. Identifies product elements using CSS selectors
2. Extracts product details from HTML structure
3. Uses breadcrumbs to identify categories
4. Cleans and formats extracted text

Technical Details

Data Extraction Process

1. **Request Handling**: The script sends HTTP requests with appropriate headers to avoid blocking.
2. **JavaScript Parsing**: Uses regex to extract and clean JavaScript data structures.
3. **HTML Parsing**: Uses BeautifulSoup to parse HTML elements and extract product information.
4. **Data Cleaning**: Handles undefined variables and cleans extracted data.
5. **Data Structuring**: Organizes extracted data into consistent dictionaries.

Handling Challenges

- **JavaScript Variables**: The script handles undefined JavaScript variables by replacing them with null values.
- **Rate Limiting**: Implements delays between requests to avoid overloading the server.
- **Error Handling**: Includes robust error handling for network issues and parsing errors.

Ethical Considerations

This scraper is designed to:

1. Respect the website's `robots.txt` directives
2. Implement rate limiting to avoid overwhelming the server
3. Only extract publicly available information
4. Be used for personal research or educational purposes only

Please ensure you have the right to scrape the target website and comply with their terms of service.

Future Improvements

- Implement proxy rotation for larger scraping tasks
- Add support for pagination to scrape multiple pages
- Implement data validation and cleaning
- Add support for extracting customer reviews
- Create a dashboard for visualizing the extracted data

License

Acknowledgments

- Beautiful Soup for HTML parsing
- Requests for HTTP requests
- Pandas for data manipulation

Data Extraction & Cleaning

The data extraction and cleaning process was critical to ensure accurate comparison between products from Courts and Harvey Norman. This section details the specific techniques used to extract structured data and prepare it for analysis.

Data Extraction Techniques

JavaScript Object Extraction

Both retailers embed rich product data in JavaScript objects within their page source:

```
def extract_js_data(html_content):
    """Extract product data from JavaScript objects in page source"""
    # Look for dataLayer or similar JS objects
    script_pattern = re.compile(r'var\s+items\s*=\s*(\[.*?\]);', re.DOTALL)
    matches = script_pattern.findall(html_content)

    products = []
    for match in matches:
        try:
            # Clean JavaScript object to make valid JSON
            cleaned_json = re.sub(r'([{,])\s*(\w+):', r'\1"\2":', match)
```

```
            cleaned_json = re.sub(r"""", "", cleaned_json)
            cleaned_json = re.sub(r',\s*}', '}', cleaned_json)

            # Handle undefined JS variables
            cleaned_json = re.sub(r':\s*\w+(?!\s*["',}])', r': null', cleaned_json)

            # Parse JSON and extract product data
            items_data = json.loads(cleaned_json)
            for item in items_data:
                # Extract and structure product information
                product = {
                    'product_id': item.get('item_id'),
                    'product_name': item.get('item_name'),
                    'brand': item.get('item_brand'),
                    'original_price': item.get('original_price', item.get('price')),
                    'current_price': item.get('price'),
                    'discount_pct': calculate_discount_percentage(
                        item.get('original_price', item.get('price')),
                        item.get('price')
                    ),
                    'category': item.get('item_category'),
                    'retailer': 'Harvey Norman' if 'harveynorman' in url else 'Courts',
                    'url': url
                }
                products.append(product)
        except json.JSONDecodeError:
            continue

    return products
```

HTML Structure Parsing

For cases where JavaScript data wasn't available:

```
def extract_product_from_html(soup, url):
    """Extract product data directly from HTML elements"""
    products = []
    product_elements = soup.select('.product-item, .product-card, .product-box')

    for element in product_elements:
        try:
            # Extract product name
            name_elem = element.select_one('.product-name, .product-title, h2, h3')
            product_name = name_elem.text.strip() if name_elem else None

            # Extract brand (often first word in product name)
            brand = extract_brand_from_name(product_name)

            # Extract prices
            price_current = extract_current_price(element)
            price_original = extract_original_price(element)

            # Calculate discount
```

```
            discount_pct = calculate_discount_percentage(price_original, price_current)

            # Create product dictionary
            product = {
                'product_name': product_name,
                'brand': brand,
                'original_price': price_original,
                'current_price': price_current,
                'discount_pct': discount_pct,
                'category': extract_category(element, soup),
                'retailer': 'Harvey Norman' if 'harveynorman' in url else 'Courts',
                'url': extract_product_url(element, url)
            }
            products.append(product)
        except Exception as e:
            # Log error and continue
            print(f"Error extracting product: {e}")
            continue

    return products
```

Data Cleaning Process

1. Brand Name Normalization

Brand names were inconsistently formatted across both retailers:

```
def normalize_brand(brand):
    """Normalize brand names for consistent comparison"""
    if not brand:
        return None

    # Convert to lowercase for comparison
    brand = brand.lower()

    # Handle abbreviated brands
    brand_mapping = {
        's': 'samsung',
        'lg': 'lg electronics',
        'hp': 'hewlett-packard',
        # Add more brand mappings as discovered
    }

    # Apply mapping if brand is in the dictionary
    return brand_mapping.get(brand, brand)
```

2. Price Extraction and Standardization

Prices needed consistent extraction and formatting:

```
def extract_current_price(element):
    """Extract current price from product element"""
```

```
    price_elem = element.select_one('.price, .product-price, .current-price')
    if not price_elem:
        return None

    # Extract numeric value from price text
    price_text = price_elem.text.strip()
    price_match = re.search(r'[\$S]?\s*(\d+[.,]\d+|\d+)', price_text)

    if price_match:
        # Convert to float and remove commas
        price = price_match.group(1).replace(',', '')
        return float(price)
    return None
```

3. Discount Calculation

Standardized discount calculations across retailers:

```
def calculate_discount_percentage(original_price, current_price):
    """Calculate discount percentage from original and current prices"""
    if not original_price or not current_price:
        return None

    if original_price <= current_price:
        return 0.0

    discount = ((original_price - current_price) / original_price) * 100
    return round(discount, 2)
```

4. Product Matching

To enable comparison across retailers, similar products needed identification:

```
def generate_product_key(product):
    """Generate a comparison key for matching similar products"""
    if not product.get('product_name'):
        return None

    # Remove special characters and numbers from product name
    name = re.sub(r'[^\w\s]', '', product['product_name'].lower())

    # Extract key product attributes from name (e.g., TV size, refrigerator capacity)
    # This is specific to product categories
    key_attributes = extract_key_attributes(name, product.get('category'))

    # Combine brand and key attributes for matching
    brand = product.get('brand', '').lower()
    if brand and key_attributes:
        return f"{brand}_{key_attributes}"
    return None
```

5. Data Deduplication

Removing duplicate products within the same retailer:

```
def remove_duplicates(products_df):
    """Remove duplicate products based on product ID or name+price combination"""
    # If product_id exists, use it for deduplication
    if 'product_id' in products_df.columns and products_df['product_id'].notna().all():
        return products_df.drop_duplicates(subset=['retailer', 'product_id'])

    # Otherwise use name and price combination
    return products_df.drop_duplicates(subset=['retailer', 'product_name', 'current_price'])
```

6. Final Data Structure

The cleaned data was structured into a pandas DataFrame with the following columns:

- `retailer`: 'Courts' or 'Harvey Norman'
- `product_id`: Unique product identifier (when available)
- `product_name`: Full product name
- `brand`: Normalized brand name
- `category`: Primary product category
- `original_price`: Original price before discount
- `current_price`: Current promotional price
- `discount_pct`: Calculated discount percentage
- `discount_amount`: Absolute discount amount (original_price - current_price)
- `url`: Product page URL
- `comparison_key`: Generated key for matching similar products across retailers