Untitled-2

August 25, 2025

[1]:
```
# --- Dependencies ---
!pip install --upgrade jinja2 pandas requests seaborn matplotlib scipy
```

```
Requirement already satisfied: jinja2 in ./.conda/lib/python3.11/site-packages
(3.1.6)
Requirement already satisfied: pandas in ./.conda/lib/python3.11/site-packages
(2.3.1)
Collecting pandas
  Downloading
pandas-2.3.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(91 kB)
Requirement already satisfied: requests in ./.conda/lib/python3.11/site-packages
(2.32.4)
Collecting requests
  Downloading requests-2.32.5-py3-none-any.whl.metadata (4.9 kB)
Requirement already satisfied: seaborn in ./.conda/lib/python3.11/site-packages
(0.13.2)
Requirement already satisfied: matplotlib in ./.conda/lib/python3.11/site-
packages (3.10.5)
Requirement already satisfied: scipy in ./.conda/lib/python3.11/site-packages
(1.16.1)
Requirement already satisfied: MarkupSafe>=2.0 in ./.conda/lib/python3.11/site-
packages (from jinja2) (3.0.2)
Requirement already satisfied: numpy>=1.23.2 in ./.conda/lib/python3.11/site-
packages (from pandas) (2.3.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
./.conda/lib/python3.11/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in ./.conda/lib/python3.11/site-
packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in ./.conda/lib/python3.11/site-
packages (from pandas) (2025.2)
Requirement already satisfied: charset_normalizer<4,>=2 in
./.conda/lib/python3.11/site-packages (from requests) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in ./.conda/lib/python3.11/site-
packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
./.conda/lib/python3.11/site-packages (from requests) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
```

```
./.conda/lib/python3.11/site-packages (from requests) (2025.8.3)
Requirement already satisfied: contourpy>=1.0.1 in ./.conda/lib/python3.11/site-
packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in ./.conda/lib/python3.11/site-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
./.conda/lib/python3.11/site-packages (from matplotlib) (4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
./.conda/lib/python3.11/site-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in ./.conda/lib/python3.11/site-
packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in ./.conda/lib/python3.11/site-
packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in ./.conda/lib/python3.11/site-
packages (from matplotlib) (3.2.3)
Requirement already satisfied: six>=1.5 in ./.conda/lib/python3.11/site-packages
(from python-dateutil>=2.8.2->pandas) (1.17.0)
Downloading
pandas-2.3.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.4
MB)
                           12.4/12.4 MB
4.8 MB/s  0:00:02m0:00:0100:01
Downloading requests-2.32.5-py3-none-any.whl (64 kB)
Installing collected packages: requests, pandas
  Attempting uninstall: requests
    Found existing installation: requests 2.32.4
    Uninstalling requests-2.32.4:
      Successfully uninstalled requests-2.32.4
  Attempting uninstall: pandas
    Found existing installation: pandas 2.3.1
    Uninstalling pandas-2.3.1:
      Successfully uninstalled pandas-2.3.1
                           2/2
[pandas]2m1/2 [pandas]
Successfully installed pandas-2.3.2 requests-2.32.5
```

[2]:
```python
# --- Setup ---
import requests
import pandas as pd
pd.set_option("display.max_columns", None)
pd.set_option("display.max_colwidth", None)
pd.set_option("display.max_rows", None)
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display, HTML
from datetime import datetime
```

```
import io
import base64
import math

# --- Constants ---
BASE_V2 = "https://api-g.weedmaps.com/discovery/v2"
BASE_V1 = "https://api-g.weedmaps.com/discovery/v1"
LATLNG = "39.642867,-104.826711"  # Aurora, CO
HEADERS = {
        "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:141.0) Gecko/20100101␣
 ↪Firefox/141.0",
        "Accept": "application/json, */*",
        "Accept-Language": "en-US,en;q=0.5",
        "Upgrade-Insecure-Requests": "1",
        "Sec-Fetch-Dest": "document",
        "Sec-Fetch-Mode": "navigate",
        "Sec-Fetch-Site": "none",
        "Sec-Fetch-User": "?1",
        "wm-user-latlng": LATLNG,
        "If-None-Match": "W/\"2d61d944c89769b44d46f9622ac2427b\"",
        "Priority": "u=0, i"
}
HEADERSV1 = {
    "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:141.0) Gecko/20100101␣
 ↪Firefox/141.0",
    "Accept": "application/json, */*",
    "Accept-Language": "en-US,en;q=0.5",
    "Authorization": "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzUxMiJ9.
 ↪eyJqdGkiOiJLUkNhMjRUeSIsImV4cCI6MTc1NDc3NjUxNywiaXNzIjoid2VlZG1hcHMuY29tIiwiaGFzdXJhIjp7ImF
 ↪MFLvc3cVp3blhNXV9RNOrram5yZtoXTxqagwl7oWlxTOywE5waTRSqOCWjiKj4bQIhn6MFt-x_JU7qQtRS7cXzBOIA-
    "wm-user-latlng": LATLNG,
    "Referer": "https://weedmaps.com/",
}

print("  Setup complete.")
```

```
 Setup complete.
```

[3]:
```
# --- Find & Select Dispensary ---
print("Searching for nearby medical dispensaries...")
lat, lng = map(float, LATLNG.split(','))
RADIUS_MI = 20
lat_deg = RADIUS_MI / 69.0
lng_deg = RADIUS_MI / (69.0 * math.cos(math.radians(lat)))
bounding_box = f"{lat - lat_deg},{lng - lng_deg},{lat + lat_deg},{lng +␣
 ↪lng_deg}"
```

```
params = {
    "latlng": LATLNG, "filter[any_retailer_services][]": "storefront",
    "filter[amenities][]": "is_medical", "filter[bounding_box]": bounding_box,
    "sort_by": "position_distance", "sort_order": "asc", "page_size": 100,
}
response = requests.get(f"{BASE_V2}/listings", headers=HEADERS, params=params)
response.raise_for_status()
listings = response.json().get("data", {}).get("listings", [])
dispensary_list_df = pd.json_normalize(listings, sep=".")
print(f"Found {len(dispensary_list_df)} total medical storefronts.")

# --- Select a Dispensary ---
# Set the target dispensary slug here, or pick a random one from the 5 closest␣
 ↪dispensaries.
# To randomly select from the 5 closest, use the following line:
#DISPENSARY_SLUG = dispensary_list_df.head(5)["slug"].sample(1).values[0]
# Or set to a specific slug, e.g.:
# DISPENSARY_SLUG = "little-brown-house"  # <-- Change this to your target
#DISPENSARY_SLUG = "magic-city-cannabis-colorado"
DISPENSARY_SLUG = "reefer-madness"
if DISPENSARY_SLUG in dispensary_list_df["slug"].values:
    dispensary_info = dispensary_list_df[dispensary_list_df['slug'] ==␣
 ↪DISPENSARY_SLUG].iloc[0]
    print(f"\n Selected Dispensary: {dispensary_info.get('name',␣
 ↪DISPENSARY_SLUG)}")
else:
    # Create a dummy object if not found, so the report can still run
    dispensary_info = pd.Series({'name': DISPENSARY_SLUG.replace('-', ' ').
 ↪title()})
    print(f"\n  Slug '{DISPENSARY_SLUG}' not found in list. Using slug as name.
 ↪")
```

```
Searching for nearby medical dispensaries…
Found 64 total medical storefronts.

 Selected Dispensary: Reefer Madness
```

[4]:
```
# --- Full Flower dataset, paginated & flattened ---
page, page_size = 1, 50
flower_pool = []

while True:
    params = {
        "filter[license_type]": "medical",
        "filter[any_client_categories][]": "flower-category-pages",
        "sort_by": "min_price",
        "sort_order": "asc",
```

```
        "page": page,
        "page_size": page_size,
        "include[]": "facets.categories",
    }
    url = f"{BASE_V1}/listings/dispensaries/{DISPENSARY_SLUG}/menu_items"
    resp = requests.get(url, headers=HEADERS, params=params)
    resp.raise_for_status()
    page_items = resp.json()["data"]["menu_items"]

    if not page_items:
        break

    flower_pool.extend(page_items)
    print(f"Fetched page {page}: {len(page_items)} items")
    if len(page_items) < page_size:
        break
    page += 1

# flatten every nested level using dot-notation keys
flower_df = pd.json_normalize(flower_pool, sep='.')
print(f"\nTOTAL flower items fetched: {len(flower_df)}")
flower_df
for col, val in flower_df.iloc[0].items():
    print(f"{col}: {val}")
```

```
Fetched page 1: 50 items
Fetched page 2: 14 items

TOTAL flower items fetched: 64
brand_endorsement: None
catalog_slug: med-generations-garden-bubba-gum-orange-tier-buds-777479435
created_at: 2025-03-20T18:32:38.342Z
current_deal_title: None
deal_ids: []
genetics_tag: None
id: 186877185
is_badged: False
is_endorsed: False
is_online_orderable: True
lab_website: None
last_ordered_date: None
license_type: medical
menu_id: 777479435
name: MED - Generations Garden - Bubba Gum / Orange-Tier Buds
ordered_from: False
pixel_url: None
position: None
```

price_visibility: visible
price_visibility_description: None
price_visibility_kickout_modal: None
price_visibility_title: None
rating: 0.0
reviews_count: 0
slug: med-generations-garden-bubba-gum-orange-tier-buds
tags: None
test_result_created_at: None
updated_at: 2025-08-12T23:02:53.077Z
test_result_expired: None
test_result_expires_in: None
avatar_image.large_url: https://images.weedmaps.com/pictures/listings/161/909/07
4/425902630_180730_StrainReview_EnjoyableXJ13_12.jpg?txt64=UHJvZHVjdCBleGFtcGxl&
txt-fit=max&txt-color=666&txt-lead=0&txt-size=24&txt-
font=Avenir+Next+Medium&txt-align=center,bottom
avatar_image.original_url: https://images.weedmaps.com/pictures/listings/161/909
/074/425902630_180730_StrainReview_EnjoyableXJ13_12.jpg?txt64=UHJvZHVjdCBleGFtcG
xl&txt-fit=max&txt-color=666&txt-lead=0&txt-size=24&txt-
font=Avenir+Next+Medium&txt-align=center,bottom
category.id: 1
category.name: Indica
category.slug: indica
edge_category.uuid: a780af3d-bdfe-41ce-a782-20f2519fd7be
edge_category.name: Flower
edge_category.slug: flower
edge_category.ancestors: []
external_ids.unit: nan
external_ids.half_ounce: None
external_ids.gram:
2c3fcb9997efe5f8f5d5d84468b4e5b41e92fe935e06768d4e3613b4374a7ec1|3466429
external_ids.two_grams: nan
external_ids.eighth: None
external_ids.ounce: None
external_ids.half_gram: nan
external_ids.quarter: None
lab_avatar_image.small_url:
https://images.weedmaps.com/static/placeholders/weedmaps-logo.jpg
lab_avatar_image.original_url:
https://images.weedmaps.com/static/placeholders/weedmaps-logo.jpg
metrics.cannabinoids: []
metrics.terpenes: []
metrics.aggregates.thc: 0.0
metrics.aggregates.thc_unit: %
metrics.aggregates.cbd: 0.0
metrics.aggregates.cbd_unit: %
metrics.aggregates.cbn: 0.0
metrics.aggregates.cbn_unit: %

```
metrics.aggregates.cbg: 0.0
metrics.aggregates.cbg_unit: %
metrics.aggregates.terpenes: 0
metrics.aggregates.terpenes_unit: %
price.id: 137467338
price.unit: gram
price.quantity: 1
price.label: 1 g
price.compliance_net_mg: 1000.0
price.price: 1.0
price.on_sale: False
price.original_price: 1.0
price.discount_label: None
price_stats.min: None
price_stats.max: None
prices.grams_per_eighth: 3.5
prices.gram: [{'id': 137467338, 'label': '1 g', 'compliance_net_mg': 1000.0,
'price': 1.0, 'on_sale': False, 'original_price': 1.0, 'units': '1',
'gram_unit_price': 1.0, 'weight': {'value': 1.0, 'unit': 'g'}}]
menu.features: ['static']
menu.listing_menu_types: []
menu.id: 777479435
prices.ounce: nan
external_ids: nan
```

[5]:
```python
# --- Process Data & Create Final DataFrame (Corrected) ---
import pandas as pd
import numpy as np
import re

print("="*60)
print("  PROCESSING RAW DATA INTO A FLAT PRICE TABLE...")
print("="*60)

OZ_TO_G = 28.35
LEGAL_LIMIT_G = 2 * OZ_TO_G

def format_grams(g):
    """Rounds gram weights to their common market values for display."""
    common_weights = [1, 3.5, 7, 14, 28, 57]
    for w in common_weights:
        if abs(g - w) < 0.4:
            return f"{w:g}g"
    return f"{round(g, 1):g}g"

final_rows = []
for item in flower_pool:
```

```
    prices = item.get("prices", {}) or {}
    all_deals_raw = (prices.get("gram") or []) + (prices.get("ounce") or [])
    if not all_deals_raw:
        continue

    #  Categorization: include "Red Tier" variants (e.g., "Red-Tier", "Red␣
↪-Tier") as Shake/Popcorn/Trim
    name = item.get('name', '') or ''
    SHAKE_PATTERN = re.compile(r'\b(shake|trim|popcorn|littles|red\s*[-]?
↪\s*tier)\b', flags=re.IGNORECASE)

    if SHAKE_PATTERN.search(name):
        report_category = 'Shake/Popcorn/Trim'
    elif len(all_deals_raw) <= 2:
        report_category = 'Pre-Pack Specialty'
    else:
        report_category = 'Bulk Value'

    for p in all_deals_raw:
        try:
            gram_unit_price = float(p.get('gram_unit_price'))
            weight_val = float((p.get('weight', {}) or {}).get('value'))
            weight_unit = ((p.get('weight', {}) or {}).get('unit') or '').
↪lower()
            price = float(p.get('price'))
            label = p.get('label')

            # Normalize to grams (assume grams unless explicitly ounce-based)
            weight_g = weight_val * OZ_TO_G if weight_unit.startswith('oz')␣
↪else weight_val

            # Basic validity checks (also enforce a 2 oz legal cap)
            if not (weight_g > 0 and price > 0 and label and weight_g <=␣
↪LEGAL_LIMIT_G):
                continue

            price_per_oz = gram_unit_price * OZ_TO_G
            size_label_g = format_grams(weight_g)

            final_rows.append({
                'name': name,
                'slug': item.get('slug'),
                'report_category': report_category,
                'size_label': size_label_g,
                'price': price,
                'price_per_oz': price_per_oz,
                'weight_g': weight_g
```

```
            })
        except (ValueError, TypeError, AttributeError):
            continue

# --- Create the final DataFrame ---
columns = ['name', 'slug', 'report_category', 'size_label', 'price',␣
 ↪'weight_g', 'price_per_oz']
price_df = pd.DataFrame(final_rows)

if not price_df.empty:
    price_df = price_df[columns]
    price_df.drop_duplicates(inplace=True)
    price_df = price_df.sort_values('price_per_oz').reset_index(drop=True)

print(f" Analysis complete. Created a flat price table with {len(price_df)}␣
 ↪purchasable items.")
display(price_df.head())
```

```
============================================================
 PROCESSING RAW DATA INTO A FLAT PRICE TABLE…
============================================================
 Analysis complete. Created a flat price table with 271 purchasable items.
```

	name	slug	report_category	size_label	price	weight_g	price_per_oz
0	MED - Generations Garden - Bubba Gum / Orange-Tier Buds	med-generations-garden-bubba-gum-orange-tier-buds	Pre-Pack Specialty	1g	1.00	1.00000	28.3500
1	MED - Legacy Grown - Four Kings / "Seeded" Tier Buds	med-legacy-grown-four-kings-orange-tier-buds	Pre-Pack Specialty	1g	1.00	1.00000	28.3500
2	MED - Long Gone Farms - Mystery Machine / Red-Tier Popcorn (Copy)	med-long-gone-farms-mystery-machine-red-tier-popcorn-copy	Shake/Popcorn/Trim	1g	1.00	1.00000	28.3500
3	MED - Vera - Fritter Runtz / Red tier Popcorn (Copy)	med-vera-fritter-runtz-red-tier-popcorn-copy	Shake/Popcorn/Trim	1g	1.00	1.00000	28.3500
4	MED - Boulder Built - Brass Billy / Red-Tier Popcorn	med-boulder-built-brass-billy-red-tier-popcorn	Shake/Popcorn/Trim	3.5g	5.01	3.54375	40.5405

[6]:
```
# Price Bands (per product), aesthetic like original, no collapse
import pandas as pd
from IPython.display import HTML, display
```

```
if 'price_df' not in globals() or price_df is None or price_df.empty:
    display(HTML("""
    <div class="p-4 mb-4 text-sm text-yellow-300 bg-yellow-900/50 rounded-lg␣
 ↪border border-yellow-700" role="alert">
      <span class="font-bold">No Data:</span> Nothing to render for this␣
 ↪dispensary.
    </div>"""))
else:
    # Canonical: cheapest $/oz, break tie by larger size, drop Shake/Popcorn/
 ↪Trim
    canonical = (price_df.sort_values(['slug','price_per_oz','weight_g'],␣
 ↪ascending=[True,True,False])
                         .groupby('slug', as_index=False).head(1))
    base = canonical[canonical['report_category']!='Shake/Popcorn/Trim'].copy()

    labels = ["  $60", "$61-$90", "$91-$120", "$121-$200", ">$200"]
    bins   = [0, 60, 90, 120, 200, float('inf')]
    bands_df =␣
 ↪(base[['name','report_category','size_label','price','price_per_oz']]
              .rename(columns={'name':'Product','report_category':
 ↪'Category','size_label':'Best Size','price':'Best Price','price_per_oz':
 ↪'Best $/Oz (28g)'}))
    bands_df['Price Band'] = pd.cut(bands_df['Best $/Oz (28g)'], bins=bins,␣
 ↪labels=labels, right=True, include_lowest=True)
    bands_df['Price Band'] = pd.Categorical(bands_df['Price Band'],␣
 ↪categories=labels, ordered=True)
    bands_df = bands_df.sort_values(['Price Band','Best $/Oz (28g)','Product']).
 ↪reset_index(drop=True)

    counts = bands_df['Price Band'].value_counts().reindex(labels, fill_value=0)
    total  = max(len(bands_df), 1)
    shares = (counts/total*100).round(0).astype(int)
    chips  = "".join(
        f"""<div class="bg-gray-800 border border-gray-700 rounded-lg p-3">
              <div class="text-sm text-gray-400">{lbl} (28g)</div>
              <div class="mt-1 text-lg font-semibold␣
 ↪text-white">{int(counts[lbl])}
                <span class="text-xs text-gray-400">({shares[lbl]}%)</span></
 ↪div>
            </div>"""
        for lbl in labels
    )
    def _tbl(sub):
        return sub[['Product','Category','Best Size','Best Price','Best $/Oz␣
 ↪(28g)']].to_html(
```

```
            index=False, classes="w-full text-left my-4 text-base", border=0,␣
 ↪escape=False,
            formatters={'Best Price':lambda x:f'${x:,.2f}', 'Best $/Oz (28g)':
 ↪lambda x:f'${x:,.2f}'})
    sections = [f"""<div class="mt-6">
          <h3 class="text-lg font-semibold text-white">{lbl}</h3>
          <div class="overflow-x-auto">{_tbl(sub)}</div>
        </div>"""
        for lbl in labels if not bands_df[bands_df['Price Band'].
 ↪astype(str)==lbl].empty
        for sub in [bands_df[bands_df['Price Band'].astype(str)==lbl]]
    ]

    html = f"""
    <section class="mb-6">
      <h2 class="text-3xl font-semibold text-cyan-400 border-b border-gray-700␣
 ↪pb-2">Price band coverage (per product)</h2>
      <div class="grid grid-cols-1 sm:grid-cols-3 lg:grid-cols-5 gap-3␣
 ↪mt-3">{chips}</div>
      {''.join(sections) if sections else "<p class='text-gray-400 mt-4'>No␣
 ↪products available after filters.</p>"}
    </section>"""
    display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

[7]:
```
# --- Optimized Medical Flower Price Report with Enhanced Error Handling &␣
 ↪Performance ---

import io
import re
import base64
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import HTML
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# ========================
# SECTION 1: DATA VALIDATION & PREPARATION
# ========================

def validate_and_prepare_data():
    """Validate input data and handle edge cases gracefully."""
```

```
    try:
        # Validate price_df exists and has required columns
        if 'price_df' not in globals() or price_df.empty:
            raise ValueError("No price data available")

        required_cols = ['name', 'slug', 'report_category', 'size_label',␣
↪'price', 'weight_g', 'price_per_oz']
        missing_cols = [col for col in required_cols if col not in price_df.
↪columns]
        if missing_cols:
            raise ValueError(f"Missing required columns: {missing_cols}")

        # Clean and validate data
        clean_df = price_df.copy()
        clean_df = clean_df.dropna(subset=['price_per_oz', 'weight_g', 'price'])
        clean_df = clean_df[clean_df['price_per_oz'] > 0]
        clean_df = clean_df[clean_df['price'] > 0]

        if clean_df.empty:
            raise ValueError("No valid price data after cleaning")

        return clean_df

    except Exception as e:
        print(f"Data validation error: {e}")
        # Return minimal dummy data to prevent complete failure
        return pd.DataFrame({
            'name': ['Sample Product'],
            'slug': ['sample-product'],
            'report_category': ['Bulk Value'],
            'size_label': ['1g'],
            'price': [10.0],
            'weight_g': [1.0],
            'price_per_oz': [283.5]
        })

def safe_dispensary_info():
    """Safely extract dispensary information with fallbacks."""
    try:
        if 'dispensary_info' in globals() and not dispensary_info.empty:
            return {
                'name': str(dispensary_info.get('name', 'Unknown Dispensary')),
                'address': str(dispensary_info.get('address', '')),
                'city': str(dispensary_info.get('city', '')),
                'state': str(dispensary_info.get('state', '')),
                'rating': float(dispensary_info.get('rating', 0)),
                'reviews_count': int(dispensary_info.get('reviews_count', 0)),
```

```
                'phone_number': str(dispensary_info.get('phone_number', 'N/A')),
                'web_url': str(dispensary_info.get('web_url', '#'))
            }
    except Exception as e:
        print(f"Dispensary info error: {e}")

    return {
        'name': 'Unknown Dispensary',
        'address': '',
        'city': '',
        'state': '',
        'rating': 0.0,
        'reviews_count': 0,
        'phone_number': 'N/A',
        'web_url': '#'
    }

# Initialize validated data
try:
    price_df_clean = validate_and_prepare_data()
    dispensary_data = safe_dispensary_info()
    print(f" Data validation complete. Processing {len(price_df_clean)} valid␣
 ↪items.")
except Exception as e:
    print(f" Critical error in data preparation: {e}")
    raise

# ========================
# SECTION 2: CORE DATA PROCESSING
# ========================

def calculate_category_order(df):
    """Calculate category order by median price with error handling."""
    try:
        if df.empty:
            return []
        return (df.groupby('report_category')['price_per_oz']
                .median()
                .sort_values()
                .index.tolist())
    except Exception:
        return df['report_category'].unique().tolist()

def calculate_savings_analysis(df):
    """Calculate bulk savings with comprehensive error handling."""
    savings_detail = pd.DataFrame()
```

```
    try:
        # Find multi-size products
        multi_size_slugs = df['slug'].value_counts()[lambda s: s > 1].index
        if len(multi_size_slugs) == 0:
            return savings_detail

        multi_size_df = df[df['slug'].isin(multi_size_slugs)].copy()

        # Get min/max rows by weight per slug
        min_rows = multi_size_df.loc[multi_size_df.groupby('slug')['weight_g'].
↪idxmin()]
        max_rows = multi_size_df.loc[multi_size_df.groupby('slug')['weight_g'].
↪idxmax()]

        # Merge and calculate savings
        small_cols = ['slug', 'name', 'report_category', 'size_label',␣
↪'weight_g', 'price', 'price_per_oz']
        large_cols = ['slug', 'size_label', 'weight_g', 'price', 'price_per_oz']

        savings_detail = pd.merge(
            min_rows[small_cols].rename(columns={
                'size_label': 'size_label_small',
                'weight_g': 'weight_g_small',
                'price': 'price_small',
                'price_per_oz': 'price_per_oz_small'
            }),
            max_rows[large_cols].rename(columns={
                'size_label': 'size_label_large',
                'weight_g': 'weight_g_large',
                'price': 'price_large',
                'price_per_oz': 'price_per_oz_large'
            }),
            on='slug',
            how='inner'
        )

        # Safe percentage calculation
        savings_detail['savings_pct'] = np.where(
            savings_detail['price_per_oz_small'] > 0,
            (1 - (savings_detail['price_per_oz_large'] /␣
↪savings_detail['price_per_oz_small'])) * 100,
            0
        )

        savings_detail['delta_per_oz'] = (
            savings_detail['price_per_oz_small'] -␣
↪savings_detail['price_per_oz_large']
```

```
        )

        # Keep only positive savings
        savings_detail = (
            savings_detail[savings_detail['savings_pct'] > 0]
            .sort_values('savings_pct', ascending=False)
            .reset_index(drop=True)
        )

    except Exception as e:
        print(f"Savings analysis error: {e}")
        savings_detail = pd.DataFrame()

    return savings_detail

def efficient_sizes_analysis(df):
    """Calculate efficient sizes with improved error handling."""
    EPS = 1e-6

    def _efficient_sizes_df(group):
        try:
            g = group[['slug','name','size_label','weight_g','price','price_per_oz']].dropna().copy()
            if g.empty:
                return pd.DataFrame()

            g['ppoz_round'] = g['price_per_oz'].round(2)
            g = g.sort_values(['ppoz_round','weight_g']).groupby('ppoz_round', as_index=False).head(1)
            g = g.sort_values('weight_g').reset_index(drop=True)

            kept = []
            best_ppoz_so_far = np.inf

            for _, row in g.iterrows():
                p = row['price_per_oz']
                if p < best_ppoz_so_far - EPS:
                    kept.append(row)
                    best_ppoz_so_far = p

            if kept:
                return pd.DataFrame(kept).reset_index(drop=True).drop(columns=['ppoz_round'])
            else:
                idx = group['price_per_oz'].idxmin()
```

```
                return group.loc[[idx],␣
↪['slug','name','size_label','weight_g','price','price_per_oz']]

        except Exception:
            # Fallback to best price per oz
            try:
                idx = group['price_per_oz'].idxmin()
                return group.loc[[idx],␣
↪['slug','name','size_label','weight_g','price','price_per_oz']]
            except Exception:
                return pd.DataFrame()

    def _sizes_badge_from_df(sizedf):
        try:
            def _key(lbl):
                try:
                    return float(lbl.replace('g',''))
                except Exception:
                    return 9e9
            labels = sorted(sizedf['size_label'].tolist(), key=_key)
            return " → ".join(labels)
        except Exception:
            return "N/A"

    # Build efficient sizes map
    eff_map = {}
    for slug, g in df.groupby('slug', sort=False):
        try:
            eff_map[slug] = _efficient_sizes_df(g)
        except Exception:
            eff_map[slug] = pd.DataFrame()

    return eff_map, _sizes_badge_from_df

def calculate_best_per_slug(df, eff_map, sizes_badge_func, savings_detail):
    """Calculate best product per slug with error handling."""
    try:
        rows = []
        for _, g in df.groupby('slug', sort=False):
            g2 = g.sort_values(['price_per_oz','weight_g'], ascending=[True,␣
↪False])
            rows.append(g2.iloc[0])

        best_per_slug = pd.DataFrame(rows).copy()
        best_per_slug['Efficient Sizes'] = best_per_slug['slug'].map(
            lambda s: sizes_badge_func(eff_map.get(s, pd.DataFrame()))
        )
```

```
        # Merge savings data
        if not savings_detail.empty:
            best_per_slug = best_per_slug.merge(
                savings_detail[['slug','savings_pct']],
                on='slug', how='left'
            )
        else:
            best_per_slug['savings_pct'] = pd.NA

        return best_per_slug

    except Exception as e:
        print(f"Best per slug calculation error: {e}")
        return pd.DataFrame()

# Execute core processing
cat_order = calculate_category_order(price_df_clean)
savings_detail = calculate_savings_analysis(price_df_clean)
eff_map, sizes_badge_func = efficient_sizes_analysis(price_df_clean)
best_per_slug = calculate_best_per_slug(price_df_clean, eff_map,␣
 ↪sizes_badge_func, savings_detail)

# =========================
# SECTION 3: EXECUTIVE SUMMARY CALCULATIONS
# =========================

def calculate_executive_metrics(best_per_slug, savings_detail):
    """Calculate all executive summary metrics with error handling."""
    try:
        if best_per_slug.empty:
            return {
                'best_ppoz': 0, 'best_name': 'N/A', 'best_size': 'N/A',␣
 ↪'best_price': 0,
                'overall_median': 0, 'overall_p25': 0, 'overall_p75': 0,
                'cat_stats': [], 'band_counts': pd.Series(), 'band_shares': pd.
 ↪Series(),
                'pct_leq60': 0, 'pct_leq90': 0, 'shake_share': 0,␣
 ↪'shake_min_ppoz': None,
                'savings_headline': {}, 'top3': pd.DataFrame(), 'verdict_label':
 ↪ 'No data available'
            }

        # Overall best value (exclude Shake/Popcorn/Trim)
        value_pool = best_per_slug[best_per_slug['report_category'] != 'Shake/
 ↪Popcorn/Trim']
```

```
        if value_pool.empty:
            value_pool = best_per_slug.copy()

        best_row = value_pool.loc[value_pool['price_per_oz'].idxmin()]

        # Distribution stats
        overall_median = float(best_per_slug['price_per_oz'].median())
        overall_p25 = float(best_per_slug['price_per_oz'].quantile(0.25))
        overall_p75 = float(best_per_slug['price_per_oz'].quantile(0.75))

        # Category stats
        cat_stats = []
        for cat in cat_order:
            sub = best_per_slug[best_per_slug['report_category']==cat]
            if not sub.empty:
                cat_stats.append({
                    'cat': cat,
                    'n_products': int(sub['slug'].nunique()),
                    'median': float(sub['price_per_oz'].median()),
                    'min': float(sub['price_per_oz'].min())
                })

        # Price bands
        band_labels = ["  $60", "$61-$90", "$91-$120", "$121-$200", ">$200"]
        band_bins = [0, 60, 90, 120, 200, np.inf]
        band_series = pd.cut(best_per_slug['price_per_oz'], bins=band_bins,␣
↪labels=band_labels, right=True, include_lowest=True)
        band_counts = band_series.value_counts().reindex(band_labels,␣
↪fill_value=0)
        band_shares = (band_counts / len(best_per_slug)).fillna(0)

        pct_leq60 = float((best_per_slug['price_per_oz'] <= 60).mean())
        pct_leq90 = float((best_per_slug['price_per_oz'] <= 90).mean())

        # Shake analysis
        shake_sub = best_per_slug[best_per_slug['report_category']=='Shake/
↪Popcorn/Trim']
        shake_share = float(len(shake_sub) / len(best_per_slug)) if␣
↪len(best_per_slug) else 0.0
        shake_min_ppoz = float(shake_sub['price_per_oz'].min()) if not␣
↪shake_sub.empty else None

        # Savings headline
        savings_headline = {}
        if not savings_detail.empty:
            top_sav = savings_detail.iloc[0]
            savings_headline = {
```

```
                'product': str(top_sav['name']),
                'pct': float(top_sav['savings_pct']),
                'small_label': str(top_sav['size_label_small']),
                'small_ppoz': float(top_sav['price_per_oz_small']),
                'large_label': str(top_sav['size_label_large']),
                'large_ppoz': float(top_sav['price_per_oz_large']),
            }

        # Top 3 products
        top3 =␣
↪(value_pool[['name','size_label','price','price_per_oz','report_category','Efficient␣
↪Sizes']]
                .sort_values('price_per_oz').head(3)
                .rename(columns={'name':'Product','size_label':'Best␣
↪Size','price':'Best Price'}))

        # Value verdict
        if pct_leq60 >= 0.50:
            verdict_label = "Strong value ( 50% of products   $60/oz, 28g norm)"
        elif pct_leq60 >= 0.25:
            verdict_label = "Mixed value (25-49% of products   $60/oz, 28g␣
↪norm)"
        else:
            verdict_label = "Premium-leaning (<25% of products   $60/oz, 28g␣
↪norm)"

        return {
            'best_ppoz': float(best_row['price_per_oz']),
            'best_name': str(best_row['name']),
            'best_size': str(best_row['size_label']),
            'best_price': float(best_row['price']),
            'overall_median': overall_median,
            'overall_p25': overall_p25,
            'overall_p75': overall_p75,
            'cat_stats': cat_stats,
            'band_counts': band_counts,
            'band_shares': band_shares,
            'pct_leq60': pct_leq60,
            'pct_leq90': pct_leq90,
            'shake_share': shake_share,
            'shake_min_ppoz': shake_min_ppoz,
            'savings_headline': savings_headline,
            'top3': top3,
            'verdict_label': verdict_label
        }

    except Exception as e:
```

```
        print(f"Executive metrics calculation error: {e}")
        return {
            'best_ppoz': 0, 'best_name': 'Error', 'best_size': 'N/A',␣
 ↪'best_price': 0,
            'overall_median': 0, 'overall_p25': 0, 'overall_p75': 0,
            'cat_stats': [], 'band_counts': pd.Series(), 'band_shares': pd.
 ↪Series(),
            'pct_leq60': 0, 'pct_leq90': 0, 'shake_share': 0, 'shake_min_ppoz':␣
 ↪None,
            'savings_headline': {}, 'top3': pd.DataFrame(), 'verdict_label':␣
 ↪'Error calculating metrics'
        }

# Calculate executive metrics
exec_metrics = calculate_executive_metrics(best_per_slug, savings_detail)

# ========================
# SECTION 4: ENHANCED VISUALIZATIONS
# ========================

def create_enhanced_visualizations(df, cat_order):
    """Create enhanced visualizations with better error handling."""
    sns.set_theme(style="whitegrid")
    plt.rcParams.update({
        'figure.facecolor': 'white',
        'axes.facecolor': '#FAFAFA',
        'font.size': 11,
        'font.family': 'sans-serif'
    })

    def _encode_fig(fig, dpi=150):
        try:
            buf = io.BytesIO()
            fig.savefig(buf, format='png', dpi=dpi, bbox_inches='tight',␣
 ↪facecolor='white')
            img = base64.b64encode(buf.getvalue()).decode('utf-8')
            plt.close(fig)
            return img
        except Exception as e:
            print(f"Figure encoding error: {e}")
            plt.close(fig)
            return ""

    # Enhanced box plot
    try:
        fig1, ax1 = plt.subplots(figsize=(14, 6))
```

```
        colors = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4', '#FFEAA7'][:
↪len(cat_order)]

        if not df.empty and len(cat_order) > 0:
            sns.boxplot(
                data=df, x="price_per_oz", y="report_category",
                order=cat_order, palette=colors, ax=ax1,
                fliersize=4, linewidth=1.5
            )

            # Add median labels
            for i, cat in enumerate(cat_order):
                try:
                    cat_data = df[df['report_category']==cat]['price_per_oz']
                    if not cat_data.empty:
                        median_val = cat_data.median()
                        ax1.text(median_val, i, f'${median_val:.0f}',
                                verticalalignment='center', fontweight='bold',
                                bbox=dict(boxstyle='round,pad=0.3',␣
↪facecolor='white', alpha=0.8))
                except Exception:
                    continue

        ax1.set_title("Price Distribution by Product Category", fontsize=16,␣
↪fontweight='bold', pad=20)
        ax1.set_xlabel("Price per Ounce ($, 28g normalized)", fontsize=12,␣
↪fontweight='medium')
        ax1.set_ylabel("")
        ax1.grid(axis='x', alpha=0.3, linestyle='--')

        img_box = _encode_fig(fig1)

    except Exception as e:
        print(f"Box plot error: {e}")
        img_box = ""

    # Enhanced ECDF
    try:
        fig2, ax2 = plt.subplots(figsize=(14, 6))

        if not df.empty and len(cat_order) > 0:
            for i, cat in enumerate(cat_order):
                try:
                    cat_data = df[df['report_category']==cat]['price_per_oz']
                    if not cat_data.empty:
                        x_vals = np.sort(cat_data)
                        y_vals = np.arange(1, len(x_vals) + 1) / len(x_vals)
```

```
                        ax2.plot(x_vals, y_vals, label=cat, color=colors[i %␣
 ↪len(colors)],
                                 linewidth=2.5, alpha=0.8)
                except Exception:
                    continue

        ax2.set_title("Cumulative Price Distribution Comparison", fontsize=16,␣
 ↪fontweight='bold', pad=20)
        ax2.set_xlabel("Price per Ounce ($, 28g normalized)", fontsize=12,␣
 ↪fontweight='medium')
        ax2.set_ylabel("Cumulative Percentage", fontsize=12,␣
 ↪fontweight='medium')
        ax2.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
        ax2.grid(alpha=0.3, linestyle='--')
        ax2.set_yticklabels([f'{int(y*100)}%' for y in ax2.get_yticks()])

        img_ecdf = _encode_fig(fig2)

    except Exception as e:
        print(f"ECDF plot error: {e}")
        img_ecdf = ""

    return img_box, img_ecdf

# Create visualizations
img_box, img_ecdf = create_enhanced_visualizations(price_df_clean, cat_order)
# FULL Price Bands (per product) generation for a collapsible section
# Build bands_df from canonical products - EXCLUDE Shake/Popcorn/Trim
base_for_bands = best_per_slug[best_per_slug['report_category'] != 'Shake/
 ↪Popcorn/Trim'].copy()

bands_df = (
    ␣
 ↪base_for_bands[['name','report_category','size_label','price','price_per_oz']]
    .rename(columns={
        'name': 'Product',
        'report_category': 'Category',
        'size_label': 'Best Size',
        'price': 'Best Price',
        'price_per_oz': 'Best $/Oz (28g)'
    })
    .copy()
)

bands_df['Price Band'] = pd.cut(
    bands_df['Best $/Oz (28g)'],
```

```
    bins=[0, 60, 90, 120, 200, float('inf')],
    labels=[" $60", "$61-$90", "$91-$120", "$121-$200", ">$200"],
    right=True, include_lowest=True
)

# Order by band then by price then by product
bands_df['Price Band'] = pd.Categorical(
    bands_df['Price Band'],
    categories=[" $60", "$61-$90", "$91-$120", "$121-$200", ">$200"],
    ordered=True
)
bands_df = bands_df.sort_values(['Price Band','Best $/Oz (28g)','Product']).
 ↪reset_index(drop=True)


# Summary counts (string HTML chips already exist as bands_html in Exec␣
 ↪Summary),
# but we will build a collapsible panel with unlimited rows per band below.
def _format_band_table_html(sub: pd.DataFrame) -> str:
    return sub[['Product','Category','Best Size','Best Price','Best $/Oz␣
 ↪(28g)']].to_html(
        index=False,
        classes="w-full text-left my-4 text-base",
        border=0,
        formatters={
            'Best Price': lambda x: f'${x:,.2f}',
            'Best $/Oz (28g)': lambda x: f'${x:,.2f}',
        },
        escape=False
    )

full_bands_sections = []
for label in [" $60", "$61-$90", "$91-$120", "$121-$200", ">$200"]:
    sub = bands_df[bands_df['Price Band'].astype(str) == label]
    if sub.empty:
        continue
    full_bands_sections.append(
        f"""
        <div class="mt-4">
          <h4 class="text-lg font-semibold text-white">{label}</h4>
          <div class="overflow-x-auto">{_format_band_table_html(sub)}</div>
        </div>
        """
    )
full_bands_html = (
    f"""
    <details class="group bg-gray-800 border border-gray-700 rounded-lg mt-4">
```

```
      <summary class="cursor-pointer select-none list-none px-4 py-3 flex␣
 ↪items-center justify-between">
        <span class="text-white font-semibold">Full Price Bands - per product␣
 ↪(28g-normalized, unlimited)</span>
        <span class="text-gray-400 text-sm group-open:hidden">Click to expand</
 ↪span>
        <span class="text-gray-400 text-sm hidden group-open:inline">Click to␣
 ↪collapse</span>
      </summary>
      <div class="px-4 pb-4 pt-0">
        {''.join(full_bands_sections) if full_bands_sections else "<p␣
 ↪class='text-gray-400 mt-2'>No products available.</p>"}
      </div>
    </details>
    """
)


# ========================
# SECTION 5: HTML GENERATION
# ========================

def generate_enhanced_html(dispensary_data, exec_metrics, savings_detail,␣
 ↪best_per_slug,
                           price_df_clean, cat_order, img_box, img_ecdf,␣
 ↪eff_map, sizes_badge_func):
    """Generate enhanced HTML with better error handling and performance."""

    EMOJIS = {'Bulk Value':' ', 'Pre-Pack Specialty':' ', 'Shake/Popcorn/Trim':
 ↪' '}

    def safe_format_currency(value):
        try:
            return f"${float(value):,.2f}"
        except Exception:
            return "$0.00"

    def safe_format_percentage(value):
        try:
            return f"{float(value):.0f}%"
        except Exception:
            return "0%"

    def build_category_kpis():
        kpi_html = ""
        for c in exec_metrics['cat_stats']:
```

```
            try:
                kpi_html += f'''
                <div class="bg-gradient-to-br from-gray-800 to-gray-700 border␣
↪border-gray-600 rounded-xl p-4 transform hover:scale-105␣
↪transition-transform duration-200">
                    <div class="text-sm text-gray-400 font-medium">{c["cat"]}</
↪div>
                    <div class="mt-2 text-xl font-bold␣
↪text-white">${c["median"]:.0f}/oz
                        <span class="text-xs text-gray-400␣
↪font-normal">(median, 28g)</span>
                    </div>
                    <div class="mt-1 text-xs text-gray-400">min ${c["min"]:.0f}␣
↪• {c["n_products"]} products</div>
                </div>
                '''
            except Exception:
                continue
        return kpi_html

    def build_price_bands():
        bands_html = ""
        band_labels = ["  $60", "$61-$90", "$91-$120", "$121-$200", ">$200"]
        band_colors = ['bg-green-600', 'bg-blue-600', 'bg-yellow-600',␣
↪'bg-orange-600', 'bg-red-600']

        for i, label in enumerate(band_labels):
            try:
                color_class = band_colors[i] if i < len(band_colors) else␣
↪'bg-gray-600'
                count = int(exec_metrics['band_counts'].get(label, 0))
                share = exec_metrics['band_shares'].get(label, 0) * 100

                bands_html += f'''
                <div class="bg-gradient-to-br from-gray-800 to-gray-700 border␣
↪border-gray-600 rounded-xl p-4 transform hover:scale-105␣
↪transition-transform duration-200">
                    <div class="flex items-center gap-2">
                        <div class="{color_class} w-3 h-3 rounded-full"></div>
                        <div class="text-sm text-gray-400 font-medium">{label}␣
↪(28g)</div>
                    </div>
                    <div class="mt-2 text-xl font-bold text-white">{count}
                        <span class="text-xs text-gray-400 font-normal">({share:
↪.0f}%)</span>
                    </div>
```

```
                </div>
                '''
            except Exception:
                continue
        return bands_html

    def build_savings_or_shake_kpi():
        if exec_metrics['savings_headline']:
            try:
                sh = exec_metrics['savings_headline']
                return f'''
                <div class="bg-gradient-to-br from-green-800 to-green-700␣
↪border border-green-600 rounded-xl p-4">
                    <div class="text-sm text-green-200 font-medium">  Largest␣
↪bulk savings</div>
                    <div class="mt-2 text-xl font-bold text-white">{sh["pct"]:.
↪0f}%</div>
                    <div class="mt-1 text-sm text-green-100">{sh["product"][:
↪30]}{'...' if len(sh["product"]) > 30 else ''}</div>
                    <div class="mt-1 text-xs text-green-200">
                        {sh["small_label"]} @ ${sh["small_ppoz"]:.0f}/oz →
                        {sh["large_label"]} @ ${sh["large_ppoz"]:.0f}/oz
                    </div>
                </div>
                '''
            except Exception:
                pass

        try:
            return f'''
            <div class="bg-gradient-to-br from-gray-800 to-gray-700 border␣
↪border-gray-600 rounded-xl p-4">
                <div class="text-sm text-gray-400 font-medium">  Shake/Popcorn␣
↪coverage</div>
                <div class="mt-2 text-xl font-bold␣
↪text-white">{exec_metrics["shake_share"]*100:.0f}% of products</div>
                {f'<div class="mt-1 text-xs text-gray-400">cheapest:␣
↪${exec_metrics["shake_min_ppoz"]:.0f}/oz (28g)</div>' if␣
↪exec_metrics["shake_min_ppoz"] is not None else ''}
            </div>
            '''
        except Exception:
            return '<div class="bg-gray-800 p-4 rounded-xl"><span␣
↪class="text-gray-400">Data unavailable</span></div>'

    def build_top3_products():
```

```
        if exec_metrics['top3'].empty:
            return '<p class="text-gray-400">No products available</p>'

        items = []
        medals = [' ', ' ', ' ']

        for i, (_, r) in enumerate(exec_metrics['top3'].iterrows()):
            try:
                medal = medals[i] if i < len(medals) else ' '
                product_name = str(r["Product"])[:40] + ('...' if␣
↪len(str(r["Product"])) > 40 else '')

                items.append(f'''
                <div class="bg-gray-800 border border-gray-600 rounded-lg p-4␣
↪flex items-center justify-between hover:bg-gray-750 transition-colors␣
↪duration-200">
                    <div class="flex items-center gap-3">
                        <span class="text-2xl">{medal}</span>
                        <div>
                            <div class="font-semibold␣
↪text-white">{product_name}</div>
                            <div class="text-sm␣
↪text-gray-400">{r["report_category"]} • {r["Best Size"]} •␣
↪{safe_format_currency(r["Best Price"])}</div>
                        </div>
                    </div>
                    <div class="text-right">
                        <div class="text-lg font-bold␣
↪text-cyan-400">{safe_format_currency(r["price_per_oz"])}/oz</div>
                        <div class="text-xs text-gray-400">28g normalized</div>
                    </div>
                </div>
                ''')
            except Exception:
                continue

        return '<div class="space-y-3">' + "".join(items) + '</div>'

    def build_category_leaderboards():
        if price_df_clean.empty or not cat_order:
            return '<p class="text-gray-400">No category data available</p>'

        sections = []
        for cat in cat_order:
            try:
                sub = price_df_clean[price_df_clean['report_category'] == cat]
                if sub.empty:
```

```
                    continue

                # Build best per slug for this category
                rows = []
                for slug, g in sub.groupby('slug', sort=False):
                    try:
                        g2 = g.sort_values(['price_per_oz','weight_g'],␣
↪ascending=[True, False])
                        row = g2.iloc[0].copy()
                        row['Efficient Sizes'] = sizes_badge_func(eff_map.
↪get(slug, pd.DataFrame()))

                        # Add savings info
                        if not savings_detail.empty:
                            spct = savings_detail.
↪loc[savings_detail['slug']==slug, 'savings_pct']
                            row['Max Savings vs Smallest'] = spct.iloc[0] if␣
↪not spct.empty else np.nan
                        else:
                            row['Max Savings vs Smallest'] = np.nan

                        rows.append(row)
                    except Exception:
                        continue

                if not rows:
                    continue

                best_df = pd.DataFrame(rows)
                best_df = best_df.sort_values(['price_per_oz','name'],␣
↪ascending=[True, True]).head(20)  # Limit for performance

                table_html =␣
↪best_df[['name','size_label','price','price_per_oz','Efficient Sizes','Max␣
↪Savings vs Smallest']].to_html(
                    index=False,
                    classes="w-full text-left text-sm bg-gray-800 rounded-lg␣
↪overflow-hidden",
                    formatters={
                        'price': safe_format_currency,
                        'price_per_oz': lambda x: f'<span class="font-semibold␣
↪text-cyan-400">{safe_format_currency(x)}</span>',
                        'Max Savings vs Smallest': lambda x: (f'<span␣
↪class="font-semibold text-green-400">{x:.0f}%</span>'
                                                              if pd.notna(x) else␣
↪'<span class="text-gray-400">-</span>')
```

```
                    },
                    escape=False,
                    table_id=f"table-{cat.lower().replace(' ', '-')}"
                )

                sections.append(f'''
                <div class="mb-8">
                    <h2 class="text-2xl font-bold text-cyan-400 border-b-2 border-cyan-400 pb-2 mb-4">
                        {EMOJIS.get(cat,' ')} {cat}
                    </h2>
                    {table_html}
                </div>
                ''')

            except Exception as e:
                print(f"Error building leaderboard for {cat}: {e}")
                continue

        return "\n".join(sections)

    def build_savings_table():
        if savings_detail.empty:
            return '<p class="text-gray-400 my-4">No multi-size products with positive ounce-price savings found.</p>'

        try:
            cols = [
                'name', 'report_category', 'size_label_small', 'weight_g_small', 'price_small', 'price_per_oz_small',
                'size_label_large', 'weight_g_large', 'price_large', 'price_per_oz_large', 'savings_pct', 'delta_per_oz'
            ]

            display_df = savings_detail[cols].rename(columns={
                'name':'Product', 'report_category':'Category', 'size_label_small':'Small Size',
                'weight_g_small':'Small (g)', 'price_small':'Small Price', 'price_per_oz_small':'Small $/oz (28g)',
                'size_label_large':'Large Size', 'weight_g_large':'Large (g)', 'price_large':'Large Price',
                'price_per_oz_large':'Large $/oz (28g)', 'savings_pct':'Savings %', 'delta_per_oz':'Δ $/oz (28g)'
            })

            return display_df.to_html(
```

```
                index=False,
                classes="w-full text-left text-sm overflow-x-auto",
                escape=False,
                formatters={
                    'Small Price': safe_format_currency,
                    'Small $/oz (28g)': safe_format_currency,
                    'Large Price': safe_format_currency,
                    'Large $/oz (28g)': safe_format_currency,
                    'Savings %': lambda x: f'<span class="font-semibold␣
↪text-green-400">{x:.0f}%</span>',
                    'Δ $/oz (28g)': lambda x: f'<span class="font-semibold␣
↪text-cyan-400">{safe_format_currency(x)}</span>',
                    'Small (g)': lambda x: f'{x:.0f}g' if abs(x - round(x)) <␣
↪1e-6 else f'{x:g}g',
                    'Large (g)': lambda x: f'{x:.0f}g' if abs(x - round(x)) <␣
↪1e-6 else f'{x:g}g',
                }
            )
        except Exception as e:
            print(f"Savings table error: {e}")
            return '<p class="text-red-400">Error generating savings table</p>'

    # Build HTML components
    cat_kpi_html = build_category_kpis()
    bands_html = build_price_bands()
    additional_kpi_html = build_savings_or_shake_kpi()
    top3_html = build_top3_products()
    category_leaderboards = build_category_leaderboards()
    savings_table_html = build_savings_table()

    # Main HTML template
    html_output = f'''
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="description" content="Medical flower price analysis report for␣
↪{dispensary_data['name']}" />
    <title>Medical Flower Price Report - {dispensary_data['name']}</title>
    <script src="https://cdn.tailwindcss.com?plugins=typography"></script>
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;
↪700&display=swap" rel="stylesheet" />
    <style>
```

```
        body {{ font-family: 'Inter', system-ui, sans-serif; }}
        .hover\\:scale-105:hover {{ transform: scale(1.05); }}
        .bg-gray-750 {{ background-color: #374151; }}
        table {{ border-collapse: collapse; }}
        th, td {{ padding: 12px 8px; border-bottom: 1px solid #374151; }}
        th {{ background-color: #1F2937; font-weight: 600; }}
        .transition-transform {{ transition: transform 0.2s ease-in-out; }}
        @media (max-width: 768px) {{
            .text-5xl {{ font-size: 2.5rem; }}
            .grid-cols-4 {{ grid-template-columns: repeat(2, 1fr); }}
        }}
    </style>
</head>
<body class="bg-gray-900 text-gray-200 min-h-screen">
    <main class="max-w-6xl mx-auto p-4 sm:p-6">

        <!-- Dispensary Header -->
        <header class="text-center mb-8">
            <h1 class="text-4xl sm:text-5xl font-extrabold text-white␣
↪mb-4">Medical Flower Price Report</h1>
            <p class="text-lg text-gray-400">Value Analysis for␣
↪{dispensary_data['name']}</p>
        </header>

        <section class="grid grid-cols-1 md:grid-cols-2 gap-6 bg-gray-800 p-6␣
↪rounded-lg border border-gray-700 mb-10">
            <div>
                <h2 class="text-2xl font-semibold␣
↪text-cyan-400">{dispensary_data['name']}</h2>
                <p class="mt-1 text-gray-300">
                    {dispensary_data['address']}<br />
                    {dispensary_data['city']}, {dispensary_data['state']}
                </p>
            </div>
            <div class="text-right space-y-1">
                <p class="text-gray-300"><strong>Rating:</strong> <span␣
↪class="text-cyan-400">{dispensary_data['rating']:.1f} ␣
↪({dispensary_data['reviews_count']} reviews)</span></p>
                <p class="text-gray-300"><strong>Phone:</strong> <a href="tel:
↪{dispensary_data['phone_number']}" class="text-cyan-400 hover:
↪underline">{dispensary_data['phone_number']}</a></p>
                <p class="text-gray-300"><strong>Menu:</strong> <a␣
↪class="text-cyan-400 hover:underline" href="{dispensary_data['web_url']}"␣
↪target="_blank" rel="noopener">View Menu</a></p>
            </div>
        </section>
```

```
        <!-- Executive Summary -->
        <section class="mb-10">
            <h2 class="text-3xl font-semibold text-cyan-400 border-b␣
↪border-gray-700 pb-2">Executive Summary</h2>
            <div class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-3␣
↪mt-4">
                <div class="bg-gray-800 border border-gray-700 rounded-lg p-4">
                    <div class="text-sm text-gray-400">Cheapest␣
↪ounce-equivalent (28g)</div>
                    <div class="mt-1 text-2xl font-bold␣
↪text-cyan-400">{safe_format_currency(exec_metrics['best_ppoz'])}/oz</div>
                    <div class="mt-1 text-xs␣
↪text-gray-400">{exec_metrics['best_name'][:40]}{'...' if␣
↪len(exec_metrics['best_name']) > 40 else ''}</div>
                    <div class="mt-1 text-xs␣
↪text-gray-500">{exec_metrics['best_size']} •␣
↪{safe_format_currency(exec_metrics['best_price'])}</div>
                </div>
                <div class="bg-gray-800 border border-gray-700 rounded-lg p-4">
                    <div class="text-sm text-gray-400">Typical price (per␣
↪product, 28g norm)</div>
                    <div class="mt-1 text-2xl font-bold␣
↪text-white">${exec_metrics['overall_median']:.0f}/oz</div>
                    <div class="mt-1 text-xs text-gray-400">IQR␣
↪${exec_metrics['overall_p25']:.0f}–${exec_metrics['overall_p75']:.0f}</div>
                </div>
                <div class="bg-gray-800 border border-gray-700 rounded-lg p-4">
                    <div class="text-sm text-gray-400">Low-price coverage␣
↪(28g)</div>
                    <div class="mt-1 text-2xl font-bold␣
↪text-white">{exec_metrics['pct_leq60']*100:.0f}%  $60/oz</div>
                    <div class="mt-1 text-xs␣
↪text-gray-400">{exec_metrics['pct_leq90']*100:.0f}%  $90/oz</div>
                </div>
                {additional_kpi_html}
            </div>

            <div class="mt-4">
                <div class="bg-emerald-900/30 border border-emerald-700␣
↪rounded-lg p-3 text-emerald-300 text-sm font-semibold">
                    Bottom line: {exec_metrics['verdict_label']}
                </div>
            </div>
```

```
            <h3 class="text-xl font-semibold text-white mt-6">Category medians␣
↪& counts</h3>
            <div class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-3␣
↪mt-2">
                {cat_kpi_html}
            </div>

            <h3 class="text-xl font-semibold text-white mt-6">Price band␣
↪coverage (per product)</h3>
            <div class="grid grid-cols-1 sm:grid-cols-3 lg:grid-cols-5 gap-3␣
↪mt-2">
                {bands_html}
            </div>
            <!-- Collapsible FULL Price Bands (unlimited rows per band) -->
            {full_bands_html}
            <h3 class="text-xl font-semibold text-white mt-6">Top 3 best-value␣
↪products (by $/oz)</h3>
            {top3_html}
        </section>

        <!-- Visuals -->
        <section class="mb-6">
            <h2 class="text-3xl font-semibold text-cyan-400 border-b␣
↪border-gray-700 pb-2">Price Distribution Visuals</h2>
            {f'<div class="mt-6 bg-gray-800 rounded-lg p-4 border␣
↪border-gray-700"><h3 class="text-xl font-semibold text-white mb-2">Box Plot␣
↪(quartiles + whiskers)</h3><img src="{img_box}"␣
↪alt="Box plot" class="mx-auto rounded bg-white p-2 shadow max-w-full h-auto"␣
↪/></div>' if img_box else '<div class="mt-6 bg-gray-800 rounded-lg p-4␣
↪border border-gray-700"><p class="text-gray-400">Box plot unavailable</p></
↪div>'}

            {f'<div class="mt-6 bg-gray-800 rounded-lg p-4 border␣
↪border-gray-700"><h3 class="text-xl font-semibold text-white mb-2">ECDF␣
↪Overlay (cumulative comparison)</h3><img src="data:image/png;
↪base64,{img_ecdf}" alt="ECDF overlay" class="mx-auto rounded bg-white p-2␣
↪shadow max-w-full h-auto" /></div>' if img_ecdf else '<div class="mt-6␣
↪bg-gray-800 rounded-lg p-4 border border-gray-700"><p␣
↪class="text-gray-400">ECDF plot unavailable</p></div>'}
        </section>

        <!-- Dynamic category leaderboards -->
        {category_leaderboards}

        <!-- FULL Bulk Savings Spotlight -->
        <section class="mb-12">
```

```
            <h2 class="text-3xl font-semibold text-cyan-400 border-b␣
 ↪border-gray-700 pb-2 mt-10">Bulk Savings Spotlight - Full Detail</h2>
            <div class="overflow-x-auto">
                {savings_table_html}
            </div>
        </section>

        <footer class="text-center text-sm text-gray-500 mt-10 border-t␣
 ↪border-gray-700 pt-4">
            Report generated on {datetime.now().strftime('%B %d, %Y at %I:%M␣
 ↪%p')}.
        </footer>
    </main>
</body>
</html>
'''

    return html_output

# Generate final HTML
try:
    html_output = generate_enhanced_html(
        dispensary_data, exec_metrics, savings_detail, best_per_slug,
        price_df_clean, cat_order, img_box, img_ecdf, eff_map, sizes_badge_func
    )

    # Save to file with date (dispensaries rarely update menus more than once a␣
 ↪day) timestamp and dispensary name
    from datetime import datetime
    timestamp = datetime.now().strftime('%Y%m%d')
    dispensary_name = dispensary_data['name'].replace(' ', '_').replace('-',␣
 ↪'_').replace('/', '-')
    # Write to file
    output_filename = f"{dispensary_name}_{timestamp}_report.html"
    # Check if output directory exists, if not create it
    import os
    output_parent_dir = 'flower_reports_showcase' # main parent directory
    output_report_dir = 'reports' # subdirectory for reports
    output_path = os.path.join(output_parent_dir, output_report_dir)
    if not os.path.exists(output_path):
        os.makedirs(output_path)
    output_path = os.path.join(output_path, output_filename)

    # Write the HTML to file
    with open(output_path, 'w', encoding='utf-8') as f:
        f.write(html_output)
```

```
    print("  Enhanced HTML report generated successfully!")
    display(HTML(html_output))

except Exception as e:
    print(f"  HTML generation error: {e}")
    # Fallback minimal HTML
    fallback_html = f'''
    <!DOCTYPE html>
    <html><head><title>Error</title></head>
    <body style="font-family: Arial, sans-serif; padding: 20px; background:␣
 ↪#1a1a1a; color: white;">
        <h1>Report Generation Error</h1>
        <p>An error occurred while generating the full report: {e}</p>
        <p>Please check your data and try again.</p>
    </body></html>
    '''
    display(HTML(fallback_html))
```

```
  Data validation complete. Processing 271 valid items.
  Enhanced HTML report generated successfully!
<IPython.core.display.HTML object>
```