Untitled-2

August 4, 2025

[1]:
```
# --- Cell 1: Setup & Dependencies ---
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

# Install dependencies silently
!pip install --upgrade jinja2 pandas requests seaborn matplotlib &> /dev/null

# --- 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",
    "Authorization": "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzUxMiJ9.
 ↪eyJqdGkiOiJLUkNhMjRUeSIsImV4cCI6MTc1NDc3NjUxNywiaXNzIjoid2VlZG1hcHMuY29tIiwiaGFzdXJhIjp7ImF
 ↪MFLvc3cVp3blhNXV9RN0rram5yZtoXTxqagwl7oWlxTOywE5waTRSq0CWjiKj4bQIhn6MFt-x_JU7qQtRS7cXzBOIA-
    "wm-user-latlng": LATLNG,
    "Referer": "https://weedmaps.com/",
}

print("  Setup complete.")
```

```
 Setup complete.
```

[2]:

```
# --- Cell 2: 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 ---
DISPENSARY_SLUG = "little-brown-house"  # <-- Change this to your target

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 62 total medical storefronts.

 Selected Dispensary: Reefer Madness Broadway
```

[3]:

```
# --- Cell 3 : 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: 38 items

TOTAL flower items fetched: 38
brand_endorsement: nan
catalog_slug: med-legacy-grown-four-kings-orange-tier-161909074
created_at: 2025-08-02T00:08:43.922Z
current_deal_title: None
deal_ids: []
genetics_tag: None
id: 199195973
is_badged: False
is_endorsed: False
is_online_orderable: True
lab_website: None
last_ordered_date: None
license_type: medical
menu_id: 161909074
```

```
name: MED - Legacy Grown - Four Kings / Orange Tier
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-legacy-grown-four-kings-orange-tier
tags: None
test_result_created_at: None
updated_at: 2025-08-04T03:06:18.988Z
test_result_expired: None
test_result_expires_in: None
avatar_image.large_url: https://images.weedmaps.com/pictures/listings/161/909/07
4/425833607_032A0023.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
/074/425833607_032A0023.jpg?txt64=UHJvZHVjdCBleGFtcGxl&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:
746be2a06713ce717dc8e93f31328900117f8ef038cacd54561b5ac122826213|1790951
external_ids.gram:
746be2a06713ce717dc8e93f31328900117f8ef038cacd54561b5ac122826213|1790951
external_ids.two_grams: nan
external_ids.eighth:
746be2a06713ce717dc8e93f31328900117f8ef038cacd54561b5ac122826213|1790951
external_ids.ounce:
746be2a06713ce717dc8e93f31328900117f8ef038cacd54561b5ac122826213|1790951
external_ids.half_gram: nan
external_ids.quarter:
746be2a06713ce717dc8e93f31328900117f8ef038cacd54561b5ac122826213|1790951
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: 155782703
price.unit: gram
price.quantity: 1
price.label: 1 g
price.compliance_net_mg: 1000.0
price.price: 1.25
price.on_sale: False
price.original_price: 1.25
price.discount_label: None
price_stats.min: None
price_stats.max: None
prices.grams_per_eighth: 3.5
prices.gram: [{'id': 155782703, 'label': '1 g', 'compliance_net_mg': 1000.0,
'price': 1.25, 'on_sale': False, 'original_price': 1.25, 'units': '1',
'gram_unit_price': 1.25, 'weight': {'value': 1.0, 'unit': 'g'}}]
prices.ounce: [{'id': 155782704, 'label': '1/8 oz', 'compliance_net_mg': 3500.0,
'price': 3.75, 'on_sale': False, 'original_price': 3.75, 'units': '1/8',
'gram_unit_price': 1.07, 'weight': {'value': 0.125, 'unit': 'oz'}}, {'id':
155782705, 'label': '1/4 oz', 'compliance_net_mg': 7000.0, 'price': 7.5,
'on_sale': False, 'original_price': 7.5, 'units': '1/4', 'gram_unit_price':
1.07, 'weight': {'value': 0.25, 'unit': 'oz'}}, {'id': 155782706, 'label': '1/2
oz', 'compliance_net_mg': 14000.0, 'price': 15.0, 'on_sale': False,
'original_price': 15.0, 'units': '1/2', 'gram_unit_price': 1.07, 'weight':
{'value': 0.5, 'unit': 'oz'}}]
menu.features: ['static']
menu.id: 161909074
brand_endorsement.brand_id: nan
brand_endorsement.brand_name: nan
brand_endorsement.brand_slug: nan
brand_endorsement.brand_avatar_image_url: nan
brand_endorsement.product_id: nan
brand_endorsement.product_name: nan
brand_endorsement.product_slug: nan
brand_endorsement.best_of_weedmaps_years: nan
brand_endorsement.best_of_weedmaps_nominee_years: nan
brand_endorsement.best_of_weedmaps: nan
brand_endorsement.best_of_weedmaps_nominee: nan
external_ids: nan
menu.listing_menu_types: nan
```

[4]:

```
# --- Cell 4: 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 144 purchasable items.
                                                       name  \
0                        MED - Legacy Grown - Four Kings / Orange Tier
1                        MED - Legacy Grown - Four Kings / Orange Tier
2                        MED - Legacy Grown - Four Kings / Orange Tier
3                        MED - Legacy Grown - Four Kings / Orange Tier
4  MED - Boulder Built - Sudz (Soap x Devil Driver) / Red -Tier Popcorn

                                               slug     report_category  \
0                 med-legacy-grown-four-kings-orange-tier          Bulk Value
1                 med-legacy-grown-four-kings-orange-tier          Bulk Value
2                 med-legacy-grown-four-kings-orange-tier          Bulk Value
3                 med-legacy-grown-four-kings-orange-tier          Bulk Value
4  med-boulder-built-soap-x-devil-driver-red-tier-popcorn  Shake/Popcorn/Trim

  size_label  price  weight_g  price_per_oz
0       3.5g   3.75   3.54375       30.3345
1         7g   7.50   7.08750       30.3345
2        14g  15.00  14.17500       30.3345
3         1g   1.25   1.00000       35.4375
4       3.5g   5.01   3.54375       40.5405
```

[5]:

```
# --- Cell 5: Final HTML Report (Dispensary Header, FULL Bulk Savings detail,␣
↪ONLY A & D plots,
#           dynamic per-category product leaderboards, 28g-normalized $/oz,␣
↪unlimited rows,
#           and clearer "efficient sizes" thresholds progression) ---

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

# =========================
# Part 1 - Data prep
# =========================

# Categories present, ordered by median $/oz (low → high)
cat_order = (
    price_df.groupby('report_category')['price_per_oz']
            .median()
            .sort_values()
            .index
            .tolist()
)

# Identify products (slugs) with multiple sizes for savings analysis
multi_size_slugs = price_df['slug'].value_counts()[lambda s: s > 1].index
multi_size_df    = price_df[price_df['slug'].isin(multi_size_slugs)].copy()

# Build rich small large savings detail per slug (FULL COLUMN SET; NO␣
↪CONDENSING)
savings_detail = pd.DataFrame()
if not multi_size_df.empty:
    # smallest vs largest by grams 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()]

    # explicit columns from each side
    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'
    )

    # compute savings metrics (per-oz)
    # NOTE: price_per_oz should already be 28g-normalized from Cell 4
    savings_detail['savings_pct']   = (1 -␣
↪(savings_detail['price_per_oz_large'] /␣
↪savings_detail['price_per_oz_small'])) * 100
    savings_detail['delta_per_oz']  = (savings_detail['price_per_oz_small'] -␣
↪savings_detail['price_per_oz_large'])
```

```
    # keep only positive savings; sort descending (we'll still render a wide␣
 ↪table)
    savings_detail = (
        savings_detail[savings_detail['savings_pct'] > 0]
        .sort_values('savings_pct', ascending=False)
        .reset_index(drop=True)
    )

# Helper: parse standard package sizes (kept for diagnostics)
STANDARD_GRAM_SIZES = {1, 3.5, 7, 14, 28, 57}
def _is_standard(label):
    try:
        return float(label.replace('g','')) in STANDARD_GRAM_SIZES
    except Exception:
        return False

# =========================
# Part 2 - Canonical product rows + efficient sizes thresholds (for␣
 ↪leaderboards)
# =========================

EPS = 1e-6

def _efficient_sizes_df(group: pd.DataFrame) -> pd.DataFrame:
    """
    Keep the SMALLEST size that unlocks each distinct $/oz tier, then retain␣
 ↪only the
    thresholds where $/oz strictly improves as size increases.
    Result is a compact progression like: 7g → 14g → 28g (if each step reduces␣
 ↪$/oz).
    """
    g = group[['slug','name','size_label','weight_g','price','price_per_oz']].
 ↪dropna(subset=['weight_g','price_per_oz']).copy()

    # Bin $/oz to cents; keep the SMALLEST weight that achieves each $/oz tier
    g['ppoz_round'] = g['price_per_oz'].round(2)
    g = (
        g.sort_values(['ppoz_round','weight_g'])     # ensures smallest weight␣
 ↪per $/oz tier
         .groupby('ppoz_round', as_index=False)
         .head(1)
    )

    # Walk in ascending weight and retain only strict improvements in $/oz
    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:
        kdf = pd.DataFrame(kept).reset_index(drop=True)
        return kdf.drop(columns=['ppoz_round'])
    else:
        # Fallback: just the cheapest-per-oz row
        idx = group['price_per_oz'].idxmin()
        return group.loc[[idx], ['slug','name','size_label','weight_g','price','price_per_oz']]

def _best_per_slug(df: pd.DataFrame) -> pd.DataFrame:
    """Cheapest $/oz per product; break ties by larger weight."""
    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])
    return pd.DataFrame(rows).copy()

def _sizes_badge_from_df(sizedf: pd.DataFrame) -> str:
    def _key(lbl):
        try:
            return float(lbl.replace('g',''))
        except Exception:
            return 9e9
    labels = sorted(sizedf['size_label'].tolist(), key=_key)
    return " → ".join(labels)  # thresholds progression

# Efficient-size map and canonical (best-per-oz) row per slug - for leaderboards & summary
eff_map = {}
for slug, g in price_df.groupby('slug', sort=False):
    eff_map[slug] = _efficient_sizes_df(g)

best_per_slug = _best_per_slug(price_df)
best_per_slug['Efficient Sizes'] = best_per_slug['slug'].map(lambda s: _sizes_badge_from_df(eff_map[s]))

# merge savings pct for canonical rows (if available)
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

# ==========================
# Part 3 - Executive Summary metrics (rich, from data below)
# ==========================

# Overall best $/oz product (exclude Shake/Popcorn/Trim)
value_pool = best_per_slug[best_per_slug['report_category'] != 'Shake/Popcorn/Trim']
if value_pool.empty:
    # Fallback to all products if the filter removes everything
    value_pool = best_per_slug.copy()

best_row   = value_pool.loc[value_pool['price_per_oz'].idxmin()]
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 distribution (canonical per product)
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 medians and product counts
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-band coverage (canonical per product)
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 coverage & cheapest in Shake
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

# Multi-size savings headline (highest savings)
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 best-value products overall (canonical, excluding Shake/Popcorn/Trim)
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'}))

# Quick value verdict based on coverage   $60/oz
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)"

# =========================
# Part 4 - Visualization (ONLY A & D, deprecation-safe)
# =========================
sns.set_theme(style="whitegrid")
```

```
def _encode_fig(fig, dpi=150):
    buf = io.BytesIO()
    fig.savefig(buf, format='png', dpi=dpi, bbox_inches='tight')
    img = base64.b64encode(buf.getvalue()).decode('utf-8')
    plt.close(fig)
    return img

# A) Box plot (set hue to avoid "palette without hue" deprecation; drop legend)
fig1, ax1 = plt.subplots(figsize=(12, 5))
sns.boxplot(
    data=price_df,
    x="price_per_oz", y="report_category",
    order=cat_order, hue="report_category",
    palette="pastel", legend=False, ax=ax1,
    fliersize=5, linewidth=1.2
)
ax1.set_title("Price Distribution by Product Type", fontsize=18, fontweight='bold', pad=12)
ax1.set_xlabel("Price per Ounce ($, 28g norm)", fontsize=13)   # <- label clarified
ax1.set_ylabel("")
img_box = _encode_fig(fig1)

# D) ECDF overlay - let seaborn manage the legend
fig4, ax4 = plt.subplots(figsize=(12, 5))
sns.ecdfplot(
    data=price_df,
    x="price_per_oz", hue="report_category",
    hue_order=cat_order, palette="Set2",
    legend=True, ax=ax4
)
ax4.set_title("Cumulative Distribution of Price per Ounce", fontsize=18, fontweight='bold', pad=12)
ax4.set_xlabel("Price per Ounce ($, 28g norm)")                # <- label clarified
ax4.set_ylabel("Cumulative fraction")
img_ecdf = _encode_fig(fig4)

# =========================
# Part 5 - HTML helpers (leaderboards + formatting)
# =========================
EMOJIS = {'Bulk Value':' ', 'Pre-Pack Specialty':' ', 'Shake/Popcorn/Trim':' '}

def product_leaderboard(df, title, top_n=None):
    """One row per product (slug): Best $/oz + Best Size + Efficient Sizes + Max Savings vs Smallest.
       Set top_n=None (default) to show ALL rows (no truncation)."""
    if df.empty:
        return ""
    # Build best-per-slug within THIS subset
    rows = []
    eff_sizes_col = []
    sav_col = []
    for slug, g in df.groupby('slug', sort=False):
        g2 = g.sort_values(['price_per_oz','weight_g'], ascending=[True, False])
        rows.append(g2.iloc[0])
        eff_sizes_col.append(_sizes_badge_from_df(eff_map[slug]))
        if not savings_detail.empty:
            spct = savings_detail.loc[savings_detail['slug']==slug,␣
↪'savings_pct']
            sav_col.append(spct.iloc[0] if not spct.empty else np.nan)
        else:
            sav_col.append(np.nan)

    best = pd.DataFrame(rows).copy()
    best['Efficient Sizes'] = eff_sizes_col
    best['savings_pct']    = sav_col

    out = (
        best[['name','size_label','price','price_per_oz','Efficient␣
↪Sizes','savings_pct']]
        .rename(columns={
            'name':'Product','size_label':'Best Size',
            'price':'Best Price','price_per_oz':'Best $/Oz (28g)',
            'savings_pct':'Max Savings vs Smallest'
        })
        .sort_values(['Best $/Oz (28g)','Product'], ascending=[True, True])
    )
    # Only cap when top_n is a positive int
    if isinstance(top_n, int) and top_n > 0:
        out = out.head(top_n)

    return (
        f'<h2 class="text-3xl font-semibold text-cyan-400 border-b␣
↪border-gray-700 pb-2 mt-10">{title}</h2>'
        + out.to_html(
            index=False,
            classes="w-full text-left my-6 text-base",
            formatters={
                'Best Price': lambda x: f'<span class="font-semibold␣
↪text-cyan-400">${x:,.2f}</span>',
                'Best $/Oz (28g)': lambda x: f'<span class="font-semibold␣
↪text-cyan-400">${x:,.2f}</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
        )
    )

def build_all_category_leaderboards(df):
    sections = []
    for cat in cat_order:
        sub = df[df['report_category'] == cat]
        if not sub.empty:
            sections.append(product_leaderboard(sub, f"{EMOJIS.get(cat,' ')} {cat}", top_n=None))  # unlimited
    return "\n".join(sections)

# =========================
# Part 6 – HTML (Dispensary Header + Executive Summary + Visuals + Leaderboards + FULL Savings)
# =========================
# Category medians chips
cat_kpi_html = ""
for c in cat_stats:
    cat_kpi_html += (
        f'<div class="bg-gray-800 border border-gray-700 rounded-lg p-3">'
        f'<div class="text-sm text-gray-400">{c["cat"]}</div>'
        f'<div class="mt-1 text-lg font-semibold text-white">${c["median"]:.0f}/oz <span class="text-xs text-gray-400">(median, 28g)</span></div>'
        f'<div class="mt-1 text-xs text-gray-400">min ${c["min"]:.0f} • {c["n_products"]} products</div>'
        f'</div>'
    )

# Price bands chips
bands_html = ""
for label in band_labels:
    bands_html += (
        f'<div class="bg-gray-800 border border-gray-700 rounded-lg p-3">'
        f'<div class="text-sm text-gray-400">{label} (28g)</div>'
        f'<div class="mt-1 text-lg font-semibold text-white">{int(band_counts[label])} '
        f'<span class="text-xs text-gray-400">({band_shares[label]*100:.0f}%)</span></div>'
        f'</div>'
    )

# Savings headline card (if available)
savings_headline_html = ""
if savings_headline:
    savings_headline_html = (
      f'<div class="bg-gray-800 border border-gray-700 rounded-lg p-4">'
      f'<div class="text-sm text-gray-400">Largest bulk savings</div>'
      f'<div class="mt-1 text-lg font-semibold␣
 ↪text-green-400">{savings_headline["pct"]:.0f}%</div>'
      f'<div class="mt-1 text-sm text-gray-300">{savings_headline["product"]}</
 ↪div>'
      f'<div class="mt-1 text-xs text-gray-400">'
      f'{savings_headline["small_label"]} @ ${savings_headline["small_ppoz"]:.
 ↪0f}/oz → '
      f'{savings_headline["large_label"]} @ ${savings_headline["large_ppoz"]:.
 ↪0f}/oz (28g)'
      f'</div>'
      f'</div>'
    )

# Top 3 list HTML
top3_html = ""
if not top3.empty:
    items = []
    for _, r in top3.iterrows():
        items.append(
            f'<li class="mb-1"><span class="text-white␣
 ↪font-semibold">{r["Product"]}</span>'
            f' <span class="text-gray-400">[{r["report_category"]}]</span>'
            f' - <span class="text-cyan-400 font-semibold">${r["price_per_oz"]:.
 ↪2f}/oz (28g)</span>'
            f' <span class="text-xs text-gray-400">({r["Best Size"]}, ${r["Best␣
 ↪Price"]:.2f})</span></li>'
        )
    top3_html = '<ul class="list-disc list-inside mt-1">' + "".join(items) + '</
 ↪ul>'

# Shake KPI (if no savings headline, show shake card; otherwise we already show␣
 ↪savings card)
shake_kpi_html = (
    f'<div class="bg-gray-800 border border-gray-700 rounded-lg p-4">'
    f'<div class="text-sm text-gray-400">Shake/Popcorn coverage</div>'
    f'<div class="mt-1 text-lg font-semibold text-white">{shake_share*100:.0f}%␣
 ↪of products</div>'
```

```
    + (f'<div class="mt-1 text-xs text-gray-400">cheapest: ${shake_min_ppoz:.
 ↪0f}/oz (28g)</div>' if shake_min_ppoz is not None else '')
    + '</div>'
)

# Verdict badge
verdict_html = (
  f'<div class="bg-emerald-900/30 border border-emerald-700 rounded-lg p-3␣
 ↪text-emerald-300 text-sm font-semibold">'
  f'Bottom line: {verdict_label}</div>'
)

# Build the FULL Bulk Savings table HTML (no condensing, wide columns)
savings_table_html = ""
if not savings_detail.empty:
    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'
    ]
    pretty = 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 $/oz (28g)'
    })
    savings_table_html = pretty.to_html(
        index=False,
        classes="w-full text-left my-6 text-base overflow-x-auto",
        formatters={
            'Small Price': lambda x: f'${x:,.2f}',
            'Small $/oz (28g)': lambda x: f'${x:,.2f}',
            'Large Price': lambda x: f'${x:,.2f}',
            'Large $/oz (28g)': lambda x: f'${x:,.2f}',
            'Savings %':  lambda x: f'<span class="font-semibold␣
 ↪text-green-400">{x:.0f}%</span>',
            'Δ $/oz (28g)':     lambda x: f'<span class="font-semibold␣
 ↪text-cyan-400">${x:,.2f}</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',
        },
        escape=False
    )

# Assemble HTML
html_output = f"""
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Medical Flower Price Report</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, -apple-system,␣
 ↪BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif; }} </style>
</head>
<body class="bg-gray-900 text-gray-200">
  <main class="max-w-5xl mx-auto p-6">

    <!-- Dispensary Header -->
    <header class="text-center mb-8">
      <h1 class="text-5xl font-extrabold text-white">Medical Flower Price␣
 ↪Report</h1>
      <p class="mt-2 text-lg text-gray-400">Value Analysis for {dispensary_info.
 ↪get('name','N/A')}</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_info.
 ↪get('name','N/A')}</h2>
        <p class="mt-1 text-gray-300">
          {dispensary_info.get('address','')}<br />
          {dispensary_info.get('city','')}, {dispensary_info.get('state','')}
        </p>
      </div>
      <div class="text-right space-y-1">
        <p class="text-gray-300"><strong>Rating:</strong> <span␣
↪class="text-cyan-400">{dispensary_info.get('rating',0):.1f} ␣
↪({int(dispensary_info.get('reviews_count',0))} reviews)</span></p>
        <p class="text-gray-300"><strong>Phone:</strong> <a href="tel:
↪{dispensary_info.get('phone_number','')}" class="text-cyan-400 hover:
↪underline">{dispensary_info.get('phone_number','N/A')}</a></p>
        <p class="text-gray-300"><strong>Menu:</strong> <a class="text-cyan-400␣
↪hover:underline" href="{dispensary_info.get('web_url','#')}"␣
↪target="_blank">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">${best_ppoz:.2f}/
↪oz</div>
          <div class="mt-1 text-xs text-gray-400">{best_name}</div>
          <div class="mt-1 text-xs text-gray-500">{best_size} • ${best_price:.
↪2f}</div>
        </div>
        <div class="bg-gray-800 border border-gray-700 rounded-lg p-4">
          <div class="text-sm text-gray-400">Typical price (per product, 28g␣
↪norm)</div>
          <div class="mt-1 text-2xl font-bold text-white">${overall_median:.0f}/
↪oz</div>
          <div class="mt-1 text-xs text-gray-400">IQR ${overall_p25:.0f}-
${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">{pct_leq60*100:.0f}%␣
↪  $60/oz</div>
          <div class="mt-1 text-xs text-gray-400">{pct_leq90*100:.0f}%   $90/
↪oz</div>
        </div>
        {(savings_headline_html if savings_headline_html else shake_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: {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>

      <h3 class="text-xl font-semibold text-white mt-6">Top 3 best-value␣
↪products (by $/oz)</h3>
      {top3_html if top3_html else '<p class="text-gray-400 mt-2">No products␣
↪available.</p>'}
    </section>

    <!-- Visuals (ONLY A & D) -->
    <section class="mb-6">
      <h2 class="text-3xl font-semibold text-cyan-400 border-b border-gray-700␣
↪pb-2">Price Distribution Visuals</h2>
      <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">A) Box Plot␣
↪(quartiles + whiskers)</h3>
        <img src="data:image/png;base64,{img_box}" alt="Box plot"␣
↪class="mx-auto rounded bg-white p-2 shadow" />
      </div>
      <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">D) 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" />
      </div>
    </section>

    <!-- Dynamic category leaderboards -->
    {build_all_category_leaderboards(price_df)}
```

```
    <!-- 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>
      {savings_table_html if savings_table_html else "<p class='text-gray-400␣
  ↪my-4'>No multi-size products with positive ounce-price savings found.</p>"}
    </section>

    <footer class="text-center text-sm text-gray-500 mt-10">
      Report generated on {datetime.now().strftime('%B %d, %Y')}.
    </footer>
  </main>
</body>
</html>
"""

display(HTML(html_output))
```

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

[6]:
```
# --- Cell 6: Full Price Bands (per product, 28g-normalized) - NO LIMITS ---

import pandas as pd
from IPython.display import HTML, display

# Use canonical "best-per-slug" rows if available (from Cell 5); otherwise,␣
  ↪derive it here
def _best_per_slug_local(df: pd.DataFrame) -> pd.DataFrame:
    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])
    return pd.DataFrame(rows).copy()

if 'best_per_slug' not in globals():
    best_per_slug = _best_per_slug_local(price_df)

# Define the price bands (same as in Cell 5)
band_labels = ["  $60", "$61-$90", "$91-$120", "$121-$200", ">$200"]
band_bins   = [0, 60, 90, 120, 200, float('inf')]

# Build a flat table of canonical products + band
bands_df = (
    ␣
  ↪best_per_slug[['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=band_bins, labels=band_labels, right=True, include_lowest=True
)

# Order rows by band then by price within band
bands_df['Price Band'] = pd.Categorical(bands_df['Price Band'],␣
 ↪categories=band_labels, ordered=True)
bands_df = bands_df.sort_values(['Price Band','Best $/Oz (28g)','Product']).
 ↪reset_index(drop=True)

# Summary counts (top)
summary_counts = (
    bands_df['Price Band']
    .value_counts()
    .reindex(band_labels, fill_value=0)
    .rename('Count')
    .to_frame()
)
summary_counts['Share'] = (summary_counts['Count'] / max(len(bands_df), 1)).
 ↪map(lambda x: f"{x*100:.0f}%")

display(HTML("<h2 style='margin:8px 0;'>Full Price Bands - per product␣
 ↪(28g-normalized)</h2>"))
display(HTML("<h3 style='margin:4px 0;'>Summary</h3>"))
display(HTML(summary_counts.to_html(classes="w-full text-left", escape=False)))

# Full, unlimited tables - one table per band (only if the band has rows)
for label in band_labels:
    sub = bands_df[bands_df['Price Band'].astype(str) == label]
    if sub.empty:
        continue
    display(HTML(f"<h3 style='margin-top:16px;'>{label}</h3>"))
    display(HTML(
        sub[['Product','Category','Best Size','Best Price','Best $/Oz (28g)']]
        .to_html(
            index=False,
            classes="w-full text-left",
            formatters={
                'Best Price': lambda x: f'${x:,.2f}',
                'Best $/Oz (28g)': lambda x: f'${x:,.2f}',
            },
            escape=False
        )
    ))
with open("output.html", "w", encoding="utf-8") as f:
    f.write(html_output)
display(HTML(html_output))
```

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

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

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

[7]:

```
# --- Cell 5: Final HTML Report (Dispensary Header, FULL Bulk Savings detail,␣
 ↪ONLY A & D plots,
#            dynamic per-category product leaderboards, 28g-normalized $/oz,␣
 ↪unlimited rows,
#            clearer "efficient sizes" thresholds progression, and a collapsible␣
 ↪FULL Price Bands section) ---

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

# ==========================
# Part 1 - Data prep
# ==========================

# Categories present, ordered by median $/oz (low → high)
cat_order = (
    price_df.groupby('report_category')['price_per_oz']
            .median()
            .sort_values()
            .index
            .tolist()
)

# Identify products (slugs) with multiple sizes for savings analysis
multi_size_slugs = price_df['slug'].value_counts()[lambda s: s > 1].index
multi_size_df    = price_df[price_df['slug'].isin(multi_size_slugs)].copy()

# Build rich small large savings detail per slug (FULL COLUMN SET; NO␣
 ↪CONDENSING)
savings_detail = pd.DataFrame()
if not multi_size_df.empty:
    # smallest vs largest by grams 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()]

    # explicit columns from each side
    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'
    )

    # compute savings metrics (per-oz) - already 28g-normalized from Cell 4
    savings_detail['savings_pct']   = (1 -␣
 ↪(savings_detail['price_per_oz_large'] /␣
 ↪savings_detail['price_per_oz_small'])) * 100
    savings_detail['delta_per_oz']  = (savings_detail['price_per_oz_small'] -␣
 ↪savings_detail['price_per_oz_large'])

    # keep only positive savings; sort descending (we'll still render a wide␣
 ↪table)
    savings_detail = (
        savings_detail[savings_detail['savings_pct'] > 0]
        .sort_values('savings_pct', ascending=False)
        .reset_index(drop=True)
    )

# Helper: parse standard package sizes (kept for diagnostics)
STANDARD_GRAM_SIZES = {1, 3.5, 7, 14, 28, 57}
def _is_standard(label):
    try:
        return float(label.replace('g','')) in STANDARD_GRAM_SIZES
    except Exception:
        return False

# =========================
# Part 2 - Canonical product rows + efficient sizes thresholds (for␣
 ↪leaderboards)
# =========================

EPS = 1e-6

def _efficient_sizes_df(group: pd.DataFrame) -> pd.DataFrame:
    """
    Keep the SMALLEST size that unlocks each distinct $/oz tier, then retain␣
 ↪only the
    thresholds where $/oz strictly improves as size increases.
    Result is a compact progression like: 7g → 14g → 28g (if each step reduces␣
 ↪$/oz).
    """
    g = group[['slug','name','size_label','weight_g','price','price_per_oz']].
 ↪dropna(subset=['weight_g','price_per_oz']).copy()

    # Bin $/oz to cents; keep the SMALLEST weight that achieves each $/oz tier
    g['ppoz_round'] = g['price_per_oz'].round(2)
    g = (
        g.sort_values(['ppoz_round','weight_g'])      # ensures smallest weight␣
 ↪per $/oz tier
          .groupby('ppoz_round', as_index=False)
          .head(1)
    )

    # Walk in ascending weight and retain only strict improvements in $/oz
    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:
        kdf = pd.DataFrame(kept).reset_index(drop=True)
        return kdf.drop(columns=['ppoz_round'])
    else:
        # Fallback: just the cheapest-per-oz row
        idx = group['price_per_oz'].idxmin()
        return group.loc[[idx],␣
 ↪['slug','name','size_label','weight_g','price','price_per_oz']]

def _best_per_slug(df: pd.DataFrame) -> pd.DataFrame:
    """Cheapest $/oz per product; break ties by larger weight."""
    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])
    return pd.DataFrame(rows).copy()

def _sizes_badge_from_df(sizedf: pd.DataFrame) -> str:
    def _key(lbl):
        try:
            return float(lbl.replace('g',''))
        except Exception:
            return 9e9
    labels = sorted(sizedf['size_label'].tolist(), key=_key)
    return " → ".join(labels)  # thresholds progression

# Efficient-size map and canonical (best-per-oz) row per slug - for␣
 ↪leaderboards & summary
eff_map = {}
```

```python
for slug, g in price_df.groupby('slug', sort=False):
    eff_map[slug] = _efficient_sizes_df(g)

best_per_slug = _best_per_slug(price_df)
best_per_slug['Efficient Sizes'] = best_per_slug['slug'].map(lambda s: _sizes_badge_from_df(eff_map[s]))

# merge savings pct for canonical rows (if available)
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

# =========================
# Part 3 - Executive Summary metrics (rich, from data below)
# =========================

# Overall best $/oz product (exclude Shake/Popcorn/Trim)
value_pool = best_per_slug[best_per_slug['report_category'] != 'Shake/Popcorn/Trim']
if value_pool.empty:
    # Fallback to all products if the filter removes everything
    value_pool = best_per_slug.copy()

best_row   = value_pool.loc[value_pool['price_per_oz'].idxmin()]
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 distribution (canonical per product)
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 medians and product counts
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-band coverage (canonical per product)
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 coverage & cheapest in Shake
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

# Multi-size savings headline (highest savings)
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 best-value products overall (canonical, excluding Shake/Popcorn/Trim)
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'}))

# Quick value verdict based on coverage   $60/oz
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)"

# =========================
# Part 4 - Visualization (ONLY A & D, deprecation-safe)
# =========================
sns.set_theme(style="whitegrid")

def _encode_fig(fig, dpi=150):
    buf = io.BytesIO()
    fig.savefig(buf, format='png', dpi=dpi, bbox_inches='tight')
    img = base64.b64encode(buf.getvalue()).decode('utf-8')
    plt.close(fig)
    return img

# A) Box plot (set hue to avoid "palette without hue" deprecation; drop legend)
fig1, ax1 = plt.subplots(figsize=(12, 5))
sns.boxplot(
    data=price_df,
    x="price_per_oz", y="report_category",
    order=cat_order, hue="report_category",
    palette="pastel", legend=False, ax=ax1,
    fliersize=5, linewidth=1.2
)
ax1.set_title("Price Distribution by Product Type", fontsize=18, fontweight='bold', pad=12)
ax1.set_xlabel("Price per Ounce ($, 28g norm)", fontsize=13)   # <- label clarified
ax1.set_ylabel("")
img_box = _encode_fig(fig1)

# D) ECDF overlay - let seaborn manage the legend
fig4, ax4 = plt.subplots(figsize=(12, 5))
sns.ecdfplot(
    data=price_df,
    x="price_per_oz", hue="report_category",
    hue_order=cat_order, palette="Set2",
    legend=True, ax=ax4
)
ax4.set_title("Cumulative Distribution of Price per Ounce", fontsize=18, fontweight='bold', pad=12)
ax4.set_xlabel("Price per Ounce ($, 28g norm)")                # <- label clarified
ax4.set_ylabel("Cumulative fraction")
```

```
img_ecdf = _encode_fig(fig4)

# =========================
# Part 5 - HTML helpers (leaderboards + formatting)
# =========================
EMOJIS = {'Bulk Value':' ', 'Pre-Pack Specialty':' ', 'Shake/Popcorn/Trim':' '}

def product_leaderboard(df, title, top_n=None):
    """One row per product (slug): Best $/oz + Best Size + Efficient Sizes +␣
 ↪Max Savings vs Smallest.
       Set top_n=None (default) to show ALL rows (no truncation)."""
    if df.empty:
        return ""
    # Build best-per-slug within THIS subset
    rows = []
    eff_sizes_col = []
    sav_col = []
    for slug, g in df.groupby('slug', sort=False):
        g2 = g.sort_values(['price_per_oz','weight_g'], ascending=[True, False])
        rows.append(g2.iloc[0])
        eff_sizes_col.append(_sizes_badge_from_df(eff_map[slug]))
        if not savings_detail.empty:
            spct = savings_detail.loc[savings_detail['slug']==slug,␣
 ↪'savings_pct']
            sav_col.append(spct.iloc[0] if not spct.empty else np.nan)
        else:
            sav_col.append(np.nan)

    best = pd.DataFrame(rows).copy()
    best['Efficient Sizes'] = eff_sizes_col
    best['savings_pct']     = sav_col

    out = (
        best[['name','size_label','price','price_per_oz','Efficient␣
 ↪Sizes','savings_pct']]
        .rename(columns={
            'name':'Product','size_label':'Best Size',
            'price':'Best Price','price_per_oz':'Best $/Oz (28g)',
            'savings_pct':'Max Savings vs Smallest'
        })
        .sort_values(['Best $/Oz (28g)','Product'], ascending=[True, True])
    )
    # Only cap when top_n is a positive int
    if isinstance(top_n, int) and top_n > 0:
        out = out.head(top_n)

    return (
        f'<h2 class="text-3xl font-semibold text-cyan-400 border-b␣
 ↪border-gray-700 pb-2 mt-10">{title}</h2>'
        + out.to_html(
            index=False,
            classes="w-full text-left my-6 text-base",
            formatters={
                'Best Price': lambda x: f'${x:,.2f}',
                'Best $/Oz (28g)': lambda x: f'<span class="font-semibold␣
 ↪text-cyan-400">${x:,.2f}</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
        )
    )

def build_all_category_leaderboards(df):
    sections = []
    for cat in cat_order:
        sub = df[df['report_category'] == cat]
        if not sub.empty:
            sections.append(product_leaderboard(sub, f"{EMOJIS.get(cat,' ')}␣
 ↪{cat}", top_n=None))  # unlimited
    return "\n".join(sections)

# =========================
# Part 5.1 - 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>
    """
)

# ========================
# Part 6 - HTML (Dispensary Header + Executive Summary + Visuals + Leaderboards + FULL Savings + Collapsible FULL Price Bands)
# ========================
# Category medians chips
cat_kpi_html = ""
for c in cat_stats:
    cat_kpi_html += (
        f'<div class="bg-gray-800 border border-gray-700 rounded-lg p-3">'
        f'<div class="text-sm text-gray-400">{c["cat"]}</div>'
        f'<div class="mt-1 text-lg font-semibold text-white">${c["median"]:.0f}/oz <span class="text-xs text-gray-400">(median, 28g)</span></div>'
        f'<div class="mt-1 text-xs text-gray-400">min ${c["min"]:.0f} • {c["n_products"]} products</div>'
        f'</div>'
    )

# Price bands chips (compact summary already shown previously)
bands_html = ""
for label in ["  $60", "$61-$90", "$91-$120", "$121-$200", ">$200"]:
    bands_html += (
        f'<div class="bg-gray-800 border border-gray-700 rounded-lg p-3">'
        f'<div class="text-sm text-gray-400">{label} (28g)</div>'
        f'<div class="mt-1 text-lg font-semibold text-white">{int(band_counts[label])} '
        f'<span class="text-xs text-gray-400">({band_shares[label]*100:.0f}%)</span></div>'
        f'</div>'
    )

# Savings headline card (if available)
savings_headline_html = ""
if savings_headline:
    savings_headline_html = (
      f'<div class="bg-gray-800 border border-gray-700 rounded-lg p-4">'
      f'<div class="text-sm text-gray-400">Largest bulk savings</div>'
      f'<div class="mt-1 text-lg font-semibold text-green-400">{savings_headline["pct"]:.0f}%</div>'
      f'<div class="mt-1 text-sm text-gray-300">{savings_headline["product"]}</div>'
      f'<div class="mt-1 text-xs text-gray-400">'
      f'{savings_headline["small_label"]} @ ${savings_headline["small_ppoz"]:.0f}/oz → '
      f'{savings_headline["large_label"]} @ ${savings_headline["large_ppoz"]:.0f}/oz (28g)'
      f'</div>'
      f'</div>'
    )

# Top 3 list HTML
top3_html = ""
if not top3.empty:
    items = []
    for _, r in top3.iterrows():
        items.append(
            f'<li class="mb-1"><span class="text-white font-semibold">{r["Product"]}</span>'
            f' <span class="text-gray-400">[{r["report_category"]}]</span>'
            f' - <span class="text-cyan-400 font-semibold">${r["price_per_oz"]:.2f}/oz (28g)</span>'
            f' <span class="text-xs text-gray-400">({r["Best Size"]}, ${r["Best Price"]:.2f})</span></li>'
        )
    top3_html = '<ul class="list-disc list-inside mt-1">' + "".join(items) + '</ul>'

# Shake KPI (if no savings headline, show shake card; otherwise we already show savings card)
shake_kpi_html = (
    f'<div class="bg-gray-800 border border-gray-700 rounded-lg p-4">'
    f'<div class="text-sm text-gray-400">Shake/Popcorn coverage</div>'
    f'<div class="mt-1 text-lg font-semibold text-white">{shake_share*100:.0f}% of products</div>'
    + (f'<div class="mt-1 text-xs text-gray-400">cheapest: ${shake_min_ppoz:.0f}/oz (28g)</div>' if shake_min_ppoz is not None else '')
    + '</div>'
)

# Verdict badge
verdict_html = (
  f'<div class="bg-emerald-900/30 border border-emerald-700 rounded-lg p-3 text-emerald-300 text-sm font-semibold">'
  f'Bottom line: {verdict_label}</div>'
)

# Build the FULL Bulk Savings table HTML (no condensing, wide columns)
savings_table_html = ""
if not savings_detail.empty:
    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'
    ]
    pretty = 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 $/oz (28g)'
    })
    savings_table_html = pretty.to_html(
        index=False,
        classes="w-full text-left my-6 text-base overflow-x-auto",
        formatters={
            'Small Price': lambda x: f'${x:,.2f}',
            'Small $/oz (28g)': lambda x: f'${x:,.2f}',
            'Large Price': lambda x: f'${x:,.2f}',
            'Large $/oz (28g)': lambda x: f'${x:,.2f}',
            'Savings %':  lambda x: f'<span class="font-semibold␣
 ↪text-green-400">{x:.0f}%</span>',
            'Δ $/oz (28g)':     lambda x: f'<span class="font-semibold␣
 ↪text-cyan-400">${x:,.2f}</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',
        },
        escape=False
    )

# Assemble HTML
html_output = f"""
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Medical Flower Price Report</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, -apple-system,␣
 ↪BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif; }} </style>
</head>
<body class="bg-gray-900 text-gray-200">
  <main class="max-w-5xl mx-auto p-6">

    <!-- Dispensary Header -->
    <header class="text-center mb-8">
      <h1 class="text-5xl font-extrabold text-white">Medical Flower Price␣
 ↪Report</h1>
      <p class="mt-2 text-lg text-gray-400">Value Analysis for {dispensary_info.
 ↪get('name','N/A')}</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_info.
 ↪get('name','N/A')}</h2>
        <p class="mt-1 text-gray-300">
          {dispensary_info.get('address','')}<br />
          {dispensary_info.get('city','')}, {dispensary_info.get('state','')}
        </p>
      </div>
      <div class="text-right space-y-1">
        <p class="text-gray-300"><strong>Rating:</strong> <span␣
 ↪class="text-cyan-400">{dispensary_info.get('rating',0):.1f} ␣
 ↪({int(dispensary_info.get('reviews_count',0))} reviews)</span></p>
        <p class="text-gray-300"><strong>Phone:</strong> <a href="tel:
 ↪{dispensary_info.get('phone_number','')}" class="text-cyan-400 hover:
 ↪underline">{dispensary_info.get('phone_number','N/A')}</a></p>
        <p class="text-gray-300"><strong>Menu:</strong> <a class="text-cyan-400␣
 ↪hover:underline" href="{dispensary_info.get('web_url','#')}"␣
 ↪target="_blank">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">${best_ppoz:.2f}/
 ↪oz</div>
          <div class="mt-1 text-xs text-gray-400">{best_name}</div>
          <div class="mt-1 text-xs text-gray-500">{best_size} • ${best_price:.
 ↪2f}</div>
        </div>
        <div class="bg-gray-800 border border-gray-700 rounded-lg p-4">
          <div class="text-sm text-gray-400">Typical price (per product, 28g␣
 ↪norm)</div>
          <div class="mt-1 text-2xl font-bold text-white">${overall_median:.0f}/
 ↪oz</div>
          <div class="mt-1 text-xs text-gray-400">IQR ${overall_p25:.0f}-
${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">{pct_leq60*100:.0f}%␣
 ↪  $60/oz</div>
          <div class="mt-1 text-xs text-gray-400">{pct_leq90*100:.0f}%   $90/
 ↪oz</div>
        </div>
        {(savings_headline_html if savings_headline_html else shake_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: {verdict_label}
        </div>
      </div>

      <h3 class="text-xl font-semibold text-white mt-6">Category medians &amp; 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 if top3_html else '<p class="text-gray-400 mt-2">No products available.</p>'}
    </section>

    <!-- Visuals (ONLY A & D) -->
    <section class="mb-6">
      <h2 class="text-3xl font-semibold text-cyan-400 border-b border-gray-700 pb-2">Price Distribution Visuals</h2>
      <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">A) Box Plot (quartiles + whiskers)</h3>
        <img src="data:image/png;base64,{img_box}" alt="Box plot" class="mx-auto rounded bg-white p-2 shadow" />
      </div>
      <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">D) 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" />
      </div>
    </section>

    <!-- Dynamic category leaderboards -->
    {build_all_category_leaderboards(price_df)}

    <!-- 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>
      {savings_table_html if savings_table_html else "<p class='text-gray-400␣
 ↪my-4'>No multi-size products with positive ounce-price savings found.</p>"}
    </section>

    <footer class="text-center text-sm text-gray-500 mt-10">
      Report generated on {datetime.now().strftime('%B %d, %Y')}.
    </footer>
  </main>
</body>
</html>
"""

with open("output.html", "w", encoding="utf-8") as f:
    f.write(html_output)
display(HTML(html_output))
```

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