Data Collection:

Install and Import

In [ ]:

```
pip install fuzzywuzzy
```

```
Requirement already satisfied: fuzzywuzzy in /usr/local/lib/python3.10/dist-packages (0.18.0)
```

In [ ]:

```
import math
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
```

```
/usr/local/lib/python3.10/dist-packages/fuzzywuzzy/fuzz.py:11: UserWarning: Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warn
ing
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning')
```

Load Data

In [ ]:

```
# Path to main data .zip file
primary_data_file_path = 'data/primary_data.zip'

# Path to secondary data .zip file
secondary_data_file_path = 'data/secondary_data.zip'
```

In [ ]:

```
cdph_df = pd.read_csv(primary_data_file_path)
cdph_df.head()
```

Out[ ]:

	Product Id	Company	Brand	Product Name	Variant	Product Discontinued Date	Product Submitted Date	Ingredient Name	Function	Unit of Measure	Concentration	Ingredient Submitted Date	Ingredient Removed Date	UPC	Body Area	
0	43485	Anastasia Beverly Hills, LLC	Anastasia Beverly HIlls	Perfect Brow Pencil (Granite shade_	NaN	NaN	7/15/2016	Titanium dioxide (CI 77891) 13463-67-7 / 1317-...	NaN	mg/g	46.81	7/15/2016	NaN	Not Available	Other (Specify):	Eyeliner
1	18358	Nail Alliance - Entity	Entity Nudite	Cool Pink Nail Sculpting Powder	NaN	NaN	6/24/2019	Titanium dioxide (CI 77891) 13463-67-7 / 1317-...	NaN	mg/g	5	6/24/2019	NaN	Not Available	Nails	Artif an
2	23202	GAP INC.	Gap Outlet	Light Pink, lip gloss (Lip trio)	Light Pink	1/1/2018	11/7/2014	Titanium dioxide (CI 77891) 13463-67-7 / 1317-...	NaN	NaN	NaN	11/7/2014	NaN	Not Available	Lips	Lip Gl
3	38662	Xtreme Color, Inc.	Hard Candy	Fierce Effects-Shadow Duo	Black-Sprinkle	NaN	9/21/2015	Titanium dioxide (CI 77891) 13463-67-7 / 1317-...	NaN	mg/g	31	9/21/2015	NaN	Not Available	Eye Area	Ey
4	38666	Xtreme Color, Inc.	Hard Candy	Fierce Effects-Shadow Duo	Brown-Sprinkle	NaN	9/21/2015	Titanium dioxide (CI 77891) 13463-67-7 / 1317-...	NaN	mg/g	125	9/21/2015	NaN	Not Available	Eye Area	Ey

In [ ]:

```
cdph_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 718660 entries, 0 to 718659
Data columns (total 18 columns):
 #   Column                     Non-Null Count   Dtype 
---  ------                     --------------   ----- 
 0   Product Id                 718660 non-null  int64 
 1   Company                    718660 non-null  object
 2   Brand                      718424 non-null  object
 3   Product Name               718660 non-null  object
 4   Variant                    200654 non-null  object
 5   Product Discontinued Date  38504 non-null   object
 6   Product Submitted Date     718660 non-null  object
 7   Ingredient Name            718660 non-null  object
 8   Function                   598645 non-null  object
 9   Unit of Measure            248781 non-null  object
 10  Concentration              250451 non-null  object
 11  Ingredient Submitted Date  718660 non-null  object
 12  Ingredient Removed Date    8303 non-null    object
 13  UPC                        718547 non-null  object
 14  Body Area                  711645 non-null  object
 15  Product Category           718660 non-null  object
 16  Product Form               707369 non-null  object
 17  Intended Market            718639 non-null  object
dtypes: int64(1), object(17)
memory usage: 98.7+ MB
```

In [ ]:

```
beauty_df = pd.read_csv(secondary_data_file_path)
beauty_df.head()
```

Out[ ]:

	Product_Name	Brand	Category	Usage_Frequency	Price_USD	Rating	Number_of_Reviews	Product_Size	Skin_Type	Gender_Target	Packaging_Type	Main_Ingredient	Cr
0	Ultra Face Mask	Drunk Elephant	Blush	Weekly	67.85	1.4	686	30ml	Sensitive	Female	Tube	Retinol	
1	Ultra Lipstick	Laura Mercier	Makeup Remover	Occasional	116.43	4.2	5483	250ml	Dry	Unisex	Bottle	Shea Butter	
2	Ultra Serum	Natasha Denona	Highlighter	Daily	90.84	1.6	5039	100ml	Sensitive	Male	Compact	Aloe Vera	
3	Divine Serum	Ilia Beauty	Face Mask	Occasional	55.17	3.2	6202	250ml	Normal	Male	Tube	Glycerin	
4	Super Foundation	Charlotte Tilbury	Highlighter	Occasional	140.56	1.7	297	100ml	Oily	Female	Compact	Glycerin	

In [ ]:

```
beauty_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Product_Name        15000 non-null  object 
 1   Brand               15000 non-null  object 
 2   Category            15000 non-null  object 
 3   Usage_Frequency     15000 non-null  object 
 4   Price_USD           15000 non-null  float64
 5   Rating              15000 non-null  float64
 6   Number_of_Reviews   15000 non-null  int64  
 7   Product_Size        15000 non-null  object 
 8   Skin_Type           15000 non-null  object 
 9   Gender_Target       15000 non-null  object 
 10  Packaging_Type      15000 non-null  object 
 11  Main_Ingredient     15000 non-null  object 
 12  Cruelty_Free        15000 non-null  bool   
 13  Country_of_Origin   15000 non-null  object 
dtypes: bool(1), float64(2), int64(1), object(10)
memory usage: 1.5+ MB
```

Data Preparation:

In [ ]:
```
cdph_df_copy = cdph_df.copy()
beauty_df_copy = beauty_df.copy()
```

In [ ]:
```
cdph_df_copy['Brand'].unique()
```

Out[ ]:
```
array([' Anastasia Beverly HIlls', ' Entity Nudite', ' Gap Outlet', ...,
       'Zuri Brazil Collection', 'Zuri Flawless', 'Zuri Naturally Sheer'],
      dtype=object)
```

In [ ]:
```
beauty_df_copy['Brand'].unique()
```

Out[ ]:
```
array(['Drunk Elephant', 'Laura Mercier', 'Natasha Denona', 'Ilia Beauty',
       'Charlotte Tilbury', 'Danessa Myricks', 'Bourjois', 'IT Cosmetics',
       'Fenty Beauty', 'Sisley', 'Juvia’s Place', 'NARS', 'ColourPop',
       'Huda Beauty', 'Tatcha', 'Kiehl’s', 'Tarte', 'Glossier',
       'Make Up For Ever', 'Anastasia Beverly Hills', 'E.l.f.',
       'Hourglass', 'Pat McGrath Labs', 'Too Faced', 'Perricone MD',
       'RMS Beauty', 'Urban Decay', 'Rare Beauty', 'Becca', 'Patrick Ta',
       'Shiseido', 'Kylie Cosmetics', 'Bite Beauty', 'Yves Saint Laurent',
       'Bobby Brown', 'Farsali', 'Morphe', 'Milk Makeup', 'Clinique',
       'KVD Beauty'], dtype=object)
```

In [ ]:
```
# Clean up the spaces at the front and back of the values in the Brand column
cleaned_cdph_df = cdph_df_copy.apply(lambda col: col.apply(lambda x: x.strip().upper() if isinstance(x, str) else x))
cleaned_beauty_df = beauty_df_copy.apply(lambda col: col.apply(lambda x: x.strip().upper() if isinstance(x, str) else x))
```

In [ ]:
```
cleaned_cdph_df.head(1)
```

Out[ ]:

	Product Id	Company	Brand	Product Name	Variant	Product Discontinued Date	Product Submitted Date	Ingredient Name	Function	Unit of Measure	Concentration	Ingredient Submitted Date	Ingredient Removed Date	UPC	Body Area	
0	43485	ANASTASIA BEVERLY HILLS, LLC	ANASTASIA BEVERLY HILLS	PERFECT BROW PENCIL (GRANITE SHADE_	NaN	NaN	7/15/2016	TITANIUM DIOXIDE (CI 77891) 13463-67-7 / 1317-...	NaN	MG/G	46.81	7/15/2016	NaN	NOT AVAILABLE	OTHER (SPECIFY):	E

Data Manipulation:

Aggregate Dataframes

In [ ]:

```
def aggregate_primary_data(df):
    # Step 1: Create a fresh copy of the dataframe
    df_copy = df.copy()

    # Step 2: Create binary columns for 'Discontinued Date' and 'Ingredient Removed Date'
    df_copy['Discontinued_Binary'] = df_copy['Product Discontinued Date'].notna().astype(int)
    df_copy['Ingredient_Removed_Binary'] = df_copy['Ingredient Removed Date'].notna().astype(int)

    # Convert 'Concentration' column to numeric, coerce errors (non-numeric values to NaN)
    df_copy['Concentration'] = pd.to_numeric(df_copy['Concentration'], errors='coerce')
    # df_copy['Concentration'] = df_copy['Concentration'].fillna(0)

    # Step 3: Group by 'Brand' and aggregate the required information
    df_grouped = df_copy.groupby('Brand').agg(
        Products_Reported_09THRU24=('Product Name', 'size'),                    # Count the number of products per brand
        Discontinued_Count_09THRU24=('Discontinued_Binary', 'sum'),          # Binary count for 'Discontinued Date'
        Ingredient_Removed_Count_09THRU24=('Ingredient_Removed_Binary', 'sum'),  # Binary count for 'Ingredient Removed Date'
        Avg_Concentration_09THRU24=('Concentration', 'mean'),                        # Average concentration
    ).reset_index()

    # Step 4: Return the final processed DataFrame
    return df_grouped

# Example usage: Preview initial rows of prepared data
agg_cpdh_df = aggregate_primary_data(cleaned_cdph_df)

print("Initial Data Preview:")
brand_rows = agg_cpdh_df.iloc[3181:3184]
print(brand_rows)
```

```
Initial Data Preview:
                    Brand  Products_Reported_09THRU24  \
3181            TOO FACED                         333   
3182  TOO FACED COSMETICS                        1476   
3183             TOP CARE                          43   

      Discontinued_Count_09THRU24  Ingredient_Removed_Count_09THRU24  \
3181                            0                                  0   
3182                          329                                  1   
3183                            0                                  0   

      Avg_Concentration_09THRU24  
3181                         NaN  
3182                      1.1956  
3183                         NaN  
```

In [ ]:

```python
def aggregate_secondary_data(df):
    df_copy = df.copy()

    # Group by 'Brand' and remove 'Product Name'
    df_grouped = df_copy.groupby('Brand').agg(
        Top_Products_Count_2024=('Product_Name', 'size'),                    # Step 3: Product Count
        Avg_Price_USD_2024=('Price_USD', 'mean'),                       # Step 4: Average Price
        Avg_Rating_2024=('Rating', 'mean'),                             # Step 4: Average Rating
    ).reset_index()

    return df_grouped

# Example usage:
agg_beauty_df = aggregate_secondary_data(cleaned_beauty_df)
agg_beauty_df['Brand'].unique()
```

Out[ ]:

```
array(['ANASTASIA BEVERLY HILLS', 'BECCA', 'BITE BEAUTY', 'BOBBY BROWN',
       'BOURJOIS', 'CHARLOTTE TILBURY', 'CLINIQUE', 'COLOURPOP',
       'DANESSA MYRICKS', 'DRUNK ELEPHANT', 'E.L.F.', 'FARSALI',
       'FENTY BEAUTY', 'GLOSSIER', 'HOURGLASS', 'HUDA BEAUTY',
       'ILIA BEAUTY', 'IT COSMETICS', 'JUVIA’S PLACE', 'KIEHL’S',
       'KVD BEAUTY', 'KYLIE COSMETICS', 'LAURA MERCIER',
       'MAKE UP FOR EVER', 'MILK MAKEUP', 'MORPHE', 'NARS',
       'NATASHA DENONA', 'PAT MCGRATH LABS', 'PATRICK TA', 'PERRICONE MD',
       'RARE BEAUTY', 'RMS BEAUTY', 'SHISEIDO', 'SISLEY', 'TARTE',
       'TATCHA', 'TOO FACED', 'URBAN DECAY', 'YVES SAINT LAURENT'],
      dtype=object)
```

Merge Aggregated Dataframes

In [ ]:

```python
agg_prod = agg_beauty_df.copy()
agg_cdph = agg_cpdh_df.copy()
```

In [ ]:

```python
# Step 1: Define a function to create a mapping of fuzzy matches
def create_brand_mapping(cdph_brands, prod_brands):
    mapping = {}
    for brand in cdph_brands:
        # Convert brand to string to avoid TypeError
        brand = str(brand)
        # Get the best match for each brand
        match, score = process.extractOne(brand, prod_brands)

        if score >= 90:  # 90% confidence threshold
            mapping[brand] = match
    return mapping
```

In [ ]:

```
# Step 2: Get unique brands from both dataframes and convert to strings
agg_cdph_unique_brands = agg_cdph['Brand'].astype(str).unique()
agg_prod_unique_brands = agg_prod['Brand'].astype(str).unique()

# Step 3: Create the mapping of fuzzy matches
agg_brand_mapping = create_brand_mapping(agg_cdph_unique_brands, agg_prod_unique_brands)

# Step 4: Map the original brands in the cdph DataFrame to the matched brands
agg_cdph['Brand'] = agg_cdph['Brand'].astype(str).map(agg_brand_mapping)

# Step 5: Merge the dataframes on 'Brand'
merged_agg_df = pd.merge(agg_cdph, agg_prod, on='Brand', how='inner', suffixes=('_cdph', '_prod'))

# Display unique brands in the merged DataFrame
print(merged_agg_df['Brand'].unique())
```

```
['LAURA MERCIER' 'ANASTASIA BEVERLY HILLS' 'URBAN DECAY' 'IT COSMETICS'
 'BITE BEAUTY' 'BOBBY BROWN' 'CHARLOTTE TILBURY' 'CLINIQUE' 'SHISEIDO'
 'DRUNK ELEPHANT' 'E.L.F.' 'FENTY BEAUTY' 'GLOSSIER' 'SISLEY' 'HOURGLASS'
 'HUDA BEAUTY' 'ILIA BEAUTY' 'KIEHL’S' 'KVD BEAUTY' 'MAKE UP FOR EVER'
 'MILK MAKEUP' 'MORPHE' 'NARS' 'NATASHA DENONA' 'PERRICONE MD' 'COLOURPOP'
 'RARE BEAUTY' 'RMS BEAUTY' 'TARTE' 'TATCHA' 'TOO FACED' 'KYLIE COSMETICS']
```

In [ ]:

```
merged_agg_df.tail(10)
```

Out[ ]:

	Brand	Products_Reported_09THRU24	Discontinued_Count_09THRU24	Ingredient_Removed_Count_09THRU24	Avg_Concentration_09THRU24	Top_Products_Count_2024	Av
52	SISLEY	2040	21	97	69.961014	392	
53	TARTE	1	0	0	0.000000	361	
54	TARTE	1	0	0	0.320000	361	
55	TARTE	3432	3	8	47.973551	361	
56	TARTE	365	0	0	74.633394	361	
57	TATCHA	1262	16	0	1.707875	374	
58	TOO FACED	333	0	0	NaN	371	
59	TOO FACED	1476	329	1	1.195600	371	
60	URBAN DECAY	1	0	0	NaN	356	
61	KYLIE COSMETICS	1	0	0	NaN	370	

In [ ]:

```
merged_agg_df.info()
```

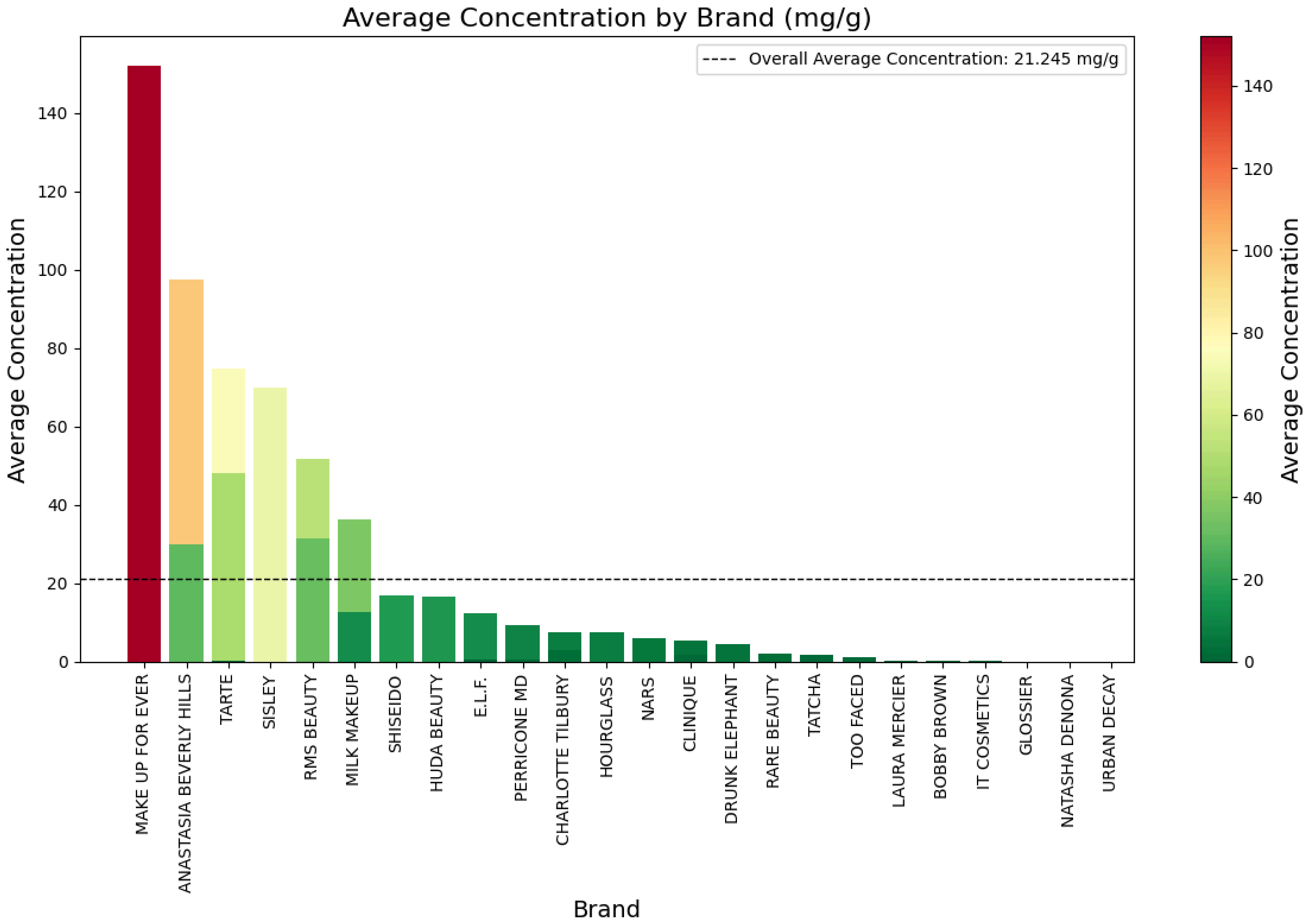
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62 entries, 0 to 61
Data columns (total 8 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Brand                            62 non-null     object 
 1   Products_Reported_09THRU24       62 non-null     int64  
 2   Discontinued_Count_09THRU24      62 non-null     int64  
 3   Ingredient_Removed_Count_09THRU24  62 non-null     int64  
 4   Avg_Concentration_09THRU24       33 non-null     float64
 5   Top_Products_Count_2024          62 non-null     int64  
 6   Avg_Price_USD_2024               62 non-null     float64
 7   Avg_Rating_2024                  62 non-null     float64
dtypes: float64(3), int64(4), object(1)
memory usage: 4.0+ KB
```

Data Visualization:

Average Concentration by Brand

In [ ]:

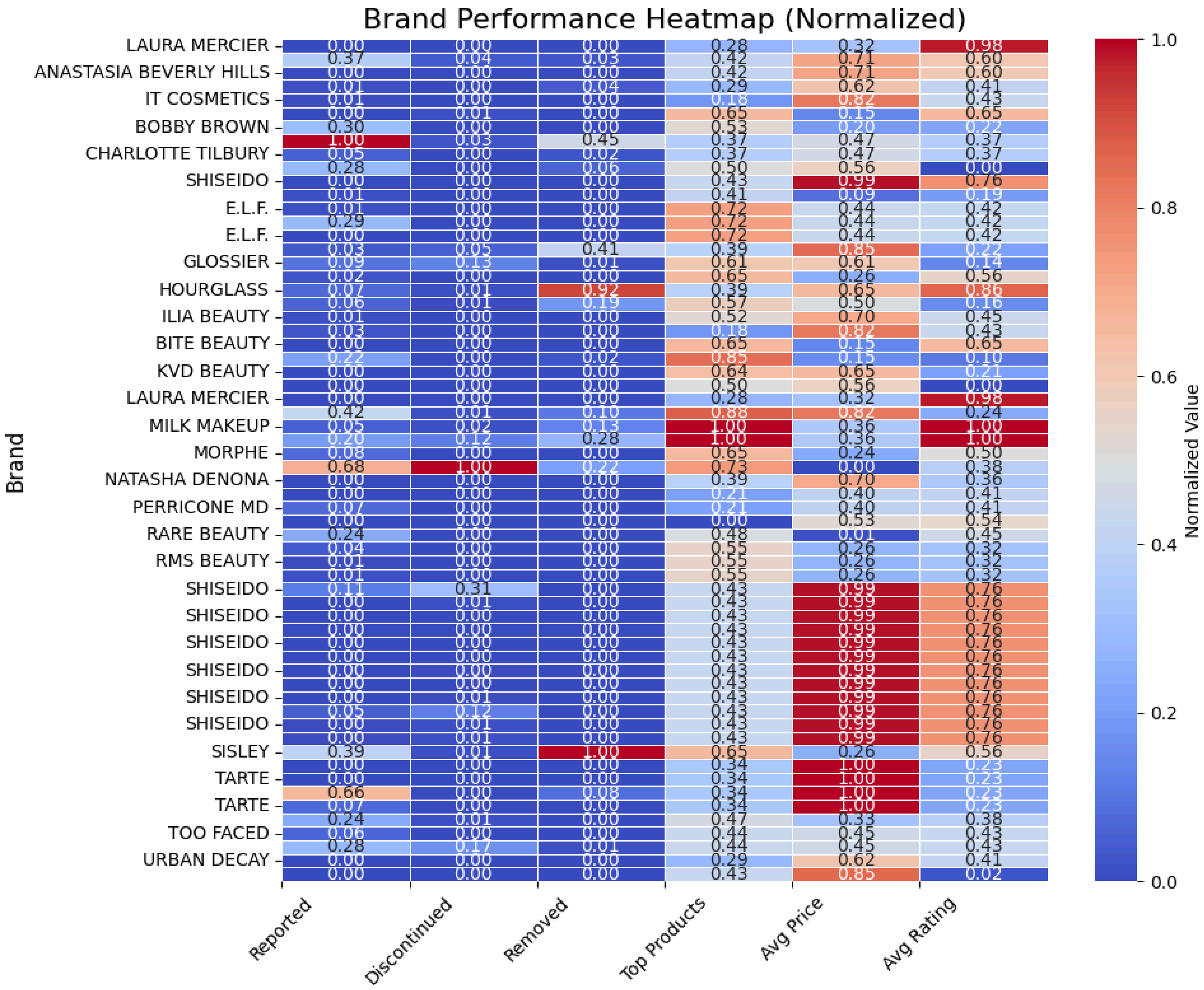
```
def plot_average_concentration_by_brand(df):
    """
    Plot the average concentration by brand with a color gradient and overall average line.

    Args:
        df (pd.DataFrame): DataFrame containing relevant data with 'Brand' and 'Avg_Concentration_09THRU24' columns.
    """
    # Sort the DataFrame by concentration for better visualization
    average_concentration_by_brand = df.copy()
    average_concentration_by_brand.sort_values(by='Avg_Concentration_09THRU24', ascending=False, inplace=True)

    # Normalize concentration values for gradient coloring
    norm = plt.Normalize(average_concentration_by_brand['Avg_Concentration_09THRU24'].min(),
                         average_concentration_by_brand['Avg_Concentration_09THRU24'].max())

    # Create a colormap
    cmap = plt.get_cmap('RdYlGn_r')  # Reverse RdYlGn to have red for highest and green for lowest

    # Map the concentrations to colors
    colors = cmap(norm(average_concentration_by_brand['Avg_Concentration_09THRU24'].values))

    # Plotting
    plt.figure(figsize=(12, 8))
    bars = plt.bar(average_concentration_by_brand['Brand'],
                   average_concentration_by_brand['Avg_Concentration_09THRU24'],
                   color=colors)

    overall_avg_concentration = average_concentration_by_brand['Avg_Concentration_09THRU24'].mean()
    plt.axhline(overall_avg_concentration, color='black', linestyle='--', linewidth=1,
                label='Overall Average Concentration: ' + str(round(overall_avg_concentration, 3)) + ' mg/g')

    # Adding titles and labels
    plt.title('Average Concentration by Brand (mg/g)', fontsize=16)
    plt.xlabel('Brand', fontsize=14)
    plt.ylabel('Average Concentration', fontsize=14)
    plt.xticks(rotation=90, fontsize=10)  # Rotate brand names for better readability

    # Create the colorbar
    sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
    sm.set_array([])

    # Specify the `ax` parameter to avoid the warning
    cbar = plt.colorbar(sm, ax=plt.gca())
    cbar.set_label('Average Concentration', fontsize=14)

    # Add legend for overall average concentration
    plt.legend()

    # Display the plot
    plt.tight_layout()  # Adjust layout to make room for the rotated x labels
    plt.show()

plot_average_concentration_by_brand(merged_agg_df)
```



Brand Performance Heatmap

```
In [ ]: # Function to normalize data using Min-Max scaling
        def min_max_normalize(df):
            """
            Normalize the DataFrame using Min-Max normalization.

            Min-Max normalization transforms features to a common scale, specifically to a range between 0 and 1.
```

```
    This is achieved using the formula:
    (df - df.min()) / (df.max() - df.min())

    Explanation of the formula:
    - df: The original DataFrame.
    - df.min(): The minimum value of each feature, allowing us to shift the data to start from 0.
    - df.max(): The maximum value of each feature, which helps scale the data such that the maximum value becomes 1.

    By applying this normalization, we ensure that all features are on the same scale, which is particularly important
    when performing distance-based calculations, such as in clustering or when using algorithms sensitive to feature scales,
    such as gradient descent. Normalized data can lead to better model performance and convergence.

    Parameters:
    df (pd.DataFrame): The DataFrame to normalize.

    Returns:
    pd.DataFrame: The normalized DataFrame.
    """
    return (df - df.min()) / (df.max() - df.min())

# Function to create a heatmap for brand performance
def brand_performance_heatmap(dataframe):
    """
    Create a heatmap to visualize brand performance across different metrics.

    This function generates a heatmap where each row represents a brand
    and each column represents a performance metric.

    Parameters:
    dataframe (pd.DataFrame): The DataFrame containing brand performance data.
    """

    # Select relevant columns for the heatmap
    metrics = [
        'Products_Reported_09THRU24',
        'Discontinued_Count_09THRU24',
        'Ingredient_Removed_Count_09THRU24',
        'Top_Products_Count_2024',
        'Avg_Price_USD_2024',
        'Avg_Rating_2024'
    ]

    # Create a new DataFrame with Brand as index and metrics as columns
    heatmap_data = dataframe.set_index('Brand')[metrics]

    # Normalize the data using Min-Max scaling
    normalized_data = min_max_normalize(heatmap_data)

    # Set up the matplotlib figure
    plt.figure(figsize=(10, 8))

    # Create the heatmap with normalized data
    ax = sns.heatmap(normalized_data, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5, cbar_kws={"label": "Normalized Value"})

    # Set the title and labels
    plt.title('Brand Performance Heatmap (Normalized)', fontsize=16)
    # Update the X-axis labels to be more concise
    plt.xticks(ticks=range(len(metrics)), labels=['Reported', 'Discontinued', 'Removed', 'Top Products', 'Avg Price', 'Avg Rating'], rotation=45, ha='center')

    plt.ylabel('Brand', fontsize=12)

    # Center the annotations
    for text in ax.texts:
        text.set_verticalalignment('center')  # Center vertically
        text.set_horizontalalignment('center')  # Center horizontally

    # Show the heatmap
    plt.tight_layout()
    plt.show()

brand_performance_heatmap(merged_agg_df)
```

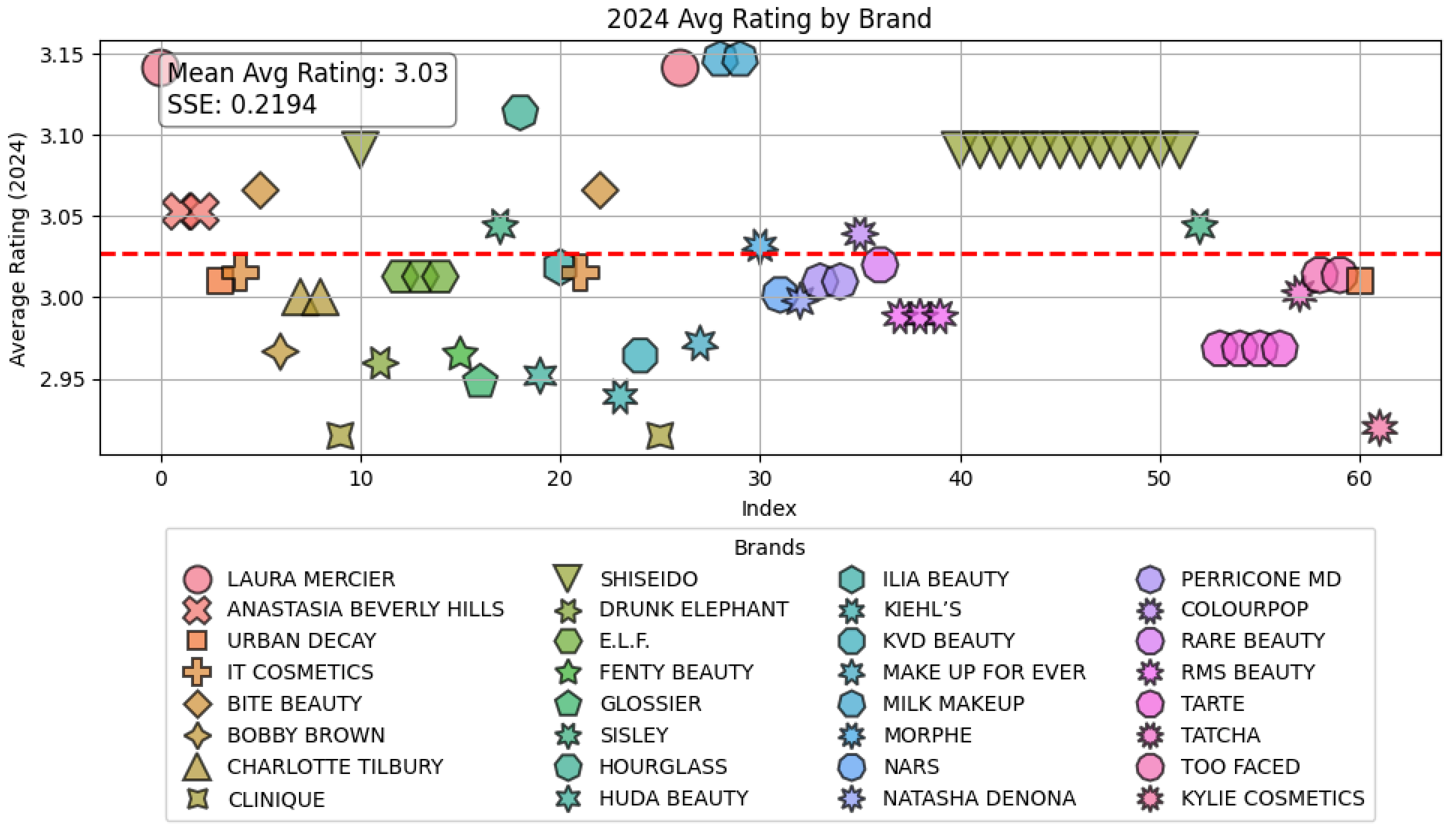


2024 Average Ratings Scatterplots

In [ ]:

```
def plot_avg_rating_with_mean(df):
    """
    Plot Avg_Rating_2024 and display the mean Avg_Rating_2024 as a horizontal line
    along with the sum of squared errors (SSE).

    Args:
        df (pd.DataFrame): DataFrame containing relevant data for Avg_Rating_2024.
    """

    # Calculate the mean and sum of squared errors (SSE)
    avg_rating_mean = df['Avg_Rating_2024'].mean()
    sse = ((df['Avg_Rating_2024'] - avg_rating_mean) ** 2).sum()
    print(f"Mean Avg Rating: {avg_rating_mean:.2f}")
    print(f"SSE: {sse:.4f}")

    plt.figure(figsize=(10, 6))

    # Scatter plot of Avg_Rating_2024 with Brand legend intact
    scatter_plot = sns.scatterplot(
        data=df,
        x=df.index,  # Using index for x-axis
        y='Avg_Rating_2024',
        hue='Brand',  # Hue based on the brand
        style='Brand',
        s=300,  # Marker size for better visibility
        alpha=0.7,  # Opacity for better visibility
        edgecolor='black'  # Outline for better visibility
    )

    # Plot a horizontal line for the mean Avg_Rating_2024 (not in the legend)
    plt.axhline(y=avg_rating_mean, color='red', linestyle='--', linewidth=2)

    # Create a legend for brands directly from the scatter plot handles
    handles, labels = scatter_plot.get_legend_handles_labels()

    # Adjust the legend to be at the bottom and span 2 columns
    brand_legend = plt.legend(
        handles,  # Only the scatter plot elements are included in the legend
        labels,
        loc='upper center',  # Position at the upper center
        borderpad=0.5,
        frameon=True,
        bbox_to_anchor=(0.5, -0.15),  # Move legend below the plot
        title='Brands',
        ncol=4,  # Span 4 columns
        fontsize='medium',
        handletextpad=0.5,  # Space between handle and text
        markerscale=.75  # Scale the size of the markers in the legend
    )

    # Add mean and SSE as a text box on the plot
    textstr = f'Mean Avg Rating: {avg_rating_mean:.2f}\nSSE: {sse:.4f}'
    plt.gca().text(
        0.05, 0.95, textstr, fontsize=12,
        verticalalignment='top',
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.5),
        transform=plt.gca().transAxes
    )

    # Add labels and title
    plt.title('2024 Avg Rating by Brand')
    plt.xlabel('Index')
    plt.ylabel('Average Rating (2024)')

    plt.grid(True)
    plt.tight_layout()
    plt.show()

plot_avg_rating_with_mean(merged_agg_df)
```

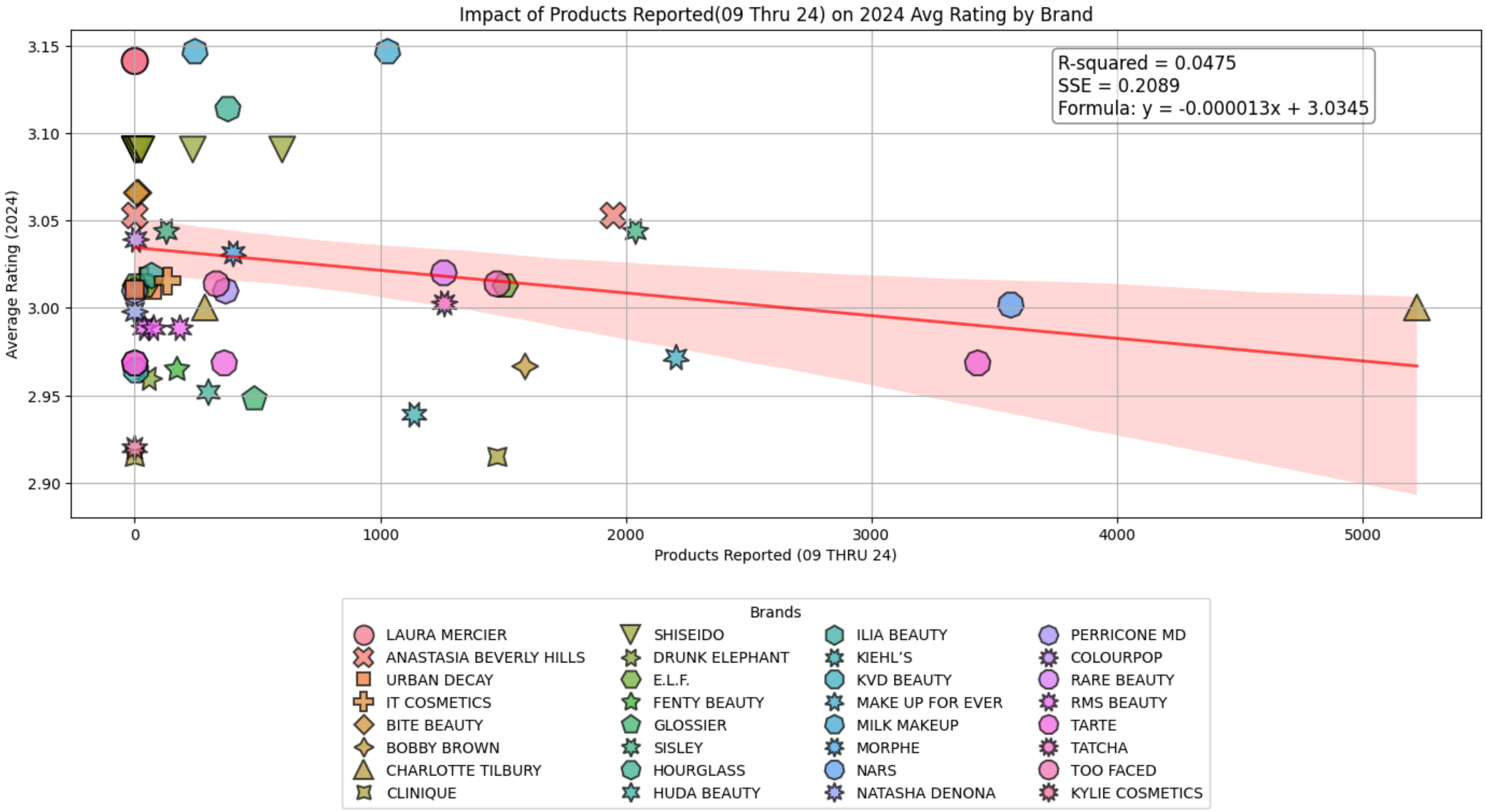
```
Mean Avg Rating: 3.03
SSE: 0.2194
```



In [ ]:
```
def linear_impact_on_avg_rating(df):
    """
    Create a chart to visualize the effects of Products Reported on Avg Rating,
    with additional information on the regression line (R-squared, SSE, and slope-intercept formula).

    Args:
        df (pd.DataFrame): DataFrame containing relevant data for brands.
    """

    # Perform linear regression to get the slope, intercept, and R-squared value
    slope, intercept, r_value, p_value, std_err = stats.linregress(
        df['Products_Reported_09THRU24'], df['Avg_Rating_2024']
```

```
)
r_squared = r_value**2  # R-squared value

# Calculate the sum of squared errors (SSE)
predicted_values = intercept + slope * df['Products_Reported_09THRU24']
sse = np.sum((df['Avg_Rating_2024'] - predicted_values) ** 2)

print(f"R-squared value: {r_squared}")
print(f"Slope: {slope}")
print(f"Intercept: {intercept}")
print(f"Sum of Squared Errors (SSE): {sse}")

plt.figure(figsize=(14, 8))

# Adding a regression line with Seaborn's regplot (without scatter points)
sns.regplot(
    data=df,
    x='Products_Reported_09THRU24',
    y='Avg_Rating_2024',
    scatter=False,  # No scatter plot, only regression line
    color='red',  # Color for the regression line
    line_kws={'linewidth': 2, 'alpha': 0.7}  # Customize the regression line
)

# Scatter plot using Seaborn
scatter_plot = sns.scatterplot(
    data=df,
    x='Products_Reported_09THRU24',
    y='Avg_Rating_2024',
    hue='Brand',  # Hue based on the brand
    alpha=0.7,  # Opacity for better visibility
    s=300,
    edgecolor='black',  # Outline for better visibility
    style='Brand',  # Different markers for different brands
)

# Create a legend for brands directly from the scatter plot handles
handles, labels = scatter_plot.get_legend_handles_labels()

# Extract the last 15 handles and labels for the Brand legend
brand_handles = handles[-32:]  # Adjust the slice if necessary
brand_labels = labels[-32:]  # Adjust the slice if necessary

# Adjust the legend to be at the bottom and span 2 columns
brand_legend = plt.legend(
    brand_handles,
    brand_labels,
    loc='upper center',  # Position at the upper center
    borderpad=0.5,
    frameon=True,
    bbox_to_anchor=(0.5, -0.15),  # Move legend below the plot
    title='Brands',
    ncol=4,  # Span 4 columns
    fontsize='medium',
    handletextpad=0.5,  # Space between handle and text
    markerscale=.75  # Scale the size of the markers in the legend
    )

    # Add labels and title
    plt.title('Impact of Products Reported(09 Thru 24) on 2024 Avg Rating by Brand')
    plt.xlabel('Products Reported (09 THRU 24)')  # Update x-axis label
    plt.ylabel('Average Rating (2024)')  # Update y-axis label

    # Add text for the regression details (R-squared, slope-intercept formula, SSE)
    plt.text(
        0.70, 0.95,
        f'R-squared = {r_squared:.4f}\n'
        f'SSE = {sse:.4f}\n'
        f'Formula: y = {slope:.6f}x + {intercept:.4f}',
        transform=plt.gca().transAxes,
        fontsize=12,
        verticalalignment='top',
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.5)
    )

    plt.grid(True)
    plt.tight_layout()

    plt.show()

linear_impact_on_avg_rating(merged_agg_df)
```

```
R-squared value: 0.04748665805526055
Slope: -1.296722634964563e-05
Intercept: 3.034479644331162
Sum of Squared Errors (SSE): 0.20894491199096163
```



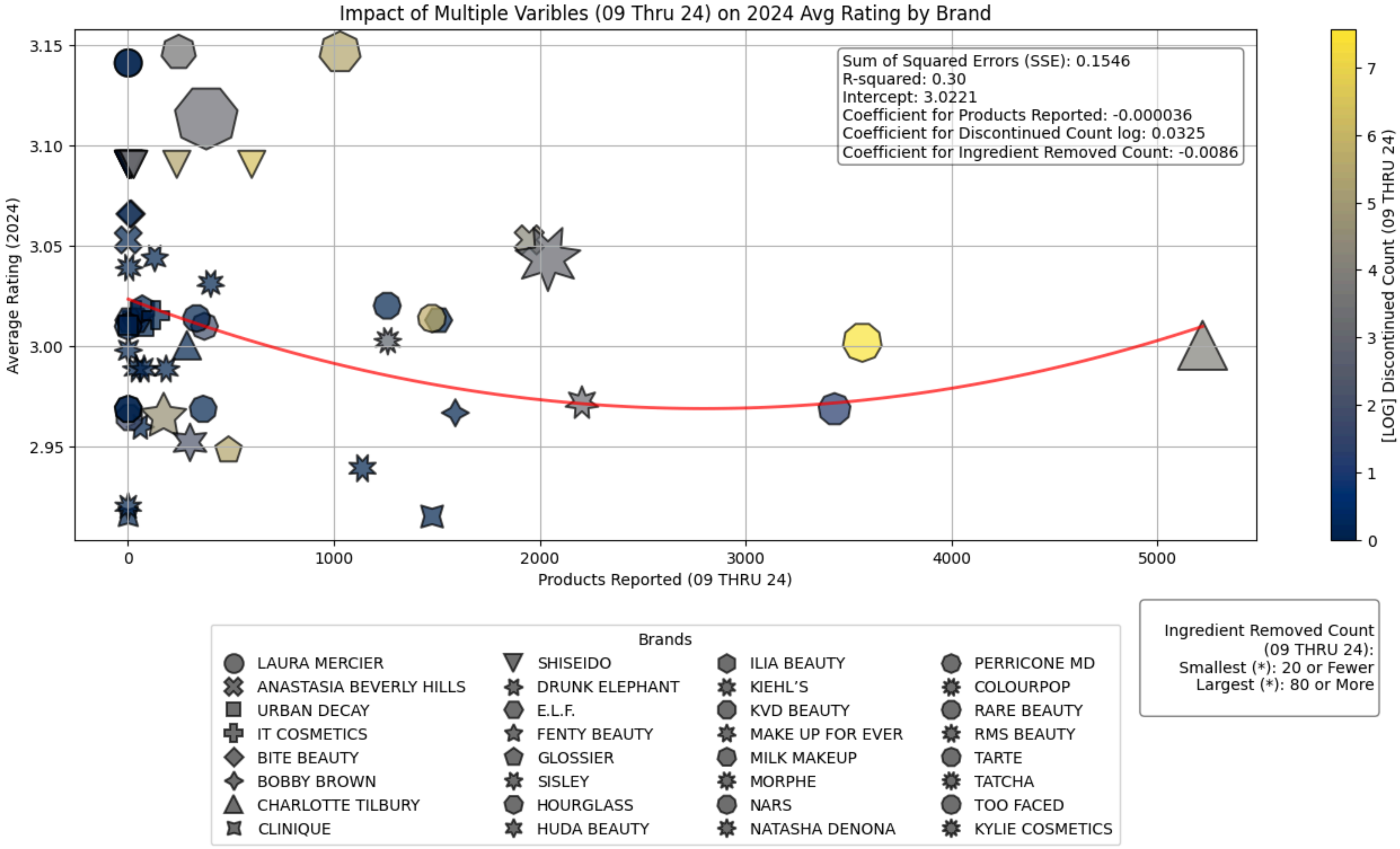
In [ ]:

```
def polynomial_impact_on_avg_rating(df):
    """
    Create a comprehensive chart to visualize the effects of Products Reported on Avg Rating,
    including discontinued and ingredient removed counts.

    Args:
    df (pd.DataFrame): DataFrame containing relevant data for brands.
    """
    # Handle log(0) case by replacing zeros with a small value (e.g., 1)
    df['Discontinued_Count_09THRU24_log'] = np.log(df['Discontinued_Count_09THRU24'].replace(0, 1))

    plt.figure(figsize=(14, 8))
    # Normalize Discontinued Count log for color mapping
    norm = plt.Normalize(df['Discontinued_Count_09THRU24_log'].min(), df['Discontinued_Count_09THRU24_log'].max())
    cmap = plt.get_cmap('cividis')  # Light-to-dark colormap

    # Prepare independent variables (X) and dependent variable (y)
```

```
X = df[['Products_Reported_09THRU24', 'Discontinued_Count_09THRU24_log', 'Ingredient_Removed_Count_09THRU24']]
y = df['Avg_Rating_2024']

# Create polynomial features
poly = PolynomialFeatures(degree=2)  # You can adjust the degree for your model
X_poly = poly.fit_transform(X)

# Fit polynomial regression model
model = LinearRegression()
model.fit(X_poly, y)

# Calculate R-squared value
r_squared = model.score(X_poly, y)

# Calculate Sum of Squared Errors (SSE)
predictions = model.predict(X_poly)
sse = np.sum((y - predictions) ** 2)

# Print R-squared and SSE
print(f"R-squared: {r_squared:.4f}")
print(f"Sum of Squared Errors (SSE): {sse:.4f}")

# Create a grid of values to predict for the regression line
X_grid = np.linspace(X['Products_Reported_09THRU24'].min(), X['Products_Reported_09THRU24'].max(), 100)

# Create predictions for each combination of Products Reported and other variables
predictions = []
for value in X_grid:
    temp_X = pd.DataFrame({
        'Products_Reported_09THRU24': [value],  # Wrap in list
        'Discontinued_Count_09THRU24_log': [df['Discontinued_Count_09THRU24_log'].mean()],  # Wrap in list
        'Ingredient_Removed_Count_09THRU24': [df['Ingredient_Removed_Count_09THRU24'].mean()],  # Wrap in list
    })
    temp_X_poly = poly.transform(temp_X)
    predictions.append(model.predict(temp_X_poly)[0])

# Plotting the regression line
plt.plot(X_grid, predictions, color='red', linewidth=2, alpha=0.7, label='Polynomial Regression Line')

# Scatter plot using Seaborn
scatter_plot = sns.scatterplot(
    data=df,
    x='Products_Reported_09THRU24',
    y='Avg_Rating_2024',
    hue='Discontinued_Count_09THRU24_log',  # Use hue for log of discontinued count
    size='Ingredient_Removed_Count_09THRU24',  # Size based on ingredient removed count
    sizes=(300, 1800),  # Size range for the markers
    palette=cmap,  # Color palette for log of discontinued count
    alpha=0.7,  # Opacity for better visibility
    edgecolor='black',  # Outline for better visibility
    style='Brand',  # Different markers for different brands
)

# Create a legend for brands directly from the scatter plot handles
handles, labels = scatter_plot.get_legend_handles_labels()
brand_handles = handles[-32:]  # Adjust the slice if necessary
```

```
brand_labels = labels[-32:]  # Adjust the slice if necessary

# Adjust the legend to be at the bottom and span 2 columns
brand_legend = plt.legend(
    brand_handles,
    brand_labels,
    loc='upper center',  # Position at the upper center
    borderpad=0.5,
    frameon=True,
    bbox_to_anchor=(0.5, -0.15),  # Move legend below the plot
    title='Brands',
    ncol=4,  # Span 4 columns
    fontsize='medium',
    handletextpad=0.5,  # Space between handle and text
    markerscale=2  # Scale the size of the markers in the legend
)

# Add a colorbar for Discontinued Count
sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
sm.set_array([])
colorbar = plt.colorbar(sm, ax=scatter_plot.axes)
colorbar.set_label('[LOG] Discontinued Count (09 THRU 24)')

# Add labels and title
plt.title('Impact of Multiple Varibles (09 Thru 24) on 2024 Avg Rating by Brand')
plt.xlabel('Products Reported (09 THRU 24)')  # Update x-axis label
plt.ylabel('Average Rating (2024)')  # Update y-axis label

# Create a text box for size annotations
size_annotation = """
Ingredient Removed Count
(09 THRU 24):
Smallest (*): 20 or Fewer
Largest (*): 80 or More
"""

# Add a text box to the plot
plt.gca().text(1.10, -.33, size_annotation, fontsize=10,
               bbox=dict(boxstyle='round', facecolor='white', alpha=0.5),
               transform=plt.gca().transAxes, ha='right')

# Add coefficients to the text box
intercept = model.intercept_
coefficients = model.coef_

print(f"Coefficient for Products Reported: {coefficients[1]:.6f}")
print(f"Coefficient for Discontinued Count log: {coefficients[2]:.4f}")
print(f"Coefficient for Ingredient Removed Count: {coefficients[3]:.4f}")

# Create a text box for SSE, R-squared, and coefficients
textstr = '\n'.join((
    r'Sum of Squared Errors (SSE): %.4f' % (sse, ),
    r'R-squared: %.2f' % (r_squared, ),
    r'Intercept: %.4f' % (intercept, ),
    r'Coefficient for Products Reported: %.6f' % (coefficients[1], ),
    r'Coefficient for Discontinued Count log: %.4f' % (coefficients[2], ),
        r'Coefficient for Ingredient Removed Count: %.4f' % (coefficients[3], )
    ))

    # Add a text box to the plot
    plt.gca().text(0.65, 0.75, textstr, transform=plt.gca().transAxes, fontsize=10,
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.5))

    plt.grid(True)
    plt.tight_layout()
    plt.show()

polynomial_impact_on_avg_rating(merged_agg_df)
```

```
R-squared: 0.2950
Sum of Squared Errors (SSE): 0.1546
Coefficient for Products Reported: -0.000036
Coefficient for Discontinued Count log: 0.0325
Coefficient for Ingredient Removed Count: -0.0086
```



Impact of Multiple Varibles (09 Thru 24) on 2024 Avg Rating by Brand

```
Sum of Squared Errors (SSE): 0.1546
R-squared: 0.30
Intercept: 3.0221
Coefficient for Products Reported: -0.000036
Coefficient for Discontinued Count log: 0.0325
Coefficient for Ingredient Removed Count: -0.0086
```