MSDS 696 Data Science Practicum II: "Recipe for Success: Data-Driven Strategies for New Restaurant Owners in San Diego" by Tia Page

Introduction

This project titled "Recipe for Success: Data-driven Strategies for Future Restaurant Owners in San Diego" aims to provide explore and present insights that will help new and prospective restaurant owners succeed in San Diego's competitive market. The goals of this project are to identify key success factors for restaurants in San Diego, recommend optimal locations in the city to establish one's restaurant, find strategies to enhance customer experience, and determine appropriate menu prices. All of these factors are essential in navigating the competitive nature of San Diego's food industry. This project utilizes Python to analyze data from various sources, including restaurant reviews, demographic data, and geographic information. The code is organized into sections for data collection, cleaning, exploratory analysis, machine learning, and visualization to achieve these goals. By leveraging these techniques, this project seeks to provide aspiring restaurant owners with the knowledge to thrive in San Diego's culinary landscape.

Problem Definition

The San Diego restaurant industry has experienced an alarming increase in closures in recent years. This unfortunate trend poses a significant threat to the local economy and culinary landscape. A local news article published by CBS8 on June 3, 2024 (https://www.cbs8.com/article/news/local/why-restaurants-closing-san-diego/509-cc12470e-d38b-4c5c-aa93-ba9f85efe90d) attributes the rise in San Diego restaurant closures to two main factors:

1. **Rising Costs of Living**: Restaurants are facing escalating costs for ingredients, labor, rent, and utilities due to inflation in recent years. Inflation has also indirectly impacted restaurant revenues, as customers who are also impacted by the rising costs of living try to save money by eating out less. This financial strain makes it difficult for many establishments to remain profitable.
2. **Overly Saturated Market**: The number of restaurants in San Diego has grown considerably over the past few years, creating intense competition. This saturation makes it challenging for new restaurants to establish themselves, as well as existing restaurants to maintain their customer base.

These factors contribute to a volatile environment for restaurants in San Diego, leading to increased closures and an uncertain future for the industry. This research project aims to provide data-driven insights and strategies to help new restaurant owners mitigate these challenges and improve their chances of success.

Data Collection

This project leverages a multi-faceted data collection approach to gather comprehensive information on San Diego restaurants and their surrounding market. The data collection process involved obtaining the following information:

- **San Diego Zip Codes and Neighborhoods:** A list of San Diego zip codes and their corresponding neighborhood names was retrieved using web scraping techniques from the website BkylnDesigns.com. This data provides the geographic foundation for the analysis, enabling the segmentation and comparison of restaurants across different neighborhoods.
- **Land Area Information:** Land area data for each San Diego zip code was obtained through web scraping from USA.com. This information allows for the calculation of population density, which can be a crucial factor in understanding restaurant performance and market saturation.
- **Demographic information** for each zip code, including population, number of households, median income, and average income, was collected from Point2Homes.com. Due to website restrictions, this data was manually compiled from an online table into an Excel spreadsheet. Demographic data provides insights into the socioeconomic characteristics of different neighborhoods, helping to understand customer profiles and potential market segments.
- **Restaurant Listings and Google Review URLs:** A list of restaurants and their corresponding Google Review URLs was extracted using the Octoparse data extraction tool. The tool's "Google Maps Listings Scraper" template was used, with the keyword "Restaurants in [insert zip code]" to target specific areas. This data provides the core restaurant information for the analysis, including names, ratings, and review counts.
- **Restaurant Reviews:** Google reviews for each restaurant were collected using the Octoparse data extraction tool's "Google Reviews Scraper" template. This involved inputting the previously collected Google Review URLs to extract individual reviews and their associated

ratings. Review data provides valuable insights into customer sentiment, preferences, and experiences, allowing for a deeper understanding of restaurant performance and areas for improvement.

Below are the steps that were taken for data collection:

1) Retrieving the **names of each neighborhood** in San Diego with respect to the city's zip codes via **webscraping** BkylnDesigns.com using **Python library BeautifulSoup** at the url: www.bklyndesigns.com/san-diego-zip-codes/

```
# List of San Diego Zipcodes with Their Neighborhood Names

# Install necessary libraries and packages
!pip install pandas requests beautifulsoup4

import pandas as pd
import requests
from bs4 import BeautifulSoup

# URL of the webpage
url = "https://bklyndesigns.com/san-diego-zip-codes/"

# Send a GET request to the webpage
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    # Parse the webpage content
    soup = BeautifulSoup(response.content, 'html.parser')

    # Find the table on the webpage
    table = soup.find('table')

    # Extract the rows from the table
    rows = table.find_all('tr')

    # Initialize a list to store the data
    data = []

    # Loop through the rows and extract ZIP code and address
    for row in rows[1:]:  # Skip the header row
        cols = row.find_all('td')
        if len(cols) >= 2:  # Ensure there are enough columns
            zip_code = cols[0].text.strip()
            address = cols[1].text.strip()
            data.append({'ZIP Code': zip_code, 'Address': address})

# Create a Data Frame from the extracted data
    df_zip = pd.DataFrame(data)
# Save Data Frame as a CSV file
    df_zip.to_csv('sd_neighborhoods.csv', index=False)

else:
    print(f"Failed to retrieve the webpage. Status code: {response.status_code}")
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (2.32.3)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (4.12.3)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests) (2024.12.14)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4) (2.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

2) Retrieving **land area information** of each San Diego zip code via **Webscraping** USA.com. Each zip code used for this code was obtained from the previous file created called "sd_neighborhoods.csv."

```
import re

# List of San Diego ZIP codes
zip_codes = ["91942", "92037", "92101", "92102", "92103", "92104", "92105", "92106",
              "92107", "92108", "92109", "92110", "92111", "92113", "92114", "92115",
              "92116", "92117", "92119", "92120", "92121", "92122", "92123", "92124",
              "92126", "92127", "92128", "92129", "92130", "92131", "92142", "92150",
              "92153"]
# Base URL
base_url = "http://www.usa.com/"

# List to store results
results = []

# Function to get land area
def get_land_area(zip_code):
    url = f"{base_url}{zip_code}-ca.htm"
    response = requests.get(url)

    if response.status_code == 200:
        soup = BeautifulSoup(response.text, 'html.parser')

        # Look for all tables
        tables = soup.find_all('table')

        for table in tables:
            for row in table.find_all('tr'):
                cells = row.find_all('td')

                # Ensure there are enough cells
                if len(cells) >= 2:
                    label = cells[0].text.strip().lower()  # Convert label to lowercase
                    value = cells[1].text.strip()  # Get the corresponding value

                    # Check for "land area"
                    if 'land area' in label:
                        # Extract just the numeric part
                        land_area_value = re.search(r'(\d+\.?\d*)\s*sq mi', value)
                        if land_area_value:
                            return land_area_value.group(1) + " sq mi"

        print(f"'Land Area' not found in tables for ZIP Code: {zip_code}")
        return None
    else:
        print(f"Failed to retrieve data for ZIP Code: {zip_code}, Status Code: {response.status_code}")
        return None

# Loop through each ZIP code and collect the results
for zip_code in zip_codes:
    land_area = get_land_area(zip_code)
    results.append({"ZIP Code": zip_code, "Land Area": land_area})

# Create a DataFrame from the results
df_land_area = pd.DataFrame(results)

# Save as CSV file
df_land_area.to_csv('sd_land_area.csv', index=False)

# Preview data
df_land_area.head(50)
```

```
'Land Area' not found in tables for ZIP Code: 92142
'Land Area' not found in tables for ZIP Code: 92150
'Land Area' not found in tables for ZIP Code: 92153
```

	ZIP Code	Land Area
0	91942	5.84 sq mi
1	92037	13.08 sq mi
2	92101	4.72 sq mi
3	92102	4.63 sq mi
4	92103	3.77 sq mi
5	92104	3.79 sq mi
6	92105	5.57 sq mi
7	92106	5.64 sq mi
8	92107	3.15 sq mi
9	92108	4.28 sq mi
10	92109	7.60 sq mi
11	92110	4.85 sq mi
12	92111	8.46 sq mi
13	92113	5.27 sq mi
14	92114	8.24 sq mi
15	92115	6.50 sq mi
16	92116	3.47 sq mi
17	92117	8.79 sq mi
18	92119	6.85 sq mi
19	92120	6.66 sq mi
20	92121	12.25 sq mi
21	92122	15.93 sq mi
22	92123	8.17 sq mi
23	92124	10.48 sq mi
24	92126	12.67 sq mi
25	92127	22.06 sq mi
26	92128	11.17 sq mi
27	92129	14.07 sq mi
28	92130	18.44 sq mi
29	92131	25.11 sq mi
30	92142	None
31	92150	None
32	92153	None

Next steps: Generate code with `df_land_area` View recommended plots New interactive sheet

3) Retrieving **demographic information** for each zip code related to population, number of households, median income, and average income via Point2Homes.com through the url: https://www.point2homes.com/US/Neighborhood/CA/San-Diego-Demographics.html#MedianIncomeByZipcode

Because the Point2Homes.com website does not permit Webscraping, I collected the demographic data for each zip code through copy and pasting the table containing the information in an Excel spreadsheet. The file with this information (sd_demographics.xlsx) can be downloaded in the Github folder for this project and can be viewed in a Python environment using the code below.

```
# Downloading demographics data for each San Diego zipcode
df_demographics = pd.read_excel('/content/sd_demographics.xlsx')

# Viewing a preview of the downloaded data
df_demographics.head()
```

```
# Changing column "ZipCode" to "Zip Code"
df_demographics = df_demographics.rename(columns={'ZipCode': 'Zip Code'})

# Previewing data
df_demographics.head()
```

	Zip Code	Population	Number of Households	Median Income	Average Income
0	92101	46025	27295	86403	121867
1	92102	40051	14482	68900	85567
2	92103	34296	18834	94210	136652
3	92104	46613	22454	86291	106742
4	92105	72688	22847	58924	77386

Next steps: Generate code with df_demographics View recommended plots New interactive sheet

4) List of restaurants and their Google Review urls were retrieved using **Octosparse data extraction tool** and the template called "Google Maps Listings Scraper (By Keyword)." For the keyword, I typed in "Restaurants in [insert zip code]." The file containing this list (sd_restaurant_listings.xlsx) can be downloaded from the Github folder of this project and can be viewed in a Python environment using the code below:

```
# Downloading demographics data for each San Diego zipcode
df_restaurant_listings = pd.read_excel('/content/sd_restaurant_listings.xlsx')

# Renaming the column for restaurant names in df_restaurant_listings
df_restaurant_listings = df_restaurant_listings.rename(columns={'Name': 'Restaurant Name'})

# Getting a list of all columns except "Zip Code"
cols = [col for col in df_restaurant_listings.columns if col != 'Zip Code']

# Appending "Zip Code" to the end of the list
cols.append('Zip Code')

# Reordering the DataFrame columns using the new list
df_restaurant_listings = df_restaurant_listings[cols]

# Viewing a preview of the downloaded data
df_restaurant_listings.head()
```

	Restaurant Name	Restaurant Rating	Review_count	Price_range	Category	Address	Tags	Deta
0	Taco Bell	3.0	1,017	$10–20	Fast Food	9821 Aero Dr	Dine-in\n·\nDrive-through\n·\nDelivery\n \nORD...	https://www.google.com/maps/place/Taco+B
1	Living Room Cafe & Bistro	3.0	882	$10–20	Mexican	2541 San Diego Ave	Dine-in\n·\nTakeout\n·\nDelivery	https://www.google.com/maps/place/Living+R
2	McDonald's	3.1	1,524	$1–10	Fast Food	4260 Nobel Dr	Dine-in\n·\nDrive-through\n·\nNo-contact deliv...	https://www.google.com/maps/place/McDonalc
3	Jack in the Box	3.5	2,417	$10–20	Fast Food	1110 C St	Dine-in\n·\nDrive-through\n·\nNo-contact deliv...	https://www.google.com/maps/place/Jack+in
4	McDonald's	3.5	1,269	$1–10	Fast Food	8929 Clairemont Mesa Blvd	Dine-in\n·\nDrive-through\n·\nNo-contact deliv...	https://www.google.com/maps/place/McDonalc

Next steps: Generate code with df_restaurant_listings View recommended plots New interactive sheet

5) Reviews from each of the above restaurants were retrieved using Octospare data extraction tool and the template called "Google Reviews Scraper." Each url from the previous "sd_restaurant_listings.xlsx" file was inputted into the template in order to obtain reviews from each restaurant on the list. The file containing the reviews (sd_restaurant_reviews.xlsx) can be downloaded from the Github folder of this project and can be viewed in a Python environment using the code below:

```
df_reviews = pd.read_excel('/content/sd_restaurant_reviews.xlsx')

# Renaming the restaurant name column in df_reviews
df_reviews = df_reviews.rename(columns={'Name': 'Restaurant Name'})
```

```
# Viewing preview of data
df_reviews.head()
```

	Restaurant Name	Category	Restaurant Rating	Rating_count	Address	Reviewer	Reviewer_page	Review_time	Review
0	100's seafood grill buffet	Buffet restaurant	5 stars	3,675	2828 Camino del Rio S, San Diego, CA 92108	Richie K	https://www.google.com/maps/contrib/1155789259...	a week ago	This is a great Seafood Buffet with plenty of ...
1	100's seafood grill buffet	Buffet restaurant	5 stars	3,675	2828 Camino del Rio S, San Diego, CA 92108	Liz Rodriguez	https://www.google.com/maps/contrib/1085513239...	4 months ago	The food here is absolutely delicious! Every d...
2	100's seafood grill buffet	Buffet restaurant	3 stars	3,675	2828 Camino del Rio S, San Diego, CA 92108	Jess D	https://www.google.com/maps/contrib/1117120582...	4 months ago	Honestly, I was so disappointed in this place....
3	100's seafood grill buffet	Buffet restaurant	5 stars	3,675	2828 Camino del Rio S, San Diego, CA 92108	Shayok	https://www.google.com/maps/contrib/1166636833...	3 months ago	Great quality seafood and all items I tried we...
4	100's seafood grill buffet	Buffet restaurant	3 stars	3,675	2828 Camino del Rio S, San Diego, CA 92108	Tim A	https://www.google.com/maps/contrib/1161764752...	a week ago	Well, I can say I went on lobster night, which...

Next steps: Generate code with df_reviews | View recommended plots | New interactive sheet

Data Cleaning & Preparation

The main goal of this portion is to create one clean and organized data containing information related to San Diego restaurants and their reviews. The desired dataset will contain columns with the following information: Restaurant name, neighborhood, population, land area, population density, number of households per neighborhood, average/median income, overall Google rating per restaurant, total number of Google reviews per restaurant, cuisine category, price level, and level of success. Throughout the data cleaning/preparation step, I will be performing EDA to see what values are missing, which rows need to be removed, and what values need to be changed or fixed.

Below are the steps for data cleaning/preparation:

1) **Merging** the collected data to get one final "sd_reviews" dataset which will ultimately be cleaned:

```
# Combining the df_zip and df_land_area data frames by merging based on the 'Zip Code' column
zip_land_area_df = pd.merge(df_zip, df_land_area, on='ZIP Code', how='left')

# Displaying the merged DataFrame
zip_land_area_df.head(50)
```

	ZIP Code	Address	Land Area
0	92101	Downtown, San Diego	4.72 sq mi
1	92102	San Diego	4.63 sq mi
2	92103	Hillcrest	3.77 sq mi
3	92104	North Park	3.79 sq mi
4	92105	City Heights	5.57 sq mi
5	92106	Point Loma	5.64 sq mi
6	92107	Ocean Beach	3.15 sq mi
7	92108	Mission Valley	4.28 sq mi
8	92109	Pacific Beach	7.60 sq mi
9	92110	Old Town	4.85 sq mi
10	92111	Linda Vista	8.46 sq mi
11	92112	Downtown P.O. Box	NaN
12	92113	Logan Heights	5.27 sq mi
13	92114	Encanto	8.24 sq mi
14	92115	College Grove	6.50 sq mi
15	92116	Normal Heights	3.47 sq mi
16	92117	Clairemont	8.79 sq mi
17	92118	Coronado	NaN
18	92119	Navajo	6.85 sq mi
19	92120	Grantville	6.66 sq mi
20	92121	Sorrento Valley	12.25 sq mi
21	92122	University	15.93 sq mi
22	92123	Serra Mesa	8.17 sq mi
23	92124	Tierrasanta	10.48 sq mi
24	92126	Mira Mesa	12.67 sq mi
25	92127	Rancho Bernardo	22.06 sq mi
26	92128	Rancho Bernardo	11.17 sq mi
27	92129	Rancho Penasquitos	14.07 sq mi
28	92130	Carmel Valley	18.44 sq mi
29	92131	Scripps Ranch	25.11 sq mi
30	92132	Naval Supply Center	NaN
31	92134	Naval Hospital	NaN
32	92135	San Diego P.O. Box	NaN
33	92136	32nd St. Naval Station	NaN
34	92137	Midway P.O. Box	NaN
35	92138	Midway P.O. Box	NaN
36	92139	Paradise Hills	NaN
37	92140	San Diego	NaN
38	92142	Tierrasanta P.O. Box	None
39	92143	San Ysidro P.O. Box	NaN
40	92145	Miramar Air Station	NaN
41	92147	ASW Training Center	NaN
42	92149	Paradise Hills P.O. Box	NaN
43	92150	Downtown P.O. Box	None
44	92152	Spawars System Center	NaN
45	92153	Nestor P.O. Box	None
46	92154	Otay Mesa	NaN

47	92155	Naval Amphibious Base	NaN
48	92158	County Jail	NaN
49	92159	Navajo P.O. Box	NaN

Next steps: Generate code with `zip_land_area_df` View recommended plots New interactive sheet

```
# Converting 'ZIP Code' in zip_land_area_df to int64 to match the type of 'Zip Code' in df_demographics
zip_land_area_df['ZIP Code'] = pd.to_numeric(zip_land_area_df['ZIP Code'], errors='coerce').astype('Int64')

# Merging the data
zip_demographics_df = pd.merge(zip_land_area_df, df_demographics, left_on='ZIP Code', right_on='Zip Code', how='left')

# Displaying the merged DataFrame
zip_demographics_df.head()
```

	ZIP Code	Address	Land Area	Zip Code	Population	Number of Households	Median Income	Average Income
0	92101	Downtown, San Diego	4.72 sq mi	92101.0	46025.0	27295.0	86403.0	121867.0
1	92102	San Diego	4.63 sq mi	92102.0	40051.0	14482.0	68900.0	85567.0
2	92103	Hillcrest	3.77 sq mi	92103.0	34296.0	18834.0	94210.0	136652.0
3	92104	North Park	3.79 sq mi	92104.0	46613.0	22454.0	86291.0	106742.0
4	92105	City Heights	5.57 sq mi	92105.0	72688.0	22847.0	58924.0	77386.0

Next steps: Generate code with `zip_demographics_df` View recommended plots New interactive sheet

```
# Combining df_restaurant listings and the previous zip_demographics data frame
restaurant_demographics_df = pd.merge(df_restaurant_listings, zip_demographics_df, left_on='Zip Code', right_on='ZIP Code', how='left')

# Displaying the merged data frame
restaurant_demographics_df.head()

# Moving "ZIP Code" to the end of restaurant_demographics_df
# Changing 'Zip Code' to 'ZIP Code' to match the actual column name
cols = [col for col in restaurant_demographics_df.columns if col != 'ZIP Code']
cols.append('ZIP Code')  # Add 'ZIP Code' to the end
restaurant_demographics_df = restaurant_demographics_df[cols]  # Reorder columns

# Displaying the updated data frame
restaurant_demographics_df.head()
```

	Restaurant Name	Restaurant Rating	Review_count	Price_range	Category	Address_x	Tags	Deta
0	Taco Bell	3.0	1,017	$10–20	Fast Food	9821 Aero Dr	Dine-in\n·\nDrive-through\n·\nDelivery\n \nORD...	https://www.google.com/maps/place/Taco+B
1	Living Room Cafe & Bistro	3.0	882	$10–20	Mexican	2541 San Diego Ave	Dine-in\n·\nTakeout\n·\nDelivery	https://www.google.com/maps/place/Living+R
2	McDonald's	3.1	1,524	$1–10	Fast Food	4260 Nobel Dr	Dine-in\n·\nDrive-through\n·\nNo-contact deliv...	https://www.google.com/maps/place/McDonalc
3	Jack in the Box	3.5	2,417	$10–20	Fast Food	1110 C St	Dine-in\n·\nDrive-through\n·\nNo-contact deliv...	https://www.google.com/maps/place/Jack+in
4	McDonald's	3.5	1,269	$1–10	Fast Food	8929 Clairemont Mesa Blvd	Dine-in\n·\nDrive-through\n·\nNo-contact deliv...	https://www.google.com/maps/place/McDonalc

Next steps: Generate code with `restaurant_demographics_df` View recommended plots New interactive sheet

```
# Combining the previous restaurant_demographics dataframe with df_reviews data frame for the final merge
# Combining restaurant_demographics_df with df_reviews
sd_reviews = pd.merge(restaurant_demographics_df, df_reviews, on='Restaurant Name', how='left')

# Displaying the merged data frame
sd_reviews.head()
```

	Restaurant Name	Restaurant Rating_x	Review_count	Price_range	Category_x	Address_x	Tags	Deta
0	Taco Bell	3.0	1,017	$10–20	Fast Food	9821 Aero Dr	Dine-in\n·\nDrive-through\n·\nDelivery\n \nORD...	https://www.google.com/maps/place/Taco+E
1	Taco Bell	3.0	1,017	$10–20	Fast Food	9821 Aero Dr	Dine-in\n·\nDrive-through\n·\nDelivery\n \nORD...	https://www.google.com/maps/place/Taco+E
2	Taco Bell	3.0	1,017	$10–20	Fast Food	9821 Aero Dr	Dine-in\n·\nDrive-through\n·\nDelivery\n \nORD...	https://www.google.com/maps/place/Taco+E
3	Taco Bell	3.0	1,017	$10–20	Fast Food	9821 Aero Dr	Dine-in\n·\nDrive-through\n·\nDelivery\n \nORD...	https://www.google.com/maps/place/Taco+E
4	Taco Bell	3.0	1,017	$10–20	Fast Food	9821 Aero Dr	Dine-in\n·\nDrive-through\n·\nDelivery\n \nORD...	https://www.google.com/maps/place/Taco+E

5 rows × 27 columns

2) Remove unnecessary columns: Address_x, Tags, Detail_URL, Category_y, Rating_count, Address, Reviewer, Reviewer_page, Review_time, Store_reply, Zip Code_y, ZIP Code, and Likes

```
# Specifying the columns to remove
columns_to_remove = ['Address_x', 'Tags', 'Detail_URL', 'Category_y', 'Rating_count', 'Address', 'Reviewer', 'Reviewer_page', 'Review_time', 'Sto

# Removing columns using drop() method
sd_reviews = sd_reviews.drop(columns=columns_to_remove)

# Previewing data
sd_reviews.head()
```

	Restaurant Name	Restaurant Rating_x	Review_count	Price_range	Category_x	Zip Code_x	Address_y	Land Area	Population	Number of Households	Median Income	Average Income	Restaurant Rating_y
0	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta P.O. Box	None	NaN	NaN	NaN	NaN	5 stars
1	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta P.O. Box	None	NaN	NaN	NaN	NaN	1 star
2	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta P.O. Box	None	NaN	NaN	NaN	NaN	1 star
3	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta P.O. Box	None	NaN	NaN	NaN	NaN	4 stars
4	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta P.O. Box	None	NaN	NaN	NaN	NaN	1 star

Next steps: Generate code with `sd_reviews` View recommended plots New interactive sheet

3) Changing the names of columns to make them more code-friendly:

```
# Changing names of columns
# Creating a dictionary mapping old column names to new column names
new_column_names = {
    "Restaurant Name": "restaurant_name",
    "Restaurant Rating_x": "rating",
    "Review_count": "review_count",
    "Category_x": "cuisine_type",
    "Zip Code_x": "zip_code",
    "Address_y": "neighborhood",
    "Land Area": "land_area",
    "Population": "population",
    "Number of Households": "num_households",
    "Median Income": "median_income",
    "Average Income": "average_income",
    "Restaurant Rating_y": "review_rating",
    "Review": "review",
    "Likes": "review_likes"
}

# Renaming columns using rename() method
sd_reviews = sd_reviews.rename(columns=new_column_names)

# Previewing data
sd_reviews.head()
```

	restaurant_name	rating	review_count	Price_range	cuisine_type	zip_code	neighborhood	land_area	population	num_households	median_inc
0	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta P.O. Box	None	NaN	NaN	I
1	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta P.O. Box	None	NaN	NaN	I
2	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta P.O. Box	None	NaN	NaN	I
3	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta P.O. Box	None	NaN	NaN	I
4	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta P.O. Box	None	NaN	NaN	I

Next steps: Generate code with `sd_reviews` View recommended plots New interactive sheet

4) After noticing that some of zip codes belong to the PO Boxes of their respective neighborhoods, I created a code to associate these zip codes with their respective neighborhoods so that the neighborhood names are written in one way instead of two. For example, the neighborhood name of the first restaurant listed in the table above is "Tierrasanta P.O. Box." This value for the "neighborhood" column will be changed from "Tierrasanta P.O. Box" to "Tierrasanta."

```
# Find the unique values of "zip_code" and "neighborhood" columns
unique_zip_codes = sd_reviews['zip_code'].unique()
unique_neighborhoods = sd_reviews['neighborhood'].unique()

# Print the unique values
print("Unique Zip Codes:", unique_zip_codes)
print("\nUnique Neighborhoods:", unique_neighborhoods)
```

```
Unique Zip Codes: [92142 92110 92122 92102 92123 92153 92126 92103 92119 92108 92129 92131
 92124 92121 92105 92104 92113 92150 92115 92117 92101 92114 92106 92120
 92109 92111 92128 92130 92116 91942 92127 92107 92037]

Unique Neighborhoods: ['Tierrasanta P.O. Box' 'Old Town' 'University' 'San Diego' 'Serra Mesa'
 'Nestor P.O. Box' 'Mira Mesa' 'Hillcrest' 'Navajo' 'Mission Valley'
 'Rancho Penasquitos' 'Scripps Ranch' 'Tierrasanta' 'Sorrento Valley'
 'City Heights' 'North Park' 'Logan Heights' 'Downtown P.O. Box'
 'College Grove' 'Clairemont' 'Downtown, San Diego' 'Encanto' 'Point Loma'
 'Grantville' 'Pacific Beach' 'Linda Vista' 'Rancho Bernardo'
 'Carmel Valley' 'Normal Heights' nan 'Ocean Beach']
```

```
# Remove "P.O. Box" from all values in "neighborhood" column
sd_reviews['neighborhood'] = sd_reviews['neighborhood'].str.replace('P.O. Box', '', regex=False)

# Get unique neighborhoods after removal
unique_neighborhoods_updated = sd_reviews['neighborhood'].unique()

# Print the updated unique values
print("Unique Neighborhoods (after removal):", unique_neighborhoods_updated)

# Preview new dataset
sd_reviews.head()
```

```
Unique Neighborhoods (after removal): ['Tierrasanta ' 'Old Town' 'University' 'San Diego' 'Serra Mesa' 'Nestor '
 'Mira Mesa' 'Hillcrest' 'Navajo' 'Mission Valley' 'Rancho Penasquitos'
 'Scripps Ranch' 'Tierrasanta' 'Sorrento Valley' 'City Heights'
 'North Park' 'Logan Heights' 'Downtown ' 'College Grove' 'Clairemont'
 'Downtown, San Diego' 'Encanto' 'Point Loma' 'Grantville' 'Pacific Beach'
 'Linda Vista' 'Rancho Bernardo' 'Carmel Valley' 'Normal Heights' nan
 'Ocean Beach']
```

	restaurant_name	rating	review_count	Price_range	cuisine_type	zip_code	neighborhood	land_area	population	num_households	median_inc
0	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	None	NaN	NaN	I
1	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	None	NaN	NaN	I
2	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	None	NaN	NaN	I
3	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	None	NaN	NaN	I
4	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	None	NaN	NaN	I

Next steps: Generate code with `sd_reviews` View recommended plots New interactive sheet

```
import pandas as pd

demographic_cols = ['land_area', 'population', 'num_households', 'median_income', 'average_income']

# Strip leading/trailing spaces from 'neighborhood' to ensure accurate grouping
sd_reviews['neighborhood'] = sd_reviews['neighborhood'].str.strip()

# Create a dictionary to store unique values for each neighborhood
neighborhood_data = {}

for neighborhood in sd_reviews['neighborhood'].unique():
    neighborhood_data[neighborhood] = {}
    for col in demographic_cols:
        # Get the first non-missing value for the current neighborhood and column
        filtered_df = sd_reviews.loc[(sd_reviews['neighborhood'] == neighborhood) & (sd_reviews[col].notna()), col]
        if not filtered_df.empty:
            first_valid_value = filtered_df.iloc[0]
            neighborhood_data[neighborhood][col] = first_valid_value  # Store the unique value

# Now, fill the DataFrame using the collected unique values
for neighborhood, data in neighborhood_data.items():
    for col, value in data.items():
        sd_reviews.loc[sd_reviews['neighborhood'] == neighborhood, col] = value

# Verify the changes
sd_reviews.head()
```

	restaurant_name	rating	review_count	Price_range	cuisine_type	zip_code	neighborhood	land_area	population	num_households	median_inc
0	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	10.48 sq mi	31121.0	11383.0	1058
1	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	10.48 sq mi	31121.0	11383.0	1058
2	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	10.48 sq mi	31121.0	11383.0	1058
3	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	10.48 sq mi	31121.0	11383.0	1058
4	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	10.48 sq mi	31121.0	11383.0	1058

Next steps: Generate code with `sd_reviews` View recommended plots New interactive sheet

5) The land area is missing for the neighborhood called "Nestor," because one of the data sources did not list it. I searched for the land area for Nestor, then manually added it to the data set.

```
# Get unique land area values
unique_land_areas = sd_reviews['land_area'].unique()

# Print the unique values
print("Unique land area values:", unique_land_areas)

# Display rows  with "nan" as their land area
# Display rows with "nan" as their land area
nan_land_area_rows = sd_reviews[sd_reviews['land_area'].isna()]
print(nan_land_area_rows)

# Neighborhood called "Nestor" shows missing demographic values when zip code
# is equal to 92153. Nestor has multiple zip codes.
# Include demographic information for 92153

# Display rows with zip code 92154
zipcode_92154_rows = sd_reviews[sd_reviews['zip_code'] == 92154]
print(zipcode_92154_rows)
```

```
 '3.47 sq mi' nan '3.15 sq mi']
                        restaurant_name  rating review_count Price_range  \
4872           Popeyes Louisiana Kitchen     3.6        1,338      $10–20
4873           Popeyes Louisiana Kitchen     3.6        1,338      $10–20
4874           Popeyes Louisiana Kitchen     3.6        1,338      $10–20
4875           Popeyes Louisiana Kitchen     3.6        1,338      $10–20
4876           Popeyes Louisiana Kitchen     3.6        1,338      $10–20
...                                  ...     ...          ...         ...
57898                    In-N-Out Burger     4.7        2,078      $10–20
57899                    In-N-Out Burger     4.7        2,078      $10–20
57900                    In-N-Out Burger     4.7        2,078      $10–20
58033  Ed Fernandez Restaurant Birrieria     4.9        4,979      $10–20
58034  Ed Fernandez Restaurant Birrieria     4.9        4,979      $10–20
```

```
...            ...       ...          ...       ...          ...
57898    Hamburger     92153       Nestor      None         NaN
57899    Hamburger     92153       Nestor      None         NaN
57900    Hamburger     92153       Nestor      None         NaN
58033      Mexican     92153       Nestor      None         NaN
58034      Mexican     92153       Nestor      None         NaN

       num_households  median_income  average_income review_rating  \
4872              NaN            NaN             NaN        1 star
4873              NaN            NaN             NaN        1 star
4874              NaN            NaN             NaN        1 star
4875              NaN            NaN             NaN        1 star
4876              NaN            NaN             NaN        1 star
...               ...            ...             ...           ...
57898             NaN            NaN             NaN       5 stars
57899             NaN            NaN             NaN       3 stars
57900             NaN            NaN             NaN           NaN
58033             NaN            NaN             NaN       4 stars
58034             NaN            NaN             NaN       5 stars

                                                   review
4872   Popeyes over here selling fried chicken skin l...
4873   Fish sandwich. Don't waste your money. Even af...
4874   I order food on Door Dash. The amount of food ...
4875   Went thru the drive thru. The 1st guy took my ...
4876   Agree with Didi comment, food is good but to w...
...                                                   ...
57898  Always amazing and fast service, the cashier n...
57899  In and out and still has one major flaw until ...
57900                                                NaN
58033  This is more of an event/entertainment than tr...
58034  Came here with the boyfriend for dinner on a F...

[7272 rows x 14 columns]
Empty DataFrame
Columns: [restaurant_name, rating, review_count, Price_range, cuisine_type, zip_code, neighborhood, land_area, population, num_households,
Index: []
```

```python
# Replace zip_code for Nestor in sd_reviews
sd_reviews.loc[sd_reviews['neighborhood'] == 'Nestor', 'zip_code'] = 92154

# Get demographic information for Nestor (zip code 92154) from zip_demographics_df
nestor_demographics = zip_demographics_df[zip_demographics_df['Zip Code'] == 92154].iloc[0]

# Fill in missing demographic information for Nestor in sd_reviews
# Mapping between sd_reviews and zip_demographics_df column names
column_mapping = {
    'land_area': 'Land Area',
    'population': 'Population',
    'num_households': 'Number of Households',
    'median_income': 'Median Income',
    'average_income': 'Average Income'
}

# Convert 'Land Area' to numeric, handling "None" and extracting numeric part
if isinstance(nestor_demographics['Land Area'], str) and nestor_demographics['Land Area'].strip() == "None":
    nestor_demographics['Land Area'] = None  # Convert "None" string to actual None
else:
    import re
    land_area_match = re.search(r'(\d+\.?\d*)', str(nestor_demographics['Land Area']))
    if land_area_match:
        nestor_demographics['Land Area'] = float(land_area_match.group(1))
    else:
        nestor_demographics['Land Area'] = None  # If no numeric part found, set to None


for sd_col, demo_col in column_mapping.items():
    sd_reviews.loc[sd_reviews['neighborhood'] == 'Nestor', sd_col] = nestor_demographics[demo_col]

# Verify the changes
print(sd_reviews[sd_reviews['neighborhood'] == 'Nestor'].head())
```

```
                restaurant_name  rating review_count Price_range cuisine_type  \
4872  Popeyes Louisiana Kitchen     3.6        1,338      $10–20      Chicken
4873  Popeyes Louisiana Kitchen     3.6        1,338      $10–20      Chicken
4874  Popeyes Louisiana Kitchen     3.6        1,338      $10–20      Chicken
4875  Popeyes Louisiana Kitchen     3.6        1,338      $10–20      Chicken
4876  Popeyes Louisiana Kitchen     3.6        1,338      $10–20      Chicken

      zip_code neighborhood land_area  population  num_households  \
4872     92154       Nestor      None     83758.0         23076.0
4873     92154       Nestor      None     83758.0         23076.0
```

```
4874    92154       Nestor      None     83758.0         23076.0
4875    92154       Nestor      None     83758.0         23076.0
4876    92154       Nestor      None     83758.0         23076.0

      median_income  average_income review_rating  \
4872        83287.0        101076.0        1 star
4873        83287.0        101076.0        1 star
4874        83287.0        101076.0        1 star
4875        83287.0        101076.0        1 star
4876        83287.0        101076.0        1 star

                                                review
4872  Popeyes over here selling fried chicken skin l...
4873  Fish sandwich. Don't waste your money. Even af...
4874  I order food on Door Dash. The amount of food ...
4875  Went thru the drive thru. The 1st guy took my ...
4876  Agree with Didi comment, food is good but to w...
```

```
# Manually set the land area for zip code 92154 since it is missing
sd_reviews.loc[sd_reviews['zip_code'] == 92154, 'land_area'] = 51.51

# Verify the changes
print(sd_reviews[sd_reviews['zip_code'] == 92154].head())
```

```
                restaurant_name  rating review_count Price_range cuisine_type  \
4872  Popeyes Louisiana Kitchen     3.6        1,338      $10–20      Chicken
4873  Popeyes Louisiana Kitchen     3.6        1,338      $10–20      Chicken
4874  Popeyes Louisiana Kitchen     3.6        1,338      $10–20      Chicken
4875  Popeyes Louisiana Kitchen     3.6        1,338      $10–20      Chicken
4876  Popeyes Louisiana Kitchen     3.6        1,338      $10–20      Chicken

      zip_code neighborhood land_area  population  num_households  \
4872     92154       Nestor     51.51     83758.0         23076.0
4873     92154       Nestor     51.51     83758.0         23076.0
4874     92154       Nestor     51.51     83758.0         23076.0
4875     92154       Nestor     51.51     83758.0         23076.0
4876     92154       Nestor     51.51     83758.0         23076.0

      median_income  average_income review_rating  \
4872        83287.0        101076.0        1 star
4873        83287.0        101076.0        1 star
4874        83287.0        101076.0        1 star
4875        83287.0        101076.0        1 star
4876        83287.0        101076.0        1 star

                                                review
4872  Popeyes over here selling fried chicken skin l...
4873  Fish sandwich. Don't waste your money. Even af...
4874  I order food on Door Dash. The amount of food ...
4875  Went thru the drive thru. The 1st guy took my ...
4876  Agree with Didi comment, food is good but to w...
```

```
sd_reviews.head()
```

	restaurant_name	rating	review_count	Price_range	cuisine_type	zip_code	neighborhood	land_area	population	num_households	median_inc
0	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	10.48 sq mi	31121.0	11383.0	1058
1	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	10.48 sq mi	31121.0	11383.0	1058
2	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	10.48 sq mi	31121.0	11383.0	1058
3	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	10.48 sq mi	31121.0	11383.0	1058
4	Taco Bell	3.0	1,017	$10–20	Fast Food	92142	Tierrasanta	10.48 sq mi	31121.0	11383.0	1058

Next steps: Generate code with sd_reviews | View recommended plots | New interactive sheet

```
# Remove "sq mi" from "land_area" column
sd_reviews['land_area'] = sd_reviews['land_area'].str.replace('sq mi', '', regex=False)
```

6) Altering the price_range column by assigning categories using binary values.

```
# Find the unique values for "Price_range" column
unique_price_ranges = sd_reviews['Price_range'].unique()
print("Unique Price Ranges:", unique_price_ranges)

# Display the NaN rows for Price_range column
nan_price_range_rows = sd_reviews[sd_reviews['Price_range'].isna()]
print(nan_price_range_rows)
```

```
    42877       4 stars  We enjoyed our brunch… There is a wait so it's...
    42878       3 stars  Food was good but for the price I expected mor...
    43247       5 stars  Definitely recommend!! Such an amazing place f...
    43248       4 stars  Cheap place for outdoor eats near the water. C...
    43495       5 stars  Food and service are great here. The portions ...
    43496        1 star  It's sad that this is the ONLY fast food left ...
    48508       5 stars  Well, that was a mind blowing experience. From...
    48509        1 star  7/26\nI went to this location on the DT I orde...
    49271       5 stars  Great little place, great bread, great service...
    49272        1 star  To begin with, the place and tables were dirty...
    52396       4 stars  My Indian friend recommended this place for bi...
    52397       5 stars  I like breakfast.\n\nFood\nWe had the breakfas...
    53276        1 star  Any kind of sweets is a big big no from here. ...
    53277       5 stars  Good price. Taste ok. Spacey\nCooked Oyster in...
    53390        1 star  I got the surf and turf taco which was almost ...
    53391       4 stars  Liked them better when they were always open u...
    53436        1 star  The worst indian food i have ever had! First o...
    53437       5 stars  Royal India's authentic Indian cuisine is unbe...
    56720       2 stars  We ordered chicken pho and chicken bun (b49). ...
    56721       3 stars  This location was hot inside. No air condition...
    56740       5 stars  Please do not let the negative reviews keep yo...
    56741       5 stars  Come here a couple times in the last few days....
    57215       4 stars  Was okay. The only thing that really bothered ...
    57216           NaN                                                NaN
    57243       3 stars  Really wanted to like this place cause it's th...
    57244       5 stars  Beautiful location overlooking the water. Ther...
    57267       4 stars  The best soy sauce fried noodles 豉油皇炒面(not lis...
    57268       5 stars  OMG the fries here are amazing.  Burgers are g...
```

```
# Show number of rows with NaN for Price_range column
nan_price_range_count = nan_price_range_rows.shape[0]
print("Number of rows with NaN for Price_range:", nan_price_range_count)

# There are only 38 rows and most of them are supermarkets instead of restaurants,
# so I will remove them.
sd_reviews = sd_reviews.dropna(subset=['Price_range'])
```

```
Number of rows with NaN for Price_range: 38
```

```
# Assign categories and binary values to the prices

# Cheap (1) = $ = $1-10 price range
# Affordable (2) = $$ = $10-20 price range
# Moderately Priced (3) = $$$ = $20-30 price range
# Expensive (4) = $$$$ = $30-50 or $50-100
# Extremely Expensive (5) = $$$$$ = $100+

# Assign categories and binary values to the prices
# Define a dictionary to map original values to binary values
price_range_mapping = {
    '$': 1,
    '$1–10': 1,
    '$$': 2,
    '$10–20': 2,
    '$$$': 3,
    '$20–30': 3,  # Updated to '$20–30'
    '$$$$': 4,
    '$30–50': 4,  # Updated to '$30–50'
    '$50–100': 4,  # Updated to '$50–100'
    '$$$$$': 5,
    '$100+': 5
}

# Use .loc for explicit assignment
for original_value, binary_value in price_range_mapping.items():
    sd_reviews.loc[sd_reviews['Price_range'] == original_value, 'Price_range'] = binary_value

# Verify the changes
unique_price_ranges = sd_reviews['Price_range'].unique()
print("Unique Price Ranges:", unique_price_ranges)

# Change name of "Price_range" column to "price_level"
sd_reviews = sd_reviews.rename(columns={'Price_range': 'price_level'})
```

```
Unique Price Ranges: [2 1 3 4 5]
```

```
sd_reviews.head()
```

	restaurant_name	rating	review_count	price_level	cuisine_type	zip_code	neighborhood	land_area	population	num_households	median_inc
0	Taco Bell	3.0	1,017	2	Fast Food	92142	Tierrasanta	10.48	31121.0	11383.0	1058
1	Taco Bell	3.0	1,017	2	Fast Food	92142	Tierrasanta	10.48	31121.0	11383.0	1058
2	Taco Bell	3.0	1,017	2	Fast Food	92142	Tierrasanta	10.48	31121.0	11383.0	1058
3	Taco Bell	3.0	1,017	2	Fast Food	92142	Tierrasanta	10.48	31121.0	11383.0	1058
4	Taco Bell	3.0	1,017	2	Fast Food	92142	Tierrasanta	10.48	31121.0	11383.0	1058

Next steps: Generate code with `sd_reviews` View recommended plots New interactive sheet

7) Performing a final cleanup to the dataset and making the following revisions:

- Remove "zip_code" column
- Make sure all numerical columns are classified as "int" or "float"
- Create columns for neighborhood population density and restaurant success level
- Make "review_rating" a numerical column by removing the words "star" and "stars"
- Altering "cuisine_type" column by assigning appropriate categories to the food served by each restaurant

```
# Remove "zip_code" column
sd_reviews = sd_reviews.drop(columns=['zip_code'])

# Remove "star" and "stars" from review_rating column
def extract_rating_number(rating_text):
    """Extracts the rating number from text containing 'star' or 'stars'."""
    try:
        # Split the string by spaces and take the first element (assumed to be the number)
        rating_number = rating_text.split()[0]
        # Convert the extracted number to an integer
        return int(rating_number)
    except (AttributeError, IndexError, ValueError):
        # Handle cases where the rating text is not in the expected format
        return None  # Or any other value you want to use for invalid ratings

# Apply the function to the 'review_rating' column
sd_reviews['review_rating'] = sd_reviews['review_rating'].apply(extract_rating_number)

# Remove rows with None values in review_rating if desired
sd_reviews = sd_reviews.dropna(subset=['review_rating'])
sd_reviews['review_rating'] = sd_reviews['review_rating'].astype(int)

# List of columns to convert to int
columns_to_convert = ['population', 'num_households', 'review_count', 'median_income', 'average_income']

# Convert columns to int, removing commas if present and handling NaN values
for column in columns_to_convert:
    if sd_reviews[column].dtype == object:  # Check if the column is of object type (likely string)
        # Replace commas and convert to numeric, then fill NaN with 0 and convert to int
        sd_reviews[column] = pd.to_numeric(sd_reviews[column].str.replace(',', ''), errors='coerce').fillna(0).astype(int)
    else:
        # Fill NaN with 0 and convert to int if not string
        sd_reviews[column] = pd.to_numeric(sd_reviews[column], errors='coerce').fillna(0).astype(int)

# Calculate population density and add the column
# Convert 'land_area' to numeric before performing division
sd_reviews['land_area'] = pd.to_numeric(sd_reviews['land_area'], errors='coerce')
sd_reviews['pop_density'] = sd_reviews['population'] / sd_reviews['land_area']

# Import numpy
import numpy as np # This line imports the NumPy library.

# Replace infinite values with NaN
sd_reviews['pop_density'] = sd_reviews['pop_density'].replace([np.inf, -np.inf], np.nan)

# Round pop_density to the nearest whole number and fill NaN with 0
sd_reviews['pop_density'] = sd_reviews['pop_density'].round().fillna(0).astype(int)

# Reorder columns to place 'pop_density' after 'population'
cols = list(sd_reviews.columns)
pop_index = cols.index('population')
cols.insert(pop_index + 1, 'pop_density')  # Insert after 'population'
cols.pop(cols.index('pop_density', pop_index + 2))  #Remove original 'pop_density' column from the end (if it exists)
sd_reviews = sd_reviews[cols]
```

```
<ipython-input-24-c3056a658a56>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  sd_reviews['review_rating'] = sd_reviews['review_rating'].astype(int)
<ipython-input-24-c3056a658a56>:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  sd_reviews[column] = pd.to_numeric(sd_reviews[column], errors='coerce').fillna(0).astype(int)
<ipython-input-24-c3056a658a56>:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  sd_reviews[column] = pd.to_numeric(sd_reviews[column], errors='coerce').fillna(0).astype(int)
```

Creating a "success_level" column that measures each restaurants success based on the following criteria:

Category 5 (Highly Successful):

• High rating (4.5 and above), high review count (above 75th percentile)

- **Binary Value** = 5

Category 4 (Successful)

• Moderately high rating (4.0 to 4.4 stars), high review count (above 75th percentile)

• High rating (4.5 and above), moderately high review count (between 50th to 75th percentile)

- **Binary Value** = 4

Category 3 (Average)

• Very low rating (less than 3.0 stars), moderate to high review count (above 50th percentile)

• Low rating (3.0 to 3.4 stars), low to high review count (be 75th percentile)

• Average rating (3.5 to 3.9 stars), moderate to high review count (above 50th percentile)

• Moderately high rating (4.0 to 4.4 stars), low to high review count (between 25th to 75th percentile)

• High rating (4.5 and above), low to moderate review count (between 25th and 50th percentile)

- **Binary Value** = 3

Category 2 (Struggling)

• Very low rating (less than 3.0 stars), low to moderate review count (25th to 50th percentile)

• Low rating (3.0 to 3.4 stars), very low to low review count (between 10th and 25th percentile)

• Average rating (3.5 to 3.9 stars), very low to moderate review count (below 50th percentile)

• Moderately high rating (4.0 to 4.4 stars), very low to low review count review count (below 25th percentile)

• High rating (4.5 and above), very low to low review count (below 25th percentile)

- **Binary Value** = 2

Category 1 (Unsuccessful)

• Very low rating (less than 3.0 stars), very low to low review count (below 25th percentile)

• Low rating (3.0 to 3.4 stars), very low review count (below 10th percentile)

• Average rating (3.5 to 3.9 stars), very low review count (below 10th percentile)

- **Binary Value** = 1

Each category will be represented by a binary value in the data set.

```
import numpy as np

# Calculate percentiles for review_count
review_count_25th = sd_reviews['review_count'].quantile(0.25)
review_count_50th = sd_reviews['review_count'].quantile(0.50)
review_count_75th = sd_reviews['review_count'].quantile(0.75)
review_count_10th = sd_reviews['review_count'].quantile(0.10)


# Create the 'success_level' column and initialize with NaN
sd_reviews['success_level'] = np.nan

# Category 5 (Highly Successful)
sd_reviews.loc[
    (sd_reviews['rating'] >= 4.5) & (sd_reviews['review_count'] > review_count_75th),
    'success_level',
] = 5

# Category 4 (Successful)
sd_reviews.loc[
    ((sd_reviews['rating'] >= 4.0) & (sd_reviews['rating'] < 4.5) & (sd_reviews['review_count'] > review_count_75th)) |
    ((sd_reviews['rating'] >= 4.5) & (sd_reviews['review_count'] > review_count_50th) & (sd_reviews['review_count'] <= review_count_75th)),
    'success_level',
] = 4

# Category 3 (Average)
sd_reviews.loc[
    ((sd_reviews['rating'] < 3.0) & (sd_reviews['review_count'] > review_count_50th)) |
    ((sd_reviews['rating'] >= 3.0) & (sd_reviews['rating'] < 3.5) & (sd_reviews['review_count'] <= review_count_75th)) |
    ((sd_reviews['rating'] >= 3.5) & (sd_reviews['rating'] < 4.0) & (sd_reviews['review_count'] > review_count_50th)) |
    ((sd_reviews['rating'] >= 4.0) & (sd_reviews['rating'] < 4.5) & (sd_reviews['review_count'] > review_count_25th) & (sd_reviews['review_count'
    ((sd_reviews['rating'] >= 4.5) & (sd_reviews['review_count'] > review_count_25th) & (sd_reviews['review_count'] <= review_count_50th)),
    'success_level',
] = 3


# Category 2 (Struggling)
sd_reviews.loc[
    ((sd_reviews['rating'] < 3.0) & (sd_reviews['review_count'] > review_count_25th) & (sd_reviews['review_count'] <= review_count_50th)) |
    ((sd_reviews['rating'] >= 3.0) & (sd_reviews['rating'] < 3.5) & (sd_reviews['review_count'] > review_count_10th) & (sd_reviews['review_count'
    ((sd_reviews['rating'] >= 3.5) & (sd_reviews['rating'] < 4.0) & (sd_reviews['review_count'] <= review_count_50th)) |
    ((sd_reviews['rating'] >= 4.0) & (sd_reviews['rating'] < 4.5) & (sd_reviews['review_count'] <= review_count_25th)) |
    ((sd_reviews['rating'] >= 4.5) & (sd_reviews['review_count'] <= review_count_25th)),
    'success_level',
] = 2


# Category 1 (Unsuccessful)
sd_reviews.loc[
    ((sd_reviews['rating'] < 3.0) & (sd_reviews['review_count'] <= review_count_25th)) |
    ((sd_reviews['rating'] >= 3.0) & (sd_reviews['rating'] < 3.5) & (sd_reviews['review_count'] <= review_count_10th)) |
    ((sd_reviews['rating'] >= 3.5) & (sd_reviews['rating'] < 4.0) & (sd_reviews['review_count'] <= review_count_10th)),
    'success_level',
] = 1

# Convert 'success_level' to integer type
sd_reviews['success_level'] = sd_reviews['success_level'].astype(int)

# Count the occurrences of each success level to confirm code is written correctly
success_level_counts = sd_reviews['success_level'].value_counts()

# Display the counts
print(success_level_counts)
```

```
success_level
3    19336
2    13778
4     8961
1     6177
```

```
5     5832
Name: count, dtype: int64
<ipython-input-25-94e35453c89a>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  sd_reviews['success_level'] = np.nan
<ipython-input-25-94e35453c89a>:57: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  sd_reviews['success_level'] = sd_reviews['success_level'].astype(int)
```

```
sd_reviews.head()
```

	restaurant_name	rating	review_count	price_level	cuisine_type	neighborhood	land_area	population	pop_density	num_households	median_
0	Taco Bell	3.0	1017	2	Fast Food	Tierrasanta	10.48	31121	2970	11383	
1	Taco Bell	3.0	1017	2	Fast Food	Tierrasanta	10.48	31121	2970	11383	
2	Taco Bell	3.0	1017	2	Fast Food	Tierrasanta	10.48	31121	2970	11383	
3	Taco Bell	3.0	1017	2	Fast Food	Tierrasanta	10.48	31121	2970	11383	
4	Taco Bell	3.0	1017	2	Fast Food	Tierrasanta	10.48	31121	2970	11383	

Next steps: Generate code with sd_reviews | View recommended plots | New interactive sheet

```
# Get the unique cuisine categories
unique_cuisine_categories = sd_reviews['cuisine_type'].unique()

# Get the number of unique cuisine categories
num_unique_cuisine_categories = len(unique_cuisine_categories)

# Print the number of unique cuisine categories
print("Number of unique cuisine categories:", num_unique_cuisine_categories)
```

```
Number of unique cuisine categories: 101
```

```
# There are 101 different cusine categories.
# Consolidating the categories into broader types
# Function to categorize cuisine types
def categorize_cuisine(cuisine):
    if cuisine in ['Japanese', 'Sushi', 'Ramen', 'Noodles', 'Chinese', 'Thai', 'Korean', 'Vietnamese', 'Taiwanese', 'Filipino', 'Asian Fusion', '
        return 'Asian'
    elif cuisine in ['American (Traditional)', 'Burgers', 'Sandwiches', 'Fast Food', 'Hot Dogs', 'Chicken Wings', 'Bar Food', 'American (New)', '
        return 'American'
    # Updated: Added Pizza to Italian
    elif cuisine in ['Italian', 'Pizza', 'Pizza Delivery']:
        return 'Italian'
    # Updated: Added Greek to Mediterranean
    elif cuisine in ['Mediterranean', 'Greek']:
        return 'Mediterranean'
    elif cuisine in ['French']:
        return 'French'
    # Updated: Added Spanish to Latin American
    elif cuisine in ['Mexican', 'Argentinian','Latin American', 'Caribbean', 'Cuban', 'Salvadoran', 'Spanish']:
        return 'Latin American'
    elif cuisine in ['Portuguese']:
        return 'Portuguese'
    elif cuisine in ['Turkish']:
        return 'Turkish'
    elif cuisine in ['Middle Eastern', 'Lebanese']:
        return 'Middle Eastern'
    elif cuisine in ['Cafes', 'Coffee & Tea', 'Bakeries', 'Desserts', 'Ice Cream & Frozen Yogurt']:
        return 'Cafes & Desserts'
    elif cuisine in ['Vegetarian', 'Vegan', 'Gluten-Free', 'Healthy']:
        return 'Health-conscious'
    else:
        return 'Other'  # For any uncategorized cuisines

# Applying the categorization function
sd_reviews['cuisine_category'] = sd_reviews['cuisine_type'].apply(categorize_cuisine)

# Printing unique values in "cuisine_category"
unique_categories = sd_reviews['cuisine_category'].unique()

print("Unique Cuisine Categories:")
for category in unique_categories:
    print(category)
```

```
Unique Cuisine Categories:
American
Latin American
Italian
Other
Asian
Mediterranean
Middle Eastern
Health-conscious
French
```

```
# Giving the restaurants labeled "Other" specific categories
# Function to categorize based on restaurant name
def categorize_by_name(name):
    name_lower = name.lower() # Convert to lowercase for case-insensitive matching
    if any(keyword in name_lower for keyword in ['vietnamese', 'pho', 'banh mi', 'thai', 'chinese', 'japanese', 'korean', 'sushi', 'ramen', 'indi
        return 'Asian'
    elif any(keyword in name_lower for keyword in ['italian', 'pizza', 'pasta', 'trattoria', 'pizzeria']):
        return 'Italian'
    elif any(keyword in name_lower for keyword in ['mexican', 'taco', 'burrito', 'taqueria', 'cantina']):
        return 'Latin American'
    elif any(keyword in name_lower for keyword in ['burger', 'fries', 'american', 'grill', 'diner', 'bbq', 'steak']):
        return 'American'
    elif any(keyword in name_lower for keyword in ['cafe', 'coffee', 'bakery', 'dessert', 'ice cream']):
        return 'Cafes & Desserts'
    elif any(keyword in name_lower for keyword in ['mediterranean', 'greek', 'gyro', 'falafel']):
        return 'Mediterranean'
    # Add more conditions as needed for other cuisines based on name patterns
    else:
        return 'Other' # If no keywords match, keep as 'Other'

# Filtering restaurants categorized as "Other"
other_restaurants = sd_reviews[sd_reviews['cuisine_category'] == 'Other']

# Applying the name-based categorization to the filtered restaurants
other_restaurants['cuisine_category'] = other_restaurants['restaurant_name'].apply(categorize_by_name)

# Updating the original DataFrame with the new categories
sd_reviews.update(other_restaurants) # Update sd_reviews with changes in other_restaurants

# Getting unique values in "cuisine_category"
unique_categories = sd_reviews['cuisine_category'].unique()

# Printing the unique values
print("Unique Cuisine Categories:")
for category in unique_categories:
    print(category)
```

```
Unique Cuisine Categories:
American
Latin American
Italian
Other
Asian
```

```
Cafes & Desserts
Mediterranean
Middle Eastern
Health-conscious
French
<ipython-input-29-04492ca463d2>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  other_restaurants['cuisine_category'] = other_restaurants['restaurant_name'].apply(categorize_by_name)
```

```
# Replacing cuisine_type with new cuisine_category column, then renaming it back to cuisine_type
sd_reviews['cuisine_type'] = sd_reviews['cuisine_category']

sd_reviews = sd_reviews.drop(columns=['cuisine_category'])


sd_reviews.head()
```

	restaurant_name	rating	review_count	price_level	cuisine_type	neighborhood	land_area	population	pop_density	num_households	median_
0	Taco Bell	3.0	1017	2	American	Tierrasanta	10.48	31121	2970	11383	
1	Taco Bell	3.0	1017	2	American	Tierrasanta	10.48	31121	2970	11383	
2	Taco Bell	3.0	1017	2	American	Tierrasanta	10.48	31121	2970	11383	
3	Taco Bell	3.0	1017	2	American	Tierrasanta	10.48	31121	2970	11383	
4	Taco Bell	3.0	1017	2	American	Tierrasanta	10.48	31121	2970	11383	

Next steps: Generate code with sd_reviews View recommended plots New interactive sheet

8) Now that the final sd_reviews data set is complete, I will perform one final cleanup of the dataset by handling missing data and outliers.

```
# Checking for missing values
missing_values = sd_reviews.isnull().sum()
print(missing_values)

# Display rows that are missing values for neighborhood column
missing_neighborhood_rows = sd_reviews[sd_reviews['neighborhood'].isnull()]
display(missing_neighborhood_rows)

# Display rows that are missing values for land_area column
missing_land_area_rows = sd_reviews[sd_reviews['land_area'].isnull()]
display(missing_land_area_rows)

# Manually set the land area for all rows with neighborhood 'Nestor'
sd_reviews.loc[sd_reviews['neighborhood'] == 'Nestor', 'land_area'] = 51.51

# Display unique neighborhood values
unique_neighborhoods = sd_reviews['neighborhood'].unique()
print(unique_neighborhoods)

# Fix names for Downtown neighborhood
```

```
sd_reviews['neighborhood'] = sd_reviews['neighborhood'].replace(['Downtown, San Diego', 'San Diego'], 'Downtown')

# Verify the changes by displaying unique neighborhood values
print(sd_reviews['neighborhood'].unique())

# Display neighborhood names that are missing values for land_area
missing_land_area_neighborhoods = sd_reviews[sd_reviews['land_area'].isnull()]['neighborhood'].unique()
print(missing_land_area_neighborhoods)

# Manually enter land area for Downtown, San Diego
# Set land_area to 324.3 for Downtown neighborhood
sd_reviews.loc[sd_reviews['neighborhood'] == 'Downtown', 'land_area'] = 324.3

# Verify the changes by displaying a few rows with neighborhood 'Downtown'
print(sd_reviews[sd_reviews['neighborhood'] == 'Downtown'][['neighborhood', 'land_area']].head())

# There are now 301 rows missing both neighborhood and land_area. Display rows
missing_values = sd_reviews.isnull().sum()
print(missing_values)

# There are only two restaurants that are missing neighborhood and land_area values.
# Researched these restaurants and discovered that they do not belong to San Diego but
# are located in the outskirts of San Diego
# Removed these restaurants from the data since they don't belong to San Diego anyway
# Only 301 rows are removed, which should not affect the data
# Remove rows with missing values in both 'land_area' and 'neighborhood'
sd_reviews = sd_reviews.dropna(subset=['land_area', 'neighborhood'])

# Display the shape of the DataFrame to check the number of rows removed
print(sd_reviews.shape)
```

```
restaurant_name        0
rating                 0
review_count           0
price_level            0
cuisine_type           0
neighborhood         301
land_area           6731
population             0
pop_density            0
num_households         0
median_income          0
average_income         0
review_rating          0
review                 0
success_level          0
dtype: int64
```

	restaurant_name	rating	review_count	price_level	cuisine_type	neighborhood	land_area	population	pop_density	num_households
39607	Farmer's Table La Mesa	4.2	3013	3	Other	NaN	NaN	0	0	0
39608	Farmer's Table La Mesa	4.2	3013	3	Other	NaN	NaN	0	0	0
39609	Farmer's Table La Mesa	4.2	3013	3	Other	NaN	NaN	0	0	0
39610	Farmer's Table La Mesa	4.2	3013	3	Other	NaN	NaN	0	0	0
39611	Farmer's Table La Mesa	4.2	3013	3	Other	NaN	NaN	0	0	0
...	...	...	...	...	...	...	...	...	...	...
39935	Farmer's Table La Mesa	4.2	3013	3	Other	NaN	NaN	0	0	0
39936	Farmer's Table La Mesa	4.2	3013	3	Other	NaN	NaN	0	0	0
39937	Farmer's Table La Mesa	4.2	3013	3	Other	NaN	NaN	0	0	0
43505	Georges at the Cove	4.4	3852	3	Other	NaN	NaN	0	0	0
43506	Georges at the Cove	4.4	3852	3	Other	NaN	NaN	0	0	0

301 rows × 15 columns

	restaurant_name	rating	review_count	price_level	cuisine_type	neighborhood	land_area	population	pop_density	num_households
4872	Popeyes Louisiana Kitchen	3.6	1338	2	Other	Nestor	NaN	83758	0	23076

4873	Popeyes Louisiana Kitchen	3.6	1338	2	Other	Nestor	NaN	83758	0	23076
4874	Popeyes Louisiana Kitchen	3.6	1338	2	Other	Nestor	NaN	83758	0	23076
4875	Popeyes Louisiana Kitchen	3.6	1338	2	Other	Nestor	NaN	83758	0	23076
4876	Popeyes Louisiana Kitchen	3.6	1338	2	Other	Nestor	NaN	83758	0	23076
...	...	...	...	...	...	...	...	...	...	...
57897	In-N-Out Burger	4.7	2078	2	American	Nestor	NaN	83758	0	23076
57898	In-N-Out Burger	4.7	2078	2	American	Nestor	NaN	83758	0	23076
57899	In-N-Out Burger	4.7	2078	2	American	Nestor	NaN	83758	0	23076
58033	Ed Fernandez Restaurant Birrieria	4.9	4979	2	Latin American	Nestor	NaN	83758	0	23076
58034	Ed Fernandez Restaurant Birrieria	4.9	4979	2	Latin American	Nestor	NaN	83758	0	23076

6731 rows × 15 columns

```
['Tierrasanta' 'Old Town' 'University' 'San Diego' 'Serra Mesa' 'Nestor'
 'Mira Mesa' 'Hillcrest' 'Navajo' 'Mission Valley' 'Rancho Penasquitos'
 'Scripps Ranch' 'Sorrento Valley' 'City Heights' 'North Park'
 'Logan Heights' 'Downtown' 'College Grove' 'Clairemont'
 'Downtown, San Diego' 'Encanto' 'Point Loma' 'Grantville' 'Pacific Beach'
 'Linda Vista' 'Rancho Bernardo' 'Carmel Valley' 'Normal Heights' nan
 'Ocean Beach']
['Tierrasanta' 'Old Town' 'University' 'Downtown' 'Serra Mesa' 'Nestor'
 'Mira Mesa' 'Hillcrest' 'Navajo' 'Mission Valley' 'Rancho Penasquitos'
 'Scripps Ranch' 'Sorrento Valley' 'City Heights' 'North Park'
 'Logan Heights' 'College Grove' 'Clairemont' 'Encanto' 'Point Loma'
 'Grantville' 'Pacific Beach' 'Linda Vista' 'Rancho Bernardo'
 'Carmel Valley' 'Normal Heights' nan 'Ocean Beach']
['Downtown' nan]
     neighborhood  land_area
2031     Downtown      324.3
2032     Downtown      324.3
2033     Downtown      324.3
2034     Downtown      324.3
2035     Downtown      324.3
restaurant_name      0
rating               0
review_count         0
price_level          0
cuisine_type         0
neighborhood       301
land_area          301
population           0
pop_density          0
num_households       0
median_income        0
average_income       0
review_rating        0
review               0
success_level        0
dtype: int64
(53783, 15)
```

After performing data collection, cleaning, and preparation, the final sd_reviews data set that will be used for this project consists of 53,783 rows and 15 columns.

```
sd_reviews.head()
```

	restaurant_name	rating	review_count	price_level	cuisine_type	neighborhood	land_area	population	pop_density	num_households	median_
0	Taco Bell	3.0	1017	2	American	Tierrasanta	10.48	31121	2970	11383	
1	Taco Bell	3.0	1017	2	American	Tierrasanta	10.48	31121	2970	11383	
2	Taco Bell	3.0	1017	2	American	Tierrasanta	10.48	31121	2970	11383	
3	Taco Bell	3.0	1017	2	American	Tierrasanta	10.48	31121	2970	11383	
4	Taco Bell	3.0	1017	2	American	Tierrasanta	10.48	31121	2970	11383	

Next steps: Generate code with sd_reviews | View recommended plots | New interactive sheet

Addressing Project Goals

As briefly outlined in the introduction, this project is guided by four core goals, each designed to provide valuable insights and actionable recommendations for aspiring restaurant owners in San Diego. These goals are:

1) **Identify Key Success Factors for Restaurants in San Diego:** This goal lays the foundation by uncovering the critical elements that contribute to restaurant success in the competitive San Diego market.

2) **Determine Appropriate Menu Prices:** This goal addresses a fundamental early decision for new restaurant owners, guiding them on establishing effective pricing strategies that balance profitability and customer value.

3) **Recommend Optimal Locations for New Restaurants:** Building upon pricing considerations, this goal focuses on identifying the most promising locations within San Diego for establishing new restaurants, increasing the likelihood of success by considering factors like demographics, competition, and market demand.

4) **Find Strategies to Enhance Customer Experience:** This goal leverages insights from the previous goals to develop actionable strategies for improving the overall dining experience for customers in San Diego restaurants, enhancing satisfaction and loyalty.

Goal #1: Identify Key Success Factors for Restaurants in San Diego, California

Feature Selection:

Target Variable: success_level

Potential Features: rating, review_count, price_level, cuisine_type, population, pop_density, num_households, median_income, average_income

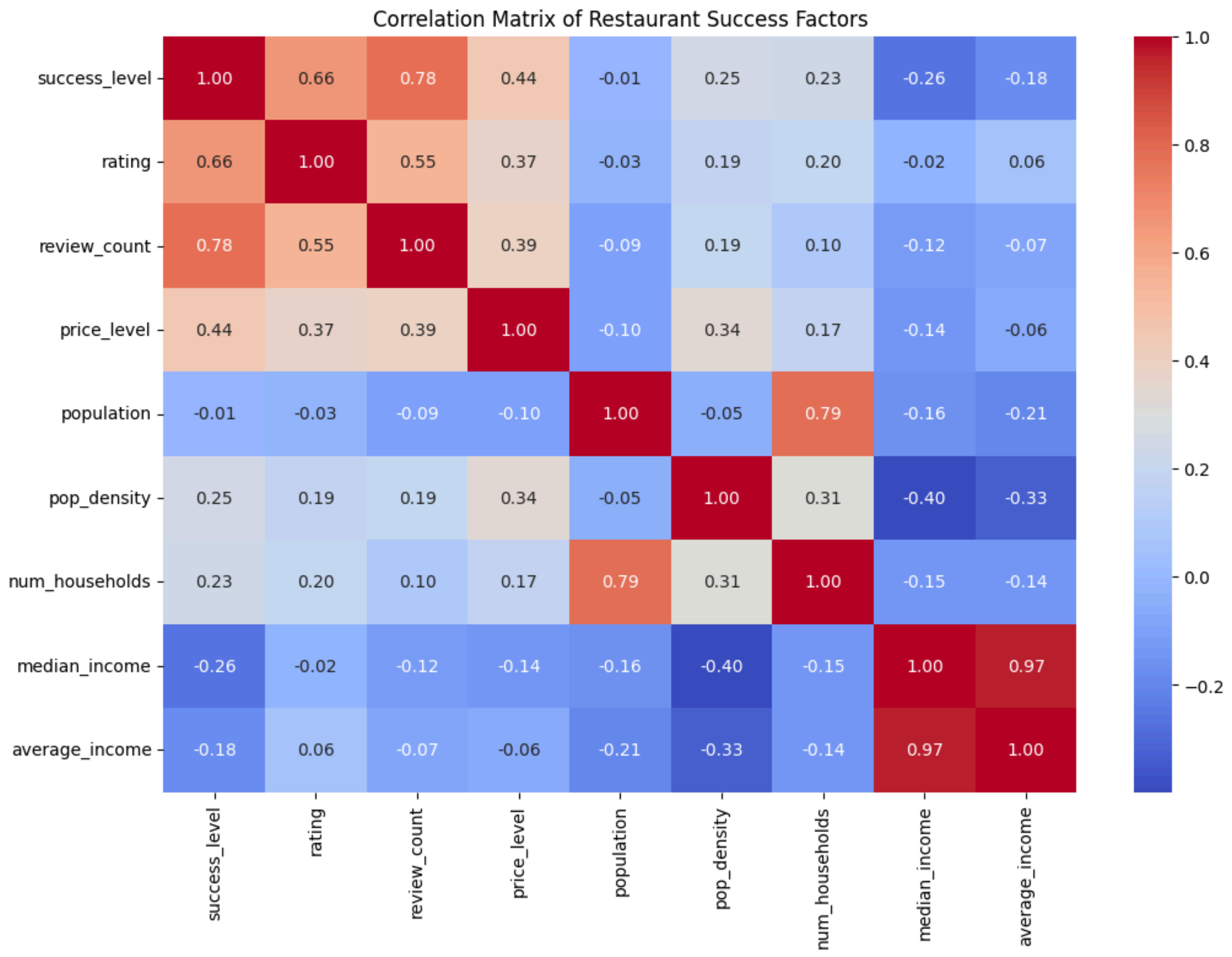
Methods for Identifying Key Success Factors:

1) **Correlation Analysis:** The below code calculates the correlation between each feature and success_level to identify potential relationships via a correlation matrix.

Results:

- **Strong Positive Correlation (0.78) between review_count and success_level:** This indicates that restaurants with more reviews tend to be more successful, although the relationship is not as strong as with rating. This could imply that a larger volume of customer feedback (regardless of individual ratings) contributes to success.
- **Strong Positive Correlation (0.66)** between **rating** and **success_level** columns indicates that restaurants with higher ratings tend to have higher levels of success. As a reminder, category 5 level of success (the highest level) signifies a restaurant with a very high rating and a large volume of reviews. Both are indicators of success because a large volume of reviews is representative of how many customers a restaurant regularly attracts and a high overall rating represents the reputation of a restaurant.
- **Moderate Positive Correlation (0.44) between price_level and success_level:** This suggests that price level may not be a significant factor in determining restaurant success in San Diego. This could imply that restaurants across different price points can achieve success if they meet other criteria.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix
correlation_matrix = sd_reviews[['success_level', 'rating', 'review_count', 'price_level', 'population', 'pop_density', 'num_households', 'median

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Restaurant Success Factors')
plt.show()
```



	success_level	rating	review_count	price_level	population	pop_density	num_households	median_income	average_income
success_level	1.00	0.66	0.78	0.44	-0.01	0.25	0.23	-0.26	-0.18
rating	0.66	1.00	0.55	0.37	-0.03	0.19	0.20	-0.02	0.06
review_count	0.78	0.55	1.00	0.39	-0.09	0.19	0.10	-0.12	-0.07
price_level	0.44	0.37	0.39	1.00	-0.10	0.34	0.17	-0.14	-0.06
population	-0.01	-0.03	-0.09	-0.10	1.00	-0.05	0.79	-0.16	-0.21
pop_density	0.25	0.19	0.19	0.34	-0.05	1.00	0.31	-0.40	-0.33
num_households	0.23	0.20	0.10	0.17	0.79	0.31	1.00	-0.15	-0.14
median_income	-0.26	-0.02	-0.12	-0.14	-0.16	-0.40	-0.15	1.00	0.97
average_income	-0.18	0.06	-0.07	-0.06	-0.21	-0.33	-0.14	0.97	1.00

2) Feature Importance Using Random Forest ML Model:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

```
# Select features and target variable
X = sd_reviews[['rating', 'review_count', 'price_level', 'population', 'pop_density', 'num_households', 'median_income', 'average_income']]
y = sd_reviews['success_level']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6, random_state=42)

# Train a Random Forest model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Get feature importances
feature_importances = model.feature_importances_

# Print feature importances
for i, feature in enumerate(X.columns):
    print(f"{feature}: {feature_importances[i]}")
```

```
rating: 0.26489700647331704
review_count: 0.44940110435660535
price_level: 0.056514444119559665
population: 0.03186800980289505
pop_density: 0.03138082622529737
num_households: 0.0389074038469417
median_income: 0.06768149543261764
average_income: 0.059349709742766155
```

Based on performing correlation analysis and building a Random Forest model, three key factors strongly linked with a restaurant's success in San Diego were identified: **review_count, rating, and price_level.**

- **review_count:** The number of customer reviews emerged as a significant indicator of success, suggesting that higher review volumes, regardless of individual ratings, contribute to restaurant visibility and overall performance. This emphasizes the importance of encouraging customer feedback and engagement to enhance a restaurant's online presence and attract potential diners.
- **rating:** Restaurant rating, as expected, played a significant role in predicting success. Higher ratings reflect greater customer satisfaction, which is fundamental for attracting and retaining customers in a competitive market like San Diego. Maintaining high-quality food, service, and ambiance are some of the most common aspects essential for achieving and sustaining positive ratings.
- **price_level:** While potentially less influential than review_count and rating, price_level showed a notable association with success. This indicates the importance of establishing an appropriate pricing strategy that aligns with customer expectations and the perceived value offered by the restaurant. Careful consideration of menu prices, target market, and competitive landscape is crucial for optimizing profitability and customer appeal.

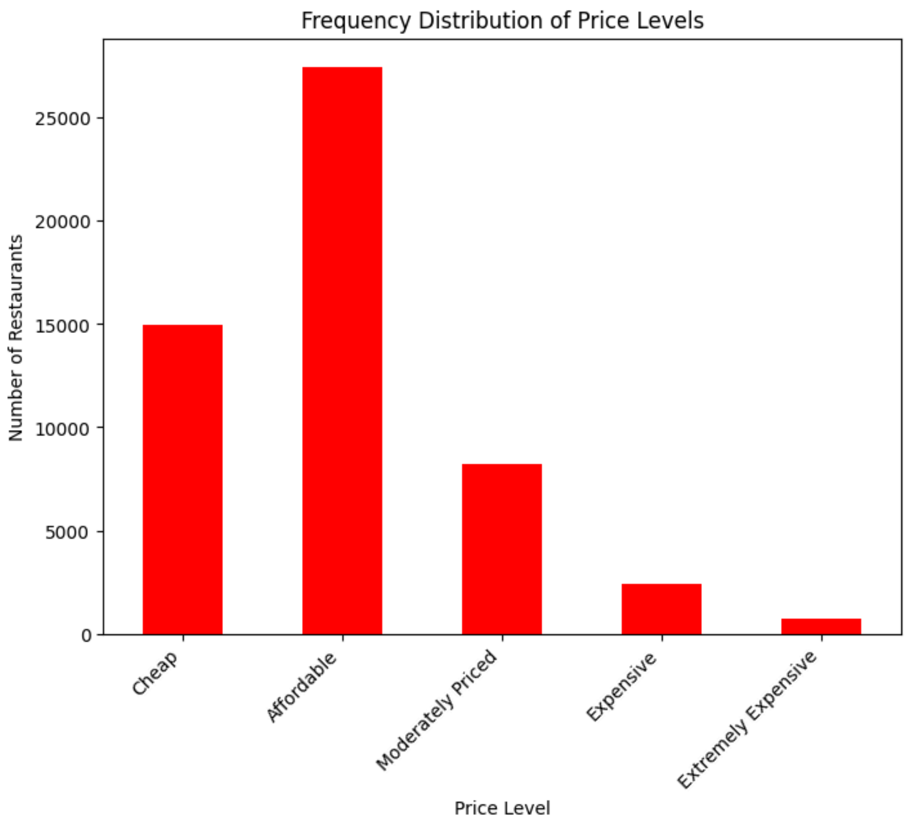
Goal #2: Determine Appropriate Menu Prices

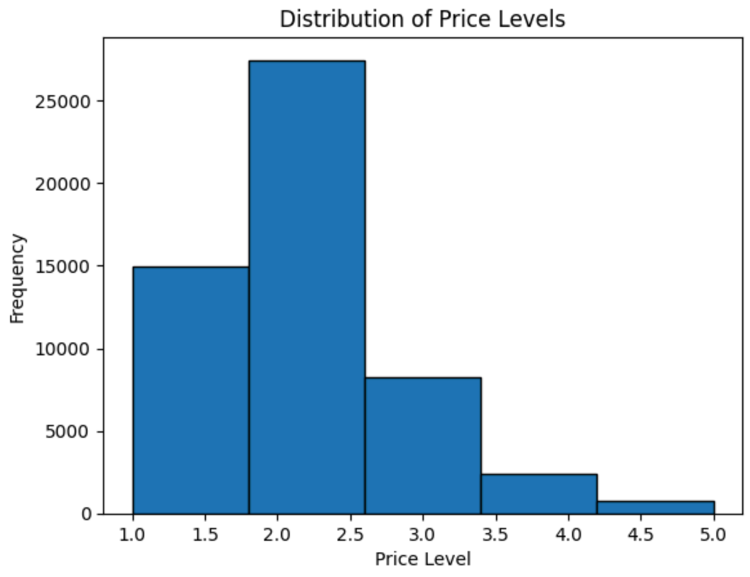
This goal focuses on establishing effective pricing strategies that balance profitability and customer value. By analyzing existing restaurant data and considering market dynamics, you aim to provide recommendations for setting appropriate menu prices for aspiring restaurant owners.

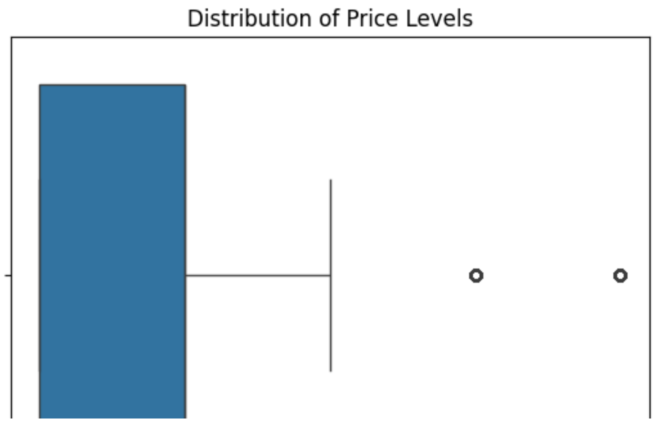
1) Explore existing price levels using sd_review dataset: the bar plot and histogram both show that the vast majority of restaurants in the data set are considered "affordable," with the typical meal ranging between $10-$ 20 dollars. The histogram is skewed to the right, further indicating a higher presence of affordable restaurants and a lower presence of expensive restaurants in San Diego.

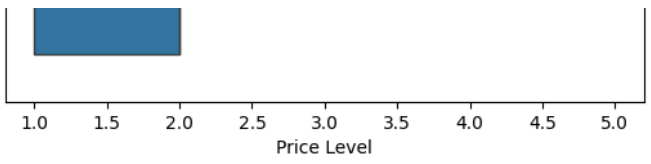
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Descriptive Statistics
price_level_stats = sd_reviews['price_level'].describe()
print(price_level_stats)

# Frequency Distribution (Bar Chart) with Original Categories and Red Color
price_level_counts = sd_reviews['price_level'].value_counts().sort_index()

# Create a dictionary mapping price levels to original categories (without dollar signs)
price_level_categories = {
    1: 'Cheap',
    2: 'Affordable',
    3: 'Moderately Priced',
    4: 'Expensive',
    5: 'Extremely Expensive'
}

# Create the bar plot with original categories as x-axis labels and red color
price_level_counts.plot(kind='bar', figsize=(8, 6), color='red')  # Set color to red
plt.title('Frequency Distribution of Price Levels')
plt.xlabel('Price Level')
plt.ylabel('Number of Restaurants')

# Convert index to numeric before plotting
price_level_counts.index = pd.to_numeric(price_level_counts.index)

# Set x-axis labels using the price_level_categories dictionary
plt.xticks(
    ticks=price_level_counts.index - 1,  # Adjust index for bar plot
    labels=[price_level_categories[level] for level in price_level_counts.index],
    rotation=45,  # Rotate labels if needed
    ha='right'  # Align labels to the right
)

plt.show()

# Visualization (Histogram) - You can keep or remove this as needed
plt.hist(sd_reviews['price_level'], bins=5, edgecolor='black')
plt.title('Distribution of Price Levels')
plt.xlabel('Price Level')
plt.ylabel('Frequency')
plt.show()

# Visualization (Box Plot) - Optional, you can keep or remove this
sns.boxplot(x='price_level', data=sd_reviews)
plt.title('Distribution of Price Levels')
plt.xlabel('Price Level')
plt.show()
```

```
count      53783
unique         5
top            2
freq       27468
Name: price_level, dtype: int64
```



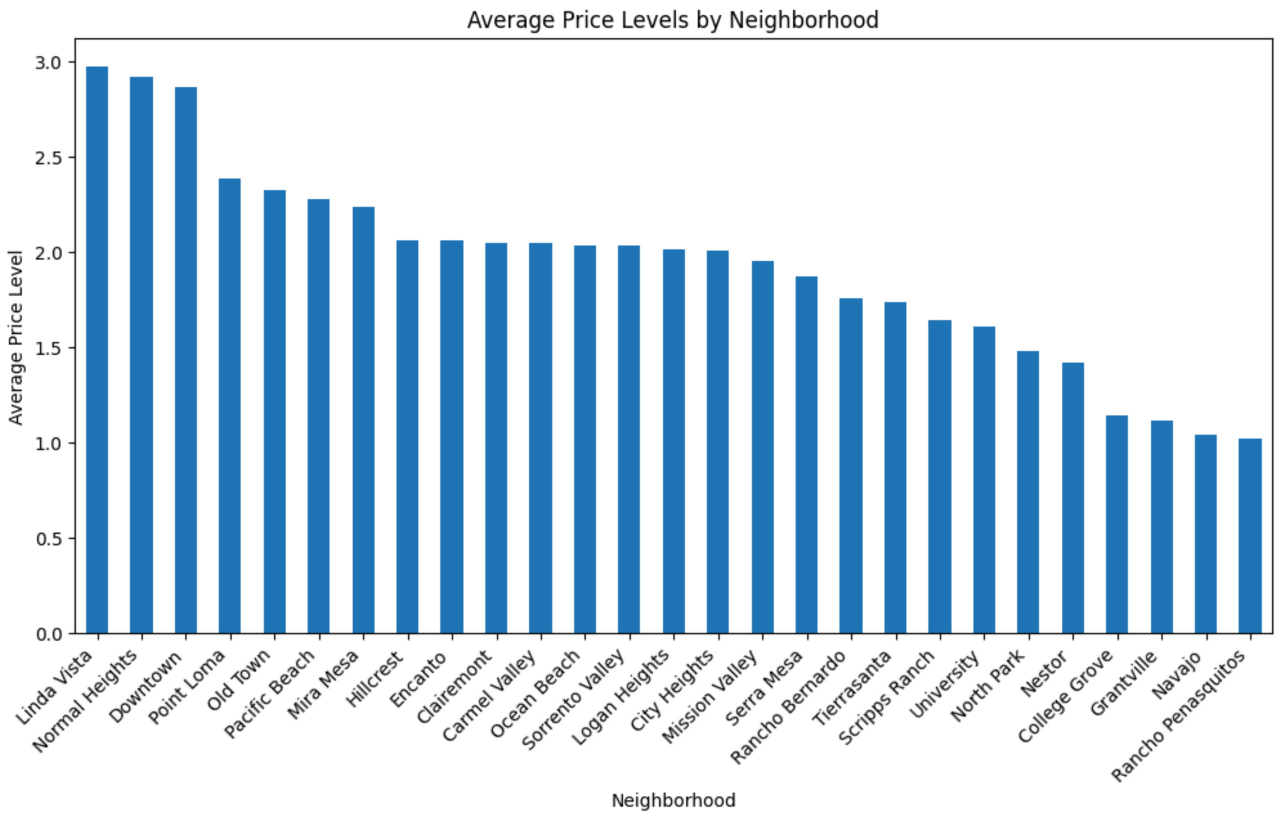


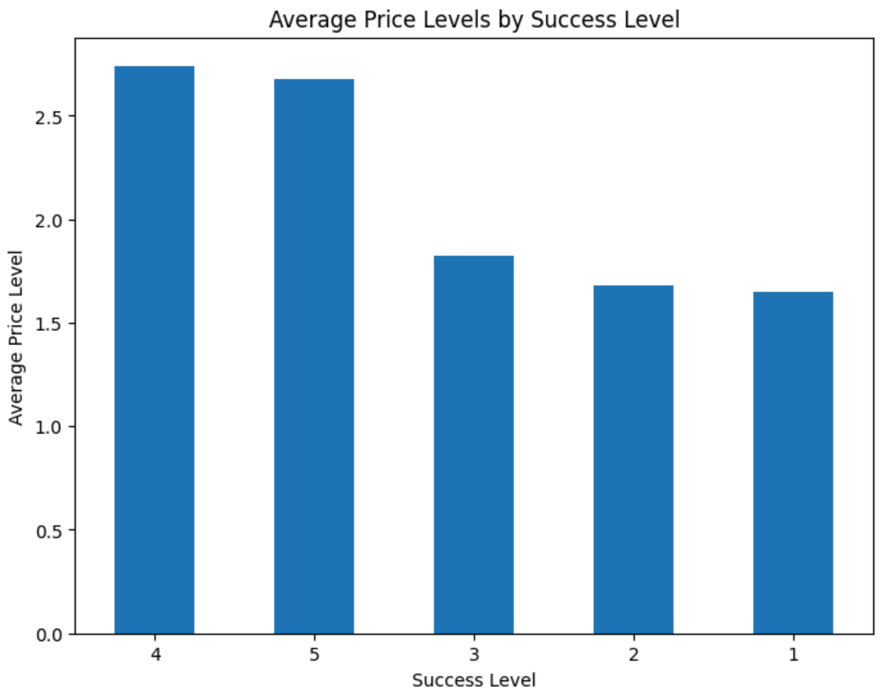




2) Identify Price-Related Factors: The below graphs indicate that the top 5 restaurants with the priciest restaurants in San Diego are Linda Vista, Normal Heights, Downtown, Point Loma, and Old Town. The Heatmap also indicates that there is a slight positive correlation between neighborhoods of higher population density and higher priced restaurants. Additionally, it is shown that the most successful restaurants tend to have higher prices.

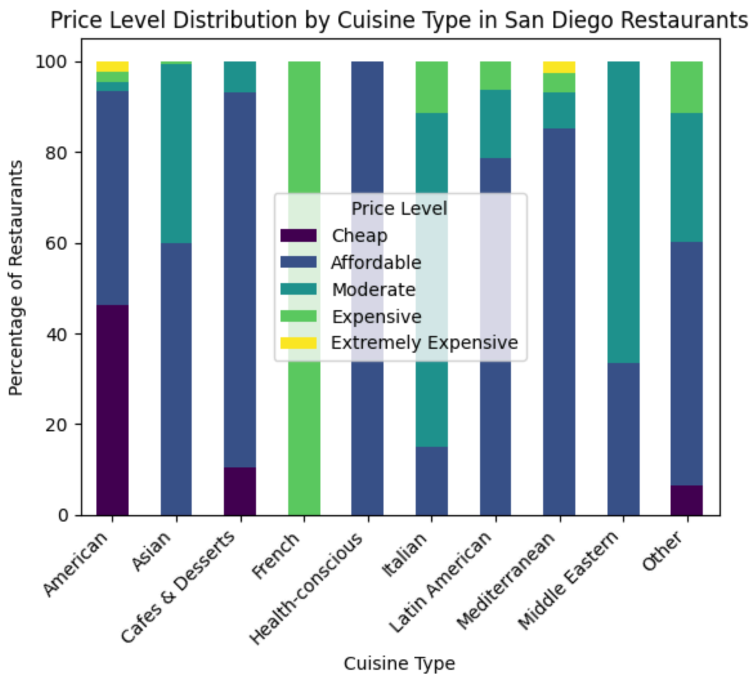
```
# Average price level by neighborhood
neighborhood_prices = sd_reviews.groupby('neighborhood')['price_level'].mean().sort_values(ascending=False)
neighborhood_prices.plot(kind='bar', figsize=(12, 6))
plt.title('Average Price Levels by Neighborhood')
plt.xlabel('Neighborhood')
plt.ylabel('Average Price Level')
plt.xticks(rotation=45, ha='right')
plt.show()

# Average price level by success level
success_level_prices = sd_reviews.groupby('success_level')['price_level'].mean().sort_values(ascending=False)
success_level_prices.plot(kind='bar', figsize=(8, 6))
plt.title('Average Price Levels by Success Level')
plt.xlabel('Success Level')
plt.ylabel('Average Price Level')
plt.xticks(rotation=0)  # Keep x-axis labels horizontal
plt.show()
```



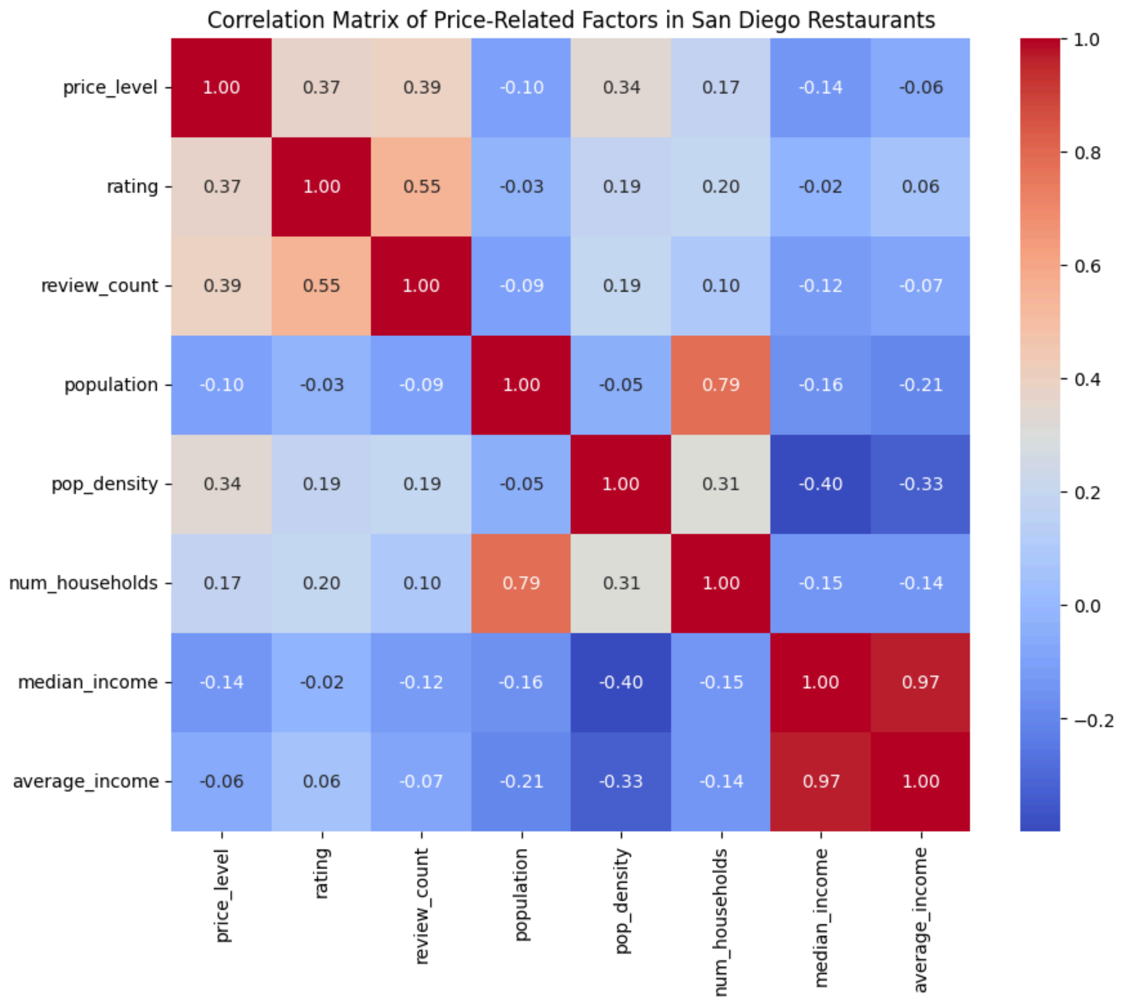


```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a cross-tabulation of cuisine type and price level
cuisine_price_dist = pd.crosstab(sd_reviews['cuisine_type'], sd_reviews['price_level'])

# Normalize the cross-tabulation to get percentages
cuisine_price_dist_pct = cuisine_price_dist.div(cuisine_price_dist.sum(axis=1), axis=0) * 100

# Create the bar plot
plt.figure(figsize=(12, 6))  # Adjust figure size as needed
cuisine_price_dist_pct.plot(kind='bar', stacked=True, colormap='viridis')
plt.title('Price Level Distribution by Cuisine Type in San Diego Restaurants')
plt.xlabel('Cuisine Type')
plt.ylabel('Percentage of Restaurants')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.legend(title='Price Level', labels=['Cheap', 'Affordable', 'Moderate', 'Expensive', 'Extremely Expensive'])
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



```
# Select numerical features for correlation analysis
numerical_features = ['price_level', 'rating', 'review_count', 'population',
                      'pop_density', 'num_households', 'median_income', 'average_income']

# Calculate correlations
correlations = sd_reviews[numerical_features].corr()

# Create the heatmap
plt.figure(figsize=(10, 8))  # Adjust figure size as needed
sns.heatmap(correlations, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Price-Related Factors in San Diego Restaurants')
plt.show()
```



	price_level	rating	review_count	population	pop_density	num_households	median_income	average_income
price_level	1.00	0.37	0.39	-0.10	0.34	0.17	-0.14	-0.06
rating	0.37	1.00	0.55	-0.03	0.19	0.20	-0.02	0.06
review_count	0.39	0.55	1.00	-0.09	0.19	0.10	-0.12	-0.07
population	-0.10	-0.03	-0.09	1.00	-0.05	0.79	-0.16	-0.21
pop_density	0.34	0.19	0.19	-0.05	1.00	0.31	-0.40	-0.33
num_households	0.17	0.20	0.10	0.79	0.31	1.00	-0.15	-0.14
median_income	-0.14	-0.02	-0.12	-0.16	-0.40	-0.15	1.00	0.97
average_income	-0.06	0.06	-0.07	-0.21	-0.33	-0.14	0.97	1.00

3) Develop Pricing Recommendations: below is an interactive machine learning model used to recommend restaurant menu prices. The purpose of this interactive model is for future restaurant owners to determine the level of affordability of their menu prices based on their personal goals, specifically the type of cuisine they want to serve and which areas of San Diego they desire to do business in. Price level ranges from 1-5, which 1 being the cheapest and 5 being the most expensive. The "Rating" scale represents predictive customer satisfaction and can also help guide restaurant owners through pricing once their restaurants are already established in order to provide fair pricing based on the overall quality of their restaurants. Lastly, The "Review Count" slide is used to recommend prices based on how busy or popular a restaurant is. As discovered in earlier steps, restaurants with higher success levels tend to have a higher number of reviews. High numbers of reviews are also indicative of a restaurant's overall popularity, as well as indicates how long a restaurant has been established. As you use the interactive model, you will notice that as the review count and rating (indicating restaurant popularity and duration of establishment) increases, the price level recommendation increases, further demonstrating that higher prices do not negatively impact a restaurant's overall successs.

```
import pandas as pd
import numpy as np
import ipywidgets as widgets
from IPython.display import display
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Feature selection and data preparation for modeling
features = ['rating', 'review_count', 'cuisine_type', 'neighborhood']
target = 'price_level'

# One-hot encoding for categorical features
sd_reviews_encoded = pd.get_dummies(sd_reviews[features + [target]], columns=['cuisine_type', 'neighborhood'])

# Split data into training and testing sets
X = sd_reviews_encoded.drop(columns=[target])
y = sd_reviews_encoded[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a linear regression model (you can explore other models as well)
model = LinearRegression()
```

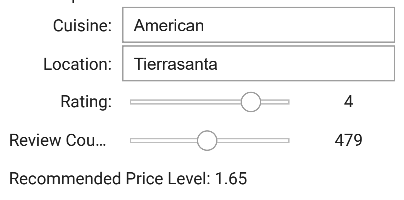
```
model.fit(X_train, y_train)

# Evaluate model performance (optional)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Create interactive widgets
cuisine_widget = widgets.Dropdown(options=sd_reviews['cuisine_type'].unique(), description='Cuisine:')
location_widget = widgets.Dropdown(options=sd_reviews['neighborhood'].unique(), description='Location:')
rating_widget = widgets.IntSlider(min=1, max=5, description='Rating:')
review_count_widget = widgets.IntSlider(min=0, max=1000, description='Review Count:')

# Create an output widget to display the prediction
output_widget = widgets.Output()

def predict_price(cuisine, location, rating, review_count):
    input_data = pd.DataFrame({
        'rating': [rating],
        'review_count': [review_count],
        'cuisine_type': [cuisine],
        'neighborhood': [location]
    })

    input_data_encoded = pd.get_dummies(input_data, columns=['cuisine_type', 'neighborhood'])

    # Align columns with training data
    missing_cols = set(X_train.columns) - set(input_data_encoded.columns)
    for col in missing_cols:
        input_data_encoded[col] = 0
    input_data_encoded = input_data_encoded[X_train.columns]

    predicted_price = model.predict(input_data_encoded)[0]

    with output_widget:
        output_widget.clear_output()  # Clear previous output
        print(f"Predicted Price Level: {predicted_price}")

# Interactive widget display
widgets.interact(predict_price,
                cuisine=cuisine_widget,
                location=location_widget,
                rating=rating_widget,
                review_count=review_count_widget)

display(output_widget)  # Display the output widget
```

```
Mean Squared Error: 0.32095525733774793
      Cuisine:  American
     Location:  Tierrasanta
       Rating:              1
  Review Cou...             0
```

```
import pandas as pd
import numpy as np
import ipywidgets as widgets
from IPython.display import display
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Feature selection and data preparation for modeling
features = ['rating', 'review_count', 'cuisine_type', 'neighborhood']  # Select relevant features
target = 'price_level'

# One-hot encoding for categorical features
sd_reviews_encoded = pd.get_dummies(sd_reviews[features + [target]], columns=['cuisine_type', 'neighborhood'])

# Split data into training and testing sets
X = sd_reviews_encoded.drop(columns=[target])
y = sd_reviews_encoded[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a linear regression model (you can explore other models as well)
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# Evaluate model performance (optional)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Create interactive widgets
cuisine_widget = widgets.Dropdown(options=sd_reviews['cuisine_type'].unique(), description='Cuisine:')
location_widget = widgets.Dropdown(options=sd_reviews['neighborhood'].unique(), description='Location:')
rating_widget = widgets.IntSlider(min=1, max=5, description='Rating:')
review_count_widget = widgets.IntSlider(min=0, max=1000, description='Review Count:')

def predict_price(cuisine, location, rating, review_count):
    # Create input data for prediction as a dictionary
    input_data = pd.DataFrame({
        'rating': [rating],
        'review_count': [review_count],
        'cuisine_type': [cuisine],
        'neighborhood': [location]
    })

    # Perform one-hot encoding and align columns
    input_data_encoded = pd.get_dummies(input_data, columns=['cuisine_type', 'neighborhood'])

    # Get missing columns from training data
    missing_cols = set(X_train.columns) - set(input_data_encoded.columns)

    # Add missing columns with 0 values
    for col in missing_cols:
        input_data_encoded[col] = 0

    # Ensure the order of columns is the same as in X_train
    input_data_encoded = input_data_encoded[X_train.columns]

    # Make prediction
    predicted_price = model.predict(input_data_encoded)[0]
    return predicted_price


# Create output widget
output_widget = widgets.Label()

# Define update function to trigger prediction
def update_output(*args):
    predicted_price = predict_price(cuisine_widget.value, location_widget.value, rating_widget.value, review_count_widget.value)
    output_widget.value = f"Recommended Price Level: {predicted_price:.2f}"  # Format to 2 decimal places

# Link widgets and update function
cuisine_widget.observe(update_output, 'value')
location_widget.observe(update_output, 'value')
rating_widget.observe(update_output, 'value')
review_count_widget.observe(update_output, 'value')

# Display widgets and output
display(cuisine_widget, location_widget, rating_widget, review_count_widget, output_widget)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` auto
  and should_run_async(code)
Mean Squared Error: 0.32095525733774793
```



```
   Cuisine: American
  Location: Tierrasanta
    Rating:           4
Review Cou...         479
Recommended Price Level: 1.65
```

Goal #3: Recommend Optimal Locations for New Restaurants

This goal focuses on identifying the most promising locations within San Diego for establishing new restaurants to increase the likelihood of success. The below code shows the distribution of sucessful restaurants in each neighborhood based on cuisine type. Because price level was previously shown not to be a distinctive factor in determining restaurants' sucess, cuisine category was the target variable for this portion. In this section, a "successful" restaurant is classified as having a success_level value of a 4 or 5. The final code also displays a Top 10 list of neighborhoods with the highest number of successful restaurants in San Diego overall.

```
import pandas as pd

# Calculate Average Success Level for each Neighborhood
neighborhood_success = sd_reviews.groupby('neighborhood')['success_level'].mean().sort_values(ascending=False)

# Get the top neighborhoods (e.g., top 10) - You need to define what 'top' means
top_neighborhoods = neighborhood_success.head(10)  # Replace 10 with the desired number of top neighborhoods

# Calculate Restaurant Density for each Neighborhood
restaurant_density = sd_reviews.groupby('neighborhood')['restaurant_name'].count() / sd_reviews.groupby('neighborhood')['land_area'].mean()

# Cuisine Type Considerations
# For each top neighborhood, explore the distribution of successful cuisine types
for neighborhood in top_neighborhoods.index:
    cuisine_distribution = sd_reviews[(sd_reviews['neighborhood'] == neighborhood) & (sd_reviews['success_level'] >= 4)]['cuisine_type'].value_co
    print(f"\nCuisine Distribution in {neighborhood} (for successful restaurants):")
    print(cuisine_distribution)
```

```
Cuisine Distribution in Normal Heights (for successful restaurants):
cuisine_type
Other             293
Italian             2
Latin American      1
Name: count, dtype: int64

Cuisine Distribution in Ocean Beach (for successful restaurants):
cuisine_type
Italian             365
American            310
Asian                 4
Other                 4
Latin American        2
Health-conscious      2
Name: count, dtype: int64

Cuisine Distribution in Pacific Beach (for successful restaurants):
cuisine_type
American            609
Other               337
Latin American      292
Cafes & Desserts      5
Name: count, dtype: int64

Cuisine Distribution in Linda Vista (for successful restaurants):
cuisine_type
Asian    1000
Other       2
Name: count, dtype: int64

Cuisine Distribution in Old Town (for successful restaurants):
cuisine_type
Italian           336
American          286
Latin American     12
Other               2
Name: count, dtype: int64

Cuisine Distribution in Downtown (for successful restaurants):
cuisine_type
Other               1553
American            1278
Italian              906
Latin American       698
Mediterranean          3
Asian                  3
Cafes & Desserts       2
Name: count, dtype: int64

Cuisine Distribution in Hillcrest (for successful restaurants):
cuisine_type
Latin American    325
American            8
Asian               2
Other               2
Italian             2
```