1. DATA OVERVIW

```
import pandas as pd
import numpy as np
import matplotlib .pyplot as plt
import seaborn as sns
df = pd.read_csv("internsip data.file.csv")
df
```

```
<ipython-input-44-3934db8852cf>:5: DtypeWarning: Columns (3,10) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("internsip data.file.csv")
```

	OrderID	Region	Country	CustID	Customer_Name	ProductSKU	Product_Category	OrderLineItem	OrderQuantity	ProductCost	ProductPrice	OrderDate
0	SO45080	Northwest	United States	14657	JOHN THOMAS	BK-R50B-58	Plants	1	1.0	413.1463	699.0982	01-01-2020
1	SO45079	Southwest	United States	29255	KYLE WASHINGTON	BK-R93R-48	Plants	1	1.0	2171.2942	3578.27	01-01-2020
2	SO45082	Australia	Australia	11455	ROSS SANZ	BK-M82B-44	Plants	1	1.0	1898.0944	3374.99	01-01-2020
3	SO45081	Canada	Canada	26782	SETH LEWIS	BK-R50B-44	Plants	1	1.0	413.1463	699.0982	01-01-2020
4	SO45083	United Kingdom	United Kingdom	14947	ALEJANDRO CHEN	BK-R93R-48	Plants	1	1.0	2171.2942	3578.27	02-01-2020
...	...	...	...	...	...	...	...	...	...	...	...	...
55905	SO74143	United Kingdom	United Kingdom	28517	TROY GONZALEZ	WB-H098	Plant Care & Seeds	3	2.0	1.8663	4.99	30-06-2022
55906	SO74143	United Kingdom	United Kingdom	28517	TROY GONZALEZ	BC-R205	Plant Care & Seeds	2	1.0	3.3623	8.99	30-06-2022
55907	SO74143	United Kingdom	United Kingdom	28517	TROY GONZALEZ	BK-R19B-52	Plants	1	1.0	343.6496	539.99	30-06-2022
55908	SO74124	France	France	21676	VALERIE GUO	PK-7098	Plant Care & Seeds	2	2.0	0.8565	2.29	30-06-2022
55909	SO74124	France	France	21676	VALERIE GUO	TI-R092	Plant Care & Seeds	1	2.0	8.0373	21.49	30-06-2022

55910 rows × 18 columns

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

Double-click (or enter) to edit

```
# last 5 rows
df.head(5)
```

	OrderID	Region	Country	CustID	Customer_Name	ProductSKU	Product_Category	OrderLineItem	OrderQuantity	ProductCost	ProductPrice	OrderDate	Acqu
0	SO45080	Northwest	United States	14657	JOHN THOMAS	BK-R50B-58	Plants	1	1.0	413.1463	699.0982	01-01-2020	
1	SO45079	Southwest	United States	29255	KYLE WASHINGTON	BK-R93R-48	Plants	1	1.0	2171.2942	3578.27	01-01-2020	
2	SO45082	Australia	Australia	11455	ROSS SANZ	BK-M82B-44	Plants	1	1.0	1898.0944	3374.99	01-01-2020	
3	SO45081	Canada	Canada	26782	SETH LEWIS	BK-R50B-44	Plants	1	1.0	413.1463	699.0982	01-01-2020	
4	SO45083	United Kingdom	United Kingdom	14947	ALEJANDRO CHEN	BK-R93R-48	Plants	1	1.0	2171.2942	3578.27	02-01-2020	

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

```
# CHECK DATRTYPE OF EACH COLUMN
df.dtypes
```

	0
OrderID	object
Region	object
Country	object
CustID	object
Customer_Name	object
ProductSKU	object
Product_Category	object
OrderLineItem	int64
OrderQuantity	float64
ProductCost	float64
ProductPrice	object
OrderDate	object
AcquisitionSource	object
TransactionID	object
Fraud	object
PaymentMethod	object
CardType	object
Gender	object

dtype: object

```
# TOTALS  ROWS AND COLUMNS IN DATASET
df.columns
```

```
Index(['OrderID', 'Region', 'Country', 'CustID', 'Customer_Name', 'ProductSKU',
       'Product_Category', 'OrderLineItem', 'OrderQuantity', 'ProductCost',
       'ProductPrice', 'OrderDate', 'AcquisitionSource', 'TransactionID',
       'Fraud', 'PaymentMethod', 'CardType', 'Gender'],
      dtype='object')
```

```
len(df.columns)
```

```
18
```

```
len(df)
```

```
55910
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55910 entries, 0 to 55909
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   OrderID            55910 non-null  object 
 1   Region             55910 non-null  object 
 2   Country            55910 non-null  object 
 3   CustID             55910 non-null  object 
 4   Customer_Name      55907 non-null  object 
 5   ProductSKU         55910 non-null  object 
 6   Product_Category   55910 non-null  object 
 7   OrderLineItem      55910 non-null  int64  
 8   OrderQuantity      55910 non-null  float64
 9   ProductCost        55910 non-null  float64
 10  ProductPrice       55910 non-null  object 
 11  OrderDate          55910 non-null  object 
 12  AcquisitionSource  55909 non-null  object 
 13  TransactionID      55910 non-null  object 
 14  Fraud              55207 non-null  object 
 15  PaymentMethod      55910 non-null  object 
 16  CardType           50687 non-null  object 
 17  Gender             49582 non-null  object 
```

```
dtypes: float64(2), int64(1), object(15)
memory usage: 7.7+ MB
```

```
# CHECK NULL VALUE IN THE DATASET
df.isnull().sum()
```

```
df[df['CardType'].isna()==1]
```

	OrderID	Region	Country	CustID	Customer_Name	ProductSKU	Product_Category	OrderLineItem	OrderQuantity	ProductCost	ProductPrice	OrderDate
9	SO45090	Southwest	United States	29170	ALEXANDRA ADAMS	BK-R93R-62	Plants	1	1.0	2171.2942	3578.27	03-01-2020
11	SO45092	Australia	Australia	18899	GLENN LIU	BK-R93R-52	Plants	1	1.0	2171.2942	3578.27	03-01-2020
14	SO45087	United Kingdom	United Kingdom	11388	JOSEPH MARTIN	BK-M82B-44	Plants	1	1.0	1898.0944	3374.99	03-01-2020
29	SO45107	Southwest	United States	29275	TAYLOR THOMAS	BK-R93R-48	Plants	1	1.0	2171.2942	3578.27	06-01-2020
35	SO45117	Northwest	United States	14727	ANNA BRYANT	BK-R50B-52	Plants	1	1.0	413.1463	699.0982	08-01-2020
...	...	...	...	...	...	...	...	...	...	...	...	...
55867	SO74111	Southwest	United States	26009	LUIS FOSTER	PK-7098	Plant Care & Seeds	3	3.0	0.8565	2.29	30-06-2022
55871	SO74146	Australia	Australia	26916	MACKENZIE KING	BK-R19B-58	Plants	1	1.0	343.6496	539.99	30-06-2022
55882	SO74132	Canada	Canada	11512	NATALIE CAMPBELL	BK-M68S-46	Plants	1	1.0	1117.8559	2071.4196	30-06-2022
55896	SO74142	Northwest	United States	22253	STEPHANIE REED	SJ-0194-L	Pots	2	1.0	41.5723	53.99	30-06-2022
55909	SO74124	France	France	21676	VALERIE GUO	TI-R092	Plant Care & Seeds	1	2.0	8.0373	21.49	30-06-2022

5223 rows × 18 columns

DATA CLEANING

```
# FILL NULL VALUES WITH 0
df["CardType"] = df["CardType"].fillna(0)
```

```
df['Gender'] = df['Gender'].fillna(0)
```

```
df[df['Customer_Name'].isna()==1]
```

	OrderID	Region	Country	CustID	Customer_Name	ProductSKU	Product_Category	OrderLineItem	OrderQuantity	ProductCost	ProductPrice	OrderDate	Ac
8573	SO53176	Australia	Australia	11096	NaN	BK-M68B-42	Plants	1	1.0	1105.8100	2049.0982	26-08-2021	
30786	SO63115	Australia	Australia	11096	NaN	CA-1098	Pots	1	1.0	5.7052	8.6442	31-01-2022	
30787	SO63115	Australia	Australia	11096	NaN	BK-T79Y-60	Plants	2	1.0	1481.9379	2384.07	31-01-2022	

```
df['Customer_Name'] = df['Customer_Name'].fillna(0)
```

```
df['Fraud'] = df['Fraud'].fillna(0)
```

```
df['AcquisitionSource'] = df['AcquisitionSource'].fillna(0)
```

```
# DROP DUPLICATES
df.duplicated()
```

	0
0	False
1	False
2	False
3	False
4	False
...	...
55905	False
55906	False
55907	False
55908	False
55909	False

55910 rows × 1 columns

dtype: bool

```
# DROP DUPLICATES METHOD
df = df.drop_duplicates()
```

```
df.columns
```

```
Index(['OrderID', 'Region', 'Country', 'CustID', 'Customer_Name', 'ProductSKU',
       'Product_Category', 'OrderLineItem', 'OrderQuantity', 'ProductCost',
       'ProductPrice', 'OrderDate', 'AcquisitionSource', 'TransactionID',
       'Fraud', 'PaymentMethod', 'CardType', 'Gender'],
      dtype='object')
```

```
df.isnull().sum()
```

	0
OrderID	0
Region	0
Country	0
CustID	0
Customer_Name	0
ProductSKU	0
Product_Category	0
OrderLineItem	0
OrderQuantity	0
ProductCost	0
ProductPrice	0
OrderDate	0
AcquisitionSource	0
TransactionID	0
Fraud	0
PaymentMethod	0
CardType	5223
Gender	0

dtype: int64

```
import pandas as pd
import numpy as np
import matplotlib .pyplot as plt
import seaborn as sns
df = pd.read_csv("internsip data.file.csv")
df
```

```
<ipython-input-60-3934db8852cf>:5: DtypeWarning: Columns (3,10) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("internsip data.file.csv")
```

	OrderID	Region	Country	CustID	Customer_Name	ProductSKU	Product_Category	OrderLineItem	OrderQuantity	ProductCost	ProductPrice	OrderDate
0	SO45080	Northwest	United States	14657	JOHN THOMAS	BK-R50B-58	Plants	1	1.0	413.1463	699.0982	01-01-2020
1	SO45079	Southwest	United States	29255	KYLE WASHINGTON	BK-R93R-48	Plants	1	1.0	2171.2942	3578.27	01-01-2020
2	SO45082	Australia	Australia	11455	ROSS SANZ	BK-M82B-44	Plants	1	1.0	1898.0944	3374.99	01-01-2020
3	SO45081	Canada	Canada	26782	SETH LEWIS	BK-R50B-44	Plants	1	1.0	413.1463	699.0982	01-01-2020
4	SO45083	United Kingdom	United Kingdom	14947	ALEJANDRO CHEN	BK-R93R-48	Plants	1	1.0	2171.2942	3578.27	02-01-2020
...	...	...	...	...	...	...	...	...	...	...	...	...
55905	SO74143	United Kingdom	United Kingdom	28517	TROY GONZALEZ	WB-H098	Plant Care & Seeds	3	2.0	1.8663	4.99	30-06-2022
55906	SO74143	United Kingdom	United Kingdom	28517	TROY GONZALEZ	BC-R205	Plant Care & Seeds	2	1.0	3.3623	8.99	30-06-2022
55907	SO74143	United Kingdom	United Kingdom	28517	TROY GONZALEZ	BK-R19B-52	Plants	1	1.0	343.6496	539.99	30-06-2022
55908	SO74124	France	France	21676	VALERIE GUO	PK-7098	Plant Care & Seeds	2	2.0	0.8565	2.29	30-06-2022
55909	SO74124	France	France	21676	VALERIE GUO	TI-R092	Plant Care & Seeds	1	2.0	8.0373	21.49	30-06-2022

55910 rows × 18 columns

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

```
df.columns
```

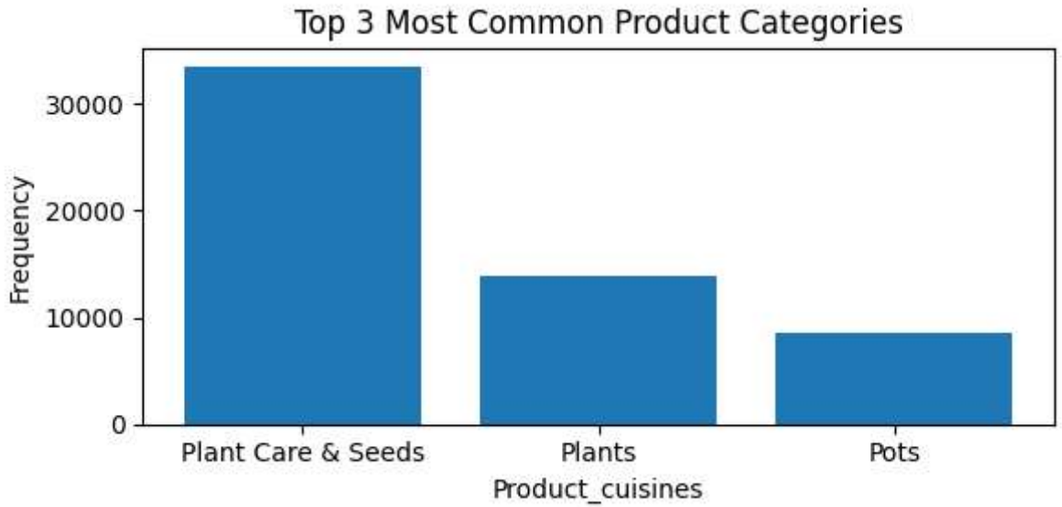
```
Index(['OrderID', 'Region', 'Country', 'CustID', 'Customer_Name', 'ProductSKU',
       'Product_Category', 'OrderLineItem', 'OrderQuantity', 'ProductCost',
       'ProductPrice', 'OrderDate', 'AcquisitionSource', 'TransactionID',
       'Fraud', 'PaymentMethod', 'CardType', 'Gender'],
      dtype='object')
```

- Task: Top Cuisines

TASK- 1. *Determine the top three most **common*** cuisines in the dataset.

```
#Determine the top three most common cuisines in the dataset.
top_cuisines = df['Product_Category'].value_counts().head(3)
plt.figure(figsize=(6,3))
plt.figure(figsize=(6,3))

plt.bar(top_cuisines.index, top_cuisines.values)

plt.xlabel('Product_cuisines')

plt.ylabel('Frequency')

plt.title('Top 3 Most Common Product Categories')

plt.xticks(rotation=0)

plt.tight_layout()

plt.show()
```

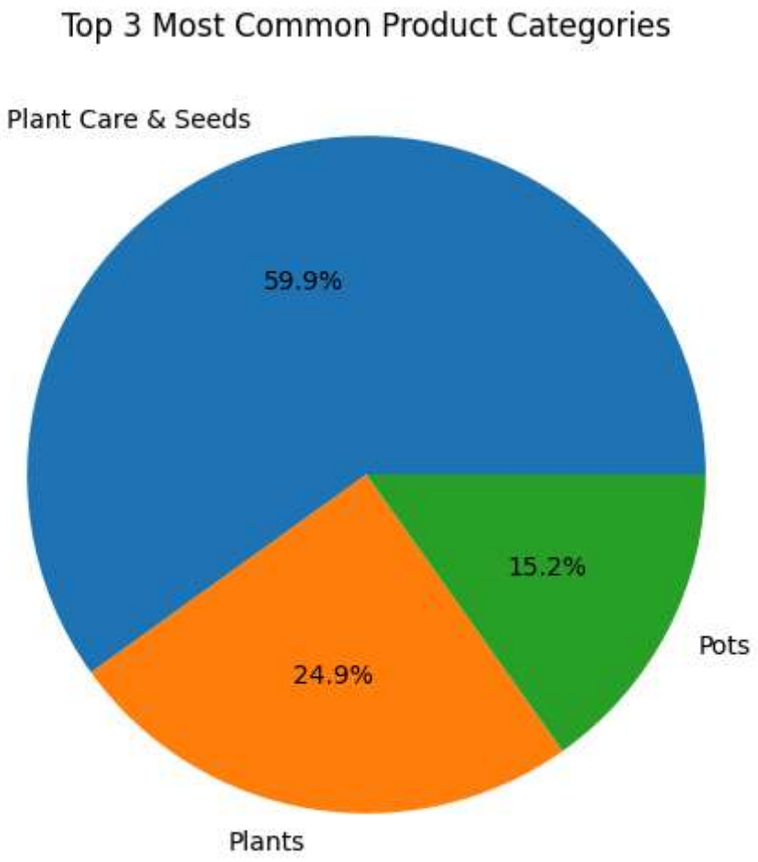
```
<Figure size 600x300 with 0 Axes>
```



Calculate the percentage of restaurants that serve each of the top cuisines

```
#Calculate the percentage of restaurants that serve each of the top cuisines
percentage = (top_cuisines/ len(df)) * 100

# print the result

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

plt.pie(percentage, labels = top_cuisines.index, autopct='%1.1f%%')

plt.title('Top 3 Most Common Product Categories')

plt.show()
```

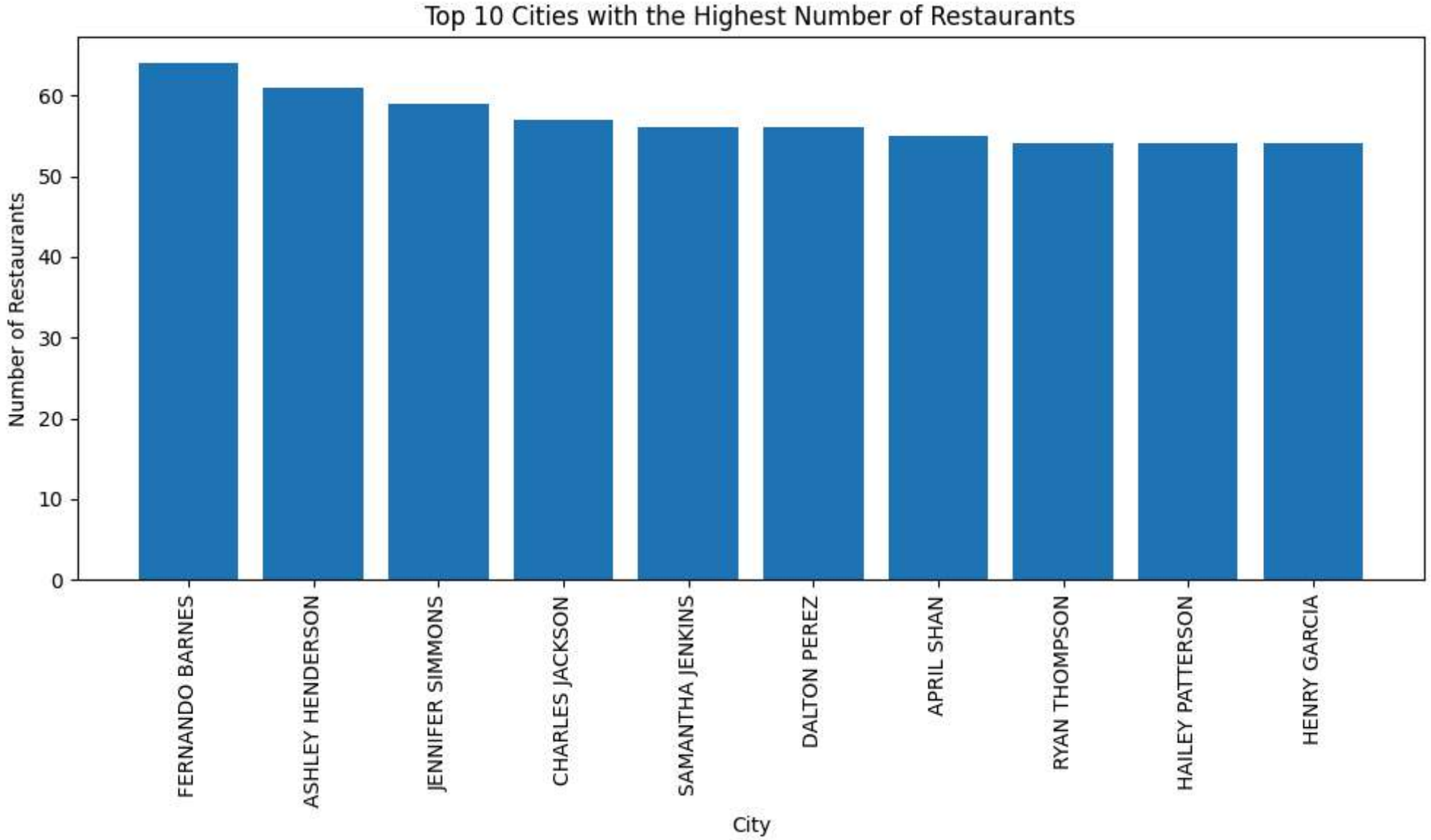
```
Product_Category
Plant Care & Seeds    59.926668
Plants                24.884636
Pots                  15.186908
Name: count, dtype: float64
```



Task: City Analysis

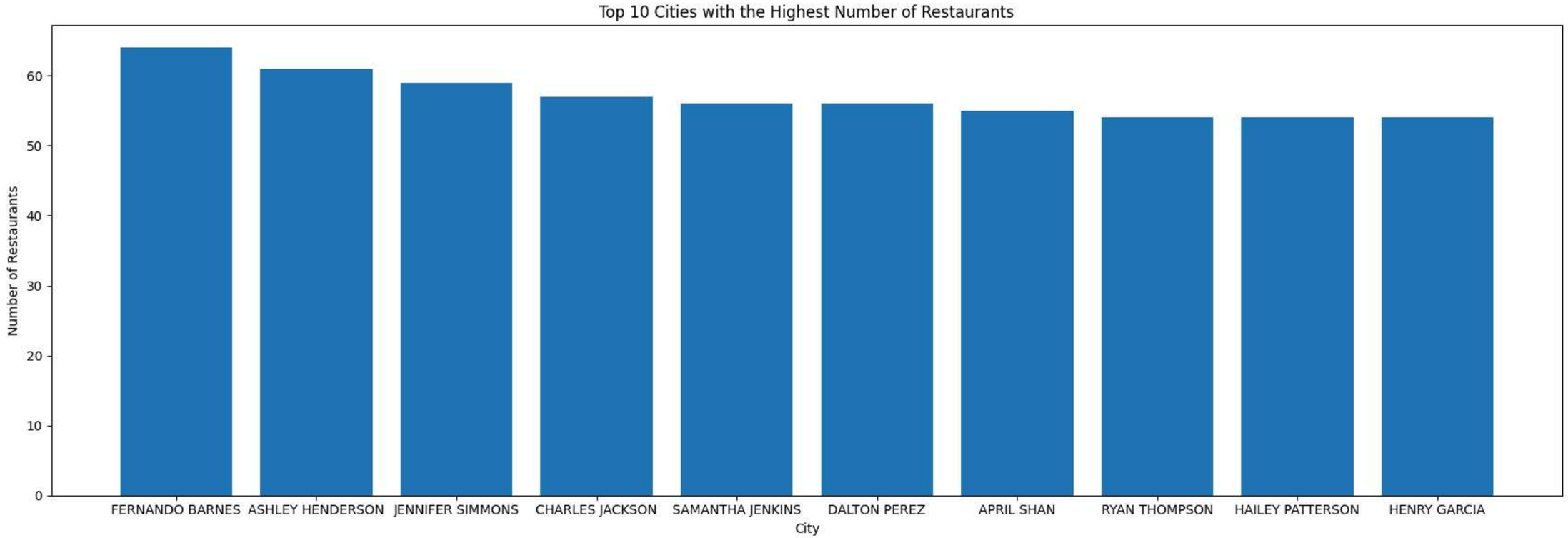
Identify the city with the highest number of restaurants in the dataset

```
city_restaurants =df['Customer_Name'].value_counts().head(10)

# create bar chart

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

plt.bar(city_restaurants.index,city_restaurants.values)

plt.xlabel('City')

plt.ylabel('Number of Restaurants')

plt.title('Top 10 Cities with the Highest Number of Restaurants')

plt.xticks(rotation=90)

plt.tight_layout()

plt.show()
```



Identify the city with the highest number of restaurants in the dataset.

```
# Calculate the number of restaurants according to the city

city_restaurants = df['Customer_Name'].value_counts().head(10)

# create bar chart

plt.figure(figsize=(17,6))

plt.bar(city_restaurants.index,city_restaurants.values)

plt.xlabel('City')

plt.ylabel('Number of Restaurants')

plt.title('Top 10 Cities with the Highest Number of Restaurants')

plt.xticks(rotation=0) # make the labels straight

plt.tight_layout()

plt.show()

# Number of restaurants by city
print(city_restaurants)
```




```
Customer_Name
FERNANDO BARNES     64
ASHLEY HENDERSON    61
JENNIFER SIMMONS    59
CHARLES JACKSON     57
SAMANTHA JENKINS    56
DALTON PEREZ        56
APRIL SHAN          55
RYAN THOMPSON       54
HAILEY PATTERSON    54
HENRY GARCIA        54
Name: count, dtype: int64
```

Double-click (or enter) to edit

Calculate the average rating for restaurants in each city.

```
#Convert ProductPrice aur ProductCost columns ko float type mein
df['ProductPrice'] = pd.to_numeric(df['ProductPrice'], errors='coerce')
df['ProductCost'] = pd.to_numeric(df['ProductCost'], errors='coerce')

#Calculate rating
df['Rating'] = (df['ProductPrice'] - df['ProductCost']) / df['ProductCost']

#Group data by 'Region' and calculate average rating
average_rating = df.groupby('Region')['Rating'].mean().reset_index()

#Rename columns for clarity
average_rating.columns = ['City', 'Average_Rating']

#Print the result
print(average_rating)
```

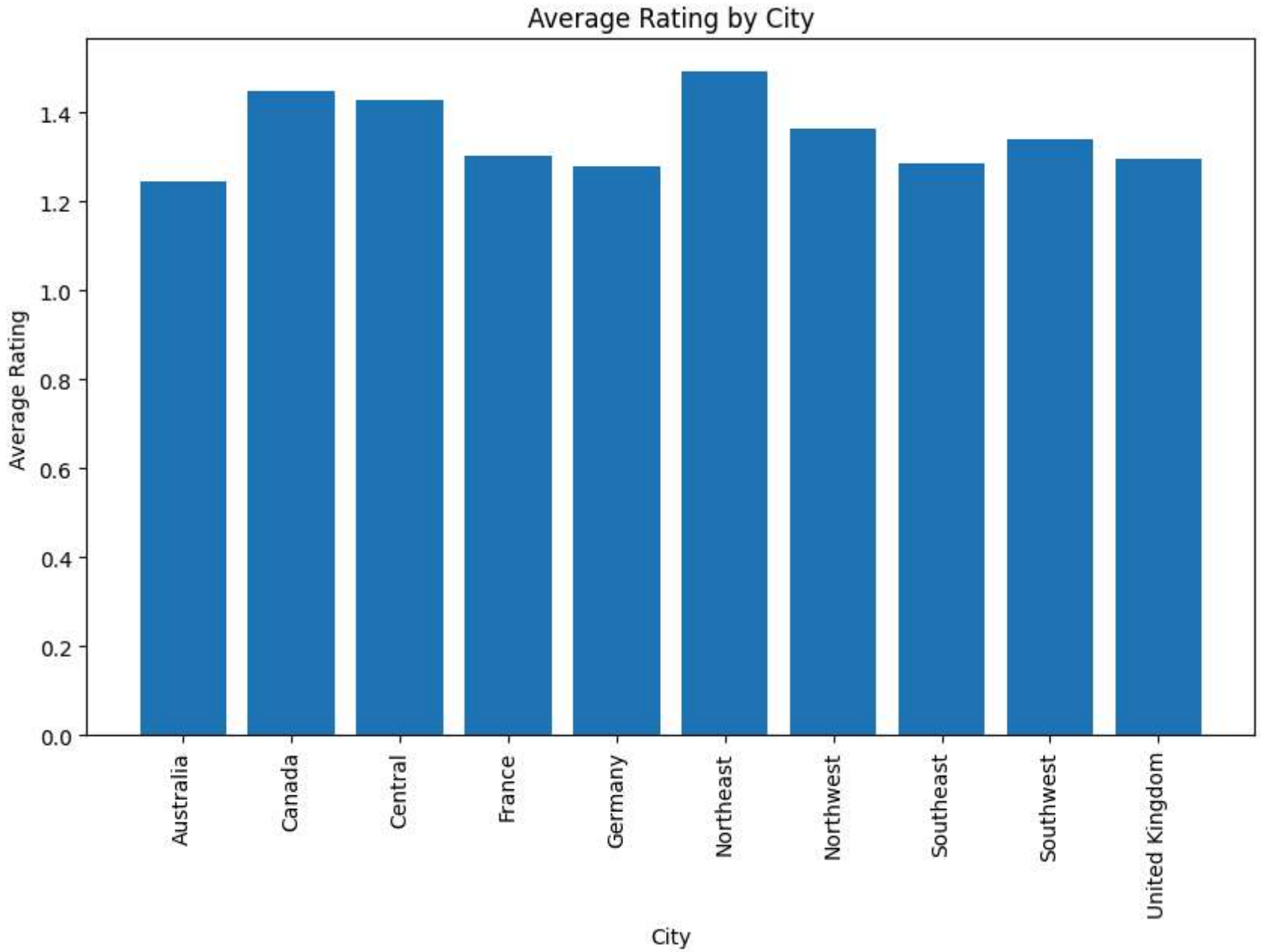
```
             City  Average_Rating
0       Australia        1.063191
1          Canada        1.346287
2         Central        0.853029
3          France        1.140909
4         Germany        1.128978
5       Northeast        1.400147
6       Northwest        1.223726
7       Southeast        0.745765
8       Southwest        1.191904
9  United Kingdom        1.152137
```

Start coding or generate with AI.

Determine the city with the highest average rating.

```
#Convert ProductPrice aur ProductCost columns ko float type mein
df['ProductPrice'] = pd.to_numeric(df['ProductPrice'], errors='coerce')
df['ProductCost'] = pd.to_numeric(df['ProductCost'], errors='coerce')

#Calculate rating
df['Rating'] = (df['ProductPrice'] - df['ProductCost']) / df['ProductCost']

#Group data by 'Region' and calculate average rating
average_rating = df.groupby('Region')['Rating'].mean().reset_index()

#Rename columns for clarity
average_rating.columns = ['City', 'Average_Rating']

#Determine city with highest average rating
highest_rating_city = average_rating.loc[average_rating['Average_Rating'].idxmax()]

#Print result
print("City with highest average rating:")
print(highest_rating_city)

#Plot bar graph
plt.figure(figsize=(10,6))
plt.bar(average_rating['City'], average_rating['Average_Rating'])
plt.xlabel('City')
plt.ylabel('Average Rating')
plt.title('Average Rating by City')
plt.xticks(rotation=90)
plt.show()
```

```
City with highest average rating:
City              Northeast
Average_Rating     1.490784
Name: 5, dtype: object
```



Task: Price Range Distribution

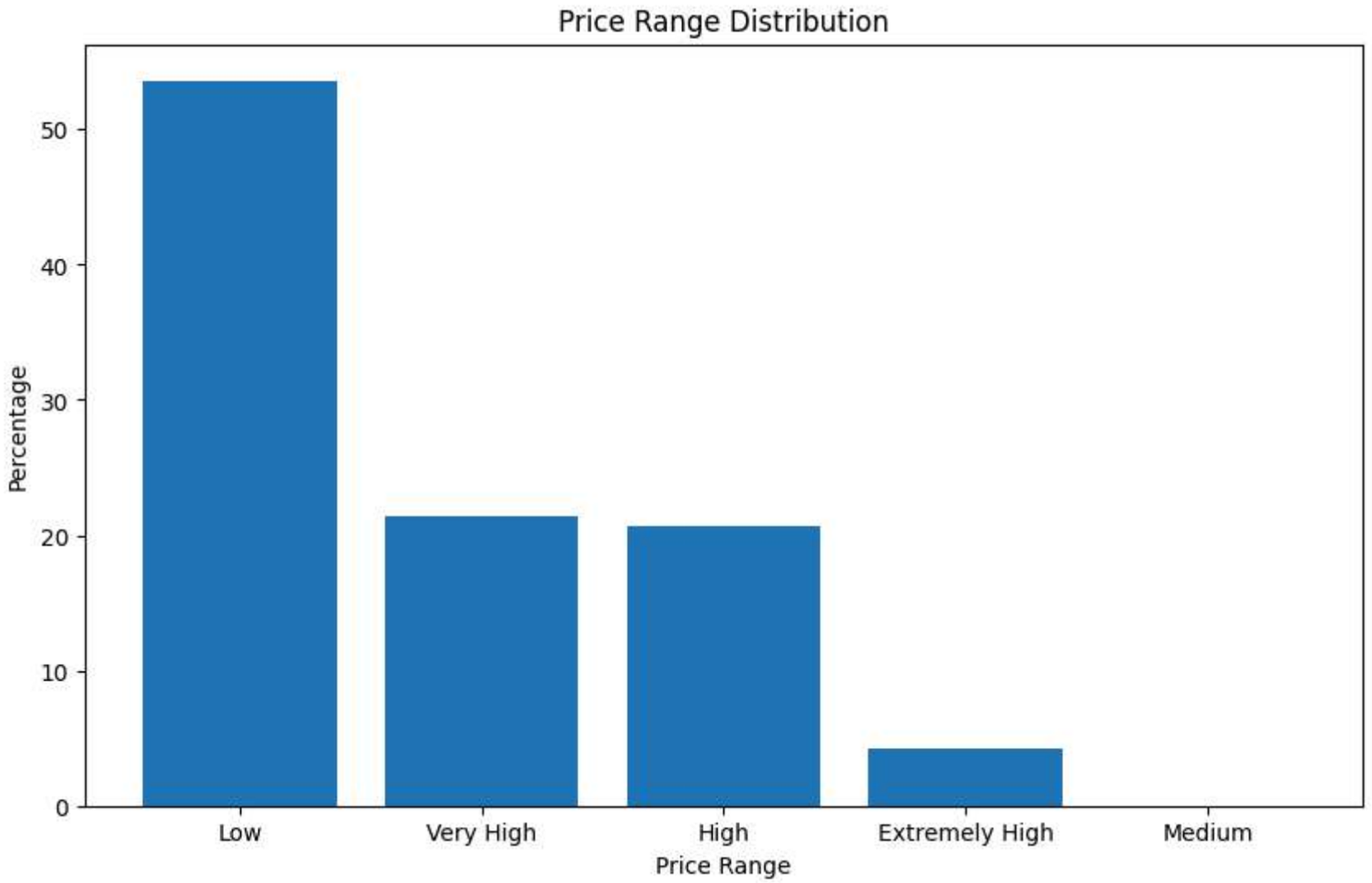
```
df.columns
```

```
Index(['OrderID', 'Region', 'Country', 'CustID', 'Customer_Name', 'ProductSKU',
       'Product_Category', 'OrderLineItem', 'OrderQuantity', 'ProductCost',
       'ProductPrice', 'OrderDate', 'AcquisitionSource', 'TransactionID',
       'Fraud', 'PaymentMethod', 'CardType', 'Gender', 'Rating'],
      dtype='object')
```

Create a histogram or bar chart to visualize the distribution of price ranges among the restaurants.

```
#Create price range categories
bins = [0, 10, 20, 30, 40, 50]
labels = ['Low', 'Medium', 'High', 'Very High', 'Extremely High']

#Assign price range categories to each restaurant
df['PriceRange'] = pd.cut(df['ProductPrice'], bins=bins, labels=labels)

#Convert PriceRange column to pandas Series
price_range_series = pd.Series(df['PriceRange'])

#Calculate percentage of restaurants in each price range category
percentage = price_range_series.value_counts(normalize=True) * 100

#Create a bar chart
plt.figure(figsize=(10,6))
plt.bar(percentage.index, percentage.values)
plt.xlabel('Price Range')
```

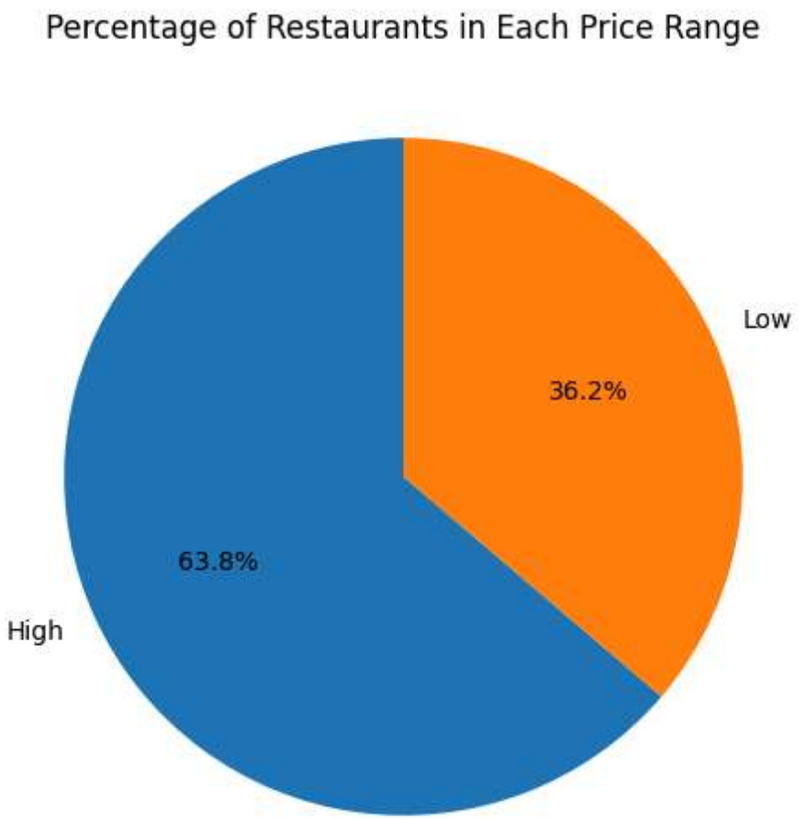
```
plt.ylabel('Percentage')
plt.title('Price Range Distribution')
plt.show()
```



Calculate the percentage of restaurants in each price range category.

```
# Define a function to categorize based on price
def categorize_price(price):
    if price < 10:
        return 'Low'
    elif 10 <= price < 20:
        return 'Medium'
    else:
        return 'High'

# Assuming 'ProductPrice' is the column containing the restaurant's price
df['PriceRange'] = df['ProductPrice'].apply(categorize_price)

# Now calculate the percentage of restaurants in each price range
price_range_counts = df['PriceRange'].value_counts()
price_range_percentage = (price_range_counts / len(df)) * 100

# Print the result
print("Percentage of Restaurants in Each Price Range:")
for price_range, percentage in price_range_percentage.items():
    print(f"{price_range}: {percentage:.2f}%")

# Create a pie chart
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.pie(price_range_counts, labels=price_range_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Percentage of Restaurants in Each Price Range')
plt.show()
```

```
Percentage of Restaurants in Each Price Range:
High: 63.75%
Low: 36.25%
```



Task: Online Delivery

```
df.columns
```

```
Index(['OrderID', 'Region', 'Country', 'CustID', 'Customer_Name', 'ProductSKU',
       'Product_Category', 'OrderLineItem', 'OrderQuantity', 'ProductCost',
```

```
       'ProductPrice', 'OrderDate', 'AcquisitionSource', 'TransactionID',
       'Fraud', 'PaymentMethod', 'CardType', 'Gender', 'Rating', 'PriceRange'],
      dtype='object')
```

Determine the percentage of restaurants that offer online delivery.

```
# Create a new column 'OnlineDelivery' based on the 'AcquisitionSource' column
df['OnlineDelivery'] = df['AcquisitionSource'].apply(lambda x: 1 if x == 'Online' else 0)

# Calculate the percentage of restaurants that offer online delivery
online_delivery_percentage = df['OnlineDelivery'].mean() * 100

# Print the result
print(f"The percentage of restaurants that offer online delivery is: {online_delivery_percentage:.2f}%")

# Optional: Create a pie chart to visualize the percentage of restaurants offering online delivery
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
plt.pie([df['OnlineDelivery'].sum(), len(df) - df['OnlineDelivery'].sum()],
        labels=['Online Delivery', 'No Online Delivery'], autopct='%1.1f%%')
plt.title('Percentage of Restaurants that Offer Online Delivery')
plt.show()
```

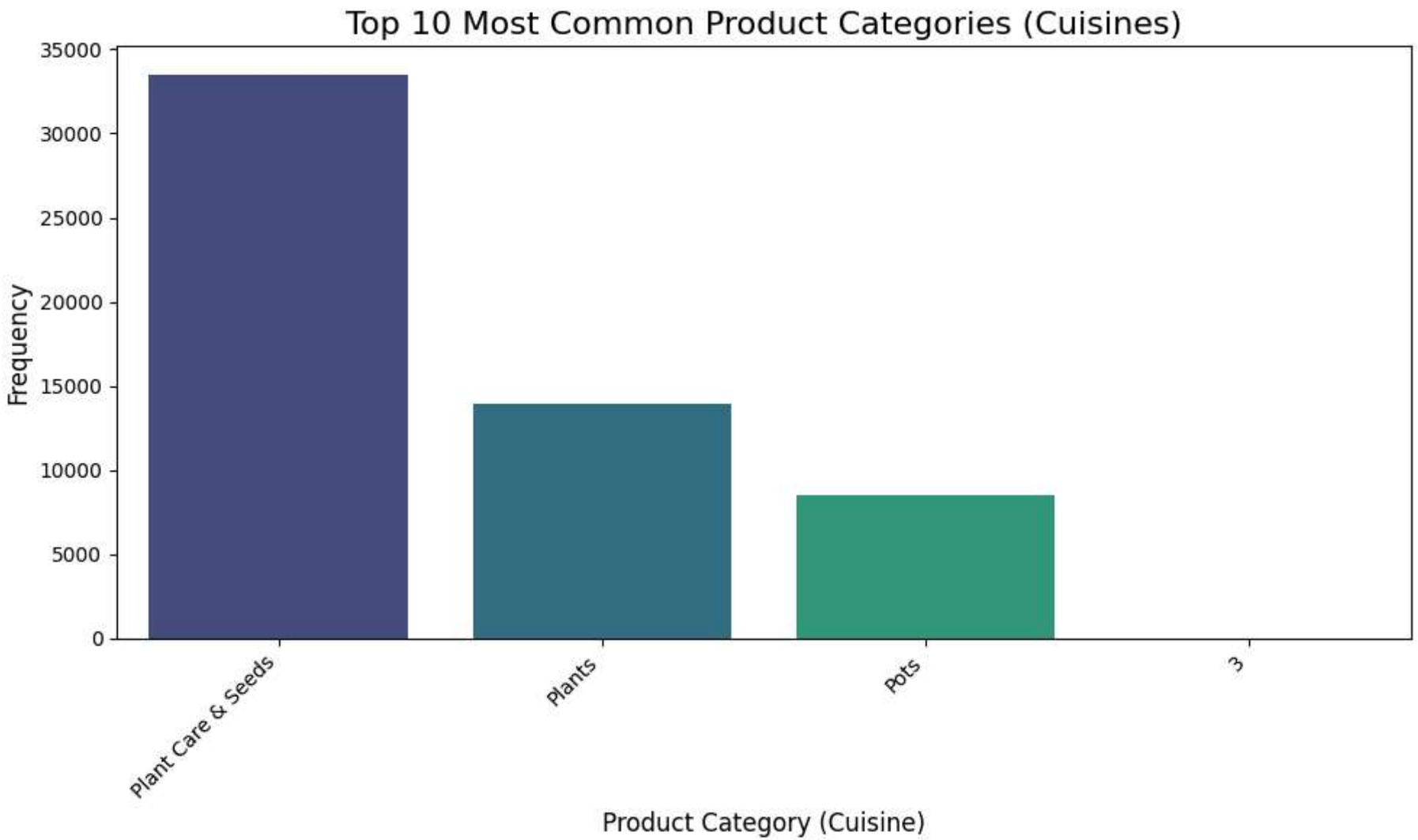
```
The percentage of restaurants that offer online delivery is: 0.00%
```

Identify the most common combinations of cuisines in the dataset

```
##Identify the most common combinations of cuisines in the dataset
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv("internsip data.file.csv")
df

# Load your dataset (example: df = pd.read_csv('your_data.csv'))
# For demonstration, let's assume df is already defined.

# Step 1: Identify the most common combinations of cuisines in ProductCategory
# First, let's check how many unique product categories exist.
product_category_counts = df['Product_Category'].value_counts()

# Step 2: Get the top 10 most common product categories (assuming these represent cuisines)
top_product_categories = product_category_counts.head(10)

# Step 3: Create a bar chart to visualize the most common product categories
plt.figure(figsize=(10, 6))
sns.barplot(x=top_product_categories.index, y=top_product_categories.values, palette='viridis')

plt.title("Top 10 Most Common Product Categories (Cuisines)", fontsize=16)
plt.xlabel("Product Category (Cuisine)", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Show the plot
plt.show()
```

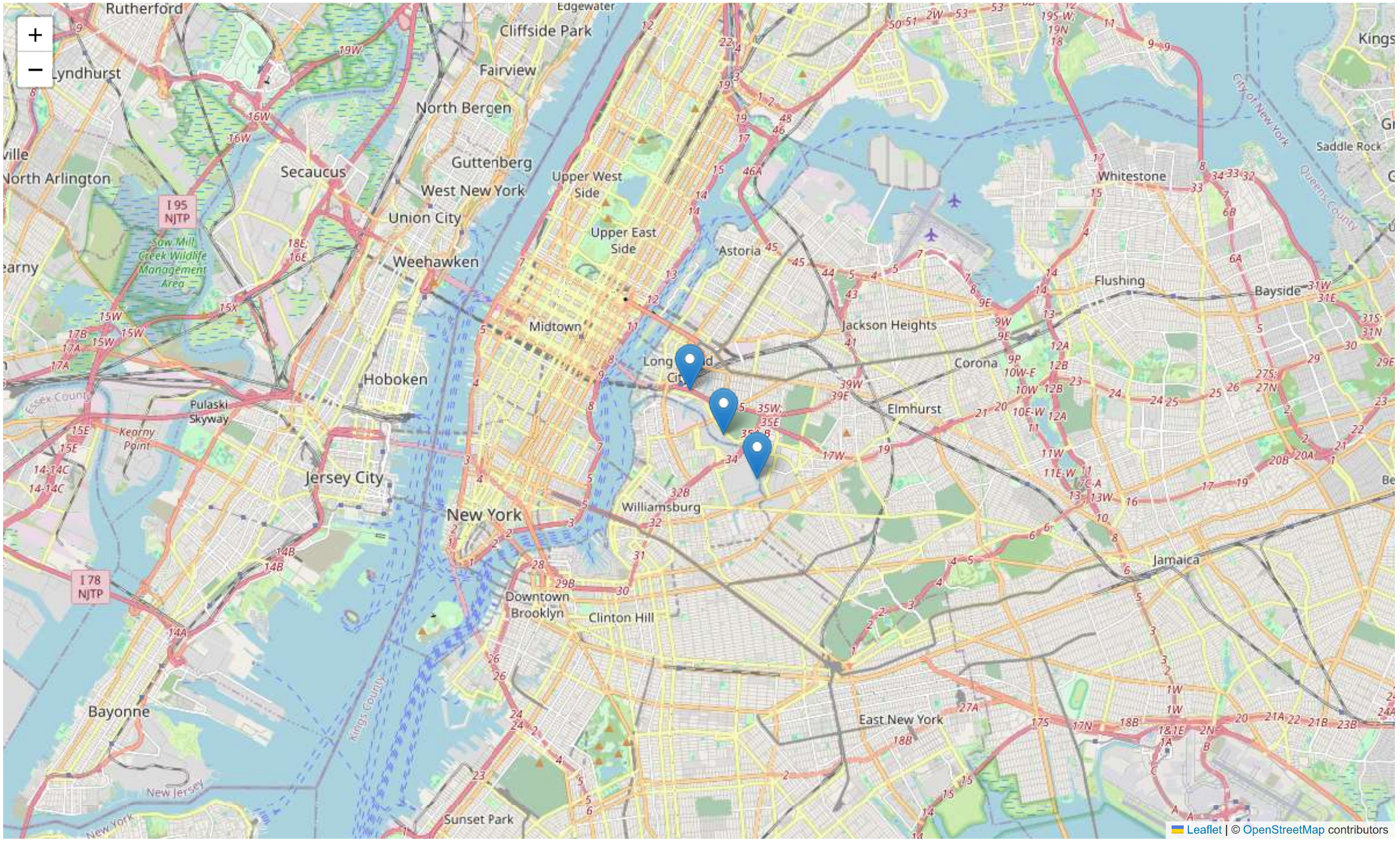
```
<ipython-input-74-b87c7f1fa8ad>:5: DtypeWarning: Columns (3,10) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("internsip data.file.csv")
<ipython-input-74-b87c7f1fa8ad>:20: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same

  sns.barplot(x=top_product_categories.index, y=top_product_categories.values, palette='viridis')
```



Plot the locations of restaurants on a map using longitude and latitude coordinates.

Double-click (or enter) to edit

```
import pandas as pd
import folium

# Sample DataFrame with restaurant locations (longitude and latitude)
# Assuming the DataFrame looks something like this:

data = {
    'Restaurant_Name': ['Restaurant A', 'Restaurant B', 'Restaurant C'],
    'Longitude': [-73.935242, -73.925242, -73.945242],
    'Latitude': [40.730610, 40.720610, 40.740610]
}

df = pd.DataFrame(data)

# Step 1: Create a base map centered around a specific latitude and longitude (e.g., New York City)
map_center = [40.730610, -73.935242]  # Latitude and Longitude of New York City
mymap = folium.Map(location=map_center, zoom_start=12)

# Step 2: Add markers for each restaurant
for _, row in df.iterrows():
    folium.Marker(location=[row['Latitude'], row['Longitude']], popup=row['Restaurant_Name']).add_to(mymap)

# Step 3: Save the map to an HTML file
mymap.save('restaurants_map.html')

# Step 4: Display the map (in a Jupyter notebook environment, you can display it inline)
mymap
```


Rutherford
Edgewater
Cliffside Park
Fairview
North Bergen
Guttenberg
Secaucus
West New York
Union City
Weehawken
Upper West Side
Upper East Side
Midtown
Astoria
Whitestone
Flushing
Bayside
Jackson Heights
Corona
Elmhurst
Hoboken
Jersey City
New York
Williamsburg
Downtown Brooklyn
Clinton Hill
Jamaica
East New York
Bayonne
Sunset Park
Pulaski Skyway
Kearny Point
Saw Mill Creek Wildlife Management Area
I 95 NJTP
I 78 NJTP
Leaflet | © OpenStreetMap contributors

Double-click (or enter) to edit

```
df.columns
```

```
Start coding or generate with AI.
```

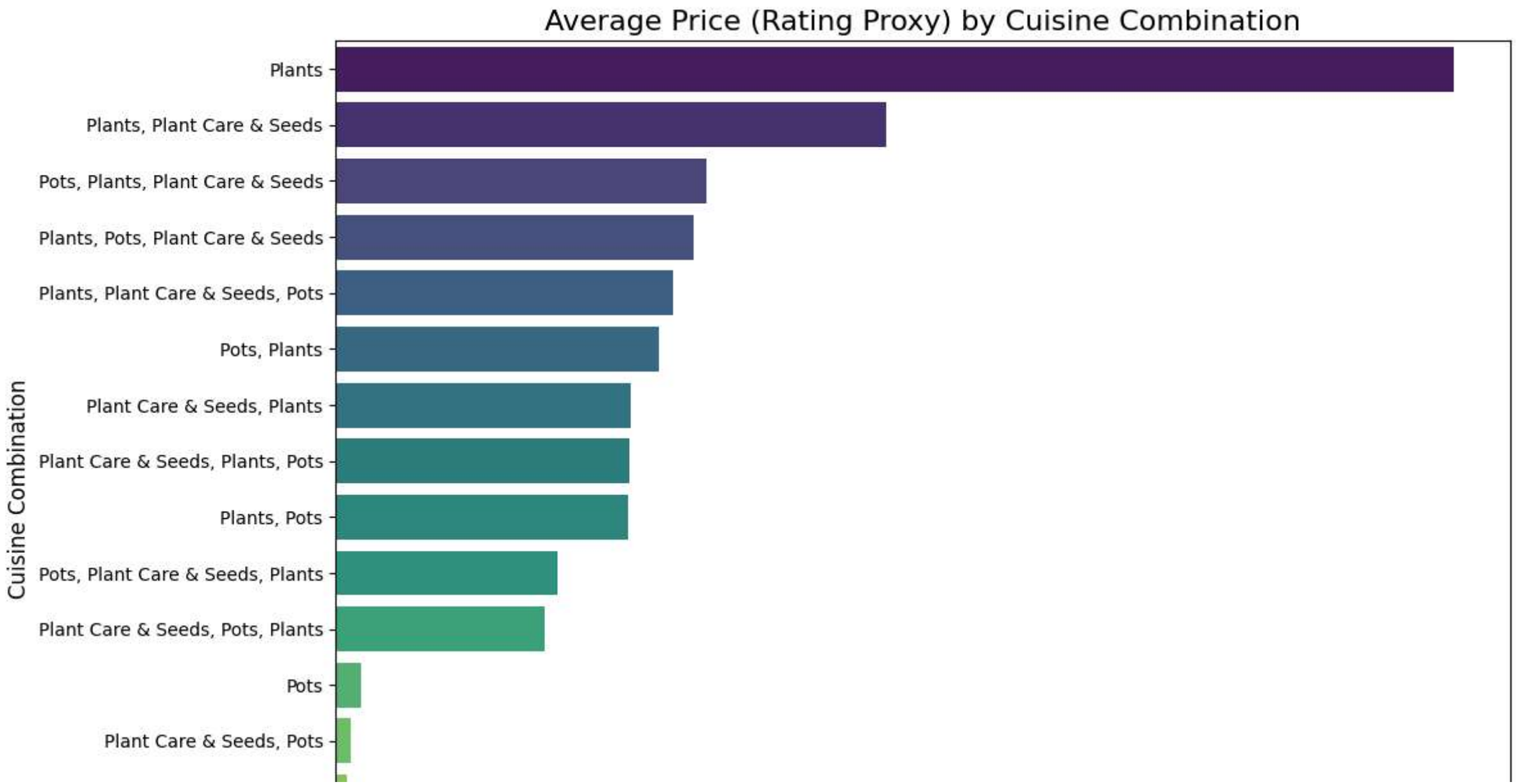
Plot the locations of restaurants on a map using longitude and latitude coordinates.

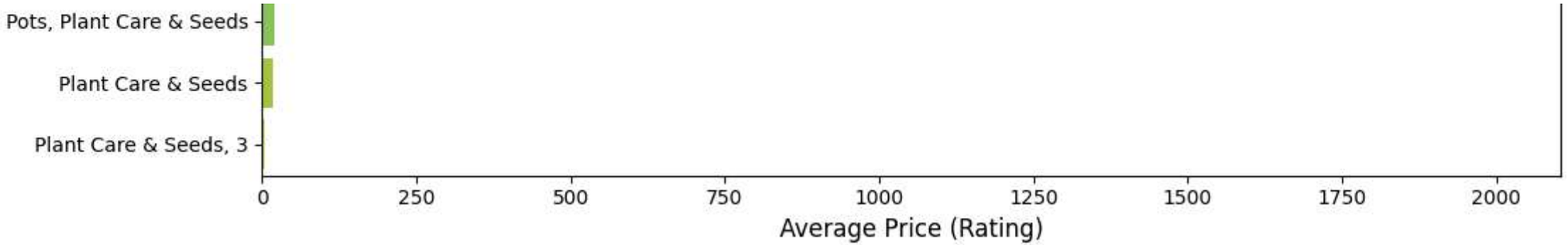
```
# Convert 'ProductPrice' to numeric, forcing errors to NaN (useful in case there are invalid values like text)
df['ProductPrice'] = pd.to_numeric(df['ProductPrice'], errors='coerce')
```

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

# Load the dataset
df = pd.read_csv("internsip data.file.csv")
df

# Step 1: Convert 'ProductPrice' to numeric (in case it's not already)
df['ProductPrice'] = pd.to_numeric(df['ProductPrice'], errors='coerce')

# Step 2: Create a sample rating or price column. Here, we use 'ProductPrice' as a proxy for ratings.
# Combine cuisines per order: Assuming 'Product_Category' represents cuisines.
order_cuisines = df.groupby('OrderID')['Product_Category'].apply(lambda x: ', '.join(x.unique())).reset_index()

# Step 3: Merge back the 'ProductPrice' to the cuisines
# This assumes 'ProductPrice' is in the original DataFrame, and we are summarizing it by 'OrderID'.
```

```
order_cuisines = order_cuisines.merge(df[['OrderID', 'ProductPrice']].drop_duplicates(), on='OrderID', how='left')

# Step 4: Find the average price for each cuisine combination
cuisine_ratings = order_cuisines.groupby('Product_Category').agg({'ProductPrice': 'mean'}).reset_index()

# Step 5: Sort and visualize the results
# Sort by mean price
cuisine_ratings_sorted = cuisine_ratings.sort_values(by='ProductPrice', ascending=False)

# Plotting the results
plt.figure(figsize=(12, 8))
sns.barplot(x='ProductPrice', y='Product_Category', data=cuisine_ratings_sorted, palette='viridis')

plt.title("Average Price (Rating Proxy) by Cuisine Combination", fontsize=16)
plt.xlabel("Average Price (Rating)", fontsize=12)
plt.ylabel("Cuisine Combination", fontsize=12)
plt.tight_layout()

# Show the plot
plt.show()
```

```
<ipython-input-35-9d8f3e36c62c>:6: DtypeWarning: Columns (3,10) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("internsip data.file.csv")
<ipython-input-35-9d8f3e36c62c>:29: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same

  sns.barplot(x='ProductPrice', y='Product_Category', data=cuisine_ratings_sorted, palette='viridis')
```





Identify any patterns or clusters of restaurants in specific areas.

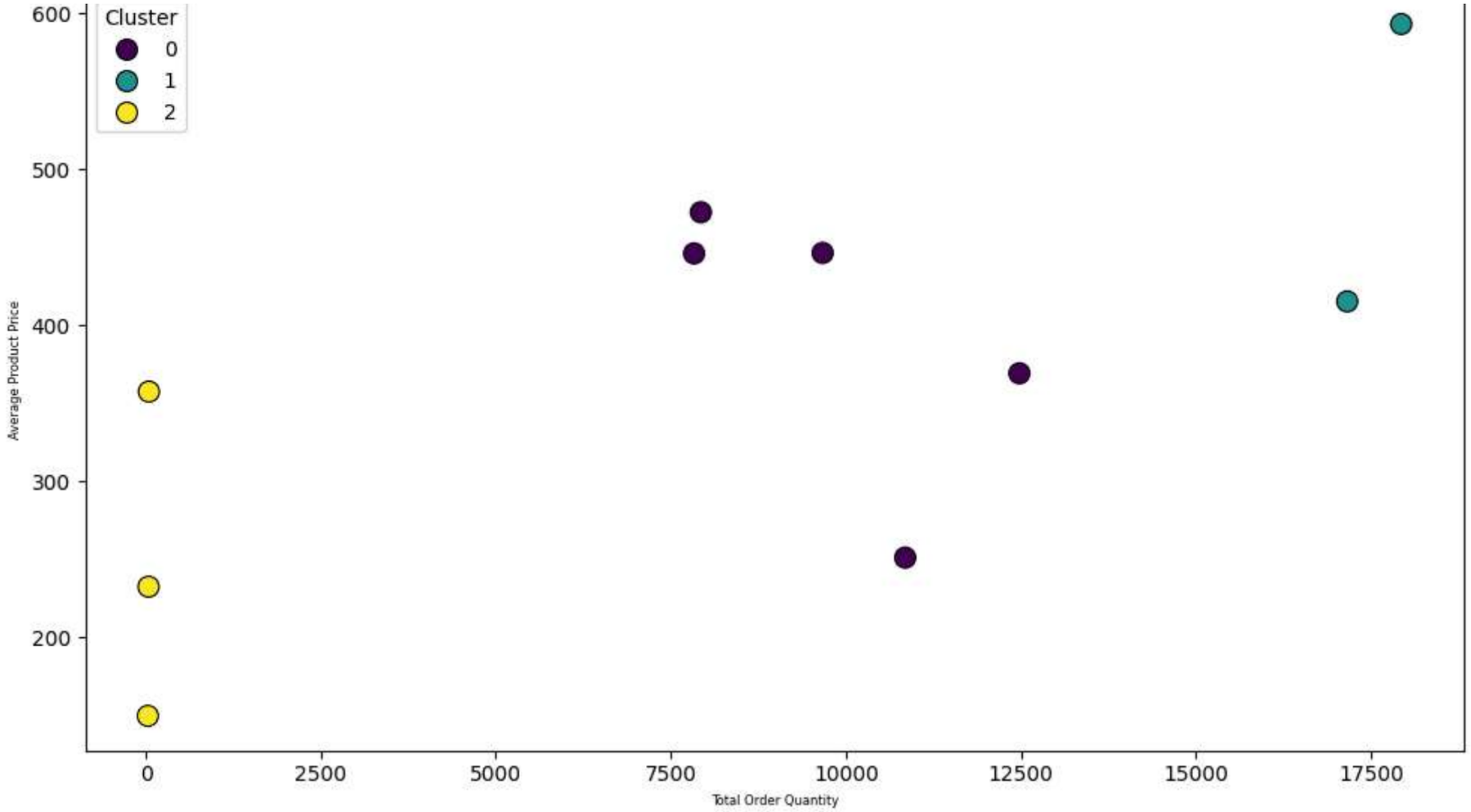
```
##Identify any patterns or clusters of restaurants in specific areas.

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans



# Step 1: Load dataset (with low_memory=False to avoid dtype warnings)
df = pd.read_csv("internsip data.file.csv", low_memory=False)

# Step 2: Check for mixed data types in the columns
# Check the data types of columns
print(df.dtypes)

# Check for non-numeric values in 'ProductPrice' and 'OrderQuantity' columns
print(df['ProductPrice'].unique())
print(df['OrderQuantity'].unique())
```

```
# Step 3: Convert 'ProductPrice' and 'OrderQuantity' to numeric, forcing errors to NaN
df['ProductPrice'] = pd.to_numeric(df['ProductPrice'], errors='coerce')
df['OrderQuantity'] = pd.to_numeric(df['OrderQuantity'], errors='coerce')

# Step 4: Handle missing values (NaN)
# Option 1: Drop rows with NaN values
df_cleaned = df.dropna(subset=['ProductPrice', 'OrderQuantity'])

# OR

# Option 2: Fill NaN values with 0 or any other default value
# df['ProductPrice'].fillna(0, inplace=True)
# df['OrderQuantity'].fillna(0, inplace=True)

# Step 5: Group by 'Region' and 'Country' and aggregate order quantities and product prices
region_country_data = df_cleaned.groupby(['Region', 'Country']).agg({
    'OrderQuantity': 'sum',  # Total order quantity per region/country
    'ProductPrice': 'mean'   # Average product price per region/country
}).reset_index()

# Step 6: Visualize Total Order Quantity by Region and Country
plt.figure(figsize=(6, 5))
sns.barplot(x='OrderQuantity', y='Region', data=region_country_data, hue='Country', palette='viridis')

plt.title("Total Order Quantity by Region and Country", fontsize=16)
plt.xlabel("Total Order Quantity", fontsize=12)
plt.ylabel("Region", fontsize=6)
plt.tight_layout()
plt.show()

# Step 7: Visualize Average Product Price by Region and Country
plt.figure(figsize=(6, 5))
sns.barplot(x='ProductPrice', y='Region', data=region_country_data, hue='Country', palette='viridis')
```

```
plt.title("Average Product Price by Region and Country", fontsize=10)
plt.xlabel("Average Product Price", fontsize=12)
plt.ylabel("Region", fontsize=6)
plt.tight_layout()
plt.show()

# Step 8: Apply K-means clustering to the aggregated data (OrderQuantity and ProductPrice)
kmeans = KMeans(n_clusters=3, random_state=42)
region_country_data['Cluster'] = kmeans.fit_predict(region_country_data[['OrderQuantity', 'ProductPrice']])

# Visualize the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='OrderQuantity', y='ProductPrice', hue='Cluster', data=region_country_data, palette='viridis', s=100, edgecolor='black')

plt.title("Clustering of Regions and Countries Based on Order Quantity and Product Price", fontsize=16)
plt.xlabel("Total Order Quantity", fontsize=6)
plt.ylabel("Average Product Price", fontsize=6)
plt.tight_layout()
plt.show()
```

```
OrderID              object
Region               object
Country              object
CustID               object
Customer_Name        object
ProductSKU           object
Product_Category     object
OrderLineItem         int64
OrderQuantity       float64
ProductCost         float64
ProductPrice         object
OrderDate            object
AcquisitionSource    object
TransactionID        object
Fraud                object
PaymentMethod        object
CardType             object
Gender               object
dtype: object
['699.0982' '3578.27' '3374.99' '3399.99' '2181.5625' '2071.4196'
 '2049.0982' '1000.4375' '2443.35' '8.99' '539.99' '3.99' '159' '32.6'
 '28.99' '4.99' '742.35' '33.6442' '1214.85' '34.99' '48.0673' '21.49'
 '2384.07' '1700.99' '9.99' '35' '21.98' '53.99' '24.99' '8.6442' '29.99'
 '769.49' '23.5481' '2.29' '564.99' '54.99' '63.5' '7.95' '120' '69.99'
 '16-08-2021']
[1.     2.     3.     1.8663]
```

Clustering of Regions and Countries Based on Order Quantity and Product Price

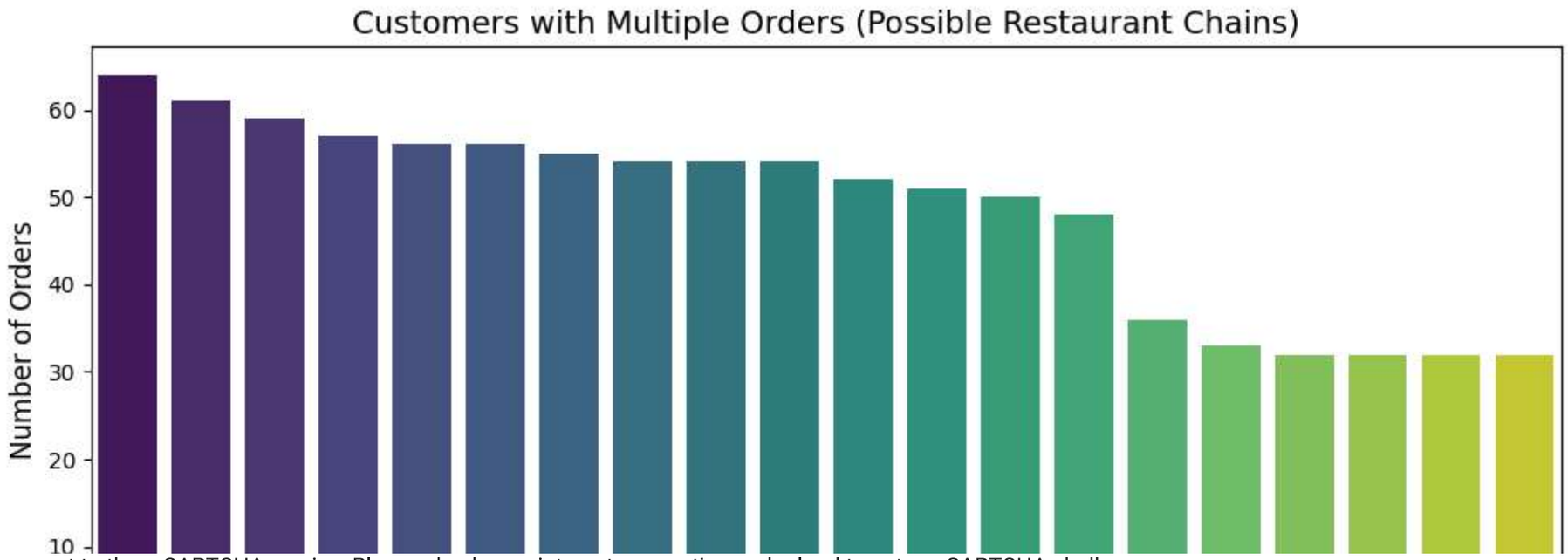

Cluster
0
1
2
600
500
400
300
200
Average Product Price
0
2500
5000
7500
10000
12500
15000
17500
Total Order Quantity

Identify if there are any restaurant chains present in the dataset.

```
#Identify if there are any restaurant chains present in the dataset.

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

# Step 1: Load the dataset with reduced memory usage
df = pd.read_csv("internsip data.file.csv", low_memory=False)
df

# Step 2: Reduce the dataset size by removing unnecessary columns (keep the most relevant ones)
df_reduced = df[['OrderID', 'Region', 'Country', 'Customer_Name', 'ProductSKU']]

# Step 3: Identify duplicated customers by counting occurrences of `Customer_Name`
customer_counts = df_reduced['Customer_Name'].value_counts()

# Step 4: Filter for customers with multiple orders (possible chains)
chain_candidates = customer_counts[customer_counts > 1].head(20)  # Limit to top 20 customers

# Step 5: Visualize the customers with multiple orders
plt.figure(figsize=(10, 6))
sns.barplot(x=chain_candidates.index, y=chain_candidates.values, palette='viridis')
plt.title("Customers with Multiple Orders (Possible Restaurant Chains)", fontsize=14)
plt.xlabel("Customer Name", fontsize=12)
plt.ylabel("Number of Orders", fontsize=12)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# Step 6: Analyze `ProductSKU` across different regions and countries
# Group by ProductSKU, Region, and Country and limit to top 20 ProductSKUs that appear across multiple regions
```

```
sku_region_counts = df_reduced.groupby(['ProductSKU', 'Region', 'Country']).size().reset_index(name='Count')

# Filter for ProductSKUs that appear in more than 1 region/country
sku_region_clusters = sku_region_counts[sku_region_counts['Count'] > 1].head(20)

# Step 7: Visualize the most frequent ProductSKU across regions/countries
plt.figure(figsize=(10, 6))
sns.countplot(data=sku_region_clusters, x='ProductSKU', palette='viridis')
plt.title("ProductSKU Across Regions/Countries (Possible Restaurant Chains)", fontsize=14)
plt.xlabel("ProductSKU", fontsize=12)
plt.ylabel("Frequency Across Regions/Countries", fontsize=12)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# Step 8: Identify customers with orders in multiple regions (possible chains)
customer_region_counts = df_reduced.groupby(['Customer_Name', 'Region']).size().reset_index(name='OrderCount')

# Filter customers with orders in more than 1 region
multi_region_customers = customer_region_counts[customer_region_counts['OrderCount'] > 1].head(20)

# Step 9: Visualize the customers with orders in multiple regions
plt.figure(figsize=(10, 6))
sns.countplot(data=multi_region_customers, x='Customer_Name', palette='viridis')
plt.title("Customers with Orders in Multiple Regions (Possible Restaurant Chains)", fontsize=14)
plt.xlabel("Customer Name", fontsize=12)
plt.ylabel("Number of Regions Ordered From", fontsize=12)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# Step 10: Print summary of multi-region customers
print("Customers with multiple orders across regions (possible chains):")
print(multi_region_customers)
```

```
<ipython-input-38-e04d01f8acfd>:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same

  sns.barplot(x=chain_candidates.index, y=chain_candidates.values, palette='viridis')
```



Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.