```
import pandas as pd
import sqlite3
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime

# --- 1. Load Datasets ---
print("--- Step 1: Loading Datasets ---")
try:
    providers_df = pd.read_csv('providers_data.csv')
    receivers_df = pd.read_csv('receivers_data.csv')
    food_listings_df = pd.read_csv('food_listings_data.csv')
    claims_df = pd.read_csv('claims_data.csv')
    print("All datasets loaded successfully!")
except FileNotFoundError as e:
    print(f"Error loading file: {e}. Please ensure all CSV files are uploaded to your Colab environment.")
    exit()

# Display initial info for a quick check
print("\n--- Providers Data Info ---")
providers_df.info()
print("\n--- Receivers Data Info ---")
receivers_df.info()
print("\n--- Food Listings Data Info ---")
food_listings_df.info()
print("\n--- Claims Data Info ---")
claims_df.info()

# --- 2. Data Preparation & Database Creation ---
print("\n--- Step 2: Data Preparation & Database Creation ---")

# Convert date columns to datetime objects in Pandas first for consistency
food_listings_df['Expiry_Date'] = pd.to_datetime(food_listings_df['Expiry_Date'])
claims_df['Timestamp'] = pd.to_datetime(claims_df['Timestamp'])
print("Date columns converted to datetime format in Pandas.")

# Create an in-memory SQLite database
conn = sqlite3.connect(':memory:')
cursor = conn.cursor()
print("In-memory SQLite database created.")

# Load Pandas DataFrames into SQLite tables
# Use if_exists='replace' to create new tables each time
providers_df.to_sql('providers', conn, if_exists='replace', index=False)
receivers_df.to_sql('receivers', conn, if_exists='replace', index=False)
food_listings_df.to_sql('food_listings', conn, if_exists='replace', index=False)
claims_df.to_sql('claims', conn, if_exists='replace', index=False)
print("Data loaded into SQLite tables.")

# Verify tables (optional)
print("\nTables in database:")
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
print(cursor.fetchall())

# --- 3. Data Analysis (SQL Queries) ---
print("\n--- Step 3: Data Analysis (SQL Queries) ---")

# Q1: How many food providers and receivers are there in each city?
print("\n--- Q1: Providers and Receivers per City ---")
query_q1 = """
SELECT
    COALESCE(p.City, r.City) AS City,
    COUNT(DISTINCT p.Provider_ID) AS NumProviders,
    COUNT(DISTINCT r.Receiver_ID) AS NumReceivers
FROM providers p
LEFT JOIN receivers r ON p.City = r.City
GROUP BY COALESCE(p.City, r.City)
ORDER BY NumProviders DESC, NumReceivers DESC;
"""
df_q1 = pd.read_sql_query(query_q1, conn)
print(df_q1.head())

# Q2: Which type of food provider contributes the most food?
print("\n--- Q2: Food Contribution by Provider Type ---")
query_q2 = """
SELECT
    fl.Provider_Type,
    SUM(fl.Quantity) AS TotalQuantity
FROM food_listings fl
GROUP BY fl.Provider_Type
ORDER BY TotalQuantity DESC;
```

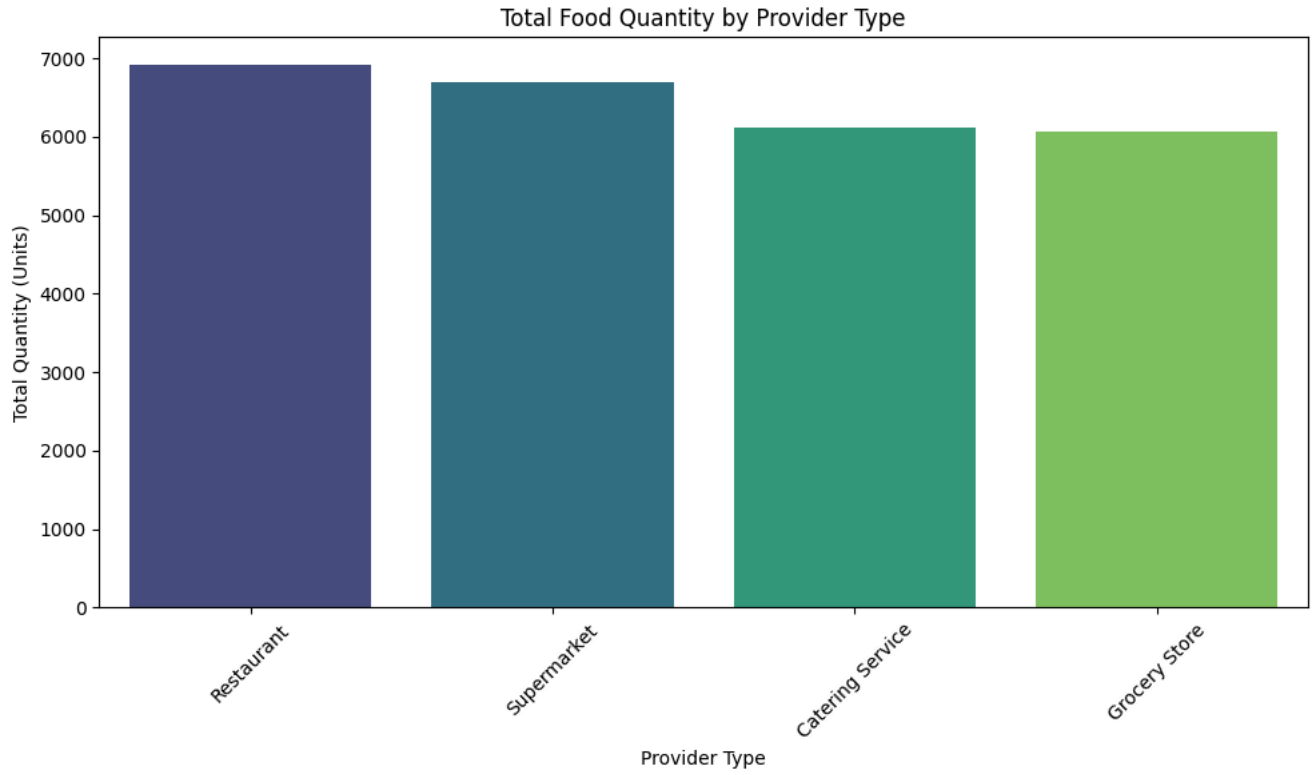
```
"""
df_q2 = pd.read_sql_query(query_q2, conn)
print(df_q2.head())
plt.figure(figsize=(10, 6))
sns.barplot(x='Provider_Type', y='TotalQuantity', data=df_q2, palette='viridis')
plt.title('Total Food Quantity by Provider Type')
plt.xlabel('Provider Type')
plt.ylabel('Total Quantity (Units)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Q3: What is the contact information of food providers in a specific city?
print("\n--- Q3: Contact Info for Providers in 'South Kellyville' ---")
specific_city_q3 = 'South Kellyville' # Example city
query_q3 = f"""
SELECT
    Name,
    Type,
    Contact
FROM providers
WHERE City = '{specific_city_q3}';
"""
df_q3 = pd.read_sql_query(query_q3, conn)
print(df_q3)

# Q4: Which receivers have claimed the most food?
print("\n--- Q4: Top Receivers by Claimed Food Quantity (Completed Claims) ---")
query_q4 = """
SELECT
    r.Name AS Receiver_Name,
    SUM(fl.Quantity) AS TotalClaimedQuantity
FROM claims c
JOIN food_listings fl ON c.Food_ID = fl.Food_ID
JOIN receivers r ON c.Receiver_ID = r.Receiver_ID
WHERE c.Status = 'Completed'
GROUP BY r.Name
ORDER BY TotalClaimedQuantity DESC;
"""
df_q4 = pd.read_sql_query(query_q4, conn)
print(df_q4.head())

# Q5: What is the total quantity of food available from all providers?
print("\n--- Q5: Total Food Available ---")
query_q5 = """
SELECT SUM(Quantity) AS TotalFoodAvailable FROM food_listings;
"""
df_q5 = pd.read_sql_query(query_q5, conn)
print(f"Total quantity of food available: {df_q5['TotalFoodAvailable'].iloc[0]} units")

# Q6: Which city has the highest number of food listings?
print("\n--- Q6: City with Highest Food Listings ---")
query_q6 = """
SELECT
    Location,
    COUNT(Food_ID) AS NumListings
FROM food_listings
GROUP BY Location
ORDER BY NumListings DESC
LIMIT 1;
"""
df_q6 = pd.read_sql_query(query_q6, conn)
print(df_q6)

# Q7: What are the most commonly available food types?
print("\n--- Q7: Most Commonly Available Food Types ---")
query_q7 = """
SELECT
    Food_Name,
    COUNT(Food_ID) AS NumListings
FROM food_listings
GROUP BY Food_Name
ORDER BY NumListings DESC
LIMIT 5;
"""
df_q7 = pd.read_sql_query(query_q7, conn)
print(df_q7.head())
plt.figure(figsize=(10, 6))
sns.barplot(x='Food_Name', y='NumListings', data=df_q7, palette='cubehelix')
plt.title('Most Commonly Available Food Types')
plt.xlabel('Food Type')
plt.ylabel('Number of Listings')
plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()

# Q8: How many food claims have been made for each food item?
print("\n--- Q8: Number of Claims per Food Item ---")
query_q8 = """
SELECT
    fl.Food_Name,
    COUNT(c.Claim_ID) AS NumClaims
FROM claims c
JOIN food_listings fl ON c.Food_ID = fl.Food_ID
GROUP BY fl.Food_Name
ORDER BY NumClaims DESC;
"""
df_q8 = pd.read_sql_query(query_q8, conn)
print(df_q8.head())

# Q9: Which provider has had the highest number of successful food claims?
print("\n--- Q9: Providers with Highest Number of Successful Claims ---")
query_q9 = """
SELECT
    p.Name AS Provider_Name,
    COUNT(c.Claim_ID) AS SuccessfulClaims
FROM claims c
JOIN food_listings fl ON c.Food_ID = fl.Food_ID
JOIN providers p ON fl.Provider_ID = p.Provider_ID
WHERE c.Status = 'Completed'
GROUP BY p.Name
ORDER BY SuccessfulClaims DESC;
"""
df_q9 = pd.read_sql_query(query_q9, conn)
print(df_q9.head())

# Q10: What percentage of food claims are completed vs. pending vs. canceled?
print("\n--- Q10: Percentage of Claim Statuses ---")
query_q10 = """
SELECT
    Status,
    CAST(COUNT(Claim_ID) AS REAL) * 100 / (SELECT COUNT(*) FROM claims) AS Percentage
FROM claims
GROUP BY Status
ORDER BY Percentage DESC;
"""
df_q10 = pd.read_sql_query(query_q10, conn)
print(df_q10)
plt.figure(figsize=(8, 8))
plt.pie(df_q10['Percentage'], labels=df_q10['Status'], autopct='%1.1f%%', startangle=90, colors=sns.color_palette('pastel'))
plt.title('Percentage of Food Claim Statuses')
plt.axis('equal')
plt.show()

# Q11: What is the average quantity of food claimed per receiver?
print("\n--- Q11: Average Quantity Claimed per Receiver (Completed Claims) ---")
query_q11 = """
SELECT
    r.Name AS Receiver_Name,
    AVG(fl.Quantity) AS AvgQuantityClaimed
FROM claims c
JOIN food_listings fl ON c.Food_ID = fl.Food_ID
JOIN receivers r ON c.Receiver_ID = r.Receiver_ID
WHERE c.Status = 'Completed'
GROUP BY r.Name
ORDER BY AvgQuantityClaimed DESC;
"""
df_q11 = pd.read_sql_query(query_q11, conn)
print(df_q11.head())

# Q12: Which meal type (breakfast, lunch, dinner, snacks) is claimed the most?
print("\n--- Q12: Most Claimed Meal Type (Completed Claims) ---")
query_q12 = """
SELECT
    fl.Meal_Type,
    COUNT(c.Claim_ID) AS NumClaims
FROM claims c
JOIN food_listings fl ON c.Food_ID = fl.Food_ID
WHERE c.Status = 'Completed'
GROUP BY fl.Meal_Type
ORDER BY NumClaims DESC;
"""
df_q12 = pd.read_sql_query(query_q12, conn)
print(df_q12)
plt.figure(figsize=(8, 6))
sns.barplot(x='Meal_Type', y='NumClaims', data=df_q12, palette='rocket')
```

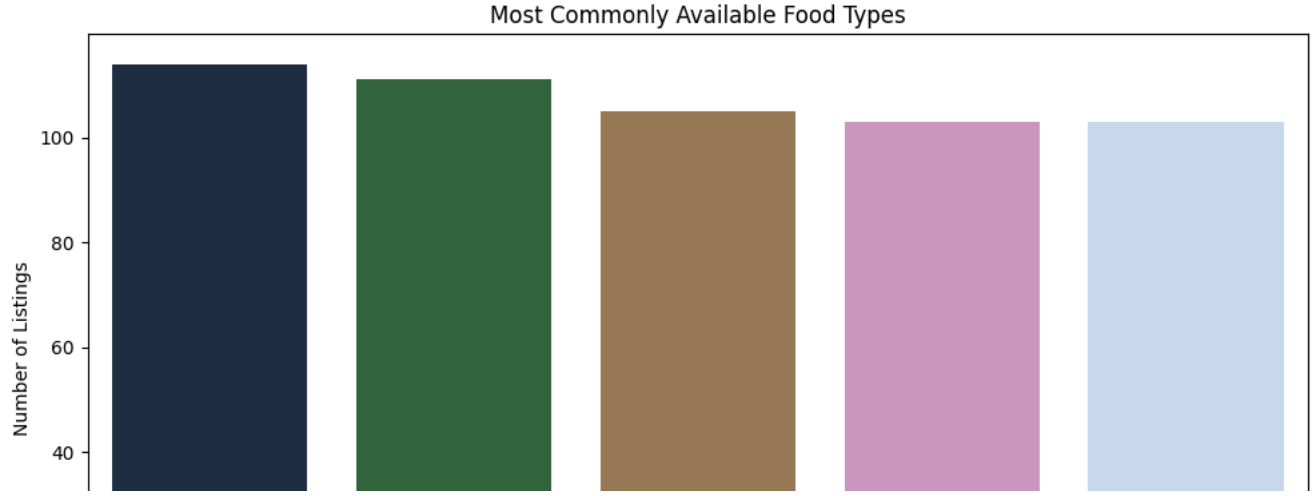
```
sns.barplot(x='Meal_Type', y='NumClaims', data=df_q12, palette='rocket')
plt.title('Most Claimed Meal Types')
plt.xlabel('Meal Type')
plt.ylabel('Number of Claims')
plt.tight_layout()
plt.show()

# Q13: What is the total quantity of food donated by each provider?
print("\n--- Q13: Total Food Donated by Each Provider ---")
query_q13 = """
SELECT
    p.Name AS Provider_Name,
    SUM(fl.Quantity) AS TotalDonatedQuantity
FROM food_listings fl
JOIN providers p ON fl.Provider_ID = p.Provider_ID
GROUP BY p.Name
ORDER BY TotalDonatedQuantity DESC;
"""
df_q13 = pd.read_sql_query(query_q13, conn)
print(df_q13.head())

# Q14: Food items nearing expiry (e.g., expiring within the next 7 days from today's date)
print("\n--- Q14: Food Items Nearing Expiry (Next 7 Days) ---")
# Note: SQLite's date functions are limited. We'll use a fixed date for demonstration
# In a real app, you'd pass a dynamic current date.
today_str = datetime.now().strftime('%Y-%m-%d') # Get current date as string
query_q14 = f"""
SELECT
    Food_Name,
    Quantity,
    Expiry_Date,
    Location
FROM food_listings
WHERE Expiry_Date > '{today_str}' AND Expiry_Date <= DATE('{today_str}', '+7 days')
ORDER BY Expiry_Date ASC;
"""
df_q14 = pd.read_sql_query(query_q14, conn)
print(df_q14.head())

# Q15: Claims by receiver type
print("\n--- Q15: Number of Claims by Receiver Type ---")
query_q15 = """
SELECT
    r.Type AS Receiver_Type,
    COUNT(c.Claim_ID) AS NumClaims
FROM claims c
JOIN receivers r ON c.Receiver_ID = r.Receiver_ID
GROUP BY r.Type
ORDER BY NumClaims DESC;
"""
df_q15 = pd.read_sql_query(query_q15, conn)
print(df_q15)
plt.figure(figsize=(8, 6))
sns.barplot(x='Receiver_Type', y='NumClaims', data=df_q15, palette='crest')
plt.title('Number of Claims by Receiver Type')
plt.xlabel('Receiver Type')
plt.ylabel('Number of Claims')
plt.tight_layout()
plt.show()

# Close the database connection
conn.close()
print("\n--- EDA and Data Analysis Complete. Database connection closed. ---")
```

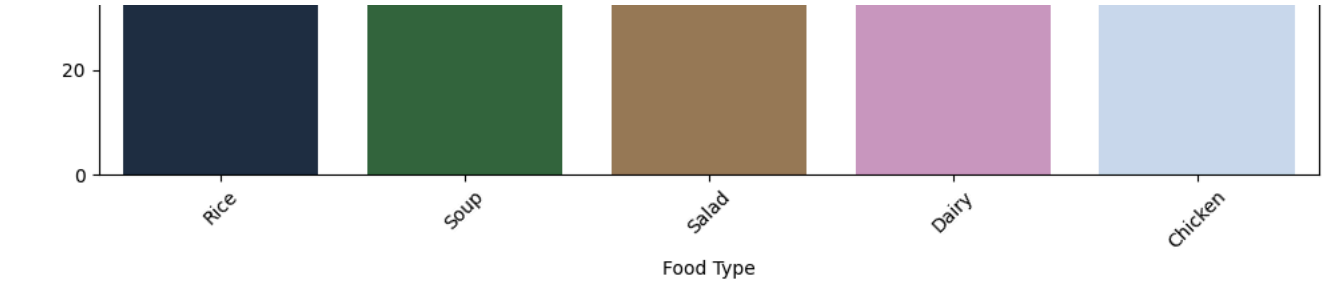
```
--- Step 1: Loading Datasets ---
All datasets loaded successfully!

--- Providers Data Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Provider_ID  1000 non-null   int64
 1   Name         1000 non-null   object
 2   Type         1000 non-null   object
 3   Address      1000 non-null   object
 4   City         1000 non-null   object
 5   Contact      1000 non-null   object
dtypes: int64(1), object(5)
memory usage: 47.0+ KB

--- Receivers Data Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Receiver_ID  1000 non-null   int64
 1   Name         1000 non-null   object
 2   Type         1000 non-null   object
 3   City         1000 non-null   object
 4   Contact      1000 non-null   object
dtypes: int64(1), object(4)
memory usage: 39.2+ KB

--- Food Listings Data Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Food_ID        1000 non-null   int64
 1   Food_Name      1000 non-null   object
 2   Quantity       1000 non-null   int64
 3   Expiry_Date    1000 non-null   object
 4   Provider_ID    1000 non-null   int64
 5   Provider_Type  1000 non-null   object
 6   Location       1000 non-null   object
 7   Food_Type      1000 non-null   object
 8   Meal_Type      1000 non-null   object
dtypes: int64(3), object(6)
memory usage: 70.4+ KB

--- Claims Data Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Claim_ID     1000 non-null   int64
 1   Food_ID      1000 non-null   int64
 2   Receiver_ID  1000 non-null   int64
 3   Status       1000 non-null   object
 4   Timestamp    1000 non-null   object
dtypes: int64(3), object(2)
memory usage: 39.2+ KB

--- Step 2: Data Preparation & Database Creation ---
Date columns converted to datetime format in Pandas.
In-memory SQLite database created.
Data loaded into SQLite tables.

Tables in database:
[('providers',), ('receivers',), ('food_listings',), ('claims',)]

--- Step 3: Data Analysis (SQL Queries) ---

--- Q1: Providers and Receivers per City ---
                       City  NumProviders  NumReceivers
0                New Carol             3             0
1  South Christopherborough             3             0
2              Lake Michael             2             1
3                New Daniel             2             1
4            North Michelle             2             1

--- Q2: Food Contribution by Provider Type ---
      Provider_Type  TotalQuantity
0        Restaurant           6923
1       Supermarket           6696
2  Catering Service           6116
3     Grocery Store           6059
/tmp/ipython-input-3486242655.py:86: FutureWarning:
```

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

  sns.barplot(x='Provider_Type', y='TotalQuantity', data=df_q2, palette='viridis')
```

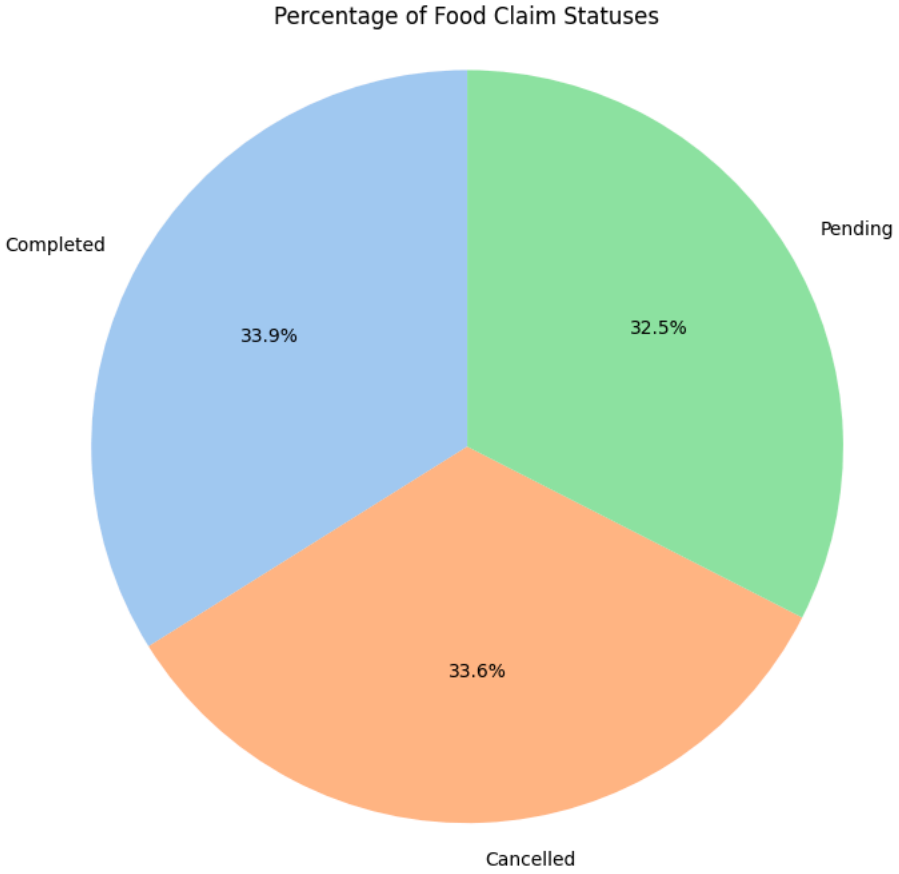



```
--- Q3: Contact Info for Providers in 'South Kellyville' ---
            Name           Type        Contact
0  Figueroa-Soto  Grocery Store  (599)442-0494

--- Q4: Top Receivers by Claimed Food Quantity (Completed Claims) ---
     Receiver_Name  TotalClaimedQuantity
0     Derek Potter                    99
1   Steven Griffin                    89
2   Peter Gonzalez                    83
3   Donna Williams                    82
4  Timothy Garrett                    80

--- Q5: Total Food Available ---
Total quantity of food available: 25794 units

--- Q6: City with Highest Food Listings ---
        Location  NumListings
0  South Kathryn            6

--- Q7: Most Commonly Available Food Types ---
  Food_Name  NumListings
0      Rice          114
1      Soup          111
2     Salad          105
3     Dairy          103
4   Chicken          103
/tmp/ipython-input-3486242655.py:160: FutureWarning:

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

  sns.barplot(x='Food_Name', y='NumListings', data=df_q7, palette='cubehelix')
```



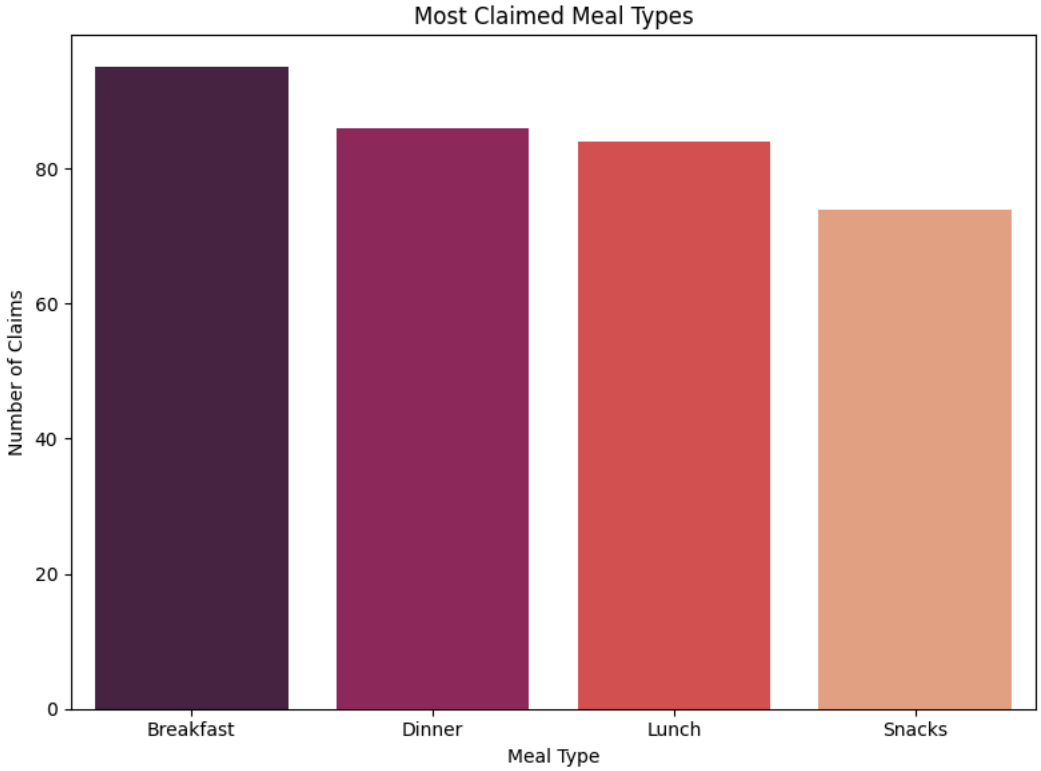



```
--- Q8: Number of Claims per Food Item ---
  Food_Name  NumClaims
0      Rice        122
1      Soup        114
2     Dairy        110
3      Fish        108
4     Salad        106

--- Q9: Providers with Highest Number of Successful Claims ---
                Provider_Name  SuccessfulClaims
0                 Barry Group                 5
1                  Miller Inc                 4
2  Harper, Blake and Alexander                4
3           Butler-Richardson                 4
4    Barnes, Castro and Curtis                4

--- Q10: Percentage of Claim Statuses ---
      Status  Percentage
0  Completed        33.9
1  Cancelled        33.6
2    Pending        32.5
```




```
--- Q11: Average Quantity Claimed per Receiver (Completed Claims) ---
       Receiver_Name  AvgQuantityClaimed
0  Thomas Villanueva                50.0
1        Nancy Silva                50.0
2        Nancy Jones                50.0
3      Mary Franklin                50.0
4         Mark Lewis                50.0

--- Q12: Most Claimed Meal Type (Completed Claims) ---
   Meal_Type  NumClaims
0  Breakfast         95
1     Dinner         86
2      Lunch         84
3     Snacks         74
/tmp/ipython-input-3486242655.py:247: FutureWarning:

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

```
  sns.barplot(x='Meal_Type', y='NumClaims', data=df_q12, palette='rocket')
```




```
--- Q13: Total Food Donated by Each Provider ---
                Provider_Name  TotalDonatedQuantity
0                  Miller Inc                   217
1                 Barry Group                   179
2  Evans, Wright and Mitchell                   158
3                 Smith Group                   150
4                Campbell LLC                   145

--- Q14: Food Items Nearing Expiry (Next 7 Days) ---
Empty DataFrame
Columns: [Food_Name, Quantity, Expiry_Date, Location]
Index: []

--- Q15: Number of Claims by Receiver Type ---
  Receiver_Type  NumClaims
0           NGO        272
1       Charity        268
2       Shelter        230
3    Individual        230
/tmp/ipython-input-3486242655.py:300: FutureWarning:

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

  sns.barplot(x='Receiver_Type', y='NumClaims', data=df_q15, palette='crest')
```