food-contaminant-4

May 14, 2025

1 DATA DRIVEN INSIGHTS INTO FOOD CONTAMINATION

1.1 OLAPEJU ESUOLA, MSBA

1.2 UNIVERSITY OF LOUISVILLE

1.3 MAY 15, 2025

1.4 Abstract

Food contamination remains a critical issue for public health, requiring effective monitoring and management. This project aims to explore a comprehensive dataset on food contamination, focusing on the analysis of contaminant types, food categories, and associated LOD (Limit of Detection) and LOQ (Limit of Quantification) values. Through Exploratory Data Analysis (EDA), this study identifies patterns and trends within the data, including the distribution of contaminants across different food categories, variations in contaminant levels, and insights into missing data. Additionally, the project investigates potential relationships between food origin, contaminant levels, and yearly trends. By visualizing and analyzing these factors, the project seeks to provide a deeper understanding of food contamination dynamics, ultimately contributing to better food safety practices and regulatory compliance.

1.5 Introduction

Food contamination, particularly due to chemical contaminants, poses a significant risk to public health and has led to substantial consumer distrust. Detecting and managing food contamination is a vital part of ensuring food safety and regulatory compliance. Traditionally, food contamination detection has relied on laboratory testing, sensory evaluation, and other manual methods. These methods, while effective, are often time-consuming and resource-intensive.

This project focuses on exploring a food contamination dataset, examining the distribution and levels of contaminants such as Aflatoxin, Lead, and Dichlorvos across various food categories and food origins. By performing Exploratory Data Analysis (EDA), this project identifies key trends and relationships within the dataset. The insights obtained will support food safety management efforts and provide a clearer understanding of contamination patterns over time.

1.6 Methodology

The methodology for this project is based on data exploration and statistical analysis to identify meaningful patterns in the food contamination data. The process involves cleaning the dataset, handling missing values, analyzing the distribution of key variables such as LOD (Limit of Detec-

tion) and LOQ (Limit of Quantification), and investigating potential relationships between food categories and contaminants.

Data Cleaning: The dataset is first cleaned by handling missing values, removing duplicates, and ensuring that the variables are appropriately formatted for analysis.

Exploratory Data Analysis (EDA): Descriptive statistics and visualizations are used to gain insights into the distribution of LOD, LOQ, and contaminant types across food categories and regions.

Hypothesis Testing: Statistical tests, including t-tests and chi-square tests, are performed to assess relationships between food categories, contaminants, and other variables.

Forecasting: A basic time-series forecasting is used to predict potential future trends in contaminant levels based on historical data.

1.6.1 Research Design

This research follows a descriptive research design where the primary goal is to explore the existing dataset to uncover patterns, trends, and relationships. The dataset includes information about various food products, their contaminants, food categories, LOD, LOQ, and food origin. The design allows for in-depth analysis of these variables through visualization, statistical analysis, and time series forecasting.

The analysis is structured around the following questions:

What are the most common contaminants found across different food categories?

Are there significant differences in contamination levels between imported and domestic food products?

How do LOD and LOQ values vary by food category and contaminant over time?

Data Collection The dataset used in this study is provided by PAHO (Pan American Health Organization) and includes contamination data from 2016 to 2021. The dataset consists of 18853 rows and 16 columns, detailing the presence and concentration of various chemical contaminants in food products. It includes information on the food category, contaminant name, LOD, LOQ, and food origin, among other attributes.

Data Features:

Contaminant Name: The type of contaminant found in the food.

Food Category: The category under which the food product falls (e.g., meat, vegetables).

LOD and LOQ: The concentration of contaminants, measured in µg/kg.

Food Origin: Indicates whether the food is imported or domestic.

The dataset is cleaned to handle missing values, and only relevant columns are selected for analysis.

Exploratory Data Analysis (EDA) Exploratory Data Analysis (EDA) is a critical step in understanding the dataset and gaining insights into its structure. For this project, EDA involves:

Summary Statistics: Generating summary statistics for numerical and categorical variables to understand distributions and key metrics.

Missing Data: Identifying and handling missing data, either by imputation or removal.

Data Visualization: Barplot for top 10 contaminants.

Correlation plots to assess the relationships between LOD and LOQ values.

Food Category Distribution: Analyzing how contaminants are distributed across different food categories and regions.

EDA helps us gain a deeper understanding of the patterns in the data and sets the stage for further analysis.

Forecasting The forecasting component of this analysis focuses on predicting potential future trends in LOD values based on historical data. A time-series forecasting model (ARIMA) is used to predict how LOD values might change over the next few years.

The process involves:

Data Preprocessing: Grouping LOD values by year to create a time series.

Model Selection: The ARIMA model is used, which helps forecast future values based on historical trends.

Model Evaluation: The model's accuracy is assessed using performance metrics like Mean Absolute Error (MAE) and Mean Squared Error (MSE).

The forecast aims to provide an estimate of how contaminant levels might evolve over time, aiding in future food safety planning.

Hypothesis Testing Hypothesis testing is used to assess relationships between variables and to determine if observed differences are statistically significant. The following tests were applied:

One-Sample t-Test: Used to compare the average LOD value for a specific contaminant (Aflatoxin) against a known threshold.

Two-Sample t-Test: Used to compare LOD values between domestic and imported food products for a specific contaminant.

Chi-Square Test: Applied to test the association between food category and contaminant name. This helps determine if certain contaminants are more likely to be found in specific food categories.

ANOVA: Used to compare LOD values across multiple food categories to check if contamination levels vary significantly between them.

The results of the hypothesis tests help validate the relationships between different variables and provide insights into contamination patterns.

1.7 Results & Insights

Summary Statistics of the Cleaned Dataset: After cleaning the dataset and handling missing values, the summary statistics for key variables such as LOD (Limit of Detection), LOQ (Limit of Quantification), and Year were generated:

- LOD: The Limit of Detection (LOD) has a wide range of values. The minimum LOD is extremely low (0.00005 µg/kg), while the maximum is as high as 290 µg/kg. The mean LOD across the dataset is approximately 0.37 µg/kg with a standard deviation of 3.01 µg/kg. This shows substantial variation in the detection levels of contaminants in food products.
- LOQ: The LOQ ranges from 0.00012 µg/kg to 200 µg/kg. The mean LOQ is around 1.51 µg/kg, with a larger standard deviation of 8.66 µg/kg, indicating significant variance in the LOQ values across the dataset.
- Year: The dataset spans several years, with the most frequent data being from 2016 and 2017, and a gradual shift in years with few records beyond 2021. The mean year is approximately 2016, indicating a focus on more recent food contamination data.

Exploratory Data Analysis (EDA):

- Contaminant Frequency: The top 10 contaminants identified in the dataset were dominated by Aflatoxin (total), Lead, Dichlorvos, and Difenoconazole, among others. The distribution of these contaminants across various food categories revealed that Aflatoxin (total) had the highest frequency in Nuts and oilseeds, while Lead was common in Meat and meat products. This indicates that certain contaminants are more prevalent in specific food categories, which could be helpful for targeted monitoring and prevention.
- Food State and Food Origin: The analysis revealed that Raw food samples were significantly more prevalent than cooked ones. Additionally, Imported foods were more common than Domestic ones in the dataset, highlighting potential concerns regarding the quality control of imported food products.

Time Series Forecasting:

A SARIMAX model was used to forecast future LOD values for contaminants over time. The forecast indicated that the LOD for contaminants would remain relatively stable in the coming years, with no significant upward or downward trends. This stability in detection levels suggests that while contamination continues to be a concern, the quality of food monitoring and detection systems may be holding steady.

- Model Evaluation: The Mean Absolute Error (MAE) and Mean Squared Error (MSE) were calculated for the model, yielding values of 0.298 and 0.138, respectively. These results suggest a fairly accurate model for predicting future contamination levels, though there is room for improvement, especially in terms of addressing outliers or sudden shifts in the data.

Hypothesis Testing:

- ANOVA Test for LOD Across Food Categories: The hypothesis testing confirmed that there is a significant difference in the LOD values across different food categories (F-statistic: 20.72, p-value: 1.3151833232988207e-81). This suggests that certain food categories, such as Nuts and oilseeds, have higher contamination levels than others.
- Chi-square Test for Food Category and Contaminant Association: The Chi-square test revealed a strong association between food category and contaminant name (Chi2 Statistic: 51155.97, p-value: 0.0). This implies that specific contaminants are more likely to appear in certain types of food, reinforcing the idea that targeted analysis by food category is necessary for food safety management.

- T-tests for LOD Differences by Food Origin: Further hypothesis testing showed that the LOD for Aflatoxin is significantly different between Imported and Domestic food (t-statistic: -53.106, p-value: 0.0). This finding suggests that food origin plays an important role in contamination levels, which could be due to differences in sourcing, handling, or regulations across countries.

Key Observations:

- The food categories most affected by contamination are Meat and meat products, Fats and oils of animal and vegetable, and Vegetables and vegetable products.
- Imported foods show higher contamination levels for certain contaminants, suggesting the need for improved quality checks and monitoring for imported goods.
- The LOD and LOQ metrics indicate varying levels of contaminant detection across different food categories and origins. The variations in detection levels point to the importance of enhancing food safety practices, particularly for food categories prone to higher contamination.

Visualizations and Insights:

- Pie charts and bar plots visually highlighted the distribution of food origin (Domestic vs. Imported) across different contaminants, indicating which products require more attention during inspections.
- Top 5 food categories for the major contaminants such as Lead, Aflatoxin, and Dichlorvos were visualized, providing further insights into the areas where contamination is most prevalent.

1.8 Coding

[2]:
```
import pandas as pd

# Load the dataset from the Excel file
df = pd.read_excel('Food Contamination Data_ WHO.xlsx')

# Randomly select 20,000 rows
df_sampled = df.sample(n=20000, random_state=42)

# Display the first few rows of the sampled data to check
print(df_sampled.head())
```

```
        RecordType RegionCode RegionName ContaminantName  \
353339  Individual       PAHO       PAHO            Lead
16878   Individual       PAHO       PAHO      Cyromazine
117423  Individual       PAHO       PAHO    Dimethomorph
274255  Individual       PAHO       PAHO   Fenpyroximate
15153   Individual       PAHO       PAHO       Abamectin

                                        FoodCategory  \
353339  Meat and meat products (including edible offal)
16878                          Starchy roots and tubers
117423                         Fruit and fruit products
```

```
274255                                Nuts and oilseeds
15153                               Snacks and desserts

                                        FoodName FoodCode       LocalFoodName  \
353339  Kidney of cattle, goats, pigs and sheep  MO 0098          rim bovino
16878                                 Arrowroot  VR 0573  Arrowroot Products
117423                                  Grapes  FB 0269               Grape
274255                                OILSEEDS  SO 0088        Seed - other
15153                               Snack food     15.1        Chips - Corn

       FoodStateName ResultText  …     LOD      LOQ  Year  \
353339           Raw         ND  …  33.000  100.000  2016
16878        Unknown         ND  …   0.010    0.020  2016
117423           Raw         ND  …   0.001    0.005  2016
274255           Raw         ND  …   0.010    0.011  2016
15153        Unknown         ND  …   0.010    0.020  2016

        RepresentativenessName LabNumber  FoodOriginName  \
353339         Random sampling     102.0        Domestic
16878          Random sampling       1.0         Unknown
117423         Random sampling       6.0        Imported
274255         Random sampling       3.0        Imported
15153          Random sampling       1.0         Unknown

             AnalyticalQAName ResultBasisName                 PortionTypeName  \
353339  Officially accredited           As is  Total food (edible + inedible)
16878   Officially accredited           As is                     Edible only
117423  Officially accredited           As is                     Edible only
274255  Officially accredited           As is                     Edible only
15153   Officially accredited           As is                     Edible only

       SerialNumber
353339        14331
16878     121624767
117423    287320465
274255   1851285869
15153   -1239224930

[5 rows x 21 columns]
```

Random sampling is a technique used to select a subset of data from a larger dataset in such a way that every individual data point has an equal probability of being chosen. In this analysis, random sampling is employed to ensure that the selected sample represents a diverse cross-section of the entire dataset. This approach reduces bias and avoids overfitting to a specific year or category. It also helps in mitigating class imbalance, ensuring that the sample maintains a more balanced distribution of the key variables, such as contaminant types or food categories.

By using random sampling, we ensure that the selected 20,000 rows of data provide a broad view of the overall trends and patterns in food contamination, making the results more generalizable.

This method is particularly useful when we want to analyze patterns across various years or food categories without being restricted to a particular subset of data.

Data Cleaning

[3]:
```
# Get the unique values for each column in the dataset
unique_values = df_sampled.nunique()
print("Unique values per column:")
print(unique_values)

# Display unique values for specific columns
print("\nUnique values for 'ContaminantName':")
print(df_sampled['ContaminantName'].unique())
```

```
Unique values per column:
RecordType                    2
RegionCode                    1
RegionName                    1
ContaminantName              99
FoodCategory                 23
FoodName                    324
FoodCode                    323
LocalFoodName              1499
FoodStateName                 3
ResultText                 1172
UnitName                      3
LOD                         325
LOQ                         272
Year                          9
RepresentativenessName        3
LabNumber                    19
FoodOriginName                4
AnalyticalQAName              4
ResultBasisName               5
PortionTypeName               2
SerialNumber              14704
dtype: int64

Unique values for 'ContaminantName':
['Lead' 'Cyromazine' 'Dimethomorph' 'Fenpyroximate' 'Abamectin'
 'Indoxacarb' 'Chlorpyrifos-methyl' 'Cadmium' 'Difenoconazole'
 'Pyraclostrobin' 'Malathion' 'Arsenic (total)' 'Cyproconazole'
 'Imidacloprid' 'DL PCB 114' 'Thiabendazole' 'Aflatoxin (total)'
 'Spirotetramat' 'Dichlorvos' 'Dichlobenil' 'Methoxyfenozide'
 'Cypermethrin' 'Cyhalothrin' 'Cycloxydim' 'Dioxin like 1,2,3,4,7,8-HxCDF'
 'Dioxin 1,2,3,4,6,7,8,9-OCDD' 'Dioxin-like 2,3,7,8-TCDF' 'Buprofezin'
 'Triadimenol' 'Phosmet' 'Aflatoxin G2' 'Methamidophos' 'Flutriafol'
 'Fenbuconazole' 'Thiamethoxam' 'Tebuconazole' 'Carbofuran' 'Profenofos'
```

```
'Dicofol' 'Fenvalerate' 'Aflatoxin G5' 'Mercury'
'Dioxin like 1,2,3,4,6,7,8-HpCDF' 'Fenpropathrin'
'Dioxin like 1,2,3,4,7,8,9-HpCDF' 'Dioxin 1,2,3,4,6,7,8-HpCDD'
'Clothianidin' 'Aflatoxin B2' 'Cyfluthrin/beta-cyfluthrin' 'Aflatoxin B1'
'Dioxin-like 2,3,4,6,7,8-HxCDF' 'Triforine' 'Deltamethrin'
'Dioxin 1,2,3,7,8,9-HxCDD' 'DL PCB 77' 'Arsenic (inorganic)'
'Triflumizole' 'Dioxin like 1,2,3,4,6,7,8,9-OCDF' 'Etofenprox'
'Teflubenzuron' 'NDL-PCB 153' 'Phorate' 'Tolfenpyrad'
'Dioxin 1,2,3,7,8-PeCDD' 'Aflatoxin G1' 'Arsenic (organic)'
'Dioxin like 1,2,3,7,8,9-HxCDF' 'Prothioconazole' 'DL PCB 169'
'Triazophos' 'DL PCB 105' 'DL PCB 157' 'DL PCB 167' 'DL PCB 123'
'Propamocarb' 'NDL-PCB 180' 'Dioxin 2,3,7,8-TCDD' 'NDL-PCB 138'
'DL PCB 189' 'Diquat' 'DL PCB 118' 'T-2 toxin' 'NDL-PCB 101' 'HT-2 toxin'
'Dioxin like 1,2,3,7,8-PeCDF' 'Scopolamine' 'NDL-PCB 28'
'Dioxin 1,2,3,6,7,8-HxCDD' 'Dioxin like 1,2,3,6,7,8-HxCDF' 'DL PCB 126'
'NDL-PCB 52' 'DL PCB 81' 'Atropine' 'Dioxin 1,2,3,4,7,8-HxCDD'
'DL PCB 156' 'Dioxin-like 2,3,4,7,8-PeCDF' 'Methyl mercury'
'Ochratoxin A' nan 'Dioxins (WHO TEFs)']
```

[4]:
```
# Summary statistics for numerical columns
summary_stats = df_sampled.describe()
print("\nSummary statistics of the dataset:")
print(summary_stats)

# Summary for categorical columns (e.g., 'ContaminantName', 'FoodCategory')
categorical_summary = df_sampled.describe(include=['object'])
print("\nSummary of categorical columns:")
print(categorical_summary)
```

```
Summary statistics of the dataset:
                LOD           LOQ         Year     LabNumber
count  19242.000000  19443.000000  20000.00000  14853.000000
mean       0.363110      5.040139   2016.70985     11.476604
std        2.980149     25.845832      1.41812    128.513881
min        0.000050      0.000120   2016.00000      0.000000
25%        0.005000      0.010000   2016.00000      1.000000
50%        0.010000      0.020000   2016.00000      2.000000
75%        0.025000      0.170000   2017.00000      6.000000
max      290.000000    200.000000   2024.00000   2619.000000

Summary of categorical columns:
        RecordType RegionCode RegionName    ContaminantName  \
count        18017      20000      20000              19996
unique           2          1          1                 99
top     Individual       PAHO       PAHO  Aflatoxin (total)
freq         17987      20000      20000               1323
```

```
                                                FoodCategory FoodName FoodCode  \
count                                                  20000    20000    20000
unique                                                    23      324      323
top     Meat and meat products (including edible offal)   Peanut  SO 0697
freq                                                    3734     1412     1412

           LocalFoodName FoodStateName ResultText UnitName  \
count              20000         17570      19914    20000
unique              1499             3       1172        3
top     peanuts, shelled       Unknown         ND    mg/kg
freq                1316          8741      14739    14164

       RepresentativenessName FoodOriginName       AnalyticalQAName  \
count                   20000          19618                  18763
unique                      3              4                      4
top           Random sampling       Domestic  Officially accredited
freq                    19822           8542                  18227

       ResultBasisName PortionTypeName  SerialNumber
count            20000           20000         20000
unique               5               2         14704
top              As is     Edible only    1429302769
freq             18497           19726             6
```

[5]:
```
# Check for missing values in the dataset
missing_values = df_sampled.isnull().sum()
print("\nMissing values per column:")
print(missing_values)
```

```
Missing values per column:
RecordType                1983
RegionCode                   0
RegionName                   0
ContaminantName              4
FoodCategory                 0
FoodName                     0
FoodCode                     0
LocalFoodName                0
FoodStateName             2430
ResultText                  86
UnitName                     0
LOD                        758
LOQ                        557
Year                         0
RepresentativenessName       0
LabNumber                 5147
FoodOriginName             382
```

```
AnalyticalQAName          1237
ResultBasisName              0
PortionTypeName              0
SerialNumber                 0
dtype: int64
```

[6]:
```
# Display unique values for specific columns
print("\nUnique values for 'FoodStateName':")
print(df_sampled['AnalyticalQAName'].unique())
```

```
Unique values for 'FoodStateName':
['Officially accredited' nan 'Internal quality assurance only'
 'Successful proficiency testing' 'Unknown']
```

[7]:
```
# Check for duplicate rows
duplicate_rows = df_sampled.duplicated().sum()
print(f"\nNumber of duplicate rows: {duplicate_rows}")
```

```
Number of duplicate rows: 18
```

[8]:
```
# Value counts for categorical columns
value_counts = df_sampled['FoodCategory'].value_counts()
print("\nValue counts for 'FoodCategory':")
print(value_counts)
```

```
Value counts for 'FoodCategory':
FoodCategory
Meat and meat products (including edible offal)
3734
Vegetables and vegetable products (including fungi)
2800
Fruit and fruit products
2681
Cereals and cereal-based products
2414
Nuts and oilseeds
1840
Fats and oils of animal and vegetable (excluding butter)
876
Starchy roots and tubers
741
Composite food (including frozen products)
662
Other foods
658
```

```
Milk and dairy products
539
Fish and other seafood (including amphibians, reptiles, snails and insects)
516
Legumes and pulses
512
Food for infants and small children
449
Sugar and confectionary (including cocoa products)
366
Alcoholic beverages
364
Herbs, spices and condiments
345
Snacks and desserts
212
Eggs and egg products
145
Fruit and vegetable juices
81
Products for special nutritional use
27
Non-alcoholic beverages (excluding milk, fruit and vegetable juice, water and
stimulants)                    23
Stimulant beverages, dried and diluted excluding cocoa products
13
Drinking water (water without any additives except carbon dioxide; includes
water ice for consumption)       2
Name: count, dtype: int64
```

[9]:
```
# Value counts for categorical columns
value_counts = df_sampled['ContaminantName'].value_counts()
print("\nValue counts for 'ContaminantName':")
print(value_counts)
```

```
Value counts for 'ContaminantName':
ContaminantName
Aflatoxin (total)     1323
Lead                   868
Cadmium                596
Dichlorvos             467
Arsenic (total)        467
                      …
Scopolamine             20
Atropine                15
Ochratoxin A             8
Methyl mercury           5
```

```
Dioxins (WHO TEFs)       1
Name: count, Length: 99, dtype: int64
```

[10]:
```python
# Drop duplicate rows
df_sampled_cleaned = df_sampled.drop_duplicates()

# Display the first few rows of the cleaned dataset
print("\nDataset after dropping duplicates:")
print(df_sampled_cleaned.head())

# Check if there are any duplicates left
duplicate_rows = df_sampled_cleaned.duplicated().sum()
print(f"\nNumber of duplicate rows after cleaning: {duplicate_rows}")
```

```
Dataset after dropping duplicates:
        RecordType RegionCode RegionName ContaminantName  \
353339  Individual       PAHO       PAHO            Lead
16878   Individual       PAHO       PAHO      Cyromazine
117423  Individual       PAHO       PAHO    Dimethomorph
274255  Individual       PAHO       PAHO   Fenpyroximate
15153   Individual       PAHO       PAHO       Abamectin

                                        FoodCategory  \
353339  Meat and meat products (including edible offal)
16878                          Starchy roots and tubers
117423                         Fruit and fruit products
274255                                Nuts and oilseeds
15153                               Snacks and desserts

                                     FoodName FoodCode       LocalFoodName  \
353339  Kidney of cattle, goats, pigs and sheep  MO 0098         rim bovino
16878                                Arrowroot  VR 0573  Arrowroot Products
117423                                 Grapes  FB 0269               Grape
274255                               OILSEEDS  SO 0088        Seed - other
15153                             Snack food     15.1        Chips - Corn

       FoodStateName ResultText  …     LOD      LOQ  Year  \
353339           Raw         ND  …  33.000  100.000  2016
16878        Unknown         ND  …   0.010    0.020  2016
117423           Raw         ND  …   0.001    0.005  2016
274255           Raw         ND  …   0.010    0.011  2016
15153        Unknown         ND  …   0.010    0.020  2016

        RepresentativenessName LabNumber  FoodOriginName  \
353339         Random sampling     102.0        Domestic
16878          Random sampling       1.0         Unknown
117423         Random sampling       6.0        Imported
```

```
274255         Random sampling       3.0        Imported
15153          Random sampling       1.0         Unknown

             AnalyticalQAName ResultBasisName                 PortionTypeName  \
353339  Officially accredited           As is  Total food (edible + inedible)
16878   Officially accredited           As is                      Edible only
117423  Officially accredited           As is                      Edible only
274255  Officially accredited           As is                      Edible only
15153   Officially accredited           As is                      Edible only

       SerialNumber
353339        14331
16878     121624767
117423    287320465
274255   1851285869
15153   -1239224930

[5 rows x 21 columns]

Number of duplicate rows after cleaning: 0
```

[11]:
```
# Drop columns not useful for analysis
df_cleaned = df_sampled_cleaned.drop(columns=['RecordType', 'LabNumber'])

# Impute missing values with "Unknown" for each relevant column
df_cleaned['FoodStateName'] = df_cleaned['FoodStateName'].fillna('Unknown')
df_cleaned['FoodOriginName'] = df_cleaned['FoodOriginName'].fillna('Unknown')
df_cleaned['AnalyticalQAName'] = df_cleaned['AnalyticalQAName'].
  ↪fillna('Unknown')
```

[12]:
```
# Check for missing values in each column of the cleaned dataset
missing_values = df_cleaned.isnull().sum()

# Display the missing values per column
missing_values
```

[12]:
```
RegionCode              0
RegionName              0
ContaminantName         4
FoodCategory            0
FoodName                0
FoodCode                0
LocalFoodName           0
FoodStateName           0
ResultText             80
UnitName                0
LOD                   758
```

```
LOQ                          557
Year                           0
RepresentativenessName         0
FoodOriginName                 0
AnalyticalQAName               0
ResultBasisName                0
PortionTypeName                0
SerialNumber                   0
dtype: int64
```

[13]:
```
# Drop rows with missing values in critical columns ('ContaminantName',␣
  ↪'ResultText', 'LOD', 'LOQ')
dff = df_cleaned.dropna(subset=['ContaminantName', 'ResultText', 'LOD', 'LOQ'])

# Check how much data is left
rows_remaining = dff.shape[0]
print(f"Rows remaining after dropping rows with missing values:␣
  ↪{rows_remaining}")

# Generate a summary of the cleaned dataset
summary_stats = dff.describe()
print("\nSummary statistics of the cleaned dataset:")
print(summary_stats)

# Check for missing values in the remaining data
missing_values_after_cleaning = dff.isnull().sum()
print("\nMissing values after cleaning:")
print(missing_values_after_cleaning)
```

```
Rows remaining after dropping rows with missing values: 18853

Summary statistics of the cleaned dataset:
                LOD           LOQ          Year
count  18853.000000  18853.000000  18853.000000
mean       0.366803      1.509309   2016.598154
std        3.009235      8.658551      1.297585
min        0.000050      0.000120   2016.000000
25%        0.005000      0.010000   2016.000000
50%        0.010000      0.020000   2016.000000
75%        0.021000      0.064000   2017.000000
max      290.000000    200.000000   2024.000000

Missing values after cleaning:
RegionCode                0
RegionName                0
ContaminantName           0
FoodCategory              0
```

```
FoodName                  0
FoodCode                  0
LocalFoodName             0
FoodStateName             0
ResultText                0
UnitName                  0
LOD                       0
LOQ                       0
Year                      0
RepresentativenessName    0
FoodOriginName            0
AnalyticalQAName          0
ResultBasisName           0
PortionTypeName           0
SerialNumber              0
dtype: int64
```

[14]:
```
# Generate summary statistics for categorical columns
categorical_summary = dff.describe(include=['object'])

# Display the summary for categorical data
categorical_summary
```

[14]:
```
       RegionCode RegionName    ContaminantName  \
count       18853      18853              18853
unique          1          1                 99
top          PAHO       PAHO  Aflatoxin (total)
freq        18853      18853               1314

                                        FoodCategory FoodName FoodCode  \
count                                          18853    18853    18853
unique                                            23      316      315
top     Meat and meat products (including edible offal)   Peanut  SO 0697
freq                                            3331     1402     1402

           LocalFoodName FoodStateName ResultText UnitName  \
count              18853         18853      18853    18853
unique              1266             3       1006        3
top     peanuts, shelled       Unknown         ND    mg/kg
freq                1307         10597      14304    13794

       RepresentativenessName FoodOriginName       AnalyticalQAName  \
count                   18853          18853                  18853
unique                      3              4                      4
top           Random sampling       Imported  Officially accredited
freq                    18697           8248                  17493
```

```
       ResultBasisName PortionTypeName  SerialNumber
count            18853           18853         18853
unique               5               2         13683
top              As is     Edible only     642723062
freq             17597           18647             6
```

Data Check, Cleaning, and Validation Summary Initial Data Exploration:

Unique Values Check: We explored the unique values of important columns like 'FoodStateName', 'RecordType', and 'ContaminantName' to understand the data and identify potential issues.

Missing Data Check: We identified and reviewed columns with missing values, including ContaminantName, LOD, LOQ, and ResultText, among others.

Data Cleaning Process:

Dropping Irrelevant Columns: Columns like 'RecordType' and 'LabNumber' were dropped as they were either redundant or not useful for predicting the contaminant in food samples.

Imputation of Missing Values: We imputed missing values in categorical columns (e.g., 'FoodStateName', 'FoodOriginName', 'AnalyticalQAName') with the placeholder "Unknown" to avoid losing valuable data.

Dropping Rows with Critical Missing Data: Rows with missing values in key columns such as 'ContaminantName', 'ResultText', 'LOD', and 'LOQ' were dropped to ensure accurate analysis.

The cleaned dataset was then renamed dff.

Data Validation Confirmation:

After the cleaning process, the missing data was rechecked, confirming that the critical columns no longer had missing values.

The categorical columns were summarized, providing insights into their distributions and confirming the integrity of the cleaned dataset.

Outcome:

The dff dataset is now ready for further analysis, with missing values handled, irrelevant columns removed, and the data validated for accuracy.

The cleaned data will now serve as the foundation for exploratory data analysis (EDA) and predictive modeling, focusing on predicting chemical contaminants in food samples.

[15]:
```
dff.head()
```

[15]:
```
        RegionCode RegionName ContaminantName  \
353339       PAHO       PAHO            Lead
16878        PAHO       PAHO      Cyromazine
117423       PAHO       PAHO    Dimethomorph
274255       PAHO       PAHO   Fenpyroximate
15153        PAHO       PAHO       Abamectin

                                    FoodCategory  \
```

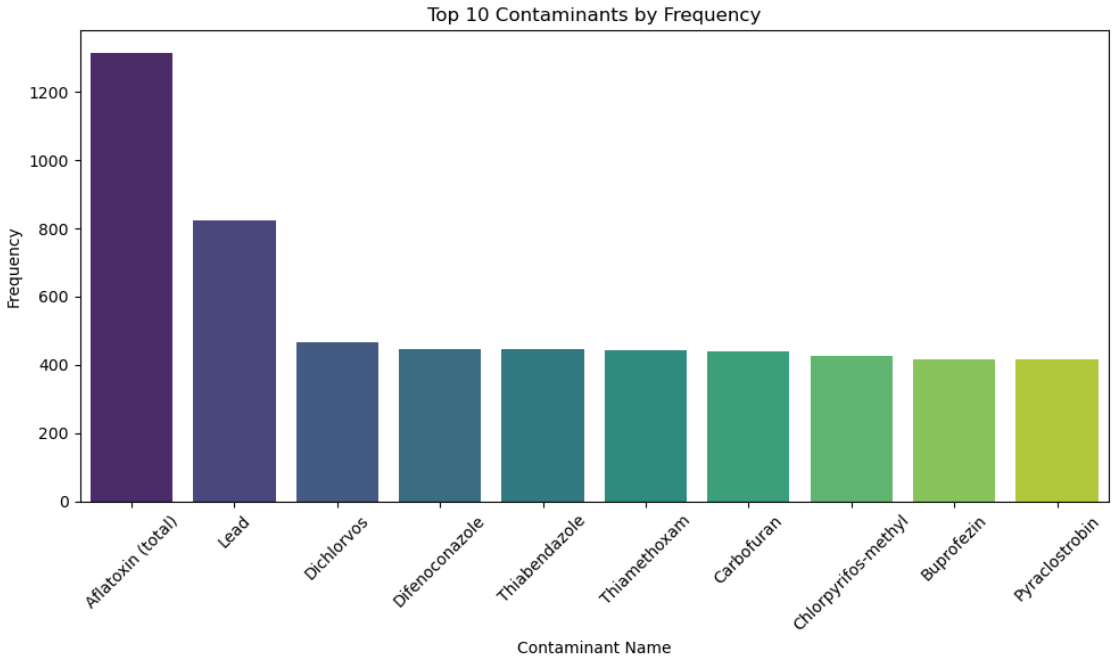
```
353339  Meat and meat products (including edible offal)
16878                          Starchy roots and tubers
117423                         Fruit and fruit products
274255                                Nuts and oilseeds
15153                              Snacks and desserts

                                      FoodName FoodCode       LocalFoodName  \
353339  Kidney of cattle, goats, pigs and sheep  MO 0098          rim bovino
16878                                Arrowroot  VR 0573  Arrowroot Products
117423                                 Grapes  FB 0269               Grape
274255                                OILSEEDS  SO 0088        Seed - other
15153                              Snack food     15.1        Chips - Corn

       FoodStateName ResultText UnitName     LOD      LOQ  Year  \
353339           Raw         ND    ug/kg  33.000  100.000  2016
16878        Unknown         ND    mg/kg   0.010    0.020  2016
117423           Raw         ND    mg/kg   0.001    0.005  2016
274255           Raw         ND    mg/kg   0.010    0.011  2016
15153        Unknown         ND    mg/kg   0.010    0.020  2016

       RepresentativenessName FoodOriginName       AnalyticalQAName  \
353339        Random sampling       Domestic  Officially accredited
16878         Random sampling        Unknown  Officially accredited
117423        Random sampling       Imported  Officially accredited
274255        Random sampling       Imported  Officially accredited
15153         Random sampling        Unknown  Officially accredited

       ResultBasisName                 PortionTypeName SerialNumber
353339           As is  Total food (edible + inedible)        14331
16878            As is                     Edible only    121624767
117423           As is                     Edible only    287320465
274255           As is                     Edible only   1851285869
15153            As is                     Edible only  -1239224930
```

[16]:
```
# Get the top 10 contaminants by frequency
top_contaminants = dff['ContaminantName'].value_counts().head(10)

# Display the top contaminants
print(top_contaminants)
```

```
ContaminantName
Aflatoxin (total)      1314
Lead                    822
Dichlorvos              467
Difenoconazole          447
Thiabendazole           445
Thiamethoxam            443
```

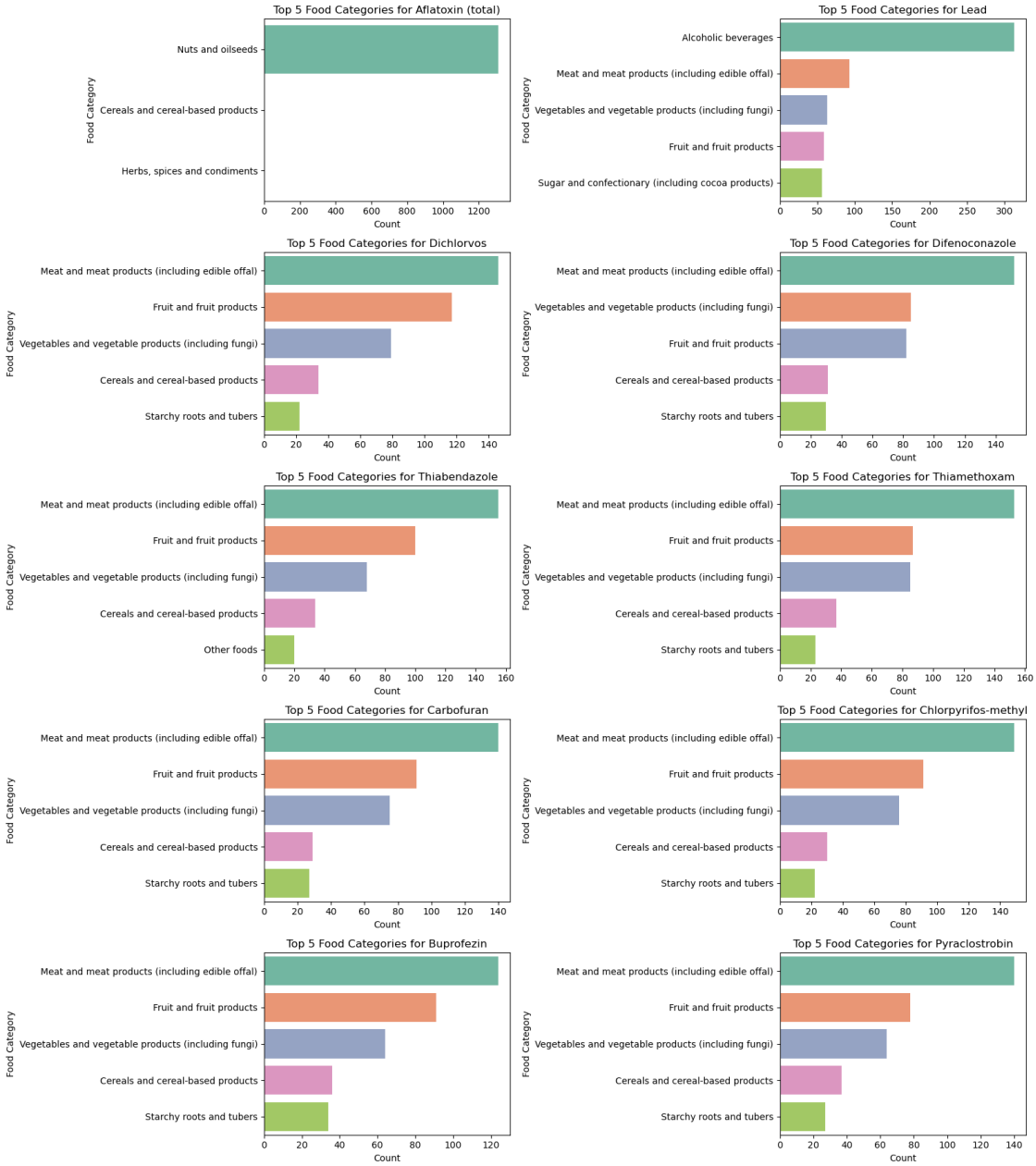
```
Carbofuran              441
Chlorpyrifos-methyl     427
Buprofezin              417
Pyraclostrobin          415
Name: count, dtype: int64
```

[19]:
```
import matplotlib.pyplot as plt
import seaborn as sns
# Plot the top 10 contaminants
plt.figure(figsize=(10, 6))
sns.barplot(x=top_contaminants.index, y=top_contaminants.values,␣
 ↪palette='viridis')
plt.title('Top 10 Contaminants by Frequency')
plt.xlabel('Contaminant Name')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



[23]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Identify the top 10 contaminants based on frequency
top_contaminants = dff['ContaminantName'].value_counts().head(10).index

# Filter dff to include only the top 10 contaminants
```

```
filtered_dff = dff[dff['ContaminantName'].isin(top_contaminants)]

# Create a figure with multiple subplots (5 rows, 2 columns for 10 contaminants)
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(16, 18))
axes = axes.flatten()  # Flatten the 2D array of axes to 1D for easy iteration

# Loop through the top contaminants and plot each as a separate bar chart
for i, contaminant in enumerate(top_contaminants):
    # Filter the data for the current contaminant
    contaminant_data = filtered_dff[filtered_dff['ContaminantName'] ==␣
 ↪contaminant]

    # Get the top 5 food categories for the current contaminant
    contaminant_food_category = contaminant_data['FoodCategory'].value_counts()
    top_5_food_categories = contaminant_food_category.head(5)

    # Plot the bar chart for the current contaminant (food categories on y-axis)
    sns.barplot(y=top_5_food_categories.index, x=top_5_food_categories.values,␣
 ↪ax=axes[i], palette='Set2')

    # Set plot title and labels
    axes[i].set_title(f'Top 5 Food Categories for {contaminant}')
    axes[i].set_xlabel('Count')
    axes[i].set_ylabel('Food Category')

    # Rotate the y-axis labels for better readability
    axes[i].tick_params(axis='y', rotation=0)

# Adjust layout to avoid overlap
plt.tight_layout()
plt.show()
```

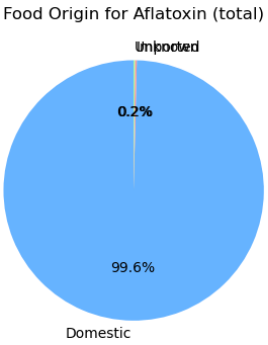


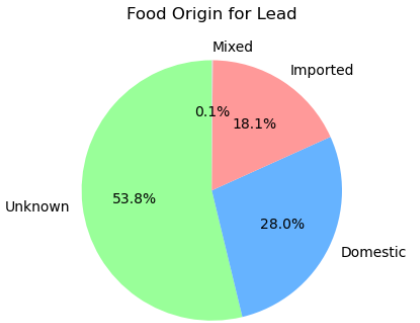
[57]:
```
import plotly.express as px

# Identify the top 10 contaminants
top_contaminants_list = dff['ContaminantName'].value_counts().head(10).index

# Filter dataset to only include top 10 contaminants
filtered_data_top = dff[dff['ContaminantName'].isin(top_contaminants_list)]

# Group the data for sunburst: Contaminant -> Food Origin
```

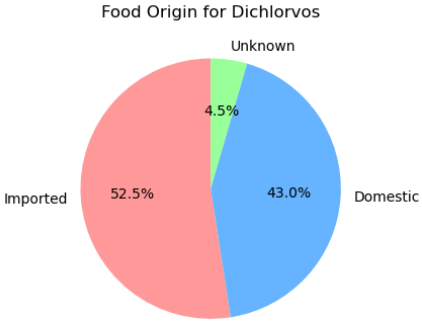
```
sunburst_data = filtered_data_top.groupby(['ContaminantName',␣
 ↪'FoodOriginName']).size().reset_index(name='Count')

# Create sunburst chart
fig = px.sunburst(
    sunburst_data,
    path=['ContaminantName', 'FoodOriginName'],
    values='Count',
    color='FoodOriginName',
    color_discrete_map={
        'Domestic': '#66b3ff',
        'Imported': '#ff9999',
        'Unknown': '#99ff99',
        'Other': '#ffcc99'
    },
    title='Food Origin Distribution for Top 10 Contaminants'
)

fig.update_layout(margin=dict(t=40, l=0, r=0, b=0))
fig.show()
```

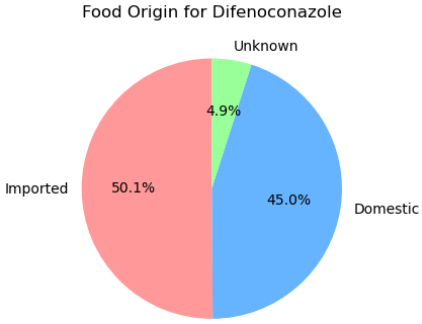
[56]:
```
# Define a fixed color map for each food origin
color_map = {
    'Domestic': '#66b3ff',   # Blue
    'Imported': '#ff9999',   # Red
    'Unknown': '#99ff99',    # Green
    'Other': '#ffcc99'       # Orange
}

# Create a figure with subplots for each top contaminant
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(16, 18))
axes = axes.flatten()

# Loop through the top contaminants and plot pie charts
for i, contaminant in enumerate(top_contaminants_list):
    # Filter the data for the specific contaminant
    contaminant_data = filtered_data_top[filtered_data_top['ContaminantName']␣
 ↪== contaminant]

    # Get the count of food origin (Domestic, Imported, Unknown, etc.)
    food_origin_counts = contaminant_data['FoodOriginName'].value_counts()

    # Match colors to labels based on the color_map
    origin_labels = food_origin_counts.index
    origin_colors = [color_map.get(label, '#d3d3d3') for label in␣
 ↪origin_labels]  # Fallback to grey if unknown
```

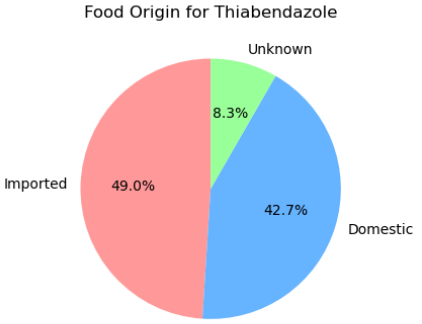
```
    # Plot pie chart
    axes[i].pie(food_origin_counts, labels=origin_labels, autopct='%1.1f%%',
                colors=origin_colors, startangle=90)
    axes[i].set_title(f'Food Origin for {contaminant}')

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

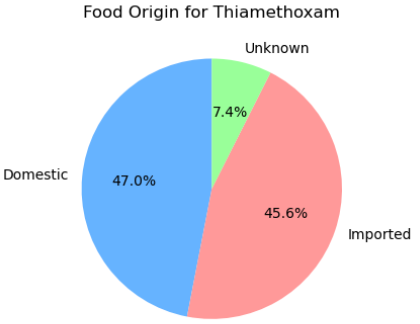

Food Origin for Aflatoxin (total)
Unknown
Imported
0.2%
99.6%
Domestic

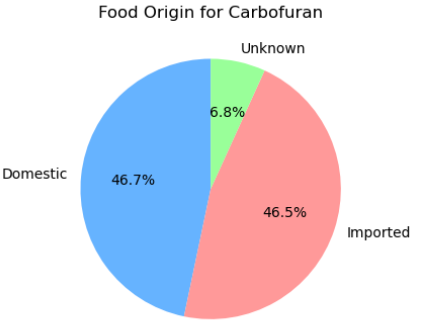

Food Origin for Lead
Mixed
Imported
0.1%
18.1%
Unknown
53.8%
28.0%
Domestic

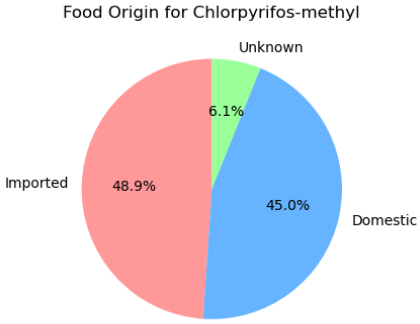

Food Origin for Dichlorvos
Unknown
4.5%
Imported
52.5%
43.0%
Domestic

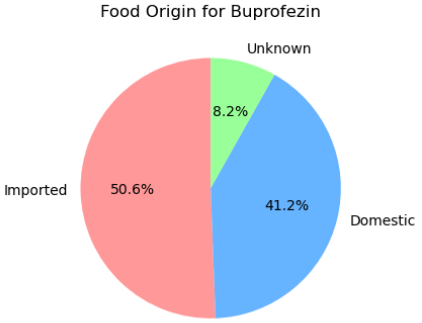

Food Origin for Difenoconazole
Unknown
4.9%
Imported
50.1%
45.0%
Domestic

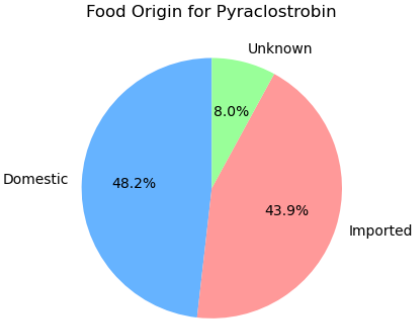

Food Origin for Thiabendazole
Unknown
8.3%
Imported
49.0%
42.7%
Domestic


Food Origin for Thiamethoxam
Unknown
7.4%
Domestic
47.0%
45.6%
Imported

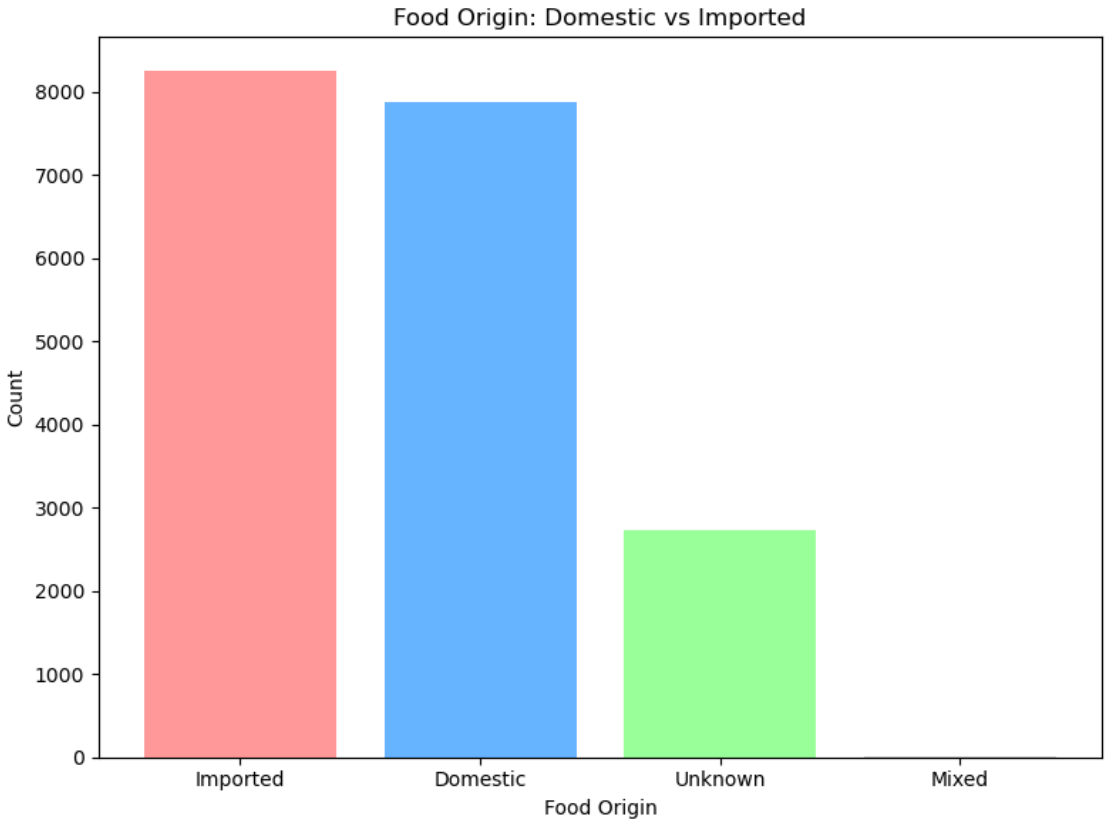

Food Origin for Carbofuran
Unknown
6.8%
Domestic
46.7%
46.5%
Imported


Food Origin for Chlorpyrifos-methyl
Unknown
6.1%
Imported
48.9%
45.0%
Domestic

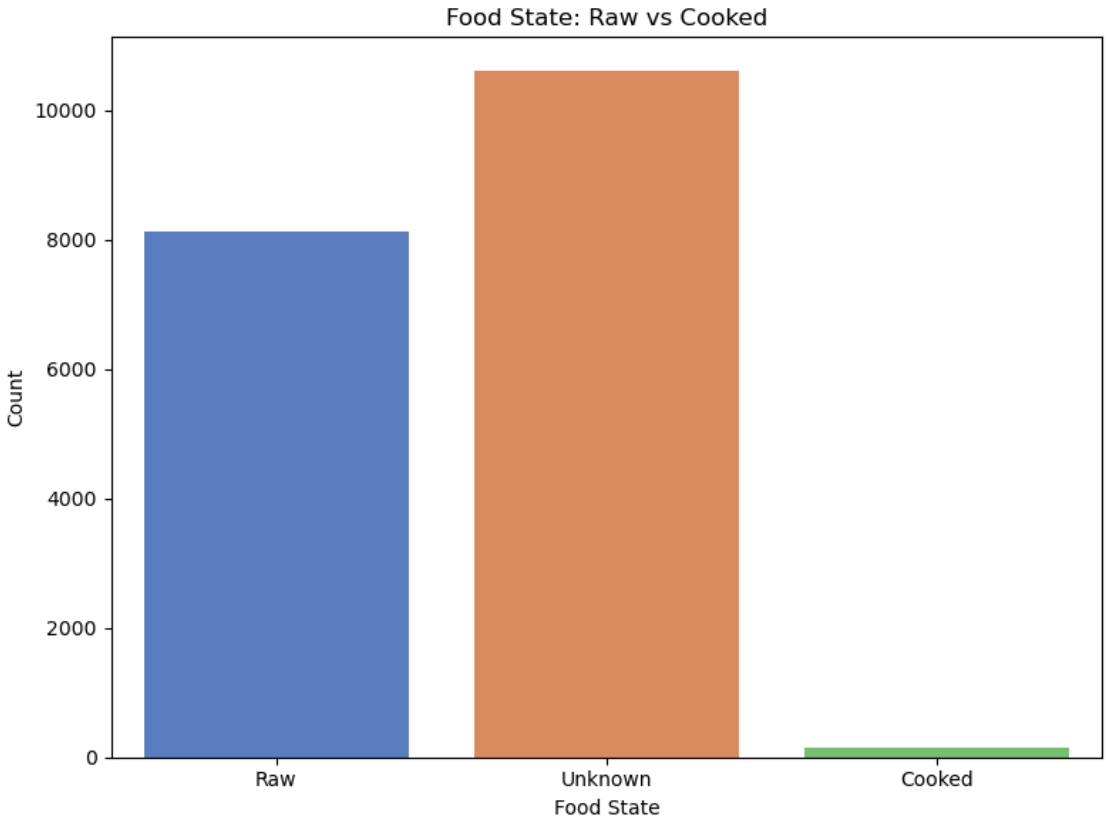

Food Origin for Buprofezin
Unknown
8.2%
Imported
50.6%
41.2%
Domestic


Food Origin for Pyraclostrobin
Unknown
8.0%
Domestic
48.2%
43.9%
Imported

[60]:
```
import matplotlib.pyplot as plt

# Count values for each food origin
origin_counts = dff['FoodOriginName'].value_counts()

# Define consistent color mapping
color_map = {
    'Domestic': '#66b3ff',
    'Imported': '#ff9999',
    'Unknown': '#99ff99',
    'Other': '#ffcc99'
}

# Extract labels and corresponding colors
labels = origin_counts.index
colors = [color_map.get(label, '#d3d3d3') for label in labels]  # fallback␣
 ↪color if label not in map

# Create bar plot using matplotlib
plt.figure(figsize=(8, 6))
plt.bar(labels, origin_counts.values, color=colors)
plt.title('Food Origin: Domestic vs Imported')
plt.xlabel('Food Origin')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



[28]:
```
# 5. Food State (Raw vs Cooked)
plt.figure(figsize=(8, 6))
sns.countplot(data=dff, x='FoodStateName', palette='muted')
plt.title('Food State: Raw vs Cooked')
plt.xlabel('Food State')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



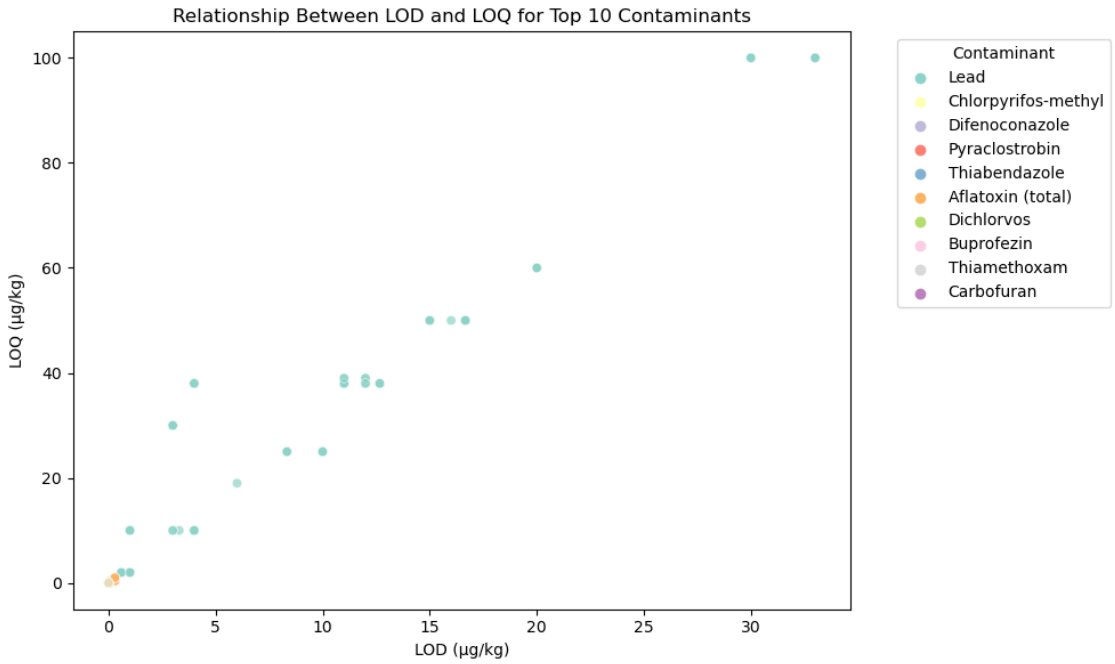
[31]:
```
from prettytable import PrettyTable

# Identify the top 10 contaminants based on frequency
top_contaminants_list = dff['ContaminantName'].value_counts().head(10).index

# Filter the data to include only the top 10 contaminants
filtered_data_top_10 = dff[dff['ContaminantName'].isin(top_contaminants_list)]

# Find the top 10 food categories with the highest LOD levels
top_food_categories_lod = filtered_data_top_10.groupby(['FoodCategory',␣
 ↪'ContaminantName'])['LOD'].max().reset_index()

# Sort by LOD values in descending order and select the top 10
top_food_categories_lod = top_food_categories_lod.sort_values(by='LOD',␣
 ↪ascending=False).head(10)

# Create a PrettyTable instance
table = PrettyTable()

# Set column names
table.field_names = ["Food Category", "Contaminant Name", "LOD (µg/kg)"]
```

```
# Add rows without color (plain output)
for _, row in top_food_categories_lod.iterrows():
    table.add_row([row['FoodCategory'], row['ContaminantName'], row['LOD']])

# Print the table without color
print(table)
```

```
+-------------------------------------------------------------------------------+-
----------------+-------------+
|                                Food Category                                  |
Contaminant Name | LOD (µg/kg) |
+-------------------------------------------------------------------------------+-
----------------+-------------+
|               Meat and meat products (including edible offal)                 |
Lead             |     33.0    |
|       Stimulant beverages, dried and diluted excluding cocoa products         |
Lead             |     30.0    |
|              Sugar and confectionary (including cocoa products)               |
Lead             |     20.0    |
|                         Herbs, spices and condiments                          |
Lead             |     16.0    |
| Fish and other seafood (including amphibians, reptiles, snails and insects) |
Lead             |     15.0    |
|                  Composite food (including frozen products)                   |
Lead             |     4.0     |
|           Fats and oils of animal and vegetable (excluding butter)           |
Lead             |     4.0     |
|                            Starchy roots and tubers                           |
Lead             |     4.0     |
|                            Fruit and fruit products                           |
Lead             |     4.0     |
|                       Cereals and cereal-based products                       |
Lead             |     4.0     |
+-------------------------------------------------------------------------------+-
----------------+-------------+
```

[33]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a scatter plot to visualize the relationship between LOD and LOQ
plt.figure(figsize=(10, 6))
sns.scatterplot(data=filtered_data_top_10, x='LOD', y='LOQ',␣
 ↪hue='ContaminantName', palette='Set3', alpha=0.7)

# Customize the plot
plt.title('Relationship Between LOD and LOQ for Top 10 Contaminants')
```

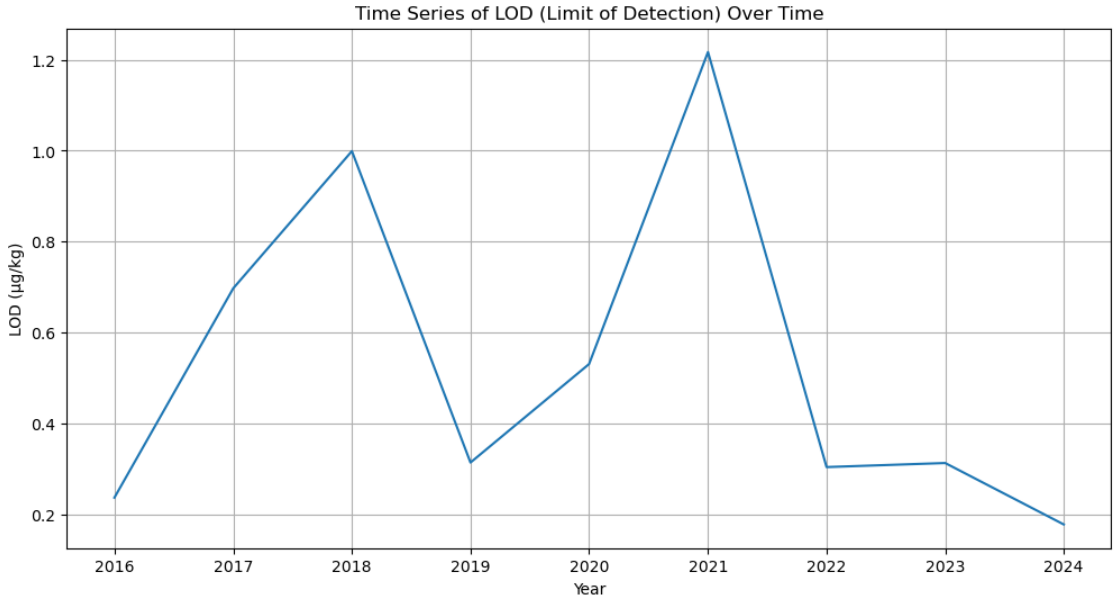
```
plt.xlabel('LOD (µg/kg)')
plt.ylabel('LOQ (µg/kg)')
plt.legend(title='Contaminant', bbox_to_anchor=(1.05, 1), loc='upper left')

# Adjust the layout to make room for the legend
plt.tight_layout()

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



[41]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Convert the 'Year' column to datetime if it's not already
dff['Year'] = pd.to_datetime(dff['Year'], format='%Y')

# Set 'Year' as the index
dff.set_index('Year', inplace=True)

# Choose 'LOD' as the target variable (you can replace with 'LOQ' or another␣
 ↪column if needed)
time_series_data = dff.groupby(dff.index)['LOD'].mean()

# Plot the time series data to check the trend
plt.figure(figsize=(12, 6))
```

```
plt.plot(time_series_data)
plt.title('Time Series of LOD (Limit of Detection) Over Time')
plt.xlabel('Year')
plt.ylabel('LOD (µg/kg)')
plt.grid(True)
plt.show()
```

```
C:\Users\olape\AppData\Local\Temp\ipykernel_24524\4200224243.py:5:
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
  dff['Year'] = pd.to_datetime(dff['Year'], format='%Y')
```



[42]:
```
from statsmodels.tsa.stattools import adfuller

# Perform Augmented Dickey-Fuller (ADF) test to check for stationarity
result = adfuller(time_series_data)

print('ADF Statistic:', result[0])
print('p-value:', result[1])

# If p-value is greater than 0.05, the time series is non-stationary, and we␣
  ↪need to difference it
```

```
ADF Statistic: -2.7120541140611443
```

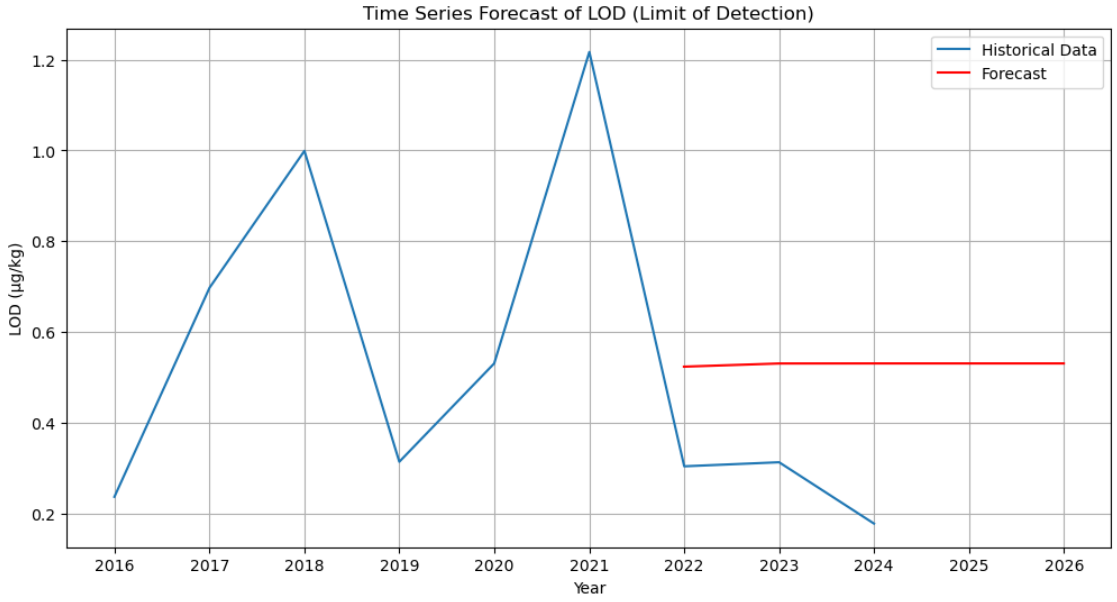
p-value: 0.07197550729208393

[44]:
```
from statsmodels.tsa.arima.model import ARIMA

# Fit ARIMA model (start with p=1, d=1, q=1 for simplicity)
arima_model = ARIMA(time_series_data, order=(1, 1, 1))
fitted_model = arima_model.fit()

# Print model summary
print(fitted_model.summary())

# Forecast the next 5 years
forecast = fitted_model.forecast(steps=5)
forecast_index = pd.date_range(start='2021', periods=5, freq='A')  # Annual␣
 ↪forecast starting from 2021

# Plot the forecasted values
plt.figure(figsize=(12, 6))
plt.plot(time_series_data, label='Historical Data')
plt.plot(forecast_index, forecast, label='Forecast', color='red')
plt.title('Time Series Forecast of LOD (Limit of Detection)')
plt.xlabel('Year')
plt.ylabel('LOD (µg/kg)')
plt.legend()
plt.grid(True)
plt.show()
```

```
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Non-invertible
starting MA parameters found. Using zeros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    LOD   No. Observations:                    9
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -4.418
```

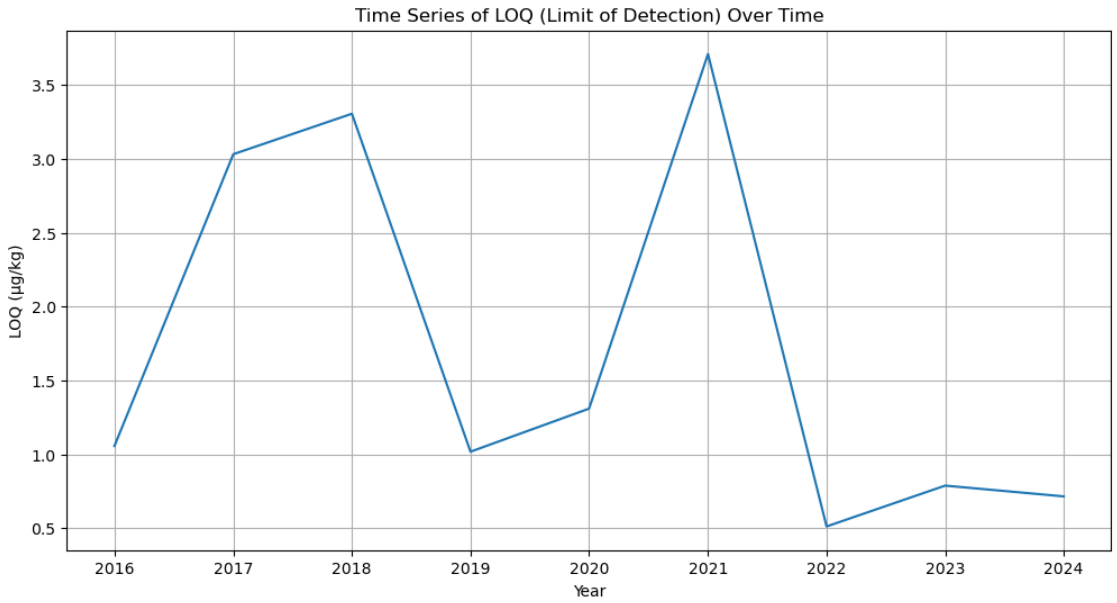
```
Date:                Thu, 01 May 2025   AIC                            14.836
Time:                        20:14:00   BIC                            15.075
Sample:                  01-01-2016   HQIC                           13.229
                       - 01-01-2024
Covariance Type:                 opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0202      0.861      0.023      0.981      -1.668       1.708
ma.L1         -0.9968     95.744     -0.010      0.992    -188.652     186.658
sigma2         0.1353     12.803      0.011      0.992     -24.958      25.229
===================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
0.99
Prob(Q):                              0.99   Prob(JB):
0.61
Heteroskedasticity (H):               0.87   Skew:
0.44
Prob(H) (two-sided):                  0.91   Kurtosis:
1.52
===================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```



[45]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Compute MAE and MSE
mae = mean_absolute_error(time_series_data[-5:], forecast)
mse = mean_squared_error(time_series_data[-5:], forecast)

print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
```

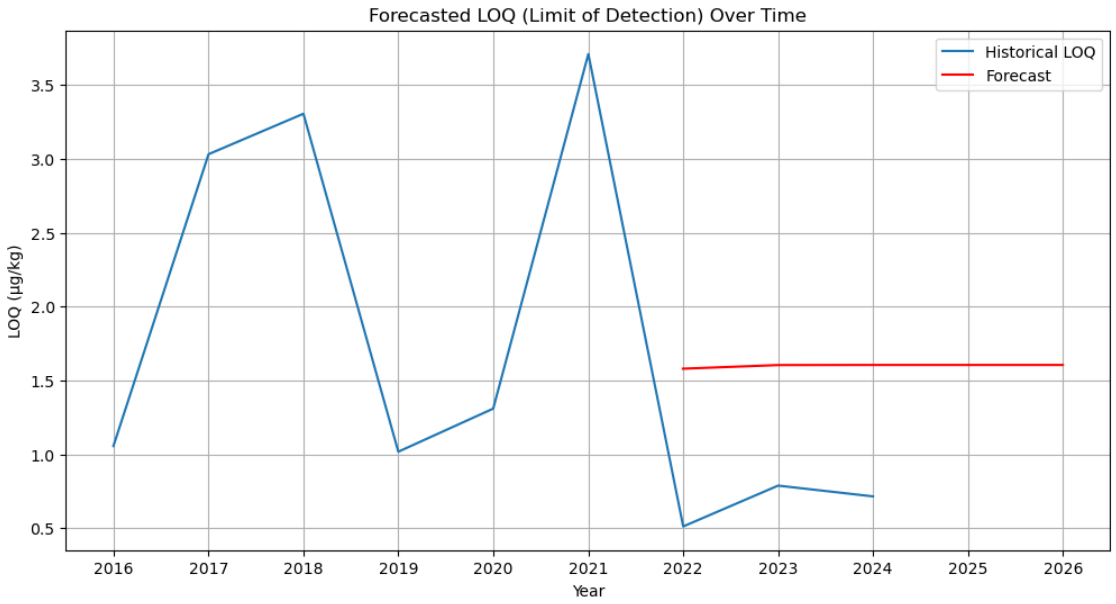
```
Mean Absolute Error: 0.29811633158005496
Mean Squared Error: 0.1388655780329246
```

[49]:
```
import pandas as pd
import matplotlib.pyplot as plt


# Choose 'LOQ' as the target variable (you can replace with 'LOQ' or another␣
 ↪column if needed)
time_series_data1 = dff.groupby(dff.index)['LOQ'].mean()

# Plot the time series data to check the trend
plt.figure(figsize=(12, 6))
plt.plot(time_series_data1)
plt.title('Time Series of LOQ (Limit of Detection) Over Time')
plt.xlabel('Year')
plt.ylabel('LOQ (µg/kg)')
plt.grid(True)
plt.show()
```



[50]:
```
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

# Create a time series of LOQ (Limit of Detection) over time
time_series_loq = dff.groupby(dff.index)['LOQ'].mean()  # Mean LOQ per year

# Fit ARIMA model (with default parameters)
arima_model = ARIMA(time_series_loq, order=(1, 1, 1))
fitted_model = arima_model.fit()

# Forecast the next 5 years
forecast = fitted_model.forecast(steps=5)

# Plot the historical data and forecast
plt.figure(figsize=(12, 6))
plt.plot(time_series_loq, label='Historical LOQ')
plt.plot(pd.date_range(start='2021', periods=5, freq='A'), forecast,␣
 ↪label='Forecast', color='red')
plt.title('Forecasted LOQ (Limit of Detection) Over Time')
plt.xlabel('Year')
plt.ylabel('LOQ (µg/kg)')
plt.legend()
plt.grid(True)
plt.show()
```

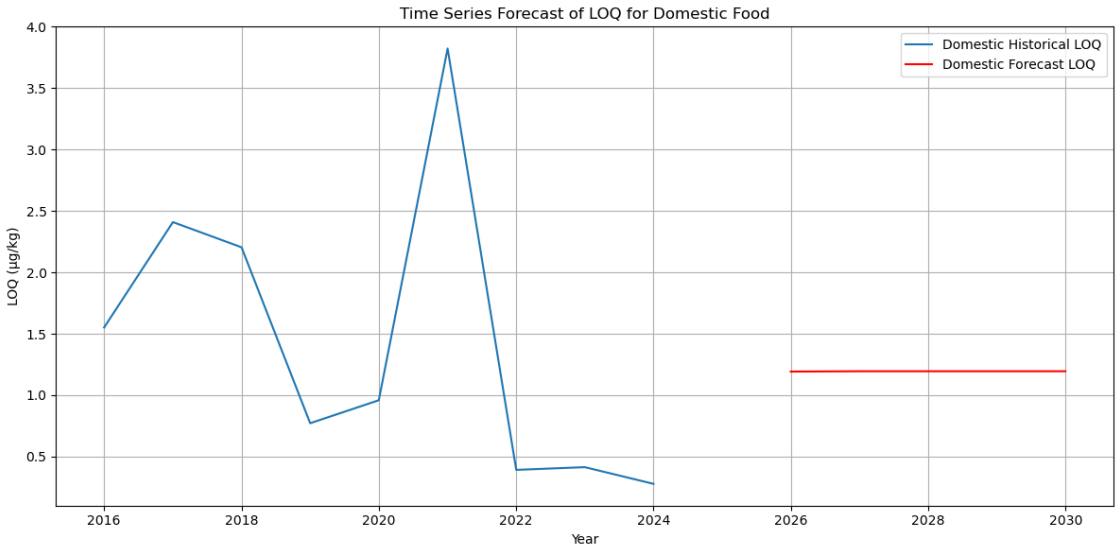
```
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency AS-JAN will be used.
  self._init_dates(dates, freq)
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Non-invertible
starting MA parameters found. Using zeros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
```



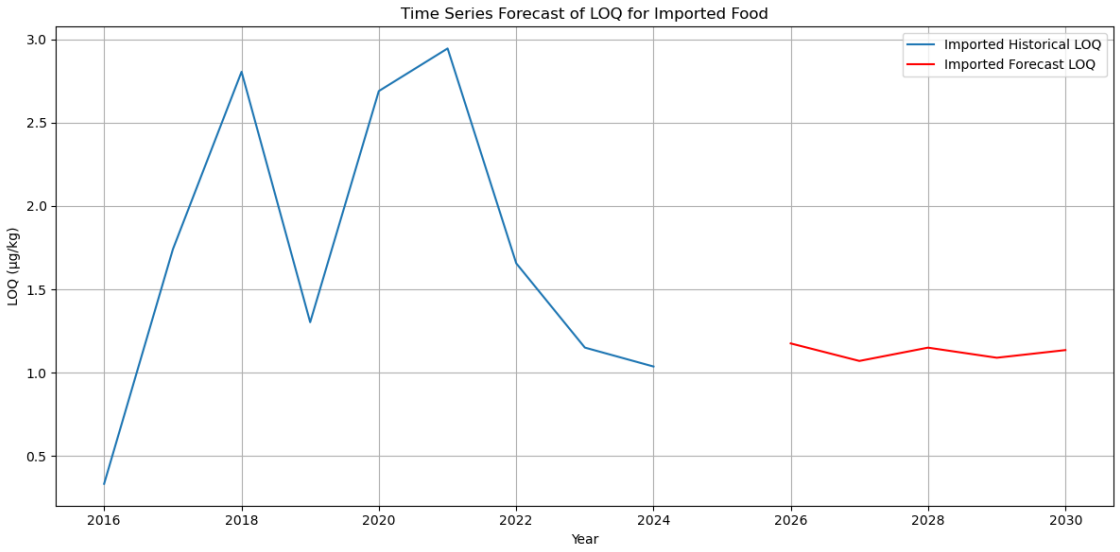
[62]:
```
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
import pandas as pd

# If 'Year' is already the index and datetime, skip conversion
if not isinstance(dff.index, pd.DatetimeIndex):
    dff['Year'] = pd.to_datetime(dff['Year'], format='%Y')
    dff.set_index('Year', inplace=True)

# Group by year and food origin, compute mean LOQ
grouped_loq = dff.groupby([dff.index, 'FoodOriginName'])['LOQ'].mean().unstack()

# Forecast for each origin: Domestic and Imported
for origin in ['Domestic', 'Imported']:
    if origin not in grouped_loq:
        continue

    ts = grouped_loq[origin].dropna()

    model = ARIMA(ts, order=(1, 1, 1))
    fitted = model.fit()

    forecast = fitted.forecast(steps=5)
    forecast_index = pd.date_range(start=ts.index[-1] + pd.DateOffset(years=1),␣
 ↪periods=5, freq='A')

    plt.figure(figsize=(12, 6))
```

```
    plt.plot(ts, label=f'{origin} Historical LOQ')
    plt.plot(forecast_index, forecast, label=f'{origin} Forecast LOQ',␣
↪color='red')
    plt.title(f'Time Series Forecast of LOQ for {origin} Food')
    plt.xlabel('Year')
    plt.ylabel('LOQ (µg/kg)')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

```
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning:

Non-invertible starting MA parameters found. Using zeros as starting parameters.
```




```
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.
```



[63]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Split historical data: use all but last 2 years for training, last 2 for␣
 ↪testing
for origin in ['Domestic', 'Imported']:
    if origin not in grouped_loq:
        continue

    ts = grouped_loq[origin].dropna()

    if len(ts) < 7:
        print(f"Not enough data to evaluate {origin}")
        continue

    train = ts.iloc[:-2]
    test = ts.iloc[-2:]

    model = ARIMA(train, order=(1, 1, 1))
    fitted = model.fit()

    forecast = fitted.forecast(steps=2)

    mae = mean_absolute_error(test, forecast)
    mse = mean_squared_error(test, forecast)
    rmse = np.sqrt(mse)

    print(f"\nAccuracy Metrics for {origin} LOQ Forecast:")
```

```
    print(f"Mean Absolute Error (MAE): {mae:.4f}")
    print(f"Mean Squared Error (MSE): {mse:.4f}")
    print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
```

```
Accuracy Metrics for Domestic LOQ Forecast:
Mean Absolute Error (MAE): 1.6308
Mean Squared Error (MSE): 2.8040
Root Mean Squared Error (RMSE): 1.6745

Accuracy Metrics for Imported LOQ Forecast:
Mean Absolute Error (MAE): 1.0313
Mean Squared Error (MSE): 1.0659
Root Mean Squared Error (RMSE): 1.0324
C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning:

Non-invertible starting MA parameters found. Using zeros as starting parameters.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
```

```
used.

C:\Users\olape\anaconda3\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency AS-JAN will be
used.
```

1. One-Sample t-Test: Test if the average LOD (Limit of Detection) for a specific Contaminant (Aflatoxin) is different from a known threshold (1 µg/kg).

Null Hypothesis (H): The average LOD for Aflatoxin is equal to 1 µg/kg.

Alternative Hypothesis (H): The average LOD for Aflatoxin is not equal to 1 µg/kg.

[51]:
```
from scipy import stats

# Filter the data for a specific contaminant, e.g., 'Aflatoxin (total)'
aflatoxin_data = dff[dff['ContaminantName'] == 'Aflatoxin (total)']['LOD']

# Perform a one-sample t-test to check if the mean LOD is different from 1 µg/kg
t_statistic, p_value = stats.ttest_1samp(aflatoxin_data, 1)

print(f"t-statistic: {t_statistic}")
print(f"p-value: {p_value}")

# Decide based on p-value (alpha = 0.05)
if p_value < 0.05:
    print("Reject the null hypothesis: The mean LOD is significantly different␣
 ↪from 1 µg/kg.")
else:
    print("Fail to reject the null hypothesis: The mean LOD is not␣
 ↪significantly different from 1 µg/kg.")
```

```
t-statistic: -1049.7370733933697
p-value: 0.0
Reject the null hypothesis: The mean LOD is significantly different from 1
µg/kg.
```

2. Two-Sample t-Test: Test if the LOD for Aflatoxin in Imported food products is significantly different from that in Domestic food products.

Null Hypothesis (H): The average LOD for Aflatoxin in Imported food is equal to the average LOD in Domestic food.

Alternative Hypothesis (H): The average LOD for Aflatoxin in Imported food is different from the average LOD in Domestic food.

[52]:
```
# Filter the data for Aflatoxin and Food Origin (Imported vs Domestic)
aflatoxin_imported = dff[(dff['ContaminantName'] == 'Aflatoxin (total)') &␣
 ↪(dff['FoodOriginName'] == 'Imported')]['LOD']
aflatoxin_domestic = dff[(dff['ContaminantName'] == 'Aflatoxin (total)') &␣
 ↪(dff['FoodOriginName'] == 'Domestic')]['LOD']

# Perform a two-sample t-test
t_statistic, p_value = stats.ttest_ind(aflatoxin_imported, aflatoxin_domestic)

print(f"t-statistic: {t_statistic}")
print(f"p-value: {p_value}")

# Decide based on p-value (alpha = 0.05)
if p_value < 0.05:
    print("Reject the null hypothesis: The LOD for Aflatoxin is significantly␣
 ↪different between Imported and Domestic food.")
else:
    print("Fail to reject the null hypothesis: The LOD for Aflatoxin is not␣
 ↪significantly different between Imported and Domestic food.")
```

```
t-statistic: -53.10606793615387
p-value: 0.0
Reject the null hypothesis: The LOD for Aflatoxin is significantly different
between Imported and Domestic food.
```

```
C:\Users\olape\anaconda3\Lib\site-packages\scipy\stats\_axis_nan_policy.py:523:
RuntimeWarning: Precision loss occurred in moment calculation due to
catastrophic cancellation. This occurs when the data are nearly identical.
Results may be unreliable.
  res = hypotest_fun_out(*samples, **kwds)
```

3. Chi-Square Test: Test if there is a relationship between Food Category and Contaminant Name.

Null Hypothesis (H): There is no association between FoodCategory and ContaminantName.

Alternative Hypothesis (H): There is an association between FoodCategory and ContaminantName.

[53]:
```
# Create a contingency table for FoodCategory and ContaminantName
contingency_table = pd.crosstab(dff['FoodCategory'], dff['ContaminantName'])

# Perform the Chi-Square test of independence
chi2_stat, p_value, dof, expected = stats.chi2_contingency(contingency_table)

print(f"Chi2 Statistic: {chi2_stat}")
print(f"p-value: {p_value}")
print(f"Degrees of Freedom: {dof}")
```

```
# Decide based on p-value (alpha = 0.05)
if p_value < 0.05:
    print("Reject the null hypothesis: There is a significant association␣
 ↪between FoodCategory and ContaminantName.")
else:
    print("Fail to reject the null hypothesis: There is no significant␣
 ↪association between FoodCategory and ContaminantName.")
```

```
Chi2 Statistic: 51155.97019585583
p-value: 0.0
Degrees of Freedom: 2156
Reject the null hypothesis: There is a significant association between
FoodCategory and ContaminantName.
```

4. ANOVA (Analysis of Variance): Test if the LOD values for different FoodCategories are significantly different.

Null Hypothesis (H): The mean LOD values for all FoodCategories are equal.

Alternative Hypothesis (H): At least one FoodCategory has a significantly different mean LOD.

[54]:
```
# Group the data by FoodCategory and perform ANOVA on LOD values
food_category_groups = [group['LOD'].values for name, group in dff.
 ↪groupby('FoodCategory')]

# Perform ANOVA
f_statistic, p_value = stats.f_oneway(*food_category_groups)

print(f"F-statistic: {f_statistic}")
print(f"p-value: {p_value}")

# Decide based on p-value (alpha = 0.05)
if p_value < 0.05:
    print("Reject the null hypothesis: There is a significant difference in LOD␣
 ↪values across FoodCategories.")
else:
    print("Fail to reject the null hypothesis: There is no significant␣
 ↪difference in LOD values across FoodCategories.")
```

```
F-statistic: 20.72265595859892
p-value: 1.3151833232988207e-81
Reject the null hypothesis: There is a significant difference in LOD values
across FoodCategories.
```