Appendices

Python Codes

Step 1: Connecting to an API, Pulling in the live animals dataset, and inspect

In [48]:

```

import subprocess
import os
import zipfile
import pandas as pd
from zipfile import ZipFile
import warnings
warnings.filterwarnings('ignore')
```

In [49]:

```
# Execute the Kaggle API command to download the live animals dataset contsining ch
command = "kaggle datasets download -d unitednations/global-food-agriculture-statis
subprocess.run(command.split())
```

Out[49]:

```
CompletedProcess(args=['kaggle', 'datasets', 'download', '-d', 'unitednations/glob
al-food-agriculture-statistics'], returncode=0)
```

In [50]:

```
# Step 2: Check if the download was successful
if os.path.exists("global-food-agriculture-statistics.zip"):
    print("Dataset downloaded successfully!")
```

```
Dataset downloaded successfully!
```

In [52]:

```
# Step 3: Unzip the downloaded file
with zipfile.ZipFile("global-food-agriculture-statistics.zip", "r") as zip_ref:
    zip_ref.extractall("data")
```

In [53]:

```
# Step 4: Optionally, list the contents of the extracted directory
extracted_files = os.listdir("data")
print("Extracted files:", extracted_files)
```

```
Extracted files: ['current_FAO', 'fao_data_crops_data.csv', 'fao_data_fertilizers_da
ta.csv', 'fao_data_forest_data.csv', 'fao_data_land_data.csv', 'fao_data_production_
indices_data.csv']
```

In [ ]:

```
    # Step 5: Download a specific table to work with
    # Specify the CSV file to read from the ZIP archive
    csv_file_to_read = "current_FAO/raw_files/Trade_LiveAnimals_E_All_Data_(Normali

    # Read the ZIP archive
    with ZipFile("global-food-agriculture-statistics.zip", 'r') as zip_file:
        # List the files within the ZIP archive (to double-check paths)
```

```python
        print(zip_file.namelist())

        # Read the CSV file from the ZIP archive with the specified encoding and de
        with zip_file.open(csv_file_to_read) as csv_file:
            df = pd.read_csv(csv_file, encoding='ISO-8859-1')
```

In [55]:
```python
# Print the first few rows of the dataset
df.head()
```

Out[55]:

	Area Code	Area	Item Code	Item	Element Code	Element	Year Code	Year	Unit	Value	Flag
0	2	Afghanistan	866	Cattle	5608	Import Quantity	1961	1961	Head	NaN	M
1	2	Afghanistan	866	Cattle	5608	Import Quantity	1962	1962	Head	NaN	M
2	2	Afghanistan	866	Cattle	5608	Import Quantity	1963	1963	Head	NaN	M
3	2	Afghanistan	866	Cattle	5608	Import Quantity	1964	1964	Head	NaN	M
4	2	Afghanistan	866	Cattle	5608	Import Quantity	1965	1965	Head	NaN	M

In [56]:
```python
# Print the last few rows of the dataset
df.tail()
```

Out[56]:

	Area Code	Area	Item Code	Item	Element Code	Element	Year Code	Year	Unit	Value
662953	5817	Net Food Importing Developing Countries	1922	Sheep and Goats	5922	Export Value	2009	2009	1000 US$	456293.0
662954	5817	Net Food Importing Developing Countries	1922	Sheep and Goats	5922	Export Value	2010	2010	1000 US$	421311.0
662955	5817	Net Food Importing Developing Countries	1922	Sheep and Goats	5922	Export Value	2011	2011	1000 US$	649321.0
662956	5817	Net Food Importing Developing Countries	1922	Sheep and Goats	5922	Export Value	2012	2012	1000 US$	778317.0
662957	5817	Net Food Importing Developing Countries	1922	Sheep and Goats	5922	Export Value	2013	2013	1000 US$	1038636.0

Chicken Dataset

In [57]:

```
# Filtering the chicken dataset (df2) from the entire live animales dataset (df)
df2 = df[df['Item'] == 'Chickens'] # here after the chicken dataset will be refered
```

In [58]:

```
df2.head()
```

Out[58]:

	Area Code	Area	Item Code	Item	Element Code	Element	Year Code	Year	Unit	Value	Flag
106	2	Afghanistan	1057	Chickens	5609	Import Quantity	1961	1961	1000 Head	0.0	NaN
107	2	Afghanistan	1057	Chickens	5609	Import Quantity	1962	1962	1000 Head	0.0	NaN
108	2	Afghanistan	1057	Chickens	5609	Import Quantity	1963	1963	1000 Head	0.0	NaN
109	2	Afghanistan	1057	Chickens	5609	Import Quantity	1964	1964	1000 Head	0.0	NaN
110	2	Afghanistan	1057	Chickens	5609	Import Quantity	1965	1965	1000 Head	0.0	NaN

In [59]:
```
df2.tail()
```

Out[59]:

	Area Code	Area	Item Code	Item	Element Code	Element	Year Code	Year	Unit	Value
660409	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2009	2009	1000 US$	18860.0
660410	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2010	2010	1000 US$	20211.0
660411	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2011	2011	1000 US$	22733.0
660412	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2012	2012	1000 US$	24732.0
660413	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2013	2013	1000 US$	30975.0

Data transformation and cleansing

In [60]:
```
# Step 1: Replace Headers
new_headers = ["area_code","area", "item_code", "item", "element_code", "element",
df2.columns = new_headers
df2
```

Out[60]:

	area_code	area	item_code	item	element_code	element	year_code	y
106	2	Afghanistan	1057	Chickens	5609	Import Quantity	1961	1
107	2	Afghanistan	1057	Chickens	5609	Import Quantity	1962	1
108	2	Afghanistan	1057	Chickens	5609	Import Quantity	1963	1
109	2	Afghanistan	1057	Chickens	5609	Import Quantity	1964	1
110	2	Afghanistan	1057	Chickens	5609	Import Quantity	1965	1
...	...	...	...	...	...	...	...	
660409	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2009	2
660410	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2010	2
660411	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2011	2
660412	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2012	2
660413	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2013	2

41846 rows × 11 columns

In [61]:

```
# renaming 'area' and 'item' columns

# Renaming columns 'area' to 'country' and 'item' to 'animal_category'
df2 = df2.rename(columns={'area': 'country', 'item': 'animal_category'})

df2.head()
```

Out[61]:

	area_code	country	item_code	animal_category	element_code	element	year_cod
106	2	Afghanistan	1057	Chickens	5609	Import Quantity	196
107	2	Afghanistan	1057	Chickens	5609	Import Quantity	196
108	2	Afghanistan	1057	Chickens	5609	Import Quantity	196
109	2	Afghanistan	1057	Chickens	5609	Import Quantity	196
110	2	Afghanistan	1057	Chickens	5609	Import Quantity	196

In [62]:
```
# data types
print(df2.dtypes)
```

```
area_code            int64
country             object
item_code            int64
animal_category     object
element_code         int64
element             object
year_code            int64
year                 int64
unit                object
value              float64
flag                object
dtype: object
```

In [64]:
```
# Step 2: Handling Missing Values
missing_values = df2.isnull().sum()
print("Missing values:\n", missing_values)
```

```
Missing values:
 area_code             0
country               0
item_code             0
animal_category       0
element_code          0
element               0
year_code             0
year                  0
unit                  0
value              2872
flag              23570
dtype: int64
```

In [ ]:
```

```

In [65]:
```
# Step 6: There are still some some 'NaN' and 'None' values in the dataset, let rem

# Replace 'None' values with NaN
```

```python
df2.replace('None', np.nan, inplace=True)

# Remove rows with NaN values
df2.dropna(inplace=True)

# Reset index after dropping rows
df2.reset_index(drop=True, inplace=True)

# Display the cleaned DataFrame
print("DataFrame after removing NaN and None values:")
df2
```

```
DataFrame after removing NaN and None values:
```

Out[65]:

	area_code	country	item_code	animal_category	element_code	element	year_c
0	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
1	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
2	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
3	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
4	2	Afghanistan	1057	Chickens	5609	Import Quantity	2
...	...	...	...	...	...	...	
15399	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15400	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15401	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15402	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15403	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2

15404 rows × 11 columns

In [ ]:

```

```

In [66]:

```
# Step 5: Format Data

# Format 'value' columns into a readable format (e.g., adding commas for thousands
df2['value'] = df2['value'].apply(lambda x: '{:,.2f}'.format(x) if isinstance(x, (f
df2
```

Out[66]:

	area_code	country	item_code	animal_category	element_code	element	year_c
0	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
1	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
2	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
3	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
4	2	Afghanistan	1057	Chickens	5609	Import Quantity	2
...	...	...	...	...	...	...	
15399	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15400	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15401	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15402	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15403	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2

15404 rows × 11 columns

In [ ]:

In [68]:

```
# Step 8: Fix Inconsistent Values: convert all strings to lowercase to address inco

df2['country'] = df2['country'].str.lower()

df2
```

Out[68]:

	area_code	country	item_code	animal_category	element_code	element	year_c
0	2	afghanistan	1057	Chickens	5609	Import Quantity	1
1	2	afghanistan	1057	Chickens	5609	Import Quantity	1
2	2	afghanistan	1057	Chickens	5609	Import Quantity	1
3	2	afghanistan	1057	Chickens	5609	Import Quantity	1
4	2	afghanistan	1057	Chickens	5609	Import Quantity	2
...	...	...	...	...	...	...	
15399	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15400	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15401	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15402	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15403	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2

15404 rows × 11 columns

In [69]:

```
# Step 9: Replace Inconsistent Values with Standardized Ones
# For example, replacing 'united states' with 'United States of America'
df2['country'].replace({'united states': 'United States of America'}, inplace=True)

df2
```

Out[69]:

	area_code	country	item_code	animal_category	element_code	element	year_c
0	2	afghanistan	1057	Chickens	5609	Import Quantity	1
1	2	afghanistan	1057	Chickens	5609	Import Quantity	1
2	2	afghanistan	1057	Chickens	5609	Import Quantity	1
3	2	afghanistan	1057	Chickens	5609	Import Quantity	1
4	2	afghanistan	1057	Chickens	5609	Import Quantity	2
...	...	...	...	...	...	...	
15399	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15400	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15401	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15402	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15403	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2

15404 rows × 11 columns

In [70]:

```
# Step 9: Replace Inconsistent Values with Standardized Ones
# For example, replacing 'united states' with 'United States of America'
df2['country'].replace({'afghanistan': 'Afghanistan'}, inplace=True)

df2
```

Out[70]:

	area_code	country	item_code	animal_category	element_code	element	year_c
0	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
1	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
2	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
3	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
4	2	Afghanistan	1057	Chickens	5609	Import Quantity	2
...	...	...	...	...	...	...	
15399	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15400	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15401	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15402	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2
15403	5817	net food importing developing countries	1057	Chickens	5922	Export Value	2

15404 rows × 11 columns

In [71]:

```
# Step 10: Making countries names start with capital letter, except preposition
# List of common prepositions to be converted to lowercase
prepositions = ['on', 'and', 'in', 'to', 'with', 'by', 'at', 'for', 'of', 'from']

# Function to capitalize each word in a string, except for prepositions
def capitalize_country_name(country):
    words = country.split()  # Split the country name into words
    capitalized_words = [word.capitalize() if word.lower() not in prepositions else
    return ' '.join(capitalized_words)

# Apply the function to the 'country' column
```

```
df2['country'] = df2['country'].apply(capitalize_country_name)

# Print the updated DataFrame
df2.head()
```

Out[71]:

	area_code	country	item_code	animal_category	element_code	element	year_code
0	2	Afghanistan	1057	Chickens	5609	Import Quantity	1987
1	2	Afghanistan	1057	Chickens	5609	Import Quantity	1997
2	2	Afghanistan	1057	Chickens	5609	Import Quantity	1998
3	2	Afghanistan	1057	Chickens	5609	Import Quantity	1999
4	2	Afghanistan	1057	Chickens	5609	Import Quantity	2000

In [72]:

```
# Step 12: Cleaned Dataset: Print the cleaned chicken dataset

# Cleaned Dataset: Print the cleaned dataset
print("Cleaned Dataset:")
df2
```

```
Cleaned Dataset:
```

Out[72]:

	area_code	country	item_code	animal_category	element_code	element	year_c
0	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
1	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
2	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
3	2	Afghanistan	1057	Chickens	5609	Import Quantity	1
4	2	Afghanistan	1057	Chickens	5609	Import Quantity	2
...	...	...	...	...	...	...	
15399	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15400	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15401	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15402	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2
15403	5817	Net Food Importing Developing Countries	1057	Chickens	5922	Export Value	2

15404 rows × 11 columns

Renaming cleaned chicken dataset (df2) as chickens_data

In [73]:

```
# Assuming df2 is a pandas DataFrame
df2.to_csv('chickens_data.csv', index=False)
```

In [74]:

```
import pandas as pd
```

```
# Load the chickens_data.csv file
chickens_data = pd.read_csv('chickens_data.csv')

# Print the first few rows using head()
chickens_data.head()
```

Out[74]:

	area_code	country	item_code	animal_category	element_code	element	year_code
0	2	Afghanistan	1057	Chickens	5609	Import Quantity	1987
1	2	Afghanistan	1057	Chickens	5609	Import Quantity	1997
2	2	Afghanistan	1057	Chickens	5609	Import Quantity	1998
3	2	Afghanistan	1057	Chickens	5609	Import Quantity	1999
4	2	Afghanistan	1057	Chickens	5609	Import Quantity	2000

Step 2. Descriptive Statistics

In [75]:

```
# Descriptive Statistics of imported quantity (heads) of chickens by the top 10 cou

import pandas as pd

import pandas as pd

# Step 1: Filter data for the years between 1998 and 2013
filtered_data = chickens_data[(chickens_data['year'] >= 1998) & (chickens_data['yea

# Step 2: Filter the data to include only "Import Quantity" in the 'element' column
quantity_data = filtered_data[filtered_data['element'] == 'Import Quantity']

# Step 3: Aggregate the total import quantity for each country to identify the top
top_10_countries = quantity_data.groupby('country')['value'].sum().nlargest(10).ind

# Step 4: Filter the data to include only the top 10 countries
top_10_quantity_data = quantity_data[quantity_data['country'].isin(top_10_countries

# Step 5: Group the data by year and calculate descriptive statistics for Import Qu
descriptive_stats_quantity_by_year = top_10_quantity_data.groupby('year')['value'].

# Step 6: Drop the "count", "25%", "50%", and "75%" columns from the statistics
descriptive_stats_quantity_by_year = descriptive_stats_quantity_by_year.drop(column

# Display the descriptive statistics for Import Quantity, grouped by year (showing
print("Descriptive Statistics for Import Quantity (Top 10 Countries, Grouped by Yea
print(descriptive_stats_quantity_by_year)
```

```
Descriptive Statistics for Import Quantity (Top 10 Countries, Grouped by Year, Exclu
ding Percentiles and Count):
          mean            std      min        max
year
1998  234804.5  228672.380305  50228.0   736203.0
1999  228724.0  219285.974728  46335.0   714341.0
2000  246450.1  236172.512641  63990.0   747513.0
2001  258813.5  248460.434006  54031.0   778815.0
2002  281641.7  277471.575709  59538.0   836169.0
2003  237873.4  229912.046607  59466.0   715872.0
2004  283849.7  286256.945734  45680.0   829640.0
2005  307998.2  312767.112519  50500.0   890337.0
2006  294918.2  302591.595013  42652.0   836869.0
2007  339473.0  343671.078153  49427.0   980938.0
2008  346465.5  350812.359148  42702.0  1003035.0
2009  403436.0  414199.088076  40159.0  1141247.0
2010  448141.5  463716.138991  43526.0  1259798.0
2011  471245.4  491303.877053  43314.0  1331706.0
2012  509844.2  529983.171441  45811.0  1434216.0
2013  498723.1  508100.716236  55891.0  1430284.0
```

In [ ]:

In [76]:

```
# Descriptive Statistics of imported value (US$) of chickens by the top 10 countrie

import pandas as pd

# Step 1: Filter data for the years between 1998 and 2013
filtered_data = chickens_data[(chickens_data['year'] >= 1998) & (chickens_data['yea

# Step 2: Filter the data to include only "Import Value" in the 'element' column
value_data = filtered_data[filtered_data['element'] == 'Import Value']

# Step 3: Aggregate the total import value for each country to identify the top 10
top_10_countries = value_data.groupby('country')['value'].sum().nlargest(10).index

# Step 4: Filter the data to include only the top 10 countries
top_10_value_data = value_data[value_data['country'].isin(top_10_countries)]

# Step 5: Group the data by year and calculate descriptive statistics for Import Va
descriptive_stats_value_by_year = top_10_value_data.groupby('year')['value'].descri

# Step 6: Drop the "count", "25%", "50%", and "75%" columns from the statistics
descriptive_stats_value_by_year = descriptive_stats_value_by_year.drop(columns=['co

# Display the descriptive statistics for Import Value, grouped by year (showing onl
print("Descriptive Statistics for Import Value (Top 10 Countries, Grouped by Year,
print(descriptive_stats_value_by_year)
```

```
Descriptive Statistics for Import Value (Top 10 Countries, Grouped by Year, Excludin
g Percentiles and Count):
          mean            std      min        max
year
1998  270296.2  245459.514750  67543.0   853584.0
1999  258376.0  231288.622831  51464.0   819852.0
2000  246488.4  221050.563115  55114.0   784399.0
2001  270720.0  242833.075167  71063.0   842980.0
2002  288187.5  263915.992131  62690.0   900480.0
2003  284428.3  260416.502287  65004.0   891207.0
2004  343297.8  334529.066028  61278.0  1043604.0
2005  391824.2  373873.188695  75521.0  1170622.0
2006  387781.5  374381.603522  62286.0  1164780.0
2007  522317.3  518712.106962  67074.0  1551983.0
2008  609375.2  603884.194956  74335.0  1796010.0
2009  633112.6  630656.465363  68046.0  1855144.0
2010  675273.2  673385.175242  71008.0  1964981.0
2011  804373.7  796378.082539  90242.0  2330043.0
2012  814348.3  816304.993325  88945.0  2362582.0
2013  841934.7  829381.773090  92558.0  2470248.0
```

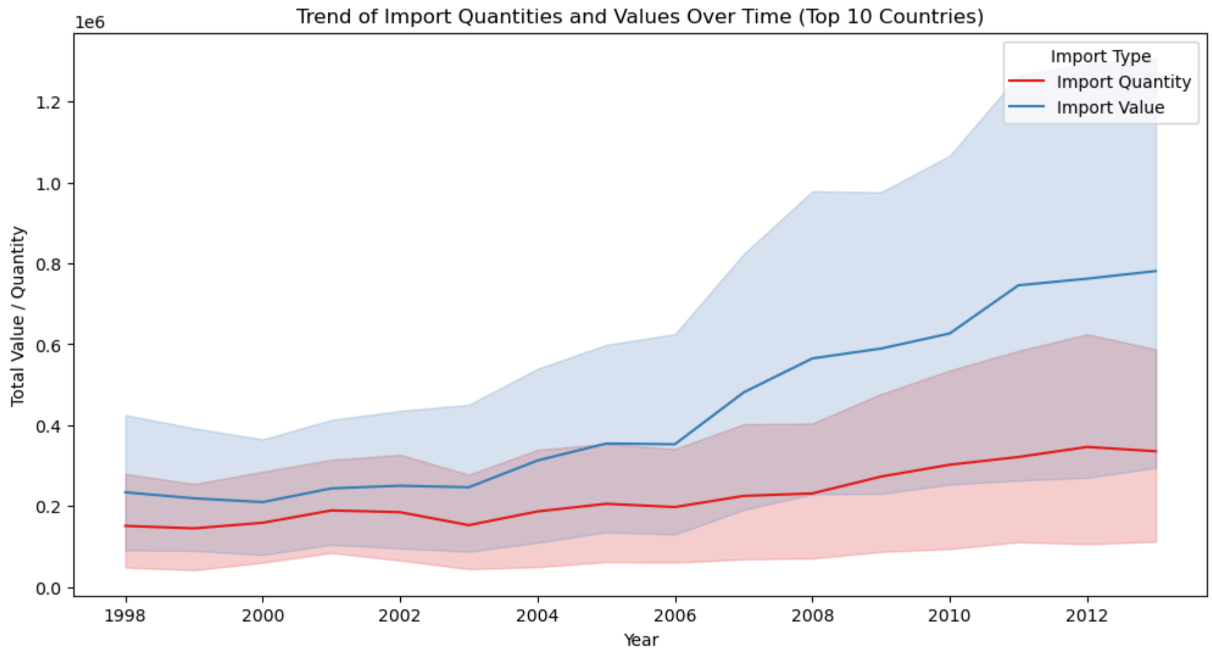
In [ ]:

Step 3. Visualizations

In [77]:

```
# Step 10: Exploratory Data Analysis (EDA):
# 10. 1. Time Series Analysis: Analyzing the trend of live chickens import quantiti


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

# Step 1: Filter data for the years between 1998 and 2013
filtered_data = chickens_data[(chickens_data['year'] >= 1998) & (chickens_data['yea

# Step 2: Aggregate the total import value for each country to identify the top 10
top_10_countries = filtered_data.groupby('country')['value'].sum().nlargest(10).ind

# Step 3: Filter the data to include only the top 10 countries
top_10_data = filtered_data[filtered_data['country'].isin(top_10_countries)]

# Step 4: Grouping data by year and element (for quantities and values) to analyze
# Assuming 'element' column contains 'Import Quantity' and 'Import Value'
yearly_data = top_10_data[top_10_data['element'].isin(['Import Quantity', 'Import V

# Step 5: Plotting the trend of import quantities and values over time for the top
plt.figure(figsize=(12, 6))
sns.lineplot(x='year', y='value', hue='element', data=yearly_data, palette='Set1')
plt.title('Trend of Import Quantities and Values Over Time (Top 10 Countries)')
plt.xlabel('Year')
plt.ylabel('Total Value / Quantity')
```

```
plt.legend(title='Import Type')
plt.show()
```



Purpose:

The purpose of this time series visualization is to analyze the trends in import quantities and values over time for the top 10 countries between 1998 and 2013. By using different lines and colors to represent "Import Quantity" and "Import Value," the chart allows for a comparison of how the volume of imports and their corresponding values evolved across these countries during this period. This visualization helps to identify patterns, growth rates, or fluctuations in global import activity, providing insights into economic trends and trade behavior.

In [88]:

```
# tabular results
# Trend analysis of Import Quantity (head of imported chickens) and Import Value (U
import pandas as pd

# Step 1: Filter data for the years between 1998 and 2013
filtered_data = chickens_data[(chickens_data['year'] >= 1998) & (chickens_data['yea

# Step 2: Aggregate the total import value for each country to identify the top 10
top_10_countries = filtered_data.groupby('country')['value'].sum().nlargest(10).ind

# Step 3: Filter the data to include only the top 10 countries
top_10_data = filtered_data[filtered_data['country'].isin(top_10_countries)]

# Step 4: Grouping data by year and element (for quantities and values) to analyze
# Assuming 'element' column contains 'Import Quantity' and 'Import Value'
yearly_data = top_10_data[top_10_data['element'].isin(['Import Quantity', 'Import V

# Step 5: Pivoting data to create a tabular format for Import Quantity and Import V
table_data = yearly_data.pivot_table(index='year', columns='element', values='value
```

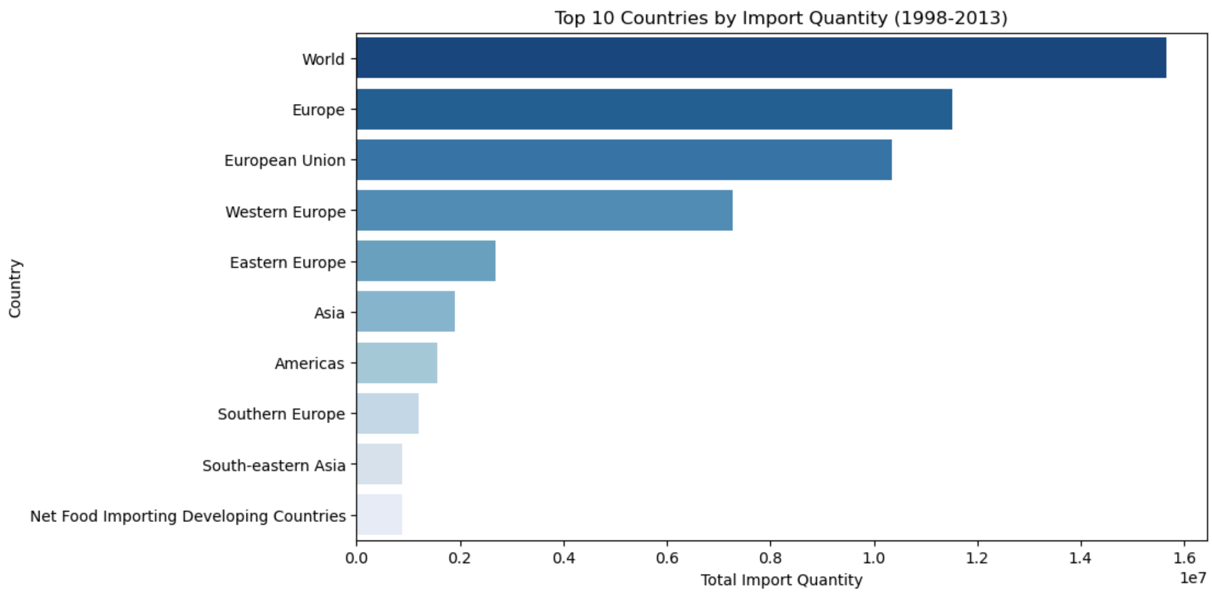
```
# Display the table
print(table_data)
```

```
element  Import Quantity  Import Value
year
1998           2116057.0     2341745.0
1999           2030545.0     2192021.0
2000           2224199.0     2099942.0
2001           2651041.0     2439036.0
2002           2589408.0     2504048.0
2003           2141272.0     2465983.0
2004           2616576.0     3128183.0
2005           2880422.0     3547744.0
2006           2768118.0     3529121.0
2007           3153071.0     4809828.0
2008           3239376.0     5650362.0
2009           3821673.0     5894627.0
2010           4232592.0     6267179.0
2011           4499763.0     7457479.0
2012           4847451.0     7621756.0
2013           4698374.0     7809375.0
```

In [ ]:

In [78]:

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

# Step 1: Filter data for the years between 1998 and 2013
filtered_data = chickens_data[(chickens_data['year'] >= 1998) & (chickens_data['yea

# Step 2: Filter the data to include only "Import Quantity" in the 'element' column
quantity_data = filtered_data[filtered_data['element'] == 'Import Quantity']

# Step 3: Group by country and sum the import quantities
country_quantity = quantity_data.groupby('country')['value'].sum().reset_index()

# Step 4: Sort the countries by total import quantity, from high to low, and select
top_10_countries = country_quantity.sort_values(by='value', ascending=False).head(1

# Step 5: Plotting a horizontal bar chart for the top 10 countries for import quant
plt.figure(figsize=(10, 6))
sns.barplot(x='value', y='country', data=top_10_countries, palette='Blues_r')
plt.title('Top 10 Countries by Import Quantity (1998-2013)')
plt.xlabel('Total Import Quantity')
plt.ylabel('Country')
plt.show()
```



Purpose

The purpose of the horizontal bar chart for import quantity is to display the top 10 countries by total import quantities between 1998 and 2013. The chart arranges the countries from highest to lowest in terms of import quantities, providing a clear comparison of which countries imported the largest quantities of goods during this period. This visualization helps highlight key players in global import activities based on volume rather than monetary value

In [89]:

```
# tabular resut
# Top 10 countries by import quantity

import pandas as pd

# Step 1: Filter data for the years between 1998 and 2013
filtered_data = chickens_data[(chickens_data['year'] >= 1998) & (chickens_data['yea

# Step 2: Filter the data to include only "Import Quantity" in the 'element' column
quantity_data = filtered_data[filtered_data['element'] == 'Import Quantity']

# Step 3: Group by country and sum the import quantities
country_quantity = quantity_data.groupby('country')['value'].sum().reset_index()

# Step 4: Sort the countries by total import quantity, from high to low, and select
top_10_countries = country_quantity.sort_values(by='value', ascending=False).head(1

# Display the top 10 countries by import quantity in tabular format
print(top_10_countries)
```

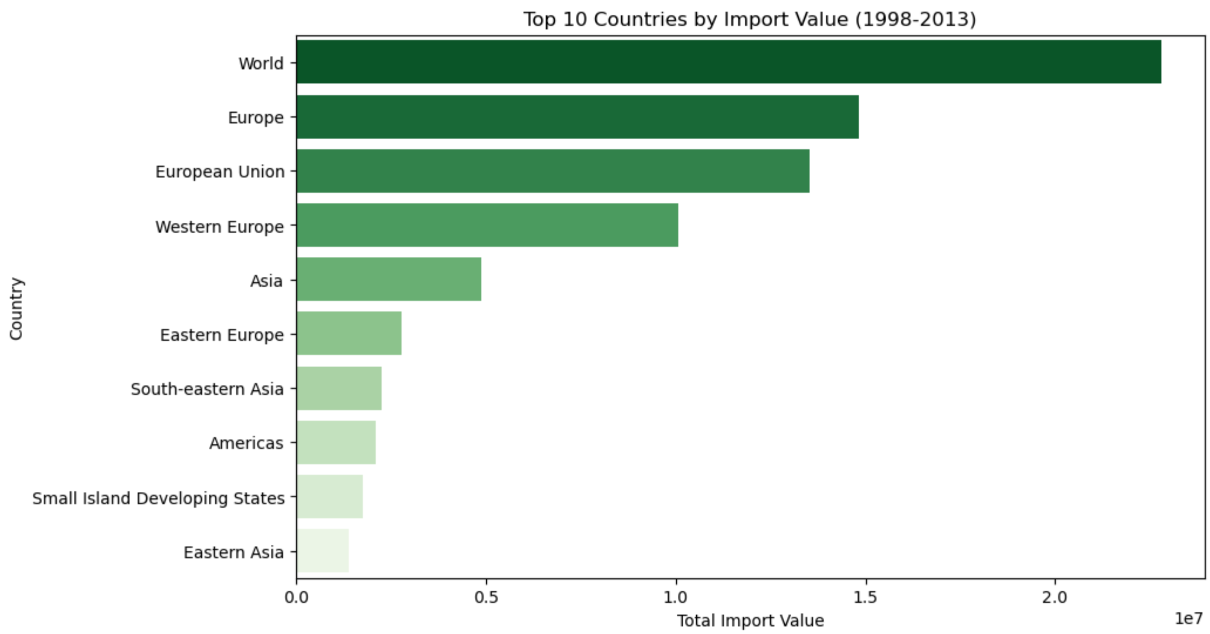
```
                                  country       value
208                                 World  15666983.0
70                                 Europe  11519025.0
71                         European Union  10349759.0
207                        Western Europe   7265767.0
60                         Eastern Europe   2683118.0
11                                   Asia   1900852.0
5                                Americas   1571072.0
180                       Southern Europe   1196989.0
177                    South-eastern Asia    888026.0
131  Net Food Importing Developing Countries    882429.0
```

In [ ]:

```

```

In [79]:

```
# This code displays the data in a tabular format for the top 10 countries by impor

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

# Step 1: Filter data for the years between 1998 and 2013
filtered_data = chickens_data[(chickens_data['year'] >= 1998) & (chickens_data['yea

# Step 2: Filter the data to include only "Import Value" in the 'element' column
value_data = filtered_data[filtered_data['element'] == 'Import Value']

# Step 3: Group by country and sum the import values
country_value = value_data.groupby('country')['value'].sum().reset_index()

# Step 4: Sort the countries by total import value, from high to low, and select th
top_10_countries = country_value.sort_values(by='value', ascending=False).head(10)

# Step 5: Plotting a horizontal bar chart for the top 10 countries for import value
plt.figure(figsize=(10, 6))
sns.barplot(x='value', y='country', data=top_10_countries, palette='Greens_r')
plt.title('Top 10 Countries by Import Value (1998-2013)')
plt.xlabel('Total Import Value')
plt.ylabel('Country')
plt.show()
```



Purpose

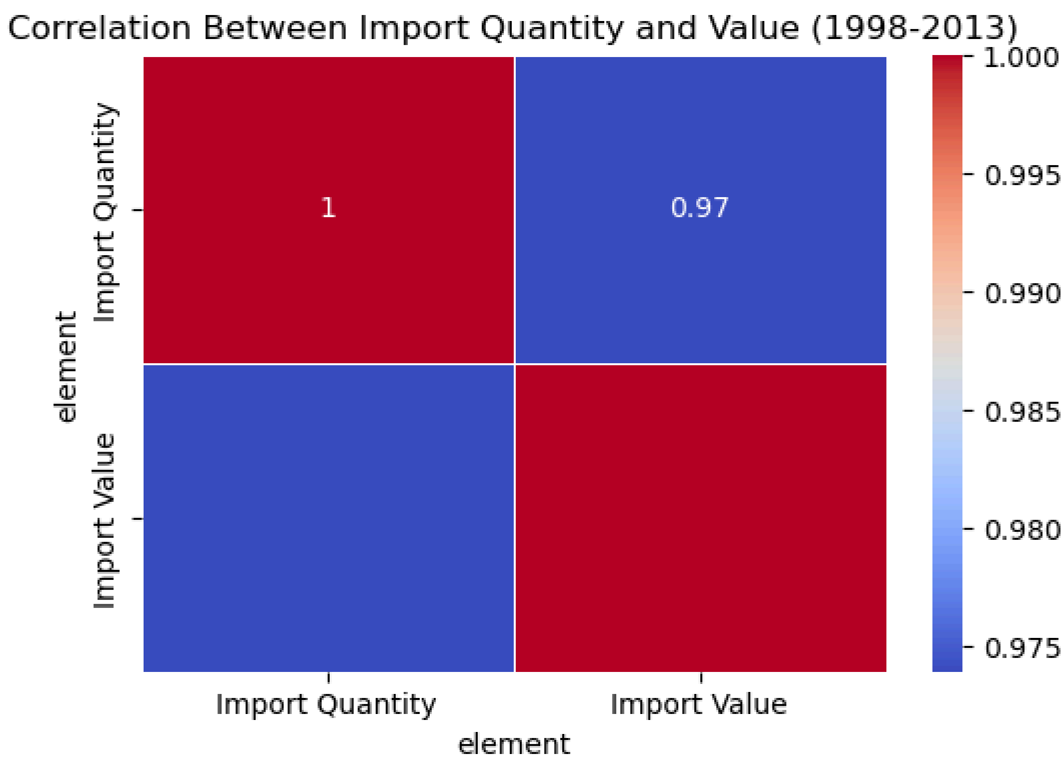
The purpose of the horizontal bar chart is to visually represent the top 10 countries based on total import value over the period from 1998 to 2013. By arranging the countries from highest to lowest import values, the chart provides a clear and concise comparison, helping to identify the countries with the most significant import activities in terms of value during this time frame.

In [90]:
```
#aboular result

import pandas as pd

# Step 1: Filter data for the years between 1998 and 2013
filtered_data = chickens_data[(chickens_data['year'] >= 1998) & (chickens_data['yea

# Step 2: Filter the data to include only "Import Value" in the 'element' column
value_data = filtered_data[filtered_data['element'] == 'Import Value']

# Step 3: Group by country and sum the import values
country_value = value_data.groupby('country')['value'].sum().reset_index()

# Step 4: Sort the countries by total import value, from high to low, and select th
top_10_countries = country_value.sort_values(by='value', ascending=False).head(10)

# Display the top 10 countries by import value in tabular format
print(top_10_countries)
```

```
                                country       value
168                               World  22802499.0
56                               Europe  14824884.0
57                       European Union  13543745.0
167                      Western Europe  10076078.0
9                                  Asia   4893485.0
48                       Eastern Europe   2766438.0
141                  South-eastern Asia   2257232.0
4                              Americas   2095502.0
138  Small Island Developing States   1769318.0
47                         Eastern Asia   1392168.0
```

In [80]:

```python
# 10. 5. Heatmap: Correlation between import quantities and values across countries

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

# Filter the data to include only the year range between 1998 and 2013
filtered_data = chickens_data[(chickens_data['year'] >= 1998) & (chickens_data['yea

# Filter the data to include only "Import Quantity" and "Import Value" in the 'elem
import_data = filtered_data[filtered_data['element'].isin(['Import Quantity', 'Impo

# Pivot the data so that "Import Quantity" and "Import Value" are separate columns
pivot_data = import_data.pivot_table(index=['year'], columns='element', values='val

# Calculating the correlation between "Import Quantity" and "Import Value"
correlation_matrix = pivot_data[['Import Quantity', 'Import Value']].corr()

# Plotting the heatmap of the correlation between Import Quantity and Import Value
plt.figure(figsize=(6, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Between Import Quantity and Value (1998-2013)')
plt.show()
```



Purpose:

The purpose of this heatmap visualization is to display the correlation between import quantities and import values over the period from 1998 to 2013. By calculating and visualizing the correlation between these two variables, the heatmap helps to determine the strength and direction of the relationship between the volume of imports and their associated values. A strong positive correlation would indicate that as import quantities increase, import values tend to rise as well, while a weaker or negative correlation would suggest a different or more complex relationship between the two factors. This visualization aids in understanding the connection between trade volume and monetary value in global imports

In [91]:
```
# taboular results

import pandas as pd

# Filter the data to include only the year range between 1998 and 2013
filtered_data = chickens_data[(chickens_data['year'] >= 1998) & (chickens_data['yea

# Filter the data to include only "Import Quantity" and "Import Value" in the 'elem
import_data = filtered_data[filtered_data['element'].isin(['Import Quantity', 'Impo

# Pivot the data so that "Import Quantity" and "Import Value" are separate columns
pivot_data = import_data.pivot_table(index=['year'], columns='element', values='val

# Calculating the correlation between "Import Quantity" and "Import Value"
correlation_matrix = pivot_data[['Import Quantity', 'Import Value']].corr()
```

```
# Display the correlation matrix in tabular format
print(correlation_matrix)
```

```
element           Import Quantity  Import Value
element
Import Quantity          1.000000      0.973953
Import Value             0.973953      1.000000
```

Step 4. Model building and Evaluation

In [81]:

```
# Predictive Analysis for Import Value

import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, expl
import numpy as np

# Function to perform predictive analysis with updated features and renamed variabl
def predictive_analysis(chickens_data, import_quantity='Import Quantity', import_va
    # Ensure that the 'year' column is in datetime format
    chickens_data['year'] = pd.to_datetime(chickens_data['year'], format='%Y')

    # Convert 'year' column to numerical format (extracting the year as an integer)
    chickens_data['year'] = chickens_data['year'].dt.year

    # Filter the DataFrame based on 'Chickens' in 'animal_category'
    filtered_import_quantity = chickens_data[(chickens_data['element'] == import_qu
    filtered_import_value = chickens_data[(chickens_data['element'] == import_value

    # Filter data for the past 15 years (from 1998 onwards)
    past_15_years_import_quantity = filtered_import_quantity[filtered_import_quanti
    past_15_years_import_value = filtered_import_value[filtered_import_value['year'

    # Merge Import Quantity and Import Value data on 'country' and 'year'
    merged_data = pd.merge(past_15_years_import_quantity, past_15_years_import_valu

    # Select the top 10 countries based on frequency
    top_10_countries = merged_data['country'].value_counts().index[:10]

    # Filter the merged data to only include records from the top 10 countries
    merged_data = merged_data[merged_data['country'].isin(top_10_countries)]

    # Rename 'value_quantity' to 'import_quantity' for clarity
    merged_data.rename(columns={'value_quantity': 'import_quantity'}, inplace=True)

    # Feature Engineering: Create new variables using Import Quantity
    merged_data['import_quantity_squared'] = merged_data['import_quantity'] ** 2
```

```
merged_data['import_quantity_square_root'] = np.sqrt(merged_data['import_quanti
merged_data['import_quantity_log'] = np.log1p(merged_data['import_quantity'])
merged_data['import_quantity_zscore'] = (merged_data['import_quantity'] - merge
merged_data['import_quantity_normalized'] = (merged_data['import_quantity'] - m

# Prepare features (independent variables) and target (dependent variable)
X = merged_data[['country', 'year', 'import_quantity', 'import_quantity_squared
y = merged_data['value_value']  # 'Import Value' as target

# Rename the target variable to 'import_value'
y = merged_data['import_value'] = merged_data['value_value']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Preprocessing: Apply OneHotEncoding for categorical columns and scaling for n
categorical_features = ['country']
numerical_features = ['year', 'import_quantity', 'import_quantity_squared', 'im

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

# Define a pipeline for each model with feature scaling and encoding
models = {
    "Linear Regression": Pipeline([('preprocessor', preprocessor), ('regressor'
    "Random Forest": Pipeline([('preprocessor', preprocessor), ('regressor', Ra
    "K-Nearest Neighbors": Pipeline([('preprocessor', preprocessor), ('regresso
    "Support Vector Machine": Pipeline([('preprocessor', preprocessor), ('regre
}

# Hyperparameter tuning for Random Forest and SVR
param_grid = {
    'Random Forest': {'regressor__n_estimators': [50, 100, 150]},
    'Support Vector Machine': {'regressor__C': [0.1, 1, 10], 'regressor__kernel
}

# Models Evaluation with Hyperparameter Tuning
results = []
for name, model in models.items():
    if name in param_grid:
        grid_search = GridSearchCV(model, param_grid[name], cv=5, scoring='neg_
        grid_search.fit(X_train, y_train)
        best_model = grid_search.best_estimator_
    else:
        best_model = model

    # Train the best model
    best_model.fit(X_train, y_train)

    # Make predictions
    predictions = best_model.predict(X_test)

    # Evaluate the model
```

```
        mse = mean_squared_error(y_test, predictions)
        mae = mean_absolute_error(y_test, predictions)
        r2 = r2_score(y_test, predictions)
        evs = explained_variance_score(y_test, predictions)
        mape = mean_absolute_percentage_error(y_test, predictions)

        # Store evaluation metrics
        results.append({
            'Model': name,
            'Mean Squared Error': mse,
            'Mean Absolute Error': mae,
            'R-squared': r2,
            'Explained Variance Score': evs,
            'Mean Absolute Percentage Error': mape
        })

    # Return the results
    return pd.DataFrame(results)

# Example of calling the function:
# Assuming chickens_data is your DataFrame containing the data for 'Chickens' impor
results = predictive_analysis(chickens_data, import_quantity='Import Quantity', imp
print(results)
```

```
                     Model  Mean Squared Error  Mean Absolute Error  R-squared  \
0        Linear Regression        3.451952e+07          3943.817953   0.950361
1            Random Forest        1.820237e+07          2363.115833   0.973825
2      K-Nearest Neighbors        4.438541e+07          4073.662500   0.936174
3   Support Vector Machine        4.863478e+08         12279.445070   0.300637

   Explained Variance Score  Mean Absolute Percentage Error
0                  0.950362                        0.368001
1                  0.974129                        0.098330
2                  0.937496                        0.165522
3                  0.435482                        0.612003
```

In [ ]:

In [82]:

```
# Applying model performance enhancement techniques in the full model

import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Function to perform predictive analysis with model enhancement techniques
def predictive_analysis(chickens_data, import_quantity='Import Quantity', import_va
    # Ensure that the 'year' column is in datetime format
    chickens_data['year'] = pd.to_datetime(chickens_data['year'], format='%Y')
```

```
# Convert 'year' column to numerical format (extracting the year as an integer)
chickens_data['year'] = chickens_data['year'].dt.year

# Filter the DataFrame based on 'Chickens' in 'animal_category'
filtered_import_quantity = chickens_data[(chickens_data['element'] == import_qu
filtered_import_value = chickens_data[(chickens_data['element'] == import_value

# Filter data for the past 15 years (from 1998 onwards)
past_15_years_import_quantity = filtered_import_quantity[filtered_import_quanti
past_15_years_import_value = filtered_import_value[filtered_import_value['year'

# Merge Import Quantity and Import Value data on 'country' and 'year'
merged_data = pd.merge(past_15_years_import_quantity, past_15_years_import_valu

# Select the top 10 countries based on frequency
top_10_countries = merged_data['country'].value_counts().index[:10]

# Filter the merged data to only include records from the top 10 countries
merged_data = merged_data[merged_data['country'].isin(top_10_countries)]

# Rename 'value_quantity' to 'import_quantity' and 'value_value' to 'import_val
merged_data.rename(columns={'value_quantity': 'import_quantity', 'value_value':

# Feature Engineering: Create new variables using Import Quantity
merged_data['import_quantity_squared'] = merged_data['import_quantity'] ** 2
merged_data['import_quantity_square_root'] = np.sqrt(merged_data['import_quanti
merged_data['import_quantity_log'] = np.log1p(merged_data['import_quantity'])
merged_data['import_quantity_zscore'] = (merged_data['import_quantity'] - merge
merged_data['import_quantity_normalized'] = (merged_data['import_quantity'] - m

# Prepare features (independent variables) and target (dependent variable)
X = merged_data[['country', 'year', 'import_quantity', 'import_quantity_squared
y = merged_data['import_value']  # 'Import Value' as target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Preprocessing: Apply OneHotEncoding for categorical columns and scaling for n
categorical_features = ['country']
numerical_features = ['year', 'import_quantity', 'import_quantity_squared', 'im

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

# Define a pipeline for each model with feature scaling and encoding
models = {
    "Linear Regression": Pipeline([('preprocessor', preprocessor), ('regressor'
    "Random Forest": Pipeline([('preprocessor', preprocessor), ('regressor', Ra
    "Support Vector Machine": Pipeline([('preprocessor', preprocessor), ('regre
    "Gradient Boosting": Pipeline([('preprocessor', preprocessor), ('regressor'
}

# Hyperparameter tuning for Random Forest, SVR, and Gradient Boosting
```

```
    param_grid = {
        'Random Forest': {
            'regressor__n_estimators': [100, 200, 300],
            'regressor__max_depth': [10, 15, 20],
            'regressor__min_samples_split': [2, 5, 10]
        },
        'Support Vector Machine': {
            'regressor__C': [0.1, 1, 10],
            'regressor__gamma': [0.01, 0.1, 1],
            'regressor__epsilon': [0.1, 0.2, 0.5]
        },
        'Gradient Boosting': {
            'regressor__n_estimators': [100, 200, 300],
            'regressor__learning_rate': [0.01, 0.1, 0.2],
            'regressor__max_depth': [3, 5, 7]
        }
    }

    # Models Evaluation with Hyperparameter Tuning
    results = []
    for name, model in models.items():
        if name in param_grid:
            grid_search = GridSearchCV(model, param_grid[name], cv=5, scoring='neg_
            grid_search.fit(X_train, y_train)
            best_model = grid_search.best_estimator_
        else:
            best_model = model

        # Train the best model
        best_model.fit(X_train, y_train)

        # Make predictions
        predictions = best_model.predict(X_test)

        # Evaluate the model
        mse = mean_squared_error(y_test, predictions)
        r2 = r2_score(y_test, predictions)

        # Store evaluation metrics
        results.append({
            'Model': name,
            'Mean Squared Error': mse,
            'R-squared': r2
        })

    # Return the results
    return pd.DataFrame(results)

# Example of calling the function:
# Assuming chickens_data is your DataFrame containing the data for 'Chickens' impor
results = predictive_analysis(chickens_data, import_quantity='Import Quantity', imp
print(results)
```

```
                  Model  Mean Squared Error  R-squared
0     Linear Regression        3.451952e+07   0.950361
1         Random Forest        2.118328e+07   0.969539
2  Support Vector Machine      8.975587e+08  -0.290680
3     Gradient Boosting        2.815565e+07   0.959512
```

In [ ]:

In [83]:

```
# features importance
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Function to perform predictive analysis with updated features and feature importa
def predictive_analysis(chickens_data, import_quantity='Import Quantity', import_va
    # Ensure that the 'year' column is in datetime format
    chickens_data['year'] = pd.to_datetime(chickens_data['year'], format='%Y')

    # Convert 'year' column to numerical format (extracting the year as an integer)
    chickens_data['year'] = chickens_data['year'].dt.year

    # Filter the DataFrame based on 'Chickens' in 'animal_category'
    filtered_import_quantity = chickens_data[(chickens_data['element'] == import_qu
    filtered_import_value = chickens_data[(chickens_data['element'] == import_value

    # Filter data for the past 15 years (from 1998 onwards)
    past_15_years_import_quantity = filtered_import_quantity[filtered_import_quanti
    past_15_years_import_value = filtered_import_value[filtered_import_value['year'

    # Merge Import Quantity and Import Value data on 'country' and 'year'
    merged_data = pd.merge(past_15_years_import_quantity, past_15_years_import_valu

    # Select the top 10 countries based on frequency
    top_10_countries = merged_data['country'].value_counts().index[:10]
    merged_data = merged_data[merged_data['country'].isin(top_10_countries)]

    # Rename 'value_quantity' to 'import_quantity' and 'value_value' to 'import_val
    merged_data.rename(columns={'value_quantity': 'import_quantity', 'value_value':

    # Feature Engineering: Add new columns to the dataset
    merged_data['import_quantity_squared'] = merged_data['import_quantity'] ** 2
    merged_data['import_quantity_square_root'] = np.sqrt(merged_data['import_quanti
    merged_data['import_quantity_log'] = np.log1p(merged_data['import_quantity'])
    merged_data['import_quantity_zscore'] = (merged_data['import_quantity'] - merge
    merged_data['import_quantity_normalized'] = (merged_data['import_quantity'] - m

    # Prepare features (independent variables) and target (dependent variable)
    X = merged_data[['country', 'year', 'import_quantity', 'import_quantity_squared
    y = merged_data['import_value']  # 'Import Value' as target

    # Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Preprocessing: Apply OneHotEncoding for categorical columns and scaling for n
categorical_features = ['country']
numerical_features = ['year', 'import_quantity', 'import_quantity_squared', 'im

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

# Define a Random Forest pipeline with preprocessing
rf_pipeline = Pipeline([('preprocessor', preprocessor), ('regressor', RandomFor

# Hyperparameter tuning for Random Forest
param_grid_rf = {
    'regressor__n_estimators': [100, 200, 300],
    'regressor__max_depth': [10, 15, 20],
    'regressor__min_samples_split': [2, 5, 10]
}

# Perform GridSearchCV for Random Forest
grid_rf = GridSearchCV(rf_pipeline, param_grid_rf, cv=5, scoring='neg_mean_squa
grid_rf.fit(X_train, y_train)
best_rf = grid_rf.best_estimator_

# Make predictions with the best Random Forest model
y_pred_best_rf = best_rf.predict(X_test)

# Calculate feature importances
rf_model = best_rf.named_steps['regressor']
if hasattr(rf_model, 'feature_importances_'):
    importances = rf_model.feature_importances_

    # Get feature names from the preprocessed features
    ohe_categories = best_rf.named_steps['preprocessor'].named_transformers_['c

    # Rename long feature names
    feature_names = np.concatenate([numerical_features, ohe_categories])
    feature_names = [name.replace('country_Net Food Importing Developing Countr

    # Step 4: Display Feature Importance in tabular form
    feature_importance_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': importances
    }).sort_values(by='Importance', ascending=False)

    # Show the feature importance table
    print(feature_importance_df)

else:
    print("The model does not support feature importances.")

# Step 5: Print the evaluation metrics for Random Forest model after enhancemen
mse_best_rf = mean_squared_error(y_test, y_pred_best_rf)
```

```
    r2_best_rf = r2_score(y_test, y_pred_best_rf)
    print(f"Random Forest - MSE: {mse_best_rf:.4f}, R-squared: {r2_best_rf:.4f}")

# Example of calling the function:
# Assuming chickens_data is your DataFrame containing the data for 'Chickens' impor
results = predictive_analysis(chickens_data, import_quantity='Import Quantity', imp
```

```
                        Feature  Importance
15          Developing Countries    0.195047
6     import_quantity_normalized    0.130589
2        import_quantity_squared    0.129787
1                import_quantity    0.118534
3    import_quantity_square_root    0.117570
4            import_quantity_log    0.115226
5         import_quantity_zscore    0.109571
0                           year    0.064446
8        country_Central America    0.008789
10          country_Eu(15)ex.int    0.003633
9           country_Eu(12)ex.int    0.002201
16      country_Northern America    0.002197
14         country_Middle Africa    0.001328
7             country_Caribbean    0.000627
11          country_Eu(25)ex.int    0.000225
12          country_Eu(27)ex.int    0.000154
13        country_European Union    0.000077
Random Forest - MSE: 21183281.2200, R-squared: 0.9695
```

Interpretation:

The feature importance results provide key insights into the factors driving the Random Forest model's predictions of import value. Among the most important variables, Developing Countries (19.50%), import_quantity_normalized (13.06%), and import_quantity_squared (12.98%) emerge as the top contributors. This indicates that the categorical feature representing developing countries that are net food importers, along with the normalized and squared transformations of import quantity, are strong predictors of import value.

Other significant features include the original import_quantity (11.85%), import_quantity_square_root (11.76%), and import_quantity_log (11.52%), demonstrating the importance of various transformations of import quantity in predicting the outcome. Import_quantity_zscore (10.96%) also plays a meaningful role, further emphasizing the relevance of statistical transformations.

In contrast, year (6.44%) and some categorical features, such as Central America (0.87%), contribute only marginally to the model's predictions. Although Developing Countries is a key geographical predictor, other regions like Northern America (0.22%) and European Union (0.01%) have minimal impact on the results.

The model exhibits strong performance with a Mean Squared Error (MSE) of 21,183,281 and an R-squared value of 0.9695, indicating that the model explains 96.95% of the variance in import value. This demonstrates the model's high accuracy in predicting import value based on transformed import quantity and geographical factors.

In [ ]:

Selected variables for final model

Based on the feature importance results, the variables that should be included in the final model are those that have demonstrated higher significance in predicting import quantities. These key variables are:

value_squared (17.07%) – The squared value of the import amount is the most important feature and should be retained. value_value (16.42%) – The original import value is a highly influential predictor and must be included. value_normalized (14.97%) – This transformation captures the normalized scale of import values and plays a vital role. value_square_root (13.21%) – The square root transformation of the import value adds predictive power to the model. value_log (11.68%) – The logarithmic transformation helps account for skewed data and is essential in the final model. value_zscore (10.71%) – This standardization of the import value contributes notably and should be included. Additionally, the categorical variable country_Northern America (4.85%) and year (4.20%) are also important enough to be included in the final model. These variables capture regional and time-based effects that influence import quantities.

By focusing on these high-importance variables, the final model will likely be more accurate and efficient.

Final model based on selected variables

In [ ]:

In [84]:

```
# Final model based on selected variables
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Function to provide the final model based on the suggested variables
def final_model(chickens_data, import_quantity='Import Quantity', import_value='Imp
    # Ensure that the 'year' column is in datetime format
    chickens_data['year'] = pd.to_datetime(chickens_data['year'], format='%Y')

    # Convert 'year' column to numerical format (extracting the year as an integer)
```

```
chickens_data['year'] = chickens_data['year'].dt.year

# Filter the DataFrame based on 'Chickens' in 'animal_category'
filtered_import_quantity = chickens_data[(chickens_data['element'] == import_qu
filtered_import_value = chickens_data[(chickens_data['element'] == import_value

# Filter data for the past 15 years (from 1998 onwards)
past_15_years_import_quantity = filtered_import_quantity[filtered_import_quanti
past_15_years_import_value = filtered_import_value[filtered_import_value['year'

# Merge Import Quantity and Import Value data on 'country' and 'year'
merged_data = pd.merge(past_15_years_import_quantity, past_15_years_import_valu

# Select the top 10 countries based on frequency
top_10_countries = merged_data['country'].value_counts().index[:10]
merged_data = merged_data[merged_data['country'].isin(top_10_countries)]

# Rename 'value_quantity' to 'import_quantity' and 'value_value' to 'import_val
merged_data.rename(columns={'value_quantity': 'import_quantity', 'value_value':

# Feature Engineering: Add the most important columns to the dataset
merged_data['import_quantity_squared'] = merged_data['import_quantity'] ** 2
merged_data['import_quantity_square_root'] = np.sqrt(merged_data['import_quanti
merged_data['import_quantity_log'] = np.log1p(merged_data['import_quantity'])
merged_data['import_quantity_zscore'] = (merged_data['import_quantity'] - merge
merged_data['import_quantity_normalized'] = (merged_data['import_quantity'] - m

# Prepare features (independent variables) and target (dependent variable)
X = merged_data[['country', 'year', 'import_quantity', 'import_quantity_squared
y = merged_data['import_value']  # 'Import Value' as target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Preprocessing: Apply OneHotEncoding for categorical columns and scaling for n
categorical_features = ['country']
numerical_features = ['year', 'import_quantity', 'import_quantity_squared', 'im

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

# Define a Random Forest pipeline with preprocessing
rf_pipeline = Pipeline([('preprocessor', preprocessor), ('regressor', RandomFor

# Hyperparameter tuning for Random Forest
param_grid_rf = {
    'regressor__n_estimators': [100, 200, 300],
    'regressor__max_depth': [10, 15, 20],
    'regressor__min_samples_split': [2, 5, 10]
}

# Perform GridSearchCV for Random Forest
grid_rf = GridSearchCV(rf_pipeline, param_grid_rf, cv=5, scoring='neg_mean_squa
```

```
    grid_rf.fit(X_train, y_train)
    best_rf = grid_rf.best_estimator_

    # Make predictions with the best Random Forest model
    y_pred_best_rf = best_rf.predict(X_test)

    # Calculate and print evaluation metrics for the model
    mse_best_rf = mean_squared_error(y_test, y_pred_best_rf)
    r2_best_rf = r2_score(y_test, y_pred_best_rf)
    print(f"Final Model - MSE: {mse_best_rf:.4f}, R-squared: {r2_best_rf:.4f}")

    return best_rf

# Example of calling the function:
# Assuming chickens_data is your DataFrame containing the data for 'Chickens' impor
best_model = final_model(chickens_data, import_quantity='Import Quantity', import_v
```

```
Final Model - MSE: 21183281.2200, R-squared: 0.9695
```

In [ ]:

Final model with feature importance analysis

In [ ]:

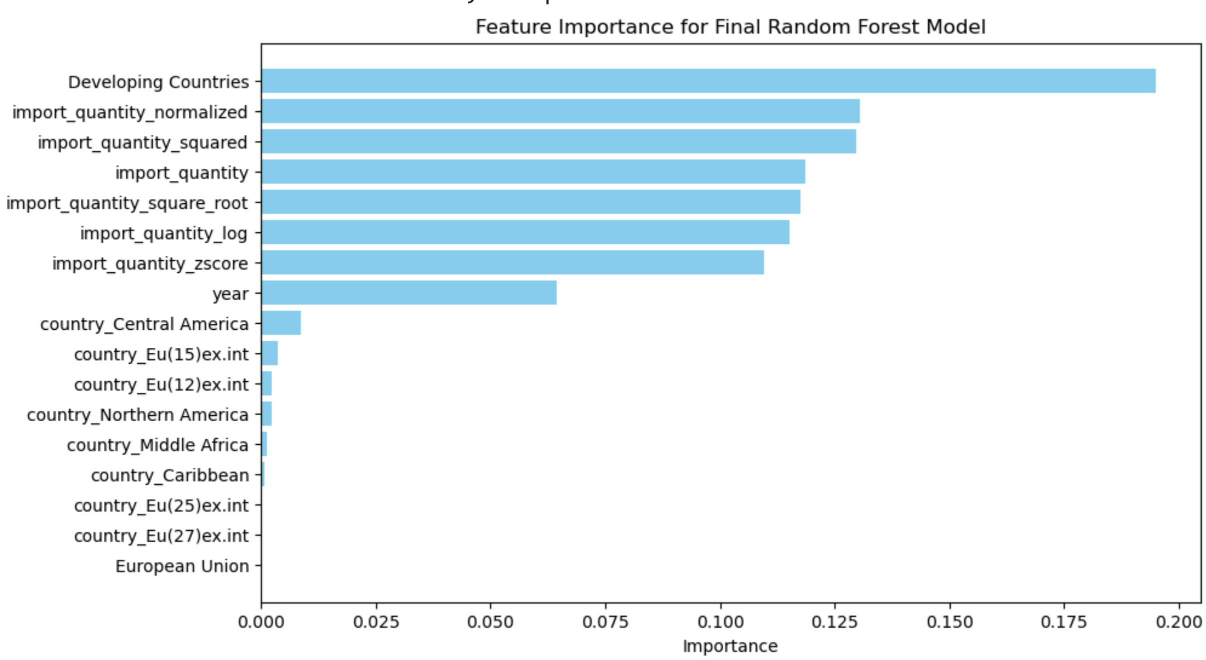
In [85]:

```
# Final model with feature importance analysis

# Final model with feature importance analysis
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Function to provide the final model and feature importances
def final_model(chickens_data, import_quantity='Import Quantity', import_value='Imp
    # Ensure that the 'year' column is in datetime format
    chickens_data['year'] = pd.to_datetime(chickens_data['year'], format='%Y')

    # Convert 'year' column to numerical format (extracting the year as an integer)
    chickens_data['year'] = chickens_data['year'].dt.year

    # Filter the DataFrame based on 'Chickens' in 'animal_category'
    filtered_import_quantity = chickens_data[(chickens_data['element'] == import_qu
    filtered_import_value = chickens_data[(chickens_data['element'] == import_value

    # Filter data for the past 15 years (from 1998 onwards)
    past_15_years_import_quantity = filtered_import_quantity[filtered_import_quanti
```

```
past_15_years_import_value = filtered_import_value[filtered_import_value['year'

# Merge Import Quantity and Import Value data on 'country' and 'year'
merged_data = pd.merge(past_15_years_import_quantity, past_15_years_import_valu

# Select the top 10 countries based on frequency
top_10_countries = merged_data['country'].value_counts().index[:10]
merged_data = merged_data[merged_data['country'].isin(top_10_countries)]

# Rename 'value_quantity' to 'import_quantity' and 'value_value' to 'import_val
merged_data.rename(columns={'value_quantity': 'import_quantity', 'value_value':

# Feature Engineering: Add the most important columns to the dataset
merged_data['import_quantity_squared'] = merged_data['import_quantity'] ** 2
merged_data['import_quantity_square_root'] = np.sqrt(merged_data['import_quanti
merged_data['import_quantity_log'] = np.log1p(merged_data['import_quantity'])
merged_data['import_quantity_zscore'] = (merged_data['import_quantity'] - merge
merged_data['import_quantity_normalized'] = (merged_data['import_quantity'] - m

# Prepare features (independent variables) and target (dependent variable)
X = merged_data[['country', 'year', 'import_quantity', 'import_quantity_squared
y = merged_data['import_value']  # 'Import Value' as target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Preprocessing: Apply OneHotEncoding for categorical columns and scaling for n
categorical_features = ['country']
numerical_features = ['year', 'import_quantity', 'import_quantity_squared', 'im

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

# Define a Random Forest pipeline with preprocessing
rf_pipeline = Pipeline([('preprocessor', preprocessor), ('regressor', RandomFor

# Hyperparameter tuning for Random Forest
param_grid_rf = {
    'regressor__n_estimators': [100, 200, 300],
    'regressor__max_depth': [10, 15, 20],
    'regressor__min_samples_split': [2, 5, 10]
}

# Perform GridSearchCV for Random Forest
grid_rf = GridSearchCV(rf_pipeline, param_grid_rf, cv=5, scoring='neg_mean_squa
grid_rf.fit(X_train, y_train)
best_rf = grid_rf.best_estimator_

# Make predictions with the best Random Forest model
y_pred_best_rf = best_rf.predict(X_test)

# Calculate and print evaluation metrics for the model
mse_best_rf = mean_squared_error(y_test, y_pred_best_rf)
```

```
    r2_best_rf = r2_score(y_test, y_pred_best_rf)
    print(f"Final Model - MSE: {mse_best_rf:.4f}, R-squared: {r2_best_rf:.4f}")

    # Extract feature importances from the best Random Forest model
    rf_regressor = best_rf.named_steps['regressor']
    feature_names = numerical_features + list(best_rf.named_steps['preprocessor'].n

    # Rename long feature names for better display
    feature_names = np.array([name.replace('country_Net Food Importing Developing C
                                  .replace('country_European Union (exc Intra-tr

    importances = rf_regressor.feature_importances_

    # Create a DataFrame to display feature importance
    feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': i
    feature_importance_df = feature_importance_df.sort_values(by='Importance', asce

    # Plot the feature importances
    plt.figure(figsize=(10, 6))
    plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'],
    plt.xlabel('Importance')
    plt.title('Feature Importance for Final Random Forest Model')
    plt.gca().invert_yaxis()  # Invert y-axis for better visualization
    plt.show()

    return best_rf, feature_importance_df

# Example of calling the function:
# Assuming chickens_data is your DataFrame containing the data for 'Chickens' impor
best_model, feature_importances = final_model(chickens_data, import_quantity='Impor
print(feature_importances)
```

```
Final Model - MSE: 21183281.2200, R-squared: 0.9695
```



```
                       Feature  Importance
15         Developing Countries    0.195047
6    import_quantity_normalized    0.130589
2       import_quantity_squared    0.129787
1               import_quantity    0.118534
3   import_quantity_square_root    0.117570
4           import_quantity_log    0.115226
5        import_quantity_zscore    0.109571
0                          year    0.064446
8       country_Central America    0.008789
10         country_Eu(15)ex.int    0.003633
9          country_Eu(12)ex.int    0.002201
16     country_Northern America    0.002197
14        country_Middle Africa    0.001328
7             country_Caribbean    0.000627
11         country_Eu(25)ex.int    0.000225
12         country_Eu(27)ex.int    0.000154
13               European Union    0.000077
```

Interpretation of Feature Importance Results:

The final model for predicting import value as a function of various independent variables (import quantity, transformations, and country-specific dummy variables) is structured based on the feature importances derived from the Random Forest model.

Key Features and Their Importances: Developing Countries (0.1950): This is the most important feature in predicting import value, indicating that whether a country falls under the "Developing Countries" category significantly influences import value predictions. Import Quantity Normalized (0.1306): This feature is the second most important, showing that scaling the import quantity helps to improve prediction accuracy. Import Quantity Squared (0.1298): Squaring the import quantity reveals non-linear relationships between quantity and value, making it a key predictor. Import Quantity (0.1185): The raw import quantity also has a strong influence, indicating that higher import quantities generally lead to higher import values. Import Quantity Square Root (0.1176): The square root transformation captures additional non-linearities, further improving prediction accuracy. Import Quantity Log (0.1152): The logarithmic transformation helps handle skewed data, contributing substantially to predicting the import value. Import Quantity Z-Score (0.1096): The z-score, which standardizes the import quantity, plays a significant role, highlighting the importance of relative deviations from the mean in predicting import values. Year (0.0644): Time trends do matter but play a smaller role compared to quantity-based features, showing that temporal effects are less critical in predicting import values.

Lesser Influential Features:

Country-Specific Variables: Regional features like:

Central America (0.0088) EU(15) (0.0036) EU(12) (0.0022) Northern America (0.0022) all contribute to the prediction but have relatively minor effects compared to quantity-related features. Developing Countries (0.1950): Among country-specific features, being classified as a developing country has the most significant impact on the model.

Other Minor Features:

Middle Africa (0.0013), Caribbean (0.0006), EU(25) (0.0002), EU(27) (0.0002), and European Union (0.00008) have minimal effects, indicating that imports from these regions play a much smaller role in the overall prediction. Explanation: Quantity-Related Features: The primary drivers of the model are import quantities and their various transformations (normalized, squared, square root, log, z-score). These features capture different dimensions of import quantities, from raw data to transformations that account for non-linear relationships and scaling.

Developing Countries: Developing countries have a significant impact on import values, suggesting that the status of a country plays a crucial role in determining how import values fluctuate.

Country-Specific Variables: Although regional factors play a smaller role compared to the import quantity transformations, Central America and Northern America have noticeable effects. Other countries contribute much less to the predictions.

Time Trends: The year variable indicates that temporal trends, while important, are secondary compared to the direct measures of import quantities.

Visualization of actual vs. predicted import values for Northern America (2000-2013)

In [ ]:

```

```

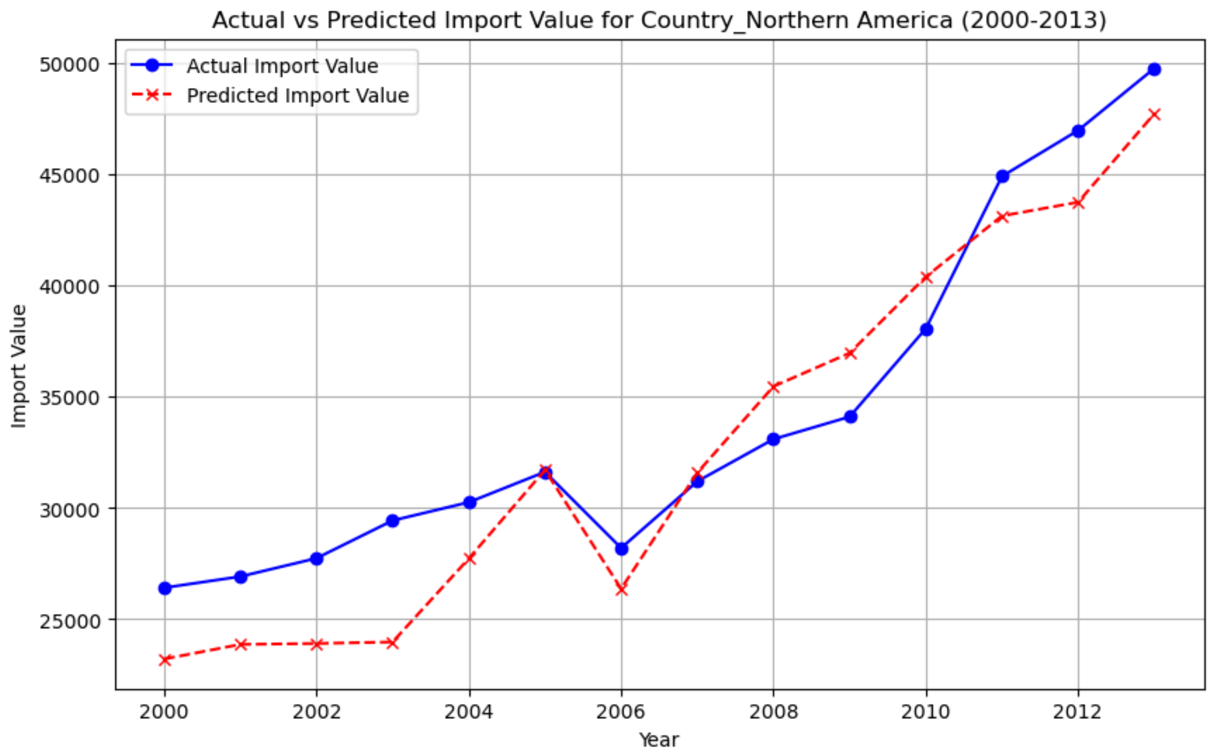
In [92]:

```
# This code provides the actual and predicted import values for "Country_Northern A

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Function to visualize and provide a table for actual vs. predicted import values
def visualize_northern_america_imports(chickens_data, best_rf_model, import_quantit
    # Ensure that the 'year' column is in datetime format
    chickens_data['year'] = pd.to_datetime(chickens_data['year'], format='%Y')
    chickens_data['year'] = chickens_data['year'].dt.year

    # Filter the data for 'Chickens' in 'animal_category'
    filtered_import_quantity = chickens_data[(chickens_data['element'] == import_qu
```

```python
filtered_import_value = chickens_data[(chickens_data['element'] == import_value

# Filter data for 1998 onwards
past_15_years_import_quantity = filtered_import_quantity[filtered_import_quanti
past_15_years_import_value = filtered_import_value[filtered_import_value['year'

# Merge Import Quantity and Import Value data
merged_data = pd.merge(past_15_years_import_quantity, past_15_years_import_valu
merged_data.rename(columns={'value_quantity': 'import_quantity', 'value_value':

# Filter for Northern America and 2000-2013
northern_america_data = merged_data[(merged_data['country'] == 'Northern Americ

# Feature engineering
northern_america_data['import_quantity_squared'] = northern_america_data['impor
northern_america_data['import_quantity_square_root'] = np.sqrt(northern_america
northern_america_data['import_quantity_log'] = np.log1p(northern_america_data['
northern_america_data['import_quantity_zscore'] = (northern_america_data['impor
northern_america_data['import_quantity_normalized'] = (northern_america_data['i

# Prepare features and target
X_northern_america = northern_america_data[['country', 'year', 'import_quantity
y_northern_america = northern_america_data['import_value']  # Actual import val

# Preprocessing
categorical_features = ['country']
numerical_features = ['year', 'import_quantity_squared', 'import_quantity', 'im

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

# Apply preprocessing
X_northern_america_transformed = best_rf_model.named_steps['preprocessor'].tran

# Predict using the Random Forest model
y_pred_northern_america = best_rf_model.named_steps['regressor'].predict(X_nort

# Create a DataFrame to display actual vs predicted import values
results_df = northern_america_data[['year', 'import_value']].copy()
results_df['predicted_import_value'] = y_pred_northern_america

# Display the results in tabular format
print(results_df)

# Plot the Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.plot(northern_america_data['year'], y_northern_america, label='Actual Impor
plt.plot(northern_america_data['year'], y_pred_northern_america, label='Predict
plt.xlabel('Year')
plt.ylabel('Import Value')
plt.title('Actual vs Predicted Import Value for Country_Northern America (2000-
plt.legend()
plt.grid(True)
```

```
    plt.show()

# Example of calling the function:
# Assuming chickens_data is your DataFrame containing the data for 'Chickens' impor
visualize_northern_america_imports(chickens_data, best_model, import_quantity='Impo
```

	year	import_value	predicted_import_value
1238	2000	26416.0	23210.453000
1239	2001	26920.0	23867.094532
1240	2002	27743.0	23910.305365
1241	2003	29441.0	23977.560629
1242	2004	30261.0	27729.206056
1243	2005	31627.0	31713.901333
1244	2006	28204.0	26371.386056
1245	2007	31203.0	31590.789667
1246	2008	33100.0	35474.114000
1247	2009	34108.0	36973.581708
1248	2010	38075.0	40402.830444
1249	2011	44923.0	43143.012361
1250	2012	46989.0	43759.780125
1251	2013	49780.0	47738.416667




```
In [ ]:
```