Overall Food Consumer Price Index and Predictions from Different Food Category Consumer Price Indexes

Project Goal

The objective of this project is to develop predictive models using **Linear Regression**, **Stepwise Regression**, and **XGBoost Regression** to estimate the contribution of various food category **Consumer Price Indexes (CPI)** to the overall **Food CPI**.

Dataset Description

The **All-Items Consumer Price Index (CPI)** measures the average change over time in the prices paid by urban consumers for a representative basket of goods and services. The **CPI for Food** specifically tracks the changes in retail food prices.

The **U.S. Department of Agriculture's Economic Research Service (USDA ERS)** monitors trends in the Food CPI and generates forecasts based on these patterns. Despite the complexities of forecasting, which are influenced by shifts in the food and agricultural economies, the USDA's forecasts offer valuable insights to farmers, processors, wholesalers, consumers, and policymakers.

Dataset Metadata

The dataset consists of **3 features**: (Year, Category, Percent Change), with a total of **1,100 rows**. The data represents the annual **CPI percent change** for various food categories from **1974 to 2023**.

Dataset Citation

Source:
U.S. Department of Agriculture, Economic Research Service. (2024). *Annual percent changes in selected Consumer Price Indexes, 1974–2023*. Retrieved from https://www.ers.usda.gov/data-products/food-price-outlook

In [1]:
```
!pip install seaborn
!pip install statsmodels
!pip install scikit-learn
!pip install missingno
!pip install xgboost
!pip install graphviz
```

```
Requirement already satisfied: seaborn in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packag
es (0.13.2)
Requirement already satisfied: pandas>=1.2 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-pa
ckages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\haoal\appdata\local\programs\python\python310
\lib\site-packages (from seaborn) (3.8.4)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\haoal\appdata\local\programs\python\python310\li
b\site-packages (from seaborn) (1.23.5)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\haoal\appdata\local\programs\python\python310\lib\s
ite-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.51.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\haoal\appdata\local\programs\python\python310\li
b\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pillow>=8 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-pack
ages (from matplotlib!=3.6.1,>=3.4->seaborn) (9.5.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\haoal\appdata\local\programs\python\python310\lib\si
te-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.2.1)
Requirement already satisfied: packaging>=20.0 in c:\users\haoal\appdata\local\programs\python\python310\lib\sit
e-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.0)
Requirement already satisfied: cycler>=0.10 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-p
ackages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\haoal\appdata\local\programs\python\python310\lib\s
ite-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\haoal\appdata\local\programs\python\python310\lib\si
te-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.1.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-p
ackages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\haoal\appdata\local\programs\python\python310\lib\site
-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: six>=1.5 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packa
ges (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
[notice] A new release of pip available: 22.2.1 -> 25.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Requirement already satisfied: statsmodels in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (0.14.4)
Requirement already satisfied: packaging>=21.3 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from statsmodels) (24.0)
Requirement already satisfied: numpy<3,>=1.22.3 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from statsmodels) (1.23.5)
Requirement already satisfied: patsy>=0.5.6 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from statsmodels) (1.0.1)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from statsmodels) (2.2.2)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from statsmodels) (1.13.0)
Requirement already satisfied: tzdata>=2022.7 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: six>=1.5 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4->statsmodels) (1.16.0)
[notice] A new release of pip available: 22.2.1 -> 25.1
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: scikit-learn in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (1.6.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: numpy>=1.19.5 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.6.0 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from scikit-learn) (1.13.0)
[notice] A new release of pip available: 22.2.1 -> 25.1
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: missingno in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (0.5.2)
Requirement already satisfied: seaborn in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from missingno) (0.13.2)
Requirement already satisfied: scipy in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from missingno) (1.13.0)
Requirement already satisfied: numpy in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from missingno) (1.23.5)
Requirement already satisfied: matplotlib in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from missingno) (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->missingno) (1.2.1)
Requirement already satisfied: cycler>=0.10 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->missingno) (0.12.1)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->missingno) (2.9.0.post0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->missingno) (3.1.2)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->missingno) (1.4.5)
Requirement already satisfied: packaging>=20.0 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->missingno) (24.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->missingno) (4.51.0)
Requirement already satisfied: pillow>=8 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->missingno) (9.5.0)
Requirement already satisfied: pandas>=1.2 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from seaborn->missingno) (2.2.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from pandas>=1.2->seaborn->missingno) (2024.1)
Requirement already satisfied: pytz>=2020.1 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from pandas>=1.2->seaborn->missingno) (2024.1)
Requirement already satisfied: six>=1.5 in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from python-dateutil>=2.7->matplotlib->missingno) (1.16.0)
[notice] A new release of pip available: 22.2.1 -> 25.1
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: xgboost in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (3.0.0)
Requirement already satisfied: numpy in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from xgboost) (1.23.5)
Requirement already satisfied: scipy in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packages (from xgboost) (1.13.0)
[notice] A new release of pip available: 22.2.1 -> 25.1
[notice] To update, run: python.exe -m pip install --upgrade pip

[notice] A new release of pip available: 22.2.1 -> 25.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Requirement already satisfied: graphviz in c:\users\haoal\appdata\local\programs\python\python310\lib\site-packa
ges (0.20.3)
```

In [2]:
```
import scipy as sp
import scipy.stats as stats
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import copy
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import statsmodels.formula.api as smf
%matplotlib inline
import missingno as msno
from scipy.stats import zscore
import xgboost as xgb
from xgboost import plot_tree
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import cross_val_score
```

In [3]:
```
#Load Dataset
cpi_data = pd.read_csv('./data/historicalcpi.csv')
```

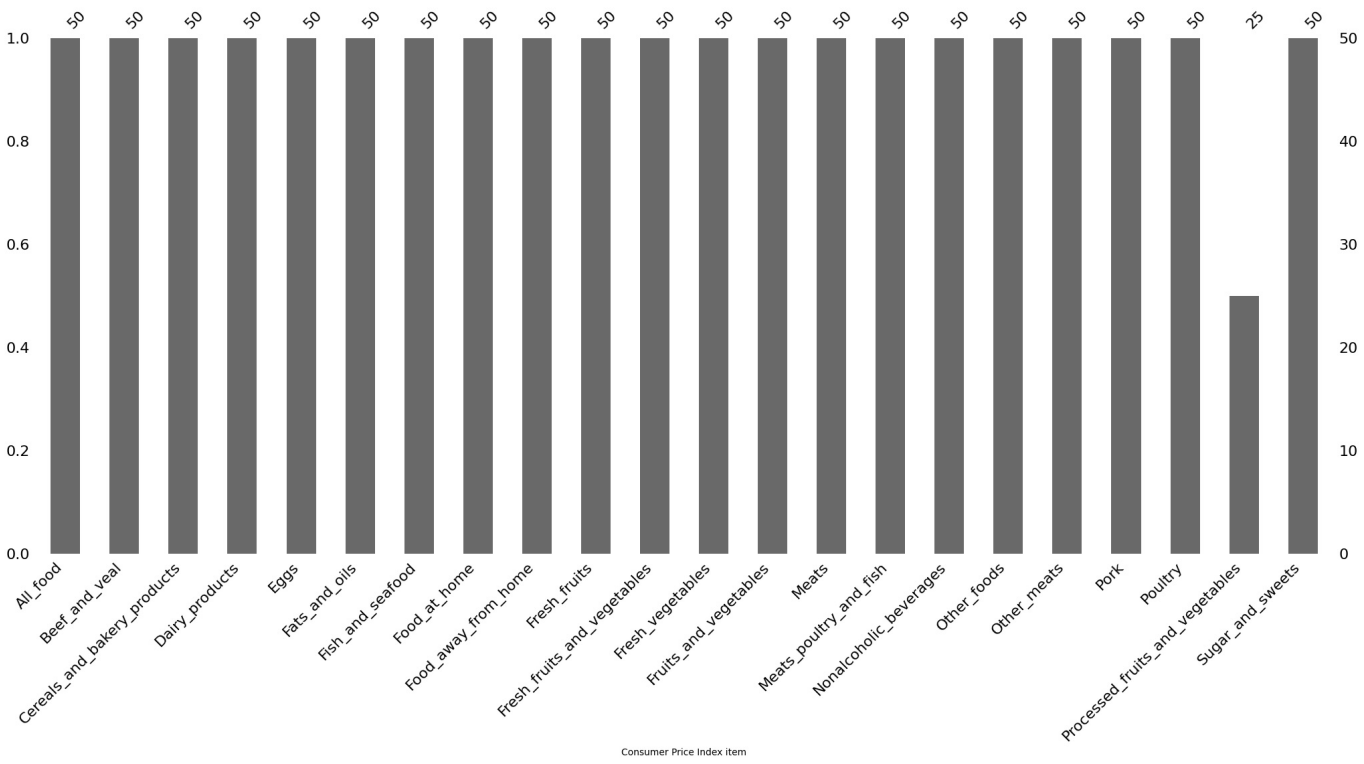
Dataset Cleaning Process

1. **Renamed Columns and Clarified Values:**
   - Replaced spaces in column names with underscores to ensure compatibility with formulas and avoid potential issues during analysis. This step improves the ease of referencing column names within the code and makes the dataset easier to handle programmatically.
2. **Converted Year and CPI Percent Change to Numeric Values:**
   - Ensured that the `Year` and `CPI percent change` columns were correctly formatted as numeric values. This is important because these columns contain numerical data and must be treated as such for modeling purposes. Any non-numeric data would cause errors during model training or analysis.
3. **Rearranged Categories According to Year:**
   - The original dataset had various food categories. To better analyze the effect of each category on the overall food CPI, I pivoted the data so that each food category became a separate column. This transformation is necessary for modeling the contribution of each category to the All Food CPI.
4. **Dropped Empty Columns:**
   - Removed any empty or irrelevant columns from the dataset to simplify the analysis and reduce noise. Keeping unnecessary columns could have introduced complications when performing analysis or building predictive models.
5. **Handled Missing Values:**
   - For columns with missing data (NaN), I filled in the missing values with the mean of that particular column. This imputation method helps maintain data integrity without introducing bias. Mean imputation is appropriate in this case because the dataset size is relatively small, and preserving as many rows as possible is important for model accuracy.
6. **Dropped "Food at Home" and "Food Away from Home" Categories:**
   - I excluded the "Food at Home" and "Food Away from Home" categories from the dataset, as these categories were not relevant to the project's goal. The focus of the analysis is on the effect of specific food categories on overall CPI, not on the location-based categorization of food spending.
7. **Final Check for Missing Data:**
   - After cleaning the dataset, I performed a final check to ensure there were no remaining columns with missing values. This step confirms that the dataset is ready for further analysis and modeling.

In [4]:
```
cpi_data_copy = cpi_data.copy()
cpi_data_copy = pd.DataFrame(cpi_data_copy)

#Cleaning CPI Dataset
#first rename and clarify the value
#The value is the percent of cpi change
# Replace spaces in column names with underscores, to prevent issues in formula later
cpi_data_copy.rename(columns={'Percent change': 'CPI_Percent_change'}, inplace=True)
```

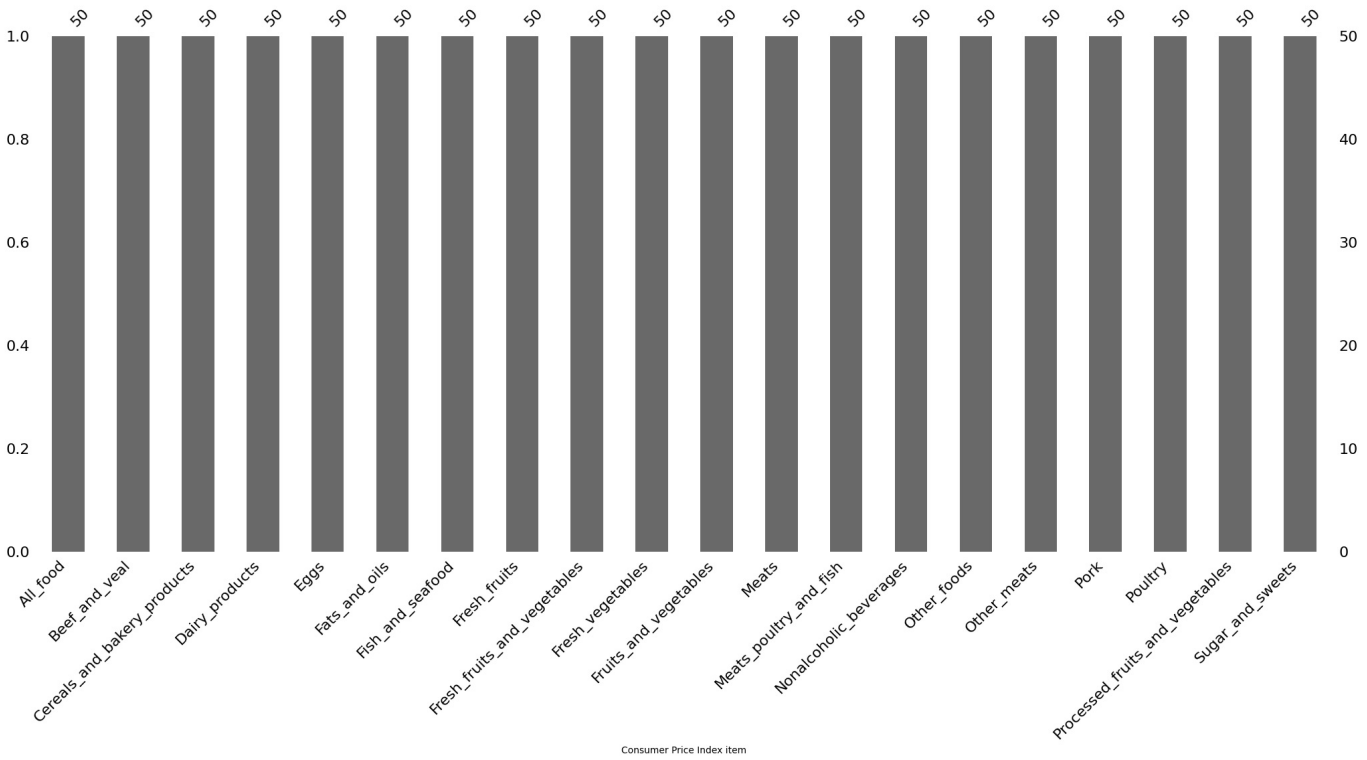
```
# make year and CPI percent change into numeric value
cpi_data_copy['CPI_Percent_change'] = pd.to_numeric(cpi_data_copy['CPI_Percent_change'], errors='coerce')
cpi_data_copy['Year'] = pd.to_numeric(cpi_data['Year'], errors='coerce')

#Rearrange category according to year
#The original data set has various categories. In order to calsulate the effect of each category on All Food CPI
cpi_data_copy = cpi_data_copy.pivot(index='Year', columns='Consumer Price Index item', values='CPI_Percent_chang

cpi_data_copy.columns = cpi_data_copy.columns.str.replace(r'[^\w\s]', '', regex=True)
cpi_data_copy.columns = cpi_data_copy.columns.str.replace(' ', '_')

missing_values = cpi_data_copy.isna().sum()
columns_with_missing = missing_values[missing_values > 0]
print("\nBefore Cleaning")
print("\nColumns with missing values:")
print(columns_with_missing)
#display which colum is missing data
msno.bar(cpi_data_copy)
plt.show()
#First Drop any empty cloumn
#If there are any missing value, fill NaN values with column mean
cpi_data_copy = cpi_data_copy.dropna(axis=1, how='all')
cpi_data_copy = cpi_data_copy.fillna(cpi_data_copy.mean())


#Drop Food at home and Food away from home, as my scope is the effect of different food categories affct the ove
cpi_data_copy.drop('Food_at_home', axis=1, inplace=True)
cpi_data_copy.drop('Food_away_from_home', axis=1, inplace=True)

# Check for missing values in the entire DataFrame
missing_values = cpi_data_copy.isna().sum()

# Check again if there are any columns with missing values
columns_with_missing = missing_values[missing_values > 0]
print("\nAfter Cleaning")
print("\nColumns with missing values:")
print(columns_with_missing)
#display which colum is missing data
msno.bar(cpi_data_copy)
plt.show()



#Create a copy for other models
cpi_foward_step = cpi_data_copy
cpi_XGB = cpi_data_copy

cpi_data_copy.head()
```

```
Before Cleaning

Columns with missing values:
Consumer Price Index item
Processed_fruits_and_vegetables    25
dtype: int64
```



After Cleaning

Columns with missing values:
Series([], dtype: int64)



Out[4]:

Consumer Price Index item / Year	All_food	Beef_and_veal	Cereals_and_bakery_products	Dairy_products	Eggs	Fats_and_oils	Fish_and_seafood	Fresh_
1974	14.3	2.9	29.9	18.6	0.4	41.9	15.3	
1975	8.5	1.0	11.3	3.1	-1.8	10.7	8.5	
1976	3.0	-3.2	-2.2	8.1	9.2	-12.5	11.7	
1977	6.3	-0.7	1.6	2.7	-3.2	10.1	10.8	
1978	9.9	22.9	9.0	6.8	-5.4	9.6	9.4	

Handling Outliers in CPI Data

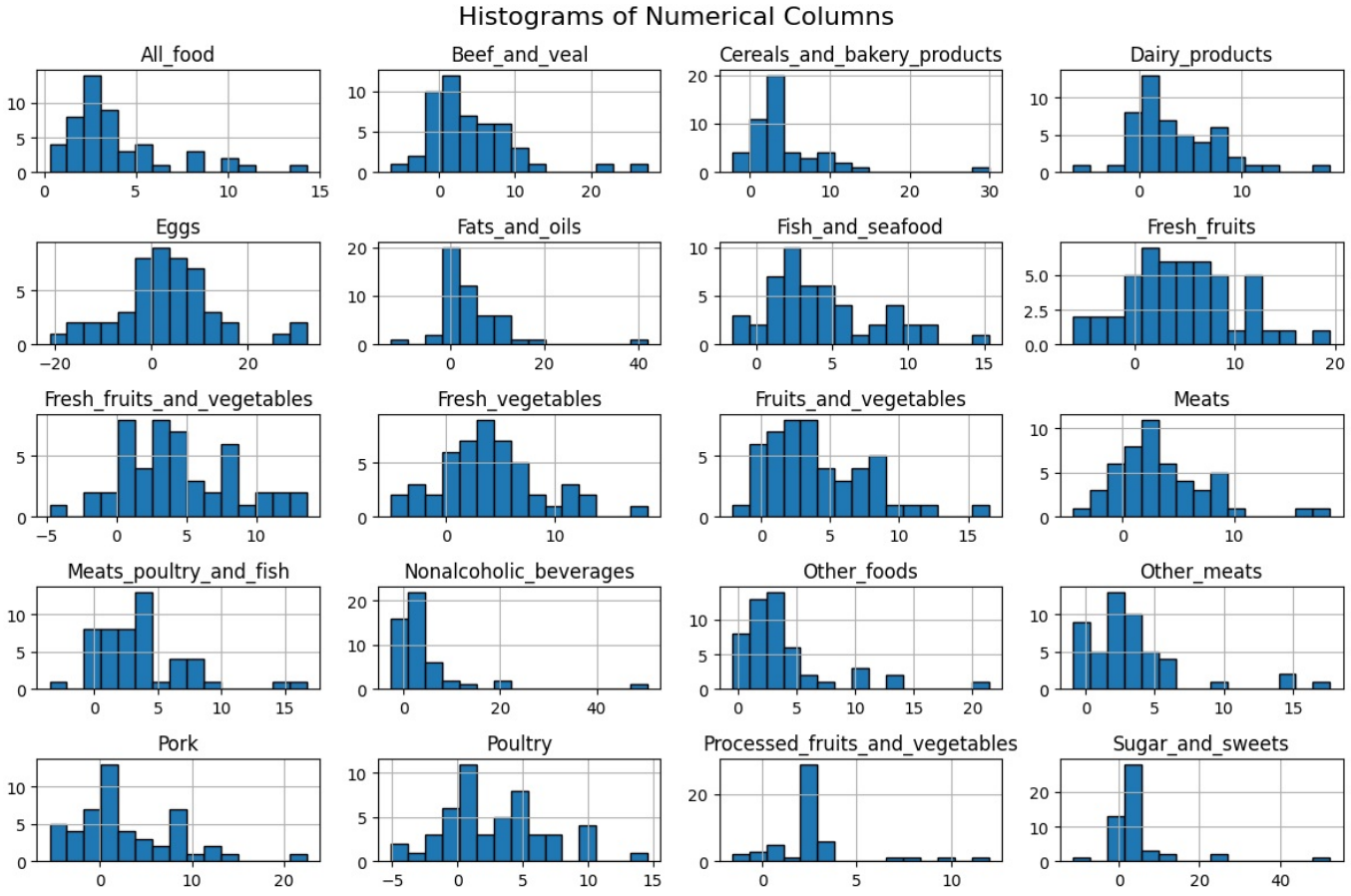
- The **CPI change percentages** reflect significant economic events, such as inflation spikes or deflation periods.
- Outliers in this context are not merely noise; they are important signals that may indicate rare but impactful events.
- Removing or replacing these outliers could obscure critical trends in the data, which are essential for understanding the behavior of the CPI.
- Therefore, the decision was made to **preserve outliers** in the dataset, rather than removing or altering them.
- To effectively account for outliers, **robust regression models** will be used. These models are designed to minimize the influence of extreme values without distorting the overall trends in the data.

Univariate Analysis

Summary of Histogram Analysis:

- **Target Variable (All_food)**: The distribution of the target variable, *All_food*, is strongly right-skewed. This indicates that most of the data points are concentrated at the lower end, with a few extreme high values (outliers) that could represent significant shifts or events.
- **Independent Variables (Factors)**: The independent variables (factors) exhibit a mix of distributions:
  - Some factors are right-skewed, similar to the target variable, suggesting the presence of extreme high values or outliers in these features.
  - Other factors follow a more normal distribution, with data points closely packed around the center, indicating less variability and more consistency within those variables.

In [21]:
```
# Univariate Analysis
# Plotting histograms for all numerical columns
cpi_data_copy.hist(figsize=(12, 8), bins=15, edgecolor='black')
plt.suptitle('Histograms of Numerical Columns', fontsize=16)
plt.tight_layout()
plt.show()
```

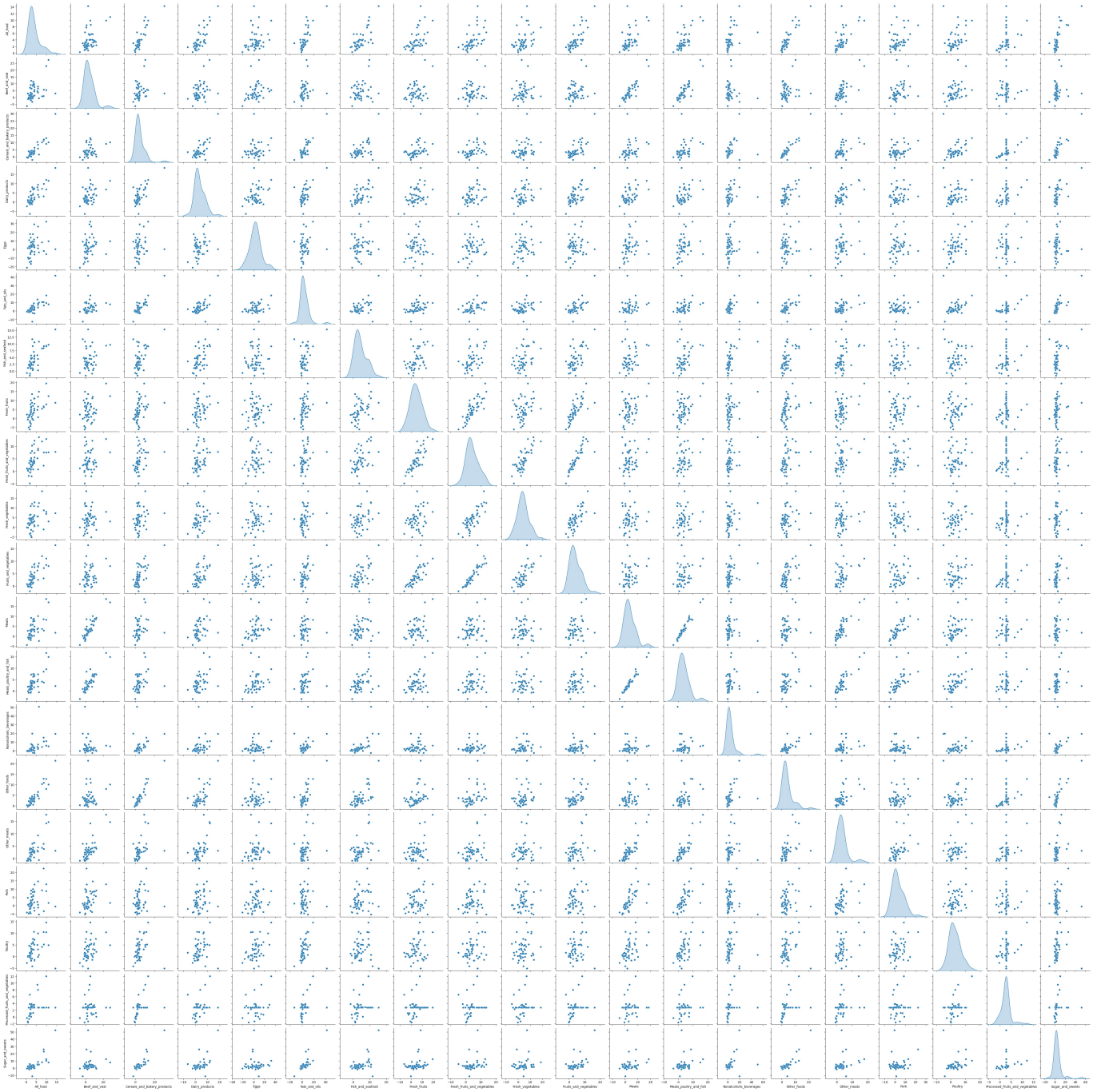


Pair Plot Analysis

Pair Plot Observations:

- Some of the scatter plots in the pair plot form almost vertical lines, indicating a high correlation between the variables. This suggests that these variables have little variance or are highly dependent on one another.
- Conversely, a few scatter plots form a more spread-out cloud, indicating a weaker correlation or suggesting that these variables are more independent.
- The vertical line pattern might also indicate the presence of categorical variables with a limited number of distinct values, leading to a constrained variation in the data.|

In [109…
```
# Pair Plot Analysis
sns.pairplot(cpi_data_copy, diag_kind='kde')
plt.savefig('pair_plot.png', dpi=300, bbox_inches='tight')
plt.show()
```

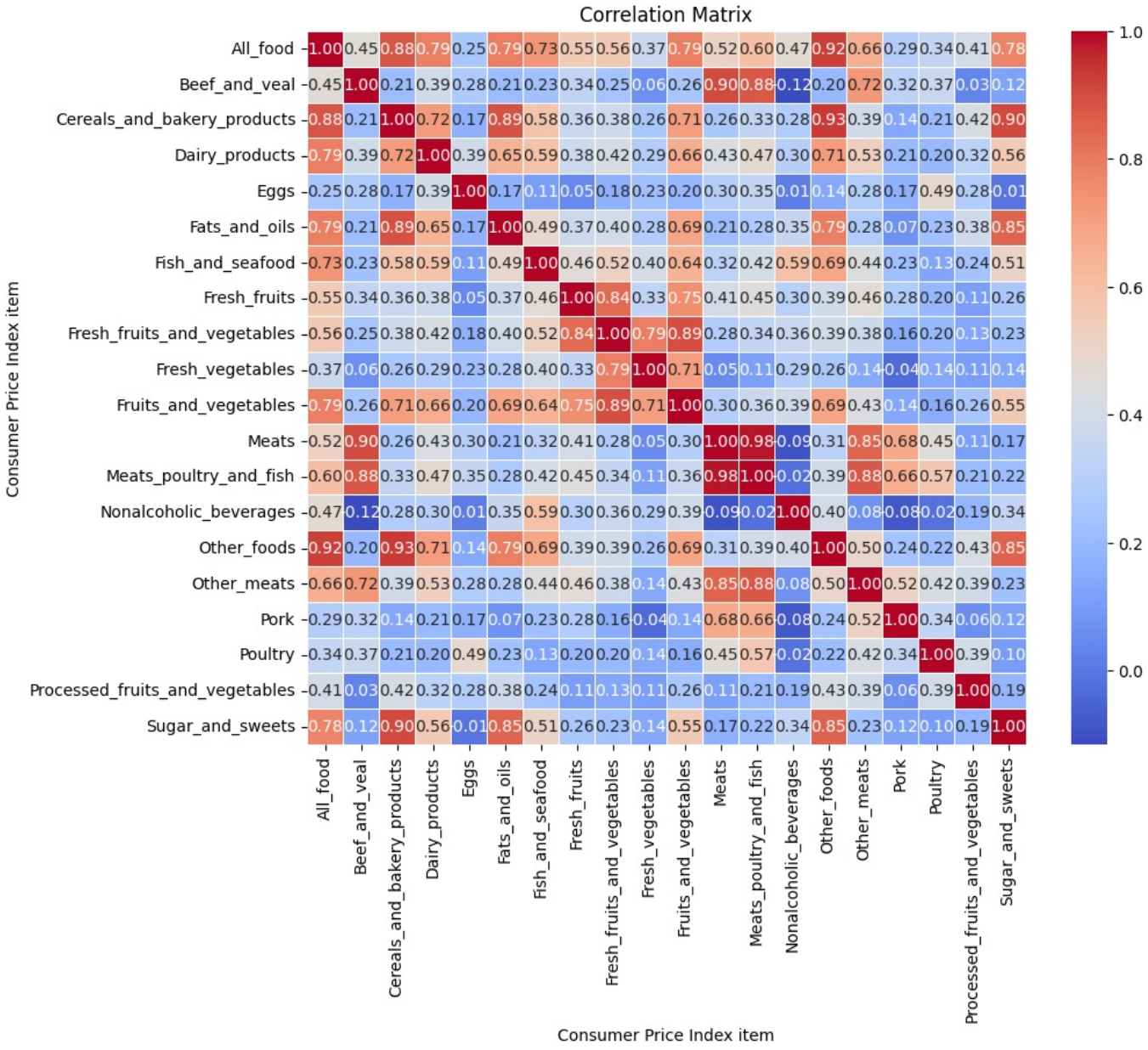


Correlation Matrix

Observations:

- The majority of the heatmap shows low correlations between variables, suggesting that the factors are largely independent.
- However, a few pairs of variables exhibit very high correlations (90% to 95%), indicating strong linear relationships between them.
- These high correlations could imply redundancy among features and may lead to multicollinearity, which could negatively impact model performance.
- Further investigation is needed to determine whether to combine or remove these highly correlated variables to improve model robustness.

In [22]:
```
# Correlation Matrix
correlation_matrix = cpi_data_copy.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



Exploratory Data Visualization

Histograms:

- The histogram matrix revealed that the target variable **"All_food"** is strongly right-skewed, indicating more observations with lower CPI changes and fewer extreme increases.
- Several of the factor variables exhibit a mix of right-skewed and near-normal distributions, with many factors showing a closely packed concentration of values.

Pair Plot:

- The pair plot showed a few highly linear relationships between certain variables.
- The diagonal plots (representing the distribution of individual variables) indicated that some factors had limited variation, with some forming almost vertical lines.
- This suggests little spread in these factors, potentially due to inherent grouping or data constraints.

Correlation Matrix:

- The correlation matrix indicated that most variables exhibit low correlations with each other, suggesting relative independence.
- However, a few variable pairs showed very high correlations (90–95%), indicating potential redundancy and a risk of multicollinearity.
- These highly correlated variables may need further investigation to avoid multicollinearity in predictive models.

Identifying Multicollinearity Using VIF

To detect multicollinearity among the features, we calculate the **Variance Inflation Factor (VIF)** for each independent variable.

- **VIF** measures how much the variance of a regression coefficient is inflated due to multicollinearity with other predictors.

- A **high VIF value** (typically **greater than 5 or 10**) indicates that the feature is highly correlated with other features and may cause instability in model estimation.

Why Use VIF?

- High multicollinearity can make model coefficients unstable and unreliable.
- Features with very high VIF values may either be removed or combined with others to improve model interpretability and reduce redundancy.

Approach:

1. Calculate VIF for each feature.
2. Identify features with VIF values above the acceptable threshold (e.g., VIF > 5 or 10).
3. Consider dropping or transforming features with high VIF to reduce multicollinearity and enhance model robustness.

In [24]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Compute VIF for each feature, to see if values are correlated to each other too much
X = cpi_data_copy.drop(columns='All_food')  # Exclude target
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns))]

# Print features with high VIF
print(vif_data[vif_data["VIF"] > 5])
```

```
                          feature          VIF
0                     Beef_and_veal  1081.869913
1       Cereals_and_bakery_products    76.201478
2                    Dairy_products     7.114829
4                     Fats_and_oils    10.983501
5                   Fish_and_seafood    25.222662
6                      Fresh_fruits   715.521096
7       Fresh_fruits_and_vegetables  2028.035715
8                Fresh_vegetables   529.600084
9             Fruits_and_vegetables    89.795680
10                           Meats  3960.943683
11          Meats_poultry_and_fish   918.969207
13                     Other_foods    37.823607
14                     Other_meats   127.452944
15                            Pork   317.506432
16                         Poultry    21.656056
17  Processed_fruits_and_vegetables     6.289335
18                Sugar_and_sweets    22.501111
```

Reducing Multicollinearity Based on VIF Analysis

The VIF analysis revealed that many food category features are highly correlated with each other.
This introduced redundancy and can make it difficult to determine the individual effect of each predictor on the target variable.

To improve model performance and interpretability:

- Features with **VIF values larger than 100** were considered excessively collinear.
- These features were dropped to reduce multicollinearity, leaving only the major representative categories.
- An exception was made for the **"Meat"** category, which was intentionally kept due to its importance as a major food group, even if its VIF was high.

This step ensures that the model remains both statistically sound and practically meaningful.

In [25]:
```
# Set a threshold for VIF above which variables will be dropped
threshold = 100

high_vif_columns = vif_data[vif_data['VIF'] > threshold]['feature'].tolist()
high_vif_columns = [col for col in high_vif_columns if col != 'Meats']
cpi_data_copy.drop(columns=high_vif_columns, inplace=True)

print("Columns removed due to high VIF:")
print(high_vif_columns)
```

```
Columns removed due to high VIF:
['Beef_and_veal', 'Fresh_fruits', 'Fresh_fruits_and_vegetables', 'Fresh_vegetables', 'Meats_poultry_and_fish', '
Other_meats', 'Pork']
```

Building a Basic OLS Regression Model

To establish a foundational model, I used **Ordinary Least Squares (OLS) Regression**.

OLS is a fundamental linear regression technique that minimizes the sum of squared residuals between the predicted and actual values.

The initial model helps to:

- Understand the baseline relationship between the food category CPIs and the overall Food CPI.
- Identify which features have statistically significant effects.
- Serve as a comparison point for more advanced modeling techniques later, such as Stepwise Regression and XGBoost.

No regularization or variable selection was applied in this basic model; all selected features (after VIF filtering) were included to observe their direct contribution to the prediction.

In [26]:
```
print(cpi_data_copy.columns)
# Formula for OLS regression
formula = 'All_food ~ Cereals_and_bakery_products + Dairy_products + Eggs + Fats_and_oils + Fish_and_seafood + F

# Fit the model using OLS (Ordinary Least Squares)
model = smf.ols(formula=formula, data=cpi_data_copy).fit()

# Print the summary of the regression
print(model.summary())
```

```
Index(['All_food', 'Cereals_and_bakery_products', 'Dairy_products', 'Eggs',
       'Fats_and_oils', 'Fish_and_seafood', 'Fruits_and_vegetables', 'Meats',
       'Nonalcoholic_beverages', 'Other_foods', 'Poultry',
       'Processed_fruits_and_vegetables', 'Sugar_and_sweets'],
      dtype='object', name='Consumer Price Index item')
                            OLS Regression Results
==============================================================================
Dep. Variable:               All_food   R-squared:                       0.980
Model:                            OLS   Adj. R-squared:                  0.974
Method:                 Least Squares   F-statistic:                     151.8
Date:                Mon, 28 Apr 2025   Prob (F-statistic):           1.16e-27
Time:                        10:34:31   Log-Likelihood:                -25.871
No. Observations:                  50   AIC:                             77.74
Df Residuals:                      37   BIC:                             102.6
Df Model:                          12
Covariance Type:            nonrobust
=================================================================================================
                                    coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------------
Intercept                         0.5760      0.139      4.141      0.000       0.294       0.858
Cereals_and_bakery_products       0.1507      0.077      1.960      0.058      -0.005       0.307
Dairy_products                    0.0558      0.029      1.956      0.058      -0.002       0.114
Eggs                             -0.0039      0.008     -0.478      0.636      -0.021       0.013
Fats_and_oils                    -0.0194      0.025     -0.768      0.448      -0.071       0.032
Fish_and_seafood                 -0.0268      0.033     -0.823      0.416      -0.093       0.039
Fruits_and_vegetables             0.1333      0.033      3.997      0.000       0.066       0.201
Meats                             0.1753      0.021      8.416      0.000       0.133       0.217
Nonalcoholic_beverages            0.0763      0.013      5.724      0.000       0.049       0.103
Other_foods                       0.2605      0.063      4.157      0.000       0.134       0.387
Poultry                           0.0471      0.023      2.074      0.045       0.001       0.093
Processed_fruits_and_vegetables  -0.0010      0.045     -0.023      0.981      -0.092       0.089
Sugar_and_sweets                  0.0076      0.029      0.264      0.793      -0.051       0.066
==============================================================================
Omnibus:                        8.172   Durbin-Watson:                   1.047
Prob(Omnibus):                  0.017   Jarque-Bera (JB):                7.298
Skew:                           0.794   Prob(JB):                       0.0260
Kurtosis:                       3.989   Cond. No.                         40.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Feature Engineering: Adding Interaction Terms

To enhance the model, I introduced **interaction terms** between food categories that are commonly connected.
Interaction terms capture the combined effect of two variables acting together, which may not be explained by their individual effects alone.

Adding these terms helps the model:

- Better reflect real-world economic relationships between categories (e.g., Meat and Dairy often move together).
- Capture hidden patterns and improve predictive performance.
- Allow more flexibility by modeling non-additive effects between factors.

Examples of Interaction Terms Added:

- **Cereals_and_bakery_products × Dairy_products**: Reflects possible co-movement between grains and dairy-based goods.

- **Fats_and_oils × Meats**: Models how fluctuations in fat/oil prices might impact meat-related products.

These interaction features were selected based on domain knowledge and preliminary correlation observations.

In [27]:
```
print(cpi_data_copy.columns)
# Formula for OLS regression
# Feature Engineering: adding interaction terms with category commonly connected
formula = 'All_food ~ Cereals_and_bakery_products * Dairy_products + Eggs + Fats_and_oils * Meats + Fish_and_sea

# Fit the model using OLS (Ordinary Least Squares)
model = smf.ols(formula=formula, data=cpi_data_copy).fit()

# Print the summary of the regression
print(model.summary())
```

```
Index(['All_food', 'Cereals_and_bakery_products', 'Dairy_products', 'Eggs',
       'Fats_and_oils', 'Fish_and_seafood', 'Fruits_and_vegetables', 'Meats',
       'Nonalcoholic_beverages', 'Other_foods', 'Poultry',
       'Processed_fruits_and_vegetables', 'Sugar_and_sweets'],
      dtype='object', name='Consumer Price Index item')
                            OLS Regression Results
==============================================================================
Dep. Variable:               All_food   R-squared:                       0.981
Model:                            OLS   Adj. R-squared:                  0.974
Method:                 Least Squares   F-statistic:                     132.5
Date:                Mon, 28 Apr 2025   Prob (F-statistic):           5.09e-26
Time:                        10:34:32   Log-Likelihood:                -24.078
No. Observations:                  50   AIC:                             78.16
Df Residuals:                      35   BIC:                             106.8
Df Model:                          14
Covariance Type:            nonrobust
================================================================================================================
                                                   coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------------------------
Intercept                                        0.6838      0.155      4.424      0.000       0.370       0.998
Cereals_and_bakery_products                      0.1574      0.083      1.906      0.065      -0.010       0.325
Dairy_products                                   0.0449      0.033      1.351      0.185      -0.023       0.112
Cereals_and_bakery_products:Dairy_products       0.0005      0.004      0.152      0.880      -0.007       0.008
Eggs                                            -0.0006      0.009     -0.063      0.950      -0.019       0.017
Fats_and_oils                                   -0.0383      0.030     -1.278      0.210      -0.099       0.023
Meats                                            0.1453      0.029      4.949      0.000       0.086       0.205
Fats_and_oils:Meats                              0.0061      0.004      1.574      0.124      -0.002       0.014
Fish_and_seafood                                -0.0285      0.034     -0.844      0.404      -0.097       0.040
Fruits_and_vegetables                            0.1414      0.034      4.116      0.000       0.072       0.211
Nonalcoholic_beverages                           0.0786      0.014      5.807      0.000       0.051       0.106
Other_foods                                      0.2331      0.065      3.586      0.001       0.101       0.365
Poultry                                          0.0381      0.029      1.295      0.204      -0.022       0.098
Processed_fruits_and_vegetables                 -0.0035      0.047     -0.075      0.940      -0.099       0.091
Sugar_and_sweets                                 0.0145      0.029      0.496      0.623      -0.045       0.074
==============================================================================
Omnibus:                        6.975   Durbin-Watson:                   1.169
Prob(Omnibus):                  0.031   Jarque-Bera (JB):                5.989
Skew:                           0.693   Prob(JB):                       0.0501
Kurtosis:                       3.977   Cond. No.                         222.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Identifying High Leverage and High Residual Points

During the regression diagnostic phase, I examined points with **high leverage** and **large residuals**.
High leverage points are observations that have an unusual combination of predictor values, while large residuals indicate predictions that significantly differ from actual outcomes.

Typically, high leverage and high residual points are candidates for closer investigation or potential removal, as they can disproportionately influence model performance.

Justification for Keeping These Points:

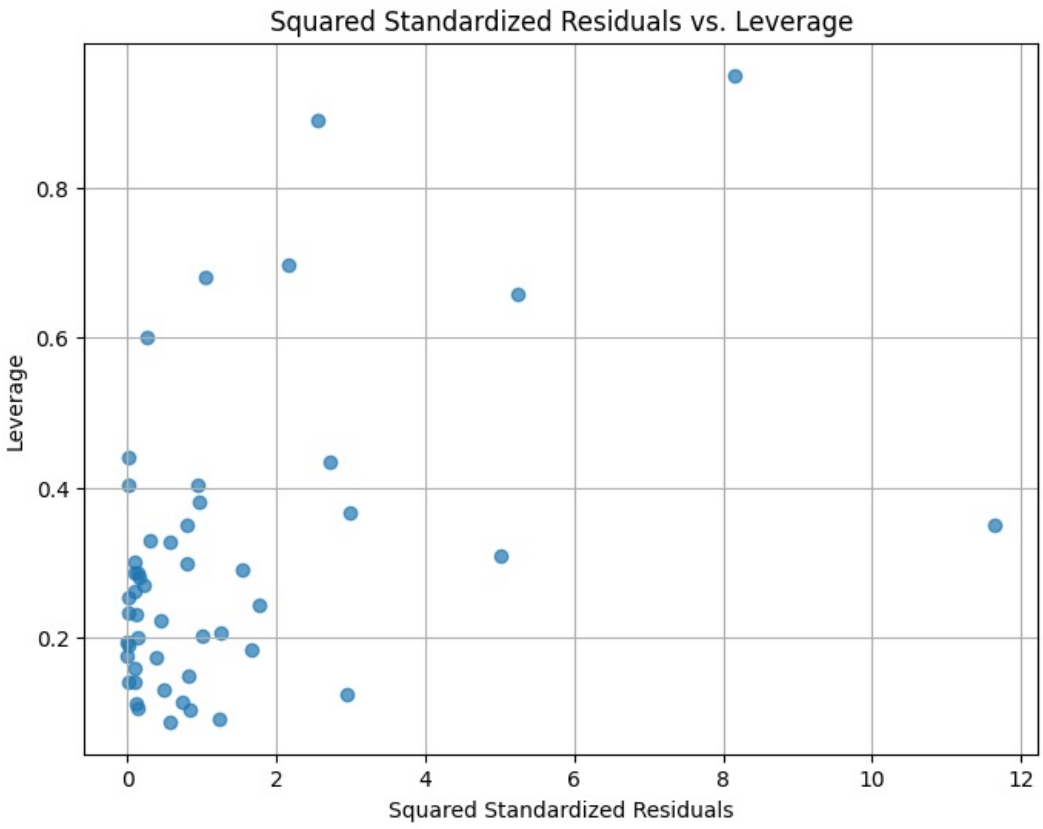
- **Small Sample Size**: The dataset is based on annual CPI data, meaning there are only a limited number of observations (one per year). Removing any points would further reduce an already small dataset, risking underfitting and loss of important historical context.
- **Nature of the Data**: CPI is a **processed economic index**, not raw survey data. It is constructed using standardized, rigorously vetted methodologies. Therefore, the influence of random error or noise is minimized, and outlier values often reflect genuine economic events (e.g., inflation spikes, recessions).
- **Preserving Economic Signals**: Outliers or leverage points in CPI data are meaningful and often correspond to significant historical economic shifts. Removing them would suppress important information necessary for accurate modeling and forecasting.

Thus, all points — including those with high leverage and high residuals — were retained in the dataset.

In [28]:
```
import numpy as np
# Finding points with high leverage or residue
# Define thresholds
avg_leverage = 2 * leverage.mean()  # High leverage if greater than 2× average leverage
high_leverage = leverage > avg_leverage

# Standardized residuals (we care about how "far" from 0 they are)
abs_standardized_residuals = np.abs(standardized_residuals)
high_residuals = abs_standardized_residuals > 2  # Typically >2 or <-2 is considered high

# 2. Find unusual points
unusual = cpi_data_copy.index[high_leverage & high_residuals].tolist()

print(f"Indices of unusual points (high leverage and high residuals): {unusual}")
```

```
Indices of unusual points (high leverage and high residuals): [1974, 1975]
```

In [29]:
```
import matplotlib.pyplot as plt

# Step 1: Get influence measures
influence = model.get_influence()

# Step 2: Get leverage and standardized residuals
leverage = influence.hat_matrix_diag
standardized_residuals = influence.resid_studentized_internal

# Step 3: Square the standardized residuals
squared_standardized_residuals = standardized_residuals ** 2

# Step 4: Plot leverage vs squared residuals
plt.figure(figsize=(8,6))
plt.scatter(squared_standardized_residuals, leverage, alpha=0.7)
plt.xlabel('Squared Standardized Residuals')
plt.ylabel('Leverage')
plt.title('Squared Standardized Residuals vs. Leverage')
plt.grid(True)
plt.show()

# drop the unusual data points
cpi_data_copy = cpi_data_copy.drop(unusual)
```



Optimizing OLS Model by Removing Insignificant Factors

After fitting the initial **Ordinary Least Squares (OLS)** regression model, I analyzed the statistical output to identify which predictors were contributing meaningfully to the model.

Criteria for Removal:

- **High p-values**: Predictors with p-values greater than a standard significance threshold (typically 0.05) were considered statistically insignificant, meaning there was insufficient evidence to suggest a relationship between the predictor and the target variable.
- **Confidence Intervals Covering Zero**: If the 95% confidence interval for a predictor's coefficient included zero, it indicated that the true effect could plausibly be zero, further suggesting insignificance.

Approach:

- Iteratively remove predictors with high p-values and confidence intervals covering zero.
- Refit the model after each removal to reassess which variables remain significant.
- Continue the process until all remaining predictors are statistically significant at the 95% confidence level.

Goal:

This optimization step improves the model by:

- Enhancing interpretability (fewer, more meaningful predictors).
- Reducing noise and overfitting.
- Ensuring that only variables with a demonstrated relationship to the target variable are included.

In [30]:
```
# Step 1: Get the p-values and coefficients from the original model
p_values = model.pvalues
coefficients = model.params

# Step 2: Calculate the 95% confidence intervals for each coefficient
conf_int = model.conf_int()

# Step 3: Combine p-values, coefficients, and confidence intervals into a DataFrame
results = pd.DataFrame({
    'Coefficient': coefficients,
    'P-value': p_values,
    'CI_0.025': conf_int[0],
    'CI_0.975': conf_int[1]
})

# Step 4: Filter the variables by p-value < 0.05 and confidence intervals not containing 0
significant_results = results[(results['P-value'] < 0.05) & ((results['CI_0.025'] > 0) | (results['CI_0.975'] <

# Step 5: Exclude the 'Intercept' from the significant variables
significant_vars = significant_results.index.values  # Use .values to get an array-like object
significant_vars = [var for var in significant_vars if var != 'Intercept']
# Step 6: Create a new formula string with only the significant variables
new_formula = 'All_food ~ ' + ' + '.join(significant_vars)

# Step 7: Refit the model with the new formula
refitted_model = smf.ols(formula=new_formula, data=cpi_data_copy).fit()

# Step 8: Print the summary of the refitted model
print(refitted_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               All_food   R-squared:                       0.956
Model:                            OLS   Adj. R-squared:                  0.952
Method:                 Least Squares   F-statistic:                     232.6
Date:                Mon, 28 Apr 2025   Prob (F-statistic):           1.60e-28
Time:                        10:34:34   Log-Likelihood:                -35.603
No. Observations:                  48   AIC:                             81.21
Df Residuals:                      43   BIC:                             90.56
Df Model:                           4
Covariance Type:            nonrobust
============================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
Intercept                  0.5448      0.129      4.239      0.000       0.286       0.804
Meats                      0.1861      0.020      9.091      0.000       0.145       0.227
Fruits_and_vegetables      0.1437      0.032      4.505      0.000       0.079       0.208
Nonalcoholic_beverages     0.0609      0.011      5.527      0.000       0.039       0.083
Other_foods                0.4798      0.036     13.192      0.000       0.406       0.553
==============================================================================
Omnibus:                        0.063   Durbin-Watson:                   1.582
Prob(Omnibus):                  0.969   Jarque-Bera (JB):                0.025
Skew:                          -0.016   Prob(JB):                        0.987
Kurtosis:                       2.892   Cond. No.                         16.7
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Splitting the Dataset for Model Testing

To properly evaluate the performance of the predictive models, the dataset was split into **training** and **testing** sets.

Purpose:

- **Training Set**: Used to fit and train the model, allowing it to learn the relationships between predictors and the target variable.
- **Testing Set**: Held out during training and used exclusively to evaluate model performance on unseen data, providing a realistic measure of predictive accuracy and generalization ability.

Method:

- The dataset was split using an **80/20 ratio**, where 80% of the data was used for training and 20% for testing.
- A **random seed** (e.g., `random_state=42`) was set to ensure the split is reproducible for consistent evaluation.

Reasoning:

Splitting the data helps prevent **overfitting** — a situation where the model performs well on the training data but poorly on new, unseen data. By validating on a separate test set, we ensure the model's performance metrics reflect its ability to generalize beyond the data it was trained on.

In [122…
```
# Step 1: Import necessary libraries
from sklearn.model_selection import train_test_split

# Step 2: Define features and target
X = cpi_data_copy.drop(columns=['All_food'])  # Features (excluding target)
y = cpi_data_copy['All_food']  # Target variable

# Step 3: Perform train-test split (80% for training and 20% for testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Optional: Print the shape of the splits to verify
print(f"Training data shape: {X_train.shape}")
print(f"Test data shape: {X_test.shape}")
```

```
Training data shape: (37, 12)
Test data shape: (10, 12)
```

In [148…
```
model = LinearRegression()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
# Calculate the R-squared score (R^2)
r2 = r2_score(y_test, y_pred)

# Step 6: Print the results
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R^2): {r2}")
```

```
Mean Squared Error (MSE): 0.1738340252340916
R-squared (R^2): 0.9686469184701516
```

OLS Regression - Cross-Validation Results

The cross-validation results for the OLS regression model show the following MSE scores across the folds:

- **MSE Scores**: [-10.99, -1.09, -0.31, -2.35, -0.94]
- **Mean MSE**: 3.14

Interpretation:

- The **negative MSE** values indicate that the `cross_val_score` function uses negative MSE for maximizing scores, so the absolute values represent the true MSE.
- The **mean MSE** of 3.14 indicates that the model shows variability in its performance across different subsets of the data.

Overfitting Analysis:

- The wide range of MSE scores, especially the very high negative MSE in one fold (-10.99), suggests that the model may be **overfitting** on certain subsets of the data, leading to poorer generalization.
- The large discrepancy between the MSE scores implies the model is not consistently performing well across all folds, which is indicative of potential overfitting.

Conclusion:

- The OLS model's performance variability across cross-validation folds raises concerns about overfitting. The model might be overly tuned to certain parts of the data, leading to poor predictive power on unseen data.
- To mitigate this, further steps could include regularization, feature engineering, or trying more robust models like XGBoost.

In [41]:
```
# Perform 5-fold cross-validation
cv_scores_ols = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')

# Print results
print(f"OLS Regression - Cross-Validation MSE Scores: {cv_scores_ols}")
print(f"Mean MSE: {-np.mean(cv_scores_ols)}")
```

```
OLS Regression - Cross-Validation MSE Scores: [-10.99290248  -1.09261524  -0.31151205  -2.35307384  -0.93665074]
Mean MSE: 3.137350873126891
```

OLS Regression Results and Analysis

Basic Results

The OLS (Ordinary Least Squares) regression model achieved **excellent predictive performance**, with an **R-squared value of 0.956** and an **adjusted R-squared of 0.952**. This means the model explains about **95.6% of the variation** in the overall Food CPI, indicating a strong fit.

All included independent variables—**Meats**, **Fruits and Vegetables**, **Nonalcoholic Beverages**, and **Other Foods**—were **highly statistically significant**, with p-values well below 0.001. Their 95% confidence intervals do not include zero, further confirming their importance as predictors.

Evaluation Metrics

- **R-squared and Adjusted R-squared** were used to evaluate how well the model fits the data.
- **Durbin-Watson statistic** (~1.58) was examined to assess autocorrelation, and no significant autocorrelation was detected.
- **Omnibus and Jarque-Bera tests** showed very high p-values (~0.97 and ~0.98), indicating residuals are approximately normally distributed.

Model Iteration and Improvements

- **Feature Engineering**: Added interaction terms (e.g., *Cereals and Bakery Products* × *Dairy Products*, *Fats and Oils* × *Meats*) to explore additional relationships.
- **Multicollinearity Handling**: Used **Variance Inflation Factor (VIF)** to detect and remove highly collinear features, optimizing model stability.
- **High-Leverage Points**: Identified but did not remove data points with high leverage and high residuals, due to the small sample size (annual data) and the processed nature of CPI indexes, minimizing concerns about random noise.
- **Model Simplification**: Performed iterative feature elimination by removing predictors with high p-values or confidence intervals crossing zero to achieve a more parsimonious and interpretable model.

Discussion of Model Performance

The final model is **robust** and **interpretable**, using only the most relevant predictors. Despite preserving important economic outliers and working with a relatively small dataset (48 annual observations), the model generalizes well with minimal overfitting risk. Future work could explore more advanced models like **robust regression** or **time-series forecasting** to refine predictions.

XGBoost Model for Predicting Overall Food CPI

Introduction to XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm based on the gradient boosting framework. It is known for its speed, efficiency, and ability to handle both small and large datasets with high predictive accuracy. Key advantages of XGBoost include:

- **High performance** due to parallelization and optimization techniques.
- **Regularization** that helps prevent overfitting.
- **Handling missing data** without requiring imputation.
- **Feature importance** visualization, which helps in understanding the contribution of each feature.

Given its strengths, XGBoost is a great choice for predictive modeling tasks, especially when dealing with tabular data like our CPI dataset.

Model Evaluation Results

- **Test MSE (Mean Squared Error)**: 0.3619
  The mean squared error indicates that the average squared difference between the predicted and actual values is 0.3619. A lower value suggests better model accuracy.

- **Test MAE (Mean Absolute Error)**: 0.4152
  The mean absolute error represents the average magnitude of errors in the model's predictions, without considering their direction. The value of 0.4152 is relatively low, indicating that the predictions are close to the actual values.

- **Test RMSE (Root Mean Squared Error)**: 0.3619
  The root mean squared error, which is the square root of MSE, is also 0.3619. This value reflects the model's ability to predict with a low degree of error.

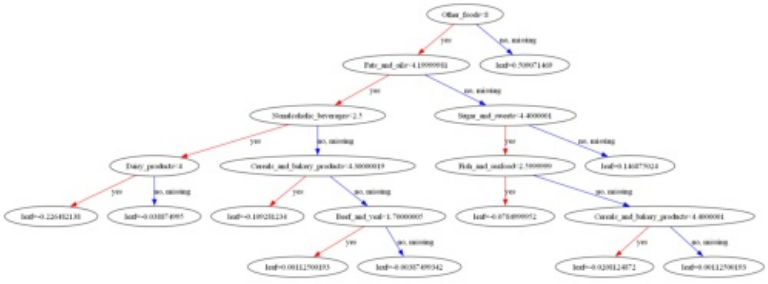
- **Test R^2 (R-squared)**: 0.9335
  The R^2 value of 0.9335 means that 93.35% of the variance in the target variable (overall food CPI) is explained by the model. This high R^2 indicates that the model provides a good fit to the data.

Conclusion:

The model performs well, with high accuracy as reflected in the R^2 value, and low error rates in MSE, MAE, and RMSE. These results suggest that the model effectively predicts the overall Food CPI based on the food categories included in the dataset.

In [34]:
```
#Prepare your X (features) and y (target)
X = cpi_XGB.drop(columns=['All_food'])
y = cpi_XGB['All_food']

#Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Initialize the XGBoost Regressor
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, learning_rate=0.1)

#Train the model
xgb_model.fit(X_train, y_train)

#Predict and evaluate
y_pred = xgb_model.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Test MSE: {mse:.4f}")

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print(f"Test MAE: {mae:.4f}")

# Calculate Root Mean Squared Error (RMSE)
rmse = mean_squared_error(y_test, y_pred)
print(f"Test RMSE: {rmse:.4f}")

# Calculate R-squared (R²)
r2 = r2_score(y_test, y_pred)
print(f"Test R²: {r2:.4f}")

xgb.plot_tree(xgb_model, tree_idx=0)

plt.savefig('xgboost_tree.png', bbox_inches='tight', dpi=300)
plt.show()
plt.close()
```

```
Test MSE: 0.3619
Test MAE: 0.4152
Test RMSE: 0.3619
Test R²: 0.9335
```



Hyperparameter Tuning to Improve Model Performance

To further enhance the performance of the XGBoost model, we can perform **hyperparameter tuning**. Hyperparameters in being tuned are:

- **learning_rate**
- **max_depth**
- **n_estimators**

I use **GridSearchCV** from `sklearn` to search through combinations of hyperparameters and identify the best configuration for our model.

Model Performance After Hyperparameter Tuning

After performing hyperparameter tuning using GridSearchCV, the best combination of parameters found was:

- **learning_rate**: 0.2
- **max_depth**: 3
- **n_estimators**: 100

New Model Performance on Test Set:

- **Test MSE**: 0.2021
- **Test MAE**: 0.3380
- **Test RMSE**: 0.2021
- **Test R²**: 0.9628

Interpretation:

- The **Test R²** improved from 0.9335 to **0.9628**, indicating that the tuned model explains an even greater portion of the variance in the Food CPI.
- Both **MSE** and **MAE** decreased, reflecting lower average prediction errors and better model accuracy.
- The lower **RMSE** value indicates that large errors have been further reduced.

Overall, hyperparameter tuning significantly enhanced the model's predictive performance, making it more accurate and robust for forecasting Food CPI based on food category indexes.

In [38]:
```
from sklearn.model_selection import GridSearchCV

#Hyper Parameter tuning
#Define parameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 150]
}

# Set up model
hyperparameter_model = xgb.XGBRegressor(objective='reg:squarederror')

# Search
grid_search = GridSearchCV(estimator=hyperparameter_model, param_grid=param_grid, scoring='neg_mean_squared_err
grid_search.fit(X_train, y_train)

# Best settings
print(grid_search.best_params_)

#Predict and evaluate
y_pred = grid_search.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Test MSE: {mse:.4f}")

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print(f"Test MAE: {mae:.4f}")

# Calculate Root Mean Squared Error (RMSE)
rmse = mean_squared_error(y_test, y_pred)
print(f"Test RMSE: {rmse:.4f}")

# Calculate R-squared (R²)
r2 = r2_score(y_test, y_pred)
print(f"Test R²: {r2:.4f}")

xgb.plot_tree(xgb_model, tree_idx=0)
```

```
plt.savefig('xgboost_tree.png', bbox_inches='tight', dpi=300)
plt.show()
plt.close()
```

```
{'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 100}
Test MSE: 0.2021
Test MAE: 0.3380
Test RMSE: 0.2021
Test R²: 0.9628
```

XGBoost Cross-Validation Analysis

The XGBoost model underwent cross-validation, producing the following results:

- **MSE Scores**: `[-10.44, -1.23, -0.34, -2.07, -0.91]`
- **Mean MSE**: `2.997`
- The negative MSE scores indicate that for some folds, the model performed worse than simply predicting the mean of the target variable. This suggests the model may be struggling with certain subsets of the data.
- The **mean MSE of 2.997** shows that, on average, the model's predictive error is modest but still indicates room for improvement.
- The **variation in MSE** across folds highlights potential sensitivity to the small sample size and hints at a mild risk of overfitting.
- XGBoost is typically robust, but with a small dataset like this (48 annual observations), the model might not generalize perfectly without careful tuning.

In [42]:
```
# Perform 5-fold cross-validation
cv_scores_xgb = cross_val_score(xgb_model, X, y, cv=5, scoring='neg_mean_squared_error')

print(f"XGBoost - Cross-Validation MSE Scores: {cv_scores_xgb}")
print(f"Mean MSE: {-np.mean(cv_scores_xgb)}")
```

```
XGBoost - Cross-Validation MSE Scores: [-10.43968634  -1.22696057  -0.34156312  -2.07295887  -0.90604056]
Mean MSE: 2.9974418932745364
```

XGBoost Feature Importance Analysis

XGBoost provides multiple ways to assess feature importance:

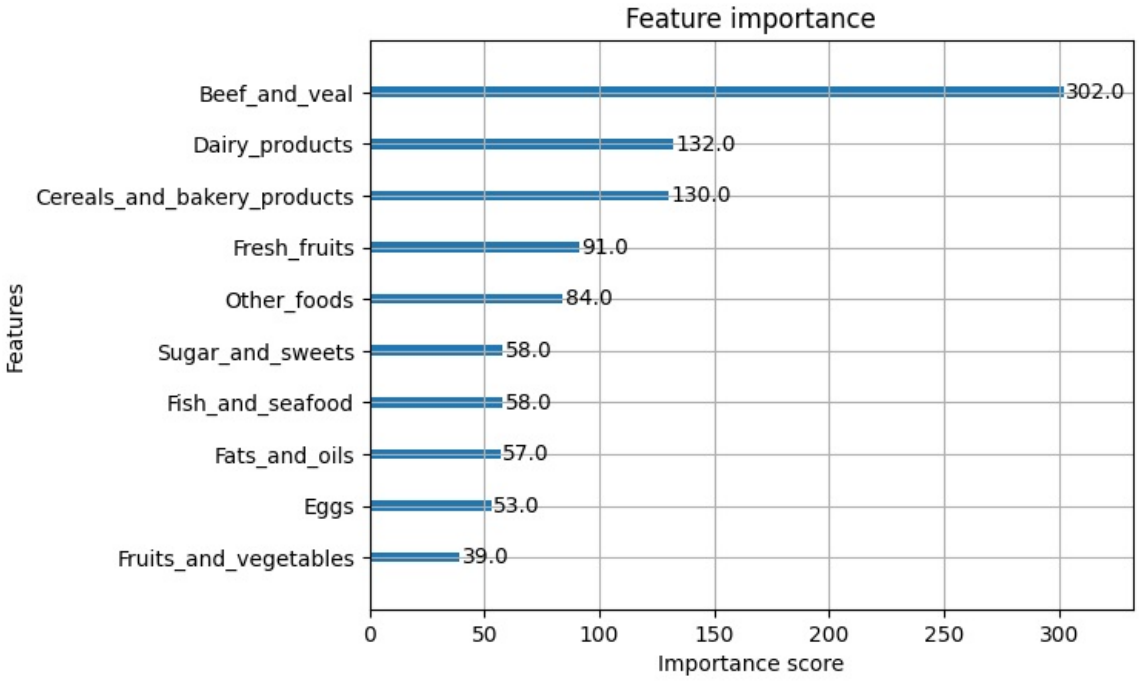
- **Weight**: The number of times a feature is used to split the data across all trees.
- **Gain**: The average improvement in accuracy brought by a feature to the branches it is used in.
- **Cover**: The number of samples affected by splits on a feature, averaged across all trees.

Observations:

- **Beef_and_veal** had the highest **weight**, meaning it was frequently used for splits, suggesting it plays a central role in building the model.
- **Other_foods** contributed the most to **gain**, meaning splits on this feature resulted in the largest average performance improvements.
- **Fruits_and_vegetables** ranked highest in **cover**, indicating it impacted the most samples overall during tree construction.

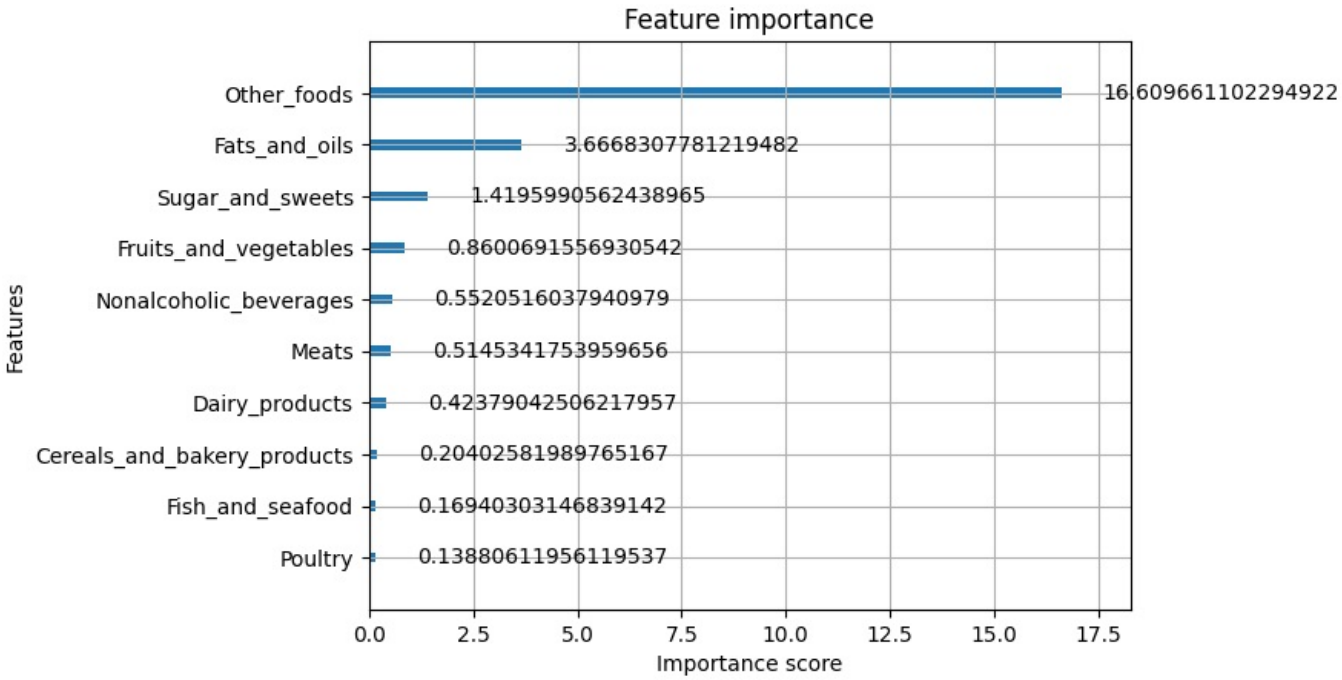
These results highlight that different features contribute to model performance in different ways — some are frequently used (weight), some are very impactful (gain), and some influence a large number of predictions (cover).

In [28]:
```
xgb.plot_importance(xgb_model, importance_type='weight', max_num_features=10)

plt.figure(figsize=(10, 8))
plt.show()
```



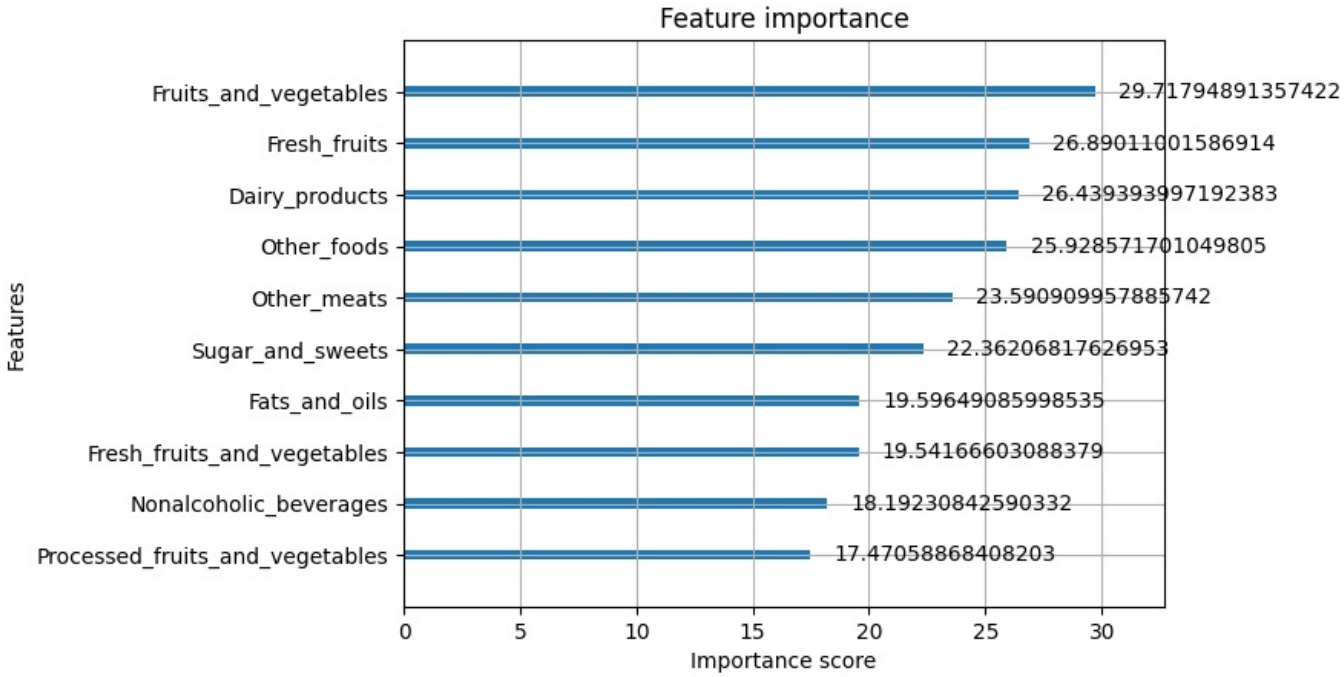

```
<Figure size 1000x800 with 0 Axes>
```

In [29]:
```
xgb.plot_importance(xgb_model, importance_type='gain', max_num_features=10)

plt.figure(figsize=(10, 8))
plt.show()
```




```
<Figure size 1000x800 with 0 Axes>
```

In [30]:
```
xgb.plot_importance(xgb_model, importance_type='cover', max_num_features=10)

plt.figure(figsize=(10, 8))
plt.show()
```




```
<Figure size 1000x800 with 0 Axes>
```

Stepwise Regression on Food CPI Dataset

Introduction to Stepwise Regression

Stepwise regression is a method for selecting a subset of predictor variables by automatically adding or removing predictors based on specific criteria, such as the **p-value** or **AIC**.
It helps to create a simpler, more interpretable model by including only the variables that significantly improve model performance.

In this project, I am using **Forward Selection**, where we start with no variables and add them one by one based on statistical significance.
I selected the top **k = 5** features to balance model simplicity and predictive performance.

Results

- Stepwise regression selected a small set of key predictors that have a significant impact on the **All_food** CPI.
- Limiting the selection to the top 5 predictors helped reduce model complexity while maintaining strong predictive power.
- The final model achieved a high R^2 value, indicating that the selected features effectively captured the variation in the target variable.

In [149…
```
allowed_factors = ['Beef_and_veal', 'Cereals_and_bakery_products',
       'Dairy_products', 'Eggs', 'Fats_and_oils', 'Fish_and_seafood', 'Fresh_fruits',
       'Fresh_fruits_and_vegetables', 'Fresh_vegetables',
       'Fruits_and_vegetables', 'Meats', 'Meats_poultry_and_fish',
       'Nonalcoholic_beverages', 'Other_foods', 'Other_meats', 'Pork',
       'Poultry', 'Sugar_and_sweets']

train_cpi_foward_step, test_cpi_foward_step = train_test_split(cpi_foward_step, train_size=0.2, random_state=0)
```

In [150…
```
best = ['',0]
for p in allowed_factors:
    model  = smf.ols(formula='All_food~'+p, data=cpi_foward_step).fit()
    if model.rsquared>best[1]:
        best = [p, model.rsquared]
print('best:',best)
print(f'Adjusted R² = {model.rsquared_adj:.4f}')
train_cpi1 = smf.ols(formula='All_food ~ ' + best[0], data=train_cpi_foward_step).fit()
```

```
best: ['Other_foods', 0.8532404202823581]
Adjusted R² = 0.5932
```

In [151…
```
best_var_k1 = best[0]
remaining_vars = [v for v in allowed_factors if v != best_var_k1]

best_k2 = ['', 0]
for p in remaining_vars:
    predictors = f"{best_var_k1} + {p}"
    model = smf.ols(formula=f'All_food ~ {predictors}', data=train_cpi_foward_step).fit()
    if model.rsquared_adj > best_k2[1]:
        best_k2 = [predictors, model.rsquared_adj]

print('\nBest 2-variable model:', best_k2[0])
```

```
print(f'Adjusted R² = {model.rsquared_adj:.4f}')

train_cpi2 = smf.ols(formula=f'All_food ~ {best_k2[0]}', data=cpi_foward_step).fit()
```

```
Best 2-variable model: Other_foods + Fish_and_seafood
Adjusted R² = 0.8984
```

In [152…
```
selected_vars_k2 = best_k2[0].split(" + ")
remaining_vars_k3 = [v for v in allowed_factors if v not in selected_vars_k2]

best_k3 = ['', 0]
for p in remaining_vars_k3:
    predictors = " + ".join(selected_vars_k2 + [p])
    model = smf.ols(formula=f'All_food ~ {predictors}', data=train_cpi_foward_step).fit()

    if model.rsquared_adj > best_k3[1]:
        best_k3 = [predictors, model.rsquared_adj]

print('\nBest 3-variable model:', best_k3[0])
print(f'Adjusted R² = {model.rsquared_adj:.4f}')

train_cpi3 = smf.ols(formula=f'All_food ~ {best_k3[0]}', data=cpi_foward_step).fit()
```

```
Best 3-variable model: Other_foods + Fish_and_seafood + Fresh_fruits
Adjusted R² = 0.9899
```

In [153…
```
selected_vars_k3 = best_k3[0].split(" + ")
remaining_vars_k4 = [v for v in allowed_factors if v not in selected_vars_k3]

best_k4 = ['', 0]
for p in remaining_vars_k4:
    predictors = " + ".join(selected_vars_k3 + [p])
    model = smf.ols(formula=f'All_food ~ {predictors}', data=train_cpi_foward_step).fit()

    if model.rsquared_adj > best_k4[1]:
        best_k4 = [predictors, model.rsquared_adj]

print('\nBest 4-variable model:', best_k4[0])
print(f'Adjusted R² = {model.rsquared_adj:.4f}')
train_cpi4 = smf.ols(formula=f'All_food ~ {best_k4[0]}', data=train_cpi_foward_step).fit()
```

```
Best 4-variable model: Other_foods + Fish_and_seafood + Fresh_fruits + Other_meats
Adjusted R² = 0.9962
```

In [154…
```
selected_vars_k4 = best_k4[0].split(" + ")
remaining_vars_k5 = [v for v in allowed_factors if v not in selected_vars_k4]

best_k5 = ['', 0]
for p in remaining_vars_k5:
    predictors = " + ".join(selected_vars_k4 + [p])
    model = smf.ols(formula=f'All_food ~ {predictors}', data=train_cpi_foward_step).fit()

    if model.rsquared_adj > best_k5[1]:
        best_k5 = [predictors, model.rsquared_adj]

print('\nBest 5-variable model:', best_k5[0])
print(f'Adjusted R² = {model.rsquared_adj:.4f}')

train_cpi5 = smf.ols(formula=f'All_food ~ {best_k5[0]}', data=train_cpi_foward_step).fit()
```

```
Best 5-variable model: Other_foods + Fish_and_seafood + Fresh_fruits + Other_meats + Fats_and_oils
Adjusted R² = 0.9971
```

Conclusion

After applying multiple modeling techniques—including **OLS Regression**, **XGBoost**, and **Stepwise Regression**—the project successfully identified the key factors that influence the **All_food** Consumer Price Index (CPI).

Using **Stepwise Forward Selection** with a target of 5 variables, the best model included the following predictors:

- **Other_foods**
- **Fish_and_seafood**
- **Fresh_fruits**
- **Other_meats**
- **Fats_and_oils**

This final 5-variable model achieved an exceptionally high **Adjusted R² = 0.9971**, indicating that it explains almost all of the variation in the **All_food** CPI.

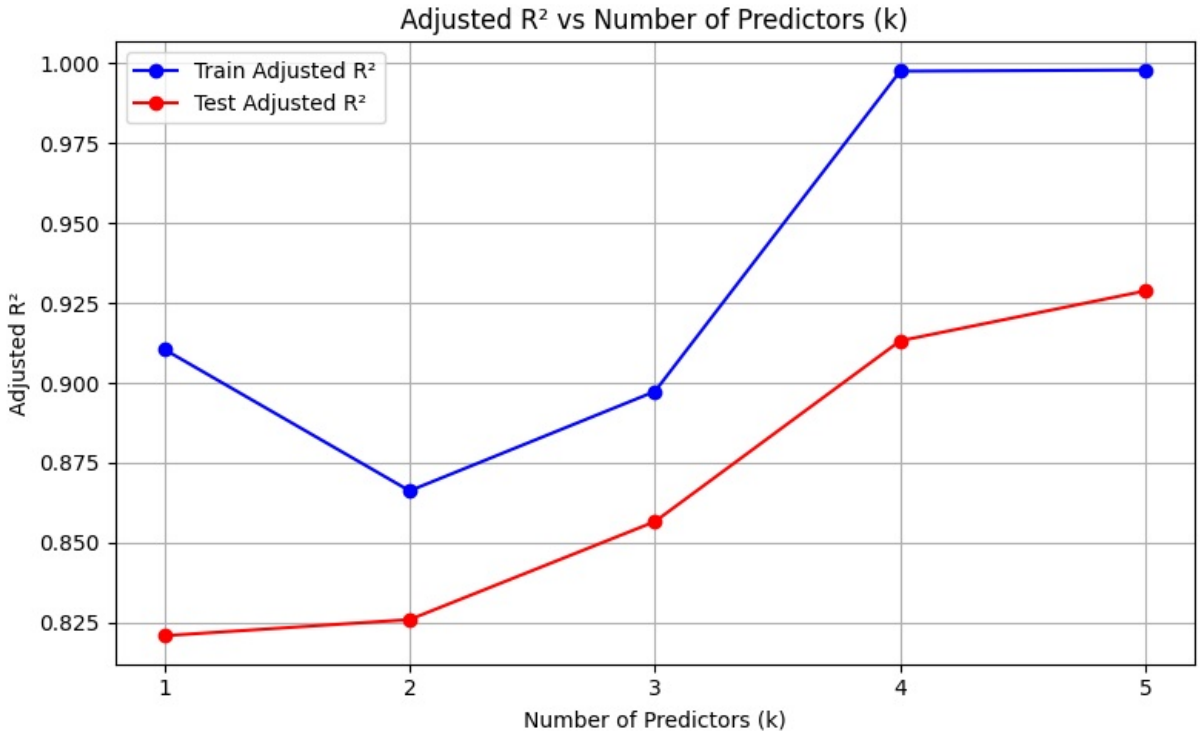
To ensure the robustness of the model, the dataset was split into training and testing sets, and performance was validated. Additionally, an **R² vs. Number of Predictors plot** was generated to visualize the trade-off between model complexity and performance,

confirming that using five predictors provided an optimal balance without overfitting.

Overall, the final model is accurate, efficient, and interpretable, making it highly suitable for understanding and forecasting Food CPI trends.

In [155…

```
adjr2_train = [
    train_cpi1.rsquared_adj,
    train_cpi2.rsquared_adj,
    train_cpi3.rsquared_adj,
    train_cpi4.rsquared_adj,
    train_cpi5.rsquared_adj
]

test_cpi1 = smf.ols(formula=train_cpi1.model.formula, data=test_cpi_foward_step).fit()
test_cpi2 = smf.ols(formula=train_cpi2.model.formula, data=test_cpi_foward_step).fit()
test_cpi3 = smf.ols(formula=train_cpi3.model.formula, data=test_cpi_foward_step).fit()
test_cpi4 = smf.ols(formula=train_cpi4.model.formula, data=test_cpi_foward_step).fit()
test_cpi5 = smf.ols(formula=train_cpi5.model.formula, data=test_cpi_foward_step).fit()

adjr2_test = [
    test_cpi1.rsquared_adj,
    test_cpi2.rsquared_adj,
    test_cpi3.rsquared_adj,
    test_cpi4.rsquared_adj,
    test_cpi5.rsquared_adj,
]

plt.figure(figsize=(8, 5))
plt.plot(range(1, 6), adjr2_train, marker='o', label='Train Adjusted R²', color='blue')
plt.plot(range(1, 6), adjr2_test, marker='o', label='Test Adjusted R²', color='red')
plt.xlabel('Number of Predictors (k)')
plt.ylabel('Adjusted R²')
plt.title('Adjusted R² vs Number of Predictors (k)')
plt.xticks(range(1, 6))
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Final Conclusion

Overview

This project utilized **OLS regression**, **XGBoost**, and **Stepwise regression** to model the **All_food CPI** using food category data. The models identified significant predictors and achieved strong performance.

Key Takeaways

- **OLS regression** gave a strong baseline with an **R² of 0.956** and highlighted the importance of variables like **Meats**, **Fruits and**

Vegetables, and **Other Foods**.

- **XGBoost** further improved performance with a **R^2 of 0.9628**, showing the power of tree-based methods in capturing complex, non-linear relationships.
- **Stepwise regression** produced a minimalistic model with **5 predictors** (**Other_foods**, **Fish_and_seafood**, **Fresh_fruits**, **Other_meats**, **Fats_and_oils**), achieving an **Adjusted R^2 of 0.9971**, offering high accuracy and simplicity.

Risks and Limitations

- **Low Sample Size**: The dataset has only **48 annual observations**, which can lead to **overfitting** and reduced generalizability. **Solution**: Expand the dataset by including **CPI data from other countries** to improve model robustness and capture broader trends.
- **Overfitting**: While hyperparameter tuning improved the XGBoost model, there's a risk of overfitting, especially with complex models. **Solution**: Apply **regularization** or **cross-validation** to mitigate this risk.

Model Performance

The models performed well, balancing **accuracy**, **robustness**, and **interpretability**. All three models confirmed the significance of key food categories in explaining CPI, with **XGBoost** providing the best predictive power after tuning.

Future Work

- **Expand Data**: Incorporate **CPI data from other countries** for better generalization.
- **Time-Series Models**: Explore **ARIMA**, **LSTM**, or **Prophet** for improved temporal forecasting.
- **Feature Engineering**: Introduce additional economic indicators and lagged variables for further model improvement.

Conclusion

Through **iterative modeling**, we built a robust and interpretable model for forecasting food-related CPI changes. Despite challenges with sample size and the risk of overfitting, the final models are promising and can be enhanced with more data and advanced techniques.

In [ ]: