Ankita, Louise, Mike

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sqlite3
import statsmodels.api as m
```

Final Project: Food Spending, Nutrient Intake & Pricing Trends

Introduction

This project explores the intersection of food spending, nutrient intake, and pricing trends in the United States. Our target audience includes policymakers, public health officials, and nutrition researchers interested in understanding how economic and consumption patterns influence dietary outcomes.

We chose this topic to address growing concerns around food affordability, nutritional quality, and public health equity. By analyzing multiple datasets — from CPI forecasts to historical food sales and nutrient intake data — we aim to answer the following questions:

1. How have food prices evolved over time, and what are their projected trends?
2. How do changes in food pricing relate to shifts in nutrient and food group intake?
3. Where are the greatest disparities or potential areas for intervention?

Through rigorous exploratory data analysis, SQL-based data wrangling, and a logistic regression model, we provide both descriptive and predictive insights into the economic and health implications of food consumption in the U.S.

1. Datasets Overview & Quality Checks

Datasets Used

- **CPI Data**: Contains percent change in consumer food prices across categories for 2024, historical averages, and YoY projections.
- **Sales Data**: Annual food sales by state from 1997–2023, including nominal and real dollars, with and without taxes/tips.
- **Per Capita Sales**: Same as above, but adjusted by population.
- **Nutrient Intake Data**: Average U.S. nutrient intake by food source and year (1977–2018).
- **Food Group Intake**: Similar to nutrient intake but by food group.
- **Food Density**: Nutrient or group density per 1000 calories, by food source.
- **Recommended Densities**: USDA guidelines for nutrient targets per 1000 calories.
- **Sample Size Table**: Survey sample sizes by demographic and year.

```
CPI = pd.read_csv("CPIForecast.csv") # Consumer Price Index data set
sales = pd.read_csv("state_sales.csv") # food sales by state with tax & tip
sales_percapita = pd.read_csv("state_sales_per_capita.csv") # food sales by state (per capita)
sales_NTT = pd.read_csv("state_sales_no_taxes_tips.csv") # food sales by state excluding tax/tip
sales_percapNTT = pd.read_csv("state_sales_per_capita_no_taxes_tips.csv") # food sales by state (per capita) excluding tax/tip
size = pd.read_csv("table-1-sample-sizes.csv") # sample size for consumption data tables
nut_intake = pd.read_csv("table-2-US-nutrient-intake-by-food-source.csv") #US nutrient intake by food source
food_group = pd.read_csv("table-5-US-food-group-intakes-by-food-source.csv") #US food group intake by food source
density = pd.read_csv("table-7-US-food-density-of-food-group-by-food-source.csv") #US food density of food groups by food source
rec_density = pd.read_csv("table-8-recommended-density-and-2017-2018-density.csv") #recommended food density
```

Initial Exploration

We used common exploratory checks to understand the shape, structure, and quality of each dataset before cleaning. Below are some examples:

- `nunique()` helped us identify how many distinct values were in each column.
- `list(df.columns)` gave us quick insight into feature names and formats.
- `isna().sum()` flagged missing values, especially in the nutrient intake data.
- We isolated and reviewed rows with missing nutrient values for further inspection.

```
# looking at the beginning of each data set
CPI.head()
```

	Top-level	Aggregate	Mid-level	Low-level	Disaggregate	Attribute	Unit	Value
0	All food	NaN	NaN	NaN	NaN	Relative importance	Percent	100.0
1	All food	NaN	NaN	NaN	NaN	Month-to-month February 2025 to March 2025	Percent change	0.4
2	All food	NaN	NaN	NaN	NaN	Year-over-year March 2024 to March 2025	Percent change	3.0
3	All food	NaN	NaN	NaN	NaN	Year-to-date avg. 2025 to avg. 2024	Percent change	1.9
4	All food	NaN	NaN	NaN	NaN	Annual 2022	Percent change	9.9

```
# looking at the end of each data set
sales_percapita.tail()
```

	Year	State	FAH sales per capita nominal U.S. dollars with taxes and tips	FAFH sales per capita nominal U.S. dollars with taxes and tips	Total sales per capita nominal U.S. dollars with taxes and tips	FAH sales per capita constant 1988 U.S. dollars with taxes and tips	FAFH sales per capita constant 1988 U.S. dollars with taxes and tips	Total sales per capita constant 1988 U.S. dollars with taxes and tips
1372	2019	Wyoming	2,873.98	2,851.21	5,725.19	1,336.29	1,196.14	2,532.43
1373	2020	Wyoming	3,121.94	2,758.76	5,880.70	1,395.92	1,106.66	2,502.58
1374	2021	Wyoming	3,389.81	3,514.07	6,903.87	1,452.35	1,344.11	2,796.46
1375	2022	Wyoming	3,591.28	3,778.54	7,369.83	1,381.16	1,347.13	2,728.29

```
# number of rows & columns
rec_density.shape
```

```
(299, 4)
```

```
# sampling each data set
sales.sample(10)
```

	Year	State	FAH sales million nominal U.S. dollars with taxes and tips	FAFH sales million nominal U.S. dollars with taxes and tips	Total sales million nominal U.S. dollars with taxes and tips	FAH sales million constant 1988 U.S. dollars with taxes and tips	FAFH sales million constant 1988 U.S. dollars with taxes and tips	Total sales million constant 1988 U.S. dollars with taxes and tips
1000	1998	Oregon	4,930.35	3,603.21	8,533.56	3420.1	2738.99	6,159.09
463	2001	Kentucky	5,858.93	4,454.42	10,313.34	4,001.81	3,097.84	7,099.65
540	1997	Maryland	7,652.77	5,274.06	12,926.83	5,691.53	4,049.85	9,741.38
901	2007	North Carolina	17,011.12	14,541.97	31,553.09	9966.7	8541.66	18,508.36
857	2017	New Mexico	4,876.20	5,054.35	9,930.55	2,311.86	2,270.60	4,582.46
698	2020	Missouri	15,516.24	13,798.16	29,314.40	7,634.71	5,679.01	13,313.72
140	2002	Colorado	8,067.49	6,980.75	15,048.24	5,065.02	4,767.93	9,832.95
709	2004	Montana	1,775.34	1,218.29	2,993.63	1,046.33	791.92	1,838.25

```
# looking at data types
sales_NTT.dtypes
```

	0
Year	int64
State	object
FAH sales million nominal U.S. dollars without taxes and tips	object
FAFH sales million nominal U.S. dollars without taxes and tips	object
Total sales million nominal U.S. dollars without taxes and tips	object
FAH sales million constant 1988 U.S. dollars without taxes and tips	object
FAFH sales million constant 1988 U.S. dollars without taxes and tips	object
Total food sales million constant 1988 U.S. dollars without taxes and tips	object

dtype: object

```
# looking at unique values in each column
sales_percapNTT.nunique()
```

	0
Year	27
State	51
FAH sales per capita nominal U.S. dollars without taxes and tips	1374
FAFH sales per capita nominal U.S. dollars without taxes and tips	1371
Total sales per capita nominal U.S. dollars without taxes and tips	1374
FAH sales per capita constant 1988 U.S. dollars without taxes and tips	1353
FAFH sales per capita constant 1988 U.S. dollars without taxes and tips	1360
Total sales per capita constant 1988 U.S. dollars without taxes and tips	1367

dtype: int64

```
# list of column names
list(size.columns)
```

```
['Demographics', 'Survey years', 'Sample size', 'Table']
```

```
# check for null values in data
nut_intake.isna().sum()
```

	0
Nutrient	0
Food source	0
Measurement	0
Nutrient:Food source	0
Survey years:Variable	0
Value	1776
Table	0
Demographics	0

dtype: int64

```
# created new dataframe to investigate observations with nulls in "Value" column
null_intake = nut_intake[nut_intake['Value'].isna()]
print("Rows where Value is null:\n", null_intake)
```

```
Rows where Value is null:
                            Nutrient   Food source Measurement  \
7397                         Energy   FAFH: School    Calories
7404                        Calcium   FAFH: School  Milligrams
7411                 Fiber, dietary   FAFH: School       Grams
7418                           Iron   FAFH: School  Milligrams
7425                        Protein   FAFH: School       Grams
```

```
...                          ...           ...         ...
36426                   Total Fat   FAFH: School       Grams
36433        Saturated fatty acids   FAFH: School       Grams
36440  Fatty acids, monounsaturated  FAFH: School       Grams
36447  Fatty acids, polyunsaturated  FAFH: School       Grams
36454                       Sodium  FAFH: School  Milligrams

                            Nutrient:Food source Survey years:Variable  Value  \
7397                        Energy :FAFH: School       1977-1978-Mean    NaN
7404                       Calcium :FAFH: School       1977-1978-Mean    NaN
7411               Fiber, dietary :FAFH: School       1977-1978-Mean    NaN
7418                          Iron :FAFH: School       1977-1978-Mean    NaN
7425                        Protein:FAFH: School       1977-1978-Mean    NaN
...                                          ...                   ...    ...
36426                    Total Fat :FAFH: School  2017-2018-SE of mean    NaN
36433        Saturated fatty acids :FAFH: School  2017-2018-SE of mean    NaN
36440  Fatty acids, monounsaturated:FAFH: School  2017-2018-SE of mean    NaN
36447  Fatty acids, polyunsaturated:FAFH: School  2017-2018-SE of mean    NaN
36454                       Sodium:FAFH: School  2017-2018-SE of mean    NaN

                                                   Table  \
7397   Table 2B2-Daily nutrient intake by food source...
7404   Table 2B2-Daily nutrient intake by food source...
7411   Table 2B2-Daily nutrient intake by food source...
7418   Table 2B2-Daily nutrient intake by food source...
7425   Table 2B2-Daily nutrient intake by food source...
...                                                  ...
36426  Table 2F3-Daily nutrient intake by food source...
36433  Table 2F3-Daily nutrient intake by food source...
36440  Table 2F3-Daily nutrient intake by food source...
36447  Table 2F3-Daily nutrient intake by food source...
36454  Table 2F3-Daily nutrient intake by food source...

                 Demographics
7397               Ages 20-64
7404               Ages 20-64
7411               Ages 20-64
7418               Ages 20-64
7425               Ages 20-64
...                       ...
36426  Edu. - College attended
36433  Edu. - College attended
36440  Edu. - College attended
36447  Edu. - College attended
36454  Edu. - College attended

[1776 rows x 8 columns]
```

Key Observations

We applied a consistent set of quality checks across all datasets (e.g., checking nulls, inspecting column names, evaluating structure). Below is a summary of key issues and opportunities found during that review:

CPI Data

- Hierarchical structure with 8 columns, many containing NULLs.
- Column labels lacked clarity; several fields were removed or renamed.

Food Sales Data (4 tables total)

- All dollar amounts were stored as strings and required conversion to numeric types.
- No missing values detected.
- Time range spans 1997 to 2023.

Sales Sample Size Description Data

- Survey years were stored as ranges, not tidy — required expansion into individual rows.
- Three NULL values in the “Sample size” field were dropped.
- Covers 1977 to 2018.

Nutrient Intake Data

- “Survey years:Variable” combined year ranges and variable labels in one field — needed splitting.
- ~4.9% missing values in the “Value” field (1,776 out of 36,456).
- Several column names required renaming for clarity.

Food Group Intake Data

- Same structure issues as Nutrient Intake: untidy year/variable field, unclear column labels.
- ~4.9% missing values (5,328 out of 109,368).

Food Density of Food Groups Data

- Same structure and quality issues as Food Group Intake.
- Also ~4.9% missing values (5,328 out of 109,368).

Recommended Food Density Data

- Some uninformative column names.
- “Value” column stored as a string, despite being numeric — required conversion.

2. Data Cleaning

After identifying quality issues in each dataset, we cleaned and transformed the data to make it tidy, analysis-ready, and consistent across sources. Below are the major cleaning steps organized by dataset.

CPI Data Cleaning

In order to clean the CPI data set, we had to decide which aggregated levels of the hierarchy would be most useful for our analysis and which attributes were most relevant. When looking at the data, we decided to remove the first column called Top-level which only had one unique value ("All Food") and the Low-level and Disaggregate columns which were too granular for our analysis. We also renamed the columns to more relevant values.

Cleaning the CPI Data

- Removed redundant hierarchical levels
- Renamed unclear columns
- Filtered to 3 relevant attributes
- Converted percent change values to numeric

```
# CPI Data Cleaning
CPI = CPI.drop(columns=['Top-level', 'Low-level', 'Disaggregate'])

# rename columns
# note to check with team if homecooked or takeout is a good field name
CPI = CPI.rename(columns = {'Aggregate': 'Homecooked or Takeout', 'Mid-level':'Food Category'})
```

Next, we reviewed the Attribute column to see what information we want to keep. From the unique values, we determined that we wanted to keep the following data points: Annual 2024, 20-year historical average, Year-over-year March 2024 to March 2025.

```
# Unique values under the Attribute column

CPI['Attribute'].unique()
```

```
array(['Relative importance',
       'Month-to-month February 2025 to March 2025',
       'Year-over-year March 2024 to March 2025',
       'Year-to-date avg. 2025 to avg. 2024', 'Annual 2022',
       'Annual 2023', 'Annual 2024', '20-year historical average',
       'Lower bound of prediction interval 2025',
       'Mid point of prediction interval 2025',
       'Upper bound of prediction interval 2025'], dtype=object)
```

```
#filtered for rows with the attributes chosen
CPI_final = CPI[CPI['Attribute'].isin(['Year-over-year March 2024 to March 2025', 'Annual 2024', '20-year historical average'])]
```

The attributes chosen all had the same unit (percent change) so we dropped the Unit column and renamed the Value column.

```
# drop column 'Unit'
CPI_final = CPI.drop(columns=['Unit'])
```

```
# renamed column 'Value'
CPI_final = CPI_final.rename(columns = {'Value':'Percent Change'})
```

Then the rows that aggregated data at the top level of the hierarchy were dropped by removing observations that had NULL values in the "Homecooked or Takeout' and "Food Category' columns.

```
CPI_final.dropna(subset= ['Homecooked or Takeout'], inplace = True) #drop rows with NULL
CPI_final.dropna(subset= ['Food Category'], inplace = True) #drop rows with NULL
```

```
CPI_final.isna().sum() # check for null values in data
```

	0
Homecooked or Takeout	0
Food Category	0
Attribute	0
Percent Change	0

dtype: int64

```
# reset index
CPI_final.reset_index(drop=True, inplace=True)

#create copy
CPI_final = CPI_final.copy(deep=True) # actual copy
```

Cleaning the Sales Data (Nominal & Per Capita)

- Converted dollar strings to numeric
- Standardized column names
- Kept both total and per capita values (with and without taxes/tips)

```
# Clean dollar fields in sales data
def clean_dollar_columns(df, cols):
    df[cols] = df[cols].replace(',', '', regex=True).apply(pd.to_numeric, errors='coerce')
    return df

sales_final = clean_dollar_columns(sales.copy(), [
    'FAH sales million nominal U.S. dollars with taxes and tips',
    'FAFH sales million nominal U.S. dollars with taxes and tips',
    'Total sales million nominal U.S. dollars with taxes and tips',
    'FAH sales million constant 1988 U.S. dollars with taxes and tips',
    'FAFH sales million constant 1988 U.S. dollars with taxes and tips',
    'Total sales million constant 1988 U.S. dollars with taxes and tips'
])
sales_final
```

	Year	State	FAH sales million nominal U.S. dollars with taxes and tips	FAFH sales million nominal U.S. dollars with taxes and tips	Total sales million nominal U.S. dollars with taxes and tips	FAH sales million constant 1988 U.S. dollars with taxes and tips	FAFH sales million constant 1988 U.S. dollars with taxes and tips	Total sales million constant 1988 U.S. dollars with taxes and tips
0	1997	Alabama	5789.24	3465.67	9254.92	4305.59	2661.22	6966.81
1	1998	Alabama	6064.19	3841.10	9905.29	4444.25	2873.78	7318.04
2	1999	Alabama	6408.42	4101.04	10509.46	4623.19	2995.03	7618.22
3	2000	Alabama	6751.17	4352.40	11103.57	4743.07	3108.13	7851.20
4	2001	Alabama	6892.75	4604.09	11496.84	4707.94	3201.94	7909.88
...	...	...	...	...	...	...	...	...
1372	2019	Wyoming	1667.24	1654.03	3321.27	775.20	693.90	1469.10
1373	2020	Wyoming	1803.43	1593.64	3397.07	806.37	639.28	1445.65
1374	2021	Wyoming	1964.56	2036.57	4001.13	841.70	778.98	1620.68
1375	2022	Wyoming	2088.80	2197.71	4286.50	803.32	783.53	1586.85
1376	2023	Wyoming	2125.13	2336.42	4461.55	781.15	775.22	1556.37

```
sales_percapita_final = clean_dollar_columns(sales_percapita.copy(), [
    'FAH sales per capita nominal U.S. dollars with taxes and tips',
    'FAFH sales per capita nominal U.S. dollars with taxes and tips',
    'Total sales per capita nominal U.S. dollars with taxes and tips',
    'FAH sales per capita constant 1988 U.S. dollars with taxes and tips',
    'FAFH sales per capita constant 1988 U.S. dollars with taxes and tips',
    'Total sales per capita constant 1988 U.S. dollars with taxes and tips'
])
sales_percapita_final
```

	Year	State	FAH sales per capita nominal U.S. dollars with taxes and tips	FAFH sales per capita nominal U.S. dollars with taxes and tips	Total sales per capita nominal U.S. dollars with taxes and tips	FAH sales per capita constant 1988 U.S. dollars with taxes and tips	FAFH sales per capita constant 1988 U.S. dollars with taxes and tips	Total sales per capita constant 1988 U.S. dollars with taxes and tips
0	1997	Alabama	1340.02	802.19	2142.20	996.60	615.98	1612.58
1	1998	Alabama	1393.73	882.80	2276.54	1021.42	660.48	1681.91
2	1999	Alabama	1466.50	938.48	2404.99	1057.97	685.38	1743.35
3	2000	Alabama	1516.38	977.59	2493.97	1065.34	698.12	1763.45
4	2001	Alabama	1542.82	1030.54	2573.36	1053.79	716.70	1770.48
...	...	...	...	...	...	...	...	...
1372	2019	Wyoming	2873.98	2851.21	5725.19	1336.29	1196.14	2532.43
1373	2020	Wyoming	3121.94	2758.76	5880.70	1395.92	1106.66	2502.58
1374	2021	Wyoming	3389.81	3514.07	6903.87	1452.35	1344.11	2796.46
1375	2022	Wyoming	3591.28	3778.54	7369.83	1381.16	1347.13	2728.29
1376	2023	Wyoming	3638.57	4000.32	7638.89	1337.45	1327.31	2664.75

```
# Clean NTT (No Taxes and Tips) sales data
sales_NTT_final = sales_NTT.copy()
sales_NTT_final = clean_dollar_columns(sales_NTT_final, [
    'FAH sales million nominal U.S. dollars without taxes and tips',
    'FAFH sales million nominal U.S. dollars without taxes and tips',
    'Total sales million nominal U.S. dollars without taxes and tips',
    'FAH sales million constant 1988 U.S. dollars without taxes and tips',
    'FAFH sales million constant 1988 U.S. dollars without taxes and tips',
    'Total food sales million constant 1988 U.S. dollars without taxes and tips'
])
sales_NTT_final
```

	Year	State	FAH sales million nominal U.S. dollars without taxes and tips	FAFH sales million nominal U.S. dollars without taxes and tips	Total sales million nominal U.S. dollars without taxes and tips	FAH sales million constant 1988 U.S. dollars without taxes and tips	FAFH sales million constant 1988 U.S. dollars without taxes and tips	Total food sales million constant 1988 U.S. dollars without taxes and tips
0	1997	Alabama	5447.90	3098.49	8546.39	4051.72	2379.27	6430.99
1	1998	Alabama	5690.14	3428.02	9118.16	4170.12	2564.73	6734.85
2	1999	Alabama	5995.80	3653.72	9649.52	4325.52	2668.35	6993.86
3	2000	Alabama	6298.34	3871.33	10169.67	4424.93	2764.59	7189.52
4	2001	Alabama	6411.99	4089.29	10501.28	4379.57	2843.92	7223.48
...	...	...	...	...	...	...	...	...
1372	2019	Wyoming	1644.87	1484.54	3129.41	764.80	622.79	1387.59
1373	2020	Wyoming	1779.59	1433.19	3212.77	795.71	574.91	1370.62
1374	2021	Wyoming	1938.77	1822.79	3761.56	830.66	697.21	1527.86
1375	2022	Wyoming	2063.55	1970.82	4034.37	793.61	702.64	1496.25
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
# Clean per capita NTT version
sales_percapNTT_final = sales_percapNTT.copy()
sales_percapNTT_final = clean_dollar_columns(sales_percapNTT_final, [
    'FAH sales per capita nominal U.S. dollars without taxes and tips',
    'FAFH sales per capita nominal U.S. dollars without taxes and tips',
    'Total sales per capita nominal U.S. dollars without taxes and tips',
    'FAH sales per capita constant 1988 U.S. dollars without taxes and tips',
    'FAFH sales per capita constant 1988 U.S. dollars without taxes and tips',
    'Total sales per capita constant 1988 U.S. dollars without taxes and tips'
])
sales_percapNTT_final
```

	Year	State	FAH sales per capita nominal U.S. dollars without taxes and tips	FAFH sales per capita nominal U.S. dollars without taxes and tips	Total sales per capita nominal U.S. dollars without taxes and tips	FAH sales per capita constant 1988 U.S. dollars without taxes and tips	FAFH sales per capita constant 1988 U.S. dollars without taxes and tips	Total sales per capita constant 1988 U.S. dollars without taxes and tips
0	1997	Alabama	1261.01	717.20	1978.20	937.84	550.72	1488.56
1	1998	Alabama	1307.77	787.86	2095.63	958.42	589.45	1547.87
2	1999	Alabama	1372.08	836.12	2208.20	989.85	610.63	1600.48
3	2000	Alabama	1414.67	869.54	2284.20	993.88	620.95	1614.83
4	2001	Alabama	1435.21	915.31	2350.52	980.29	636.56	1616.85
...	...	...	...	...	...	...	...	...
1372	2019	Wyoming	2835.42	2559.04	5394.45	1318.36	1073.56	2391.93
1373	2020	Wyoming	3080.66	2481.00	5561.66	1377.46	995.24	2372.70
1374	2021	Wyoming	3345.31	3145.19	6490.50	1433.28	1203.02	2636.30
1375	2022	Wyoming	3547.88	3388.44	6936.32	1364.46	1208.06	2572.52
1376	2023	Wyoming	3589.80	3590.88	7180.68	1319.52	1191.45	2510.98

Cleaning the Sample Size Data

- Dropped nulls and unnecessary columns
- Converted ranges like `1977–78` into one row per year

```
# Drop redundant column
if "Table" in size.columns:
    size.drop(columns=["Table"], inplace=True)

# Clean text/remove space
size.columns = size.columns.str.strip()
size["Demographics"] = size["Demographics"].str.strip()
size["Survey years"] = size["Survey years"].str.strip()
```

```
size.dropna(subset= ['Sample size'], inplace = True) #drop rows with NULL
# Convert 'Sample size' to integer
size["Sample size"] = size["Sample size"].astype(int)

# Remove duplicate rows
size = size.drop_duplicates()


# Expand survey year ranges (e.g., '1977–1978') into separate rows per year
expanded_rows = []
for _, row in size.iterrows():
    if '–' in row["Survey years"]:
        start_year, end_year = map(int, row["Survey years"].split('–'))
        for year in range(start_year, end_year + 1):
            new_row = row.copy()
            new_row["Survey year"] = year
            expanded_rows.append(new_row)
    else:
        new_row = row.copy()
        new_row["Survey year"] = int(row["Survey years"])
        expanded_rows.append(new_row)
```

To split up the Survey Years column

```
# Convert list of expanded rows to a DataFrame
size_final = pd.DataFrame(expanded_rows)

size_final = size_final.drop(columns=["Survey years"])
size_final = size_final[
    (size_final["Survey year"] >= 1977) & (size_final["Survey year"] <= 2018)
]
size_final.head()
```

	Demographics	Sample size	Survey year
0	U.S. aged 2 and above	41471	1977
0	U.S. aged 2 and above	41471	1978
1	Male	18303	1977
1	Male	18303	1978
2	Female	23168	1977

Cleaning the Nutrient Intake Data

- Dropped null values and filtered for "mean" entries
- Split combined fields into tidy format
- Expanded year ranges into individual rows
- Created long and wide versions for analysis

```
nut_intake
```

	Nutrient	Food source	Measurement	Nutrient:Food source	Survey years:Variable	Value	Table	Demographics
0	Energy	Total	Calories	Energy :Total	1977-1978-Mean	1806.88	Table 2-Daily nutrient intake by food source f...	US consumers aged 2 and above
1	Energy	FAH	Calories	Energy :FAH	1977-1978-Mean	1462.27	Table 2-Daily nutrient intake by food source f...	US consumers aged 2 and above
2	Energy	FAFH	Calories	Energy :FAFH	1977-1978-Mean	344.61	Table 2-Daily nutrient intake by food source f...	US consumers aged 2 and above
3	Energy	FAFH: Restaurant	Calories	Energy :FAFH: Restaurant	1977-1978-Mean	61.17	Table 2-Daily nutrient intake by food source f...	US consumers aged 2 and above
4	Energy	FAFH: Fast food	Calories	Energy :FAFH: Fast food	1977-1978-Mean	110.45	Table 2-Daily nutrient intake by food source f...	US consumers aged 2 and above
...	...	...	...	...	...	...	...	...
					2017-2018-SE of		Table 2F3-Daily nutrient	Edu. - College

```
# Step 1: Drop rows with missing 'Value'
nut_intake_cleaned = nut_intake.dropna(subset=['Value']).copy()

# Step 2: Extract survey years and statistic type
nut_intake_cleaned[['Survey_Years', 'Statistic']] = nut_intake_cleaned['Survey years:Variable'].str.extract(r'(\d{4}-\d{4})-(\w+

# Step 3: Filter to only "Mean" values
nut_intake_mean = nut_intake_cleaned[nut_intake_cleaned['Statistic'].str.lower().str.strip() == 'mean'].copy()

# Step 4: Convert 'Value' to numeric
nut_intake_mean['Value'] = pd.to_numeric(nut_intake_mean['Value'], errors='coerce')
nut_intake_mean.dropna(subset=['Value'], inplace=True)

# Step 5: Split 'Nutrient:Food source' into 3 components
split_cols = nut_intake_mean['Nutrient:Food source'].str.split(':', n=2, expand=True)
nut_intake_mean['Group'] = split_cols[0].str.strip()       # Nutrient
nut_intake_mean['Source'] = split_cols[1].str.strip()
nut_intake_mean['Subsource'] = split_cols[2].str.strip() if split_cols.shape[1] > 2 else None

# Step 6: Map 'Source' values to full names
source_map = {
    'FAH': 'Food at Home',
    'FAFH': 'Food Away From Home',
    'Total': 'Total'
}
nut_intake_mean['Source'] = nut_intake_mean['Source'].map(source_map)

# Step 7: Expand multi-year ranges into individual years
def expand_years(row):
    try:
        start, end = map(int, row['Survey_Years'].split('-'))
        return pd.DataFrame({
            'Year': list(range(start, end + 1)),
            'Group': row['Group'],
            'Source': row['Source'],
            'Subsource': row['Subsource'],
            'Value': row['Value']
        })
    except:
        return pd.DataFrame()

# Step 8: Apply the transformation
nut_intake_long_final = pd.concat(
    [expand_years(row) for _, row in nut_intake_mean.iterrows()],
    ignore_index=True
)

# Optional: clean trailing whitespace
nut_intake_long_final['Subsource'] = nut_intake_long_final['Subsource'].str.strip()
nut_intake_long_final.sample(10)
```

	Year	Group	Source	Subsource	Value
37557	2018	Total Fat	Food Away From Home	Fast food	14.83
32226	1998	Saturated fatty acids	Food Away From Home	None	8.19
33375	2016	Calcium	Total	None	972.54
17913	2008	Protein	Food Away From Home	Fast food	16.59
9717	2006	Fatty acids, monounsaturated	Food Away From Home	Fast food	7.01
7650	2005	Cholesterol	Food Away From Home	Restaurant	18.71
39879	1996	Fatty acids, polyunsaturated	Food Away From Home	Others	1.48
8609	2018	Fiber, dietary	Food Away From Home	Others	0.84
18209	2012	Carbohydrate	Food Away From Home	None	96.17
14388	2015	Protein	Food Away From Home	None	28.19

```
# Pivot to wide format
nut_intake_wide_final = nut_intake_long_final.pivot_table(
    index="Year",
    columns=["Source", "Group"],  # Group = Nutrient
    values="Value"
)

# Flatten multi-index column names for ease of use
nut_intake_wide_final.columns = [
    f"{src} | {nutrient}" for src, nutrient in nut_intake_wide_final.columns
]
nut_intake_wide_final.reset_index(inplace=True)

# Preview result
nut_intake_wide_final.head()
```

	Year	Food Away From Home | Calcium	Food Away From Home | Carbohydrate	Food Away From Home | Cholesterol	Food Away From Home | Energy	Food Away From Home | Fatty acids, monounsaturated	Food Away From Home | Fatty acids, polyunsaturated	Food Away From Home | Fiber, dietary	Food Away From Home | Iron	Food Away From Home | Protein	...	T Chole
0	1977	55.403086	15.149753	22.018395	139.883827	2.262469	1.013086	0.841358	0.843827	5.850864	...	329
1	1978	55.403086	15.149753	22.018395	139.883827	2.262469	1.013086	0.841358	0.843827	5.850864	...	329
2	1989	72.255376	23.139032	28.831398	204.888817	3.372043	1.687849	1.276129	1.272581	8.119247	...	258
3	1990	72.255376	23.139032	28.831398	204.888817	3.372043	1.687849	1.276129	1.272581	8.119247	...	258
4	1991	72.255376	23.139032	28.831398	204.888817	3.372043	1.687849	1.276129	1.272581	8.119247	...	258

5 rows × 37 columns

Cleaning the Food Group Intake Dataset

This dataset originally came in a wide format, with year blocks spread across multiple columns. To make the data tidy and analysis-ready, we:

- Dropped rows with missing values
- Split combined columns into `Survey_Years` and `Statistic` (e.g., `2005–2006–Mean`)
- Filtered only for "Mean" intake values
- Converted numeric strings to float
- Expanded year ranges into individual rows per year
- Produced a long-format table: one row per food group, source, and year

```
# Step 1: Drop rows with missing 'Value'
food_group_cleaned = food_group.dropna(subset=['Value']).copy()

# Step 2: Extract survey years and statistic type
food_group_cleaned[['Survey_Years', 'Statistic']] = food_group_cleaned['Survey years:Variable'].str.extract(r'(\d{4}–\d{4})–(\w+

# Step 3: Filter to only "Mean" values
food_group_mean = food_group_cleaned[food_group_cleaned['Statistic'].str.lower().str.strip() == 'mean'].copy()

# Step 4: Convert 'Value' to numeric
food_group_mean['Value'] = pd.to_numeric(food_group_mean['Value'], errors='coerce')
```

```
food_group_mean.dropna(subset=['Value'], inplace=True)

# Step 5: Split 'Food group:Food source' into 3 components
split_cols = food_group_mean['Food group:Food source'].str.split(':', n=2, expand=True)
food_group_mean['Group'] = split_cols[0]
food_group_mean['Source'] = split_cols[1]
food_group_mean['Subsource'] = split_cols[2] if split_cols.shape[1] > 2 else None
# Map 'Source' values to full names
source_map = {
    'FAH': 'Food at Home',
    'FAFH': 'Food Away From Home',
    'Total': 'Total'
}
food_group_mean['Source'] = food_group_mean['Source'].map(source_map)

# Step 6: Expand multi-year ranges into individual years
def expand_years(row):
    try:
        start, end = map(int, row['Survey_Years'].split('-'))
        return pd.DataFrame({
            'Year': list(range(start, end + 1)),
            'Group': row['Group'],
            'Source': row['Source'],
            'Subsource': row['Subsource'],
            'Value': row['Value']
        })
    except:
        return pd.DataFrame()

# Step 7: Apply the transformation
food_group_long_final = pd.concat(
    [expand_years(row) for _, row in food_group_mean.iterrows()],
    ignore_index=True
)
food_group_mean['Subsource'] = (
    split_cols[2].str.strip() if split_cols.shape[1] > 2 else None
)
food_group_long_final.sample(10)
```

	Year	Group	Source	Subsource	Value
20794	1990	Protein foods, low Omega-3 fatty fish	Food at Home	None	0.19
16315	2006	Fruit, whole fruit	Food Away From Home	None	0.06
94529	2018	Protein foods, nuts and seeds	Food Away From Home	Restaurant	0.01
108888	1994	Vegetable, others	Total	None	0.53
65331	2008	Added sugars	Food Away From Home	Others	1.11
[illegible]	[illegible]	Vegetable, total	Food Away From Home	Fast food	[illegible]
6313	2018	Vegetable, red and orange	Food Away From Home	Others	0.01
74544	2017	Fruit, juice	Total	None	0.22
82008	1989	Discretionary fats	Food Away From Home	Restaurant	3.67
63131	1994	Discretionary fats and oils	Food Away From Home	School	2.56

```
# Pivot to wide format with Subsource included
food_group_wide_final = food_group_long_final.pivot_table(
    index="Year",
    columns=["Source", "Subsource", "Group"],
    values="Value"
)

# Flatten multi-index column names
food_group_wide_final.columns = [
    " | ".join(filter(None, [src, subsrc, grp])) for src, subsrc, grp in food_group_wide_final.columns
]
food_group_wide_final.reset_index(inplace=True)

food_group_wide_final.sample(10)
```

	Year	Food Away From Home | Fast food | Added sugars	Food Away From Home | Fast food | Dairy, cheese	Food Away From Home | Fast food | Dairy, fluid milk	Food Away From Home | Fast food | Dairy, total	Food Away From Home | Fast food | Dairy, yogurt	Food Away From Home | Fast food | Discretionary fats	Food Away From Home | Fast food | Discretionary fats and oils	Food Away From Home | Fast food | Discretionary oils	Food Away From Home | Fast food | Energy	...	Food Away From Home | School | Protein foods, poultry	Ho Scho Pro fo prod
12	2005	2.690500	0.187000	0.046500	0.235000	0.0000	7.786000	11.968500	4.184000	347.324000	...	0.049231	0.00
1	1978	1.206471	0.023529	0.038824	0.064118	0.0000	2.915294	3.449412	0.532941	102.594118	...	0.035385	0.00
24	2017	2.465500	0.189000	0.060000	0.252000	0.0015	6.560500	12.476000	5.913000	354.105000	...	0.027692	0.00
25	2018	2.465500	0.189000	0.060000	0.252000	0.0015	6.560500	12.476000	5.913000	354.105000	...	0.027692	0.00
4	1991	2.334500	0.084000	0.060500	0.144500	0.0000	6.839500	8.607500	1.768500	252.475000	...	0.019231	0.00
14	2007	2.492500	0.175000	0.034000	0.212000	0.0000	6.973000	10.772500	3.801000	306.063000	...	0.058462	0.00
6	1995	2.877000	0.114500	0.035000	0.151000	0.0000	7.641500	9.419000	1.778000	272.560000	...	0.028462	0.00
5	1994	2.877000	0.114500	0.035000	0.151000	0.0000	7.641500	9.419000	1.778000	272.560000	...	0.028462	0.00
13	2006	2.690500	0.187000	0.046500	0.235000	0.0000	7.786000	11.968500	4.184000	347.324000	...	0.049231	0.00
10	2003	3.189500	0.218500	0.036500	0.256000	0.0005	10.220000	13.390000	3.169500	376.790500	...	0.035385	0.00

10 rows × 145 columns

Cleaning the Food Density Dataset

This dataset measures how much of each food group was consumed per 1,000 calories. Like other intake tables, it needed to be reshaped into tidy format.

Steps:

- Dropped rows with missing values
- Extracted `Survey_Years` and `Statistic` fields
- Filtered to include only "Mean" values
- Converted values to numeric
- Expanded year ranges (e.g., `1999–2000`) into individual years
- Produced both long and wide versions for analysis

```
# Step 1: Drop rows with missing 'Value'
density_cleaned = density.dropna(subset=['Value']).copy()

# Step 2: Extract survey years and statistic type
density_cleaned[['Survey_Years', 'Statistic']] = density_cleaned['Survey years:Variable'].str.extract(r'(\d{4}-\d{4})-(\w+)', ex

# Step 3: Filter to only "Mean" values
density_mean = density_cleaned[density_cleaned['Statistic'].str.lower().str.strip() == 'mean'].copy()

# Step 4: Convert 'Value' to numeric
density_mean['Value'] = pd.to_numeric(density_mean['Value'], errors='coerce')
density_mean.dropna(subset=['Value'], inplace=True)

# Step 5: Split 'Food group:Food source' into components
split_cols = density_mean['Food group:Food source'].str.split(':', n=2, expand=True)
density_mean['Group'] = split_cols[0].str.strip()
density_mean['Source'] = split_cols[1].str.strip()
density_mean['Subsource'] = split_cols[2].str.strip() if split_cols.shape[1] > 2 else None

# Step 6: Map source codes to full names
source_map = {
    'FAH': 'Food at Home',
    'FAFH': 'Food Away From Home',
    'Total': 'Total'
}
density_mean['Source'] = density_mean['Source'].map(source_map)

# Step 7: Expand multi-year ranges into individual years
def expand_years(row):
    try:
        start, end = map(int, row['Survey_Years'].split('-'))
        return pd.DataFrame({
            'Year': list(range(start, end + 1)),
```

```
            'Group': row['Group'],
            'Source': row['Source'],
            'Subsource': row['Subsource'],
            'Value': row['Value']
        })
    except:
        return pd.DataFrame()

# Step 8: Apply transformation
density_long_final = pd.concat(
    [expand_years(row) for _, row in density_mean.iterrows()],
    ignore_index=True
)

# Clean trailing whitespace
density_long_final['Subsource'] = density_long_final['Subsource'].str.strip()
```

```
density_long_final.sample(10)
```

	Year	Group	Source	Subsource	Value
69223	1990	Vegetable, red and orange	Food Away From Home	Fast food	0.02
53797	2008	Vegetable, dark green	Total	None	0.06
68782	1977	Protein foods, organ meats	Food Away From Home	Restaurant	0.07
61744	2017	Protein foods, eggs	Food Away From Home	None	0.26
30863	2014	Protein foods, meats (beef, veal, pork, lamb, ...	Food Away From Home	Others	0.46
77934	2005	Discretionary fats and oils	Food at Home	None	25.53
102069	1989	Grains, total	Food Away From Home	School	2.50
7498	1990	Legumes	Total	None	0.05
21497	1995	Vegetables, other starchy	Food Away From Home	Fast food	0.01
71387	2006	Discretionary fats and oils	Food Away From Home	Restaurant	34.74

```
# Pivot to wide format
density_wide_final = density_long_final.pivot_table(
    index="Year",
    columns=["Source", "Group"],  # Group = Food Group
    values="Value"
)

# Flatten multi-index column names for ease of use
density_wide_final.columns = [
    f"{src} | {group}" for src, group in density_wide_final.columns
]
density_wide_final.reset_index(inplace=True)

# Preview result
density_wide_final.head()
```

	Year	Food Away From Home | Added sugars	Food Away From Home | Dairy, cheese	Food Away From Home | Dairy, fluid milk	Food Away From Home | Dairy, total	Food Away From Home | Dairy, yogurt	Food Away From Home | Discretionary fats	Food Away From Home | Discretionary fats and oils	Food Away From Home | Discretionary oils	Food Away From Home | Energy	...	Total | Protein foods, poultry	To Pr f pro
0	1977	10.863580	0.180741	0.730370	0.915926	0.004074	26.880000	31.645185	4.765432	139.883827	...	0.595882	0.0
1	1978	10.863580	0.180741	0.730370	0.915926	0.004074	26.880000	31.645185	4.765432	139.883827	...	0.595882	0.0
2	1989	9.974946	0.251935	0.525806	0.787957	0.009247	25.427742	31.506882	6.078710	204.888817	...	0.694000	0.0
3	1990	9.974946	0.251935	0.525806	0.787957	0.009247	25.427742	31.506882	6.078710	204.888817	...	0.694000	0.0
4	1991	9.974946	0.251935	0.525806	0.787957	0.009247	25.427742	31.506882	6.078710	204.888817	...	0.694000	0.0

5 rows × 109 columns

Cleaning the Recommended Food Density Dataset

This dataset provides USDA-recommended nutrient and food group intake per 1,000 calories. Cleaning steps included:

- Dropped irrelevant columns
- Converted numeric strings to actual numbers
- Filtered only rows containing USDA recommendations (not actual intake)
- Renamed key columns for clarity and SQL compatibility

```
#drop "table" column
rec_density = rec_density.drop(columns=['Table'])
```

```
#string to numeric values
cols = ['Value']
rec_density[cols] = pd.to_numeric(rec_density[cols].stack(), errors='coerce').unstack()
```

Next, we looked at the variables and determined that we only needed the recommended food density from this table since the other information (actual food densities) was available in our Food Group Density data set so we filtered for only observations with the recommended food density amounts.

```
# look at unique values of "variable"
unique_values = rec_density['Variable'].unique()
unique_values
```

```
array(['Recommended density*:Nutrient or food group amount per 1,000 calories',
       '2017-2018 Actual density-Total-Nutrient or food group amount per 1,000 calories ',
       '2017-2018 Actual density-FAH-Nutrient or food group amount per 1,000 calories ',
       '2017-2018 Actual density-FAFH-Nutrient or food group amount per 1,000 calories ',
       '2017-2018 Actual density-Restaurant-Nutrient or food group amount per 1,000 calories ',
       '2017-2018 Actual density-Fast food-Nutrient or food group amount per 1,000 calories ',
       '2017-2018 Actual density-School-Nutrient or food group amount per 1,000 calories ',
       '2017-2018 density as a ratio of the recommended density-Total-Ratio of actual density to the recommended density',
       '2017-2018 density as a ratio of the recommended density-FAH-Ratio of actual density to the recommended density',
       '2017-2018 density as a ratio of the recommended density-FAFH-Ratio of actual density to the recommended density',
       '2017-2018 density as a ratio of the recommended density-Restaurant-Ratio of actual density to the recommended
density',
       '2017-2018 density as a ratio of the recommended density-Fast food-Ratio of actual density to the recommended
density',
       '2017-2018 density as a ratio of the recommended density-School-Ratio of actual density to the recommended
density'],
      dtype=object)
```

```
#filtered for observations that only include recommended density of food
rec_density_final = rec_density[rec_density['Variable'].isin(['Recommended density*:Nutrient or food group amount per 1,000 calc
```

```
# drop the 'variable' column
rec_density_final = rec_density_final.drop(columns=['Variable'])
```

```
#rename value column
rec_density_final = rec_density_final.rename(columns = {'Value': 'RecDensity_per1000cal'})
```

SQL Database

```
conn = sqlite3.connect("final.db")
c = conn.cursor()
```

```
# Run create table sql query
c.execute("""CREATE TABLE IF NOT EXISTS CPI_final
             (Homecooked_or_Takeout, Food Category, Attribute,
       Percent Change)""")
```

```
<sqlite3.Cursor at 0x7895ce2ff040>
```

```
CPI_final.to_sql('final', conn, if_exists='replace', index=False)
```

```
209
```

```
tables = {
    'nut_intake_long_final': nut_intake_long_final,
    'nut_intake_wide_final': nut_intake_wide_final,
    'food_group_long_final': food_group_long_final,
    'food_group_wide_final': food_group_wide_final,
```

```
    'density_long_final': density_long_final,
    'density_wide_final': density_wide_final,
    'rec_density_final': rec_density_final,
    'sales_final': sales_final,
    'sales_percapita_final': sales_percapita_final,
    'sales_NTT_final': sales_NTT_final,
    'sales_percapNTT_final': sales_percapNTT_final,
    'size_final': size_final
}

for name, df in tables.items():
    df.to_sql(name, conn, if_exists='replace', index=False)
```

3. Exploratory Data Analysis (EDA)

This section explores high-level trends and patterns in food consumption, nutrient intake, and sales. We use descriptive statistics, group summaries, and plots to highlight insights that inform our modeling and interpretation in later sections.

Table 1: Intake by Food Group

This table aggregates total intake by food group and calculates each group's share of total intake across all sources and years. It identifies which food groups dominate the American diet and supports comparison of dietary patterns at a high level.

```
pd.read_sql_query("SELECT * FROM food_group_long_final LIMIT 5;", conn)
```

	Year	Group	Source	Subsource	Value
0	1977	Energy	Total	None	1806.88
1	1978	Energy	Total	None	1806.88
2	1977	Energy	Food at Home	None	1462.27
3	1978	Energy	Food at Home	None	1462.27
4	1977	Energy	Food Away From Home	None	344.61

```
query = """
SELECT DISTINCT "Group"
FROM food_group_long_final
ORDER BY "Group"
"""

food_groups = pd.read_sql_query(query, conn)
food_groups
```

	Group
0	Added sugars
1	Dairy, cheese
2	Dairy, fluid milk
3	Dairy, total
4	Dairy, yogurt
5	Discretionary fats
6	Discretionary fats and oils
7	Discretionary oils
8	Energy
9	Fruit, citrus, melon, and berries
10	Fruit, juice
11	Fruit, other
12	Fruit, total
13	Fruit, whole fruit
14	Grains, non-whole grains
15	Grains, total
16	Grains, whole grains
17	Legumes
18	Protein foods, cured meat
19	Protein foods, eggs
20	Protein foods, high Omega-3 fatty fish
21	Protein foods, low Omega-3 fatty fish
22	Protein foods, meats (beef, veal, pork, lamb, ...
23	Protein foods, meats, poultry, and fish
24	Protein foods, nuts and seeds
25	Protein foods, organ meats
26	Protein foods, poultry
27	Protein foods, soy products
28	Protein foods, total
29	Vegetable, dark green
30	Vegetable, others

```
query = """
SELECT
    Year,
    "Group" AS Food_Group,
    SUM(Value) AS Total_Value
FROM food_group_long_final
WHERE Source = 'Total'
  AND "Group" IN (
    'Dairy, total',
    'Fruit, total',
    'Grains, whole grains',
    'Grains, non-whole grains',
    'Legumes',
    'Protein foods, total',
    'Vegetable, total'
)
GROUP BY Year, "Group"
ORDER BY Year, Total_Value DESC
"""

totals_by_group = pd.read_sql_query(query, conn)
totals_by_group.sample(10)
```

	Year	Food_Group	Total_Value
106	2008	Protein foods, total	109.22
139	2012	Legumes	2.44
172	2017	Fruit, total	19.15
126	2011	Grains, non-whole grains	117.10
75	2003	Grains, whole grains	11.40
23	1990	Dairy, total	31.27
56	1997	Grains, non-whole grains	117.77
68	1998	Grains, whole grains	14.23
173	2017	Grains, whole grains	16.64
171	2017	Vegetable, total	26.87

```
# Filter and prepare data
earliest_year = totals_by_group['Year'].min()
latest_year = totals_by_group['Year'].max()

# Pull data for the two years and normalize to % of total
data_compare = totals_by_group[totals_by_group['Year'].isin([earliest_year, latest_year])].copy()
data_compare['Share'] = data_compare.groupby('Year')['Total_Value'].transform(lambda x: x / x.sum() * 100)

# Pivot for plotting
pivot_df = data_compare.pivot(index='Food_Group', columns='Year', values='Share').fillna(0)

# Sort by 2018 values for consistent visual
pivot_df = pivot_df.sort_values(by=latest_year)

# Plot
plt.figure(figsize=(10, 7))
pivot_df[[latest_year, earliest_year]].plot(
    kind='barh',
    width=0.7,
    figsize=(12, 8),
    color=['#4C72B0', '#DD8452']
)
plt.xlabel("Share of Total Intake (%)")
plt.ylabel("Food Group")
plt.title(f"Diet Composition: {earliest_year} vs. {latest_year}")
plt.legend([f"{earliest_year}", f"{latest_year}"], title="Year")
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

<Figure size 1000x700 with 0 Axes>

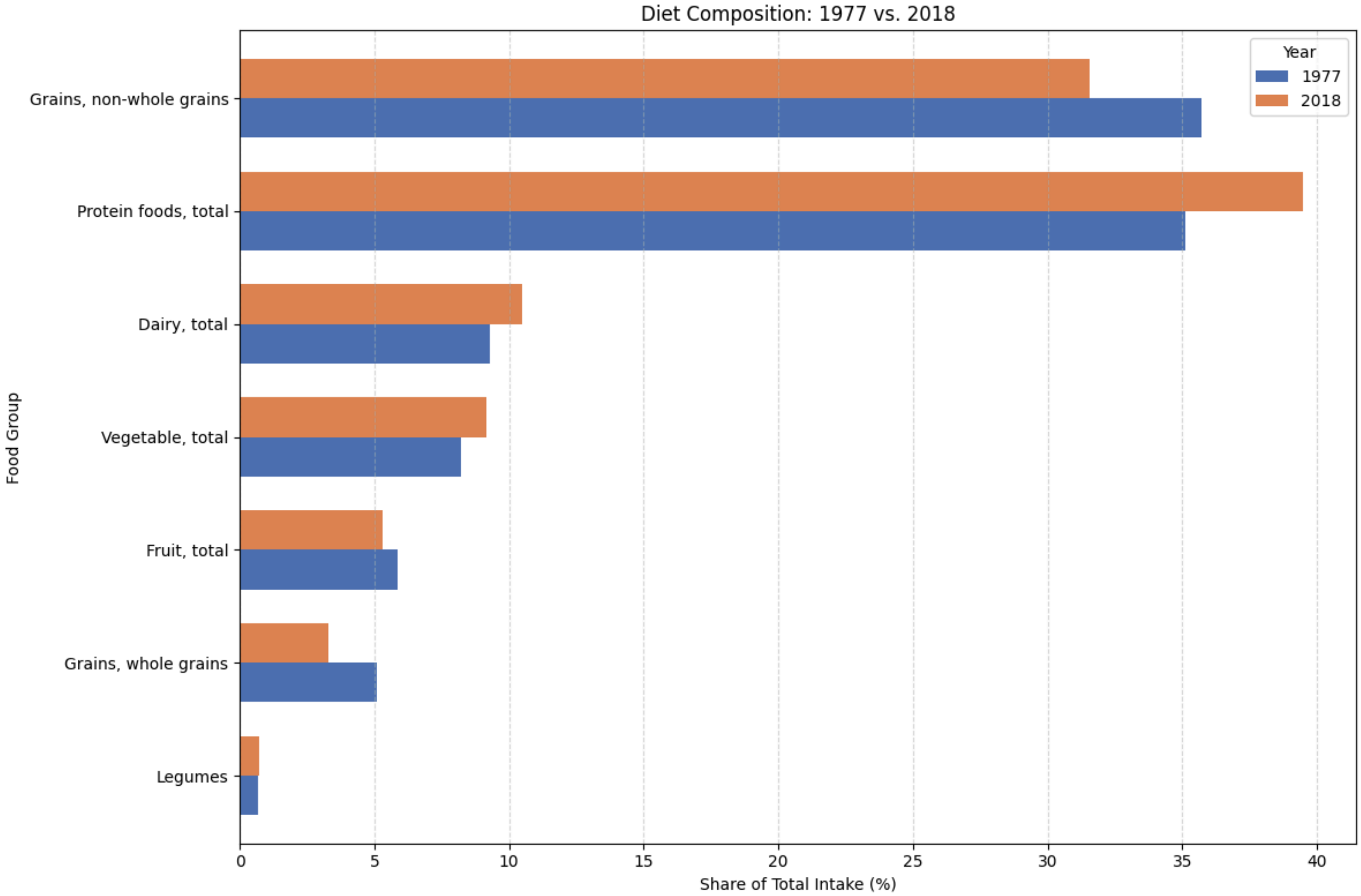


Table 2: Sales by Region

This table summarizes annual food sales across U.S. regions, including food at home (FAH) and food away from home (FAFH). It helps assess regional spending trends and differences in consumer behavior over time.

```
# defining states in each region
NE = ['Connecticut', 'Maine', 'Massachusetts','New Hampshire','New Jersey', 'New York',
      'Pennsylvania','Rhode Island', 'Vermont']
MW = ['Illinois', 'Indiana', 'Iowa', 'Kansas', 'Michigan', 'Minnesota', 'Missouri',
      'Nebraska', 'North Dakota', 'Ohio', 'South Dakota', 'Wisconsin']
SO = ['Alabama', 'Arkansas', 'Delaware', 'District of Columbia', 'Florida', 'Georgia',
      'Kentucky', 'Louisiana', 'Maryland', 'Mississippi', 'North Carolina', 'Oklahoma',
      'South Carolina', 'Tennessee', 'Texas', 'Virginia', 'West Virginia']
WE = ['Alaska', 'Arizona', 'California', 'Colorado', 'Hawaii', 'Idaho', 'Montana',
      'Nevada', 'New Mexico', 'Oregon', 'Utah', 'Washington', 'Wyoming']
# making a copy
sales_table = sales_final.copy()

#assign regions values
sales_table['Region'] = 'West'

sales_table.loc[sales_table['State'].isin(NE), 'Region'] = 'Northeast'
sales_table.loc[sales_table['State'].isin(MW), 'Region'] = 'Midwest'
sales_table.loc[sales_table['State'].isin(SO), 'Region'] = 'South'


# Grouping and summing FAH and FAFH sales in nominal dollars by region
grouped_salest2 = sales_table.groupby(['Year', 'Region'])[
    ['FAH sales million nominal U.S. dollars with taxes and tips',
     'FAFH sales million nominal U.S. dollars with taxes and tips']
].sum().reset_index()
```

```
# Optional: Rename columns for readability
grouped_salest2.columns =['Year', 'Region', 'FAH Sales (Nominal $M)', 'FAFH Sales (Nominal $M)']

# Display the result
print(grouped_salest2)
```

```
     Year     Region  FAH Sales (Nominal $M)  FAFH Sales (Nominal $M)
0    1997    Midwest                84211.93                 62835.62
1    1997  Northeast                70359.12                 52124.59
2    1997      South               134202.41                 99944.66
3    1997       West                86259.38                 69004.36
4    1998    Midwest                85663.43                 65617.76
..    ...        ...                     ...                      ...
103  2022       West               284692.24                332132.93
104  2023    Midwest               192202.94                223858.30
105  2023  Northeast               169898.57                229296.24
106  2023      South               410956.38                487555.32
107  2023       West               291693.39                373124.18

[108 rows x 4 columns]
```

```
# adding total sales column
grouped_salest2['Total Sales (Nominal $M)'] = (
    grouped_salest2['FAH Sales (Nominal $M)'] + grouped_salest2['FAFH Sales (Nominal $M)']
)
print(grouped_salest2)
```

```
     Year     Region  FAH Sales (Nominal $M)  FAFH Sales (Nominal $M)  \
0    1997    Midwest                84211.93                 62835.62
1    1997  Northeast                70359.12                 52124.59
2    1997      South               134202.41                 99944.66
3    1997       West                86259.38                 69004.36
4    1998    Midwest                85663.43                 65617.76
..    ...        ...                     ...                      ...
103  2022       West               284692.24                332132.93
104  2023    Midwest               192202.94                223858.30
105  2023  Northeast               169898.57                229296.24
106  2023      South               410956.38                487555.32
107  2023       West               291693.39                373124.18

     Total Sales (Nominal $M)
0                   147047.55
1                   122483.71
2                   234147.07
3                   155263.74
4                   151281.19
..                        ...
103                 616825.17
104                 416061.24
105                 399194.81
106                 898511.70
107                 664817.57

[108 rows x 5 columns]
```

```
plt.figure(figsize=(10, 6))
sns.lineplot(
    data=grouped_salest2,
    x='Year',
    y='Total Sales (Nominal $M)',
    hue='Region',
    marker='o'
)
plt.title("Total Food Sales by Region (Nominal USD)", fontsize=14)
plt.xlabel("Year", fontsize=12)
plt.ylabel("Total Sales (in Millions)", fontsize=12)
plt.legend(title='Region')
plt.grid(True)
plt.tight_layout()
plt.savefig("regional_sales_trend.png", dpi=300)
plt.show()
```

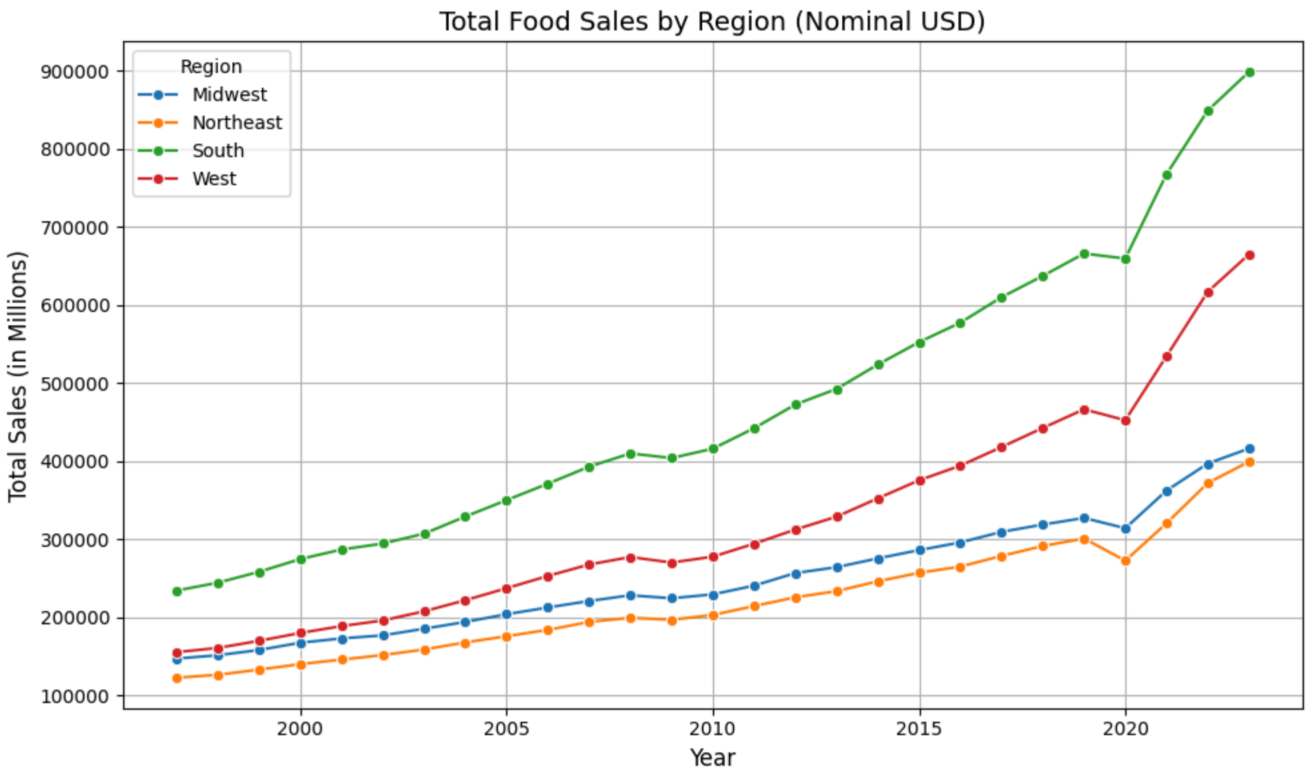


Table 3: Nutrient Intake Trends (2007–2017)

This table compares average daily intake of key nutrients between 2007 and 2017. It highlights shifts in dietary composition over the decade and helps identify increasing or declining nutrient trends.

```
query = """WITH filtered AS (
    SELECT * FROM nut_intake_wide_final WHERE Year IN (2007, 2017)
),
long_format AS (
    SELECT Year, 'Energy (kcal)' AS Nutrient, "Total | Energy" AS Value FROM filtered
    UNION ALL SELECT Year, 'Total Fat (%)', "Total | Total Fat" FROM filtered
    UNION ALL SELECT Year, 'Saturated Fat (%)', "Total | Saturated fatty acids" FROM filtered
    UNION ALL SELECT Year, 'Fiber (g)', "Total | Fiber, dietary" FROM filtered
    UNION ALL SELECT Year, 'Protein (g)', "Total | Protein" FROM filtered
    UNION ALL SELECT Year, 'Calcium (mg)', "Total | Calcium" FROM filtered
    UNION ALL SELECT Year, 'Iron (mg)', "Total | Iron" FROM filtered
    UNION ALL SELECT Year, 'Sodium (mg)', "Total | Sodium" FROM filtered
),
pivoted AS (
    SELECT
        Nutrient,
        MAX(CASE WHEN Year = 2007 THEN Value END) AS Intake_2007,
        MAX(CASE WHEN Year = 2017 THEN Value END) AS Intake_2017
    FROM long_format
    GROUP BY Nutrient
)
SELECT
    Nutrient,
    ROUND(Intake_2007, 2) AS Intake_2007,
    ROUND(Intake_2017, 2) AS Intake_2017,
    ROUND((Intake_2017 - Intake_2007) * 100.0 / Intake_2007, 2) AS Percent_Change
FROM pivoted
ORDER BY Percent_Change DESC;
"""
table3_sql_result = pd.read_sql(query, conn)

table3_sql_result
```

	Nutrient	Intake_2007	Intake_2017	Percent_Change
0	Total Fat (%)	76.49	83.75	9.50
1	Fiber (g)	14.92	16.13	8.17
2	Saturated Fat (%)	25.69	27.61	7.48
3	Calcium (mg)	923.30	954.39	3.37
4	Energy (kcal)	2031.14	2074.63	2.14
5	Protein (g)	76.58	77.23	0.85
6	Sodium (mg)	3412.38	3348.33	-1.88
7	Iron (mg)	14.53	13.93	-4.10

```
# Plot
sns.barplot(
    data=table3_sql_result,
    x='Percent_Change',
    y='Nutrient',
    hue='Nutrient',
    palette='coolwarm',
    dodge=False,
    legend=False
)

plt.title("Percent Change in Nutrient Intake (2007 to 2017)", fontsize=14)
plt.xlabel("Percent Change")
plt.ylabel("Nutrient")
plt.axvline(0, color='black', linestyle='--')
plt.grid(True)
plt.tight_layout()
plt.savefig("chart_nutrient_change_2007_2017.png", dpi=300)
plt.show()
```

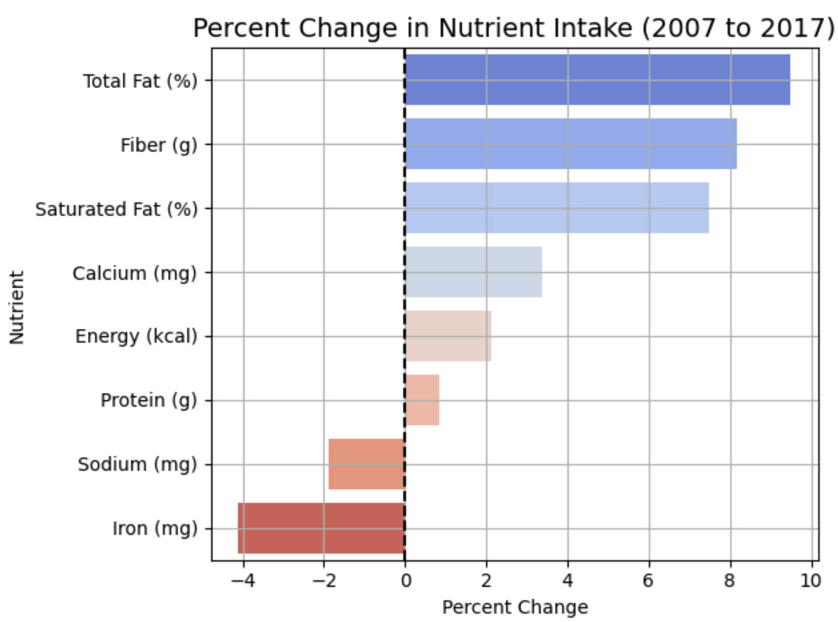


Table 4: Actual vs Recommended Densities

This table compares U.S. nutrient intake per 1,000 calories to USDA-recommended targets. It includes actual values, deltas, and percent deviation over time for core nutrients like fiber, sodium, and saturated fat — helping assess over- or under-consumption.

```
query = """
WITH nutrient_map AS (
    SELECT 'Fiber (g)' AS RecLabel, 'Total | Fiber, dietary' AS ActualColumn
    UNION ALL SELECT 'Saturated fats (percent of calories)**', 'Total | Saturated fatty acids'
    UNION ALL SELECT 'Calcium (mg)', 'Total | Calcium'
    UNION ALL SELECT 'Iron (mg)', 'Total | Iron'
```

```
    UNION ALL SELECT 'Sodium (mg)', 'Total | Sodium'
),
base AS (
    SELECT
        r."Nutrient or Food group" AS Nutrient,
        r.RecDensity_per1000cal AS Recommended,
        n.Year,
        CASE
            WHEN r."Nutrient or Food group" = 'Fiber (g)' THEN n."Total | Fiber, dietary"
            WHEN r."Nutrient or Food group" = 'Saturated fats (percent of calories)**' THEN n."Total | Saturated fatty acids"
            WHEN r."Nutrient or Food group" = 'Calcium (mg)' THEN n."Total | Calcium"
            WHEN r."Nutrient or Food group" = 'Iron (mg)' THEN n."Total | Iron"
            WHEN r."Nutrient or Food group" = 'Sodium (mg)' THEN n."Total | Sodium"
        END AS Actual
    FROM rec_density_final r
    JOIN nutrient_map m ON r."Nutrient or Food group" = m.RecLabel
    JOIN nut_intake_wide_final n
),
final AS (
    SELECT
        Nutrient,
        Year,
        ROUND(Recommended, 2) AS Recommended,
        ROUND(Actual, 2) AS Actual,
        ROUND(Actual - Recommended, 2) AS Delta,
        ROUND((Actual - Recommended) * 100.0 / Recommended, 2) AS Percent_Deviation
    FROM base
    WHERE Actual IS NOT NULL
)
SELECT * FROM final
ORDER BY Nutrient, Year;
"""

# Run SQL and store result
table4_sql_result = pd.read_sql(query, conn)
table4_sql_result.head()
```

	Nutrient	Year	Recommended	Actual	Delta	Percent_Deviation
0	Calcium (mg)	1977	500.0	763.77	263.77	52.75
1	Calcium (mg)	1978	500.0	763.77	263.77	52.75
2	Calcium (mg)	1989	500.0	755.22	255.22	51.04
3	Calcium (mg)	1990	500.0	755.22	255.22	51.04
4	Calcium (mg)	1991	500.0	755.22	255.22	51.04

SQL Code for Tables

```
# Prepare a simplified sales table for SQL export (removes constant dollar columns)
q1_sales_table = sales_table.copy(deep=True)
q1_sales_table = q1_sales_table.drop(columns=[
    'FAH sales million constant 1988 U.S. dollars with taxes and tips',
    'FAFH sales million constant 1988 U.S. dollars with taxes and tips',
    'Total sales million constant 1988 U.S. dollars with taxes and tips'
])
q1_sales_table.columns = [
    'Year', 'State', 'FAH Sales (Nominal $M)',
    'FAFH Sales (Nominal $M)', 'Total Sales (Nominal $M)', 'Region'
]

# Add summary tables to export list
tables.update({
    'grouped_salest2': grouped_salest2,
    'q1_sales_table': q1_sales_table,
    'table3_sql_result': table3_sql_result,
    'table4_sql_result': table4_sql_result
})

# Export all tables to SQLite
for name, df in tables.items():
    df.to_sql(name, conn, if_exists='replace', index=False)
```

State Share of Total Expenditure per Region

This query provides the share of total expenditures from each state of a region. The code takes the sum of "Total Sales" of all the states in each region then calculates the share expenditures for each state. This is ordered by region and total sales amount so users can see the states with the largest share of expenditures by region.

```
query=""" SELECT *,
            "Total Sales (Nominal $M)"/sum("Total Sales (Nominal $M)") OVER (PARTITION BY Region) AS Share_Total_Sales
            FROM q1_sales_table
            WHERE Year == 2023
            ORDER BY Region, "Total Sales (Nominal $M)" DESC;
"""

max_statesales = pd.read_sql(query, conn)
max_statesales
```

	Year	State	FAH Sales (Nominal $M)	FAFH Sales (Nominal $M)	Total Sales (Nominal $M)	Region	Share_Total_Sales
0	2023	Illinois	31607.92	47590.62	79198.54	Midwest	0.190353
1	2023	Ohio	32550.73	39170.16	71720.89	Midwest	0.172381
2	2023	Michigan	29275.19	29591.38	58866.57	Midwest	0.141485
3	2023	Indiana	18526.06	21727.52	40253.58	Midwest	0.096749
4	2023	Missouri	17417.12	20312.24	37729.37	Midwest	0.090682
5	2023	Minnesota	15336.33	18962.58	34298.91	Midwest	0.082437
6	2023	Wisconsin	17198.63	16978.36	34177.00	Midwest	0.082144
7	2023	Iowa	10086.13	8945.12	19031.24	Midwest	0.045741
8	2023	Kansas	9205.34	9003.01	18208.35	Midwest	0.043764
9	2023	Nebraska	6089.59	6387.32	12476.91	Midwest	0.029988
10	2023	South Dakota	2631.97	2865.92	5497.89	Midwest	0.013214
11	2023	North Dakota	2277.93	2324.07	4602.00	Midwest	0.011061
12	2023	New York	56701.58	87901.18	144602.77	Northeast	0.362236
13	2023	Pennsylvania	37821.57	40430.96	78252.53	Northeast	0.196026
14	2023	New Jersey	26251.61	35500.04	61751.65	Northeast	0.154691
15	2023	Massachusetts	22230.43	33013.25	55243.68	Northeast	0.138388
16	2023	Connecticut	10531.23	13878.26	24409.48	Northeast	0.061147
17	2023	New Hampshire	5337.49	5951.72	11289.21	Northeast	0.028280
18	2023	Maine	5461.52	5440.01	10901.53	Northeast	0.027309
19	2023	Rhode Island	3467.07	4907.45	8374.52	Northeast	0.020979
20	2023	Vermont	2096.07	2273.37	4369.43	Northeast	0.010946
21	2023	Texas	100304.68	121880.75	222185.44	South	0.247282
22	2023	Florida	75983.09	95626.28	171609.38	South	0.190993
23	2023	Georgia	35109.47	39627.44	74736.91	South	0.083179
24	2023	North Carolina	33857.31	39120.07	72977.38	South	0.081220
25	2023	Virginia	28570.15	30885.90	59456.05	South	0.066172
26	2023	Tennessee	22376.02	26798.71	49174.73	South	0.054729
27	2023	Maryland	19270.43	21953.07	41223.50	South	0.045880
28	2023	South Carolina	15687.11	20416.25	36103.36	South	0.040181
29	2023	Louisiana	14123.52	16186.11	30309.63	South	0.033733
30	2023	Alabama	14488.29	14944.97	29433.26	South	0.032758
31	2023	Kentucky	12823.98	14259.95	27083.93	South	0.030143
32	2023	Oklahoma	11129.55	12697.43	23826.97	South	0.026518
33	2023	Arkansas	9301.51	8885.27	18186.78	South	0.020241
34	2023	Mississippi	8175.85	8613.45	16789.30	South	0.018686

Chart 1: Fatty Acid Trends

This chart shows trends in fatty acid intake over time. Saturated fat intake has remained relatively steady, while polyunsaturated fat has gradually increased (might reflect a shift towards healthier fat sources in the American diet). Monounsaturated fat intake peaked around

2006 and then slightly declined. Monounsaturated and saturated fat intake follow a similar trend likely because they frequently co-occur in the same foods, such as meat, dairy, and cooking oils. As a result, shifts in consumption of those foods tend to affect both fat types in parallel.

35	2023	District of Columbia	2252.62	7226.97	9479.59	South	0.010550
36	2023	West Virginia	4431.08	4596.70	9027.79	South	0.010047

```
# Nutrients with meaningful values
fatty_acid_cols = [
    "Total | Fatty acids, monounsaturated",
    "Total | Fatty acids, polyunsaturated",
    "Total | Saturated fatty acids"
]


# Extract Year + selected nutrients
chart1 = nut_intake_wide_final[["Year"] + fatty_acid_cols].copy()

# Plot
plt.figure(figsize=(10, 6))
for nutrient in fatty_acid_cols:
    label = nutrient.replace("Total ", "").strip()
    plt.plot(chart1["Year"], chart1[nutrient], marker='o', label=label)

# Embellishments
plt.title("Fatty Acid Intake Over Time", fontsize=14)
plt.xlabel("Year", fontsize=12)
plt.ylabel("Total Intake (grams or equivalent units)", fontsize=12)
plt.legend(title="Fatty Acid Type")
plt.grid(True)
plt.tight_layout()

# Save to file
plt.savefig("fatty_acids.png", dpi=300)

# Show on screen
plt.show()
```

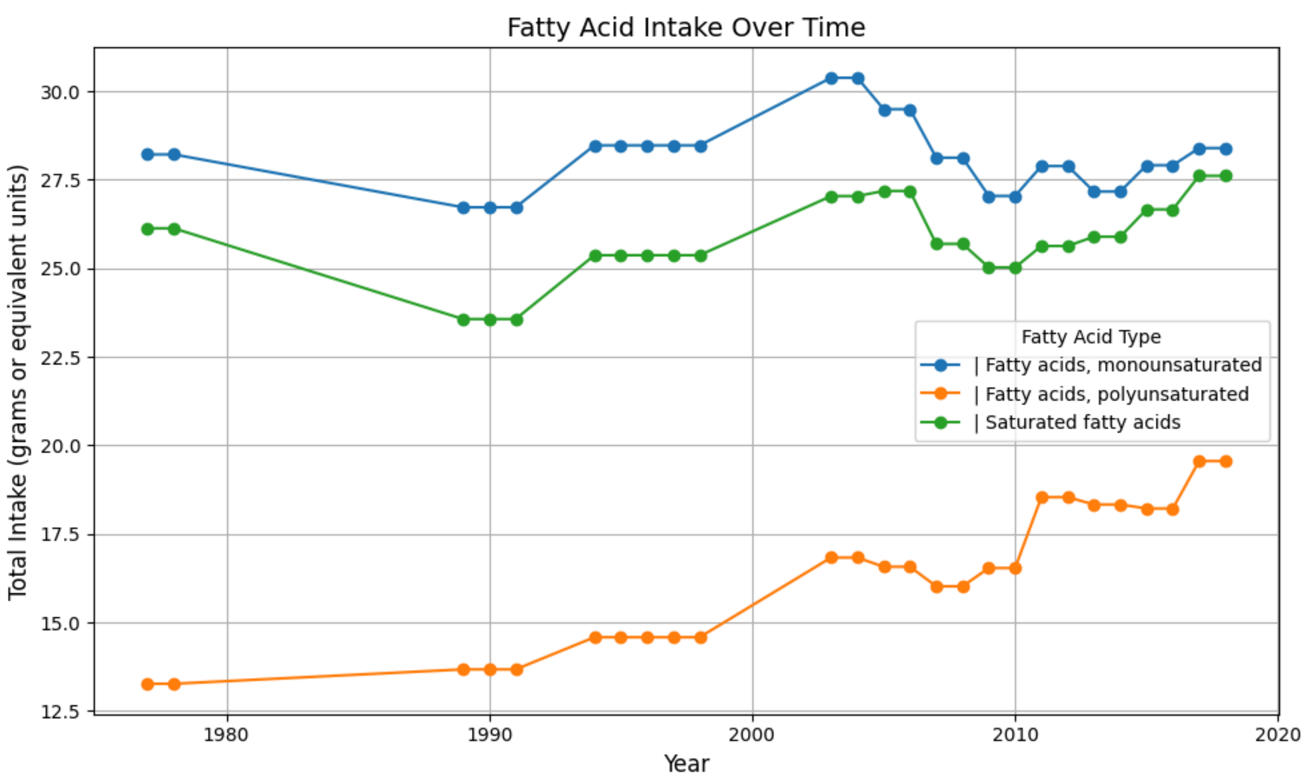


Chart 2: FAH vs FAFH by Region

This chart shows the share of food at home and food away from home sales by region. From this we can see that in most regions, people are spending around the same amount of money on food at home and eating out. We can also see that most of the food expenditure occurs in the South region of the United States.

```
region_sales = sales_table.groupby('Region')[[
    'FAH sales million nominal U.S. dollars with taxes and tips',
    'FAFH sales million nominal U.S. dollars with taxes and tips'
]].sum().reset_index()

region_sales.columns =['Region', 'FAH Sales (Nominal $M)', 'FAFH Sales (Nominal $M)']

region_sales['Total Sales (Nominal $M)'] = (
    region_sales['FAH Sales (Nominal $M)'] + region_sales['FAFH Sales (Nominal $M)']
)

region_melted = pd.melt(region_sales, id_vars='Region', var_name='Type', value_name='Sales ($M)')

plt.figure(figsize=(8, 6))
sns.barplot(data=region_melted, x='Region', y='Sales ($M)', hue='Type')
plt.title('FAH vs FAFH Sales by Region')
plt.tight_layout()

# Save to file
plt.savefig("region_sales.png", dpi=300)

# Show chart
plt.show()
```

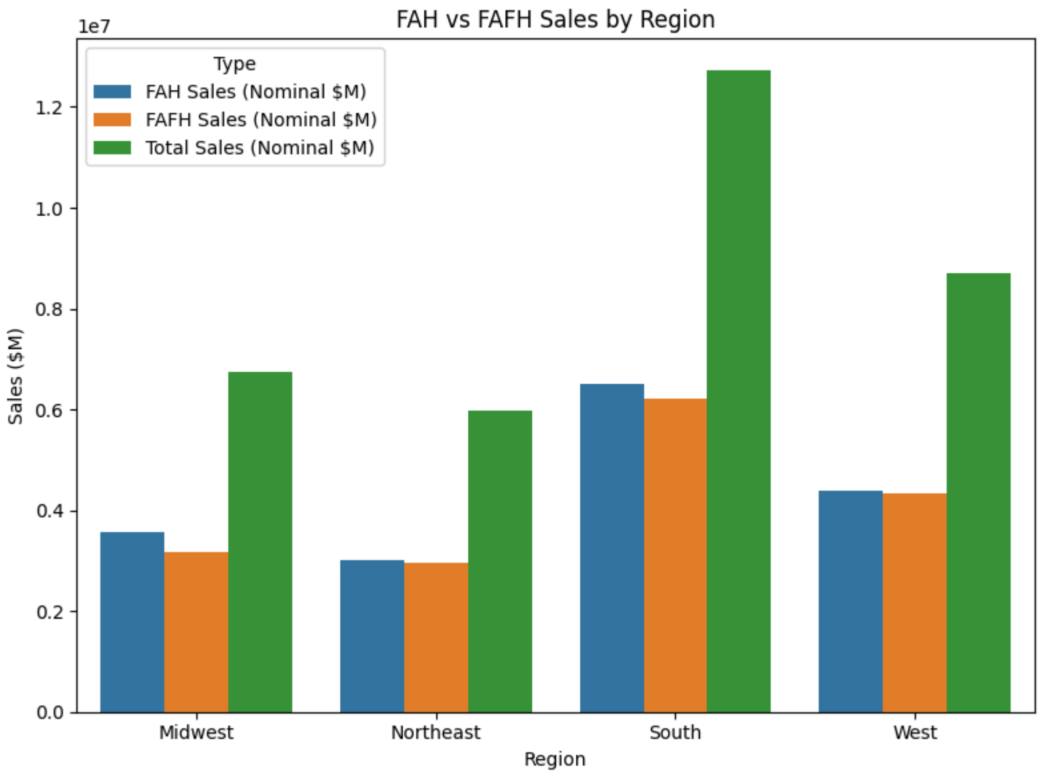


Chart 3: Sodium by Source

This chart compares sodium intake over time from food-at-home (FAH) and four food-away-from-home (FAFH) categories: fast food, restaurants, schools, and others. FAH consistently contributes the highest overall sodium intake, likely due to the volume and frequency of home-prepared food. However, when viewed collectively, FAFH categories—particularly fast food and restaurant meals—make up a substantial share of sodium intake.

```
query = """
SELECT
    Year,
    "Food Away From Home | Sodium" AS FAFH_Sodium,
    "Food at Home | Sodium" AS FAH_Sodium,
    "Total | Sodium" AS Total_Sodium
FROM nut_intake_wide_final
ORDER BY Year;
"""

sodium_data = pd.read_sql(query, conn)
```

```
# Melt to long format for plotting
sodium_long = sodium_data.melt(id_vars='Year', var_name='Source', value_name='Sodium Intake')

# Clean up labels for legend
sodium_long['Source'] = sodium_long['Source'].replace({
    'FAFH_Sodium': 'Food Away From Home',
    'FAH_Sodium': 'Food at Home',
    'Total_Sodium': 'Total'
})

# Plot
plt.figure(figsize=(12, 6))
sns.lineplot(data=sodium_long, x='Year', y='Sodium Intake', hue='Source', marker='o')
plt.title("Sodium Intake by Source Over Time", fontsize=14)
plt.xlabel("Year", fontsize=12)
plt.ylabel("Sodium Intake (mg)", fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.savefig("chart3_sodium_by_source.png", dpi=300)
plt.show()
```

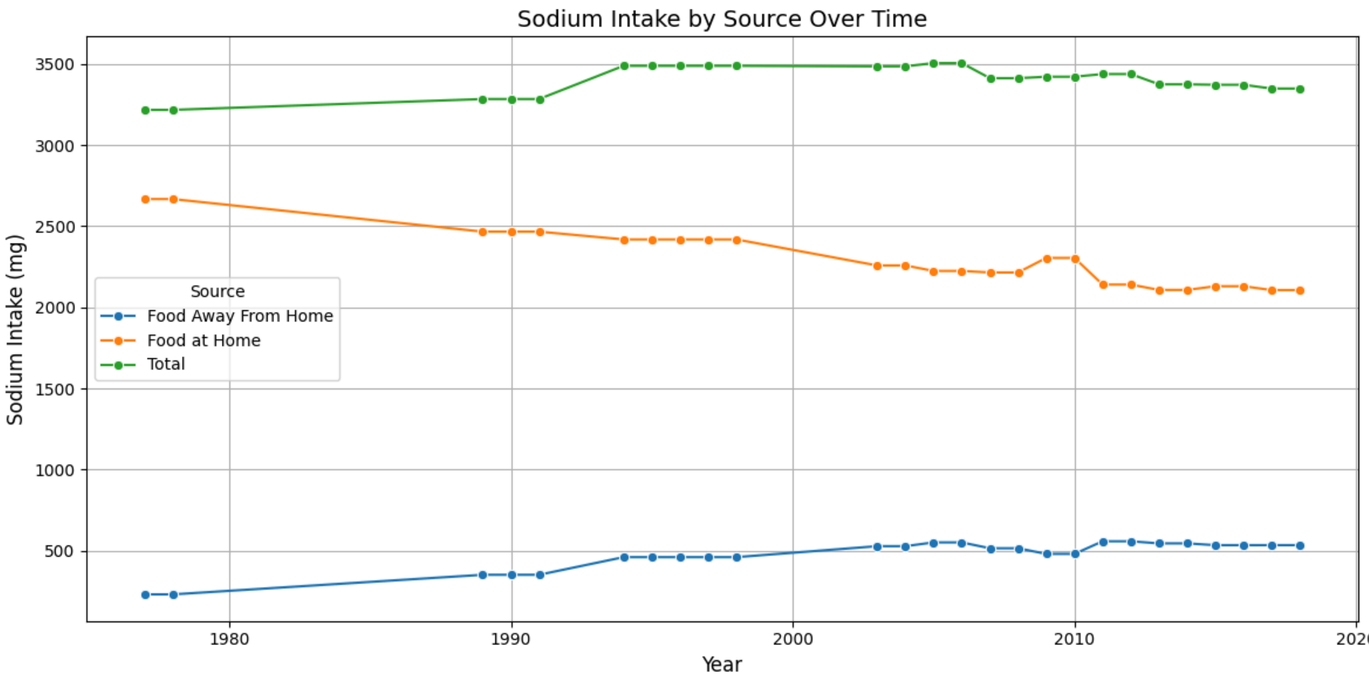


Chart 4: Normalized FAFH Nutrients

This heatmap shows the normalized intake levels of key nutrients consumed from Food Away From Home (FAFH) sources, such as restaurants and fast food, between 1977 and 2018. Each nutrient is scaled individually to show relative changes over time (from 0% to 100% of its peak value), allowing for clear comparison of trend patterns regardless of absolute quantity.

Notable patterns include:

- A sharp rise in nearly all nutrients during the 1990s and early 2000s.
- A plateau or modest decline in nutrients like sodium and saturated fat post-2010, suggesting possible impacts from public health awareness and reformulation efforts.
- Persistently low relative growth in fiber intake, indicating a nutritional gap that remains unaddressed.

This normalized view helps highlight **temporal trends** in nutrient density without being skewed by the absolute values of inherently high-quantity nutrients like calories or sodium.

```
query = """
WITH long_format AS (
    SELECT Year, 'Calcium' AS Nutrient, "Food Away From Home | Calcium" AS Value FROM nut_intake_wide_final
```

```
    UNION ALL SELECT Year, 'Fiber, dietary', "Food Away From Home | Fiber, dietary" FROM nut_intake_wide_final
    UNION ALL SELECT Year, 'Iron', "Food Away From Home | Iron" FROM nut_intake_wide_final
    UNION ALL SELECT Year, 'Protein', "Food Away From Home | Protein" FROM nut_intake_wide_final
    UNION ALL SELECT Year, 'Energy', "Food Away From Home | Energy" FROM nut_intake_wide_final
    UNION ALL SELECT Year, 'Saturated fatty acids', "Food Away From Home | Saturated fatty acids" FROM nut_intake_wide_final
    UNION ALL SELECT Year, 'Sodium', "Food Away From Home | Sodium" FROM nut_intake_wide_final
),
min_max AS (
    SELECT Nutrient,
           MIN(Value) AS Min_Val,
           MAX(Value) AS Max_Val
    FROM long_format
    GROUP BY Nutrient
)
SELECT
    l.Year,
    l.Nutrient,
    ROUND((l.Value - m.Min_Val) * 1.0 / (m.Max_Val - m.Min_Val), 4) AS Normalized_Value
FROM long_format l
JOIN min_max m ON l.Nutrient = m.Nutrient
WHERE l.Value IS NOT NULL
ORDER BY l.Nutrient, l.Year;
"""

chart4_sql_result = pd.read_sql(query, conn)

# Pivot for heatmap
heatmap_data = chart4_sql_result.pivot(index='Nutrient', columns='Year', values='Normalized_Value')

# Ensure numeric dtype
heatmap_data = heatmap_data.apply(pd.to_numeric, errors='coerce')

# Plot heatmap
plt.figure(figsize=(14, 6))
sns.heatmap(heatmap_data, cmap='YlGnBu', linewidths=0.5, linecolor='gray')
plt.title('Normalized Trends in Food Away From Home Nutrients', fontsize=14)
plt.xlabel('Year')
plt.ylabel('Nutrient')
plt.tight_layout()
plt.savefig("chart4_fafh_heatmap.png", dpi=300)
plt.show()
```

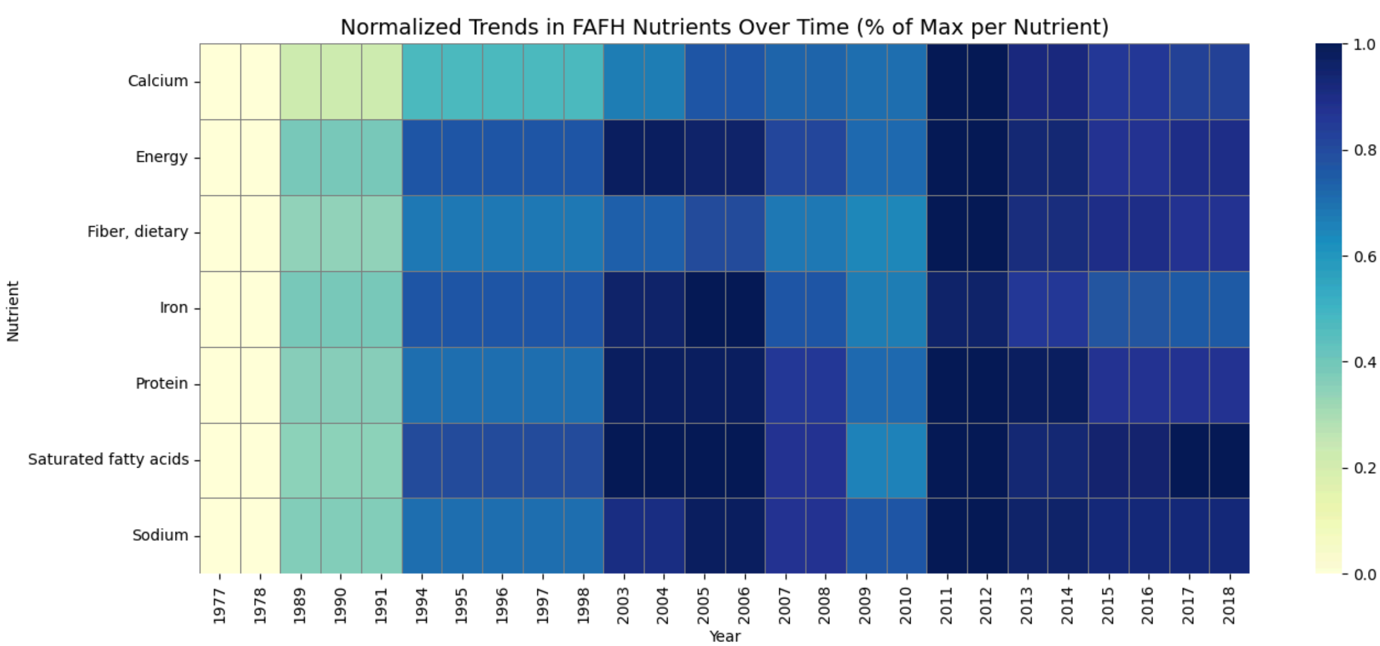


Chart 5: Per Capita Sales Boxplot

This boxplot displays the distribution of **nominal per capita food sales** (including taxes and tips) across U.S. states between 1997 and 2023. Each box shows the interquartile range (IQR), median, and outliers for total annual food sales per resident.

Key observations:

- **States like Nevada, D.C., and Hawaii consistently spend more on food per capita**, likely due to tourism, higher cost of living, and urban density.
- **Lower-spending states**, including West Virginia and Mississippi, may reflect regional income differences or more reliance on food prepared at home.
- **Wide ranges in states like California and Florida** suggest shifting economic conditions, population diversity, or seasonal dynamics.

This visualization helps identify regional disparities in food spending, offering insights into economic behavior and potential areas for targeted nutrition or policy interventions.

```
query = """
-- Query: Distribution of Per Capita Food Sales by State (Chart 5)
-- What: This query retrieves total per capita food sales (including taxes and tips) across all U.S. states from 1997 to 2023.
-- Why: To analyze how food spending per person varies by state and identify spending patterns or disparities.
-- How: This is a simple select query that pulls the relevant per capita column from the cleaned sales dataset.

SELECT
    State,
    Year,
    "Total sales per capita nominal U.S. dollars with taxes and tips" AS PerCapitaSales
FROM sales_percapita_final
WHERE "Total sales per capita nominal U.S. dollars with taxes and tips" IS NOT NULL
ORDER BY State, Year;
"""

chart5_sql_result = pd.read_sql(query, conn)
chart5_sql_result.head()
```

	State	Year	PerCapitaSales
0	Alabama	1997	2142.20
1	Alabama	1998	2276.54
2	Alabama	1999	2404.99
3	Alabama	2000	2493.97
4	Alabama	2001	2573.36

```
# Sort states by median for clarity
sorted_states = chart5_sql_result.groupby('State')['PerCapitaSales'].median().sort_values(ascending=False).index
chart5_sql_result['State'] = pd.Categorical(chart5_sql_result['State'], categories=sorted_states, ordered=True)

# Plot boxplot
plt.figure(figsize=(10, 14))
sns.boxplot(data=chart5_sql_result, y='State', x='PerCapitaSales', orient='h')
plt.title('Distribution of Per Capita Food Sales by State (Sorted by Median Spending)')
plt.xlabel('Total Sales per Capita (Nominal USD)')
plt.ylabel('State')
plt.tight_layout()
plt.savefig("per capita food sales.png")
plt.show()
```

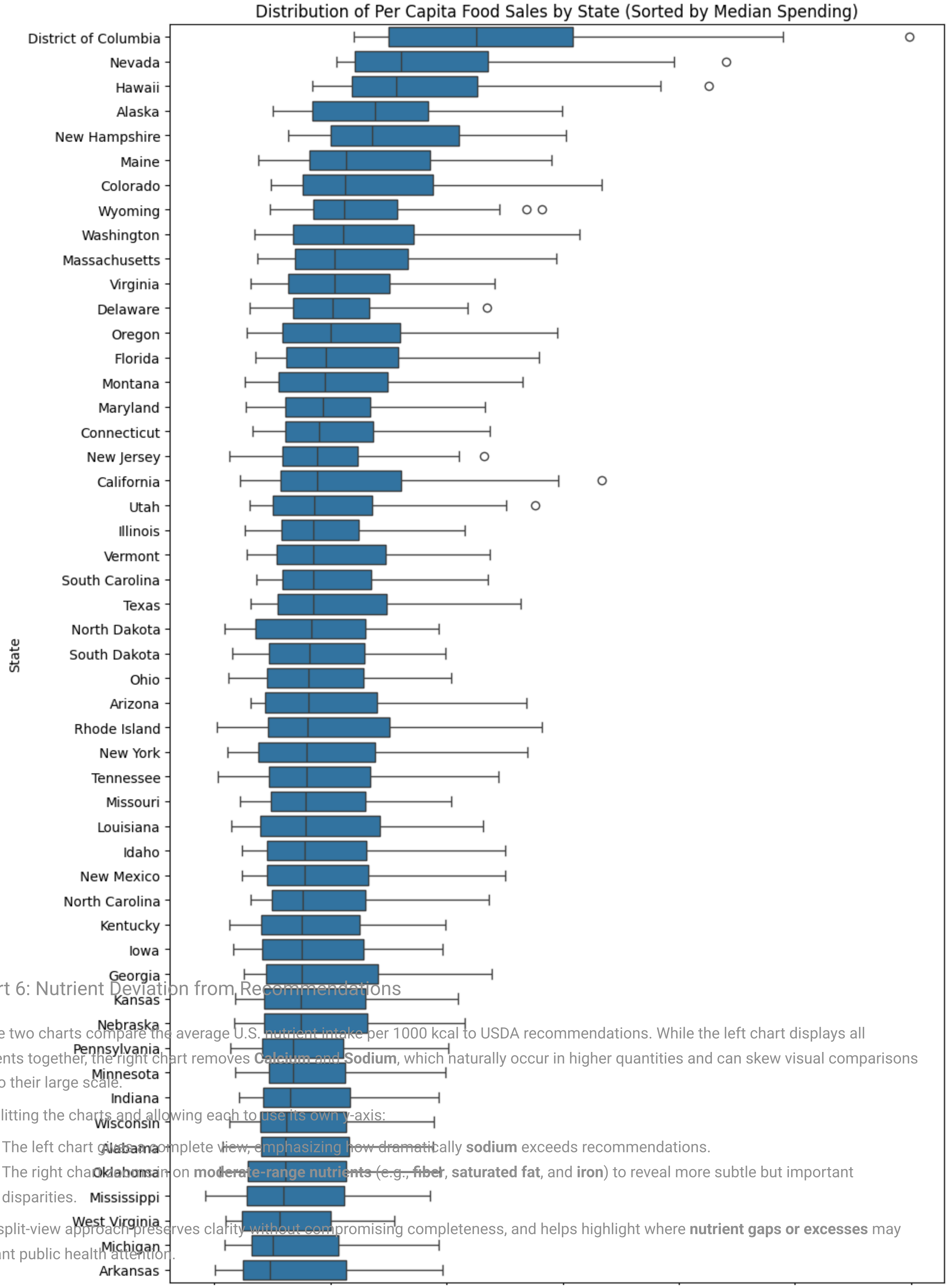


Chart 6: Nutrient Deviation from Recommendations

These two charts compare the average U.S. nutrient intake per 1000 kcal to USDA recommendations. While the left chart displays all nutrients together, the right chart removes **Calcium and Sodium**, which naturally occur in higher quantities and can skew visual comparisons due to their large scale.

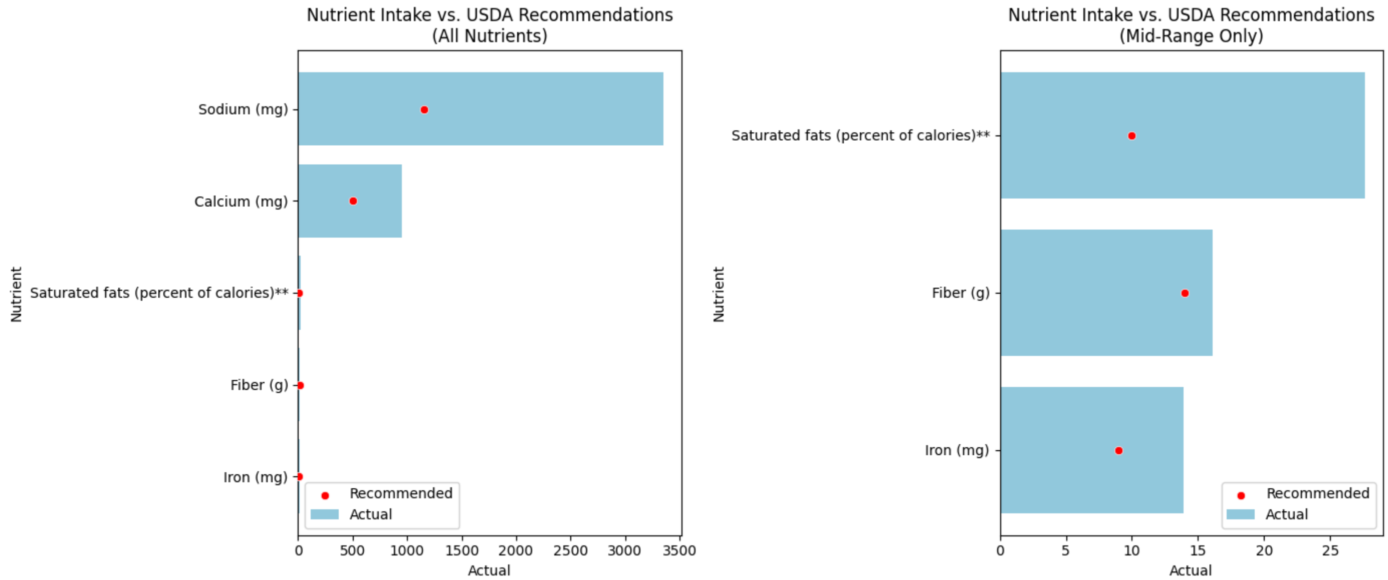
By splitting the charts and allowing each to use its own y-axis:

- The left chart gives a complete view, emphasizing how dramatically **sodium** exceeds recommendations.
- The right chart zooms in on **moderate-range nutrients** (e.g., **fiber**, **saturated fat**, and **iron**) to reveal more subtle but important disparities.

This split-view approach preserves clarity without compromising completeness, and helps highlight where **nutrient gaps or excesses** may warrant public health attention.

```
query = """
WITH nutrient_map AS (
    SELECT 'Fiber (g)' AS RecLabel, 'Total | Fiber, dietary' AS ActualColumn
    UNION ALL SELECT 'Saturated fats (percent of calories)**', 'Total | Saturated fatty acids'
    UNION ALL SELECT 'Calcium (mg)', 'Total | Calcium'
    UNION ALL SELECT 'Iron (mg)', 'Total | Iron'
```

```
    UNION ALL SELECT 'Sodium (mg)', 'Total | Sodium'
),
base AS (
    SELECT
        r."Nutrient or Food group" AS Nutrient,
        r.RecDensity_per1000cal AS Recommended,
        n.Year,
        CASE
            WHEN r."Nutrient or Food group" = 'Fiber (g)' THEN n."Total | Fiber, dietary"
            WHEN r."Nutrient or Food group" = 'Saturated fats (percent of calories)**' THEN n."Total | Saturated fatty acids"
            WHEN r."Nutrient or Food group" = 'Calcium (mg)' THEN n."Total | Calcium"
            WHEN r."Nutrient or Food group" = 'Iron (mg)' THEN n."Total | Iron"
            WHEN r."Nutrient or Food group" = 'Sodium (mg)' THEN n."Total | Sodium"
        END AS Actual
    FROM rec_density_final r
    JOIN nutrient_map m ON r."Nutrient or Food group" = m.RecLabel
    JOIN nut_intake_wide_final n
),
final AS (
    SELECT
        Nutrient,
        Year,
        ROUND(Recommended, 2) AS Recommended,
        ROUND(Actual, 2) AS Actual,
        ROUND(Actual - Recommended, 2) AS Delta,
        ROUND((Actual - Recommended) * 100.0 / Recommended, 2) AS Percent_Deviation
    FROM base
    WHERE Actual IS NOT NULL
)
SELECT * FROM final
ORDER BY Nutrient, Year;
"""

chart6_sql_untidy = pd.read_sql(query, conn)

# Filter to most recent year
chart_latest = chart6_sql_untidy[chart6_sql_untidy['Year'] == chart6_sql_untidy['Year'].max()]

# Prepare full and zoomed views
chart_full = chart_latest.sort_values(by='Actual', ascending=False).copy()
chart_zoomed = chart_latest[~chart_latest['Nutrient'].isin(['Calcium (mg)', 'Sodium (mg)'])].copy()
chart_zoomed = chart_zoomed.sort_values(by='Actual', ascending=False)

# Plot split-view bar charts
fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=False)

# Left: All nutrients
sns.barplot(data=chart_full, x='Actual', y='Nutrient', ax=axes[0], color='skyblue', label='Actual')
sns.scatterplot(data=chart_full, x='Recommended', y='Nutrient', ax=axes[0], color='red', label='Recommended', zorder=10)
axes[0].set_title("Nutrient Intake vs. USDA Recommendations\n(All Nutrients)")
axes[0].legend()

# Right: Without Calcium & Sodium
sns.barplot(data=chart_zoomed, x='Actual', y='Nutrient', ax=axes[1], color='skyblue', label='Actual')
sns.scatterplot(data=chart_zoomed, x='Recommended', y='Nutrient', ax=axes[1], color='red', label='Recommended', zorder=10)
axes[1].set_title("Nutrient Intake vs. USDA Recommendations\n(Mid-Range Only)")
axes[1].legend()

plt.tight_layout()
plt.savefig("chart6_nutrient_deviation_splitview.png", dpi=300)
plt.show()
```



4. Modeling: Spending vs Nutrient Intake

Step 1: Join SQL Data

We start by joining national-level nutrient intake data with average per capita food spending data across all U.S. states from 1997 to 2023. This SQL query uses a JOIN and filters for completeness:

```
query = """
-- SQL Query: Join Spending and Nutrient Intake Data
-- Why: To evaluate if higher spending is associated with healthier nutrient consumption.
-- Techniques used: Join, filtering, derived columns

SELECT
    s.Year,
    s."Total sales per capita nominal U.S. dollars with taxes and tips" AS PerCapitaSpending,
    n."Total | Fiber, dietary" AS Fiber,
    n."Total | Protein" AS Protein,
    n."Total | Saturated fatty acids" AS SatFat,
    AVG(s."Total sales per capita nominal U.S. dollars with taxes and tips") OVER (PARTITION BY s.Year) AS Rolling_Avg_Spendin
FROM
    sales_percapita_final s
JOIN
    nut_intake_wide_final n
    ON s.Year = n.Year
WHERE
    s."Total sales per capita nominal U.S. dollars with taxes and tips" IS NOT NULL
    AND n."Total | Fiber, dietary" IS NOT NULL
    AND n."Total | Protein" IS NOT NULL
    AND n."Total | Saturated fatty acids" IS NOT NULL
"""

# Run SQL and load results
df_model = pd.read_sql(query, conn)

# Group to year level: average spending, national nutrient values (since nutrients already represent national data)
df_avg = df_model.groupby('Year', as_index=False).agg({
    'PerCapitaSpending': 'mean',
    'Fiber': 'first',
    'Protein': 'first',
```

Step 2: Regression Plots

```
fig, axs = plt.subplots(1, 3, figsize=(18, 5))
nutrients = ['Fiber', 'Protein', 'SatFat']
titles = ['Fiber Intake vs Spending', 'Protein Intake vs Spending', 'Saturated Fat vs Spending']

for i, nutrient in enumerate(nutrients):
    sns.regplot(
        ax=axs[i],
        x='PerCapitaSpending',
        y=nutrient,
        data=df_avg,
        scatter_kws={'s': 40},
        line_kws={'color': 'red'}
    )
    axs[i].set_title(titles[i])
    axs[i].set_xlabel('Per Capita Food Spending (Nominal $)')
    axs[i].set_ylabel(f'{nutrient} Intake')
```