Data Import and Setup

Importing essential libraries for data analysis:

- pandas: For data manipulation
- matplotlib & seaborn: For visualization
- json: For reading JSON data
- re: For text processing

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

Data Loading and DataFrame Creation

Loading recipe data from JSON file:

- Reading merged recipe data from 'merged_recipes.json'
- Converting JSON structure to pandas DataFrame for analysis

```
# Load the merged JSON file
with open('merged_recipes.json', 'r', encoding='utf-8') as f:
    merged_data = json.load(f)

# Convert to DataFrame
df = pd.DataFrame(merged_data)
```

Time Conversion Function

Helper function to standardize recipe time formats:

- Converts various time formats to minutes
- Handles 'PT' format (ISO duration)
- Processes 'mins' format
- Returns 0 for missing/invalid values

```
# Function to convert time to minutes
def convert_time_to_minutes(time_str):
    if pd.isna(time_str) or time_str == '-':
        return 0

    # Handle "PT" format
    if 'PT' in str(time_str):
        hours = 0
        minutes = 0
        if 'H' in time_str:
```

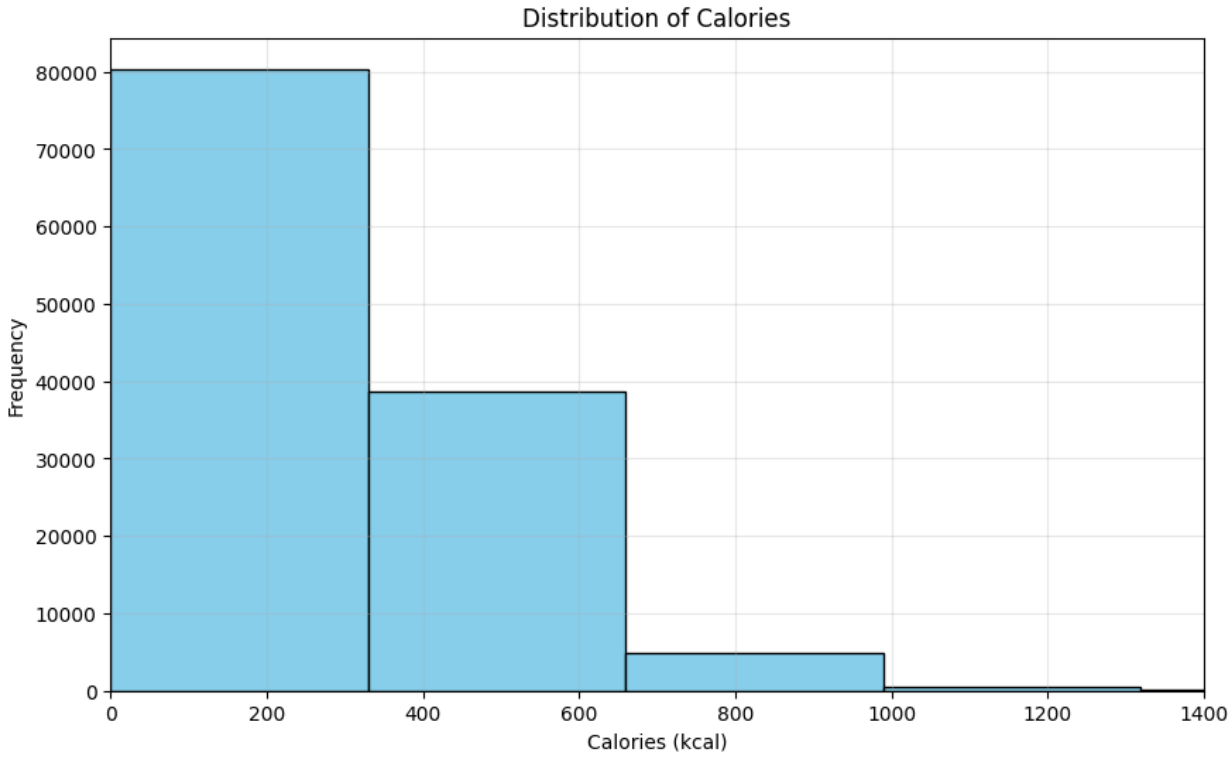
```
            hours = int(re.search(r'PT(\d+)H', time_str).group(1))
        if 'M' in time_str:
            minutes = int(re.search(r'(\d+)M', time_str).group(1))
        return hours * 60 + minutes

    # Handle "X mins" format
    if 'mins' in str(time_str):
        return int(re.search(r'(\d+)', time_str).group(1))

    return 0
```

Nutritional Data Visualization

Converting and visualizing caloric distribution:

- Converts nutritional values to numeric format
- Creates histogram of calorie distribution
- Limited to 1750 kcal for better visualization
- Shows frequency of recipes across calorie ranges

```
# Convert calories, fat, protein, and carbohydrate to numeric
numeric_columns = ['calories', 'fat', 'protein', 'carbohydrate']
for col in numeric_columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Now for the calorie distribution plot you shared:
plt.figure(figsize=(10, 6))
plt.hist(df['calories'].dropna(), bins=20, edgecolor='black',
color='skyblue')
plt.xlim(0, 1400)
plt.title('Distribution of Calories')
plt.xlabel('Calories (kcal)')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
plt.show()
```



Macronutrient Distribution Analysis

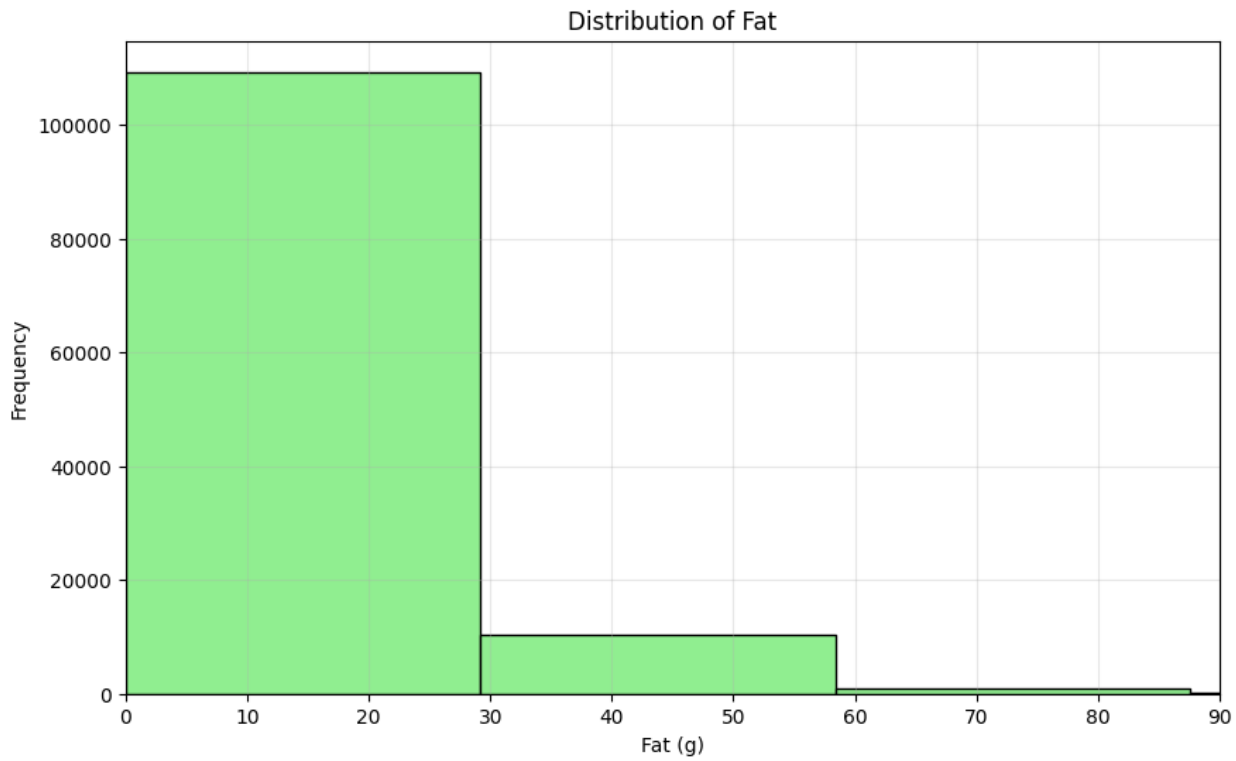
Visualizing distribution of key macronutrients:

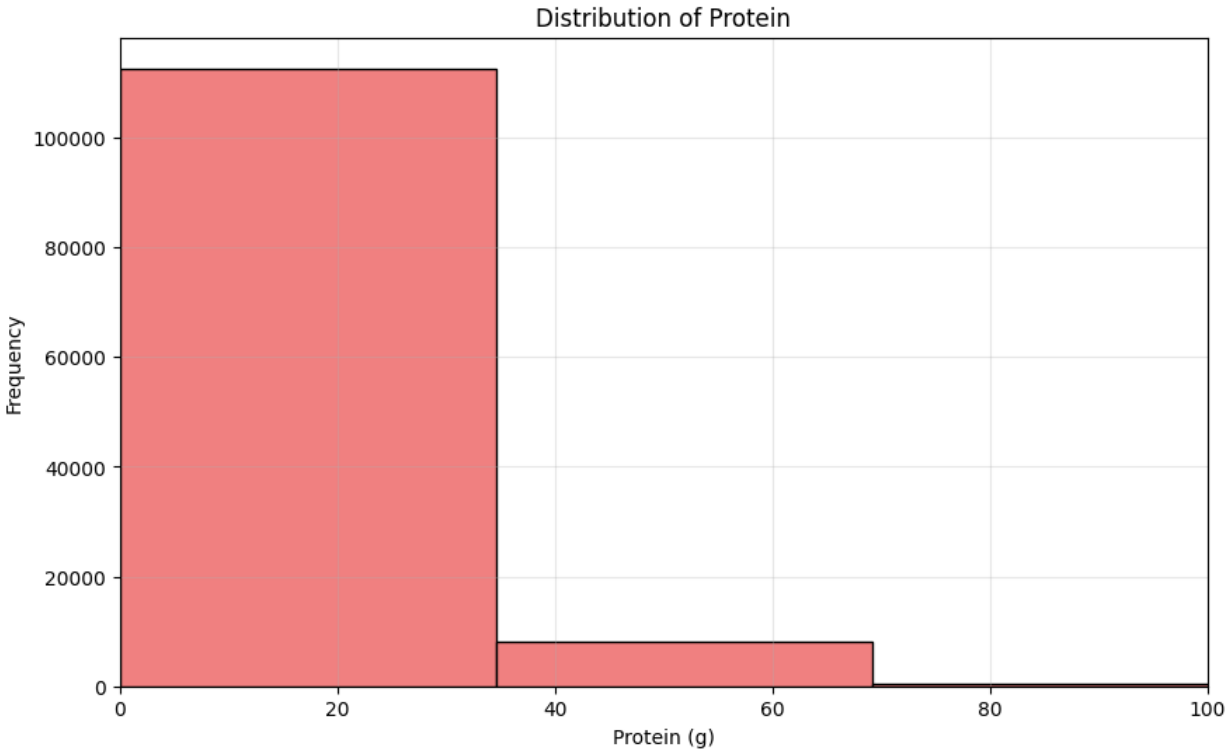
- Fat: Range 0-100g (green histogram)
- Protein: Range 0-100g (coral histogram)
- Carbohydrates: Range 0-250g (purple histogram)

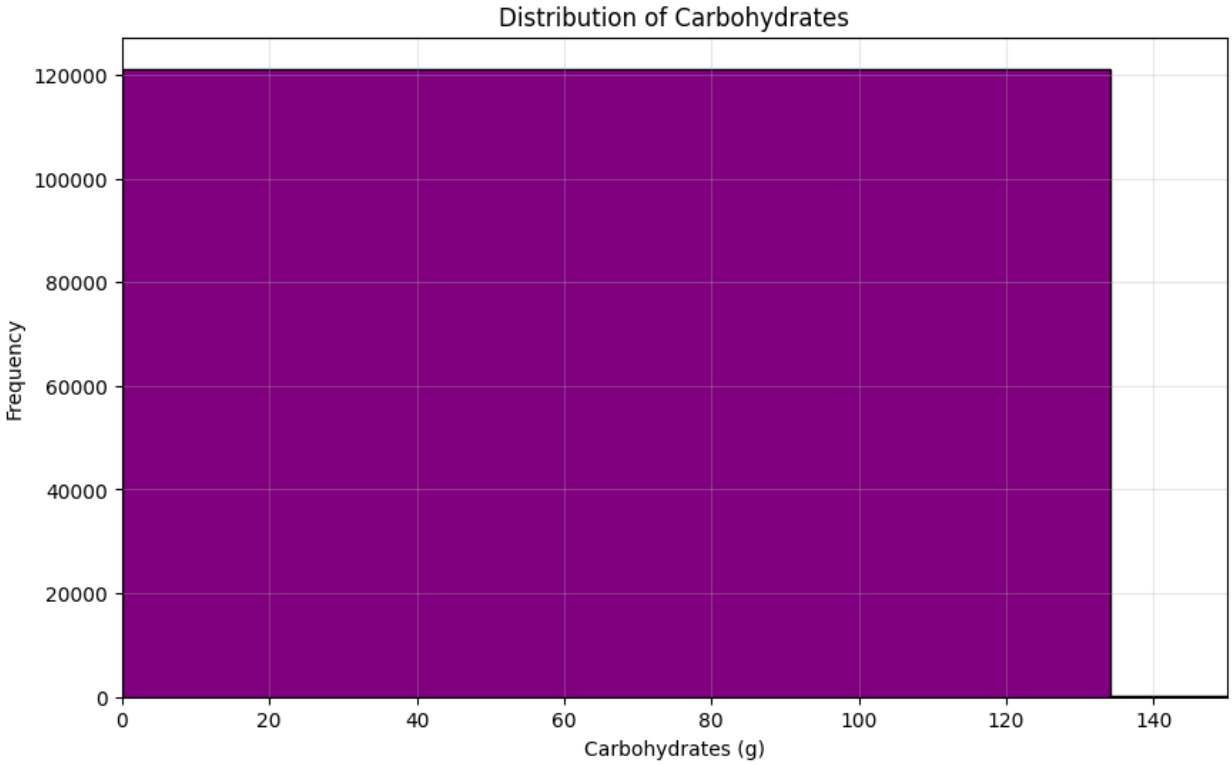
Each plot shows frequency distribution with consistent formatting and grid lines for easy comparison.

```
# Fat Distribution
plt.figure(figsize=(10, 6))
plt.hist(df['fat'].dropna(), bins=15, edgecolor='black',
color='lightgreen')
plt.xlim(0, 90)
plt.title('Distribution of Fat')
plt.xlabel('Fat (g)')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
plt.show()

# Protein Distribution
plt.figure(figsize=(10, 6))
plt.hist(df['protein'].dropna(), bins=15, edgecolor='black',
color='lightcoral')
```

```
plt.xlim(0, 100)
plt.title('Distribution of Protein')
plt.xlabel('Protein (g)')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
plt.show()

# Carbohydrate Distribution
plt.figure(figsize=(10, 6))
plt.hist(df['carbohydrate'].dropna(), bins=15, edgecolor='black',
color='purple')
plt.xlim(0, 150)
plt.title('Distribution of Carbohydrates')
plt.xlabel('Carbohydrates (g)')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
plt.show()
```



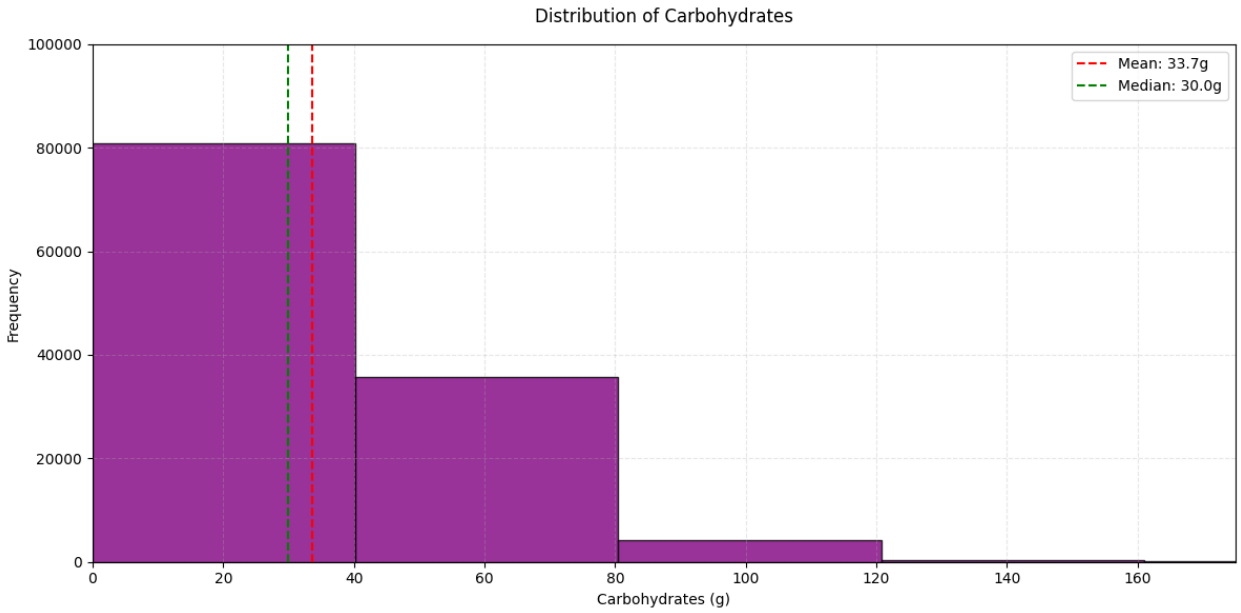





```
plt.figure(figsize=(12, 6))

# Create histogram with adjusted parameters
```

```
plt.hist(df['carbohydrate'].dropna(),
         bins=50,  # Increased bins for more detail
         edgecolor='black',
         color='purple',
         alpha=0.8)

# Set axis limits
plt.xlim(0, 175)
plt.ylim(0, 100000)  # Adjust if needed based on your data

# Add title and labels
plt.title('Distribution of Carbohydrates', pad=15, fontsize=12)
plt.xlabel('Carbohydrates (g)', fontsize=10)
plt.ylabel('Frequency', fontsize=10)

# Add grid
plt.grid(True, alpha=0.3, linestyle='--')

# Add mean and median lines for reference
mean_carb = df['carbohydrate'].mean()
median_carb = df['carbohydrate'].median()
plt.axvline(mean_carb, color='red', linestyle='--', label=f'Mean:
{mean_carb:.1f}g')
plt.axvline(median_carb, color='green', linestyle='--',
label=f'Median: {median_carb:.1f}g')

# Add legend
plt.legend()

# Adjust layout
plt.tight_layout()

# Optional: Print summary statistics
print(f"Number of recipes: {len(df['carbohydrate'].dropna())}")
print(f"Mean carbohydrates: {mean_carb:.1f}g")
print(f"Median carbohydrates: {median_carb:.1f}g")
print(f"Min carbohydrates: {df['carbohydrate'].min():.1f}g")
print(f"Max carbohydrates: {df['carbohydrate'].max():.1f}g")

plt.show()
```

```
Number of recipes: 121373
Mean carbohydrates: 33.7g
Median carbohydrates: 30.0g
Min carbohydrates: 0.0g
Max carbohydrates: 2012.0g
```



Recipe Time Analysis

Processing and visualizing recipe preparation and cooking times:

Data Processing:

- Converts ISO time format (PT1H30M) to minutes
- Handles both hours and minutes components
- Manages missing/invalid data

Visualizations:

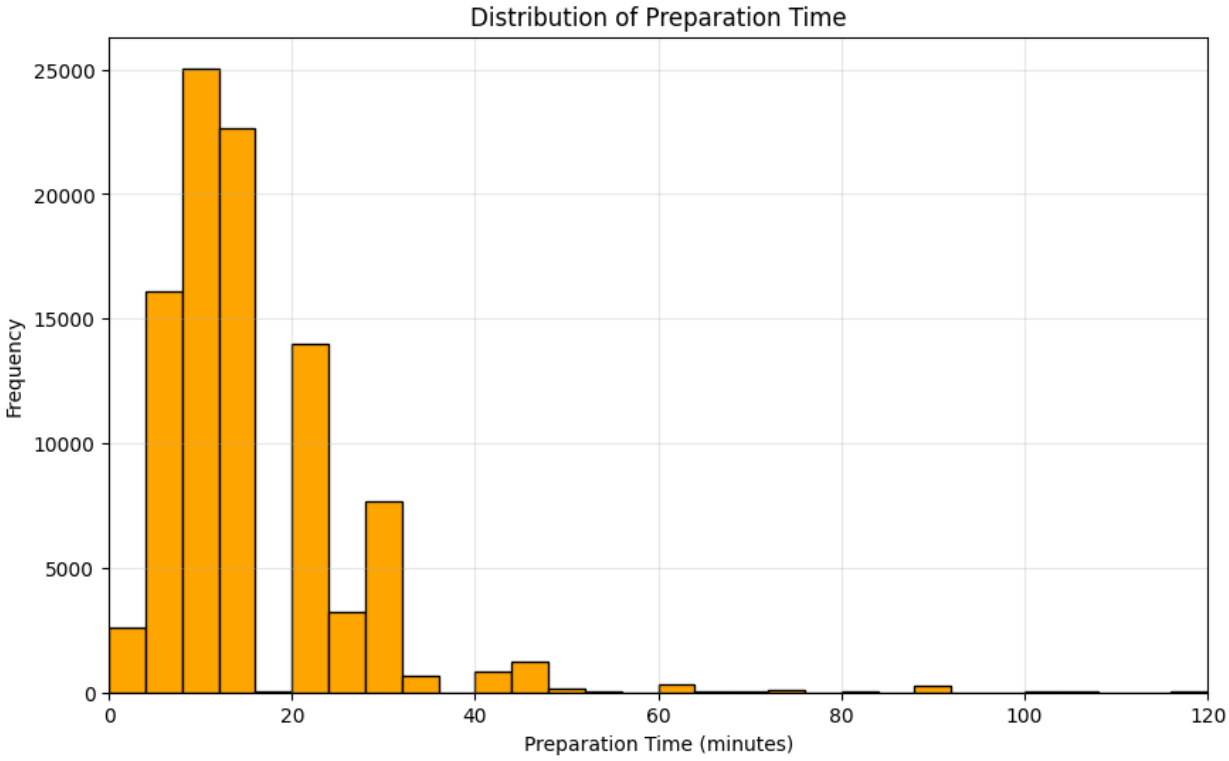
1. Preparation Time:
   - Range: 0-120 minutes (2 hours)
   - Orange histogram showing frequency distribution
2. Cooking Time:
   - Range: 0-180 minutes (3 hours)
   - Brown histogram showing frequency distribution

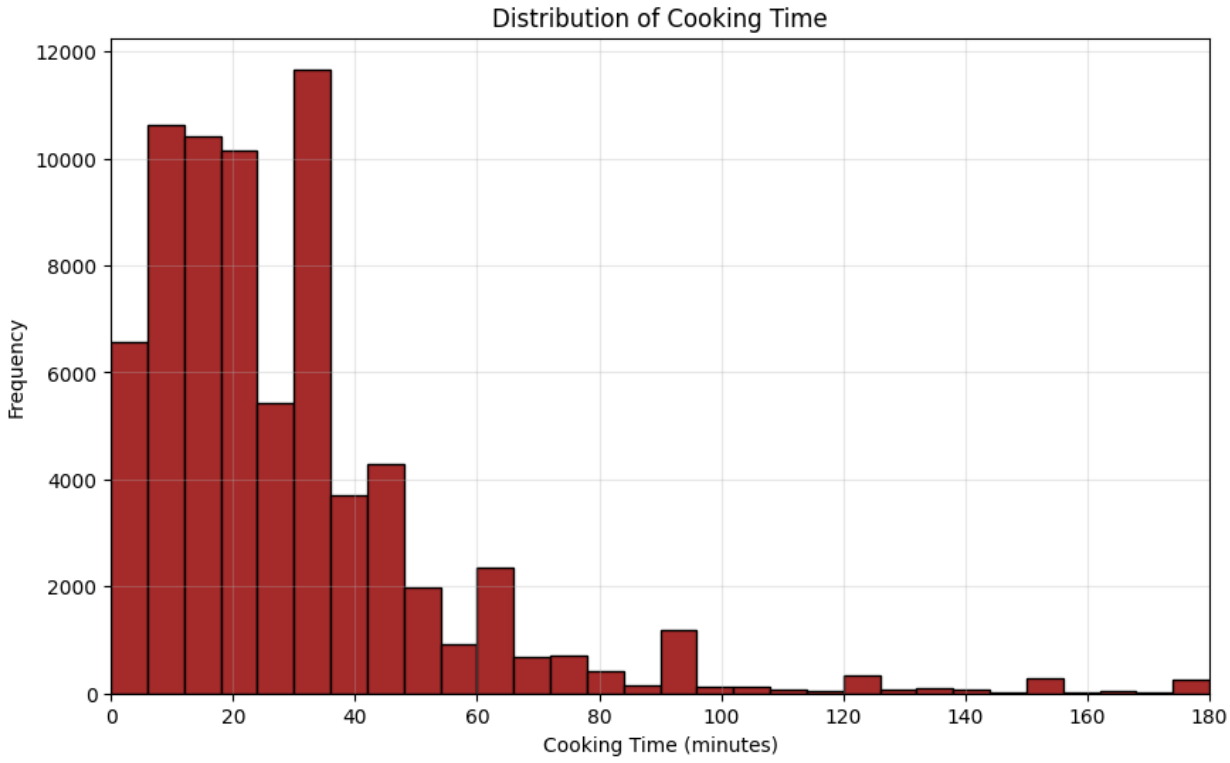
Both plots use 30 bins for detailed distribution view.

```
# Convert prep_time and cook_time from ISO format to minutes
def convert_time_to_minutes(time_str):
    if pd.isna(time_str):
        return None
    try:
        hours = int(time_str.split('H')[0].split('T')[1]) if 'H' in
time_str else 0
        minutes = int(time_str.split('H')[1].split('M')[0]) if 'H' in
time_str else int(time_str.split('T')[1].split('M')[0])
```

```
        return hours * 60 + minutes
    except:
        return None

# Convert times to minutes
df['prep_time_minutes'] =
df['prep_time'].apply(convert_time_to_minutes)
df['cook_time_minutes'] =
df['cook_time'].apply(convert_time_to_minutes)


# Convert prep_time and cook_time from ISO format to minutes
def convert_time_to_minutes(time_str):
    if pd.isna(time_str):
        return None
    try:
        hours = int(time_str.split('H')[0].split('T')[1]) if 'H' in
time_str else 0
        minutes = int(time_str.split('H')[1].split('M')[0]) if 'H' in
time_str else int(time_str.split('T')[1].split('M')[0])
        return hours * 60 + minutes
    except:
        return None

# Convert times to minutes
df['prep_time_minutes'] =
df['prep_time'].apply(convert_time_to_minutes)
df['cook_time_minutes'] =
df['cook_time'].apply(convert_time_to_minutes)

# Preparation Time Distribution with adjusted parameters
plt.figure(figsize=(10, 6))
plt.hist(df['prep_time_minutes'].dropna(),
         bins=30,  # increased number of bins
         edgecolor='black',
         color='orange',
         range=(0, 120))  # limit to 2 hours (120 minutes) for better
visualization
plt.xlim(0, 120)  # x-axis limit to 2 hours
plt.title('Distribution of Preparation Time')
plt.xlabel('Preparation Time (minutes)')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
plt.show()

# Cooking Time Distribution with adjusted parameters
plt.figure(figsize=(10, 6))
plt.hist(df['cook_time_minutes'].dropna(),
         bins=30,
         edgecolor='black',
```

```
         color='brown',
         range=(0, 180))  # limit to 3 hours (180 minutes) for better
visualization
plt.xlim(0, 180)  # x-axis limit to 3 hours
plt.title('Distribution of Cooking Time')
plt.xlabel('Cooking Time (minutes)')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
plt.show()



# # Cooking Time Distribution
# plt.figure(figsize=(10, 6))
# plt.hist(df['cook_time_minutes'].dropna(), bins=20,
edgecolor='black', color='brown')
# plt.xlim(0, 1000)
# plt.title('Distribution of Cooking Time')
# plt.xlabel('Cooking Time (minutes)')
# plt.ylabel('Frequency')
# plt.grid(True, alpha=0.3)
# plt.show()
```





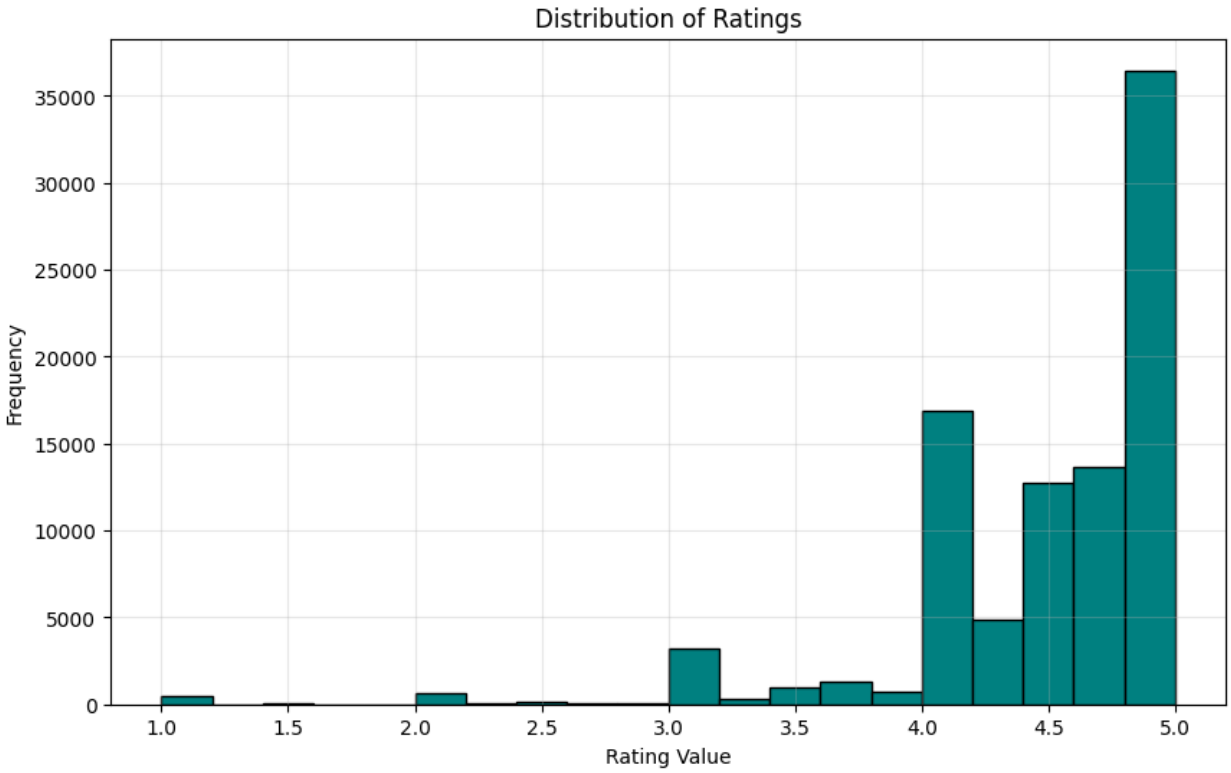
User Feedback Analysis

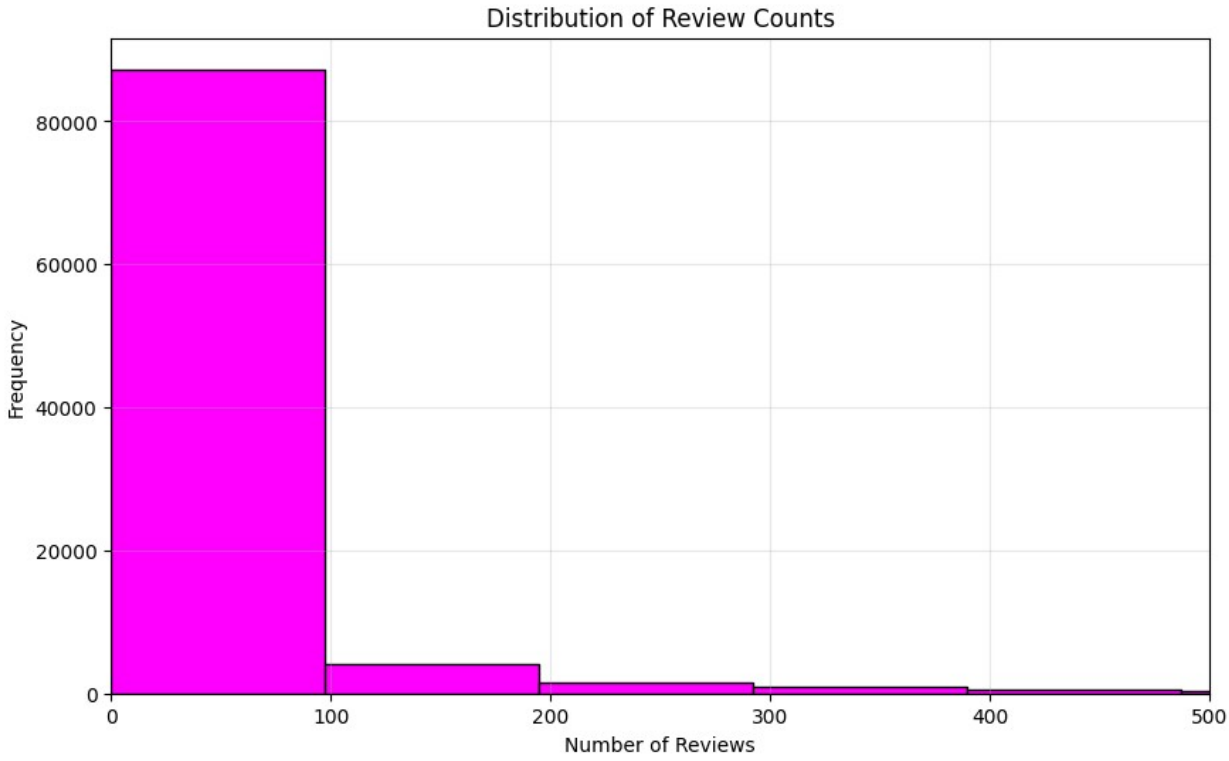
Analyzing recipe ratings and review patterns:

1. Rating Distribution:
   - Teal histogram showing rating frequency
   - Shows how recipes are rated (1-5 scale)
   - Helps identify common rating patterns
2. Review Count Distribution:
   - Magenta histogram limited to 1000 reviews
   - Shows how many reviews recipes typically receive
   - Identifies highly-reviewed vs rarely-reviewed recipes

```
# Convert rating to numeric if it's not already
df['rating'] = pd.to_numeric(df['rating'], errors='coerce')
df['review_count'] = pd.to_numeric(df['review_count'],
errors='coerce')

# Rating Distribution
plt.figure(figsize=(10, 6))
plt.hist(df['rating'].dropna(),
         bins=20,
         edgecolor='black',
         color='teal')
plt.title('Distribution of Ratings')
```

```
plt.xlabel('Rating Value')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
plt.show()

# Review Count Distribution
plt.figure(figsize=(10, 6))
plt.hist(df['review_count'].dropna(),
         bins=30,
         edgecolor='black',
         color='magenta')
plt.xlim(0, 500)
plt.title('Distribution of Review Counts')
plt.xlabel('Number of Reviews')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
plt.show()
```





Top Recipe Categories Analysis

Visualizing most common recipe categories:

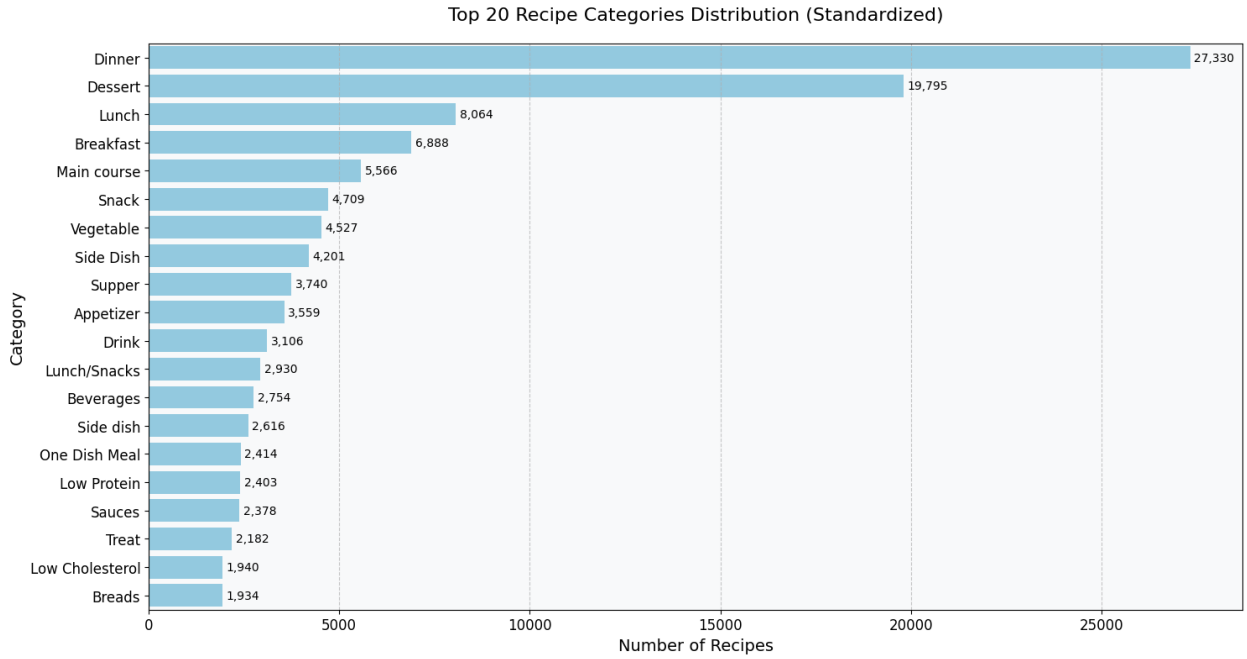
- Shows top 20 categories by recipe count
- Horizontal bar plot for better readability
- Categories sorted by frequency
- Clear view of recipe distribution across categories

Key Features:

- Horizontal orientation for better category label visibility
- Sky blue bars for clean visualization
- Adjusted figure size (15x8) for optimal display

```
# Step 1: Extract and standardize all categories
standardized_categories = []

for _, row in df.iterrows():
    if isinstance(row['category'], list):  # For list format in JSON
        for cat in row['category']:
            if cat is not None and cat != 'uncategorized':  # Check
for None
                # Split compound categories and add each part
                for subcat in cat.split(','):
                    standardized_categories.append(subcat.strip())
```

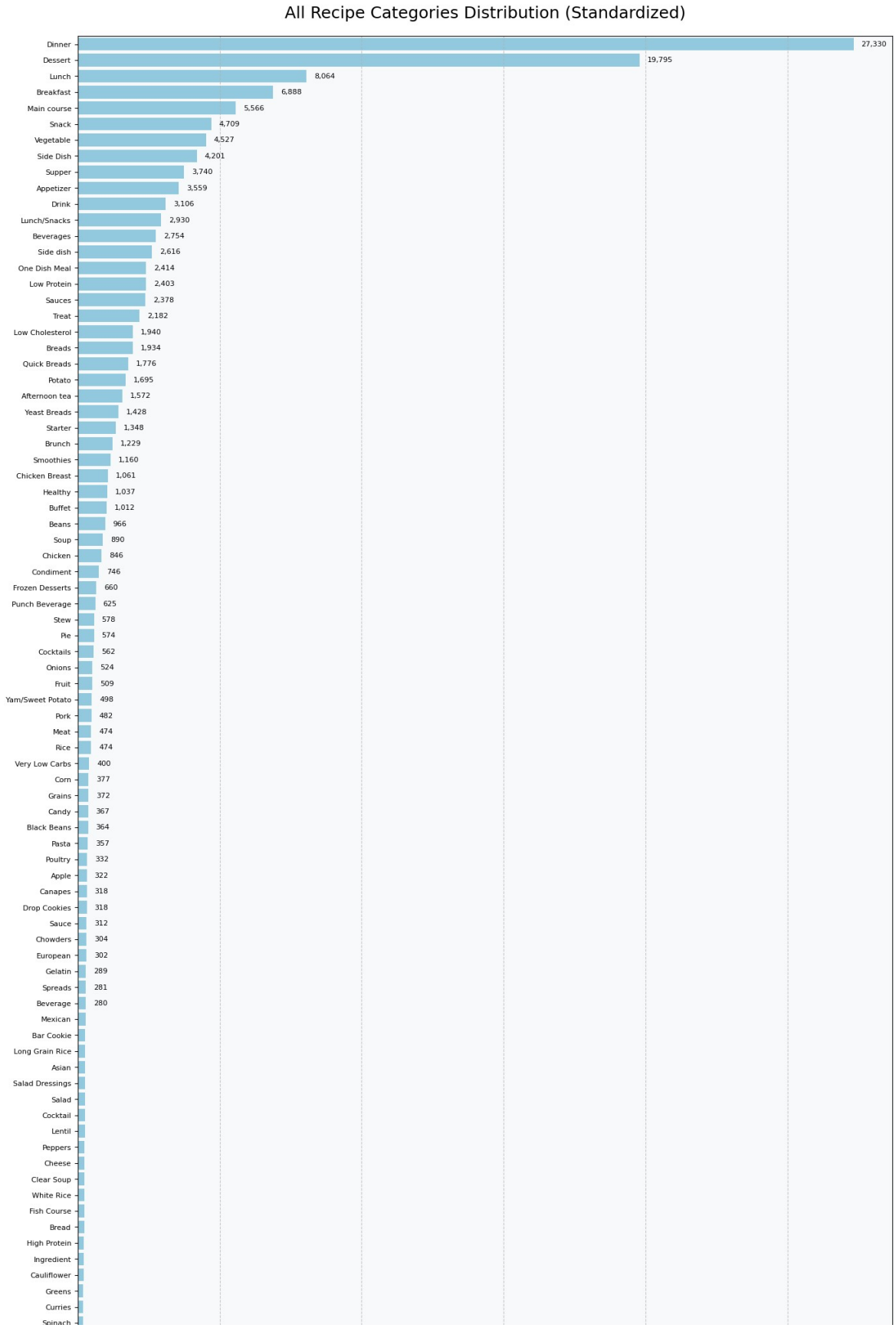
```
    elif isinstance(row['category'], str):  # For string format
        for cat in row['category'].split(','):
            if cat.strip() != 'uncategorized':
                standardized_categories.append(cat.strip())

# Count occurrences of each standardized category
category_counts = pd.Series(standardized_categories).value_counts()

# Take top 20 categories
top_20_categories = category_counts.head(20)

# Create the plot with enhanced styling
plt.figure(figsize=(15, 8))
bars = sns.barplot(x=top_20_categories.values,
                   y=top_20_categories.index,
                   color='skyblue',
                   orient='h')

# Add count labels to the bars
for i, v in enumerate(top_20_categories.values):
    plt.text(v + 100, i, f'{v:,}', va='center')

# Enhance the plot
plt.title('Top 20 Recipe Categories Distribution (Standardized)',
fontsize=16, pad=20)
plt.xlabel('Number of Recipes', fontsize=14)
plt.ylabel('Category', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add grid lines for better readability
plt.grid(axis='x', linestyle='--', alpha=0.7)

# Set background color
ax = plt.gca()
ax.set_facecolor('#f8f9fa')

# Adjust layout
plt.tight_layout()
plt.show()

# Print the actual counts
print("\nStandardized Category Counts:")
for category, count in top_20_categories.items():
    print(f"{category}: {count}")
```




```
Standardized Category Counts:
Dinner: 27330
Dessert: 19795
Lunch: 8064
Breakfast: 6888
Main course: 5566
Snack: 4709
Vegetable: 4527
Side Dish: 4201
Supper: 3740
Appetizer: 3559
Drink: 3106
Lunch/Snacks: 2930
Beverages: 2754
Side dish: 2616
One Dish Meal: 2414
Low Protein: 2403
Sauces: 2378
Treat: 2182
Low Cholesterol: 1940
Breads: 1934

# Step 1: Extract and standardize all categories
standardized_categories = []

for _, row in df.iterrows():
    if isinstance(row['category'], list):  # For list format in JSON
        for cat in row['category']:
            if cat is not None and cat != 'uncategorized':  # Check
```

```
for None
                # Split compound categories and add each part
                for subcat in cat.split(','):
                    standardized_categories.append(subcat.strip())
    elif isinstance(row['category'], str):  # For string format
        for cat in row['category'].split(','):
            if cat.strip() != 'uncategorized':
                standardized_categories.append(cat.strip())

# Count occurrences of each standardized category
category_counts = pd.Series(standardized_categories).value_counts()

# Calculate figure height based on number of categories (allow ~0.25
inch per category)
fig_height = max(30, len(category_counts) * 0.25)  # Minimum 30
inches, scales with categories

# Create the plot with enhanced spacing
plt.figure(figsize=(15, fig_height))
bars = sns.barplot(x=category_counts.values,
                   y=category_counts.index,
                   color='skyblue',
                   orient='h')

# Add count labels to the bars with better positioning
# Only add labels to bars that are visible enough
for i, v in enumerate(category_counts.values):
    if v > max(category_counts.values) * 0.01:  # Only label bars with
significant values
        plt.text(v + max(category_counts.values)*0.01, i, f'{v:,}',
va='center', fontsize=8)

# Enhance the plot
plt.title('All Recipe Categories Distribution (Standardized)',
fontsize=18, pad=20)
plt.xlabel('Number of Recipes', fontsize=16)
plt.ylabel('Category', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=8)  # Smaller font for y-axis labels

# Add grid lines for better readability
plt.grid(axis='x', linestyle='--', alpha=0.7)

# Set background color
ax = plt.gca()
ax.set_facecolor('#f8f9fa')

# Increase spacing between y-axis labels
plt.subplots_adjust(left=0.3, right=0.95)  # More space for labels
```

```
# Adjust layout with more padding
plt.tight_layout(pad=2.0)

# Save the figure with high resolution
plt.savefig('all_recipe_categories.png', dpi=300, bbox_inches='tight')
plt.show()

# Print the total number of unique categories
print(f"\nTotal unique categories: {len(category_counts)}")

# Save the data to CSV
category_df = pd.DataFrame({'Category': category_counts.index,
'Count': category_counts.values})
category_df.to_csv('recipe_categories_count.csv', index=False)
```


All Recipe Categories Distribution (Standardized)
Dinner 27,330
Dessert 19,795
Lunch 8,064
Breakfast 6,888
Main course 5,566
Snack 4,709
Vegetable 4,527
Side Dish 4,201
Supper 3,740
Appetizer 3,559
Drink 3,106
Lunch/Snacks 2,930
Beverages 2,754
Side dish 2,616
One Dish Meal 2,414
Low Protein 2,403
Sauces 2,378
Treat 2,182
Low Cholesterol 1,940
Breads 1,934
Quick Breads 1,776
Potato 1,695
Afternoon tea 1,572
Yeast Breads 1,428
Starter 1,348
Brunch 1,229
Smoothies 1,160
Chicken Breast 1,061
Healthy 1,037
Buffet 1,012
Beans 966
Soup 890
Chicken 846
Condiment 746
Frozen Desserts 660
Punch Beverage 625
Stew 578
Pie 574
Cocktails 562
Onions 524
Fruit 509
Yam/Sweet Potato 498
Pork 482
Meat 474
Rice 474
Very Low Carbs 400
Corn 377
Grains 372
Candy 367
Black Beans 364
Pasta 357
Poultry 332
Apple 322
Canapes 318
Drop Cookies 318
Sauce 312
Chowders 304
European 302
Gelatin 289
Spreads 281
Beverage 280
Mexican
Bar Cookie
Long Grain Rice
Asian
Salad Dressings
Salad
Cocktail
Lentil
Peppers
Cheese
Clear Soup
White Rice
Fish Course
Bread
High Protein
Ingredient
Cauliflower
Greens
Curries
Spinach

```
Total unique categories: 269
```

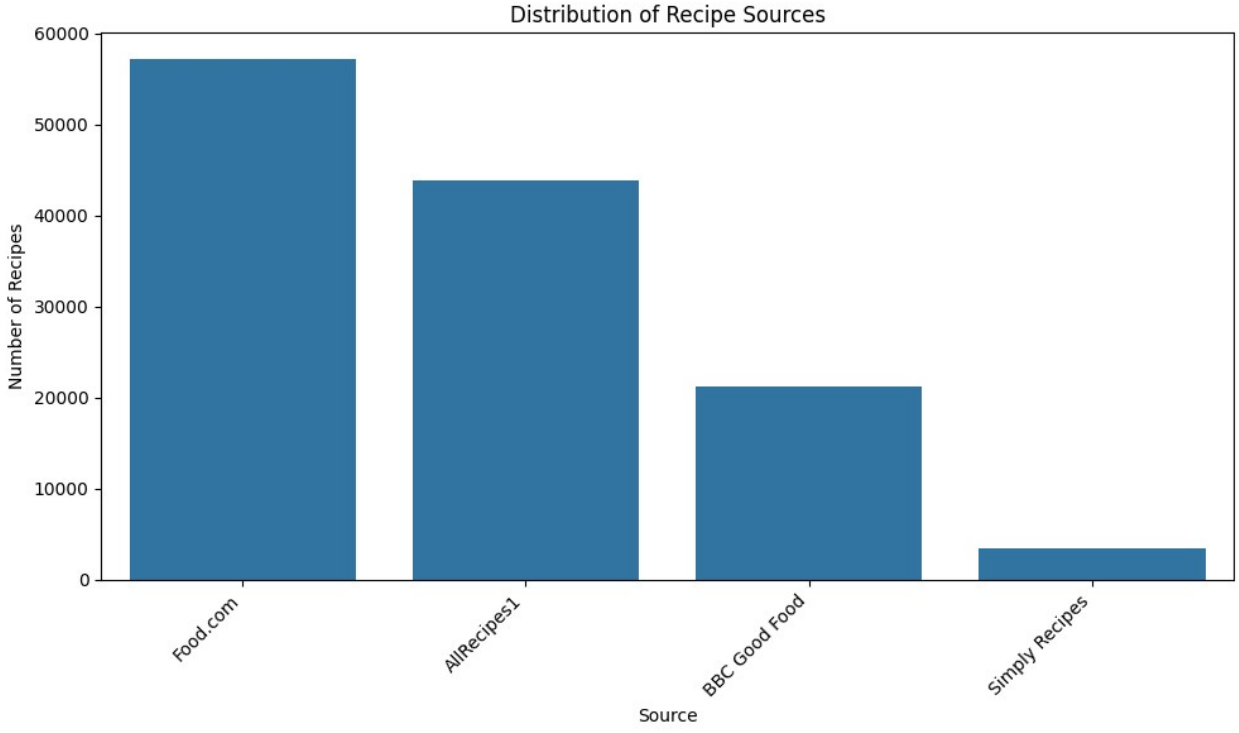
Recipe Source Analysis

Analyzing recipe distribution across different sources:

- Vertical bar plot showing recipe count by source
- Labels rotated 45° for readability
- Shows which platforms contribute most recipes
- Helps identify dominant recipe sources in dataset

Note: Bar heights represent number of recipes from each source

```python
# 5. Source Distribution
plt.figure(figsize=(10, 6))
source_counts = df['source'].value_counts()
sns.barplot(x=source_counts.index, y=source_counts.values)
plt.title('Distribution of Recipe Sources')
plt.xlabel('Source')
plt.ylabel('Number of Recipes')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Correlation Analysis

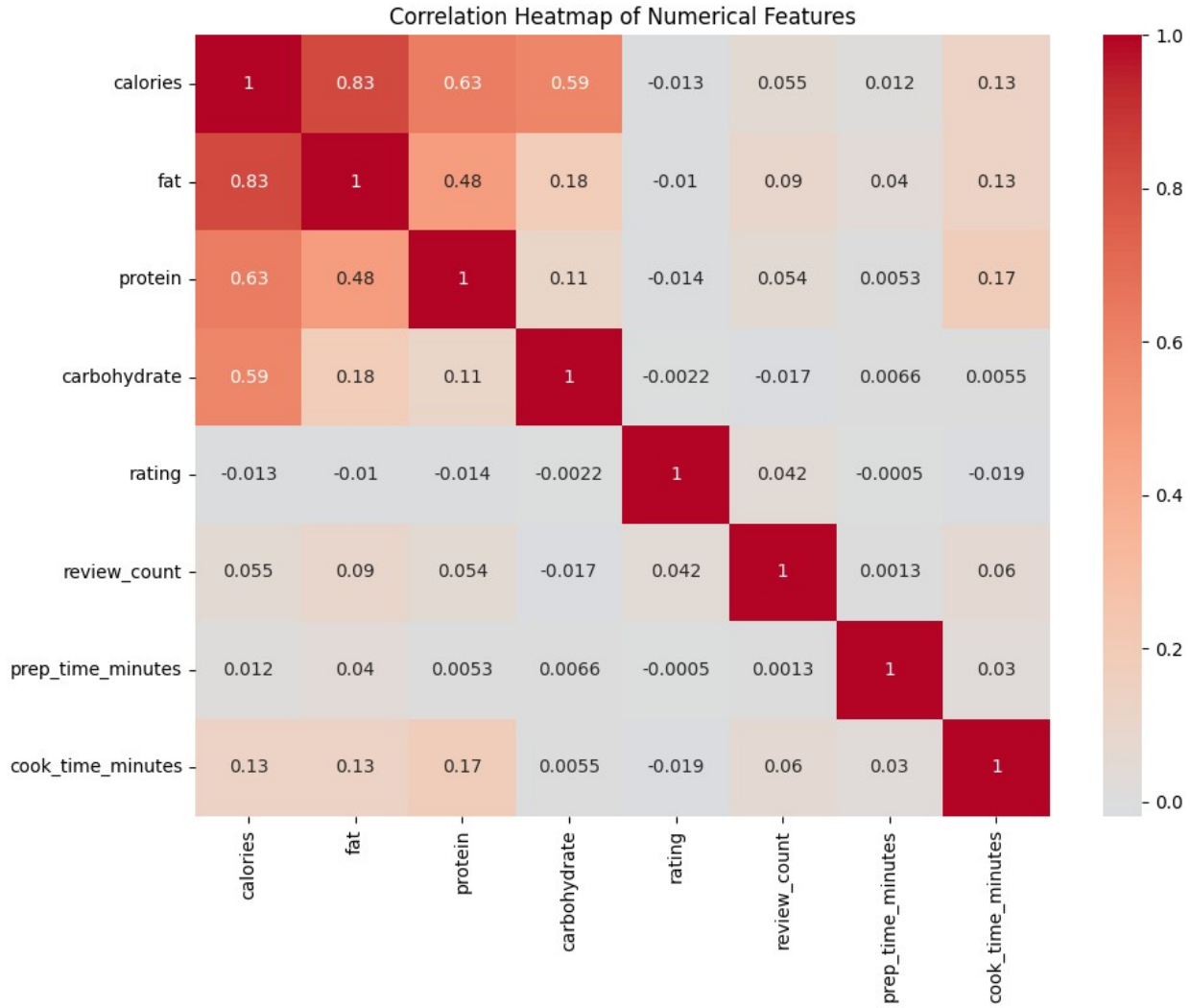
Heatmap showing relationships between numerical features:

- Nutrients: calories, fat, protein, carbohydrates
- User Data: ratings, review counts
- Time Metrics: prep time, cook time

Features:

- Cool-warm color scheme (negative to positive correlations)
- Annotated correlation values
- 10x8 figure size for clarity

```
# 6. Correlation Heatmap
plt.figure(figsize=(10, 8))
correlation_columns = ['calories', 'fat', 'protein', 'carbohydrate',
'rating', 'review_count', 'prep_time_minutes', 'cook_time_minutes']
correlation_matrix = df[correlation_columns].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Heatmap of Numerical Features')
plt.tight_layout()
plt.show()
```




```
# Print some basic statistics
print("\nBasic Statistics:")
print(df[numeric_columns + ['prep_time_minutes',
'cook_time_minutes']].describe())


Basic Statistics:
            calories            fat        protein   carbohydrate  \
count  124599.000000  120740.000000  121128.000000  121373.000000
mean      291.085405      11.934932      11.995641      33.693894
std       197.548262      13.083652      13.161884      26.229857
min         0.000000       0.000000       0.000000       0.000000
25%       151.200000       3.000000       3.000000      15.400000
50%       257.100000       8.000000       6.700000      30.000000
75%       393.000000      16.000000      18.000000      47.000000
max      6594.000000     437.900000     519.000000    2012.000000

       prep_time_minutes  cook_time_minutes
```

```
count       95432.000000       74572.000000
mean           16.568363          36.597302
std           104.554063          65.332459
min             0.000000           0.000000
25%            10.000000          15.000000
50%            15.000000          20.000000
75%            20.000000          40.000000
max         30240.000000        4325.000000
```

Total Recipe Time Analysis

Detailed visualization of total recipe duration (prep + cook time):

- Limited to recipes under 3 hours
- 60 bins for detailed distribution view
- Mean and median lines for reference
- Summary statistics included

Key Features:

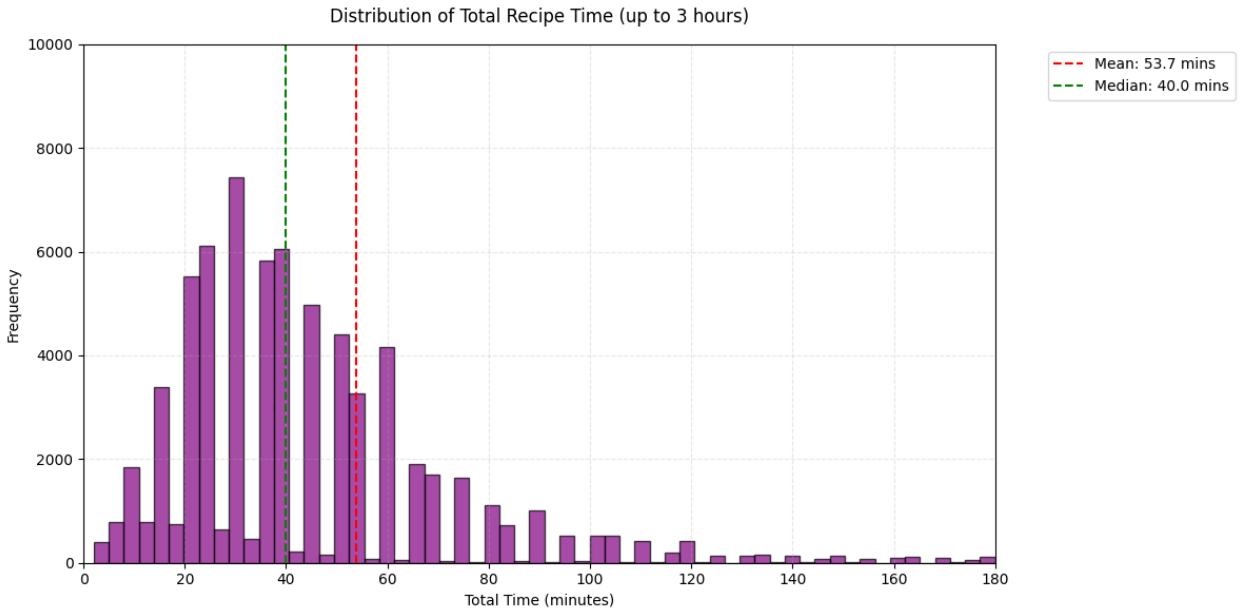
- Purple histogram with black edges
- Grid lines for readability
- Y-axis capped at 10,000 for detail
- Statistical overlay (mean/median)

Statistics printed:

- Total recipe count
- Mean, median times
- Time range (min to max)

```
# Calculate total time
df['total_time_minutes'] = df['prep_time_minutes'] +
df['cook_time_minutes']

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

# Create histogram with smaller bins and focus on more common
durations
plt.hist(df['total_time_minutes'].dropna()[df['total_time_minutes'] <=
180],  # Filter to ≤ 180 minutes
         bins=60,  # More bins for finer detail
         color='purple',
         edgecolor='black',
         alpha=0.7)

# Limit axes to show more detail
plt.xlim(0, 180)
plt.ylim(0, 10000)  # Adjust y-axis limit to show more detail
```

```python
# Add title and labels
plt.title('Distribution of Total Recipe Time (up to 3 hours)', pad=15,
fontsize=12)
plt.xlabel('Total Time (minutes)', fontsize=10)
plt.ylabel('Frequency', fontsize=10)

# Add grid for better readability
plt.grid(True, alpha=0.3, linestyle='--')

# Add mean and median lines
mean_time = df['total_time_minutes'].mean()
median_time = df['total_time_minutes'].median()
plt.axvline(mean_time, color='red', linestyle='--', label=f'Mean:
{mean_time:.1f} mins')
plt.axvline(median_time, color='green', linestyle='--',
label=f'Median: {median_time:.1f} mins')

# Add legend
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

# Adjust layout
plt.tight_layout()

plt.show()

# Optional: Print summary statistics
print(f"Number of recipes: {len(df['total_time_minutes'].dropna())}")
print(f"Mean time: {mean_time:.1f} minutes")
print(f"Median time: {median_time:.1f} minutes")
print(f"Min time: {df['total_time_minutes'].min():.1f} minutes")
print(f"Max time: {df['total_time_minutes'].max():.1f} minutes")
```




```
Number of recipes: 71775
Mean time: 53.7 minutes
Median time: 40.0 minutes
Min time: 2.0 minutes
Max time: 7245.0 minutes
```

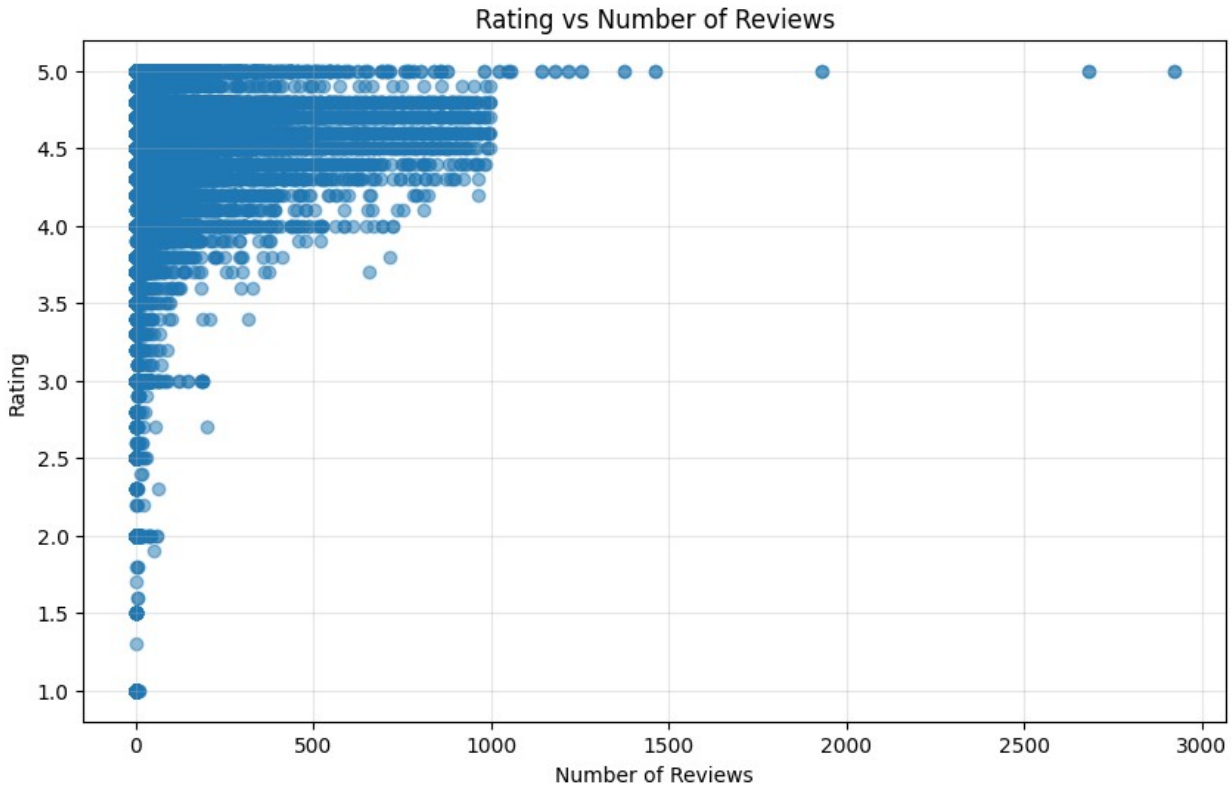
Rating vs Reviews Analysis

Scatter plot examining relationship between ratings and review counts:

- X-axis: Number of reviews per recipe
- Y-axis: Rating value (1-5)
- Alpha=0.5 for better visibility of overlapping points
- Grid lines added for reference

Purpose: Visualizes if highly reviewed recipes tend to have different ratings

```
plt.figure(figsize=(10, 6))
plt.scatter(df['review_count'], df['rating'], alpha=0.5)
plt.xlabel('Number of Reviews')
plt.ylabel('Rating')
plt.title('Rating vs Number of Reviews')
plt.grid(True, alpha=0.3)
plt.show()
```



Category Calorie Analysis

Horizontal bar plot showing average calories across top 10 recipe categories:

- Categories sorted by average calorie content
- Helps identify high vs low calorie recipe types
- Shows caloric patterns across popular categories

Features:

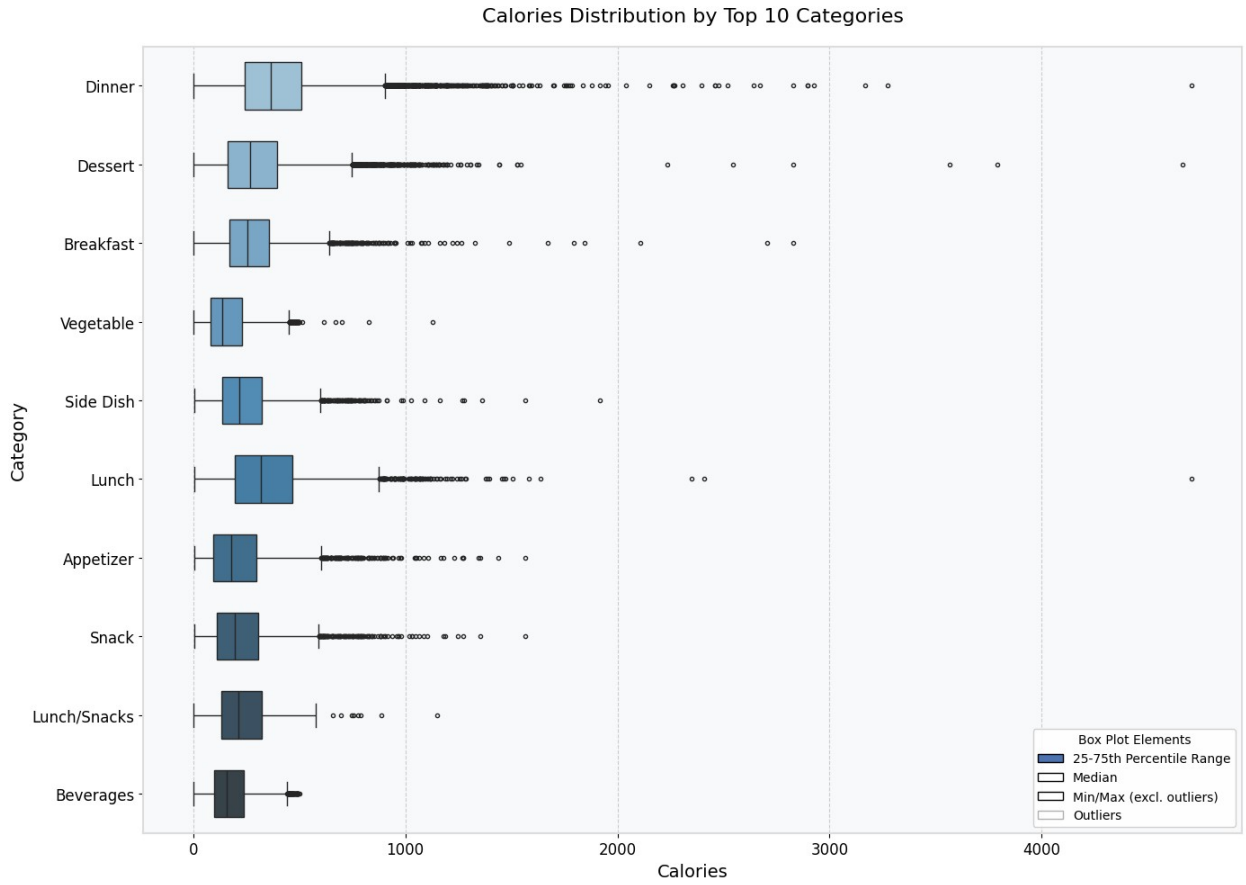
- Horizontal orientation for better category readability
- Sorted bars for easy comparison
- 12x6 figure size for clear visualization

```
# Get top 10 categories excluding 'uncategorized'
all_categories = df['category'].explode().value_counts()
top_categories =
all_categories[~all_categories.index.isin(['uncategorized'])].head(10)
.index

# Create DataFrame for box plot
box_plot_data = []
for _, row in df.iterrows():
    if isinstance(row['category'], list):
        for cat in row['category']:
            if cat in top_categories and cat != 'uncategorized':
```

```
                try:
                    calories = float(row['calories'])
                    if not pd.isna(calories):  # Skip NaN values
                        box_plot_data.append({
                            'Category': cat,
                            'Calories': calories
                        })
                except:
                    pass  # Skip if calories can't be converted to
float

# Convert to DataFrame
plot_df = pd.DataFrame(box_plot_data)

# Create enhanced box plot with clean visualization
plt.figure(figsize=(14, 10))

# Create box plot with improved styling
ax = sns.boxplot(data=plot_df,
                 x='Calories',
                 y='Category',
                 orient='h',
                 order=top_categories,
                 palette='Blues_d',  # Color palette
                 width=0.6,          # Box width
                 fliersize=3)        # Outlier point size

# Add a subtle grid for better readability
ax.grid(True, linestyle='--', alpha=0.6, axis='x')

# Customize the plot
plt.title('Calories Distribution by Top 10 Categories', fontsize=16,
pad=20)
plt.xlabel('Calories', fontsize=14)
plt.ylabel('Category', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add a subtle background color
ax.set_facecolor('#f8f9fa')

# Add a border to the plot
for spine in ax.spines.values():
    spine.set_edgecolor('#dddddd')
    spine.set_linewidth(1.5)

# Add a legend explaining the box plot
from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor='#4c72b0', edgecolor='black', label='25-75th
```

```
Percentile Range'),
    Patch(facecolor='white', edgecolor='black', label='Median'),
    Patch(facecolor='white', edgecolor='black', linestyle='-',
label='Min/Max (excl. outliers)'),
    Patch(facecolor='white', edgecolor='black', label='Outliers',
alpha=0.3)
]
plt.legend(handles=legend_elements, loc='lower right',
           title='Box Plot Elements', frameon=True, framealpha=0.9)

plt.tight_layout()
plt.show()

# Print statistics for each category
print("\nSummary Statistics for Each Category:")
for category in top_categories:
    if category in plot_df['Category'].values:
        stats = plot_df[plot_df['Category'] == category]
['Calories'].describe()
        print(f"\n{category}:")
        print(f"Count: {int(stats['count'])}")
        print(f"Mean: {int(stats['mean'])}")
        print(f"Median: {int(plot_df[plot_df['Category'] == category]
['Calories'].median())}")
        print(f"Min: {int(stats['min'])}")
        print(f"Max: {int(stats['max'])}")
```

```
C:\Users\nisha\AppData\Local\Temp\ipykernel_14096\4205353152.py:28:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  ax = sns.boxplot(data=plot_df,
```



Summary Statistics for Each Category:

Dinner:
Count: 20174
Mean: 397
Median: 365
Min: 1
Max: 4709

Dessert:
Count: 17362
Mean: 299
Median: 269
Min: 0
Max: 4664

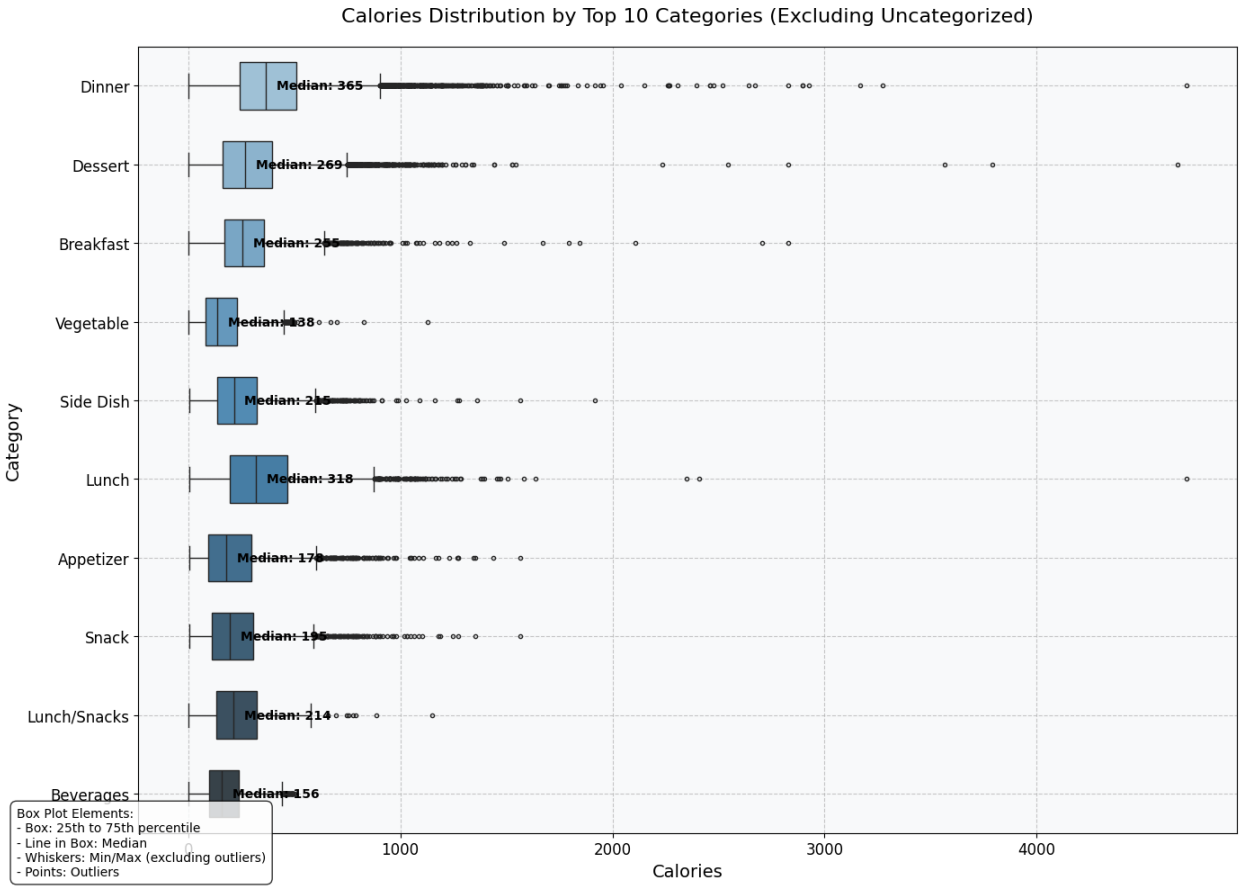
Breakfast:
Count: 6007
Mean: 277
Median: 255
Min: 0
Max: 2830

```
Vegetable:
Count: 4261
Mean: 167
Median: 138
Min: 0
Max: 1127

Side Dish:
Count: 4192
Mean: 245
Median: 215
Min: 3
Max: 1918

Lunch:
Count: 4002
Mean: 356
Median: 318
Min: 3
Max: 4709

Appetizer:
Count: 3540
Mean: 221
Median: 178
Min: 3
Max: 1564

Snack:
Count: 3220
Mean: 231
Median: 195
Min: 5
Max: 1564

Lunch/Snacks:
Count: 2930
Mean: 229
Median: 214
Min: 0
Max: 1150

Beverages:
Count: 2754
Mean: 176
Median: 156
Min: 0
Max: 499
```

```
# Get top 10 categories excluding 'uncategorized'
all_categories = df['category'].explode().value_counts()
top_categories =
all_categories[~all_categories.index.isin(['uncategorized'])].head(10)
.index

# Create DataFrame for box plot
box_plot_data = []
for _, row in df.iterrows():
    if isinstance(row['category'], list):
        for cat in row['category']:
            if cat in top_categories and cat != 'uncategorized':
                try:
                    calories = float(row['calories'])
                    if not pd.isna(calories):  # Skip NaN values
                        box_plot_data.append({
                            'Category': cat,
                            'Calories': calories
                        })
                except:
                    pass  # Skip if calories can't be converted to
float

# Convert to DataFrame
plot_df = pd.DataFrame(box_plot_data)

# Create enhanced box plot
plt.figure(figsize=(14, 10))  # Larger figure size

# Create box plot with improved styling
ax = sns.boxplot(data=plot_df,
                 x='Calories',
                 y='Category',
                 orient='h',
                 order=top_categories,
                 palette='Blues_d',  # Color palette
                 width=0.6,          # Box width
                 fliersize=3)        # Outlier point size

# Add a grid for better readability
ax.grid(True, linestyle='--', alpha=0.7)

# Add median values as text (with NaN handling)
for i, category in enumerate(top_categories):
    if category in plot_df['Category'].values:  # Check if category
exists in data
        median = plot_df[plot_df['Category'] == category]
['Calories'].median()
        if not pd.isna(median):  # Check if median is not NaN
            ax.text(median + 50, i, f'Median: {int(median)}',
```

```
                    verticalalignment='center', fontweight='bold')

# Customize the plot
plt.title('Calories Distribution by Top 10 Categories (Excluding
Uncategorized)', fontsize=16, pad=20)
plt.xlabel('Calories', fontsize=14)
plt.ylabel('Category', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add a subtle background color
ax.set_facecolor('#f8f9fa')

# Add statistics as text
stats_text = "Box Plot Elements:\n"
stats_text += "- Box: 25th to 75th percentile\n"
stats_text += "- Line in Box: Median\n"
stats_text += "- Whiskers: Min/Max (excluding outliers)\n"
stats_text += "- Points: Outliers"

plt.figtext(0.02, 0.02, stats_text, fontsize=10,
            bbox=dict(facecolor='white', alpha=0.8,
boxstyle='round,pad=0.5'))

plt.tight_layout()
plt.show()

# Print statistics for each category
print("\nSummary Statistics for Each Category:")
for category in top_categories:
    if category in plot_df['Category'].values:
        stats = plot_df[plot_df['Category'] == category]
['Calories'].describe()
        print(f"\n{category}:")
        print(f"Count: {int(stats['count'])}")
        print(f"Mean: {int(stats['mean'])}")
        print(f"Median: {int(plot_df[plot_df['Category'] == category]
['Calories'].median())}")
        print(f"Min: {int(stats['min'])}")
        print(f"Max: {int(stats['max'])}")
```

```
C:\Users\nisha\AppData\Local\Temp\ipykernel_14096\4170943501.py:28:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  ax = sns.boxplot(data=plot_df,
```

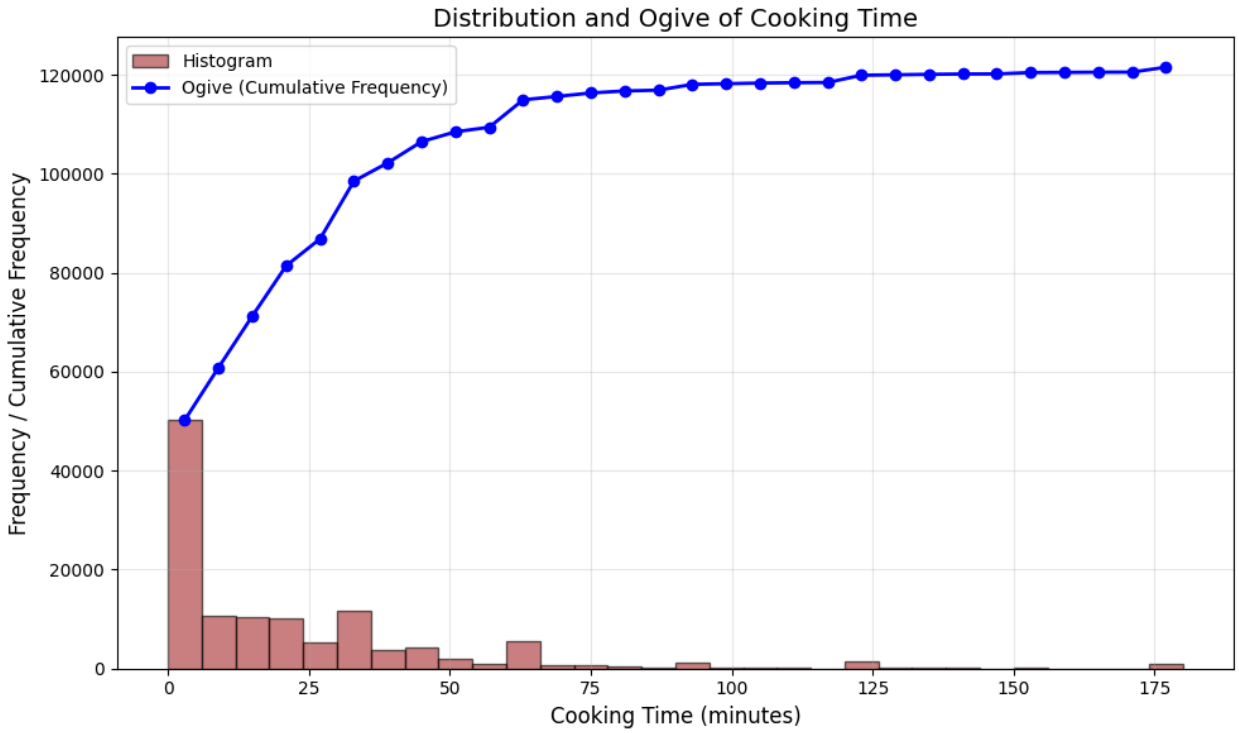



```
Summary Statistics for Each Category:

Dinner:
Count: 20174
Mean: 397
Median: 365
Min: 1
Max: 4709

Dessert:
Count: 17362
Mean: 299
Median: 269
Min: 0
Max: 4664

Breakfast:
Count: 6007
Mean: 277
Median: 255
Min: 0
Max: 2830
```

```
Vegetable:
Count: 4261
Mean: 167
Median: 138
Min: 0
Max: 1127

Side Dish:
Count: 4192
Mean: 245
Median: 215
Min: 3
Max: 1918

Lunch:
Count: 4002
Mean: 356
Median: 318
Min: 3
Max: 4709

Appetizer:
Count: 3540
Mean: 221
Median: 178
Min: 3
Max: 1564

Snack:
Count: 3220
Mean: 231
Median: 195
Min: 5
Max: 1564

Lunch/Snacks:
Count: 2930
Mean: 229
Median: 214
Min: 0
Max: 1150

Beverages:
Count: 2754
Mean: 176
Median: 156
Min: 0
Max: 499
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json

# --- Load data ---
with open('merged_recipes.json', 'r', encoding='utf-8') as f:
    merged_data = json.load(f)

df = pd.DataFrame(merged_data)

# --- Time conversion ---
def convert_time_to_minutes(time_str):
    if pd.isna(time_str):
        return None
    try:
        time_str = time_str.replace('PT', '')
        hours = 0
        minutes = 0
        if 'H' in time_str:
            hours = int(time_str.split('H')[0])
            time_str = time_str.split('H')[1]
        if 'M' in time_str:
            minutes = int(time_str.split('M')[0])
        return hours * 60 + minutes
    except:
        return None

df['cook_time_minutes'] =
df['cook_time'].apply(convert_time_to_minutes)

# --- Histogram data ---
cook_times = df['cook_time_minutes'].dropna()
bins = np.linspace(0, 180, 31)  # 30 bins from 0 to 180
freq, bin_edges = np.histogram(cook_times, bins=bins)

# Midpoints of each bin
midpoints = (bin_edges[:-1] + bin_edges[1:]) / 2

# Cumulative frequency for ogive
cumulative_freq = np.cumsum(freq)

# --- Plot histogram ---
plt.figure(figsize=(10, 6))
plt.hist(cook_times, bins=bins, edgecolor='black', color='brown',
alpha=0.6, label='Histogram')

# --- Plot ogive (cumulative line graph) ---
plt.plot(midpoints, cumulative_freq, marker='o', color='blue',
label='Ogive (Cumulative Frequency)', linewidth=2)
```

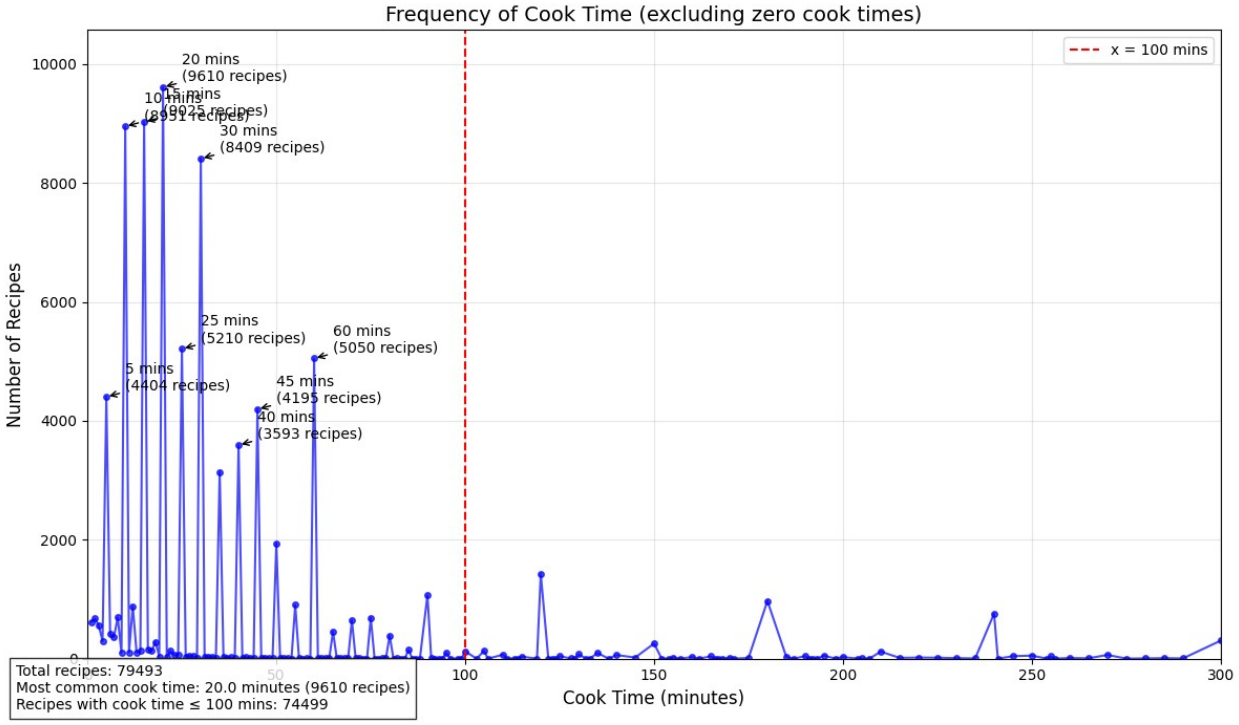
```
# Labels and styling
plt.title('Distribution and Ogive of Cooking Time', fontsize=14)
plt.xlabel('Cooking Time (minutes)', fontsize=12)
plt.ylabel('Frequency / Cumulative Frequency', fontsize=12)
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json

# --- Load data ---
with open('merged_recipes.json', 'r', encoding='utf-8') as f:
    merged_data = json.load(f)

df = pd.DataFrame(merged_data)

# --- Time Conversion ---
def convert_time_to_minutes(time_str):
    if pd.isna(time_str):
        return None
    try:
        time_str = time_str.replace('PT', '')
```

```
        hours = 0
        minutes = 0
        if 'H' in time_str:
            hours = int(time_str.split('H')[0])
            time_str = time_str.split('H')[1]
        if 'M' in time_str:
            minutes = int(time_str.split('M')[0])
        return hours * 60 + minutes
    except:
        return None

# Apply conversion
df['cook_time_minutes'] =
df['cook_time'].apply(convert_time_to_minutes)
df['total_time_minutes'] =
df['total_time'].apply(convert_time_to_minutes)

# --- Filter where cook_time_minutes <= 100 and > 0 ---
filtered_df = df[(df['cook_time_minutes'] > 0) &
(df['cook_time_minutes'] <= 300)].dropna(subset=['cook_time_minutes'])

# Get frequencies
sorted_times = filtered_df['cook_time_minutes'].sort_values().values
unique, counts = np.unique(sorted_times, return_counts=True)
frequencies = dict(zip(unique, counts))

# Create the plot
plt.figure(figsize=(12, 7))

# Plot the data
plt.plot(unique, counts, marker='o', linestyle='-', color='blue',
markersize=4, alpha=0.7)

# Add a vertical line at x = 100 to show the cutoff
plt.axvline(100, color='red', linestyle='--', linewidth=1.5, label='x
= 100 mins')

# Add annotations for peaks
peak_indices = np.where(counts > np.percentile(counts, 95))[0]
for idx in peak_indices:
    x_val = unique[idx]
    y_val = counts[idx]
    plt.annotate(f'{int(x_val)} mins\n({y_val} recipes)',
                 xy=(x_val, y_val),
                 xytext=(x_val+5, y_val+100),
                 arrowprops=dict(arrowstyle='->'))

# Styling
plt.title('Frequency of Cook Time (excluding zero cook times)',
fontsize=14)
```

```
plt.xlabel('Cook Time (minutes)', fontsize=12)
plt.ylabel('Number of Recipes', fontsize=12)
plt.grid(True, alpha=0.3)
plt.legend()

# Set axis limits
plt.xlim(0, 300)
plt.ylim(0, max(counts) * 1.1)  # Give some headroom above the highest
bar

# Add some context
plt.figtext(0.02, 0.02,
            f"Total recipes: {len(filtered_df)}\n"
            f"Most common cook time: {unique[np.argmax(counts)]}
minutes ({max(counts)} recipes)\n"
            f"Recipes with cook time ≤ 100 mins: {sum(counts[unique <=
100])}",
            fontsize=10, bbox=dict(facecolor='white', alpha=0.8))

plt.tight_layout()
plt.show()

# Print some statistics
print(f"Total recipes (excluding zero cook time): {len(filtered_df)}")
print(f"Most common cook time: {unique[np.argmax(counts)]} minutes
({max(counts)} recipes)")
print(f"Recipes with cook time ≤ 100 mins: {sum(counts[unique <=
100])} ({sum(counts[unique <= 100])/len(filtered_df)*100:.1f}%)")
```



```
Total recipes (excluding zero cook time): 79493
Most common cook time: 20.0 minutes (9610 recipes)
Recipes with cook time ≤ 100 mins: 74499 (93.7%)
```