mcdonalds main book

April 30, 2023

[1]:
```
pip install ipynbcompress
```

```
Collecting ipynbcompress
  Downloading ipynbcompress-0.4.0-py2.py3-none-any.whl (5.9 kB)
Requirement already satisfied: jsonschema in c:\users\dell\anaconda3\lib\site-
packages (from ipynbcompress) (4.17.3)
Requirement already satisfied: ipython in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
ipynbcompress) (8.11.0)
Requirement already satisfied: Pillow in c:\users\dell\anaconda3\lib\site-
packages (from ipynbcompress) (9.4.0)
Collecting hurry.filesize
  Downloading hurry.filesize-0.9.tar.gz (2.8 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting climate
  Downloading climate-0.1.0-py3-none-any.whl (1.1 kB)
Requirement already satisfied: setuptools in c:\users\dell\anaconda3\lib\site-
packages (from hurry.filesize->ipynbcompress) (65.6.3)
Requirement already satisfied: pygments>=2.4.0 in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
ipython->ipynbcompress) (2.14.0)
Requirement already satisfied: traitlets>=5 in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
ipython->ipynbcompress) (5.9.0)
Requirement already satisfied: backcall in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
ipython->ipynbcompress) (0.2.0)
Requirement already satisfied: prompt-toolkit!=3.0.37,<3.1.0,>=3.0.30 in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
ipython->ipynbcompress) (3.0.38)
Requirement already satisfied: colorama in c:\users\dell\anaconda3\lib\site-
packages (from ipython->ipynbcompress) (0.4.6)
Requirement already satisfied: matplotlib-inline in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
ipython->ipynbcompress) (0.1.6)
Requirement already satisfied: stack-data in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
```

```
ipython->ipynbcompress) (0.6.2)
Requirement already satisfied: jedi>=0.16 in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
ipython->ipynbcompress) (0.18.2)
Requirement already satisfied: pickleshare in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
ipython->ipynbcompress) (0.7.5)
Requirement already satisfied: decorator in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
ipython->ipynbcompress) (5.1.1)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in
c:\users\dell\anaconda3\lib\site-packages (from jsonschema->ipynbcompress)
(0.18.0)
Requirement already satisfied: attrs>=17.4.0 in
c:\users\dell\anaconda3\lib\site-packages (from jsonschema->ipynbcompress)
(22.1.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
jedi>=0.16->ipython->ipynbcompress) (0.8.3)
Requirement already satisfied: wcwidth in
c:\users\dell\appdata\roaming\python\python310\site-packages (from prompt-
toolkit!=3.0.37,<3.1.0,>=3.0.30->ipython->ipynbcompress) (0.2.6)
Requirement already satisfied: asttokens>=2.1.0 in
c:\users\dell\appdata\roaming\python\python310\site-packages (from stack-
data->ipython->ipynbcompress) (2.2.1)
Requirement already satisfied: pure-eval in
c:\users\dell\appdata\roaming\python\python310\site-packages (from stack-
data->ipython->ipynbcompress) (0.2.2)
Requirement already satisfied: executing>=1.2.0 in
c:\users\dell\appdata\roaming\python\python310\site-packages (from stack-
data->ipython->ipynbcompress) (1.2.0)
Requirement already satisfied: six in
c:\users\dell\appdata\roaming\python\python310\site-packages (from
asttokens>=2.1.0->stack-data->ipython->ipynbcompress) (1.16.0)
Building wheels for collected packages: hurry.filesize
  Building wheel for hurry.filesize (setup.py): started
  Building wheel for hurry.filesize (setup.py): finished with status 'done'
  Created wheel for hurry.filesize: filename=hurry.filesize-0.9-py3-none-any.whl
size=4140
sha256=6cc1c0c64b99b78022c69a5db544af0ebde7a5820ecf6a99c78dd18c52a3d7d0
  Stored in directory: c:\users\dell\appdata\local\pip\cache\wheels\e9\af\17\1c4
cd045d88f20d450522470819d85349c3388c151af64590b
Successfully built hurry.filesize
Installing collected packages: hurry.filesize, climate, ipynbcompress
Successfully installed climate-0.1.0 hurry.filesize-0.9 ipynbcompress-0.4.0
Note: you may need to restart the kernel to use updated packages.
```

[1]:
```
#Importing the libraries
import pandas as pd
import numpy as np
#Seaborn is a library for making statistical graphics in Python.
#It builds on top of matplotlib and integrates closely with pandas data␣
 ↪structures.
import seaborn as sns
#Matplotlib is a comprehensive library for creating static, animated, and␣
 ↪interactive visualizations in Python.
from matplotlib import pyplot as plt
```

[3]:
```
#Importing the dataset
data = pd.read_csv("mcdonalds.csv")
data
```

```
C:\Users\Dell\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3444:
DtypeWarning: Columns (6) have mixed types.Specify dtype option on import or set
low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

[3]:

	lat	lon	alt	is_broken	is_active	status	state	\
0	-73.988281	40.718830	0	False	True	working	NY	
1	-74.005090	40.728794	0	False	True	working	NY	
2	-73.993408	40.729197	0	False	True	working	NY	
3	-73.985855	40.726555	0	False	True	working	NY	
4	-73.991692	40.691383	0	True	True	broken	NY	
…	…	…	…	…	…	…	…	
16666	13.475643	52.514265	0	False	False	inactive	NaN	
16667	13.429812	54.076239	0	False	False	inactive	NaN	
16668	8.787059	53.100934	0	False	False	inactive	NaN	
16669	11.409059	53.628227	0	False	False	inactive	NaN	
16670	11.405999	53.903701	0	False	False	inactive	NaN	

	city	street	country	…	Carbohydrates	\
0	New York	114 Delancey St	USA	…	31	
1	New York	208 Varick St	USA	…	30	
2	New York	724 Broadway	USA	…	29	
3	New York	102 1st Ave	USA	…	30	
4	Brooklyn	82 Court St	USA	…	30	
…	…	…	…	…	…	
16666	Berlin	Frankfurter Allee 117	DE	…	65	
16667	Greifswald	Anklamer Landstr. 1	DE	…	80	
16668	Bremen	Waller Heerstr. 101	DE	…	98	
16669	Schwerin	Marienplatz 5-7	DE	…	64	
16670	Wismar	Zierower Landstr. 3	DE	…	79	

	Carbohydrates (% Daily Value)	Dietary Fiber	Dietary (% Daily Value)	\

```
0                                10              4                       17
1                                10              4                       17
2                                10              4                       17
3                                10              4                       17
4                                10              4                       17
…                                …               …                       …
16666                            22              1                        3
16667                            27              1                        4
16668                            33              1                        4
16669                            21              0                        0
16670                            26              0                        0

       Sugars  Protein  Vitamin A (% Daily Value)  Vitamin C (% Daily Value)  \
0           3       17                         10                          0
1           3       18                          6                          0
2           2       14                          8                          0
3           2       21                         15                          0
4           2       21                          6                          0
…       …       …                          …                          …
16666      57        7                         15                          0
16667      71        9                         15                          0
16668      88       11                         20                          0
16669      57        7                         15                          0
16670      71        9                         20                          0

       Calcium (% Dailly Value)  Iron (% Daily Value)
0                            25                    15
1                            25                     8
2                            25                    10
3                            30                    15
4                            25                    10
…                           …                     …
16666                        20                     4
16667                        25                     4
16668                        35                     4
16669                        25                     2
16670                        30                     2

[16671 rows x 35 columns]
```

[4]:
```
#To view the data type of the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16671 entries, 0 to 16670
Data columns (total 35 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
```

```
 0   lat                              16671 non-null  float64
 1   lon                              16671 non-null  float64
 2   alt                              16671 non-null  int64
 3   is_broken                        16671 non-null  bool
 4   is_active                        16671 non-null  bool
 5   status                           16671 non-null  object
 6   state                            12725 non-null  object
 7   city                             16663 non-null  object
 8   street                           16671 non-null  object
 9   country                          16671 non-null  object
 10  last_checked                     16671 non-null  object
 11  Category                         16671 non-null  object
 12  Item                             16671 non-null  object
 13  Serving Sizes                    16671 non-null  object
 14  Calories                         16671 non-null  int64
 15  Calories from Fat                16671 non-null  int64
 16  Total  Fat                       16671 non-null  float64
 17  Total fat (% Daily Value)        16671 non-null  int64
 18  Saturated  Fat                   16671 non-null  float64
 19  Saturated Fat(% Daily Value)     16671 non-null  int64
 20  Trans Fat                        16671 non-null  float64
 21  Cholesterol                      16671 non-null  int64
 22  Cholesterol (% Daily Value)      16671 non-null  int64
 23  Sodium                           16671 non-null  int64
 24  Sodium(% Daily Value)            16671 non-null  int64
 25  Carbohydrates                    16671 non-null  int64
 26  Carbohydrates (% Daily Value)    16671 non-null  int64
 27  Dietary Fiber                    16671 non-null  int64
 28  Dietary (% Daily Value)          16671 non-null  int64
 29  Sugars                           16671 non-null  int64
 30  Protein                          16671 non-null  int64
 31  Vitamin A (% Daily Value)        16671 non-null  int64
 32  Vitamin C (% Daily Value)        16671 non-null  int64
 33  Calcium (% Dailly Value)         16671 non-null  int64
 34  Iron (% Daily Value)             16671 non-null  int64
dtypes: bool(2), float64(5), int64(19), object(9)
memory usage: 4.2+ MB
```

[5]:
```
# Describing
data.describe()
```

[5]:

	lat	lon	alt	Calories	\
count	16671.000000	16671.000000	16671.000000	16671.000000	
mean	-76.600800	40.406232	0.001380	366.630976	
std	37.309030	7.250517	0.037119	240.198535	
min	-159.368738	19.517590	0.000000	0.000000	
25%	-95.572842	34.927834	0.000000	210.000000	

```
50%      -83.780231     40.213555      0.000000    340.000000
75%      -75.261739     44.860442      0.000000    500.000000
max       14.968594     64.859406      1.000000   1880.000000

       Calories from Fat    Total  Fat  Total fat (% Daily Value)  \
count       16671.000000  16671.000000                16671.000000
mean          126.768580     14.449013                   21.856877
std           127.538673     14.903415                   21.824655
min             0.000000      0.000000                    0.000000
25%            20.000000      2.500000                    4.000000
50%           100.000000     11.000000                   17.000000
75%           200.000000     23.000000                   35.000000
max          1060.000000    118.000000                  182.000000

       Saturated  Fat  Saturated Fat(% Daily Value)    Trans Fat   …  \
count    16671.000000                  16671.000000  16671.000000  …
mean         6.057735                     30.030412      0.531042  …
std          5.376344                     26.562317      5.342884  …
min          0.000000                      0.000000      0.000000  …
25%          1.000000                      5.000000      0.000000  …
50%          5.000000                     24.000000      0.000000  …
75%         10.000000                     48.000000      0.000000  …
max         21.000000                    102.000000     87.000000  …

       Carbohydrates  Carbohydrates (% Daily Value)  Dietary Fiber  \
count   16671.000000                   16671.000000   16671.000000
mean       47.382341                      15.916322       1.802711
std        28.147142                       9.731002       3.192167
min         0.000000                       0.000000       0.000000
25%        30.000000                      10.000000       0.000000
50%        44.000000                      15.000000       1.000000
75%        60.000000                      20.000000       3.000000
max       141.000000                      59.000000      47.000000

       Dietary (% Daily Value)        Sugars       Protein  \
count             16671.000000  16671.000000  16671.000000
mean                  6.848179     29.519765     13.463080
std                   8.107610     28.666848     11.555046
min                   0.000000      0.000000      0.000000
25%                   0.000000      6.000000      4.000000
50%                   5.000000     18.000000     12.000000
75%                  10.000000     48.000000     19.000000
max                  90.000000    128.000000     87.000000

       Vitamin A (% Daily Value)  Vitamin C (% Daily Value)  \
count               16671.000000               16671.000000
mean                   13.566553                   8.586408
```

```
std                    24.411895                  26.276445
min                     0.000000                   0.000000
25%                     2.000000                   0.000000
50%                     8.000000                   0.000000
75%                    15.000000                   4.000000
max                   170.000000                 240.000000

       Calcium (% Dailly Value)  Iron (% Daily Value)
count              16671.000000          16671.000000
mean                  21.010437              7.754184
std                   16.970142              8.697815
min                    0.000000              0.000000
25%                    6.000000              0.000000
50%                   20.000000              4.000000
75%                   30.000000             15.000000
max                   70.000000             40.000000

[8 rows x 24 columns]
```

[6]:
```
print('Dimension of the dataset   ' + str(data.shape))
```

```
Dimension of the dataset   (16671, 35)
```

It has 16671 rows and 35 columns in the given dataset

[7]:
```
#to view the number of coloumns and rows
data.shape
```

[7]:
```
(16671, 35)
```

[8]:
```
#to view the total number of coloumns
data.columns
```

[8]:
```
Index(['lat', 'lon', 'alt', 'is_broken', 'is_active', 'status', 'state',
       'city', 'street', 'country', 'last_checked', 'Category', 'Item',
       'Serving Sizes', 'Calories', 'Calories from Fat', 'Total  Fat',
       'Total fat (% Daily Value)', 'Saturated  Fat',
       'Saturated Fat(% Daily Value)', 'Trans Fat ', 'Cholesterol',
       'Cholesterol (% Daily Value)', 'Sodium ', 'Sodium(% Daily Value)',
       'Carbohydrates', 'Carbohydrates (% Daily Value)', 'Dietary Fiber',
       'Dietary (% Daily Value)', 'Sugars', 'Protein',
       'Vitamin A (% Daily Value)', 'Vitamin C (% Daily Value)',
       'Calcium (% Dailly Value)', 'Iron (% Daily Value)'],
      dtype='object')
```

[9]:
```
# NUMERIC FEATURES
#To check the staststical figures
data.describe()
```

[9]:

	lat	lon	alt	Calories	\
count	16671.000000	16671.000000	16671.000000	16671.000000	
mean	-76.600800	40.406232	0.001380	366.630976	
std	37.309030	7.250517	0.037119	240.198535	
min	-159.368738	19.517590	0.000000	0.000000	
25%	-95.572842	34.927834	0.000000	210.000000	
50%	-83.780231	40.213555	0.000000	340.000000	
75%	-75.261739	44.860442	0.000000	500.000000	
max	14.968594	64.859406	1.000000	1880.000000	

	Calories from Fat	Total Fat	Total fat (% Daily Value)	\
count	16671.000000	16671.000000	16671.000000	
mean	126.768580	14.449013	21.856877	
std	127.538673	14.903415	21.824655	
min	0.000000	0.000000	0.000000	
25%	20.000000	2.500000	4.000000	
50%	100.000000	11.000000	17.000000	
75%	200.000000	23.000000	35.000000	
max	1060.000000	118.000000	182.000000	

	Saturated Fat	Saturated Fat(% Daily Value)	Trans Fat	…	\
count	16671.000000	16671.000000	16671.000000	…	
mean	6.057735	30.030412	0.531042	…	
std	5.376344	26.562317	5.342884	…	
min	0.000000	0.000000	0.000000	…	
25%	1.000000	5.000000	0.000000	…	
50%	5.000000	24.000000	0.000000	…	
75%	10.000000	48.000000	0.000000	…	
max	21.000000	102.000000	87.000000	…	

	Carbohydrates	Carbohydrates (% Daily Value)	Dietary Fiber	\
count	16671.000000	16671.000000	16671.000000	
mean	47.382341	15.916322	1.802711	
std	28.147142	9.731002	3.192167	
min	0.000000	0.000000	0.000000	
25%	30.000000	10.000000	0.000000	
50%	44.000000	15.000000	1.000000	
75%	60.000000	20.000000	3.000000	
max	141.000000	59.000000	47.000000	

	Dietary (% Daily Value)	Sugars	Protein	\
count	16671.000000	16671.000000	16671.000000	
mean	6.848179	29.519765	13.463080	
std	8.107610	28.666848	11.555046	
min	0.000000	0.000000	0.000000	
25%	0.000000	6.000000	4.000000	
50%	5.000000	18.000000	12.000000	

```
75%                  10.000000     48.000000     19.000000
max                  90.000000    128.000000     87.000000

       Vitamin A (% Daily Value)  Vitamin C (% Daily Value)  \
count               16671.000000               16671.000000
mean                   13.566553                   8.586408
std                    24.411895                  26.276445
min                     0.000000                   0.000000
25%                     2.000000                   0.000000
50%                     8.000000                   0.000000
75%                    15.000000                   4.000000
max                   170.000000                 240.000000

       Calcium (% Dailly Value)  Iron (% Daily Value)
count               16671.000000          16671.000000
mean                   21.010437              7.754184
std                    16.970142              8.697815
min                     0.000000              0.000000
25%                     6.000000              0.000000
50%                    20.000000              4.000000
75%                    30.000000             15.000000
max                    70.000000             40.000000

[8 rows x 24 columns]
```

1 DEALING WITH NULL VALUES

[10]:
```
#To view the null values and the count of the null values
#A null value in a relational database is used when the value in a column is␣
 ↪unknown or missing.
data.isnull().sum()
```

[10]:
```
lat                    0
lon                    0
alt                    0
is_broken              0
is_active              0
status                 0
state               3946
city                   8
street                 0
country                0
last_checked           0
Category               0
Item                   0
Serving Sizes          0
```

```
Calories                            0
Calories from Fat                   0
Total  Fat                          0
Total fat (% Daily Value)           0
Saturated  Fat                      0
Saturated Fat(% Daily Value)        0
Trans Fat                           0
Cholesterol                         0
Cholesterol (% Daily Value)         0
Sodium                              0
Sodium(% Daily Value)               0
Carbohydrates                       0
Carbohydrates (% Daily Value)       0
Dietary Fiber                       0
Dietary (% Daily Value)             0
Sugars                              0
Protein                             0
Vitamin A (% Daily Value)           0
Vitamin C (% Daily Value)           0
Calcium (% Dailly Value)            0
Iron (% Daily Value)                0
dtype: int64
```

[11]:
```
#dropping the null values in the state coloumn
#dropna() function is used to remove missing values
data.dropna(axis=0, inplace=True, subset=['state'])
```

[12]:
```
# returns the number of missing values in the data set.
data.isnull().sum()
```

[12]:
```
lat                             0
lon                             0
alt                             0
is_broken                       0
is_active                       0
status                          0
state                           0
city                            0
street                          0
country                         0
last_checked                    0
Category                        0
Item                            0
Serving Sizes                   0
Calories                        0
Calories from Fat               0
Total  Fat                      0
```

```
Total fat (% Daily Value)        0
Saturated  Fat                   0
Saturated Fat(% Daily Value)     0
Trans Fat                        0
Cholesterol                      0
Cholesterol (% Daily Value)      0
Sodium                           0
Sodium(% Daily Value)            0
Carbohydrates                    0
Carbohydrates (% Daily Value)    0
Dietary Fiber                    0
Dietary (% Daily Value)          0
Sugars                           0
Protein                          0
Vitamin A (% Daily Value)        0
Vitamin C (% Daily Value)        0
Calcium (% Dailly Value)         0
Iron (% Daily Value)             0
dtype: int64
```

[13]:
```
#dropping the null values in the city coloumn
data.dropna(axis=0, inplace=True, subset=['city'])
```

[14]:
```
data.isnull().sum()
```

[14]:
```
lat                              0
lon                              0
alt                              0
is_broken                        0
is_active                        0
status                           0
state                            0
city                             0
street                           0
country                          0
last_checked                     0
Category                         0
Item                             0
Serving Sizes                    0
Calories                         0
Calories from Fat                0
Total  Fat                       0
Total fat (% Daily Value)        0
Saturated  Fat                   0
Saturated Fat(% Daily Value)     0
Trans Fat                        0
Cholesterol                      0
```

```
Cholesterol (% Daily Value)      0
Sodium                           0
Sodium(% Daily Value)            0
Carbohydrates                    0
Carbohydrates (% Daily Value)    0
Dietary Fiber                    0
Dietary (% Daily Value)          0
Sugars                           0
Protein                          0
Vitamin A (% Daily Value)        0
Vitamin C (% Daily Value)        0
Calcium (% Dailly Value)         0
Iron (% Daily Value)             0
dtype: int64
```

[15]:
```
#Converting the float datatype to int datatype in children coloumn
data['Calories'] = data['Calories'].astype(np.int64)
```

[16]:
```
#unique() function is used to find the unique elements of an array.
data['Calories'].unique()
```

[16]:
```
array([ 300,  250,  370,  450,  400,  430,  460,  520,  410,  470,  480,
        510,  570,  540,  420,  550,  500,  620,  670,  740,  800,  640,
        690, 1090, 1150,  990, 1050,  350,  150,  290,  260,  530,  600,
        610,  750,  240,  720,  380,  440,  590,  360,  630,  190,  280,
        940, 1880,  390,  140,  220,  340,  330,  230,  110,   20,   15,
        160,   45,  200,  100,    0,  270,  130,   80,  180,  170,  210,
        310,  320,  120,  760,  660,  820,  850,  560,  700,  650,  930,
        810,   25], dtype=int64)
```

2 INFERENTIAL STATASTICS

[17]:
```
#The highest Trans Fat value's average daily rate
print('The highest range of average Trans Fat daily rate is: {}'.
  ↪format(data['Trans Fat '].max()),'\n','The name of that country is: {}'.
  ↪format(data.iloc[data['Trans Fat '].idxmax()]['country']))
```

```
The highest range of average Trans Fat daily rate is: 87.0
 The name of that country is: USA
```

[18]:
```
# The average range of 'Trans Fat value'
print('The average range of Trans Fat  value is: {}'.format(round(data['Trans␣
  ↪Fat '].mean(),2)))
```

```
The average range of Trans Fat  value is: 0.53
```

The average range of Trans Fat value is 0.53. which on an average the daily rate of a trans fat value in usa

[19]:
```
# The mean of proteins'
print('The Mean of Protein is : {}'.format(round(data['Protein'].mean())))
```

```
The Mean of Protein is : 13
```

[20]:
```
# The median of proteins'
#median the middle value in a set of data
print('The median of Protein is : {}'.format(round(data['Protein'].median())))
```

```
The median of Protein is : 12
```

[21]:
```
# The mode of proteins'
# mode The most frequent number-that is, the number that occurs the highest␣
 ↪number of times.
print('The mode of Protein is : {}'.format(round(data['Protein'].mode())))
```

```
The mode of Protein is : 0    0
dtype: int64
```

3 Information about the dataset - Object Type - Categorical Variable - Unique

[22]:
```
print(data['Category'].unique())
```

```
['Break Fast' 'Beef & Pork' 'Chicken & Fish' 'Salads' 'Snacks & Sides'
 'Desserts' 'Beverages' 'Coffee & Tea' 'Smoothies & Shakes']
```

4 Plot graphically which food categories have the highest and lowest varieties

We already know the highest from above max frequency is Coffee & Tea, from describe function output. We will reconfirm our findings below.

[23]:
```
#How many menu items are in each cateogry?
data.Category.value_counts()
```

[23]:
```
Chicken & Fish        4200
Beverages             1804
Smoothies & Shakes    1789
Coffee & Tea          1545
Salads                1323
Desserts              1283
Snacks & Sides         577
Beef & Pork            180
Break Fast              24
Name: Category, dtype: int64
```

[24]:
```
#How many menu items are in each cateogry? - different way
#The count() method is one of Python's built-in functions.
#It returns the number of times a given value occurs in a string or a list
food_categories = data.groupby('Category').count()
food_categories_count = food_categories[['Item']].sort_values('Item',ascending=␣
 ↪False)
food_categories_count
```

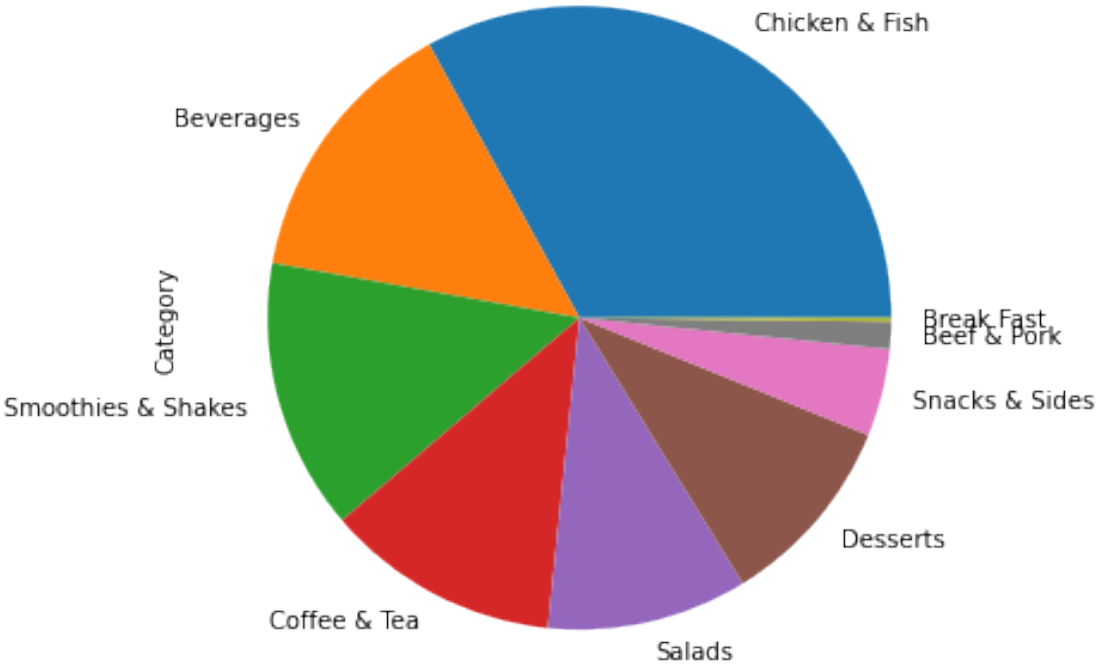
[24]:

Category	Item
Chicken & Fish	4200
Beverages	1804
Smoothies & Shakes	1789
Coffee & Tea	1545
Salads	1323
Desserts	1283
Snacks & Sides	577
Beef & Pork	180
Break Fast	24

5 UNIVARIATE ANALYSIS

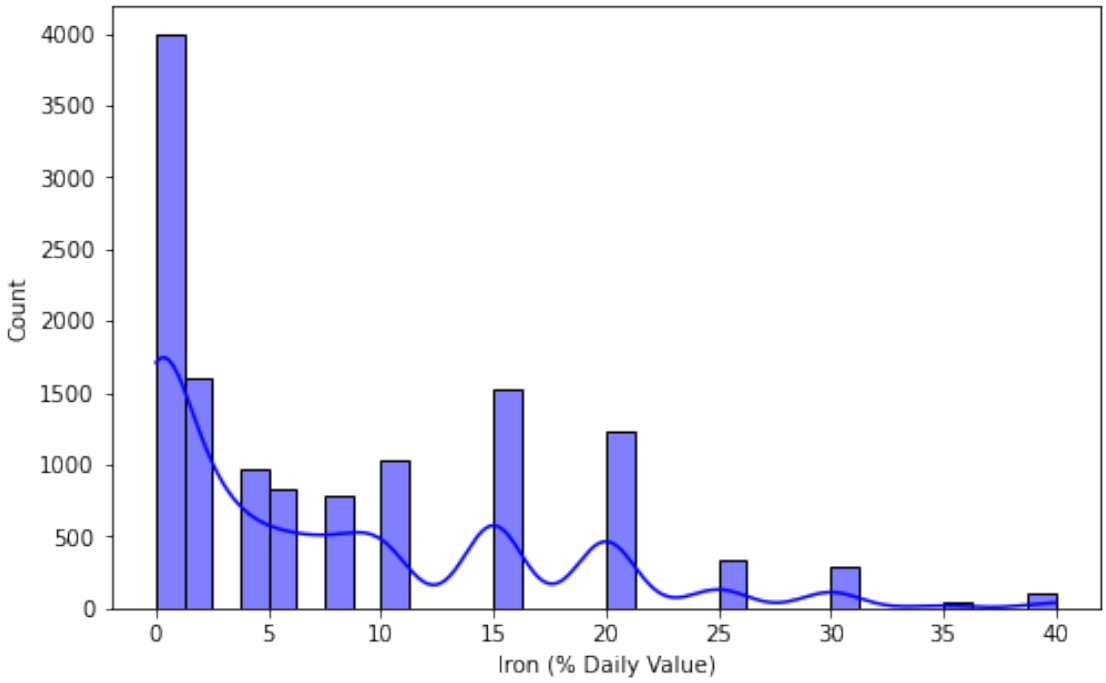
[25]:
```
#Create a pie chart that includes the relative proportions of each cateogry of␣
 ↪food (Pie chart for the "Category" column using the value_counts() method.
plt.figure(figsize=(9,6))
data.Category.value_counts().plot.pie()
```

[25]:
```
<AxesSubplot:ylabel='Category'>
```

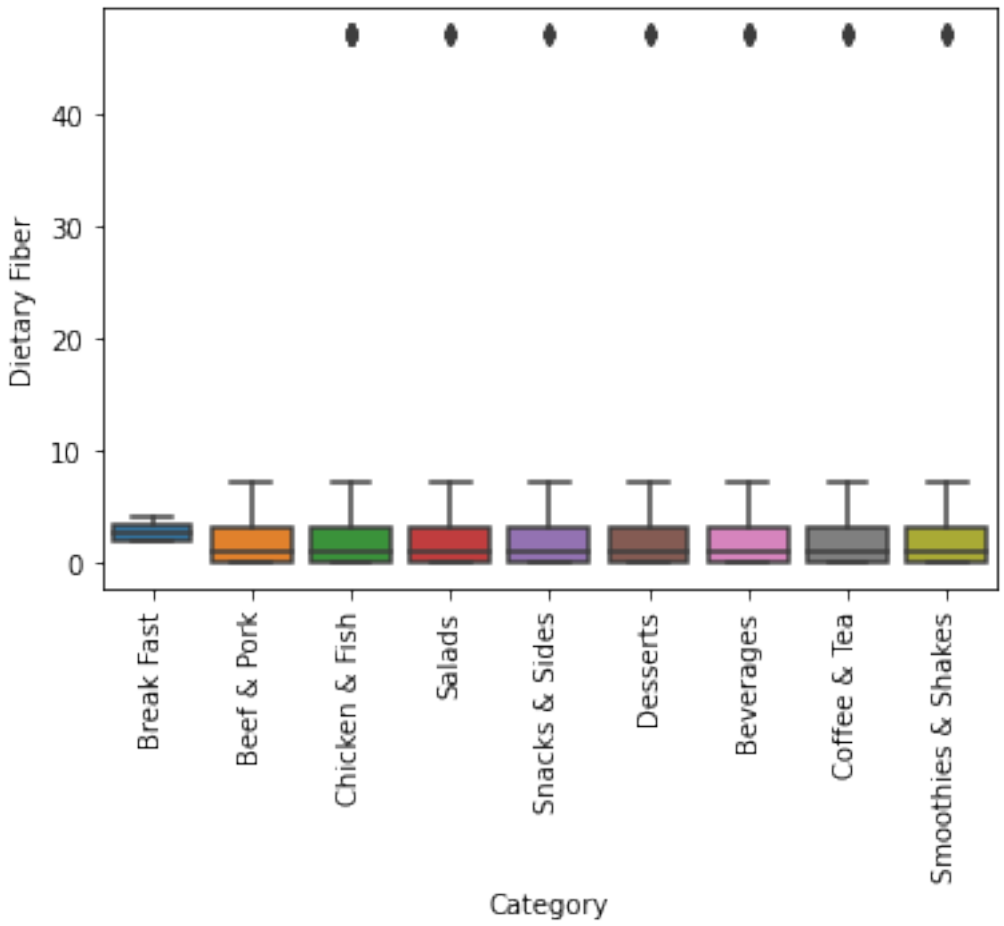


[26]:
```
# with histogram - sns.histplot(kde=TRUE)
#A histogram is a graph that shows the frequency of numerical data using␣
 ↪rectangles.
plt.figure(figsize=(8, 5))
plt.xlabel='count'
plt.ylabel='Iron (% Daily Value)'
sns.histplot(x='Iron (% Daily Value)', data=data, kde=True, color='blue')
```

[26]:
```
<AxesSubplot:xlabel='Iron (% Daily Value)', ylabel='Count'>
```



[27]:
```
#displaying the distribution of data based on a five number summary
#("minimum", first quartile [Q1], median, third quartile [Q3] and "maximum").
sns.boxplot(data= data, x = 'Category',y = 'Dietary Fiber')
plt.xticks(rotation=90)
plt.show()
```

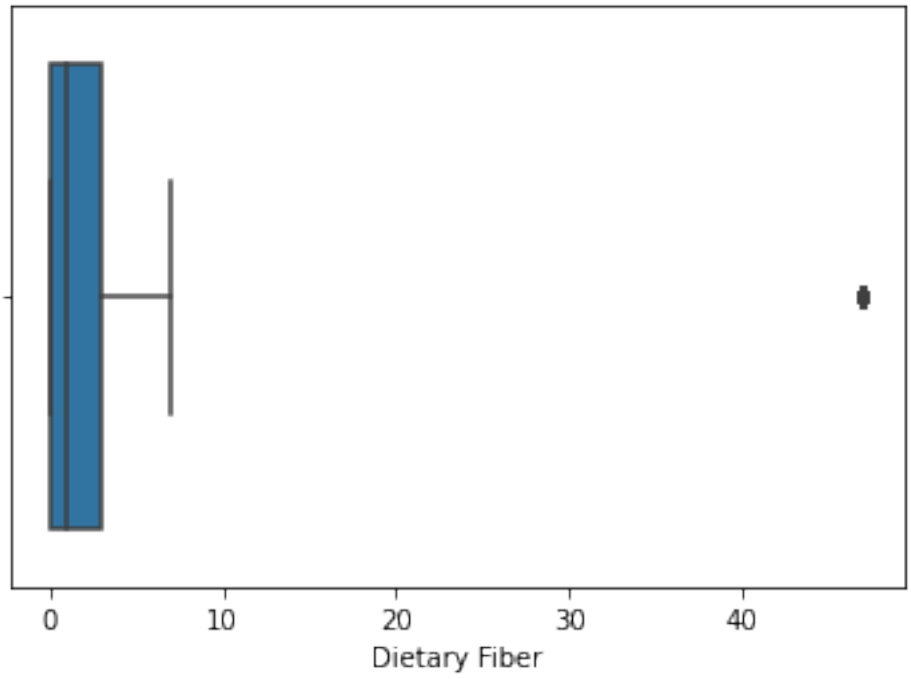


The above boxplot shows very clearly Salad contribution to the dietary fibre.

[28]:
```
#boxplot
sns.boxplot(data['Dietary Fiber'])
```

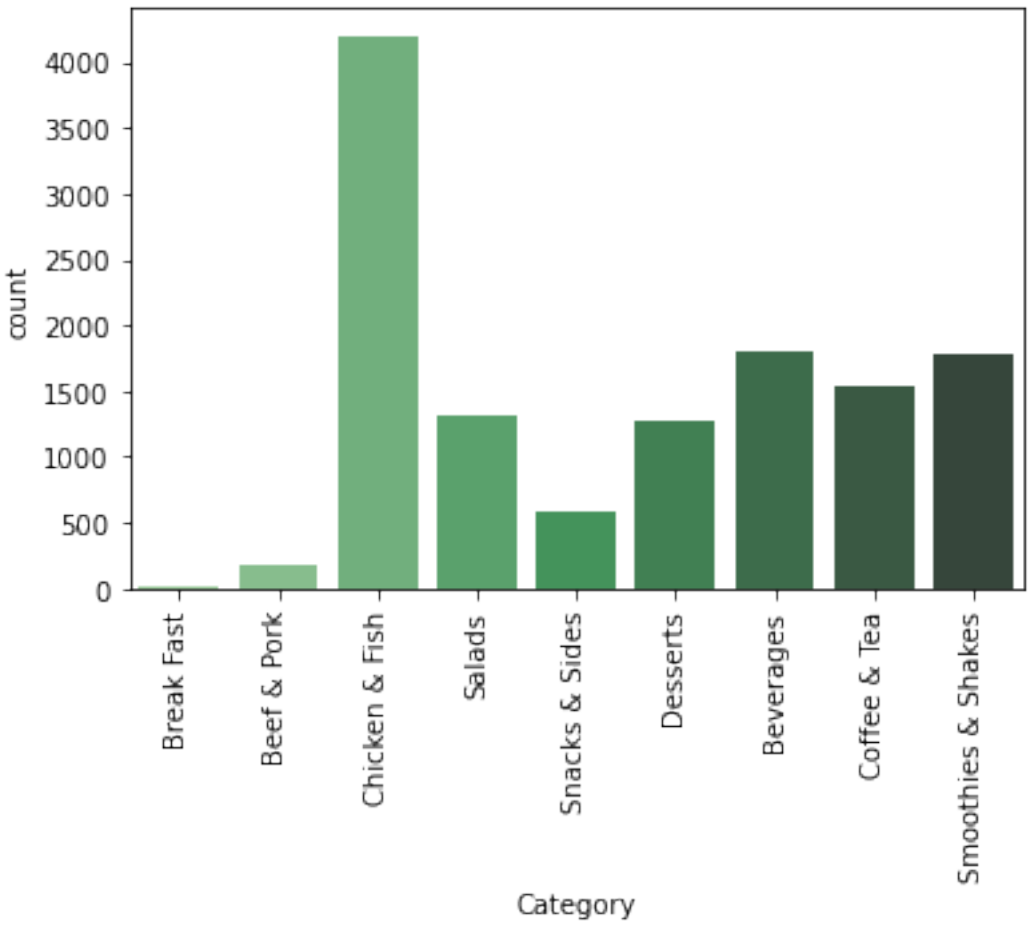
```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[28]:
```
<AxesSubplot:xlabel='Dietary Fiber'>
```



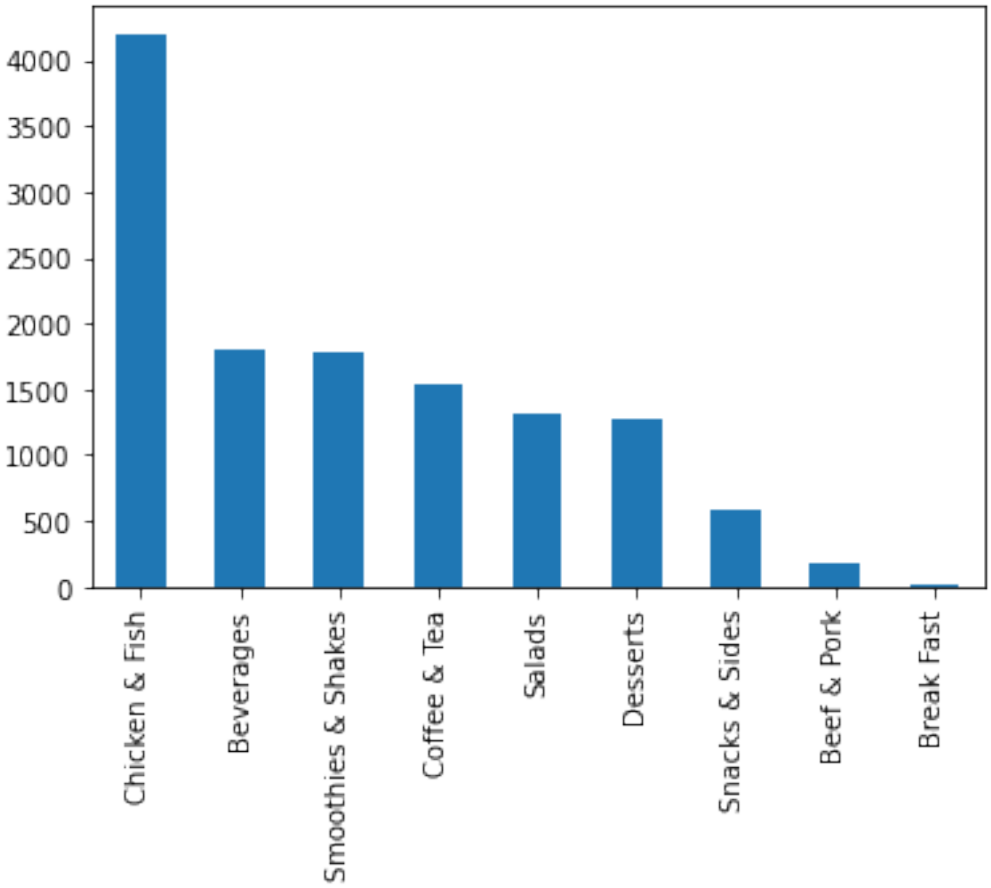
[29]:
```
# Count plot for meal categories
# count plot Show the counts of observations in each categorical bin using bars.
g = sns.countplot(x="Category", data=data, palette="Greens_d");
g.set_xticklabels(g.get_xticklabels(), rotation=90)
```

[29]:
```
[Text(0, 0, 'Break Fast'),
 Text(1, 0, 'Beef & Pork'),
 Text(2, 0, 'Chicken & Fish'),
 Text(3, 0, 'Salads'),
 Text(4, 0, 'Snacks & Sides'),
 Text(5, 0, 'Desserts'),
 Text(6, 0, 'Beverages'),
 Text(7, 0, 'Coffee & Tea'),
 Text(8, 0, 'Smoothies & Shakes')]
```



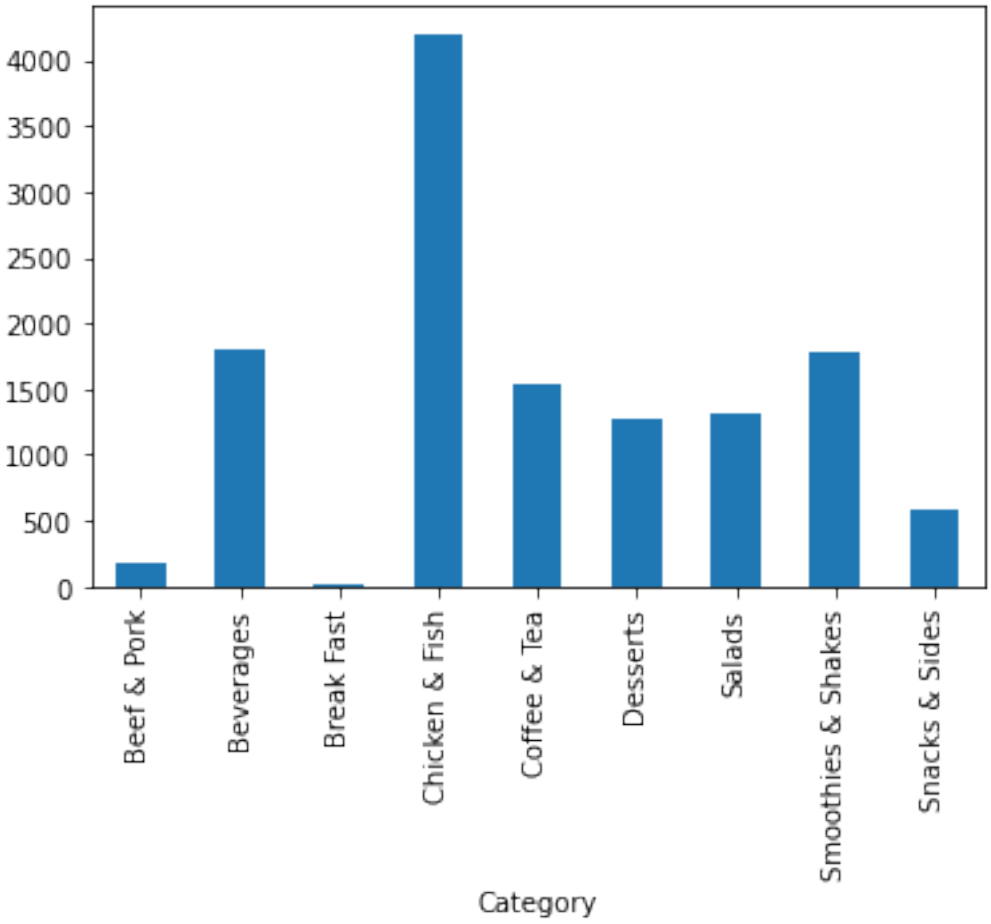
[30]:
```
data.Category.value_counts().plot.bar()
```

[30]: <AxesSubplot:>



[31]:
```
data.groupby('Category')['Item'].count().plot(kind='bar')
```

[31]: <AxesSubplot:xlabel='Category'>



6 McDonald's mostly sells coffee and tea and sells a small amount of Salads in comparison

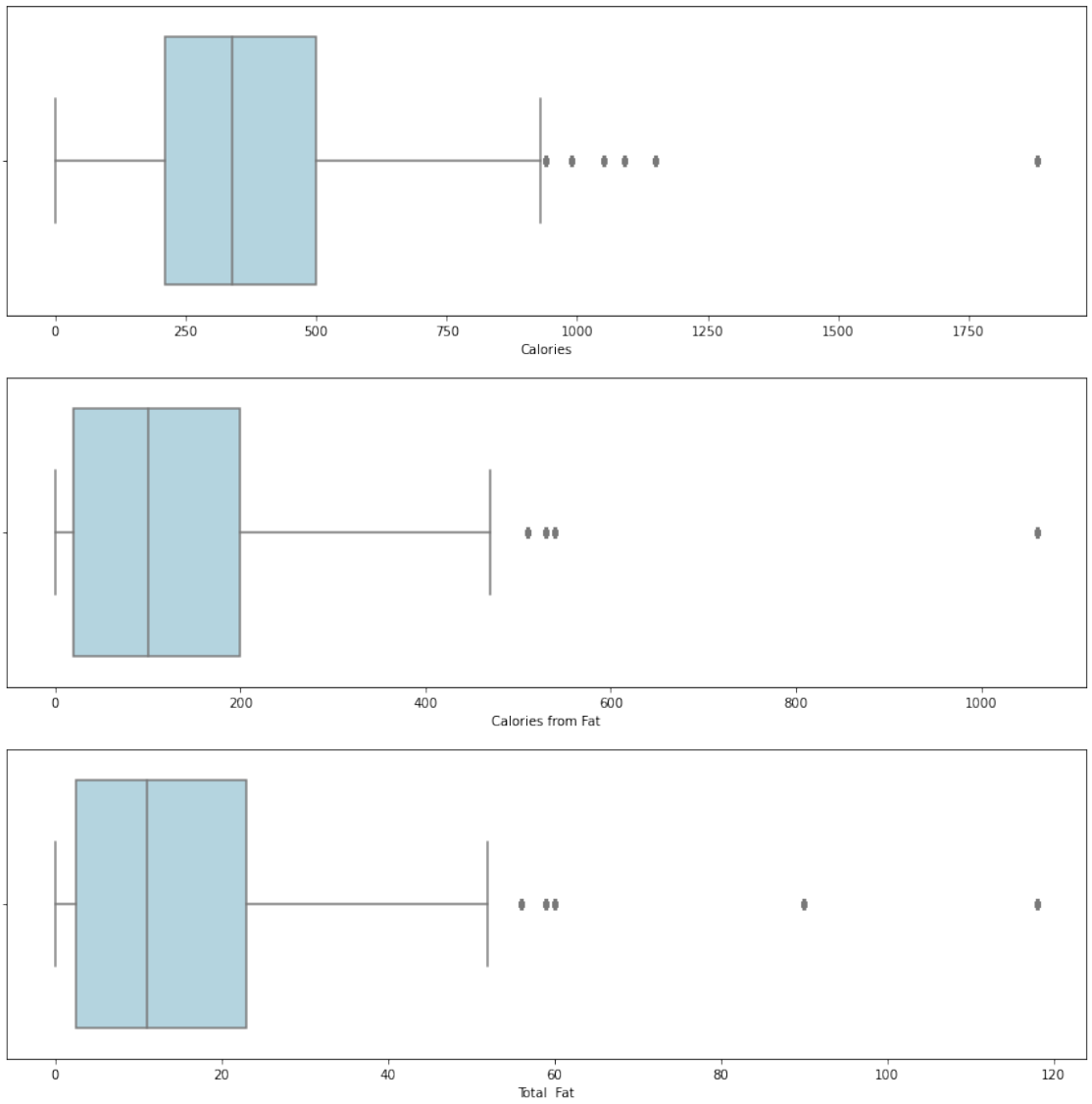
I had no idea that the mass majority of the items on McDonald's menu were coffee and Tea, But I'm not surprised the item they sell the least of is Salads 36 % of McDonald s menu is comprised of Coffee and Tea while only 2 % is comprised of salads

7 Which all variables have an outlier?

Dealing with outliers

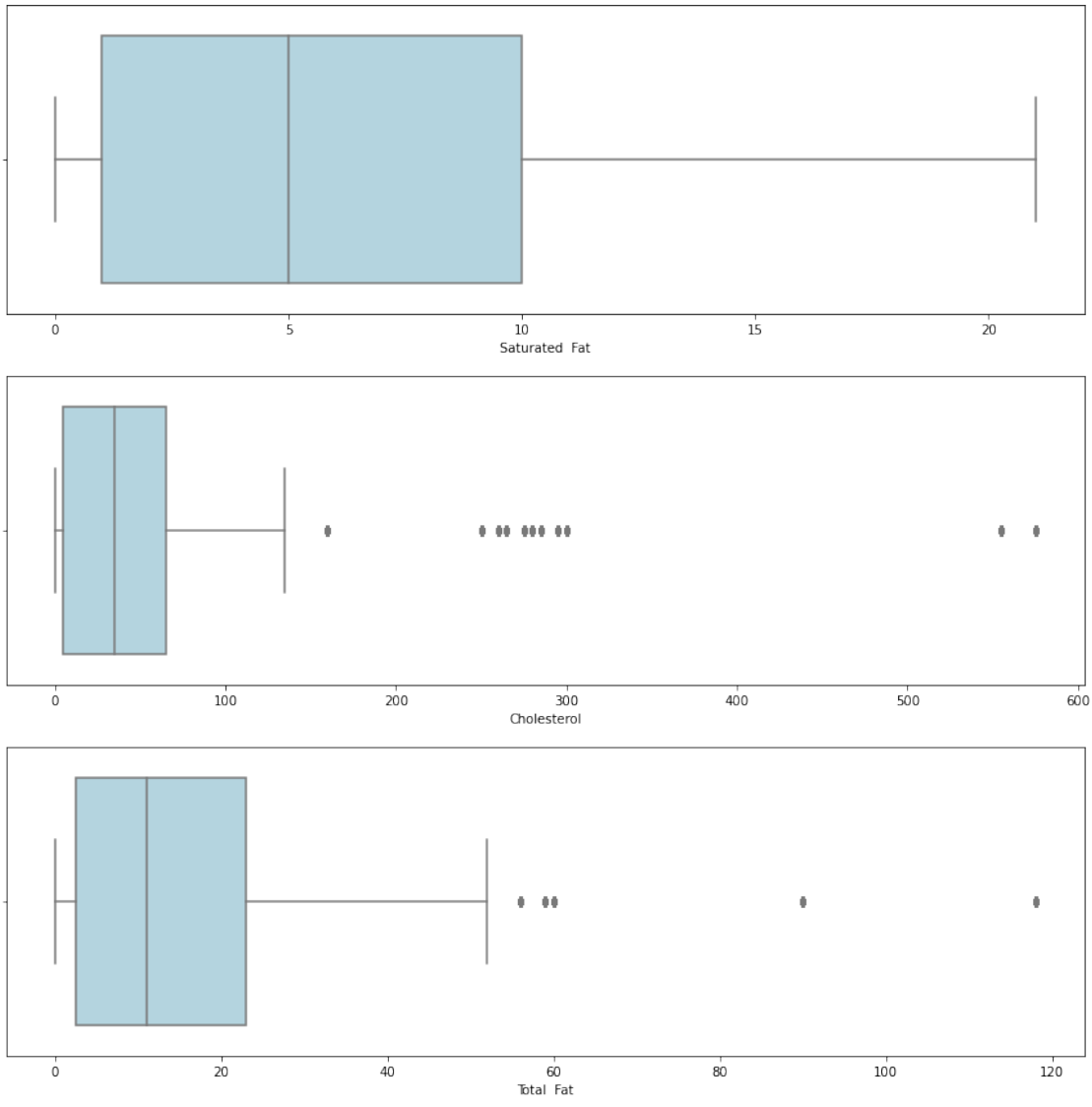
[32]:
```
#Checking for the outliers - first set
plt.figure(figsize= (15,15))
plt.subplot(3,1,1)
sns.boxplot(x= data['Calories'], color='lightblue')

plt.subplot(3,1,2)
```

```
sns.boxplot(x= data['Calories from Fat'], color='lightblue')

plt.subplot(3,1,3)
sns.boxplot(x= data['Total  Fat'], color='lightblue')


plt.show()
```



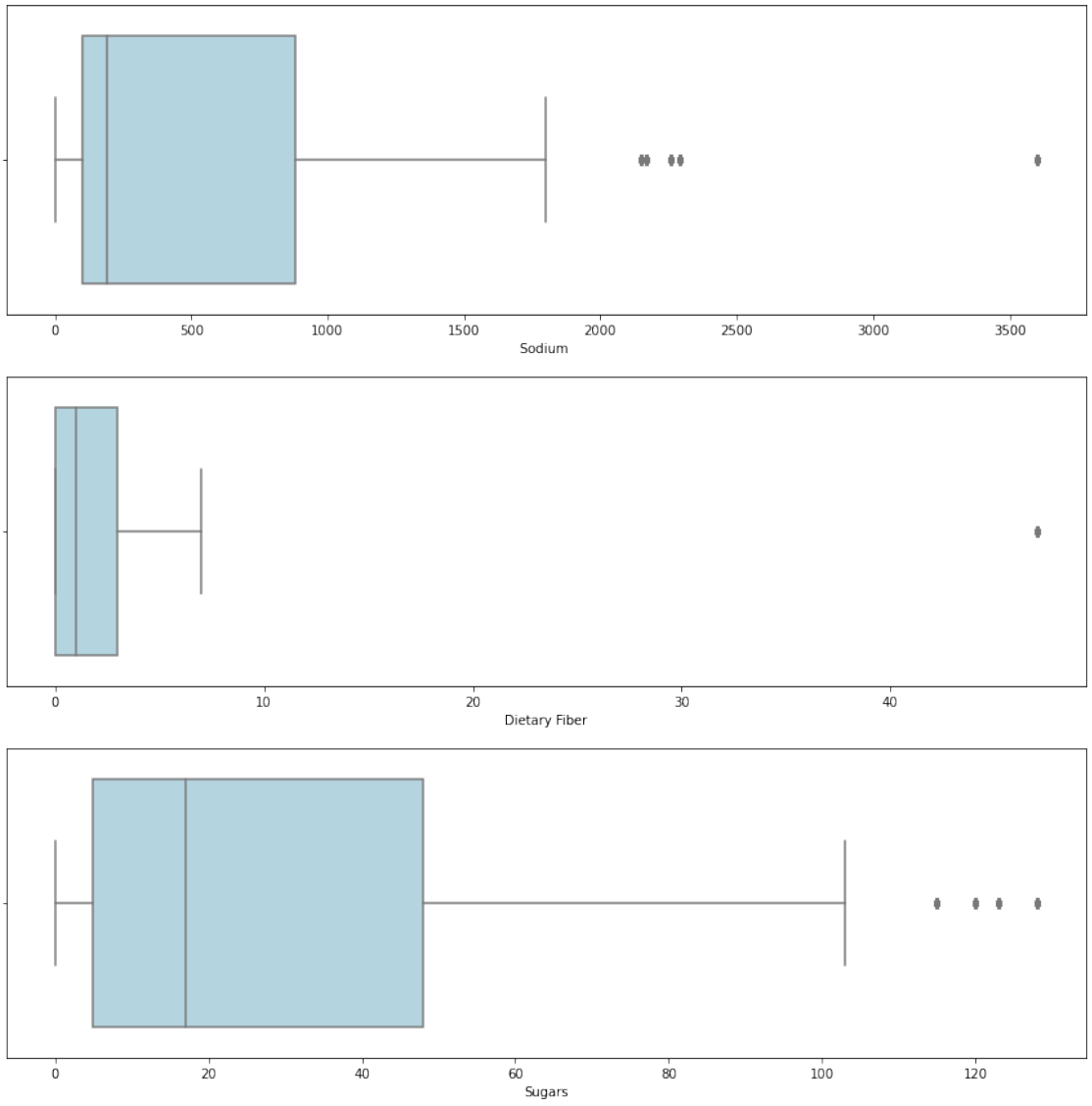
[33]:
```
#Checking for the outliers - second set
plt.figure(figsize= (15,15))
plt.subplot(3,1,1)
sns.boxplot(x= data['Saturated  Fat'], color='lightblue')
```

```
plt.subplot(3,1,2)
sns.boxplot(x= data['Cholesterol'], color='lightblue')

plt.subplot(3,1,3)
sns.boxplot(x= data['Total  Fat'], color='lightblue')


plt.show()
```

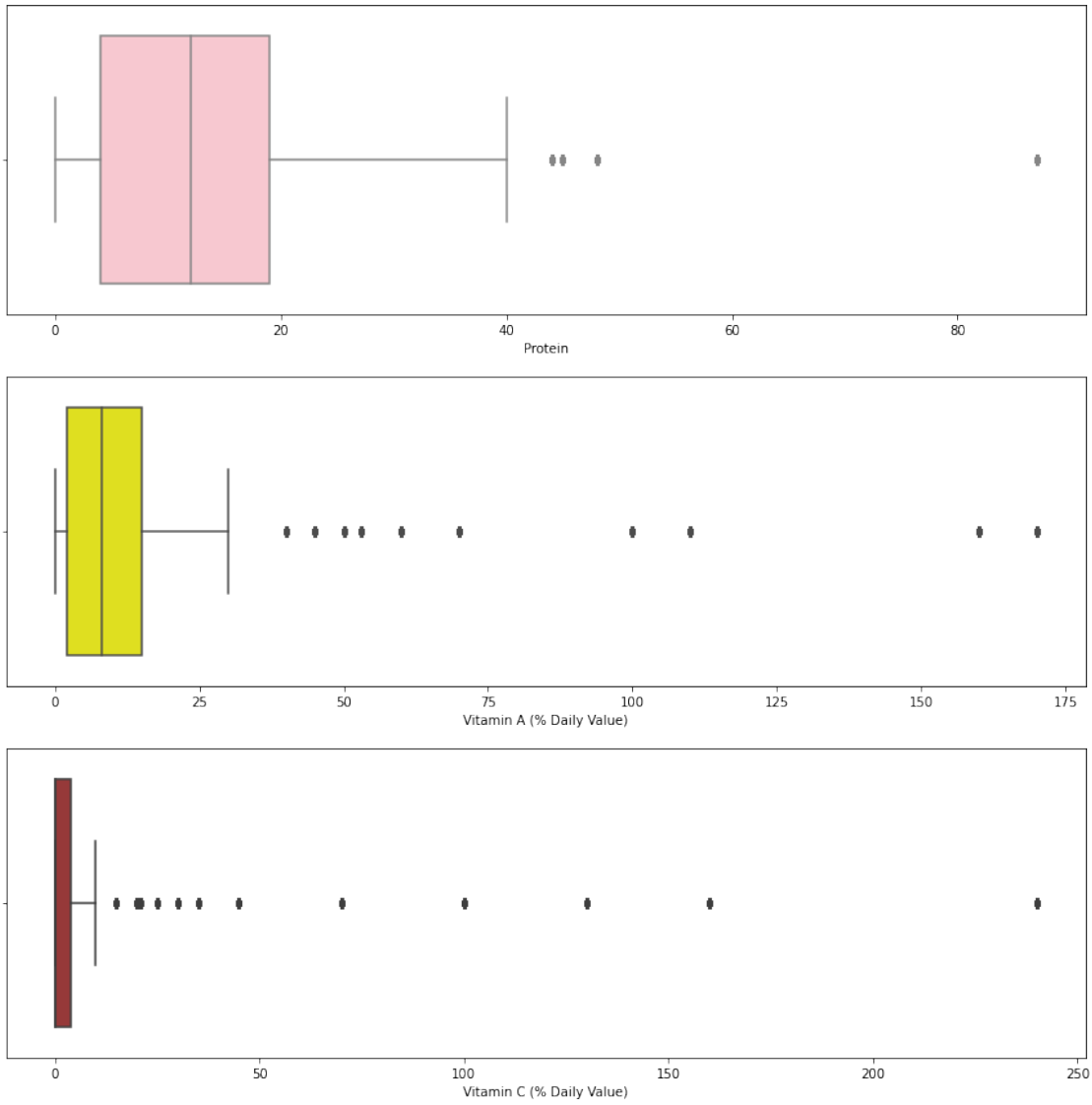


[34]:
```
#Checking for the outliers - third set
plt.figure(figsize= (15,15))
```

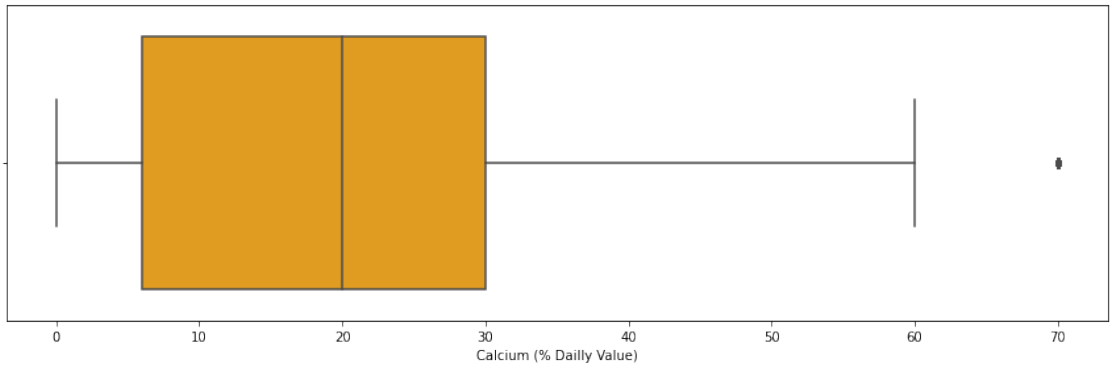
```
plt.subplot(3,1,1)
sns.boxplot(x= data['Sodium '], color='lightblue')

plt.subplot(3,1,2)
sns.boxplot(x= data['Dietary Fiber'], color='lightblue')

plt.subplot(3,1,3)
sns.boxplot(x= data['Sugars'], color='lightblue')


plt.show()
```

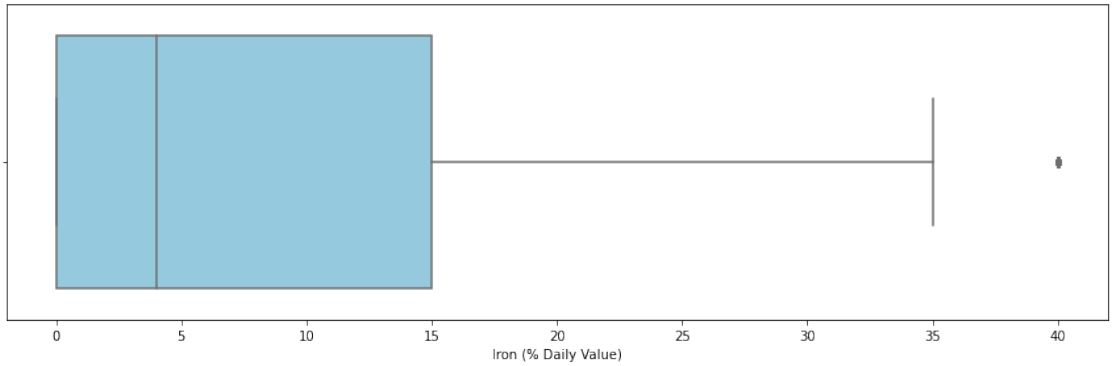


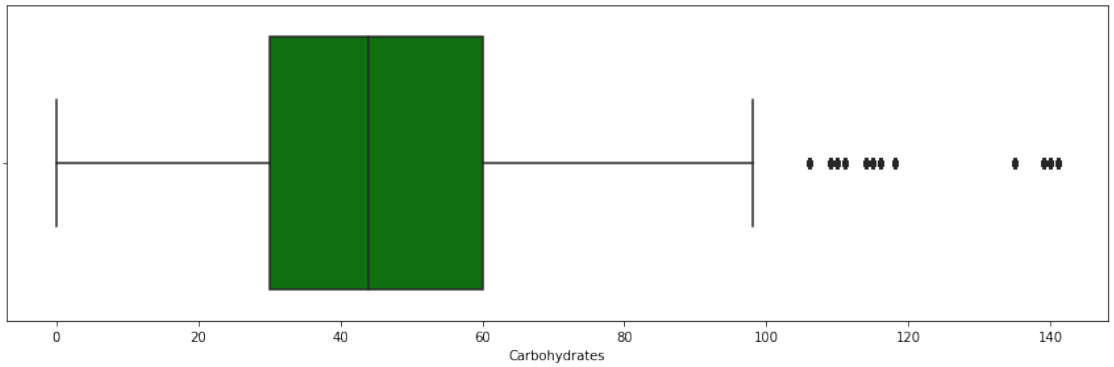
[35]:
```
#Checking for the outliers - fourth set
plt.figure(figsize= (15,15))
plt.subplot(3,1,1)
sns.boxplot(x= data['Protein'], color='pink')

plt.subplot(3,1,2)
sns.boxplot(x= data['Vitamin A (% Daily Value)'], color='yellow')

plt.subplot(3,1,3)
sns.boxplot(x= data['Vitamin C (% Daily Value)'], color='brown')


plt.show()
```



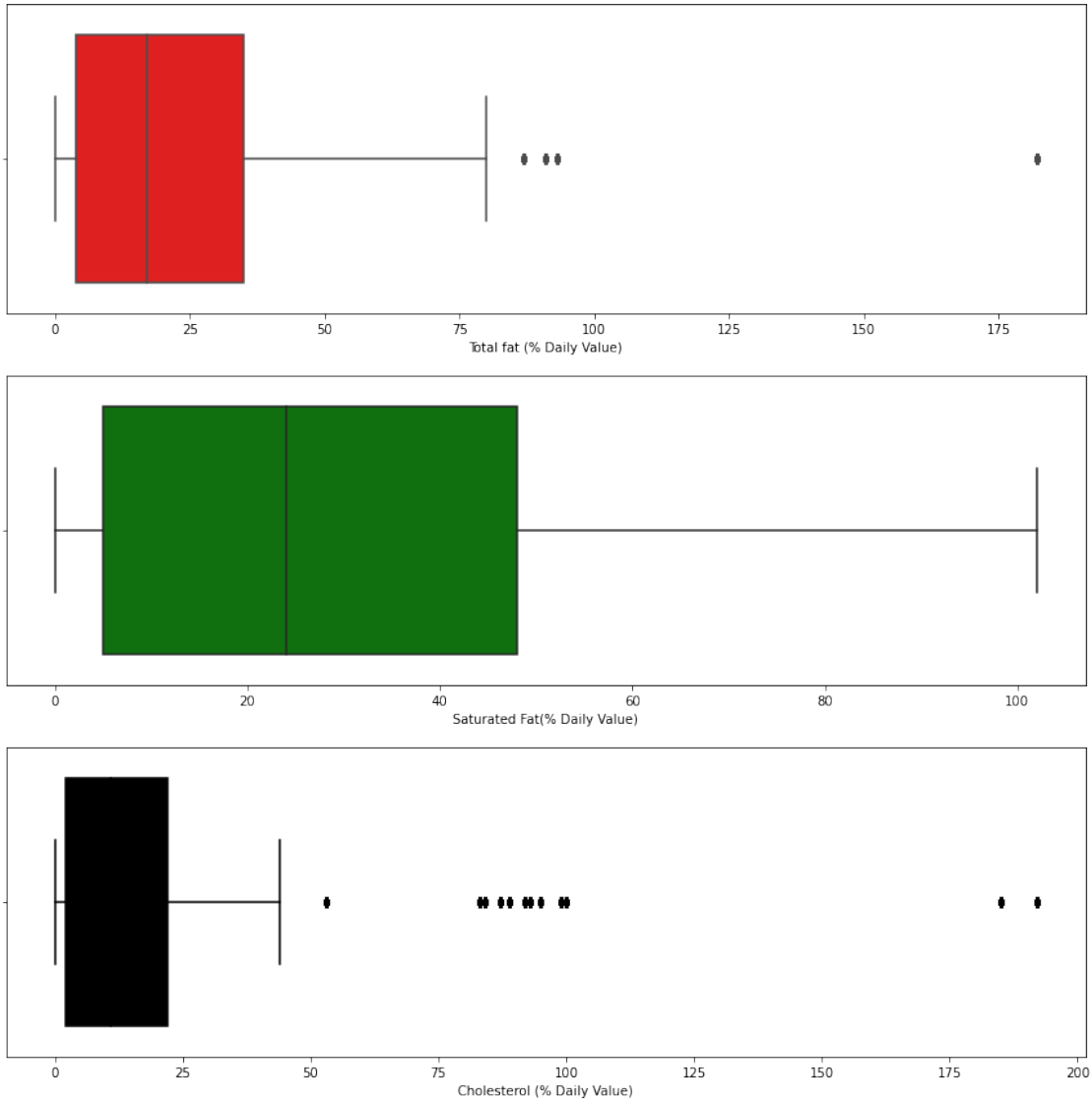
[36]:
```
#Checking for the outliers - fifth set
plt.figure(figsize= (15,15))
plt.subplot(3,1,1)
sns.boxplot(x= data['Calcium (% Dailly Value)'], color='orange')

plt.subplot(3,1,2)
sns.boxplot(x= data['Iron (% Daily Value)'], color='skyblue')

plt.subplot(3,1,3)
sns.boxplot(x= data['Carbohydrates'], color='green')


plt.show()
```



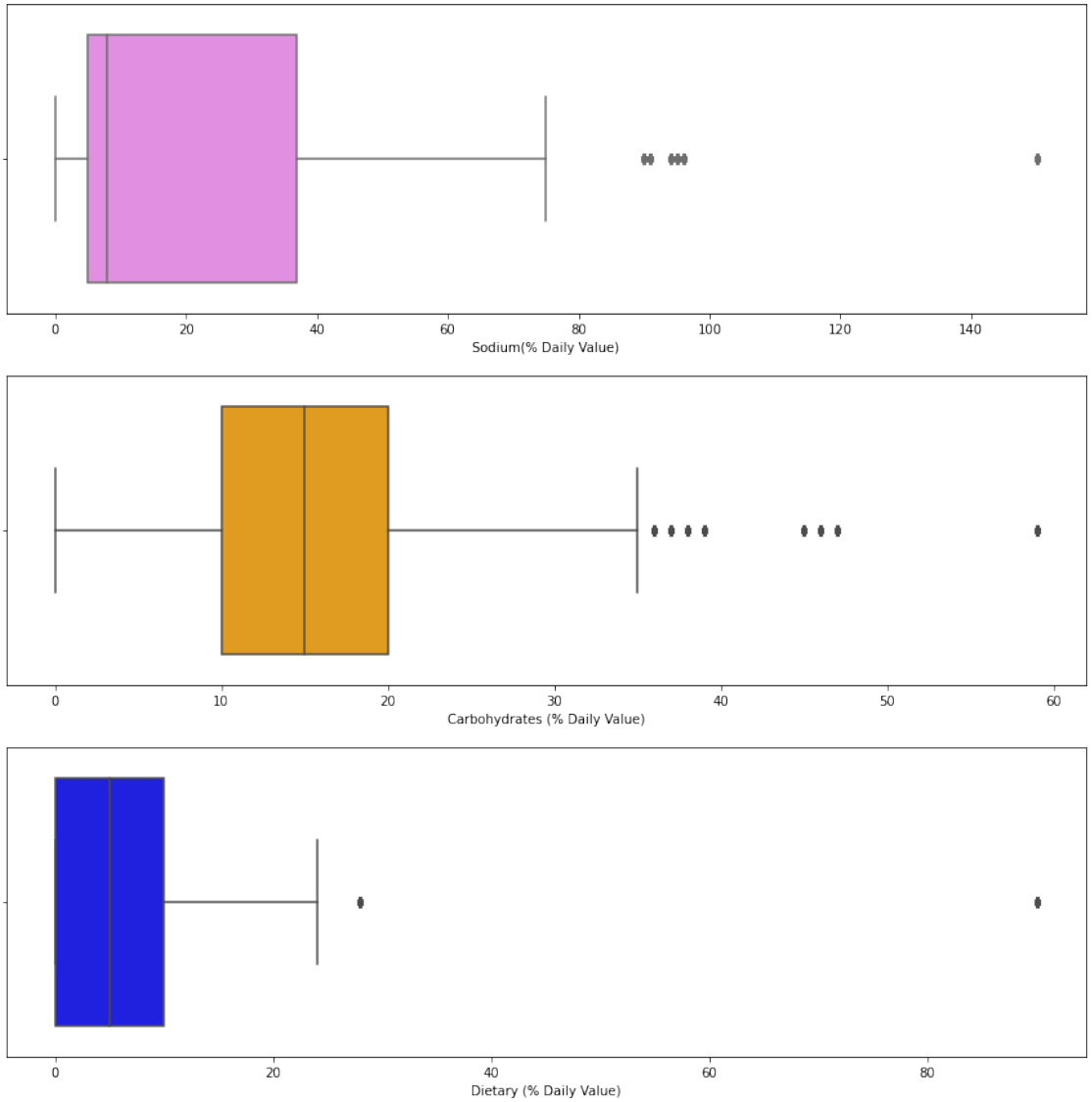




[37]:
```
#Checking for the outliers - sixth set
plt.figure(figsize= (15,15))
plt.subplot(3,1,1)
sns.boxplot(x= data['Total fat (% Daily Value)'], color='red')

plt.subplot(3,1,2)
sns.boxplot(x= data['Saturated Fat(% Daily Value)'], color='green')

plt.subplot(3,1,3)
sns.boxplot(x= data['Cholesterol (% Daily Value)'], color='black')


plt.show()
```



[38]:
```
#Checking for the outliers - seventh set
plt.figure(figsize= (15,15))
plt.subplot(3,1,1)
sns.boxplot(x= data['Sodium(% Daily Value)'], color='violet')

plt.subplot(3,1,2)
sns.boxplot(x= data['Carbohydrates (% Daily Value)'], color='orange')

plt.subplot(3,1,3)
sns.boxplot(x= data['Dietary (% Daily Value)'], color='blue')


plt.show()
```



Observations Based on the visualization above, the following variables have outliers:

Calories Outlier Calories From Fat Outlier Total Fat Outlier Total Fat (% Daily Value) Outlier Trans Fat Outlier Cholesterol Outlier Cholesterol (% Daily Value) Outlier Sodium Outlier Sodium (% daily value) value). Outlier Carbohydrate Outlier Carbohydrate (% 1-day value) Outlier Dietary fiber (% 1-day value) Outlier Sugar Outlier Outlier Protein Outlier Vitamin A (% 1-day value) Outlier Vitamin C (% 1-day value) Outlier Calcium (% 1-day value) Outlier Iron (% 1-day value)) Outlier

Saturated fat No outliers Dietary fiber No outliers Saturated fat (% daily value) No outliers

[39]:
```
print(np.where(data['Protein']>99000))
```

```
(array([], dtype=int64),)
```

[40]:
```
#Finding IQR values
#The interquartile range, or IQR, contains the second and third quartiles, or␣
 ↪the middle half of the dataset.
#Sort the data.
#Calculate Q1 and Q3.
#IQR = Q3 - Q1.
Q1 = np.percentile(data['Protein'], 25,
interpolation = 'midpoint')
Q3 = np.percentile(data['Protein'], 75,
interpolation = 'midpoint')
IQR = Q3 - Q1
#Printing Q1 value
print(Q1)
#Printing Q3 value
print(Q3)
#printing IQR value
print(IQR)
```

```
4.0
19.0
15.0
```

[41]:
```
# Above Upper bound
upper = data['Protein'] >= (Q3+1.5*IQR)
print("Upper bound:",upper)
print(np.where(upper))
# Below Lower bound
lower = data['Protein'] <= (Q1-1.5*IQR)
print("Lower bound:", lower)
print(np.where(lower))
#Old shape
# Upper bound
upper = np.where(data['Protein'] >= (Q3+1.5*IQR))
# Lower bound
lower = np.where(data['Protein'] <= (Q1-1.5*IQR))
```

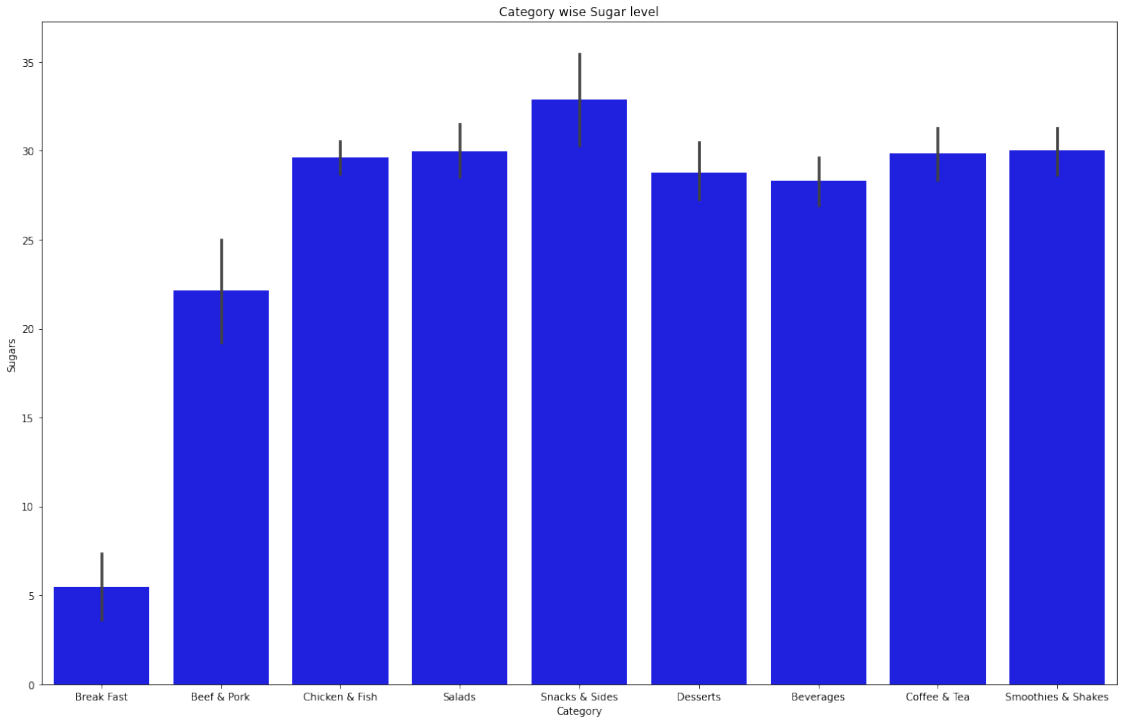
```
Upper bound: 0        False
1        False
2        False
3        False
4        False
         …
12720    False
12721    False
12722    False
12723    False
12724    False
Name: Protein, Length: 12725, dtype: bool
```

```
(array([   47,    81,    82,   260,   308,   342,   343,   521,   569,
          603,   604,   782,   830,   864,   865,  1043,  1091,  1125,
         1126,  1304,  1352,  1386,  1387,  1565,  1613,  1647,  1648,
         1826,  1874,  1908,  1909,  2087,  2135,  2169,  2170,  2348,
         2396,  2430,  2431,  2609,  2657,  2691,  2692,  2870,  2918,
         2952,  2953,  3131,  3179,  3213,  3214,  3392,  3440,  3474,
         3475,  3653,  3701,  3735,  3736,  3914,  3962,  3996,  3997,
         4175,  4223,  4257,  4258,  4436,  4484,  4518,  4519,  4697,
         4745,  4779,  4780,  4958,  5006,  5040,  5041,  5219,  5267,
         5301,  5302,  5480,  5528,  5562,  5563,  5741,  5789,  5823,
         5824,  6002,  6050,  6084,  6085,  6263,  6311,  6345,  6346,
         6524,  6572,  6606,  6607,  6785,  6833,  6867,  6868,  7046,
         7094,  7128,  7129,  7307,  7355,  7389,  7390,  7568,  7616,
         7650,  7651,  7829,  7877,  7911,  7912,  8090,  8138,  8172,
         8173,  8351,  8399,  8433,  8434,  8612,  8660,  8694,  8695,
         8873,  8921,  8955,  8956,  9134,  9182,  9216,  9217,  9395,
         9443,  9477,  9478,  9656,  9704,  9738,  9739,  9917,  9965,
         9999, 10000, 10178, 10226, 10260, 10261, 10439, 10487, 10521,
        10522, 10700, 10748, 10782, 10783, 10961, 11009, 11043, 11044,
        11222, 11270, 11304, 11305, 11483, 11531, 11565, 11566, 11744,
        11792, 11826, 11827, 12005, 12053, 12087, 12088, 12266, 12314,
        12348, 12349, 12527, 12575, 12609, 12610], dtype=int64),)
Lower bound: 0        False
1        False
2        False
3        False
4        False
         …
12720    False
12721    False
12722    False
12723    False
12724    False
Name: Protein, Length: 12725, dtype: bool
(array([], dtype=int64),)
```

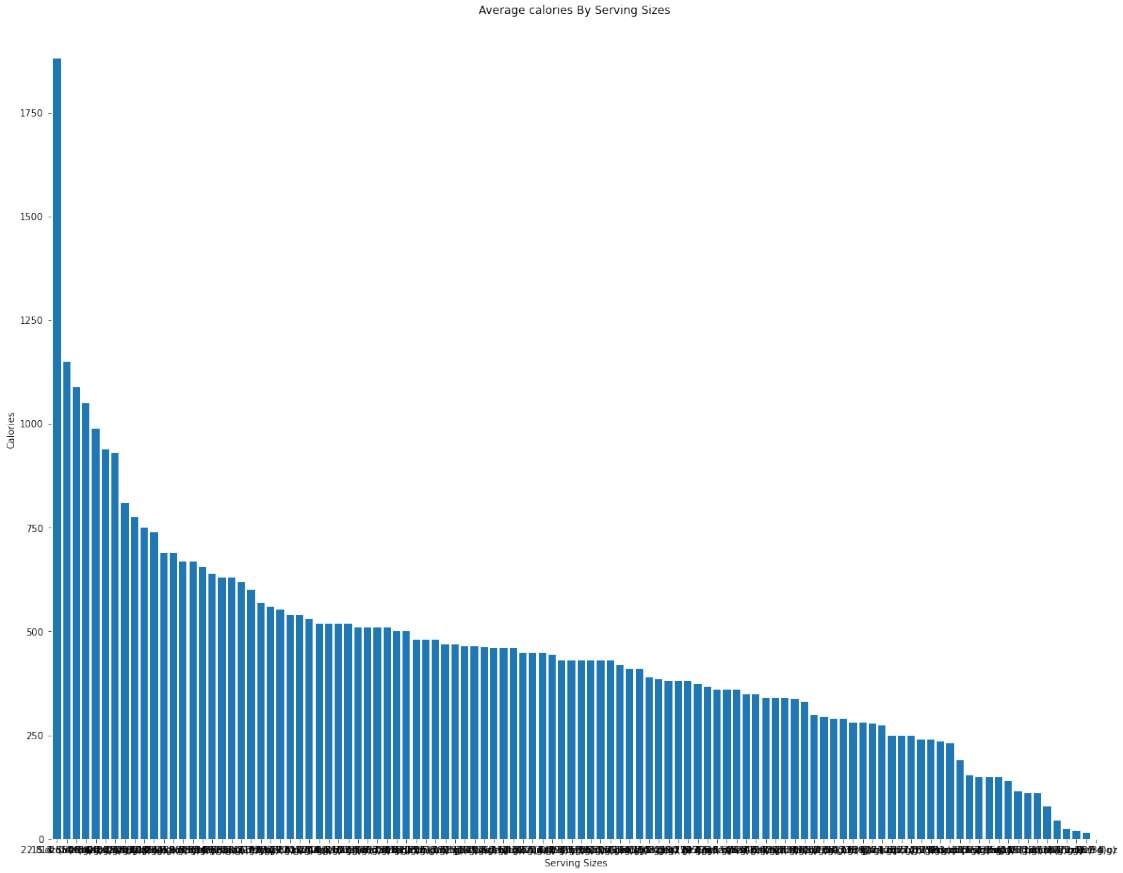
8 Bivariate Analysis

[42]:
```
#barplot (categorical-numerical)
#Bar Plot It shows the relationship between a numeric and a categoric variable
plt.figure(figsize=(19,12))
plt.title("Category wise Sugar level")
sns.barplot(data=data, x='Category',y='Sugars',color='blue')
```

[42]:
```
<AxesSubplot:title={'center':'Category wise Sugar level'}, xlabel='Category',
ylabel='Sugars'>
```

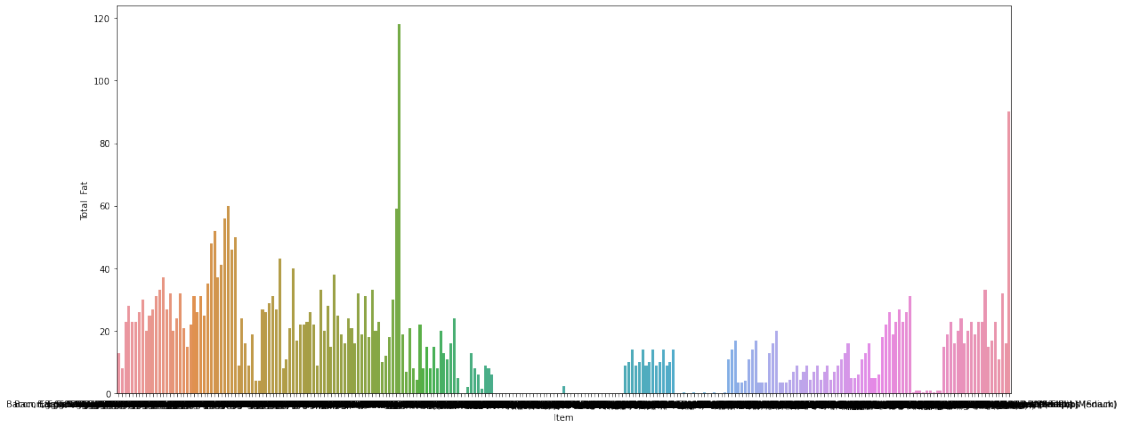


[43]:
```
#Average calories By Serving Sizes
fig, ax = plt.subplots(figsize=(20, 16))
data.groupby(['Serving Sizes'])['Calories'].mean().sort_values(ascending=False).
 ↪plot(kind = 'bar', width= 0.8)
ax.set(title = "Average calories By Serving Sizes ",
       xlabel = "Serving Sizes",
       ylabel = "Calories")
plt.setp(ax.get_xticklabels(), rotation = 0)
plt.box(False)
plt.show()
```



[44]:
```
#barplot (numerical-numerical)
plt.figure(figsize=(18,8))
sns.barplot(data=data,x='Item',y='Total  Fat')
```

[44]: <AxesSubplot:xlabel='Item', ylabel='Total Fat'>



9 Which variables have the highest correlation? Plotting them and finding out the value?

[45]:
```
#data.corr() is used to find the pairwise correlation of all columns in the␣
 ↪Pandas Dataframe in Python
data.corr()
```

[45]:

	lat	lon	alt	is_broken	is_active	Calories	Calories from Fat
lat	1.000000	0.093435	0.048351	-0.022762	-0.023976	0.015536	0.018928
lon	0.093435	1.000000	0.027292	-0.021100	0.050238	-0.003319	-0.001158
alt	0.048351	0.027292	1.000000	0.009427	-0.006096	-0.027054	-0.022861
is_broken	-0.022762	-0.021100	0.009427	1.000000	0.051964	0.012236	0.012350
is_active	-0.023976	0.050238	-0.006096	0.051964	1.000000	-0.005347	-0.005481
Calories	0.015536	-0.003319	-0.027054	0.012236	-0.005347	1.000000	0.904251
Calories from Fat	0.018928	-0.001158	-0.022861	0.012350	-0.005481	0.904251	1.000000
Total Fat	0.017377	0.000702	-0.022453	0.009958	-0.001612	0.829252	0.937349
Total fat (% Daily Value)	0.018764	-0.000846	-0.022866	0.011802	-0.004625	0.897622	0.997122
Saturated Fat	0.017210	-0.000808	-0.015567	0.009976	-0.005846	0.814828	0.827777
Saturated Fat(% Daily Value)	0.017756	-0.001759	-0.015899	0.010520	-0.006767	0.837459	0.845512
Trans Fat	-0.000832	0.005274	-0.003880	-0.004269			
Cholesterol	0.014973	-0.005617	-0.017008	0.013731			
Cholesterol (% Daily Value)	0.014767	-0.005400	-0.017024	0.013331			
Sodium	0.019422	-0.002369	-0.029982	0.013529			
Sodium(% Daily Value)	0.018719	-0.001020	-0.030057	0.012172			
Carbohydrates	0.005334	0.002240	-0.016983	0.007067			
Carbohydrates (% Daily Value)	0.004802	0.003466	-0.016908	0.005768			
Dietary Fiber	0.005119	0.007351	-0.016542	0.002323			
Dietary (% Daily Value)	0.009160	0.006329	-0.025283	0.009794			
Sugars	-0.007202	-0.001414	0.000601	-0.000720			
Protein	0.014349	-0.013741	-0.037457	0.011402			
Vitamin A (% Daily Value)	0.003054	0.000943	-0.011062	0.011978			
Vitamin C (% Daily Value)	-0.000170	-0.002116	0.002532	0.001727			
Calcium (% Dailly Value)	-0.005577	-0.018911	-0.026397	0.002228			
Iron (% Daily Value)	0.019633	-0.006326	-0.026931	0.013729			

```
Trans Fat                        0.008529 -0.045754          -0.003297
Cholesterol                     -0.003879  0.596891           0.682409
Cholesterol (% Daily Value)     -0.003248  0.588024           0.678138
Sodium                          -0.005239  0.715409           0.847922
Sodium(% Daily Value)           -0.003115  0.683676           0.825282
Carbohydrates                    0.000032  0.770084           0.457826
Carbohydrates (% Daily Value)    0.001976  0.724422           0.433543
Dietary Fiber                    0.004956  0.187993           0.252400
Dietary (% Daily Value)          0.003294  0.363986           0.423591
Sugars                           0.003825  0.243861          -0.121377
Protein                         -0.006628  0.760078           0.789850
Vitamin A (% Daily Value)       -0.008917  0.098206           0.052132
Vitamin C (% Daily Value)        0.003535 -0.070587          -0.088040
Calcium (% Dailly Value)        -0.005558  0.417961           0.156445
Iron (% Daily Value)            -0.007783  0.640864           0.735178

                                Total  Fat  Total fat (% Daily Value)  \
lat                               0.017377                   0.018764
lon                               0.000702                  -0.000846
alt                              -0.022453                  -0.022866
is_broken                         0.009958                   0.011802
is_active                        -0.001612                  -0.004625
Calories                          0.829252                   0.897622
Calories from Fat                 0.937349                   0.997122
Total  Fat                        1.000000                   0.960067
Total fat (% Daily Value)         0.960067                   1.000000
Saturated  Fat                    0.846331                   0.840620
Saturated Fat(% Daily Value)      0.824645                   0.850372
Trans Fat                         0.343370                   0.068595
Cholesterol                       0.636837                   0.679098
Cholesterol (% Daily Value)       0.661325                   0.680742
Sodium                            0.788602                   0.844557
Sodium(% Daily Value)             0.849969                   0.839108
Carbohydrates                     0.458991                   0.461774
Carbohydrates (% Daily Value)     0.506648                   0.452196
Dietary Fiber                     0.542984                   0.315012
Dietary (% Daily Value)           0.621967                   0.468732
Sugars                           -0.077794                  -0.113985
Protein                           0.809256                   0.802082
Vitamin A (% Daily Value)         0.081926                   0.056639
Vitamin C (% Daily Value)        -0.075562                  -0.087984
Calcium (% Dailly Value)          0.171948                   0.161966
Iron (% Daily Value)              0.705846                   0.736499

                                Saturated  Fat  …  Carbohydrates  \
lat                                  0.017210  …       0.005334
lon                                 -0.000808  …       0.002240
```

alt	-0.015567	…	-0.016983
is_broken	0.009976	…	0.007067
is_active	-0.005846	…	0.000032
Calories	0.814828	…	0.770084
Calories from Fat	0.827777	…	0.457826
Total Fat	0.846331	…	0.458991
Total fat (% Daily Value)	0.840620	…	0.461774
Saturated Fat	1.000000	…	0.591765
Saturated Fat(% Daily Value)	0.993065	…	0.591746
Trans Fat	0.219216	…	0.105654
Cholesterol	0.617240	…	0.268601
Cholesterol (% Daily Value)	0.629869	…	0.272410
Sodium	0.569185	…	0.199351
Sodium(% Daily Value)	0.601452	…	0.214025
Carbohydrates	0.591765	…	1.000000
Carbohydrates (% Daily Value)	0.606411	…	0.978324
Dietary Fiber	0.320642	…	0.171269
Dietary (% Daily Value)	0.375839	…	0.222184
Sugars	0.208105	…	0.761465
Protein	0.615834	…	0.359072
Vitamin A (% Daily Value)	0.079855	…	0.088630
Vitamin C (% Daily Value)	-0.171383	…	-0.031934
Calcium (% Dailly Value)	0.404832	…	0.589264
Iron (% Daily Value)	0.576024	…	0.213534

	Carbohydrates (% Daily Value)	Dietary Fiber	\
lat	0.004802	0.005119	
lon	0.003466	0.007351	
alt	-0.016908	-0.016542	
is_broken	0.005768	0.002323	
is_active	0.001976	0.004956	
Calories	0.724422	0.187993	
Calories from Fat	0.433543	0.252400	
Total Fat	0.506648	0.542984	
Total fat (% Daily Value)	0.452196	0.315012	
Saturated Fat	0.606411	0.320642	
Saturated Fat(% Daily Value)	0.583440	0.228709	
Trans Fat	0.307050	0.871210	
Cholesterol	0.254464	0.187409	
Cholesterol (% Daily Value)	0.275037	0.258460	
Sodium	0.182314	0.296899	
Sodium(% Daily Value)	0.245536	0.498796	
Carbohydrates	0.978324	0.171269	
Carbohydrates (% Daily Value)	1.000000	0.344378	
Dietary Fiber	0.344378	1.000000	
Dietary (% Daily Value)	0.343467	0.925022	
Sugars	0.756648	-0.049434	

Protein	0.380895	0.456365
Vitamin A (% Daily Value)	0.105507	0.253047
Vitamin C (% Daily Value)	-0.025531	0.094382
Calcium (% Dailly Value)	0.581542	0.068308
Iron (% Daily Value)	0.211042	0.382708

	Dietary (% Daily Value)	Sugars	Protein	\
lat	0.009160	-0.007202	0.014349	
lon	0.006329	-0.001414	-0.013741	
alt	-0.025283	0.000601	-0.037457	
is_broken	0.009794	-0.000720	0.011402	
is_active	0.003294	0.003825	-0.006628	
Calories	0.363986	0.243861	0.760078	
Calories from Fat	0.423591	-0.121377	0.789850	
Total Fat	0.621967	-0.077794	0.809256	
Total fat (% Daily Value)	0.468732	-0.113985	0.802082	
Saturated Fat	0.375839	0.208105	0.615834	
Saturated Fat(% Daily Value)	0.312984	0.196740	0.608441	
Trans Fat	0.632833	0.134453	0.197884	
Cholesterol	0.322974	-0.139194	0.548194	
Cholesterol (% Daily Value)	0.374124	-0.131062	0.560884	
Sodium	0.503694	-0.429028	0.848245	
Sodium(% Daily Value)	0.644771	-0.394490	0.873909	
Carbohydrates	0.222184	0.761465	0.359072	
Carbohydrates (% Daily Value)	0.343467	0.756648	0.380895	
Dietary Fiber	0.925022	-0.049434	0.456365	
Dietary (% Daily Value)	1.000000	-0.153208	0.608581	
Sugars	-0.153208	1.000000	-0.159161	
Protein	0.608581	-0.159161	1.000000	
Vitamin A (% Daily Value)	0.342420	0.057374	0.226444	
Vitamin C (% Daily Value)	0.134013	-0.065568	-0.040515	
Calcium (% Dailly Value)	0.080367	0.600722	0.332603	
Iron (% Daily Value)	0.586751	-0.360482	0.785155	

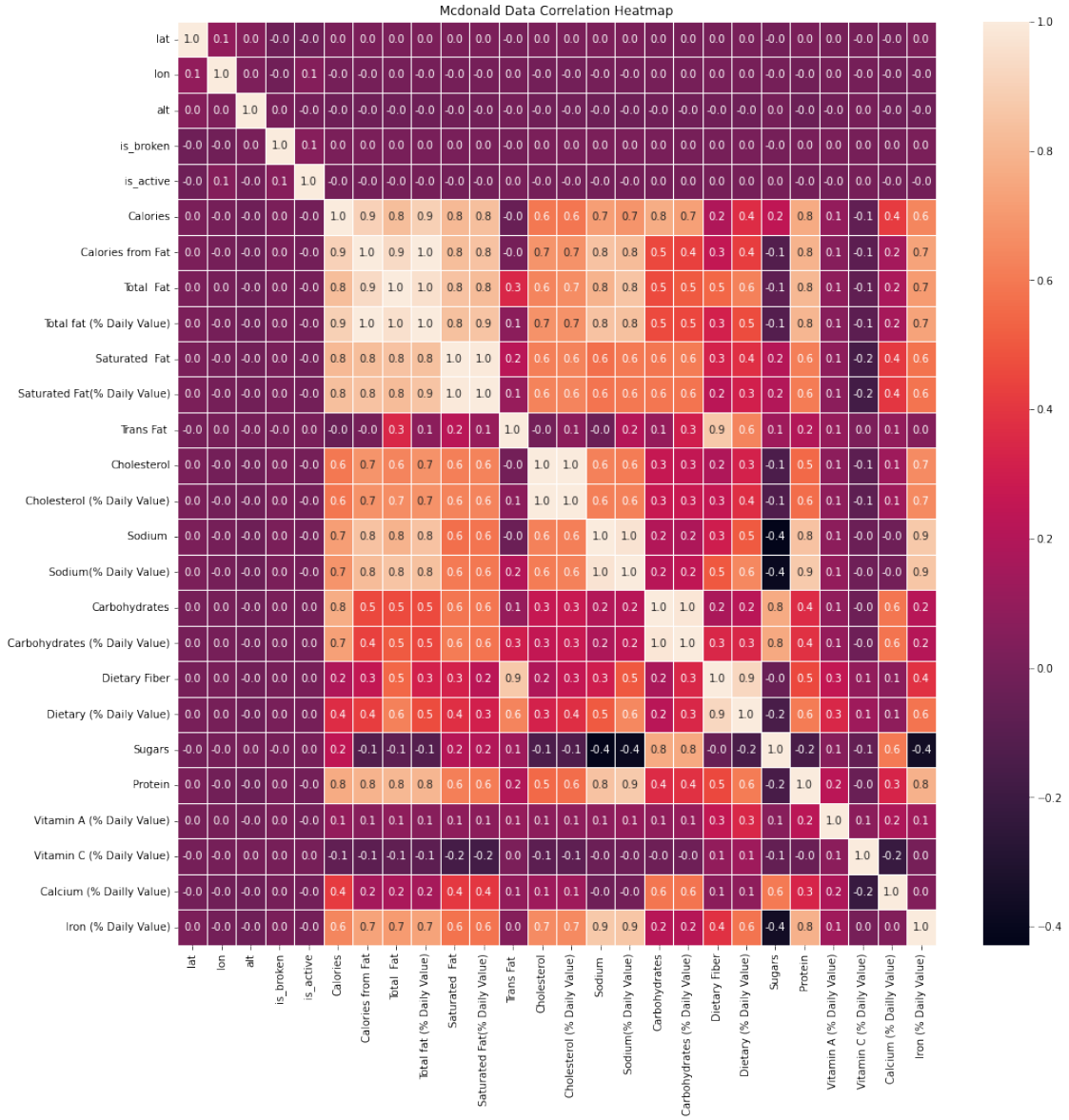
	Vitamin A (% Daily Value)	\
lat	0.003054	
lon	0.000943	
alt	-0.011062	
is_broken	0.011978	
is_active	-0.008917	
Calories	0.098206	
Calories from Fat	0.052132	
Total Fat	0.081926	
Total fat (% Daily Value)	0.056639	
Saturated Fat	0.079855	
Saturated Fat(% Daily Value)	0.070074	
Trans Fat	0.104886	

```
Cholesterol                                 0.076553
Cholesterol (% Daily Value)                 0.084463
Sodium                                      0.077703
Sodium(% Daily Value)                       0.100157
Carbohydrates                               0.088630
Carbohydrates (% Daily Value)               0.105507
Dietary Fiber                               0.253047
Dietary (% Daily Value)                     0.342420
Sugars                                      0.057374
Protein                                     0.226444
Vitamin A (% Daily Value)                   1.000000
Vitamin C (% Daily Value)                   0.071741
Calcium (% Dailly Value)                    0.183240
Iron (% Daily Value)                        0.139543

                            Vitamin C (% Daily Value)  \
lat                                         -0.000170
lon                                         -0.002116
alt                                          0.002532
is_broken                                    0.001727
is_active                                    0.003535
Calories                                    -0.070587
Calories from Fat                           -0.088040
Total  Fat                                  -0.075562
Total fat (% Daily Value)                   -0.087984
Saturated  Fat                              -0.171383
Saturated Fat(% Daily Value)                -0.175308
Trans Fat                                    0.022822
Cholesterol                                 -0.083728
Cholesterol (% Daily Value)                 -0.081607
Sodium                                      -0.032714
Sodium(% Daily Value)                       -0.025448
Carbohydrates                               -0.031934
Carbohydrates (% Daily Value)               -0.025531
Dietary Fiber                                0.094382
Dietary (% Daily Value)                      0.134013
Sugars                                      -0.065568
Protein                                     -0.040515
Vitamin A (% Daily Value)                    0.071741
Vitamin C (% Daily Value)                    1.000000
Calcium (% Dailly Value)                    -0.212910
Iron (% Daily Value)                         0.001425

                            Calcium (% Dailly Value)  Iron (% Daily Value)
lat                                        -0.005577              0.019633
lon                                        -0.018911             -0.006326
alt                                        -0.026397             -0.026931
```

is_broken	0.002228	0.013729
is_active	-0.005558	-0.007783
Calories	0.417961	0.640864
Calories from Fat	0.156445	0.735178
Total Fat	0.171948	0.705846
Total fat (% Daily Value)	0.161966	0.736499
Saturated Fat	0.404832	0.576024
Saturated Fat(% Daily Value)	0.400991	0.583408
Trans Fat	0.091970	0.048968
Cholesterol	0.129332	0.654229
Cholesterol (% Daily Value)	0.134617	0.653690
Sodium	-0.026838	0.869149
Sodium(% Daily Value)	-0.009860	0.859140
Carbohydrates	0.589264	0.213534
Carbohydrates (% Daily Value)	0.581542	0.211042
Dietary Fiber	0.068308	0.382708
Dietary (% Daily Value)	0.080367	0.586751
Sugars	0.600722	-0.360482
Protein	0.332603	0.785155
Vitamin A (% Daily Value)	0.183240	0.139543
Vitamin C (% Daily Value)	-0.212910	0.001425
Calcium (% Dailly Value)	1.000000	0.035591
Iron (% Daily Value)	0.035591	1.000000

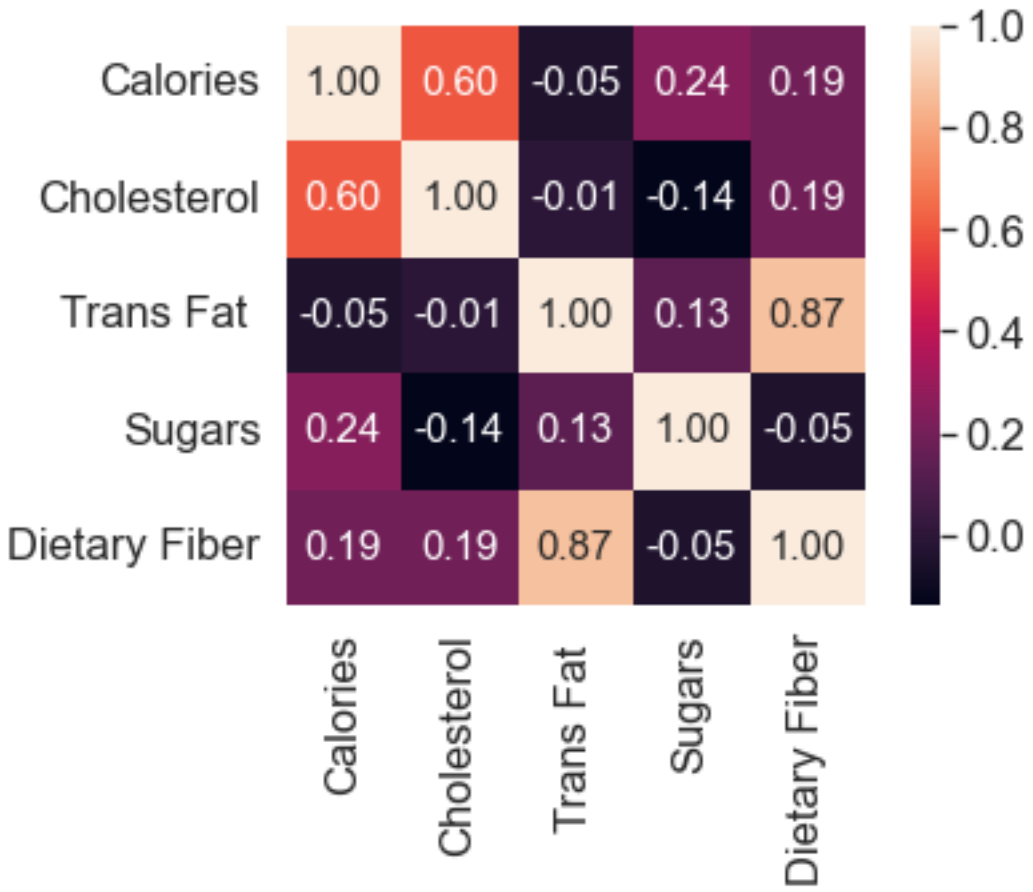
[26 rows x 26 columns]

[46]:
```
#Heatmaps are a method of representing data graphically where values are␣
 ↪depicted by color,
#making it easy to visualize complex data
f,ax = plt.subplots(figsize=(16, 16))
sns.heatmap(data.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
plt.title("Mcdonald Data Correlation Heatmap")
plt.show()
```



It can be seen that there are various parameters with strong correlations between them. Let's plot these into each of these powerful relationships one by one.

[47]:
```
cols = ['Calories','Cholesterol','Trans Fat ','Sugars','Dietary Fiber']
cm = np.corrcoef(data[cols].values.T)
sns.set(font_scale = 1.5)
#annot = Truewrite the data value in each cell
hm = sns.heatmap(cm,cbar = True, annot = True,square = True, fmt = '.2f',␣
 ↪annot_kws = {'size':15}, yticklabels = cols, xticklabels = cols)
```

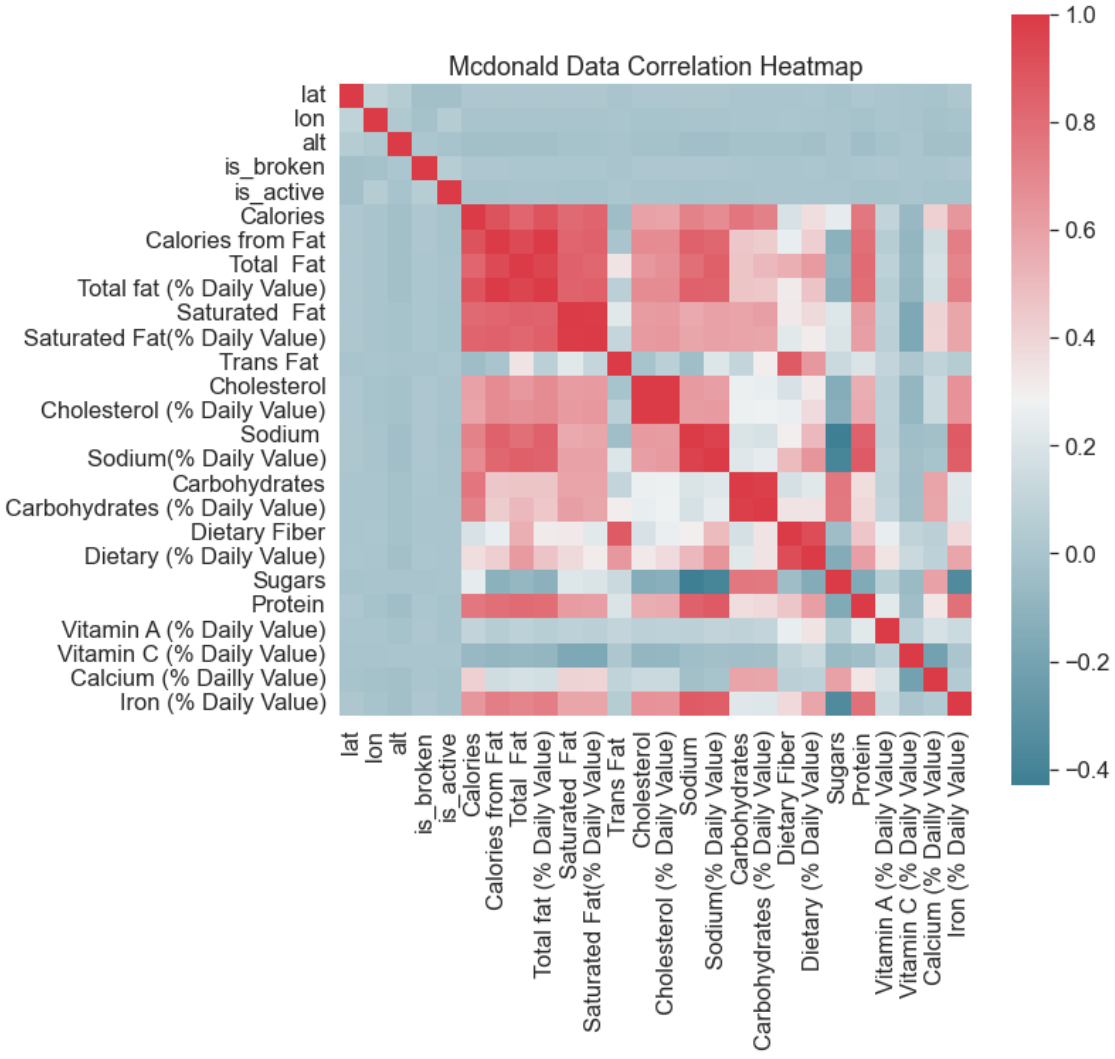


Well, The correlation matrix confirmed the idea that it is best to avoid cholesterol and trans fats.They are not only harmful to the body, but also increase the calorie content of food. If you want to know the average calorie content of a product, you can look at the distribution. Or take a look at her one of the central trend's leading figures.

[48]:
```
corr = data.corr()
plt.figure(figsize=(10,10))
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.
 ↪diverging_palette(220, 10, as_cmap=True),
            square=True)
plt.title("Mcdonald Data Correlation Heatmap")

plt.show()
```

```
C:\Users\Dell\AppData\Local\Temp/ipykernel_27924/3030253878.py:3:
DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To
silence this warning, use `bool` by itself. Doing this will not modify any
behavior and is safe. If you specifically wanted the numpy scalar type, use
`np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
```

```
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True),
```



Well, the correlation matrix confirms our findings CONCLUSION WHAT TO AVOID? Chicken McNuggets:- Packed with protein, but packed with calories (over 1750). A large breakfast of hotcakes (regular biscuits): - Contains excessive amounts of cholesterol. Double Quarter Pounder with Cheese:- If you are on a diet, avoid this item as it contains high amounts of trans fat.

10 Which item contributes maximum to the Sodium intake?

[49]:
```
#importing libraries
#import plotly.graph_objs as go : This has the functions for generating graph␣
 ↪objects.
```

```
import plotly.graph_objs as go
#The Plotly offline mode also enables you to save graphs locally.
import plotly.offline as py
#contains functions that can create entire figures at once, and is referred to␣
 ↪as Plotly Express
import plotly.express as px
#unction is used for Jupyter notebook, and will display the plots within the␣
 ↪notebook. plot() creates
#an HTML page that is saved locally to be opened in a web browser.
from plotly.offline import iplot
```

[50]:
```
#scatter plot
#A scatter plot (aka scatter chart, scatter graph) uses dots to represent␣
 ↪values for
#two different numeric variables. The position of each dot on the horizontal␣
 ↪and vertical axis indicates
#values for an individual data point.
#Scatter plots are used to observe relationships between variables.
fig = px.scatter(data, x="Sodium(% Daily Value)",y='Calories', color="Category",
                 size='Calories', hover_name="Item",trendline="ols",
                 template="plotly_dark",marginal_x="box",marginal_y="box")
fig.show()
```

The scatter plots for Sodium (% Daily Value) seem to follow a similar distribution of points whereby MacDonald food items contributing the greatest amount of sodium are scaled largest. As evinced by the largest red circular plot, the 40-piece Chicken McNuggets are the greatest contributor to Sodium intake.

The Big Breakfast range with Hotcakes follow up as a close second as a contributor to the sodium amount.

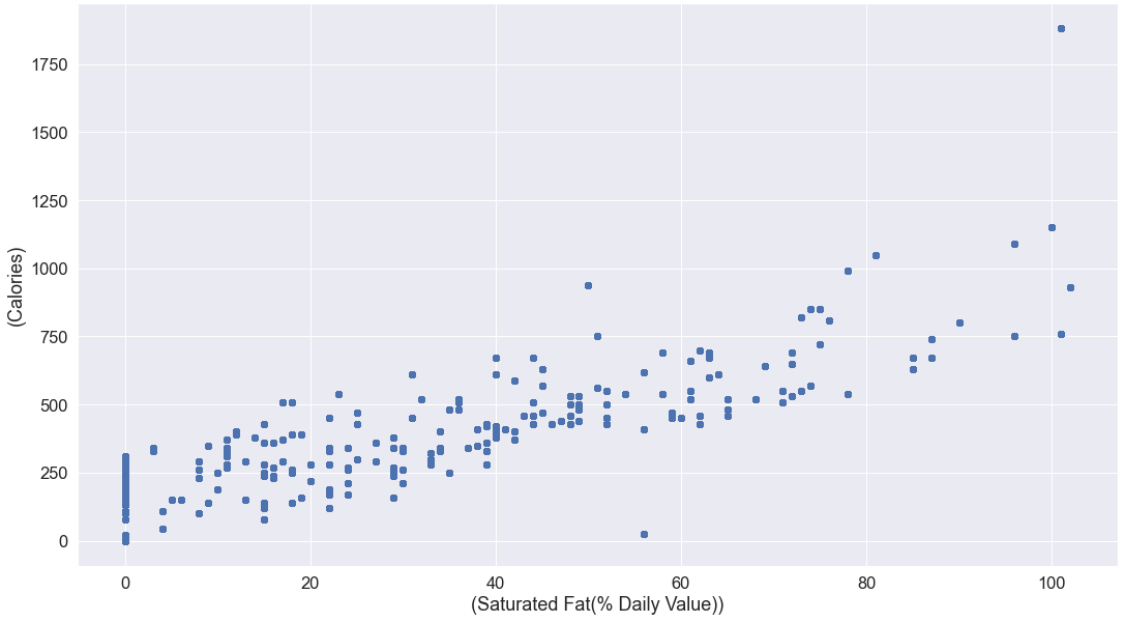
Greatest amount of Sodium : Chicken McNuggets (40 piece)

[51]:
```
#The purpose of the code below is to group all of the foods by category and to␣
 ↪plot the
#average calary count of items in the group.

Calories = data.drop('Item', axis = 1)
Calories = data.groupby(["Category"])["Calories"].mean()
Calories = Calories.sort_values(ascending=False)
print(Calories)


Category = ['Chicken & Fish', 'Smoothies & Shakes', 'Breakfast',
'Beef & Pork', 'Coffee & Tea', 'Salads', 'Snacks &␣
 ↪Sides','Desserts','Beverages']
```

```
Category
Break Fast            449.166667
```

```
Snacks & Sides         375.415945
Beverages              374.157428
Salads                 367.018141
Chicken & Fish         366.130952
Coffee & Tea           365.864078
Smoothies & Shakes     363.247624
Desserts               363.086516
Beef & Pork            331.666667
Name: Calories, dtype: float64
```

[52]:
```
# Scatter plot
fig, ax = plt.subplots(figsize = (18,10))
ax.scatter(data['Saturated Fat(% Daily Value)'], data['Calories'])

# x-axis label
ax.set_xlabel('(Saturated Fat(% Daily Value))')

# y-axis label
ax.set_ylabel('(Calories)')
plt.show()
```



11 plot of the Carbohydrates vs. Calories coloured by menu item type

[53]:
```
# The function creates a Scatter graph object (go) and uses the data frame .
 ↪isin() selection to extract the requested information
def make_scatter(data,category,x_cat,y_cat):
    return  go.Scatter(
       x = data[data['Category'].isin([category])][x_cat],
       y = data[data['Category'].isin([category])][y_cat],
       mode = "markers",
       name = category,
       text=  data.Item)
```

[54]:
```
# Define our categories to plot
x_cat = 'Calories'; y_cat = 'Carbohydrates'# Create a list of scatter plots to␣
 ↪view all at once
data = [make_scatter(data,cat,x_cat,y_cat) for cat in
   data.Category.unique().tolist()]# Define the plot layout (title, ticks etc.)
layout = dict(title = 'McDonalds Nutrition',
   xaxis= dict(title= 'Calories',ticklen=5,zeroline= False),
   yaxis= dict(title= 'Carbohydrates(g)',ticklen= 5,zeroline=False))# Finally␣
 ↪we will plot the data with the layout
fig = dict(data = data, layout = layout)
iplot(fig)
```

[55]:
```
import pandas as pd
#The glob module finds all the pathnames matching a specified pattern according␣
 ↪to the rules used by the Unix shell
from glob import glob
filenames = glob('mcdonalds.csv')
data =[pd.read_csv(f) for f in filenames]
data = pd.concat(data, ignore_index=True)
data = data[['Protein: 11'].str.contains("Sugars", na=False)]
```

```
C:\Users\Dell\AppData\Local\Temp/ipykernel_27924/359397765.py:5: DtypeWarning:

Columns (6) have mixed types.Specify dtype option on import or set
low_memory=False.
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_27924/359397765.py in <module>
      5 data =[pd.read_csv(f) for f in filenames]
      6 data = pd.concat(data, ignore_index=True)
----> 7 data = data[['Protein: 11'].str.contains("Sugars", na=False)]
```

```
AttributeError: 'list' object has no attribute 'str'
```

Probably better to have Breakfast at home.

[56]:
```
# max() function returns the item with the highest value
x = max(data['Cholesterol'])
data[(data.Cholesterol ==x)]
#the most dangerous for your heart breakfasts.It can be your last ...
 ↪Cholesterol is a cause of heart attacks
```

[56]:

	lat	lon	alt	is_broken	is_active	status	state	\
31	-73.986854	40.669625	0	False	True	working	NY	
32	-74.050780	40.743783	0	False	True	working	NJ	
292	-73.622795	40.862324	0	False	True	working	NY	
293	-74.361633	40.523605	0	False	True	working	NJ	
553	-72.661171	40.915020	0	False	True	working	NY	
…	…	…	…	…	…	…	…	
15953	8.283701	52.710689	0	True	True	broken	NaN	
16213	8.914177	49.871571	0	False	True	working	NaN	
16214	13.866699	52.556400	0	False	True	working	NaN	
16474	12.171020	50.497059	0	False	True	working	NaN	
16475	9.730143	50.552731	0	False	True	working	NaN	

	city	street	country	…	Carbohydrates	\
31	Brooklyn	289 9th St	USA	…	111	
32	Jersey City	260 Central Ave	USA	…	116	
292	Glen Cove	193 Glen St	USA	…	111	
293	Edison	1075 Route 1 S	USA	…	116	
553	Riverhead	30 Flanders Rd	USA	…	111	
…	…	…	…	…	…	
15953	Vechta	Friesenstr. 1	DE	…	116	
16213	Groß-Umstadt	Georg-August-Zinn-Str. 100a	DE	…	111	
16214	Strausberg	Herrenseeallee 15a	DE	…	116	
16474	Plauen	Äussere Reichenbacher Str. 64	DE	…	111	
16475	Künzell	Danziger Str. 12	DE	…	116	

	Carbohydrates (% Daily Value)	Dietary Fiber	Dietary (% Daily Value)	\
31	37	6	23	
32	39	7	28	
292	37	6	23	
293	39	7	28	
553	37	6	23	
…	…	…	…	
15953	39	7	28	
16213	37	6	23	
16214	39	7	28	
16474	37	6	23	

```
16475                          39             7                        28

       Sugars  Protein  Vitamin A (% Daily Value)  Vitamin C (% Daily Value)  \
31         17       36                         15                          2
32         17       36                         15                          2
292        17       36                         15                          2
293        17       36                         15                          2
553        17       36                         15                          2
…         …       …                          …                          …
15953      17       36                         15                          2
16213      17       36                         15                          2
16214      17       36                         15                          2
16474      17       36                         15                          2
16475      17       36                         15                          2

       Calcium (% Dailly Value)  Iron (% Daily Value)
31                           25                    40
32                           30                    40
292                          25                    40
293                          30                    40
553                          25                    40
…                           …                     …
15953                        30                    40
16213                        25                    40
16214                        30                    40
16474                        25                    40
16475                        30                    40

[128 rows x 35 columns]
```

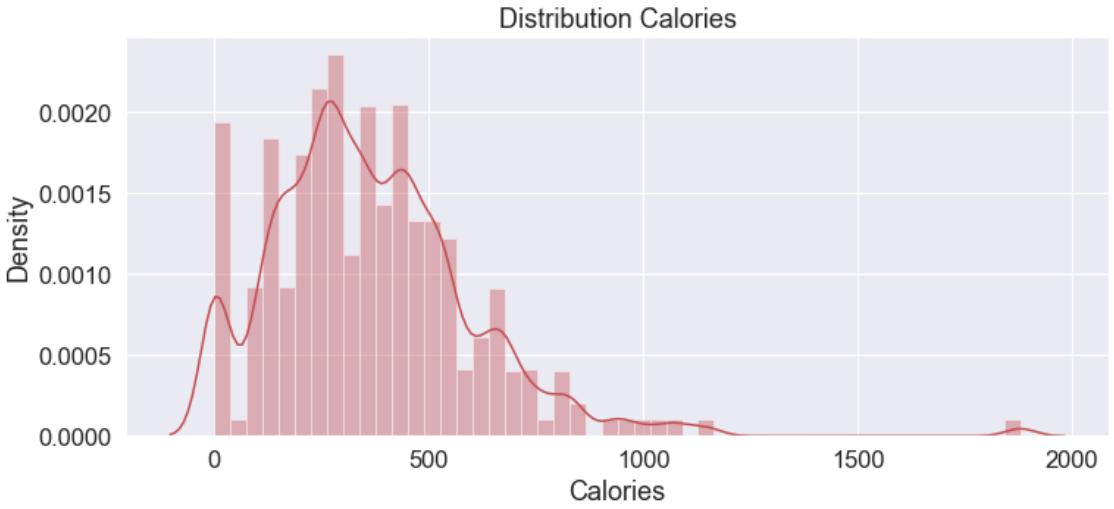
[57]:
```
data.at[82,'Item']
```

[57]:
```
'Chicken McNuggets (40 piece)'
```

[58]:
```
#dist plot
#distribution plot is suitable for comparing range and distribution for groups␣
 ↪of numerical data.
plt.figure(figsize=(12,5))
plt.title("Distribution Calories")
ax = sns.distplot(data["Calories"], color = 'r')

print(data.Calories.mean())
print(data.Calories.median())
```

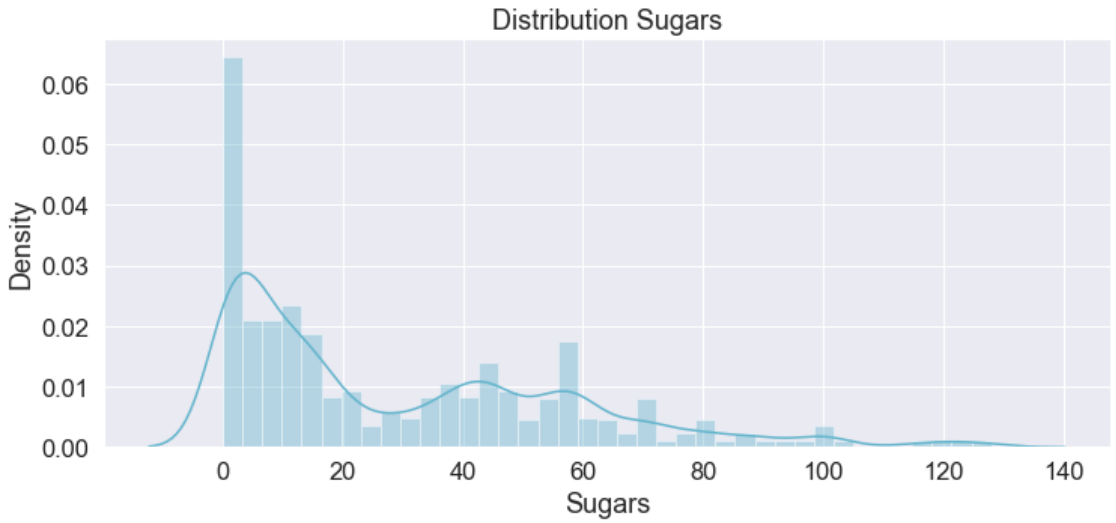
```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
```

```
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).


366.63097594625395
340.0
```



[59]:
```
plt.figure(figsize=(12,5))
plt.title("Distribution Sugars")
ax = sns.distplot(data["Sugars"], color = 'c')


print(data.Sugars.mean())
print(data.Sugars.median())
```

```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).


29.519764861136103
18.0
```

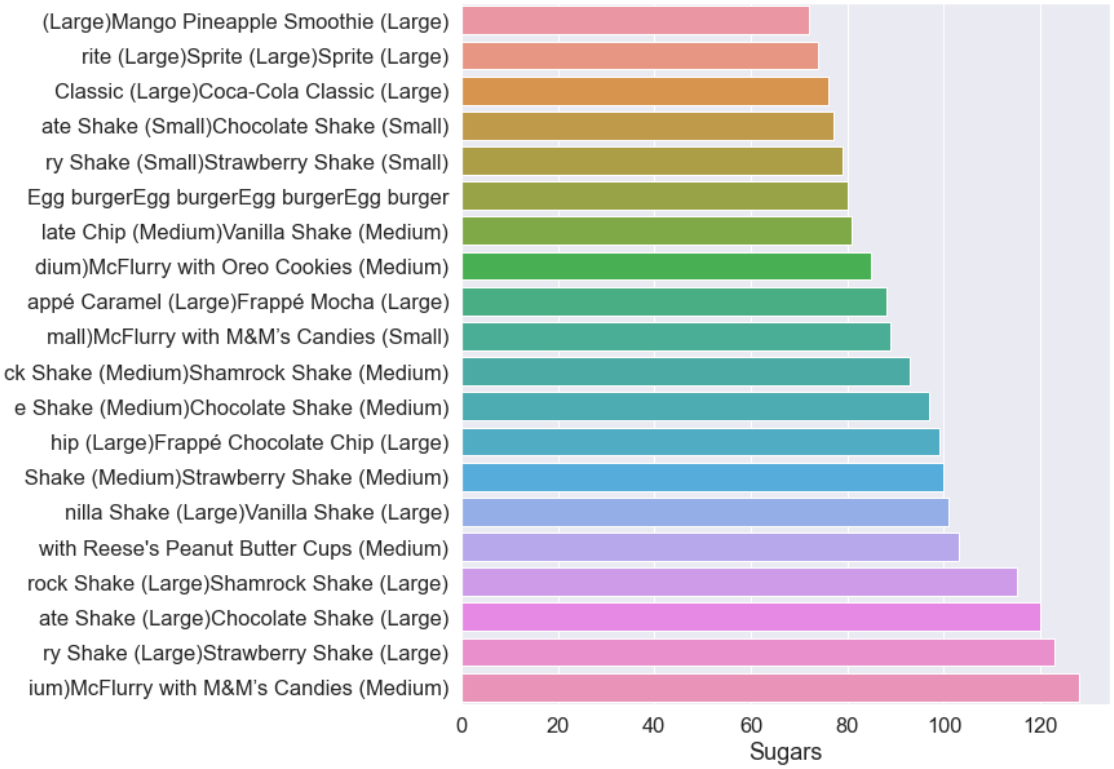


the products contain relatively little sugar. Most of the sugar contained in shakes,

[60]:
```
def plot(grouped):
    item = grouped["Item"].sum()
    item_list = item.sort_index()
    item_list = item_list[-20:]
    plt.figure(figsize=(9,10))
    graph = sns.barplot(item_list.index,item_list.values)
    labels = [aj.get_text()[-40:] for aj in graph.get_yticklabels()]
    graph.set_yticklabels(labels)
```

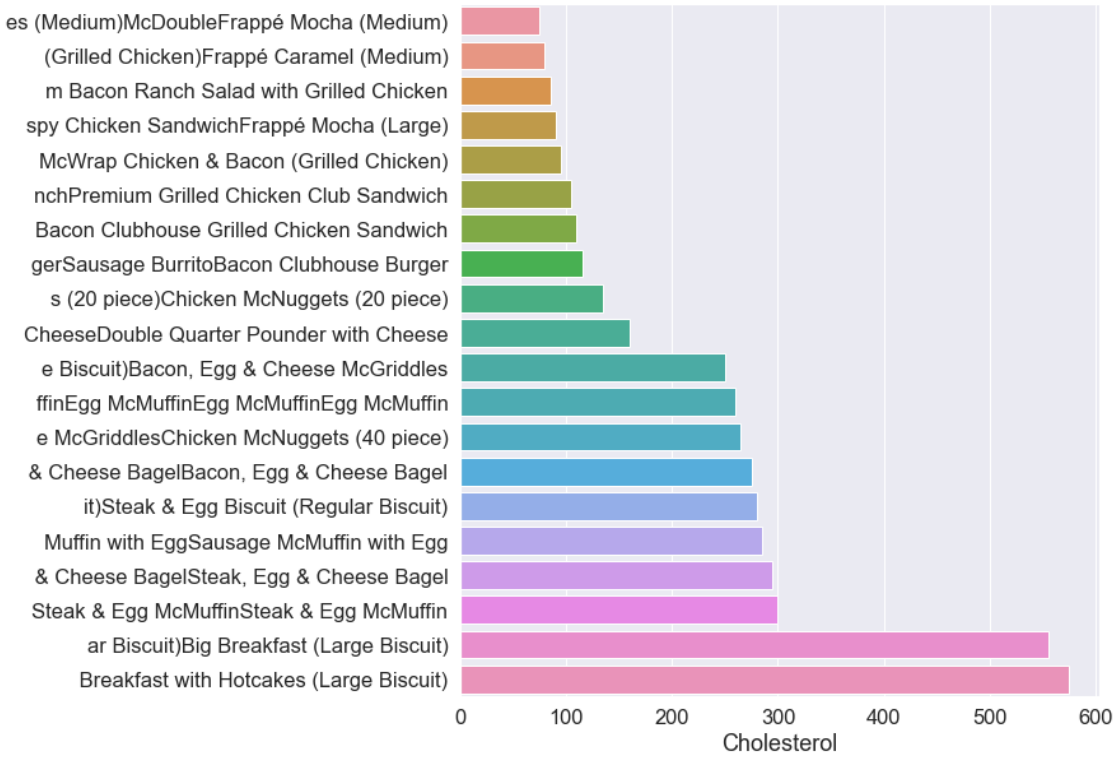
[61]:
```
sugar = data.groupby(data["Sugars"])
plot(sugar)
```

```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
```



[62]:
```
fats = data.groupby(data["Cholesterol"])
plot(fats)
```

```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
```



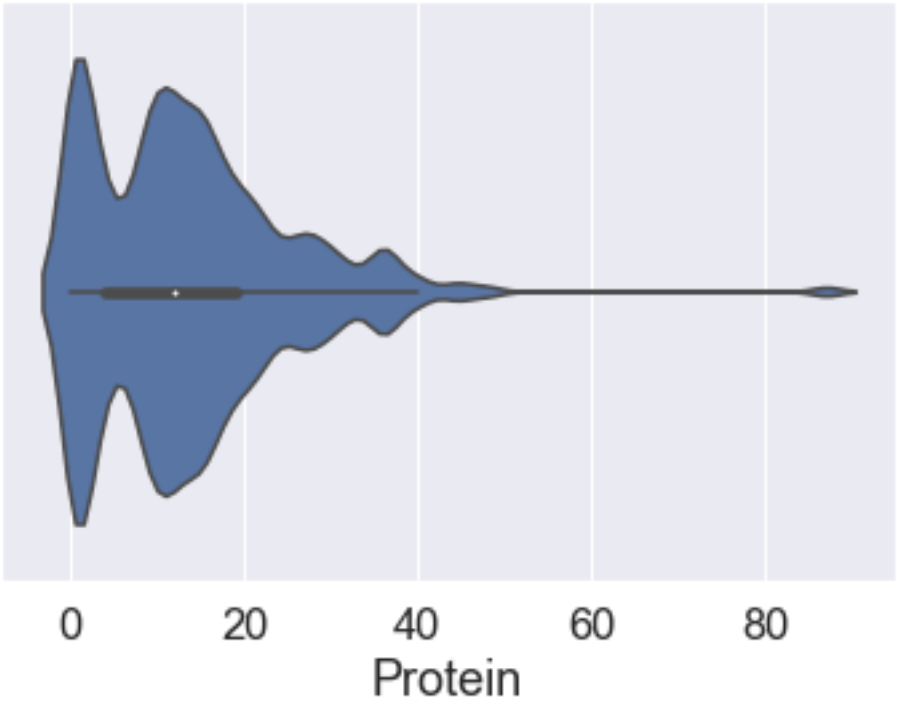
[63]:
```
#What menu item has the most sugar? Using idxmax and loc
data.loc[data.Sugars.idxmax() ].Item
```

[63]: 'McFlurry with M&M's Candies (Medium)'

[64]:
```
#A violin plot depicts distributions of numeric data for one or more groups␣
 ↪using density curves
sns.violinplot(data['Protein'])
```

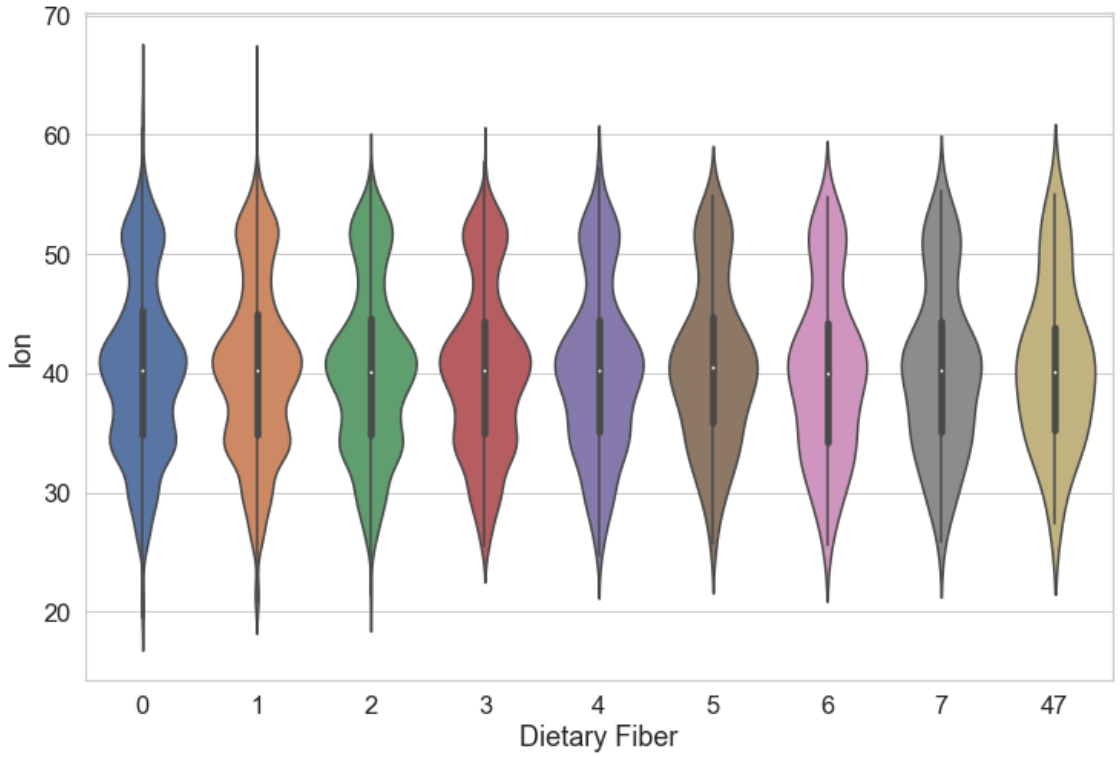
```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
```

[64]: <AxesSubplot:xlabel='Protein'>



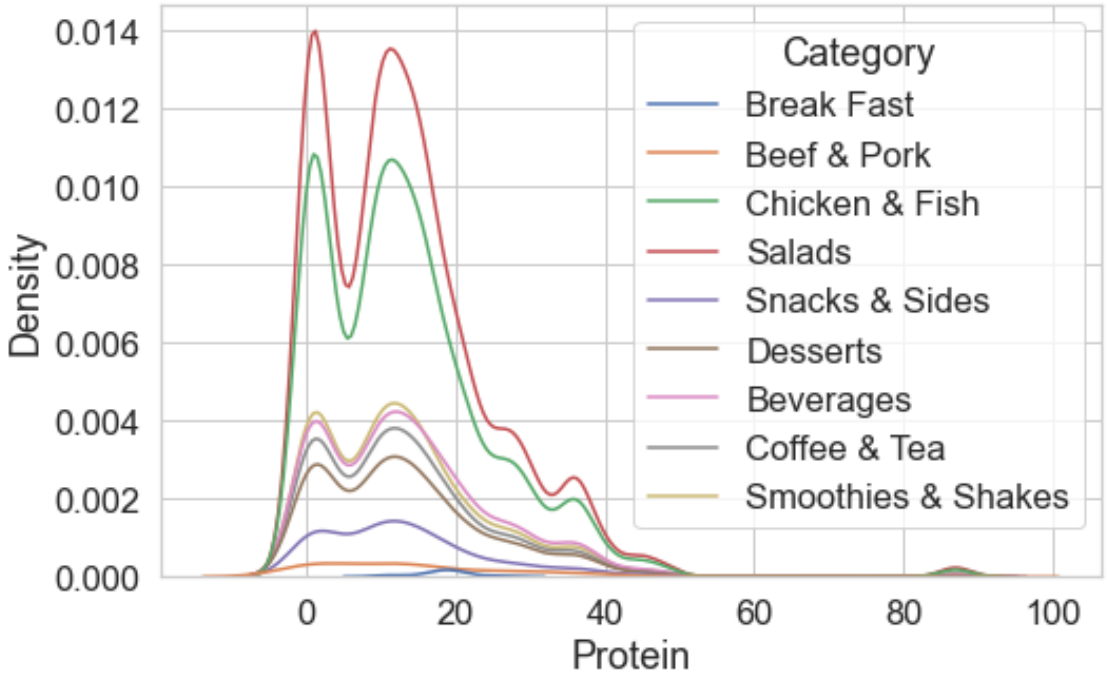
[65]:
```
# Set theme
sns.set_style('whitegrid')

# Violin plot
plt.figure(figsize=(12,8)) # Set plot dimensions
sns.violinplot(x='Dietary Fiber', y='lon', data=data)
```

[65]: <AxesSubplot:xlabel='Dietary Fiber', ylabel='lon'>



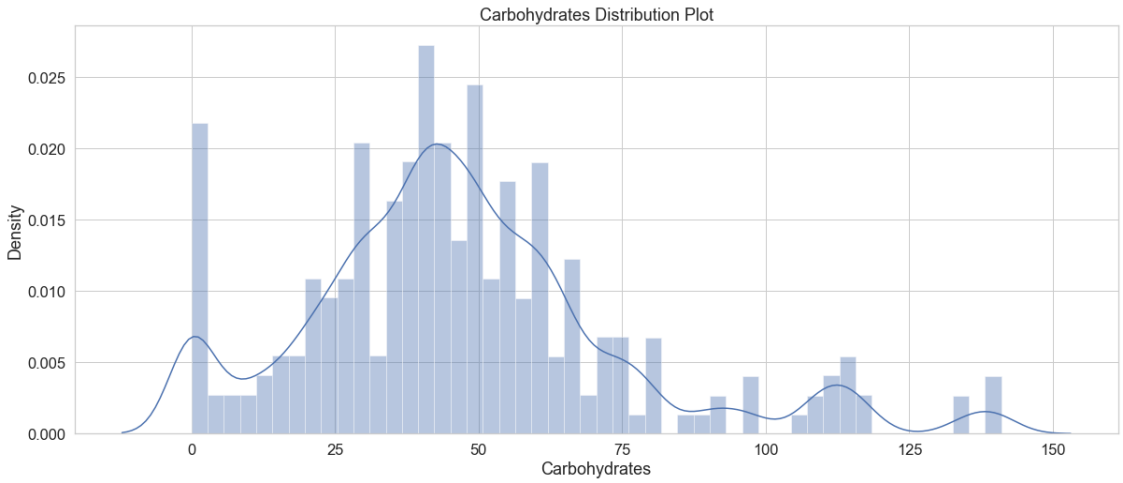
[66]:
```
#using KDE plot
plt.figure(figsize=(8, 5))
sns.kdeplot(x='Protein', data=data, hue='Category')
```

[66]: <AxesSubplot:xlabel='Protein', ylabel='Density'>



[67]:
```
#distplot
plt.figure(figsize=(20,8))
plt.title('Carbohydrates Distribution Plot')
sns.distplot(data.Carbohydrates)
plt.show()
```

```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```



12 Multivariate Analysis

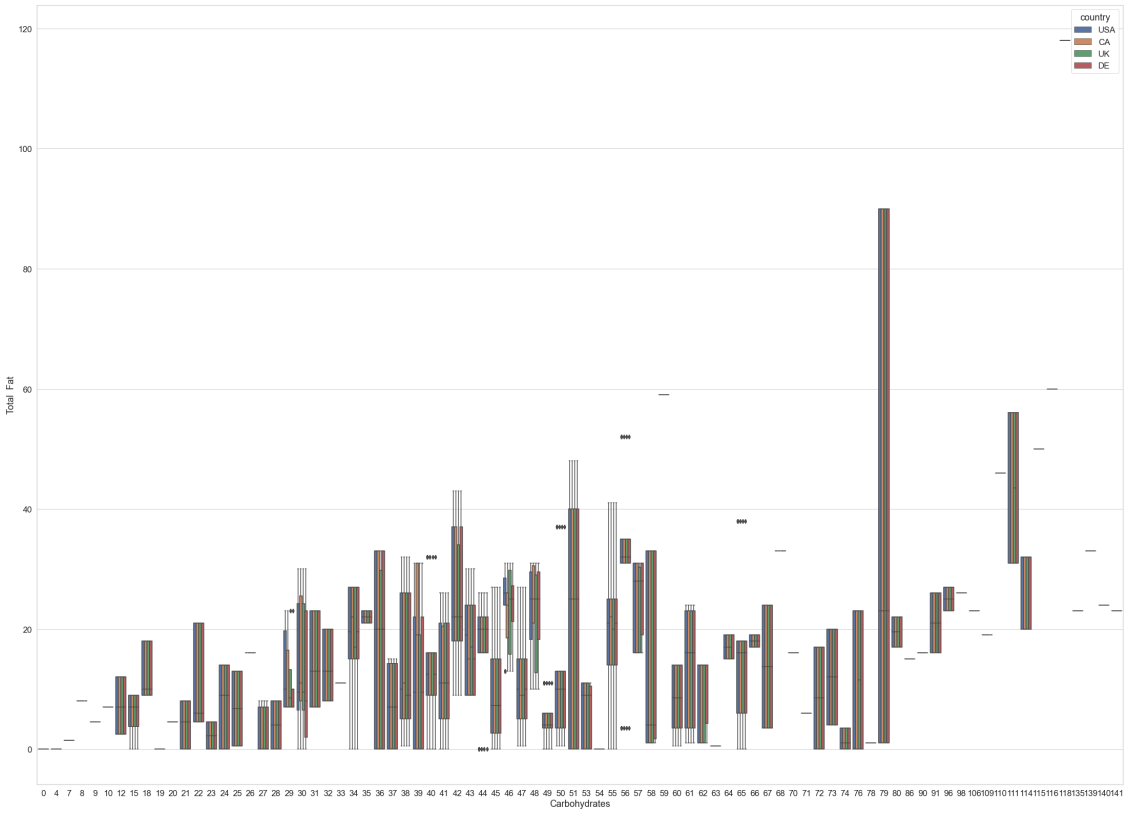
1.We perform multivariate analysis with more than 2 variables for any combination of categorical and continuous variables.

2.The combination can be: Categorical & Categorical, Categorical & Continuous and Continuous & Continuous.

3.Different methods are used to tackle these combinations during analysis process.

13 BoxPlot

[68]:
```
fig, ax1 = plt.subplots(figsize=(38,28))
testPlot = sns.boxplot(ax=ax1, x='Carbohydrates', y='Total  Fat',␣
 ↪hue='country', data=data)
```

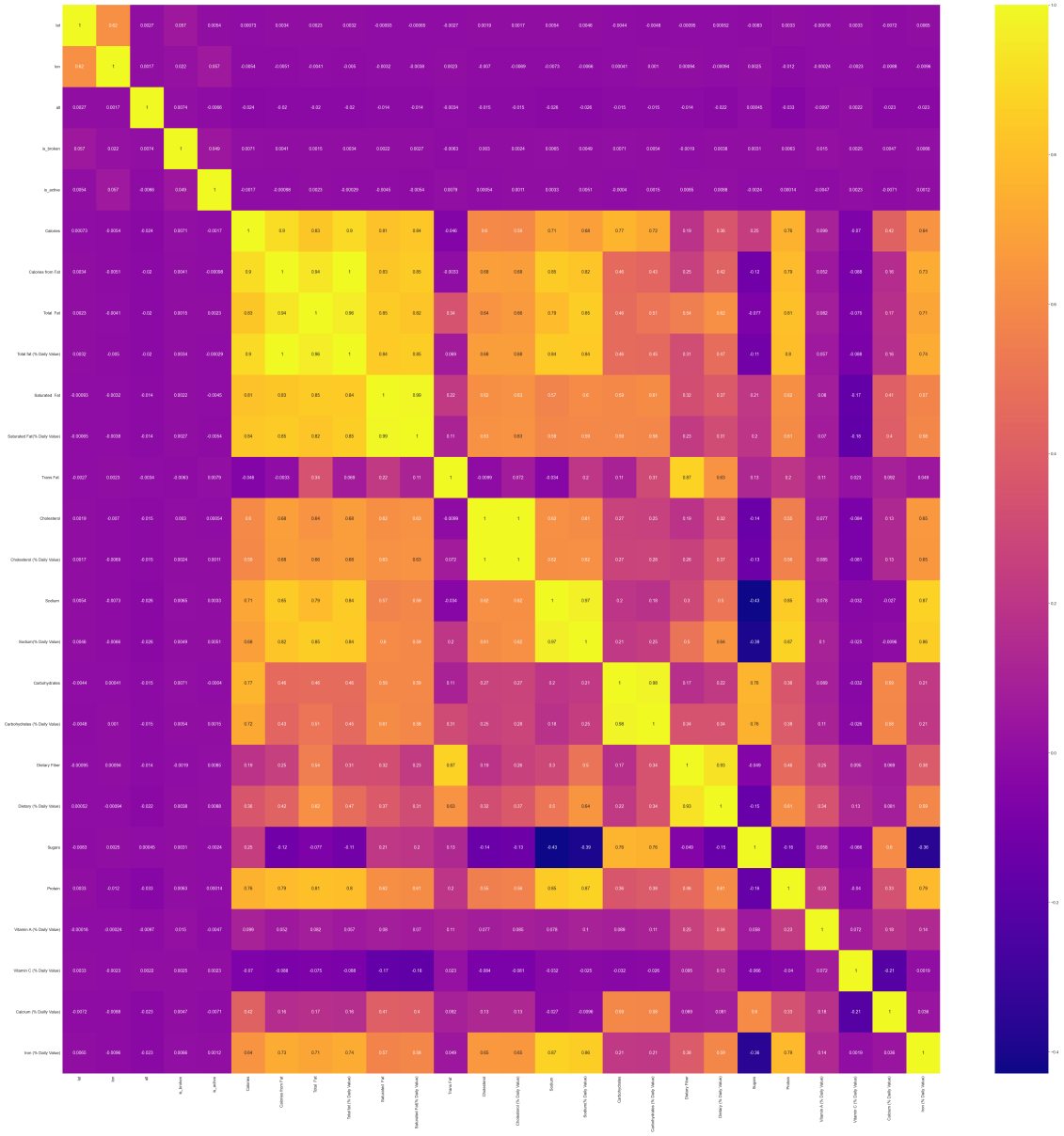


A box and whisker plot—also called a box plot—displays the five-number summary of a set of data. The five-number summary is the minimum, first quartile, median, third quartile, and maximum. In a box plot, we draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median. The whiskers go from each quartile to the minimum or maximum.

14 Heat map

[69]:
```
## Co-relation matrix
#A correlation matrix is a table showing correlation coefficients between␣
 ↪variables.
#Each cell in the table shows the correlation between two variables.
fig,ax = plt.subplots(figsize = (80,80))
corr = data.corr()
sns.heatmap(corr,annot=True,cmap = 'plasma')
```

[69]: <AxesSubplot:>

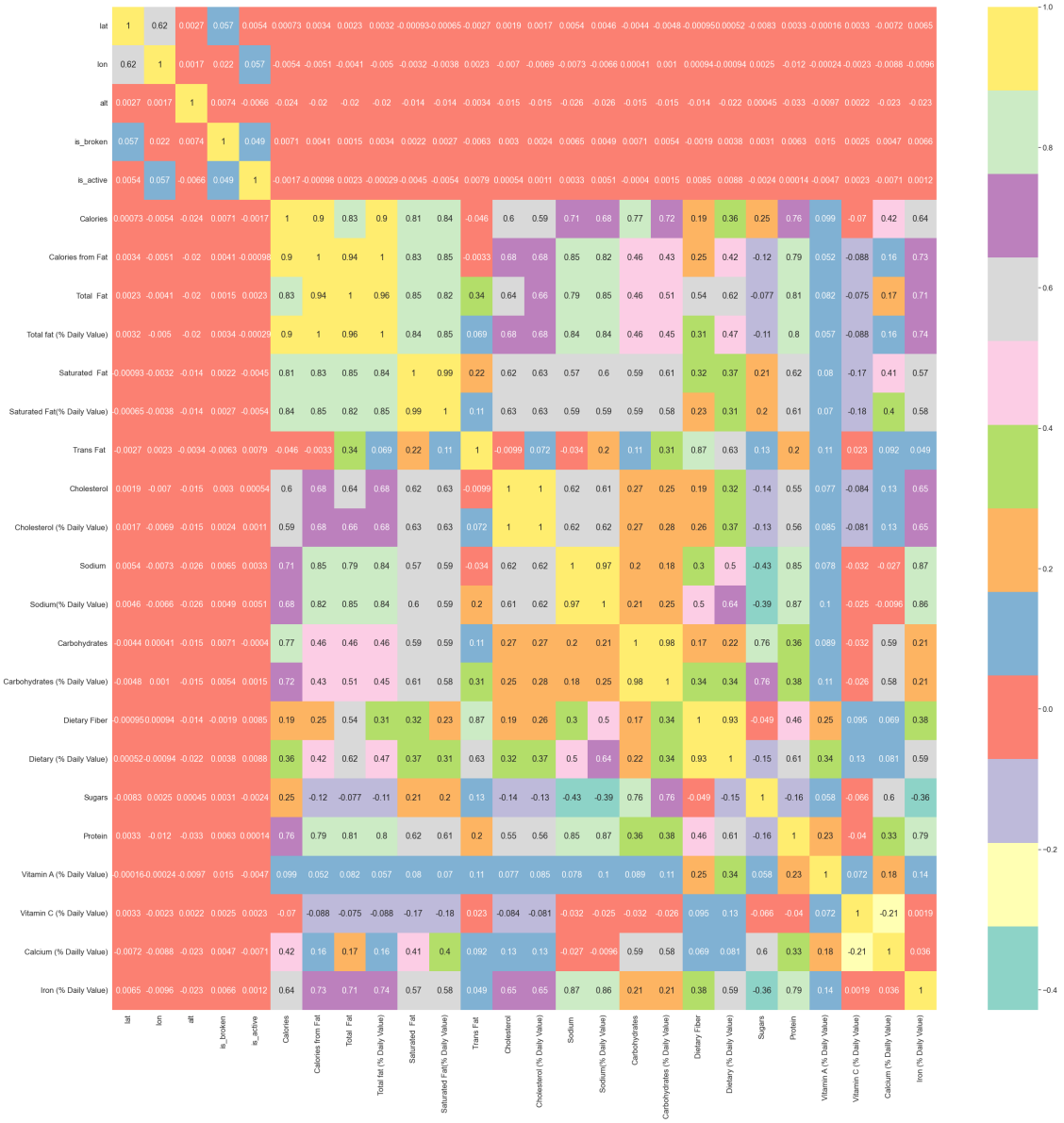


It can be seen that there are various parameters with strong correlations between them. Let's plot these into each of these powerful relationships one by one.

From the above graph we can say that there is a positve coorelation between Iron(%Daily Value) and lat and a negative coorelation between lat and lon. Also there is a negative coorelation between lon and lat, Age and sugars.

[70]:
```
## Co-relation matrix
fig,ax = plt.subplots(figsize = (40,40))
corr = data.corr()
sns.heatmap(corr,annot=True,cmap = 'Set3')
```

[70]: <AxesSubplot:>

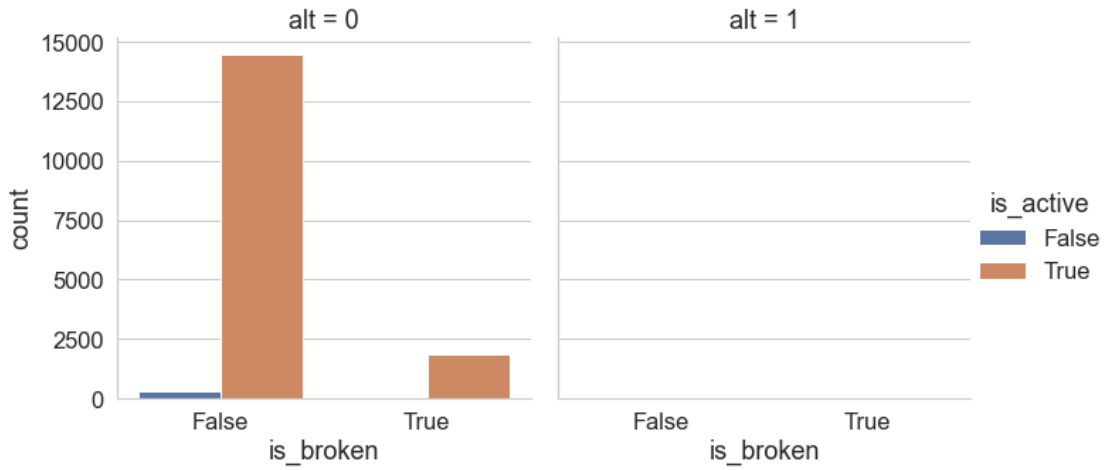


iron and sugara are weakly co releated calcium percentage daily value and saturated fat % daily value are strongly corelated vitamin c% daily value and calcium % daily value are weakly corelated protein and sugar are weakly corelated calcium% daily value and saturaed fat are strongly corelated

15 CatPlot

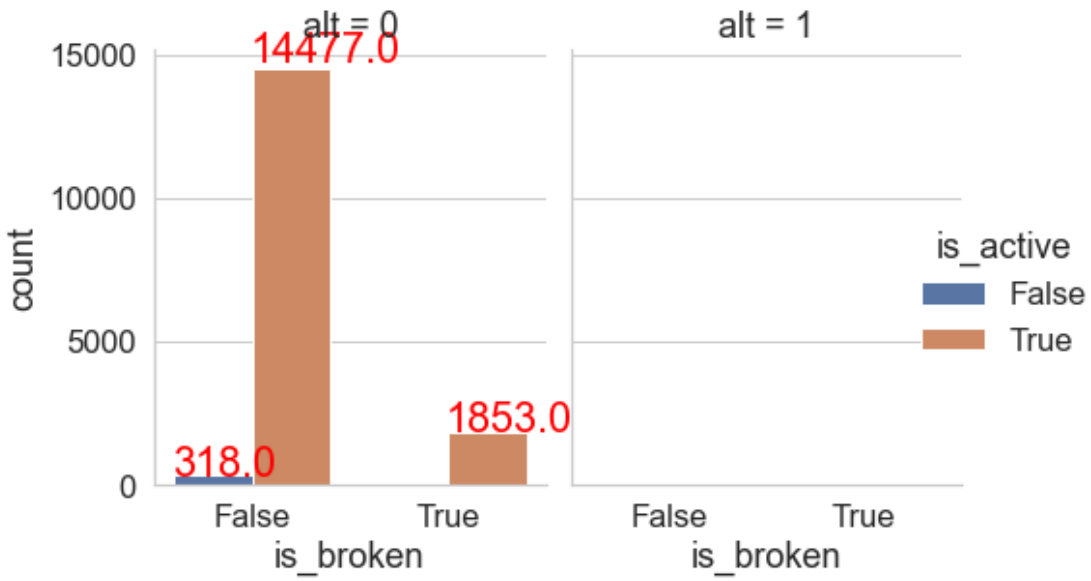
[71]:
```
# Lets more elaborate survived data with is_active and is_broken and we will␣
 ↪use catplot
#Figure-level interface for drawing categorical plots onto a FacetGrid
sns.catplot(data=data,col = "alt",x = "is_broken", hue="is_active",kind =␣
 ↪"count")
```

[71]:
```
<seaborn.axisgrid.FacetGrid at 0x1ec20c70df0>
```



[72]:
```
g = sns.catplot(data=data,col = "alt",x = "is_broken", hue="is_active",kind =␣
 ↪"count",legend=True)
g.fig.set_size_inches(8,5)
g.fig.subplots_adjust(top=0.81,right=0.86)

ax = g.facet_axis(0,0)
for p in ax.patches:
    ax.text(p.get_x() - 0.01,
            p.get_height() * 1.02,
           '{0:.1f}'.format(p.get_height()),
            color='red',
            rotation='horizontal',
            size='large')
```

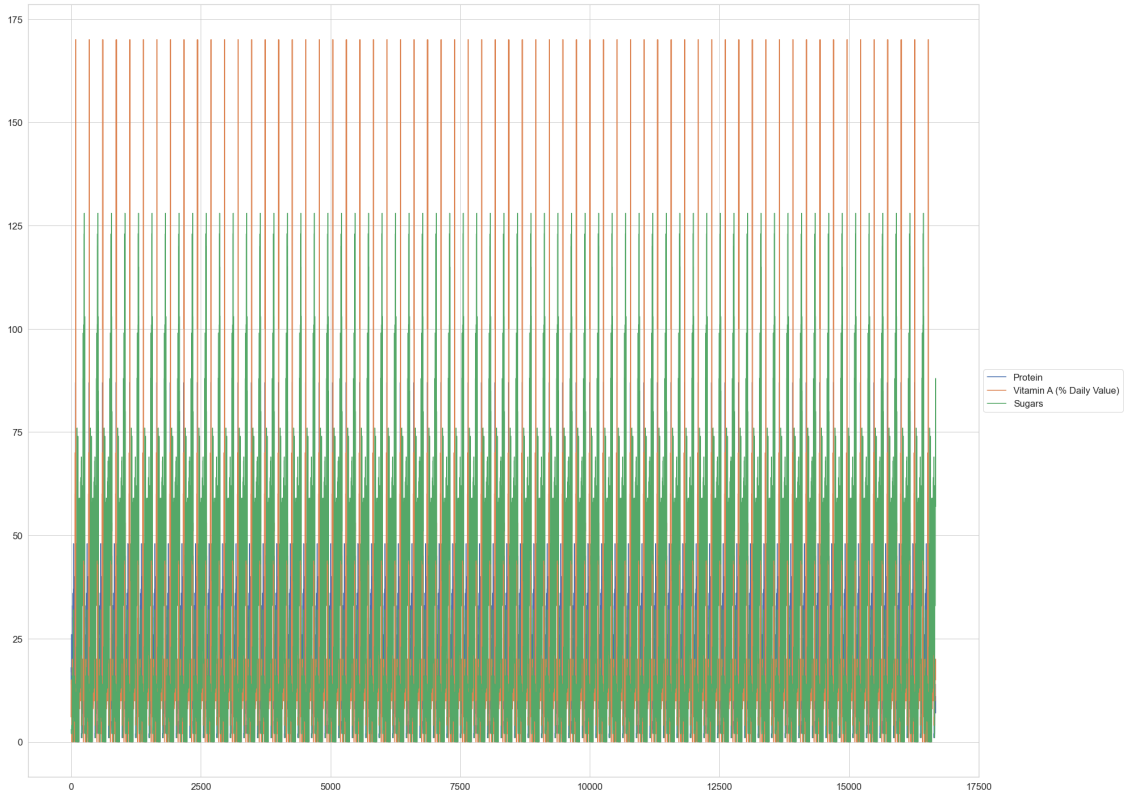
```
posx and posy should be finite values
posx and posy should be finite values
```



16 Profile Plot

Profile plot, used to shows the variation in each of the variables, by plotting the value of each of the variables for each of the samples.

[73]:
```
ax = data[["Protein","Vitamin A (% Daily Value)","Sugars"]].
↪plot(figsize=(30,25))
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5));
```



Profile plots provide another useful graphical summary of the data. These are only meaningful if all variables have the same units of measurement. They are not meaningful if the variables have different units of measurement. For example, some variables may be measured in grams while other variables are measured in centimeters. In this case, profile plots should not be constructed.

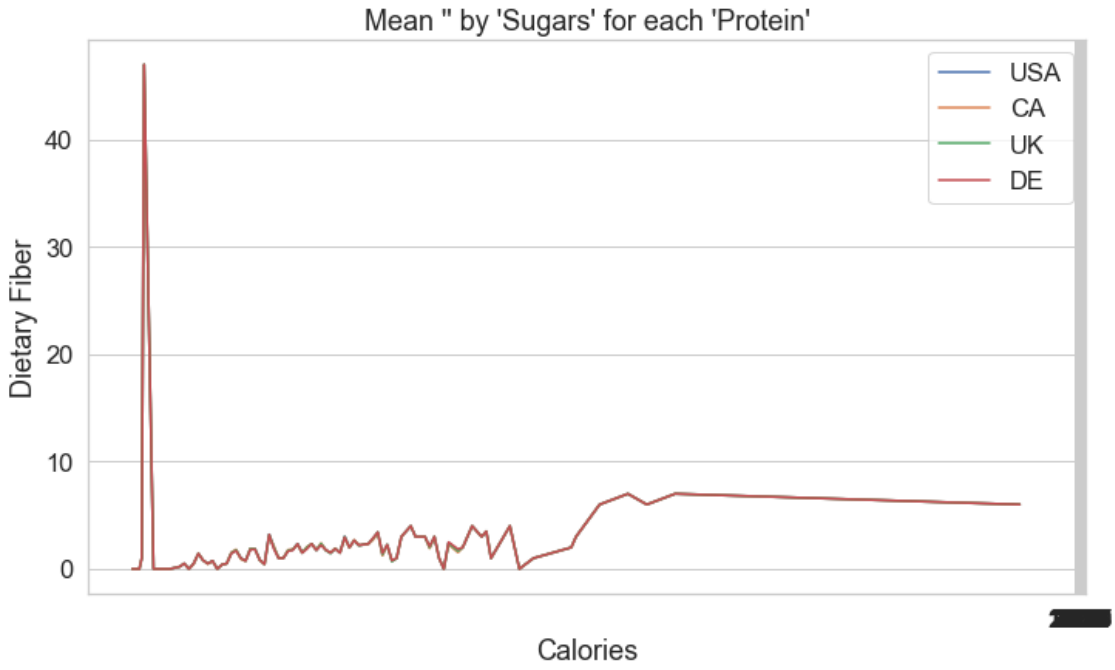
In the traditional profile plot, the samples mean for each group are plotted against the variables.

17 Calculating Summary Statistics for Multivariate Data

18 Plotting Line Plot

[74]:
```
fig, axes = plt.subplots(1, 1, figsize=(10, 6), constrained_layout =True)
ax3 = sns.lineplot(data=data, x='Calories', y='Dietary Fiber', hue='country',␣
 ↪ci=None)
ax3.set_xticks(np.arange(1996, 2020, 1))
ax3.set_title("Mean '' by 'Sugars' for each 'Protein'")

ax3.legend(loc='upper right')
```

[74]:
```
<matplotlib.legend.Legend at 0x1ec277beb50>
```



A line chart or line graph or curve chart is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments. It is a basic type of chart common in many fields. From the above figure we can infer that I have analyzed dietary fibre and calories as the numerical column whereas country categorial by plotting a line plot

19 ScatterPlot

[75]:
```
#scatter plot
fig = px.scatter(data, x="Sodium(% Daily Value)",y='Calories', color="Category",
                 size='Calories', hover_name="Item",trendline="ols",
                 template="plotly_dark",marginal_x="box",marginal_y="box")
fig.show()
```

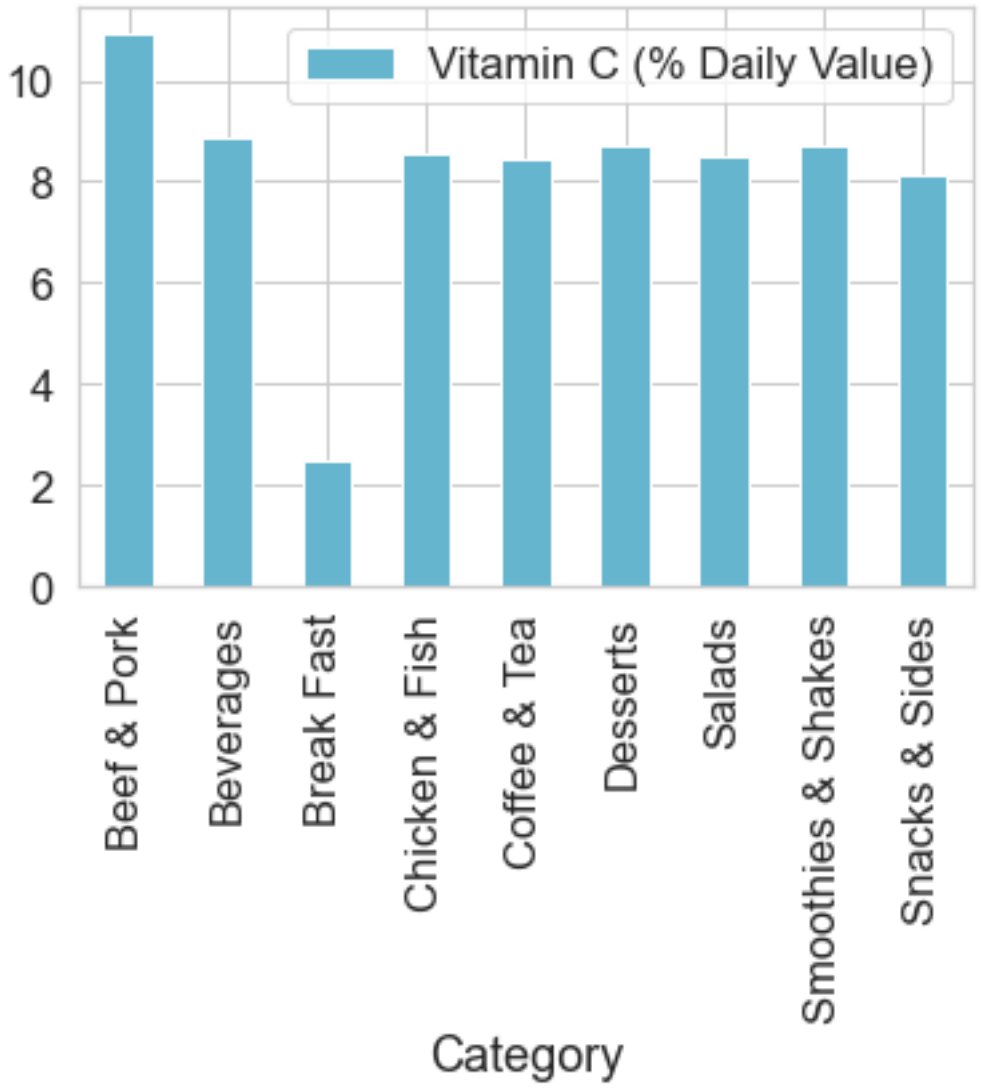
The scatter plots for Sodium (% Daily Value) seem to follow a similar distribution of points whereby MacDonald food items contributing the greatest amount of sodium are scaled largest. As evinced by the largest red circular plot, the 40-piece Chicken McNuggets are the greatest contributor to Sodium intake.

The Big Breakfast range with Hotcakes follow up as a close second as a contributor to the sodium amount.

Greatest amount of Sodium : Chicken McNuggets (40 piece)

[76]:
```
data.pivot_table('Vitamin C (% Daily Value)', 'Category').plot(kind='bar',␣
↪stacked=True, color = 'c')
```

[76]: <AxesSubplot:xlabel='Category'>



[77]:
```
#Check for item which contains nosugar.
print("Number of items in the menu: "+str(len(data.index)))
print("Number of items without sugar in the menu:␣
 ↪"+str(len(data.loc[data['Sugars'] == 0])))
print(data.loc[data['Sugars'] == 0])
```

```
Number of items in the menu: 16671
Number of items without sugar in the menu: 1600
            lat        lon  alt  is_broken  is_active   status state  \
38    -73.981850  40.757629    0      False       True  working    NY
78    -73.946077  40.789984    0      False       True  working    NY
79    -73.947781  40.632336    0      False       True  working    NY
```

```
80    -73.967846  40.800014    0       True       True   broken    NY
81    -74.027731  40.622159    0       True       True   broken    NY
…          …        …  …        …         …      …   …
16582   8.209411  50.075860    0      False       True  working   NaN
16583  11.437441  48.739624    0      False       True  working   NaN
16588   9.005489  48.687497    0      False       True  working   NaN
16589   9.183821  48.790858    0      False       True  working   NaN
16590   9.149583  48.920187    0       True       True   broken   NaN

              city                    street country  … Carbohydrates  \
38        New York              1188 6th Ave     USA  …             15
78        New York              1872 3rd Ave     USA  …             12
79        Brooklyn         2154 Nostrand Ave     USA  …             18
80        New York             2726 Broadway     USA  …             30
81        Brooklyn               430 86th St     USA  …             59
…            …                         …     …  …           …
16582    Wiesbaden      Dotzheimer Str. 182b      DE  …              0
16583   Ingolstadt        Münchener Str. 134      DE  …              0
16588    Böblingen  Wolfgang-Brumme-Allee 27      DE  …              0
16589    Stuttgart         Mailänder Platz 7      DE  …              0
16590  Ludwigsburg             Porschestr. 1      DE  …              0

      Carbohydrates (% Daily Value) Dietary Fiber Dietary (% Daily Value)  \
38                                5             2                        6
78                                4             1                        2
79                                6             1                        4
80                               10             2                        6
81                               20             3                       12
…                               …           …                       …
16582                             0             0                        0
16583                             0             0                        0
16588                             0             0                        0
16589                             0             0                        0
16590                             0             0                        0

       Sugars  Protein  Vitamin A (% Daily Value)  Vitamin C (% Daily Value)  \
38          0        1                          0                          2
78          0        9                          0                          2
79          0       13                          0                          2
80          0       22                          0                          4
81          0       44                          0                          8
…       …       …                         …                       …
16582       0        0                          0                          0
16583       0        0                          0                          0
16588       0        0                          0                          0
16589       0        0                          0                          0
16590       0        0                          0                          0
```

```
       Calcium (% Dailly Value)  Iron (% Daily Value)
38                            0                     2
78                            0                     2
79                            2                     4
80                            2                     6
81                            4                    10
…                           …                     …
16582                         0                     0
16583                         0                     0
16588                         0                     0
16589                         0                     0
16590                         0                     0

[1600 rows x 35 columns]
```

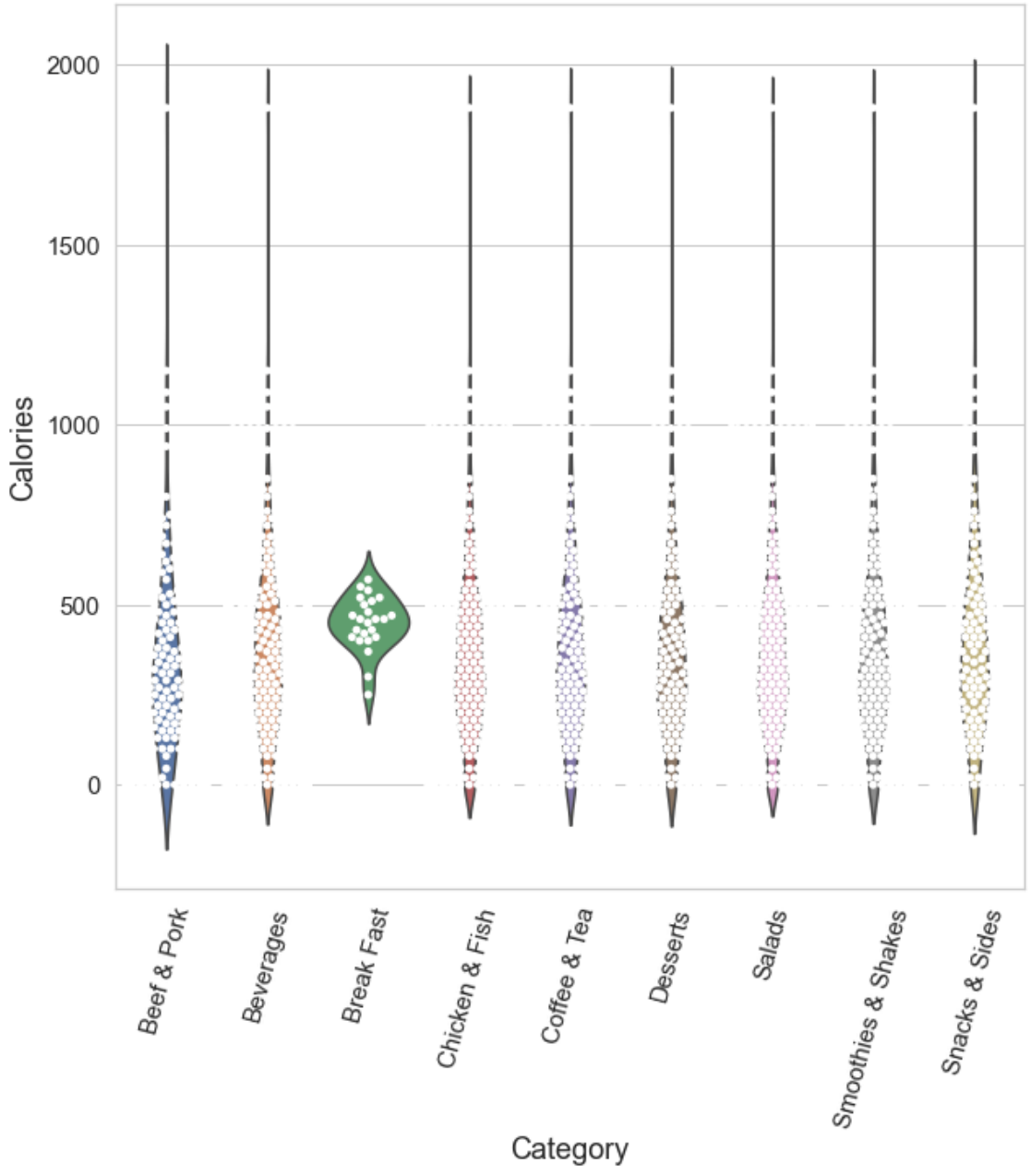
20 Violinplot & Swarmplot

[78]:
```
data['Category'] = data['Category'].astype('category')

plt.figure(figsize=(10,10))
sns.violinplot(x="Category", y="Calories", data=data, inner=None)
sns.swarmplot(x="Category", y="Calories", data=data, color="white",␣
 ↪edgecolor="gray")
plt.xticks(rotation = 75,fontsize=15)
plt.yticks(fontsize=15)
#plt.xlabel="Category"#
plt.xlabel('Category',fontsize=15)
plt.ylabel('Calories',fontsize=15)
plt.title('Calories by Category', fontsize=15)
plt.tight_layout()
plt.show()

n_bins = 5
x = np.
 ↪arange(min(data['Calories']),max(data['Calories']),(max(data['Calories'])-min(data['Calorie
 ↪n_bins)
plt.figure(figsize=(10,10))
plt.hist(data['Calories'], bins=n_bins, density=True)
plt.xticks(x,fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('Calories',fontsize=15)
plt.ylabel('Probability',fontsize=15)
plt.title('Calories Distribution', fontsize=15)
plt.tight_layout()
plt.show()
```

```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296:
```

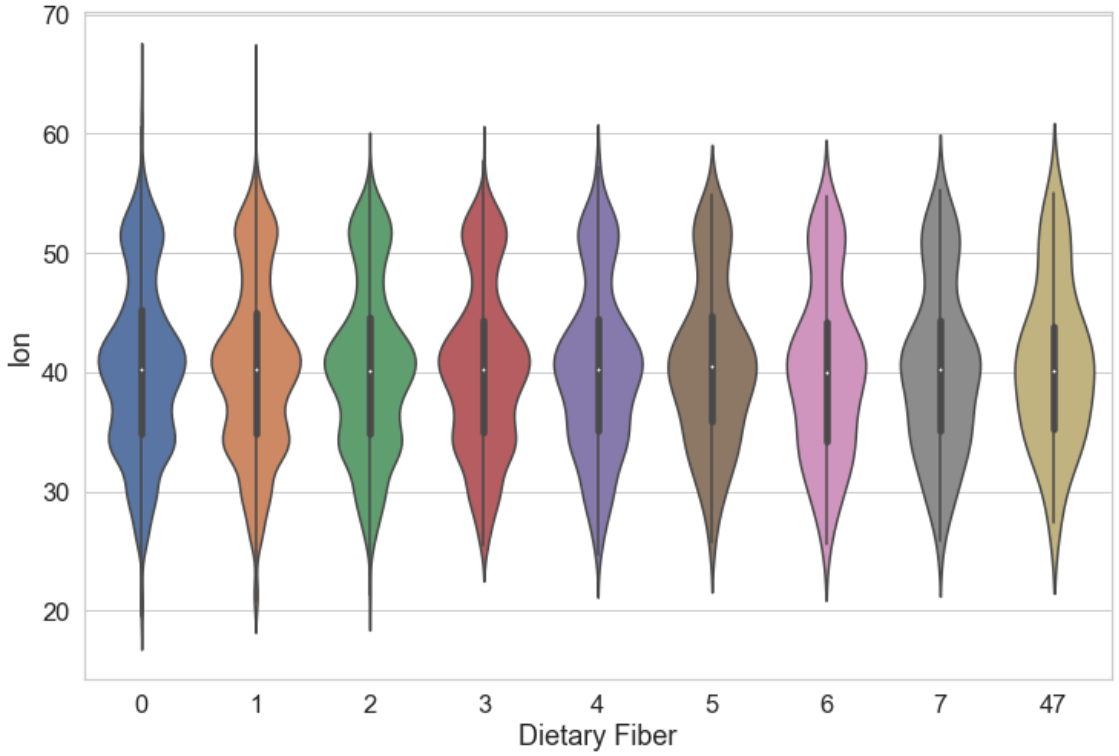
```
UserWarning:

11.7% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296:
UserWarning:

78.0% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296:
UserWarning:

89.7% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296:
UserWarning:

75.6% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296:
UserWarning:

71.9% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296:
UserWarning:

91.8% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296:
UserWarning:

78.4% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\categorical.py:1296:
UserWarning:

48.7% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_27924/1164815244.py in <module>
      7 plt.yticks(fontsize=15)
      8 #plt.xlabel="Category"#
----> 9 plt.xlabel('Category',fontsize=15)
     10 plt.ylabel('Calories',fontsize=15)
     11 plt.title('Calories by Category', fontsize=15)

TypeError: 'str' object is not callable
```



[79]:
```
# Set theme
sns.set_style('whitegrid')

# Violin plot
plt.figure(figsize=(12,8)) # Set plot dimensions
sns.violinplot(x='Dietary Fiber', y='lon', data=data)
```

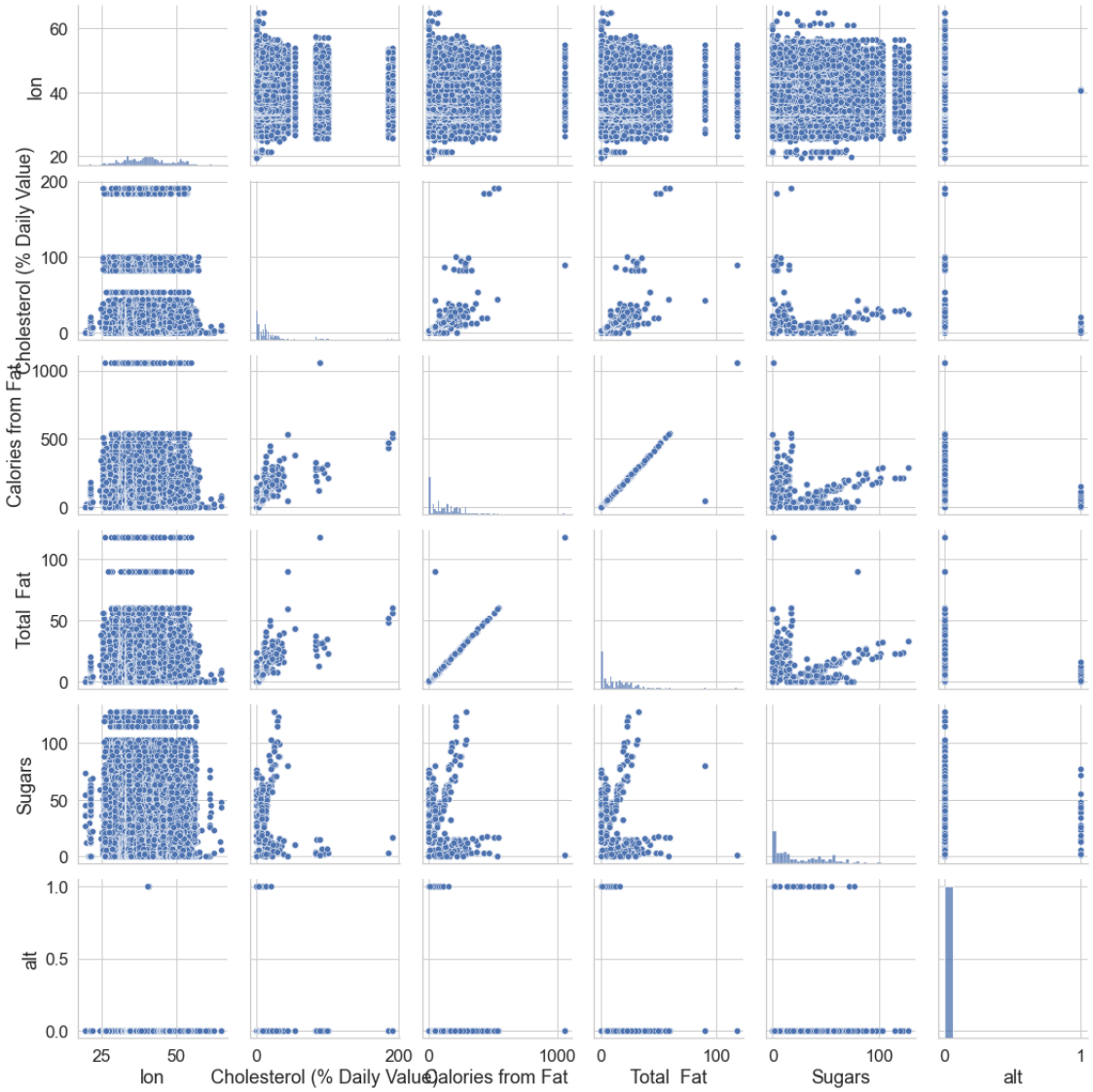
[79]: <AxesSubplot:xlabel='Dietary Fiber', ylabel='lon'>



we investigate the same thing as in the previous case, however, we set split=True. By doing so, instead of 8 violins, we end up with four — each side of the violin corresponds to a different column.

21 Pair Plot

[80]:
```
#pairs plot allows us to see both distribution of single variables and␣
 ↪relationships between two variables
num_var = ['lon', 'Cholesterol (% Daily Value)', 'Calories from Fat', 'Total ␣
 ↪Fat', 'Sugars', 'alt','country' ]
sns.pairplot(data[num_var], kind='scatter', diag_kind='hist')
plt.show()
```



Pair plot is used to understand the best set of features to explain a relationship between two variables or to form the most separated clusters. It also helps to form some simple classification models by drawing some simple lines or make linear separation in our data-set

I have defined a variable num_var. Here lon, Cholesterol (% Daily Value), Calories from Fat,Total Fat,`alt and`Sugars are numerical variables and country is the categorical variable.

22 is_broken vs is_active

[81]:
```
t = pd.crosstab(data['is_broken'], data['is_active']).sum(axis = 0)
t
```

[81]:
```
is_active
False      319
True     16352
dtype: int64
```

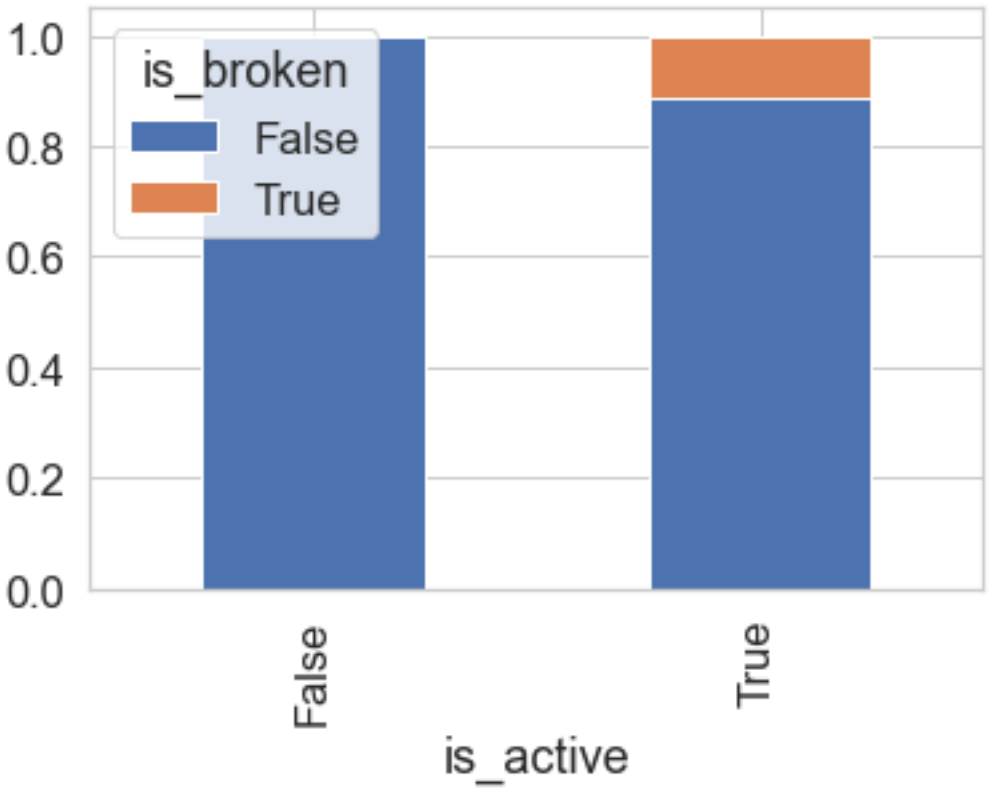
[82]:
```
pd.crosstab(data['is_broken'], data['is_active'], ).div(t, axis = 1)
```

[82]:
```
is_active  False      True
is_broken
False        1.0  0.886436
True         0.0  0.113564
```

23 Stacked Bar Chart

[83]:
```
pd.crosstab(data['is_broken'], data['is_active'], ).div(t, axis = 1).T.
↪plot(kind = 'bar', stacked = True)
```

[83]:
```
<AxesSubplot:xlabel='is_active'>
```

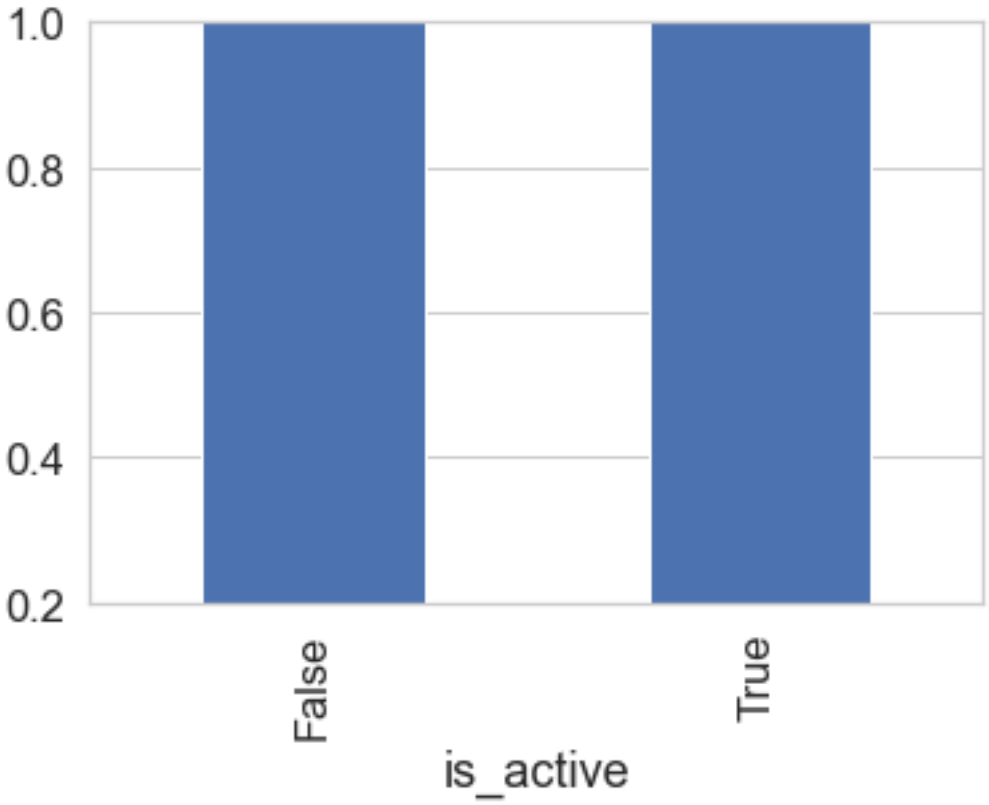


Proportion of is_broken visits converted into Revenue(17%) and proportion of broken visits converted into revenue(14%) is almost same. is_broken feature may not be much useful for predicting the target variable(is_active).

[84]:
```
print(t.div(t,axis = 0))
t.div(t,axis = 0).plot(kind = 'bar', stacked = True)
plt.ylim(0.2,1)
```

```
is_active
False    1.0
True     1.0
dtype: float64
```

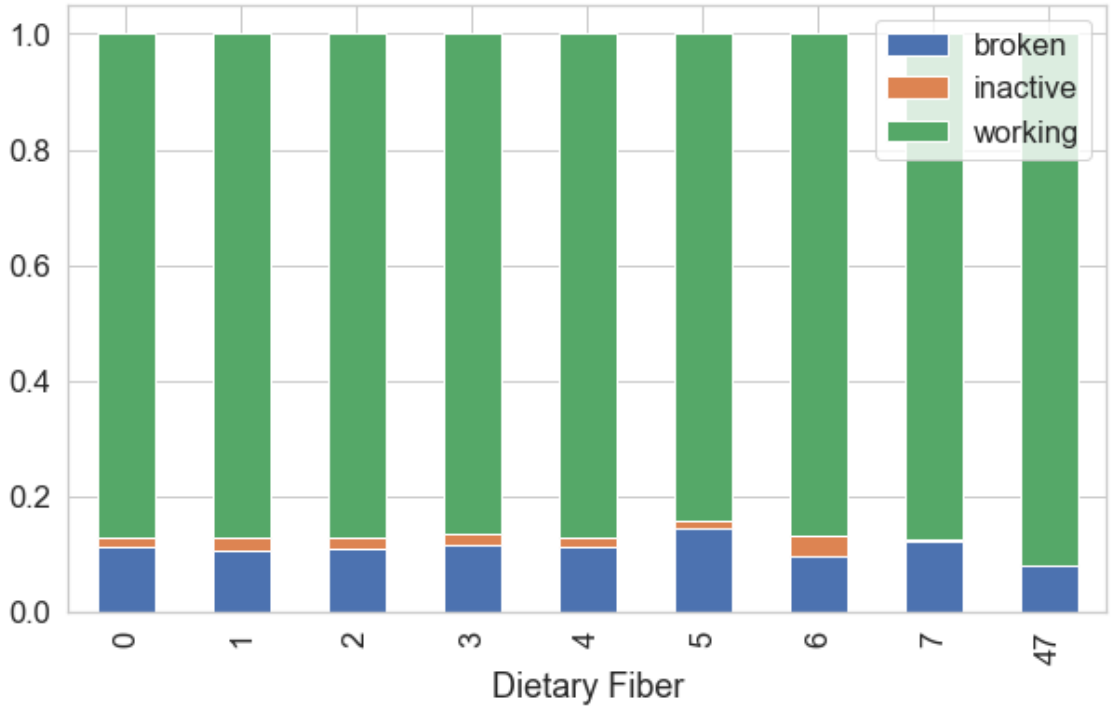
[84]: (0.2, 1.0)



[85]:
```
plt.figure(figsize = (10,6))
pd.crosstab(data['Dietary Fiber'], data['status']).div(pd.
  ↪crosstab(data['Dietary Fiber'], data['status']).sum(axis = 1), axis = 0).
  ↪plot(kind = 'bar', stacked = True, figsize = (10,6))
plt.legend(loc = 'upper right')
```

[85]: <matplotlib.legend.Legend at 0x1ec1dbc7730>

```
<Figure size 720x432 with 0 Axes>
```



Returning type Dietary Fiber almost uses all the status source in high proprtion than other Dietary types. For New visitors mostly the trafficSource is of type 5, 7. Other type visitor only have used type 20 traffic source. There is a huge differnce in difference in proportion of different Dietary Fiber type through out different status sources

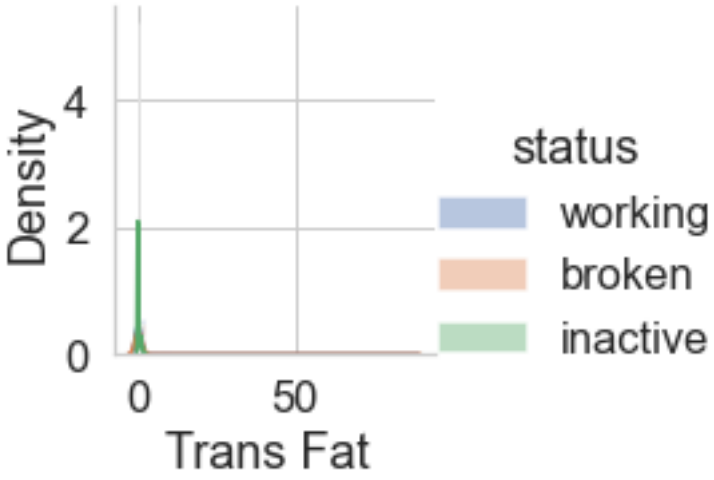
24 DistPlot

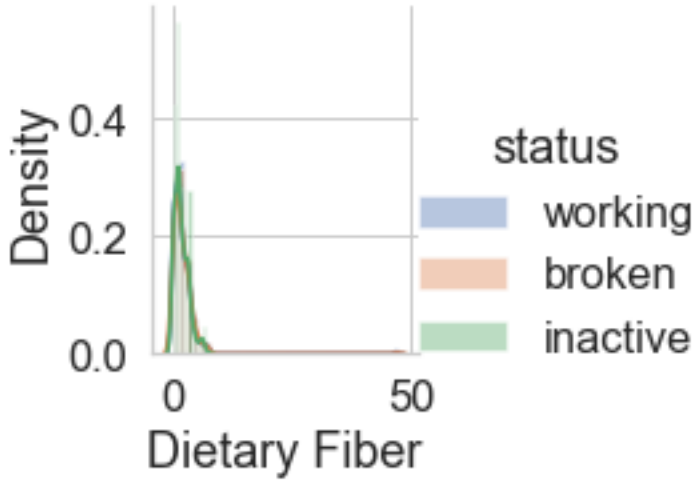
[86]:
```
plot = sns.FacetGrid(data, hue="status")
plot.map(sns.distplot, "Trans Fat ").add_legend()

plot = sns.FacetGrid(data, hue="status")
plot.map(sns.distplot, "Dietary Fiber").add_legend()

plot = sns.FacetGrid(data, hue="country")
plot.map(sns.distplot, "Sugars").add_legend()

plot = sns.FacetGrid(data, hue="country")
plot.map(sns.distplot, "Dietary Fiber").add_legend()

plt.show()
```

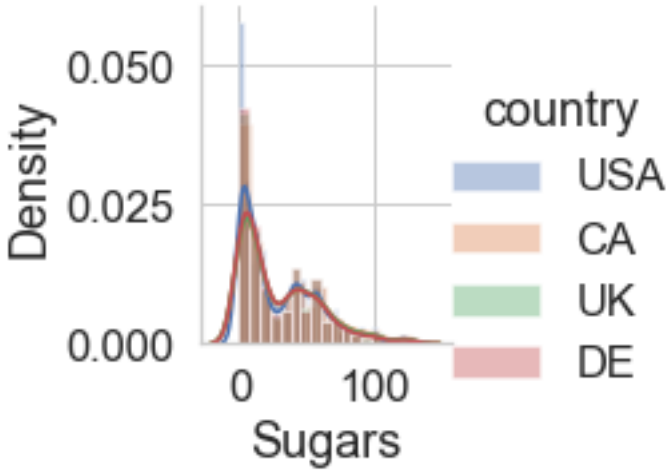
```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:
```

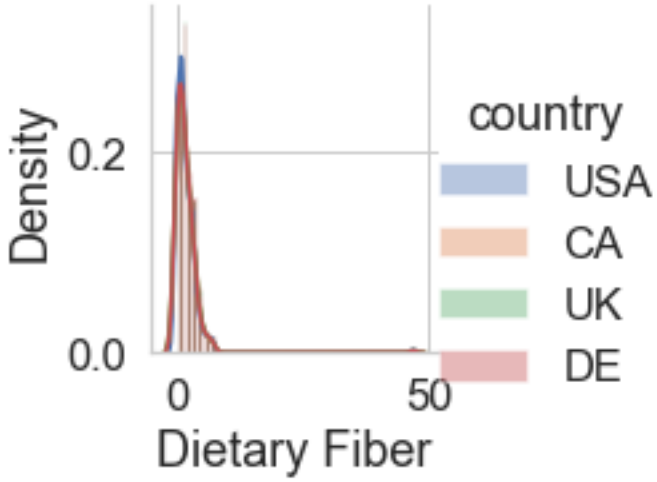
```
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:
```

```
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```


Density
4
2
0
0
50
Trans Fat
status
working
broken
inactive


Density
0.4
0.2
0.0
0
50
Dietary Fiber
status
working
broken
inactive





This function has been deprecated and will be removed in seaborn v0.14.0. It has been replaced by histplot() and displot(), two functions with a modern API and many more capabilities. distplot() function is used to plot the distplot. The distplot represents the univariate distribution of data i.e. data distribution of a variable against the density distribution. The seaborn. distplot() function accepts the data variable as an argument and returns the plot with the density distribution.

25 Countplot

[87]:
```
## Helper Functions

def multivariate_plot(data=data, x=None, hue=None, xlabel=None, rotation=None,␣
 ↪bar_label=True,
```

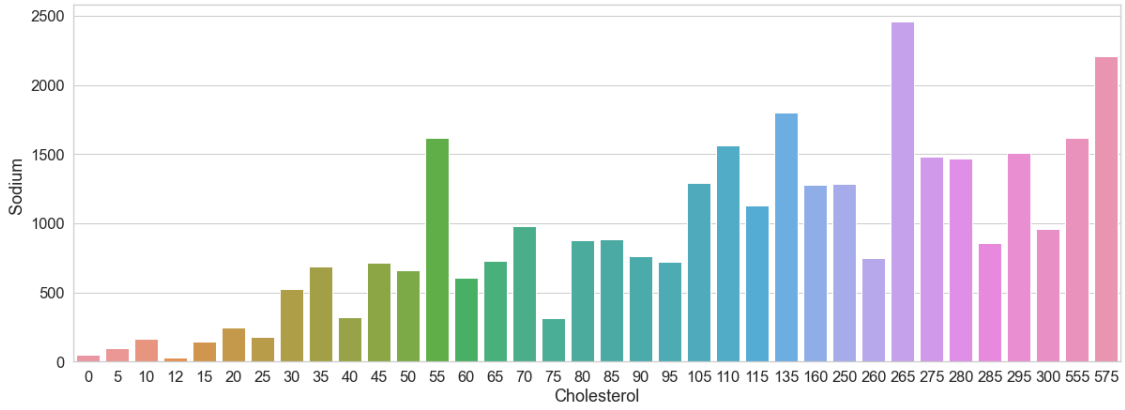
```
                      legend=False, rotate_label=False):
    fig, ax = plt.subplots(1, 1, figsize=(20, 7))
    data = data.groupby(x).mean()["Sodium "].sort_values().reset_index().copy()
    sns.barplot(data=data, x=x, y="Sodium ", hue=hue, ci=None)
    plt.ylabel(ylabel="Sodium usage", size=12)
    plt.xlabel(xlabel=xlabel, size=12)
    plt.title(label=f'Salaries by {xlabel}', size=16)
    plt.xticks(rotation=rotation)
    if legend:
        plt.legend()
    if bar_label and rotate_label:
        ax.bar_label(ax.containers[0], label_type='edge', size=10, padding=3,
                     color="#7f7f7f", rotation="vertical")
    elif bar_label and not rotate_label:
        ax.bar_label(ax.containers[0], label_type='edge', size=12, padding=1,
                     color="#7f7f7f")
    sns.despine(bottom=True, left=True)
    plt.show()
```

[88]:
```
def multivariate_count_plot(data=data, x=None, hue=None, xlabel=None,␣
↪hue_label=None, rotation=None,
                            legend=False, bar_label=False, convert=False):
    fig ,ax = plt.subplots(1, 1, figsize=(20, 7))
    sns.countplot(data=data, x=x, hue=hue)
    plt.ylabel(ylabel="", size=12)
    plt.xlabel(xlabel=xlabel, size=12)
    plt.title(label=f'{hue_label} vs. {xlabel}', size=16)
    plt.xticks(rotation=rotation)
    if bar_label:
        for i in range(data[hue].nunique()):
            ax.bar_label(ax.containers[i], label_type='edge', size=10,␣
↪padding=1,
                     color="#7f7f7f")
    if legend:
        if convert:
            country_labels = coco.CountryConverter().convert(data[hue].
↪unique(), to='name_short')
            plt.legend(title=hue_label, labels=country_labels)
        else:
            plt.legend(title=hue_label)
    sns.despine(bottom=True, left=True)
    plt.show()
```

[89]:
```
multivariate_plot(x="Cholesterol", xlabel="Cholesterol")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
~\AppData\Local\Temp/ipykernel_27924/477801976.py in <module>
----> 1 multivariate_plot(x="Cholesterol", xlabel="Cholesterol")

~\AppData\Local\Temp/ipykernel_27924/732953804.py in multivariate_plot(data, x,
 ↪hue, xlabel, rotation, bar_label, legend, rotate_label)
      6     data = data.groupby(x).mean()["Sodium "].sort_values().reset_index( .
 ↪copy()
      7     sns.barplot(data=data, x=x, y="Sodium ", hue=hue, ci=None)
----> 8     plt.ylabel(ylabel="Sodium usage", size=12)
      9     plt.xlabel(xlabel=xlabel, size=12)
     10     plt.title(label=f'Salaries by {xlabel}', size=16)

TypeError: 'str' object is not callable
```



[90]:
```
#Considering 3 variables
fig = px.bar(data, x="Category", y="Item",
             color="Item", barmode="group",facet_col="status")

#Setting up the title
fig.update_layout(
    margin=dict(l=20, r=20, t=50, b=20),
    paper_bgcolor="LightSteelBlue",
    title_text='Does Item  and category has anything to do with influence in␣
 ↪status?'
)
fig.show()
```

25.0.1 Insights:

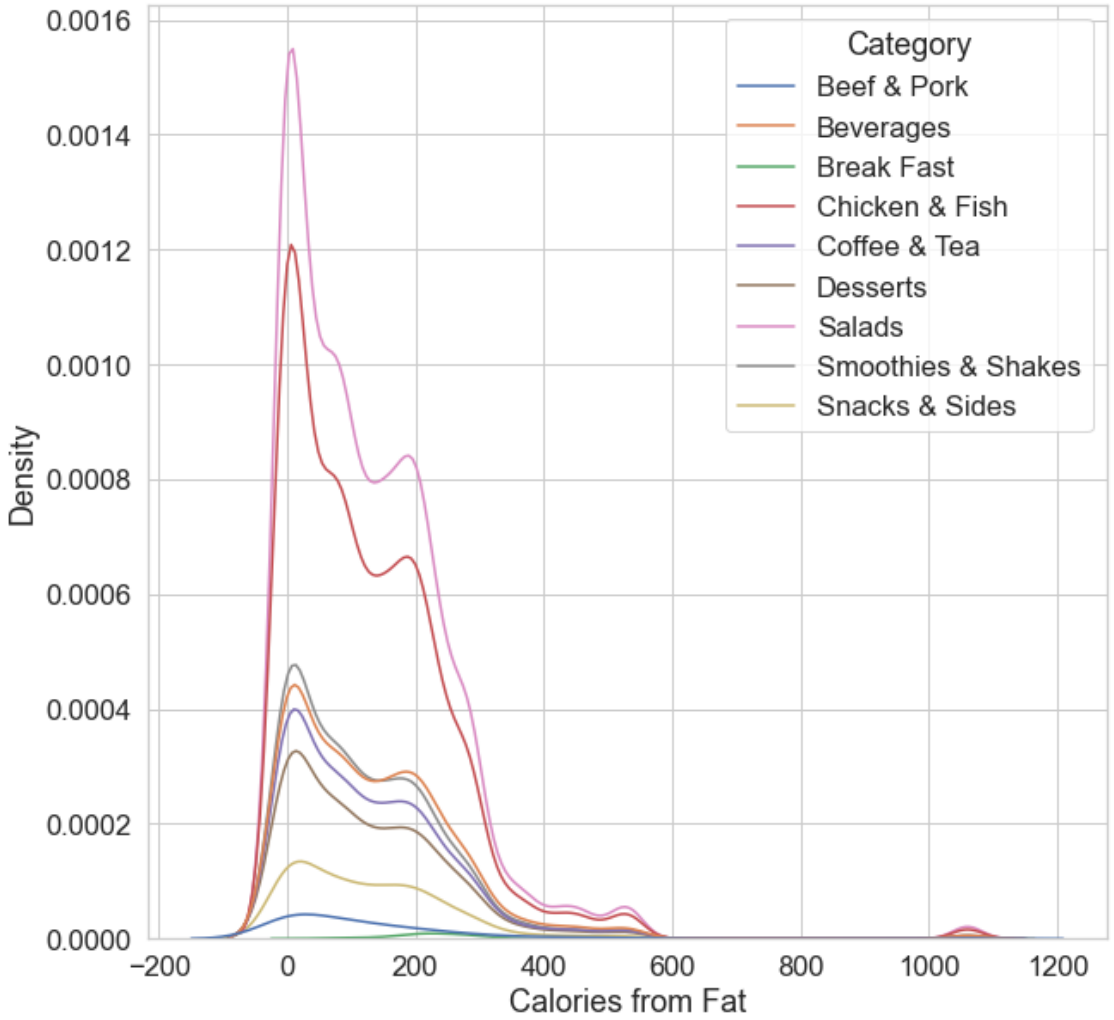
It can be seen that there is a high category when the Items are good and especially during the weekends. There are nearly 45 respondents have agreed that they consider time as an important factor during delivery during weekends with a good Items condition.And the status displays that whether

the orders are being taken from particular location It will display whether it is working/active or not working/inactive.

26 KDEPLOT

[91]:
```
#A kernel density estimate (KDE) plot is a method for visualizing the␣
 ↪distribution of observations in a dataset
plt.figure(figsize=(10, 10))
sns.kdeplot(data = data, x = 'Calories from Fat',hue = 'Category')
```

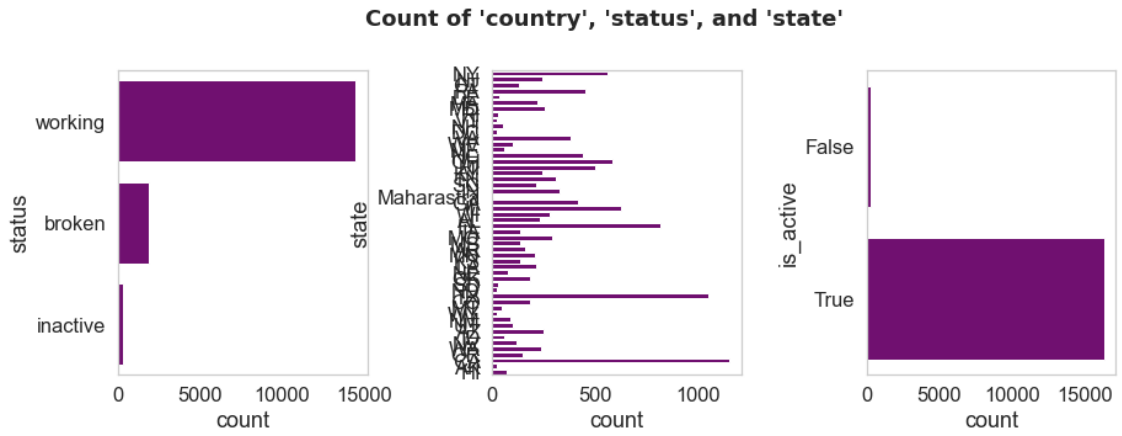
[91]: <AxesSubplot:xlabel='Calories from Fat', ylabel='Density'>



[92]:
```
# Count plots for 'country', 'status', 'state'.
categorical_fields = ['country','status','state','is_active']
x=0
```

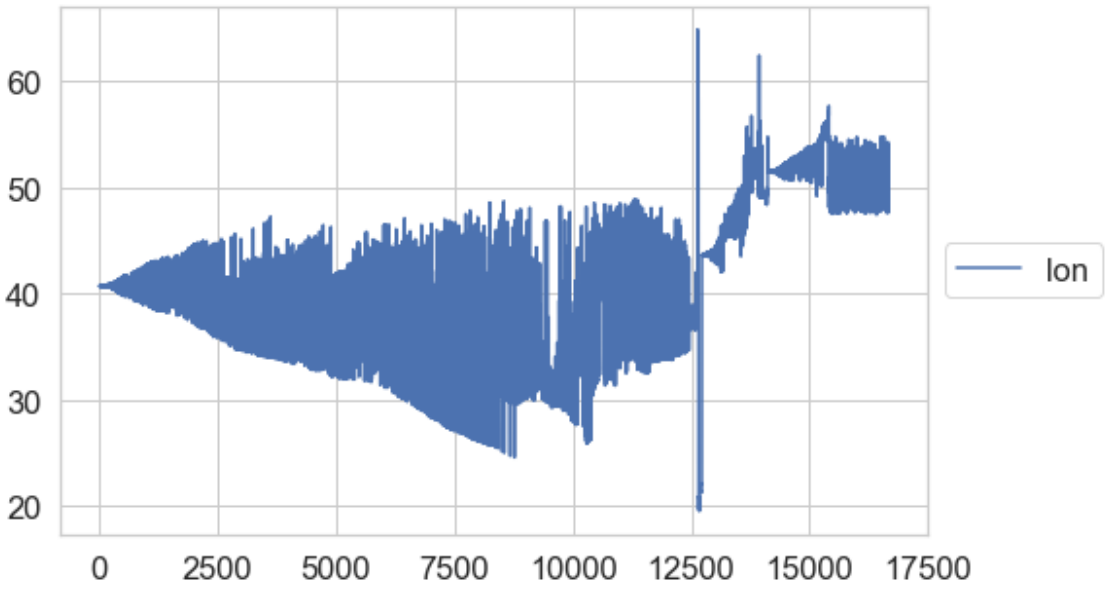
```
fig=plt.figure(figsize=(20,10))
plt.subplots_adjust(wspace = 0.5)
plt.suptitle("Count of 'country', 'status', and 'state'", x=0.4 ,y=0.95,␣
 ↪family='Sherif', size=18, weight='bold')
for i in data[categorical_fields[1:]]:
    ax = plt.subplot(241+x)
    ax = sns.countplot(data=data, y=i, color='purple')
    plt.grid(axis='x')
    x+=1
```

```
findfont: Font family ['Sherif'] not found. Falling back to DejaVu Sans.
```



Show the counts of observations in each categorical bin using bars. A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for countplot() , so you can compare counts across nested variables.

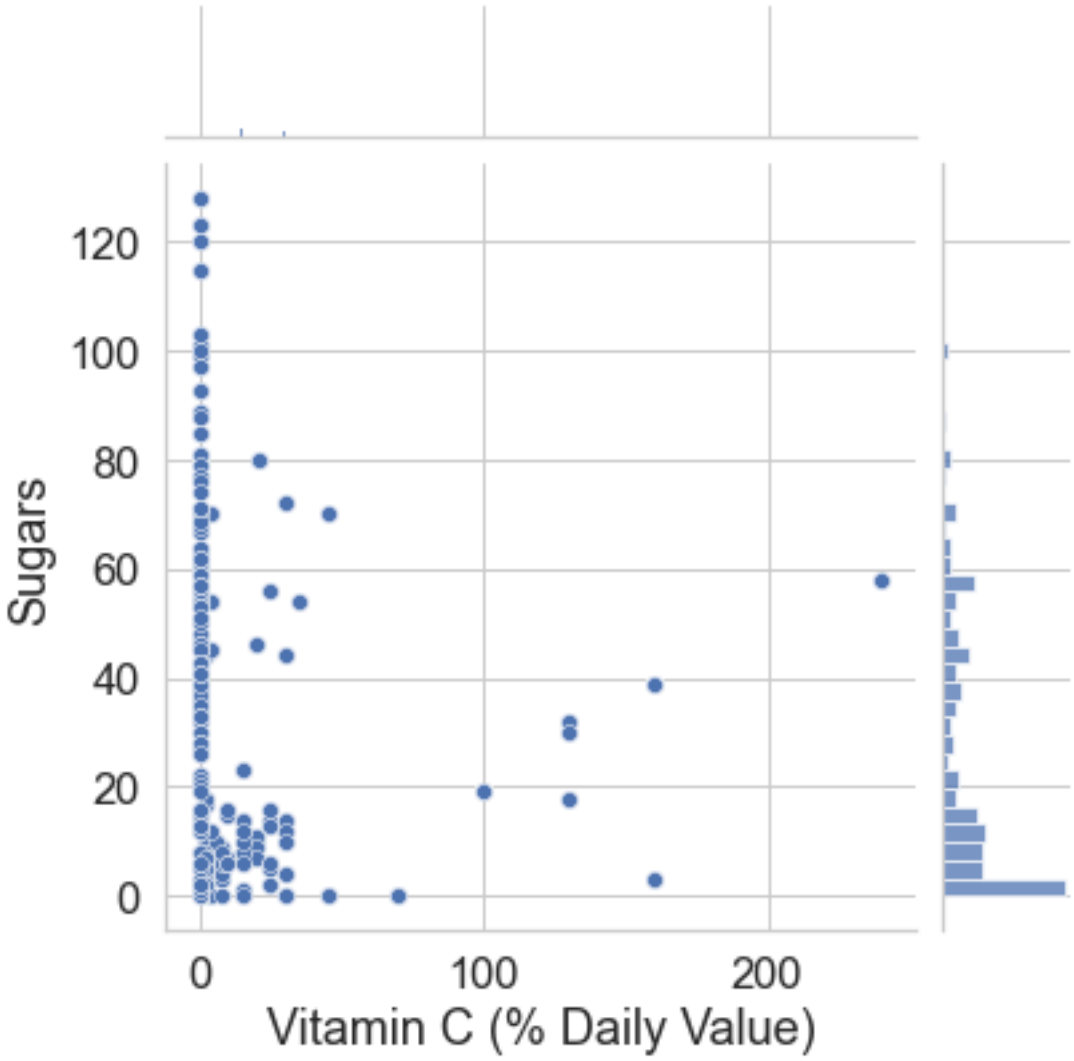
[93]:
```
ax = data[["status","lon"]].plot(figsize=(8,5))
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5));
```



27 Jointplot

[94]:
```
#Draw a plot of two variables with bivariate and univariate graphs.
#This function provides a convenient interface to the JointGrid class
sns.jointplot(data=data, x='Vitamin C (% Daily Value)', y='Sugars')
```

[94]: <seaborn.axisgrid.JointGrid at 0x1ec2b25f6d0>



[95]:
```
# separating numerical, categorical and dateTime features
cat_data = data.select_dtypes(exclude=[np.number, np.datetime64])
num_data = data.select_dtypes(exclude=[np.object, np.datetime64])
date_time_data = data.select_dtypes(include=[np.datetime64])
cat_data.head(2)
num_data.head(2)
# date_time_data.head(2)
```

```
C:\Users\Dell\AppData\Local\Temp/ipykernel_27924/514487366.py:3:
DeprecationWarning:

`np.object` is a deprecated alias for the builtin `object`. To silence this
warning, use `object` by itself. Doing this will not modify any behavior and is
safe.
```

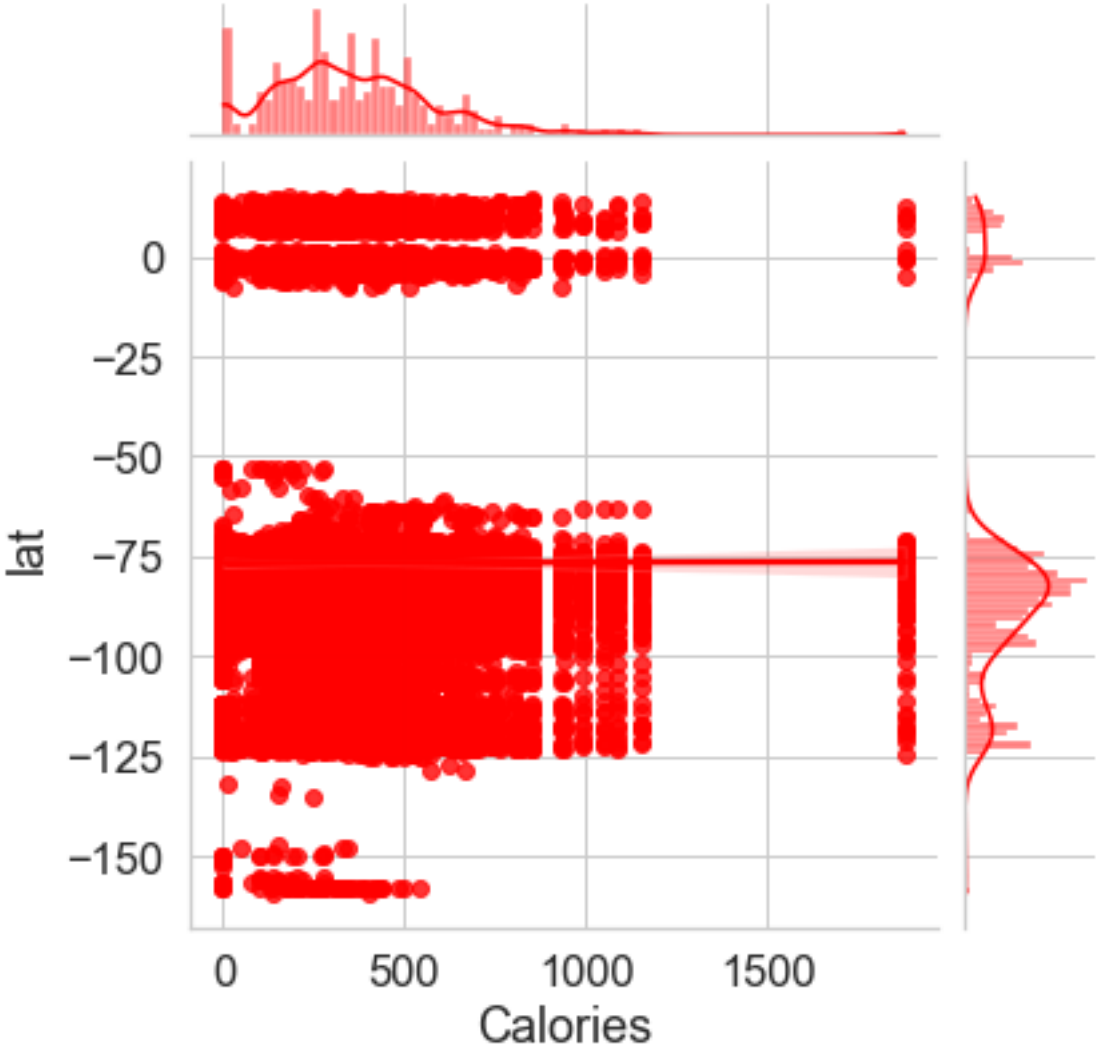
```
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
```

[95]:
```
         lat        lon  alt  is_broken  is_active    Category  Calories  \
0 -73.988281  40.718830    0      False       True  Break Fast       300
1 -74.005090  40.728794    0      False       True  Break Fast       250

   Calories from Fat  Total  Fat  Total fat (% Daily Value)  …  \
0                120                 13.0                         20  …
1                 70                  8.0                         12  …

   Carbohydrates  Carbohydrates (% Daily Value)  Dietary Fiber  \
0             31                             10              4
1             30                             10              4

   Dietary (% Daily Value)  Sugars  Protein  Vitamin A (% Daily Value)  \
0                       17       3       17                         10
1                       17       3       18                          6

   Vitamin C (% Daily Value)  Calcium (% Dailly Value)  Iron (% Daily Value)
0                          0                        25                    15
1                          0                        25                     8

[2 rows x 27 columns]
```

[96]:
```
plt.figure(dpi = 100, figsize = (5,4))
print("Joint plot of price with Other Variables ==> \n")
for i in  data.columns:
    if i != 'Calories' and i != 'Sugars' and i != 'country' and i != 'state'␣
 ↪and i != 'city' and i != 'street' and i != 'is_active' and i != 'is_broken'␣
 ↪and i != 'Item'and i!='lat ' and i!='lon' and i!='alt' and i!='status'and i!
 ↪='last_checked'and i!='Category' and i!='Item'and i!='Serving Sizes' and i!
 ↪='Saturated Fat(% Daily Value)' and i!='Calories from Fat' and i!='Total ␣
 ↪Fat' and i!='Total fat (% Daily Value)' and i!='Saturated  Fat' and i!
 ↪='Trans Fat 'and i!='Cholesterol'and i!='Cholesterol (% Daily Value)' and i!
 ↪='Sodium 'and i!='Sodium(% Daily Value)'and i!='Carbohydrates'and i!
 ↪='Carbohydrates (% Daily Value)'and i!='Dietary Fiber'and i!='Dietary (%␣
 ↪Daily Value)'and i!='Protein' and i!='Vitamin A (% Daily Value)'and i!
 ↪='Vitamin C (% Daily Value)'and i!='Calcium (% Dailly Value)' and i!='Iron␣
 ↪(% Daily Value)' :
        print(f"Correlation between Calories and {i} ==> ",data.corr().
 ↪loc['Calories'][i])
        sns.jointplot(x='Calories',y=i,data=data,kind = 'reg',color = 'red')
        plt.show()
```

```
Joint plot of price with Other Variables ==>
```

```
Correlation between Calories and lat ==>  0.0007253588166736518
<Figure size 500x400 with 0 Axes>
```

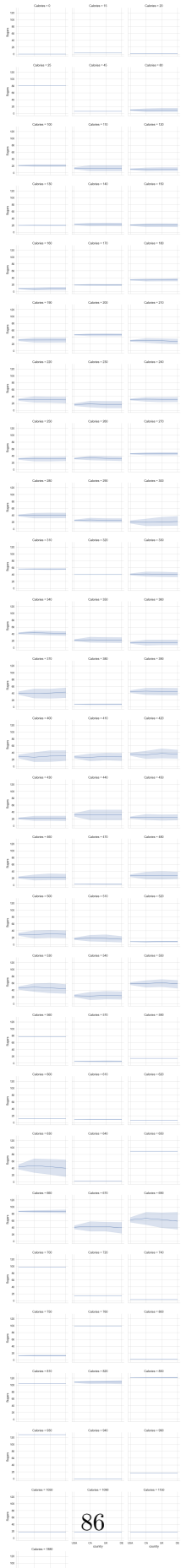


Joint Plots allow displaying the relationship between two variables as well as the 1D profile of both variables as shown below . We pass in the kind of graph we want to use to display the relationship between the two variables. In the first graph, we used a regression.

28 Relplot

[97]:
```
#figure-level function for visualizing statistical relationships using two␣
 ↪common approaches: scatter plots and line plots.
sns.relplot(col="Calories", y="Sugars", kind="line",x='country', data=data,␣
 ↪col_wrap=3)
```

[97]: <seaborn.axisgrid.FacetGrid at 0x1ec28434700>


Calories = 0
Calories = 10
Calories = 20
Calories = 30
Calories = 40
Calories = 50
Calories = 100
Calories = 110
Calories = 120
Calories = 130
Calories = 140
Calories = 150
Calories = 160
Calories = 170
Calories = 180
Calories = 190
Calories = 200
Calories = 210
Calories = 220
Calories = 230
Calories = 240
Calories = 250
Calories = 260
Calories = 270
Calories = 280
Calories = 290
Calories = 300
Calories = 310
Calories = 320
Calories = 330
Calories = 340
Calories = 350
Calories = 360
Calories = 370
Calories = 380
Calories = 390
Calories = 400
Calories = 410
Calories = 420
Calories = 430
Calories = 440
Calories = 450
Calories = 460
Calories = 470
Calories = 480
Calories = 500
Calories = 510
Calories = 520
Calories = 530
Calories = 540
Calories = 550
Calories = 560
Calories = 570
Calories = 590
Calories = 600
Calories = 610
Calories = 620
Calories = 630
Calories = 640
Calories = 650
Calories = 660
Calories = 670
Calories = 690
Calories = 700
Calories = 720
Calories = 740
Calories = 770
Calories = 790
Calories = 800
Calories = 810
Calories = 820
Calories = 830
Calories = 910
Calories = 940
Calories = 990
Calories = 1030
Calories = 1090
Calories = 1130
Calories = 1860
Frequency
country
USA
CA
MX
DE
GB
FR

[ ]: