NAME : BISHWAJIT SEN PROJECT NAME : FAST FOOD ANALYSIS

This dataset is webscrapped and pdf-to-csv converted data, which is organised and cleaned prior and only requires rows are gathered and exported. This includes various Fast Food Chain Giants like KFC, McDonald's, Burger King, Dominos, Pizza Hut and Starbucks.

The columns are as follows:

Company - Name of the Fast Food Company

Category - The category of the meal

Product - Name of the Meal

Per Serve Size - Quantity of the Meal Served

Energy (kCal) - Energy from the Meal in Kilo Calories

Carbohydrates (g) - Carbohydrates obtained from the Meal in grams

Protein (g) - Proteins obtained from the Meal in grams

Fiber (g) - Fibers obtained from the Meal in grams

Sugar (g) - Sugars obtained from the Meal in grams

Total Fat (g) - Total Fats obtained from the Meal in grams

Saturated Fat (g) - Saturated Fats obtained from the Meal in grams

Trans Fat (g) - Trans Fat obtained from the Meal in grams

Cholesterol (mg) - Cholesterol obtained from the Meal in grams

Sodium (mg) - Sodium obtained from the Meal in grams

Fast food analysis (KFC, McDonald's, Burger King, Dominos, Pizza Hut and Starbucks)

In [112…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Importing Data

In [113…
```
data = pd.read_csv(r"C:\Users\Dell\OneDrive\Desktop\PRACTICE\FOOD\archive (4)\Nutrition_Value_Dataset.csv")
data.head()
```

Out[113]:

	Company	Category	Product	Per Serve Size	Energy (kCal)	Carbohydrates (g)	Protein (g)	Fiber (g)	Sugar (g)	Total Fat (g)	Saturated Fat (g)	Trans Fat (g)	Cholesterol (mg)	Sodium (mg)
0	Pizza Hut	All Meals	Corn n Cheese (Personal)	143.5 g	432.60	65.64	17.91	3.85	0.0	10.93	5.14	0.16	16.19	499.72
1	Pizza Hut	All Meals	Country Feast (Personal)	178 g	407.60	67.11	16.73	7.19	0.0	8.03	3.24	0.11	66.80	818.00
2	Pizza Hut	All Meals	Double Cheese (Personal)	143 g	423.33	59.97	18.26	3.49	0.0	12.27	5.23	0.18	19.75	638.22
3	Pizza Hut	All Meals	Double Paneer Supreme (Personal)	174.5 g	474.03	52.86	20.07	3.79	0.0	20.26	9.25	0.33	71.72	1128.11
4	Pizza Hut	All Meals	Farmer`s Pick (Personal)	177 g	408.16	53.93	19.91	2.46	0.0	12.53	4.90	0.14	48.09	942.67

In [114…
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 530 entries, 0 to 529
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Company            530 non-null    object
 1   Category           530 non-null    object
 2   Product            530 non-null    object
 3   Per Serve Size     530 non-null    object
 4   Energy (kCal)      530 non-null    float64
 5   Carbohydrates (g)  530 non-null    float64
 6   Protein (g)        530 non-null    float64
 7   Fiber (g)          530 non-null    float64
 8   Sugar (g)          530 non-null    float64
 9   Total Fat (g)      530 non-null    float64
 10  Saturated Fat (g)  530 non-null    float64
 11  Trans Fat (g)      530 non-null    float64
 12  Cholesterol (mg)   530 non-null    float64
 13  Sodium (mg)        443 non-null    float64
dtypes: float64(10), object(4)
memory usage: 58.1+ KB
```

In [115…

```
data.describe()
```

Out[115]:

	Energy (kCal)	Carbohydrates (g)	Protein (g)	Fiber (g)	Sugar (g)	Total Fat (g)	Saturated Fat (g)	Trans Fat (g)	Cholesterol (mg)	Sodium (mg)
count	530.000000	530.000000	530.000000	530.0000	530.000000	530.000000	530.000000	530.000000	530.000000	443.000000
mean	377.304470	39.442302	16.710340	inf	8.276226	13.228528	5.946302	0.279443	118.620434	422.431578
std	338.315722	21.196943	24.425784	NaN	12.186627	10.740934	4.844024	3.265604	502.865434	471.300077
min	0.000000	0.000000	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	180.750000	24.227500	5.000000	1.6900	0.000000	5.000000	2.500000	0.030000	9.462500	1.450000
50%	329.645000	38.570000	11.445000	3.0000	3.200000	12.600000	5.240000	0.110000	30.085000	220.000000
75%	451.812500	54.120000	20.005000	5.9975	9.850000	18.360000	8.095000	0.207500	77.535000	754.975000
max	2880.000000	137.800000	203.000000	inf	64.220000	82.600000	33.000000	75.260000	9769.700000	2399.490000

In [116…

```
data
```

Out[116]:

	Company	Category	Product	Per Serve Size	Energy (kCal)	Carbohydrates (g)	Protein (g)	Fiber (g)	Sugar (g)	Total Fat (g)	Saturated Fat (g)	Trans Fat (g)	Cholesterol (mg)	Sodium (mg)
0	Pizza Hut	All Meals	Corn n Cheese (Personal)	143.5 g	432.60	65.64	17.91	3.85	0.0	10.93	5.14	0.16	16.19	499.72
1	Pizza Hut	All Meals	Country Feast (Personal)	178 g	407.60	67.11	16.73	7.19	0.0	8.03	3.24	0.11	66.80	818.00
2	Pizza Hut	All Meals	Double Cheese (Personal)	143 g	423.33	59.97	18.26	3.49	0.0	12.27	5.23	0.18	19.75	638.22
3	Pizza Hut	All Meals	Double Paneer Supreme (Personal)	174.5 g	474.03	52.86	20.07	3.79	0.0	20.26	9.25	0.33	71.72	1128.11
4	Pizza Hut	All Meals	Farmer`s Pick (Personal)	177 g	408.16	53.93	19.91	2.46	0.0	12.53	4.90	0.14	48.09	942.67
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
525	Dominos	All Meals	CHICKEN WINGS	174.0	352.90	24.50	35.30	5.76	10.4	12.60	4.80	0.19	308.70	NaN
526	Dominos	All Meals	CRISPY CHICKEN STRIPS	156.0	461.20	51.30	26.70	5.16	3.6	13.90	4.30	0.17	713.07	NaN
527	Dominos	All Meals	ZINGY PARCEL CHICKEN	210.0	667.50	17.30	19.20	16.92	10.6	31.30	14.10	0.56	541.49	NaN
528	Dominos	All Meals	TACO MEXICANA - CHICKEN	97.0	322.60	34.20	9.30	5.40	3.9	16.50	4.50	0.18	564.30	NaN
529	Dominos	All Meals	CALZONE POCKETS - CHICKEN	248.0	582.90	101.00	18.60	5.04	12.2	11.60	4.20	0.17	1171.60	NaN

530 rows × 14 columns

There is a problem in fiber column

Data Cleaning

In [117…

```
#Lowercase columns name
data.columns = data.columns.str.strip().str.lower()

#Convert first 3 columns in strings and lowercase the values
data.iloc[:,[0,1,2]] = data.iloc[:,[0,1,2]].astype(str)

for i in range(3):
    data.iloc[:,i] = data.iloc[:,i].str.lower()
data
```

Out[117]:

	company	category	product	per serve size	energy (kcal)	carbohydrates (g)	protein (g)	fiber (g)	sugar (g)	total fat (g)	saturated fat (g)	trans fat (g)	cholesterol (mg)	sodium (mg)
0	pizza hut	all meals	corn n cheese (personal)	143.5 g	432.60	65.64	17.91	3.85	0.0	10.93	5.14	0.16	16.19	499.72
1	pizza hut	all meals	country feast (personal)	178 g	407.60	67.11	16.73	7.19	0.0	8.03	3.24	0.11	66.80	818.00
2	pizza hut	all meals	double cheese (personal)	143 g	423.33	59.97	18.26	3.49	0.0	12.27	5.23	0.18	19.75	638.22
3	pizza hut	all meals	double paneer supreme (personal)	174.5 g	474.03	52.86	20.07	3.79	0.0	20.26	9.25	0.33	71.72	1128.11
4	pizza hut	all meals	farmer`s pick (personal)	177 g	408.16	53.93	19.91	2.46	0.0	12.53	4.90	0.14	48.09	942.67
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
525	dominos	all meals	chicken wings	174.0	352.90	24.50	35.30	5.76	10.4	12.60	4.80	0.19	308.70	NaN
526	dominos	all meals	crispy chicken strips	156.0	461.20	51.30	26.70	5.16	3.6	13.90	4.30	0.17	713.07	NaN
527	dominos	all meals	zingy parcel chicken	210.0	667.50	17.30	19.20	16.92	10.6	31.30	14.10	0.56	541.49	NaN
528	dominos	all meals	taco mexicana - chicken	97.0	322.60	34.20	9.30	5.40	3.9	16.50	4.50	0.18	564.30	NaN
529	dominos	all meals	calzone pockets - chicken	248.0	582.90	101.00	18.60	5.04	12.2	11.60	4.20	0.17	1171.60	NaN

530 rows × 14 columns

In [118…
```
#Drop Sodium column
data.drop('sodium (mg)', axis=1, inplace=True)
```

In [119…
```
idx = data['fiber (g)'].argmax()
data.iloc[idx,:]
```

Out[119]:
```
company                              burger king
category                               all meals
product               chicken wings fried ( 2pcs)
per serve size                              78*
energy (kcal)                             167.5
carbohydrates (g)                           0.9
protein (g)                                17.7
fiber (g)                                   inf
sugar (g)                                   0.0
total fat (g)                              10.3
saturated fat (g)                           4.0
trans fat (g)                               0.0
cholesterol (mg)                         182.31
Name: 166, dtype: object
```

Let's drop this (or these) row(s)

In [120…
```
# Replace infinite updated data with nan
data.replace([np.inf, -np.inf], np.nan, inplace=True)
# Drop rows with NaN
data.dropna(inplace=True)
```

In [121…
```
data.describe()
```

Out[121]:

	energy (kcal)	carbohydrates (g)	protein (g)	fiber (g)	sugar (g)	total fat (g)	saturated fat (g)	trans fat (g)	cholesterol (mg)
count	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000
mean	373.818906	39.515364	16.094789	4.688276	8.403065	13.025326	5.891456	0.283726	85.201686
std	335.824683	20.604596	23.592723	5.700365	12.236276	10.345230	4.740833	3.290395	193.457779
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	180.750000	24.550000	5.000000	1.680000	0.000000	4.850000	2.500000	0.040000	9.427500
50%	322.900000	38.800000	10.950000	3.000000	3.290000	12.595000	5.240000	0.120000	30.000000
75%	450.630000	54.120000	19.667500	5.767500	10.315000	18.097500	8.080000	0.210000	73.267500
max	2880.000000	101.000000	203.000000	45.000000	64.220000	82.600000	33.000000	75.260000	2119.750000

In [122…
```
data.shape
```

Out[122]:
```
(522, 13)
```

8 rows were dropped

EDA

Numeric Columns

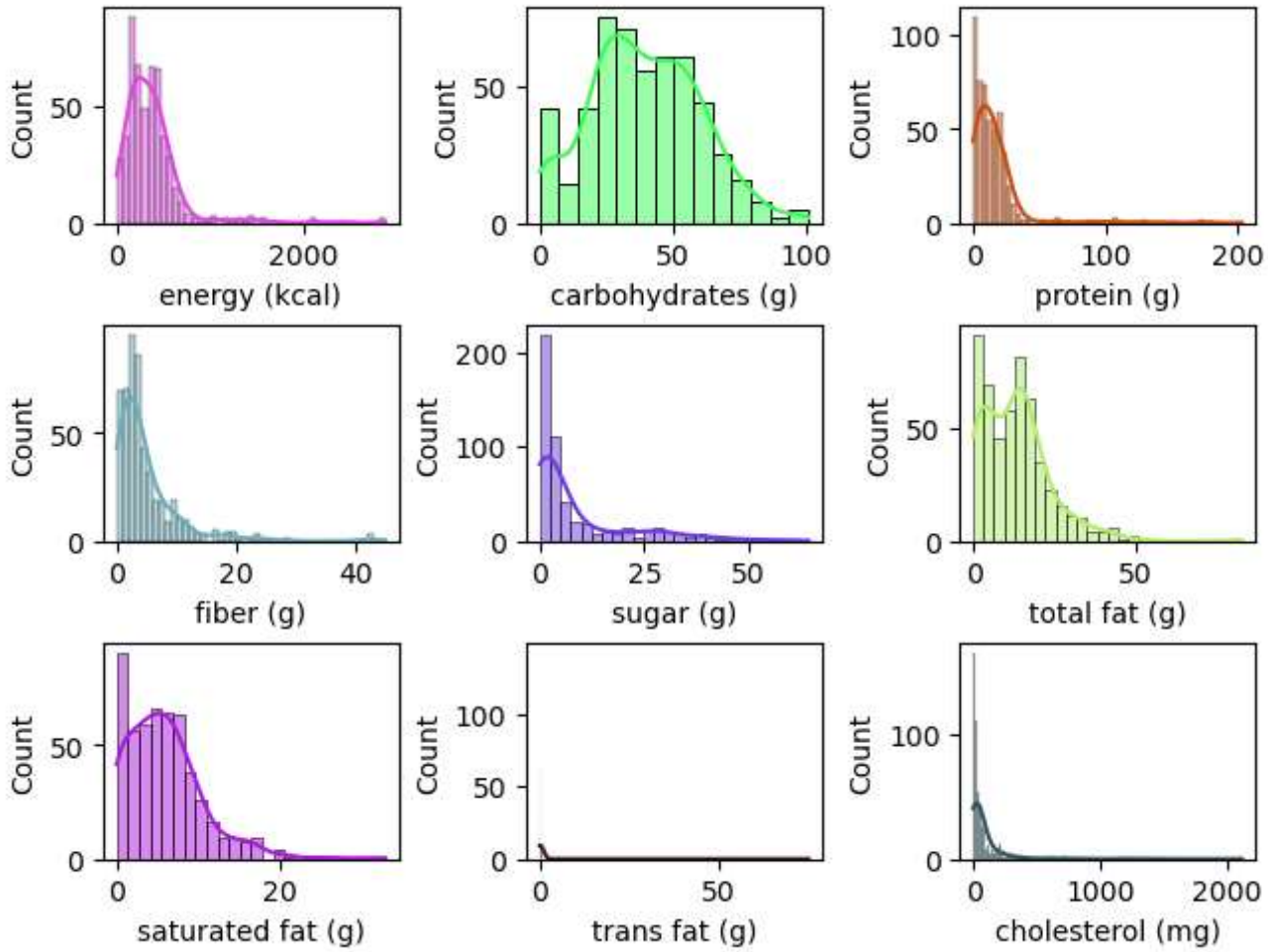
In [123…
```
numeric_cols = data.select_dtypes(include=['float']).columns
num_df = data.loc[:,numeric_cols]
num_df.describe()
```

Out[123]:

	energy (kcal)	carbohydrates (g)	protein (g)	fiber (g)	sugar (g)	total fat (g)	saturated fat (g)	trans fat (g)	cholesterol (mg)
count	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000
mean	373.818906	39.515364	16.094789	4.688276	8.403065	13.025326	5.891456	0.283726	85.201686
std	335.824683	20.604596	23.592723	5.700365	12.236276	10.345230	4.740833	3.290395	193.457779
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	180.750000	24.550000	5.000000	1.680000	0.000000	4.850000	2.500000	0.040000	9.427500
50%	322.900000	38.800000	10.950000	3.000000	3.290000	12.595000	5.240000	0.120000	30.000000
75%	450.630000	54.120000	19.667500	5.767500	10.315000	18.097500	8.080000	0.210000	73.267500
max	2880.000000	101.000000	203.000000	45.000000	64.220000	82.600000	33.000000	75.260000	2119.750000

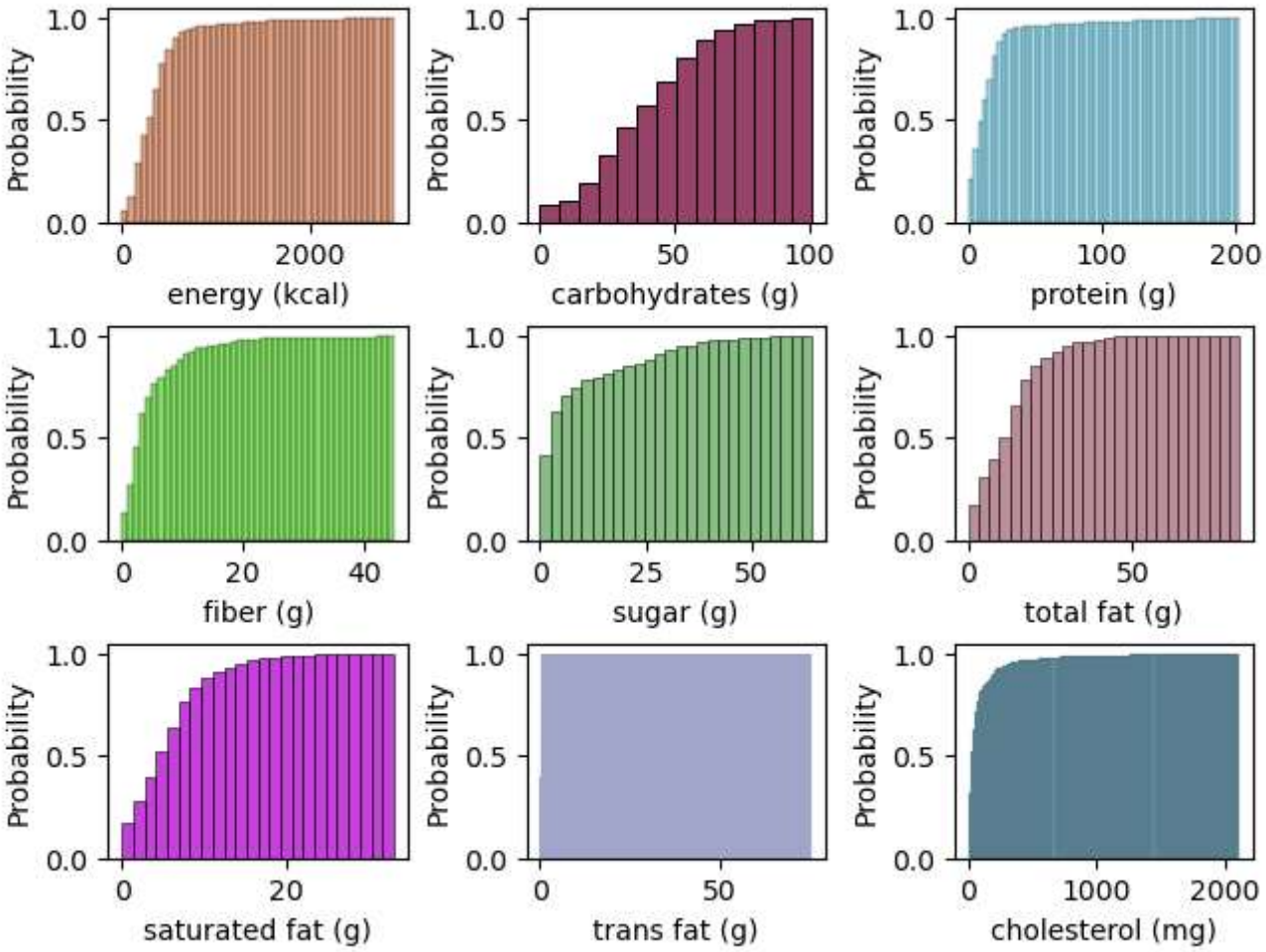
In [124…

```
_, axes = plt.subplots(3, 3, constrained_layout=True)
for col, ax in zip(num_df.columns, axes.flatten()):

    sns.histplot(num_df[col], kde=True, color=np.random.rand(3), ax=ax)
```



In [125…

```
_, axes = plt.subplots(3, 3, constrained_layout=True)
for col, ax in zip(num_df.columns, axes.flatten()):

    sns.histplot(num_df[col], stat='probability', cumulative=True, color=np.random.rand(3), ax=ax)
```



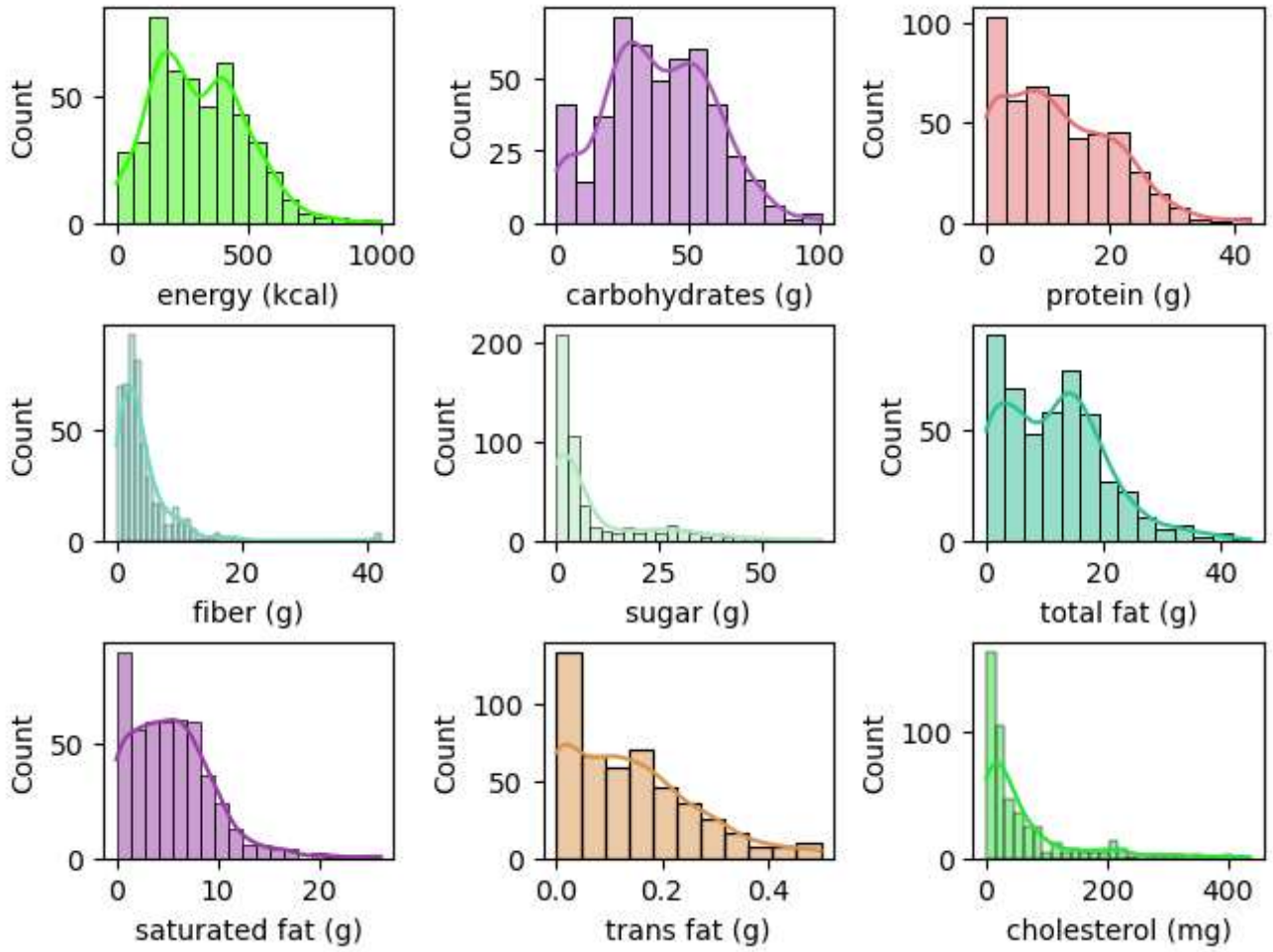
A thing one can notice is that trans fat column has 75% of data at 0.2 while the max is at 75. There are clearly outliers. \ This charateristic is similar in cholesterol. Let's try to understand better what happens in fiber column. \ In general most of the data are in the first half of the range.

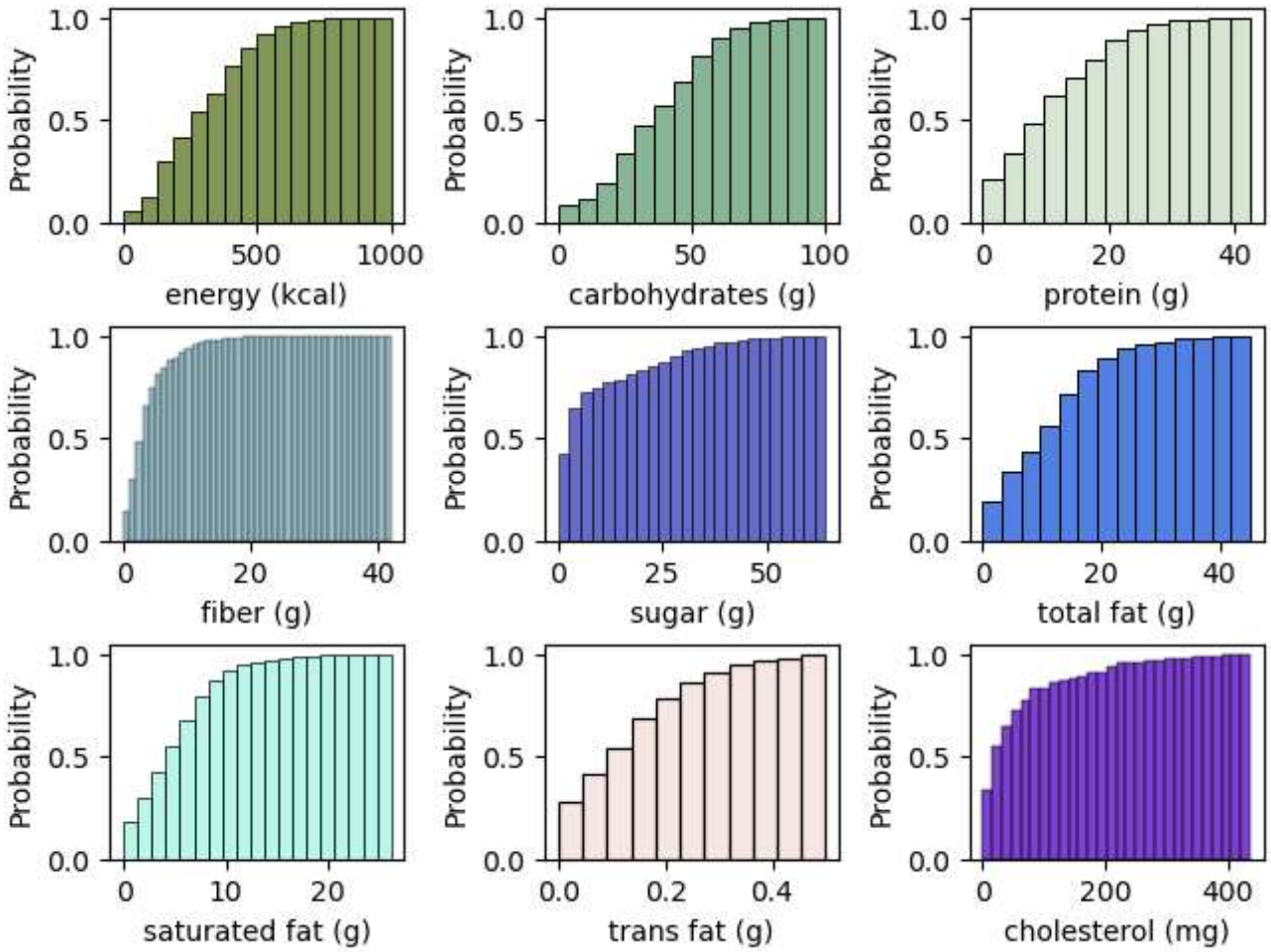
Now we'll drop outliers based on the empirical cumulative and see correlations beween features.

In [126…
```
filters = (num_df['energy (kcal)']<=1000) & (num_df['protein (g)']<=50) & (num_df['trans fat (g)']<=0.5) & (num_df['cho
num_df_out = num_df[filters]
```

In [127…
```
_, axes = plt.subplots(3, 3, constrained_layout=True)
for col, ax in zip(num_df_out.columns, axes.flatten()):
    sns.histplot(num_df_out[col], kde=True, ax=ax, color=np.random.rand(3))
```

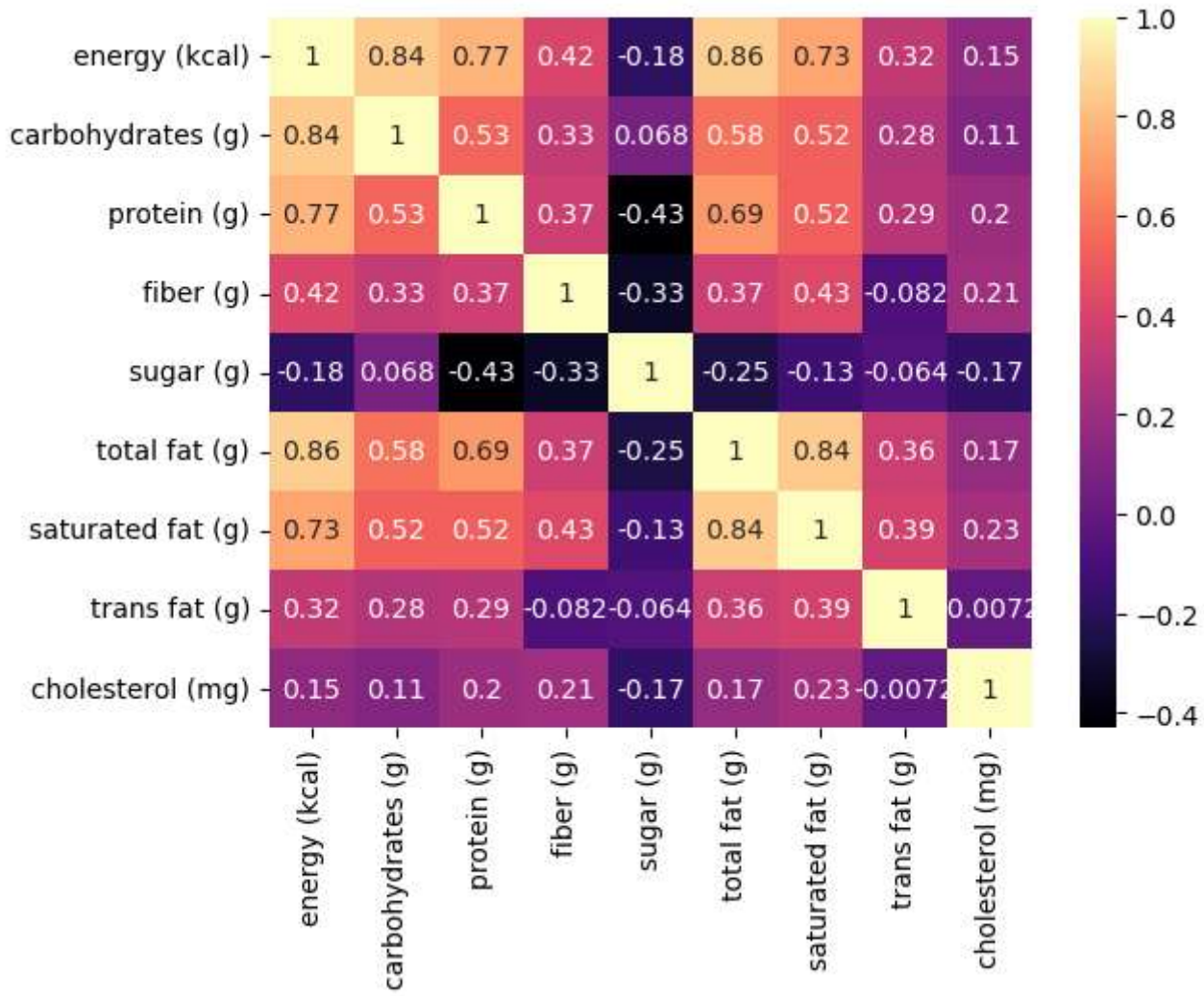
```
plt.show()

_, axes = plt.subplots(3, 3, constrained_layout=True)
for col, ax in zip(num_df_out.columns, axes.flatten()):
    sns.histplot(num_df_out[col], cumulative=True, stat='probability', ax=ax, color=np.random.rand(3))
```





In [128…
```
sns.heatmap(num_df_out.corr(), annot=True, cmap='magma')
num_df_out.shape
```

Out[128]:
```
(480, 9)
```



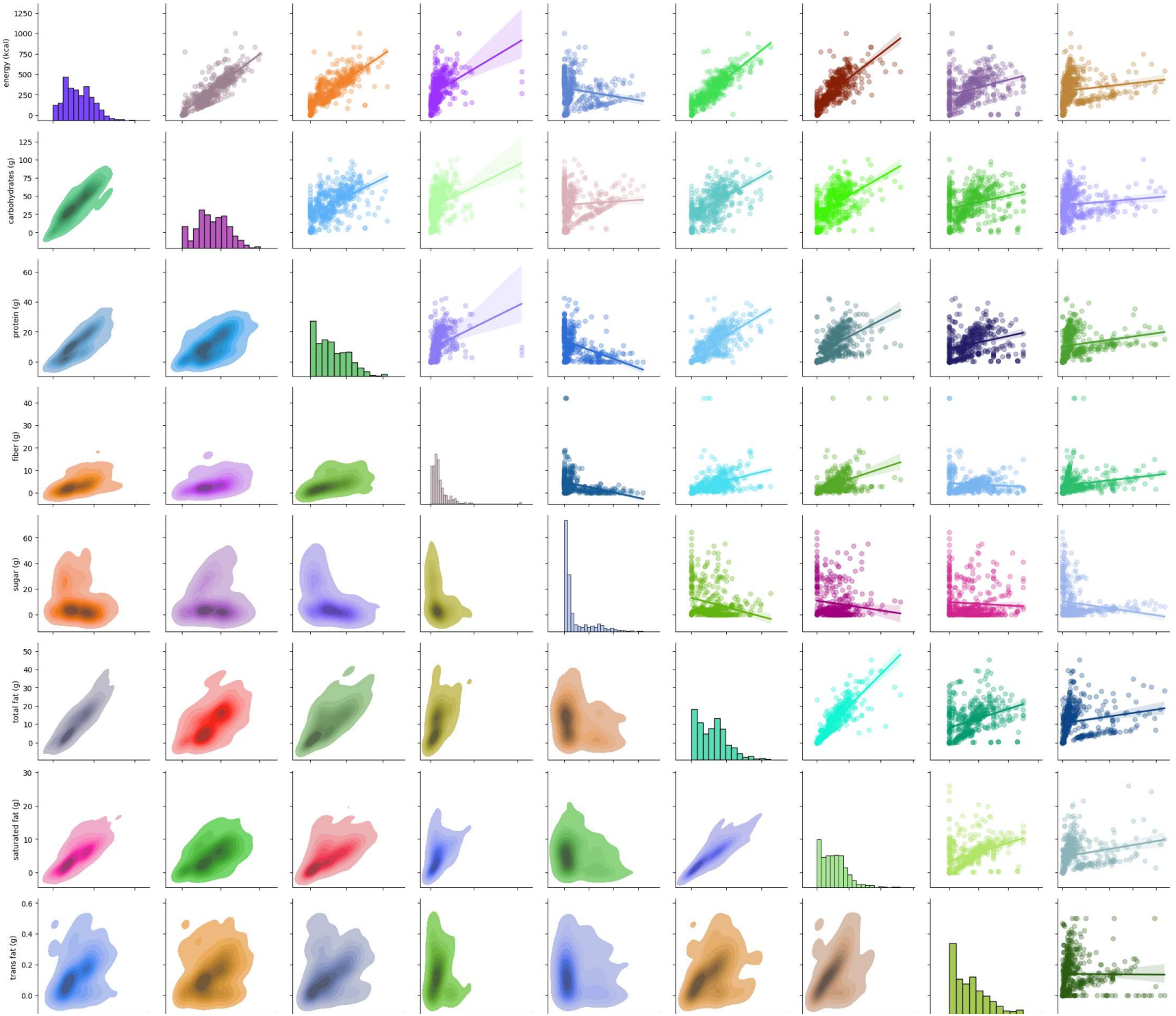
Excluding outliers we dropped 50 rows. \ There are some high correlations beetween features let's visualize these.

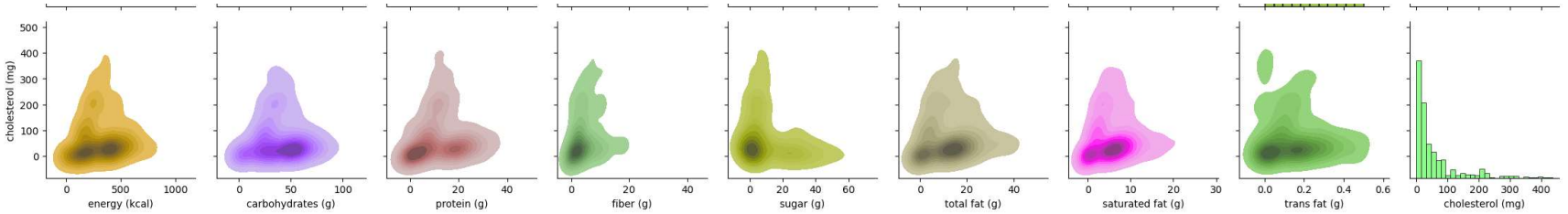
In [129…

```
#Some cool settings
def my_regplot(x, y, **kwargs):
    kwargs['color'] = np.random.rand(3)
    sns.regplot(x=x, y=y, scatter_kws={'alpha':0.3}, **kwargs)

def my_kde(x, y, **kwargs):
    kwargs['color'] = np.random.rand(3)
    sns.kdeplot(x=x, y=y, fill = True,**kwargs)
```

```
def my_hist(x, **kwargs):
    kwargs['color'] = np.random.rand(3)
    sns.histplot(x, **kwargs)

graph = sns.PairGrid(num_df_out)
graph.map_diag(my_hist)
graph.map_upper(my_regplot)
graph.map_lower(my_kde)
```

Out[129]: <seaborn.axisgrid.PairGrid at 0x1e9fb2a21a0>


energy (kcal)
carbohydrates (g)
protein (g)
fiber (g)
sugar (g)
total fat (g)
saturated fat (g)
trans fat (g)



We can notice a correlation of carbohydrates, proteins and fats with energy as one could expect, in particular with fats. \ Fats contribute 9 kcal/g while proteins and carbohydrates about 4-4.5 kcal/g so it's well explained by that. \ Also a correlation between proteins and fats due to fast food meat. \ Last but not least saturated fats and fats.

Categorical features

In [130…

```
cat_cols = data.select_dtypes(exclude=['float']).columns
cat_df = data.loc[:,cat_cols]
cat_df
```

Out[130]:

	company	category	product	per serve size
0	pizza hut	all meals	corn n cheese (personal)	143.5 g
1	pizza hut	all meals	country feast (personal)	178 g
2	pizza hut	all meals	double cheese (personal)	143 g
3	pizza hut	all meals	double paneer supreme (personal)	174.5 g
4	pizza hut	all meals	farmer`s pick (personal)	177 g
...	...	...	...	...
525	dominos	all meals	chicken wings	174.0
526	dominos	all meals	crispy chicken strips	156.0
527	dominos	all meals	zingy parcel chicken	210.0
528	dominos	all meals	taco mexicana - chicken	97.0
529	dominos	all meals	calzone pockets - chicken	248.0

522 rows × 4 columns

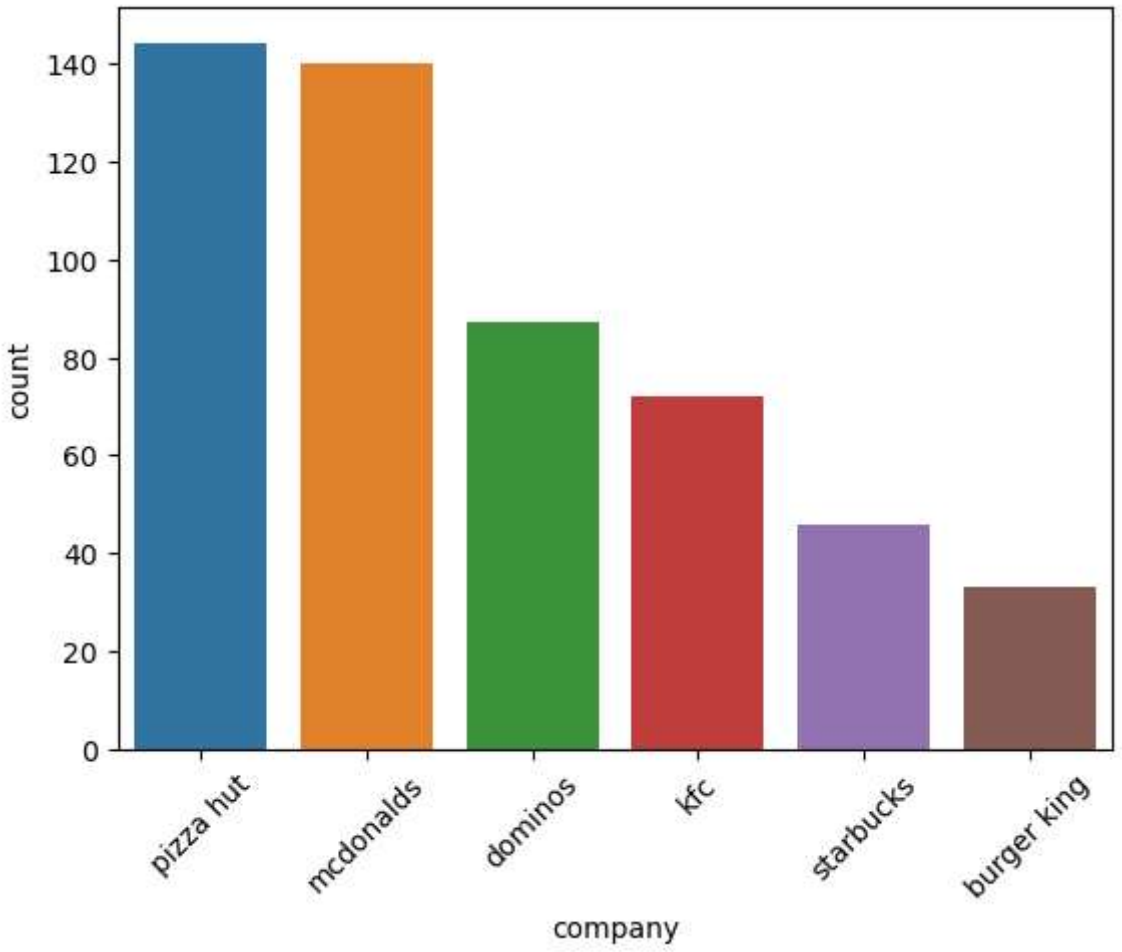
In [131…

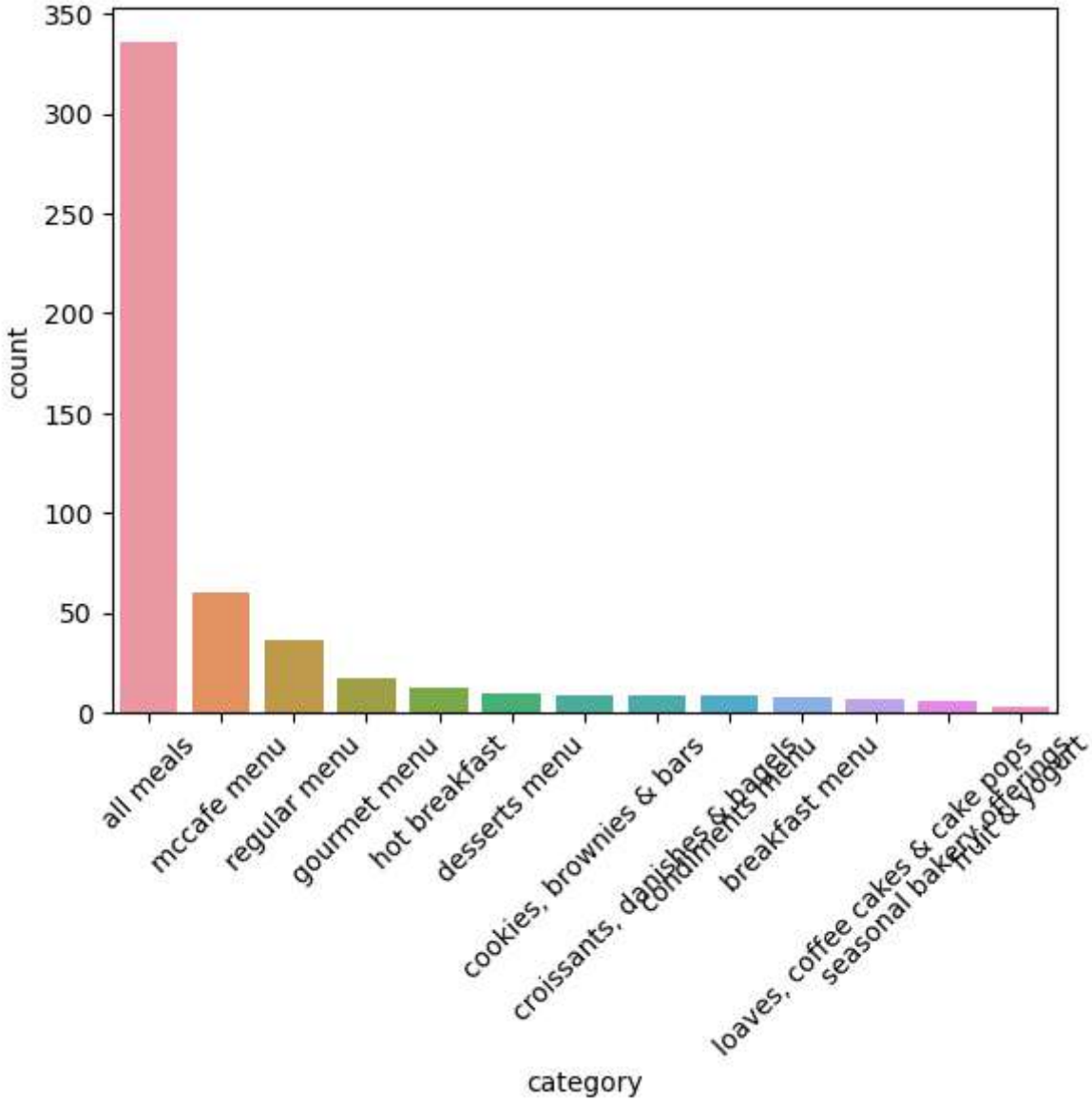
```
for col in cat_df.columns:
    print(cat_df[col].value_counts())
```

```
pizza hut      144
mcdonalds      140
dominos         87
kfc             72
starbucks       46
burger king     33
Name: company, dtype: int64
all meals                           336
mccafe menu                          60
regular menu                         36
gourmet menu                         17
hot breakfast                        12
desserts menu                        10
cookies, brownies & bars              9
croissants, danishes & bagels         9
condiments menu                       9
breakfast menu                        8
loaves, coffee cakes & cake pops      7
seasonal bakery offerings             6
fruit & yogurt                        3
Name: category, dtype: int64
medium fries                     3
corn n cheese (personal)         2
veggie lover (medium)            2
chicken supreme (medium)         2
chicken sausage (medium)         2
                                ..
mc egg burger for happy meal     1
mc egg masala burger             1
green chilli kebab naan          1
chicken kebab burger             1
calzone pockets - chicken        1
Name: product, Length: 456, dtype: int64
330.0     9
250 ml    7
80.0      7
75.0      6
79.0      6
         ..
117       1
40        1
80        1
52        1
248.0     1
Name: per serve size, Length: 330, dtype: int64
```

In [132…

```
for col in cat_df.columns[:-2]:
    sns.countplot(data=cat_df, x=col, order=cat_df[col].value_counts().index)
    plt.xticks(rotation=45)
    plt.show()
```

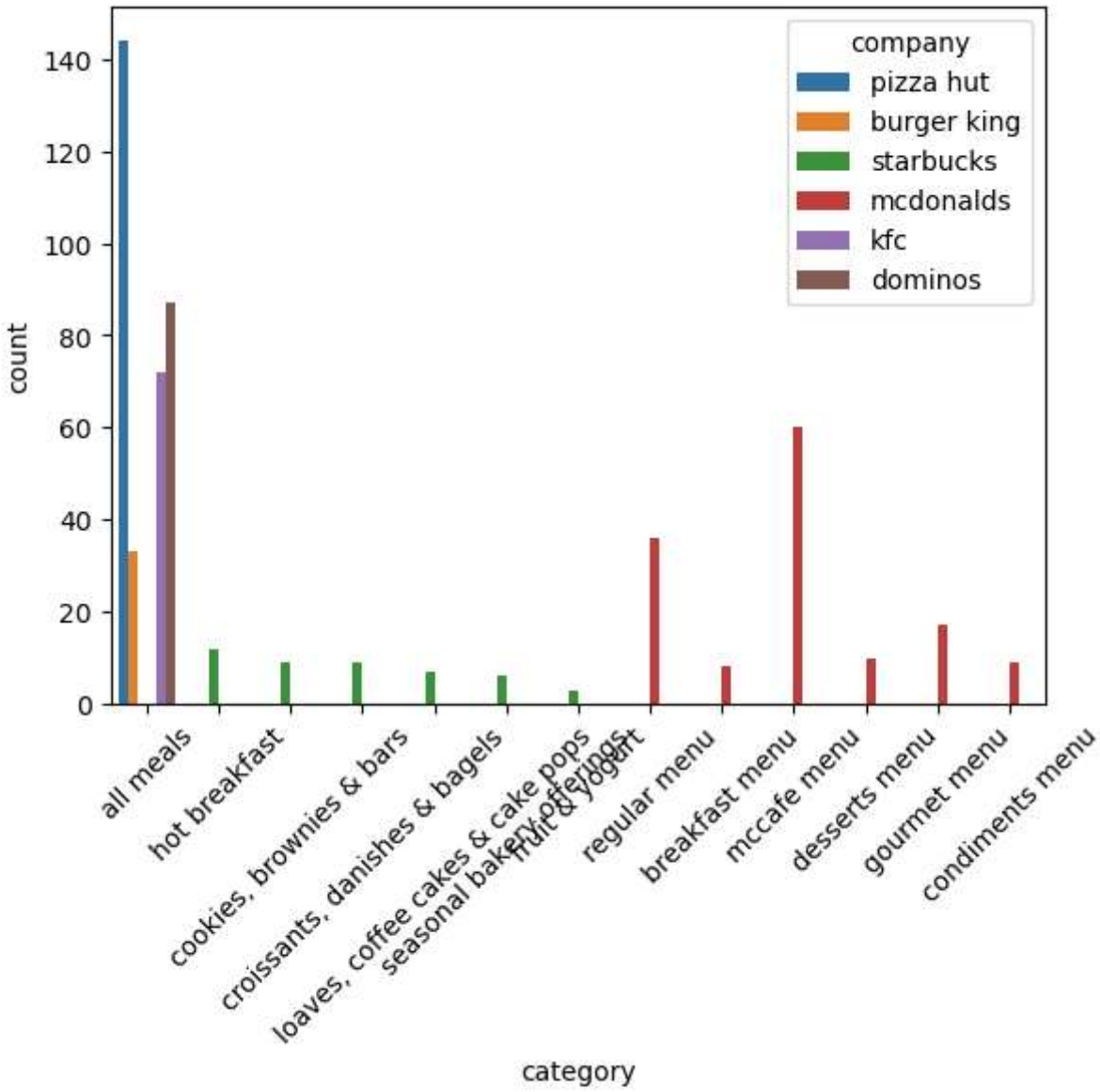




In [133…

```
sns.countplot(data=cat_df, x='category', hue='company')
plt.xticks(rotation=45)
```

Out[133]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]),
 [Text(0, 0, 'all meals'),
  Text(1, 0, 'hot breakfast'),
  Text(2, 0, 'cookies, brownies & bars'),
  Text(3, 0, 'croissants, danishes & bagels'),
  Text(4, 0, 'loaves, coffee cakes & cake pops'),
  Text(5, 0, 'seasonal bakery offerings'),
  Text(6, 0, 'fruit & yogurt'),
  Text(7, 0, 'regular menu'),
  Text(8, 0, 'breakfast menu'),
  Text(9, 0, 'mccafe menu'),
  Text(10, 0, 'desserts menu'),
  Text(11, 0, 'gourmet menu'),
  Text(12, 0, 'condiments menu')])
```



We can note that starbucks and mcdonald's have their own categories.

Now let's try to put all together and try to have a better insight. \ As first we need to apply the same filters applied on the numerical part to all dataframe.

In [134…
```
df = data[filters].copy()
df
```

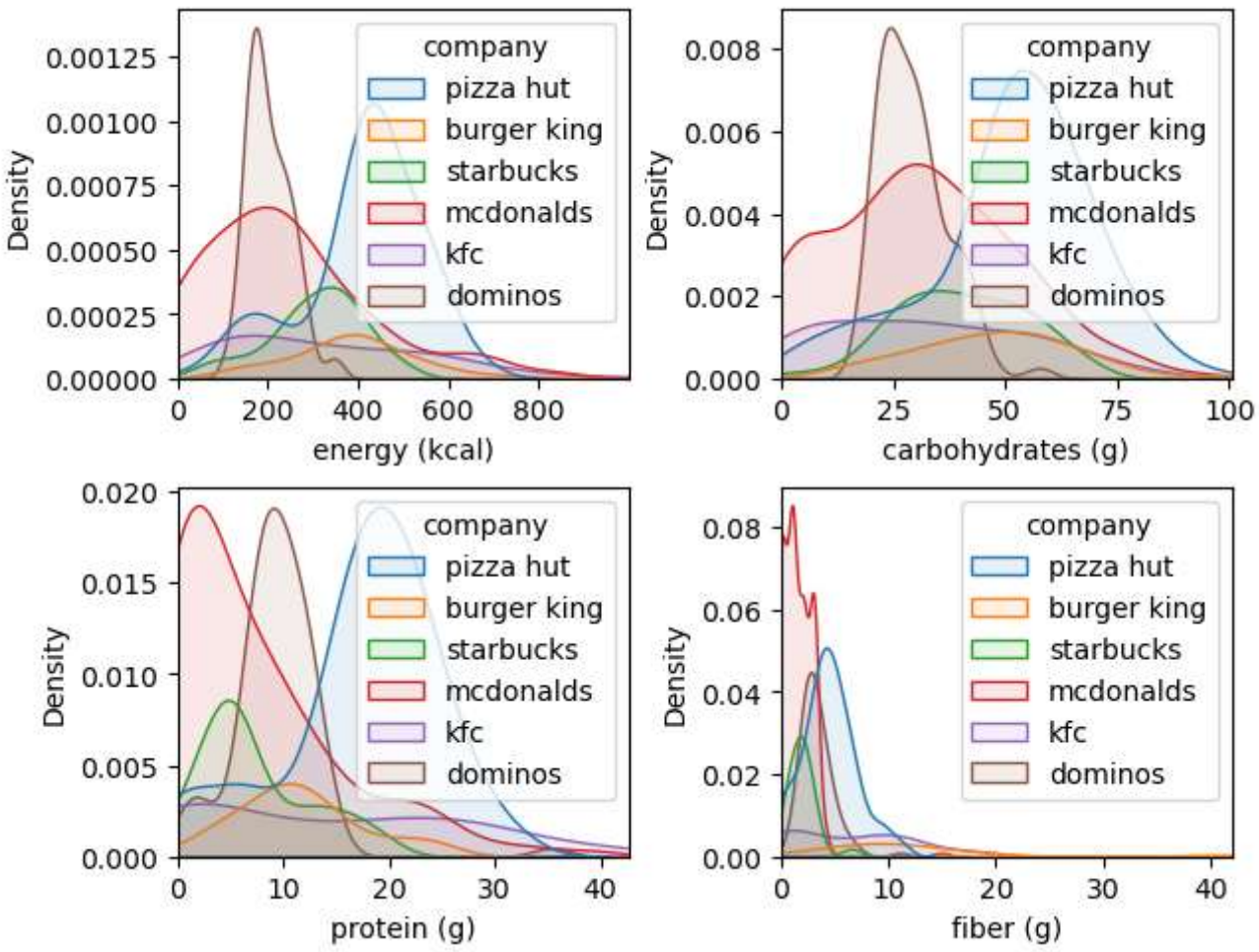
Out[134]:

	company	category	product	per serve size	energy (kcal)	carbohydrates (g)	protein (g)	fiber (g)	sugar (g)	total fat (g)	saturated fat (g)	trans fat (g)	cholesterol (mg)
0	pizza hut	all meals	corn n cheese (personal)	143.5 g	432.60	65.64	17.91	3.85	0.0	10.93	5.14	0.16	16.19
1	pizza hut	all meals	country feast (personal)	178 g	407.60	67.11	16.73	7.19	0.0	8.03	3.24	0.11	66.80
2	pizza hut	all meals	double cheese (personal)	143 g	423.33	59.97	18.26	3.49	0.0	12.27	5.23	0.18	19.75
3	pizza hut	all meals	double paneer supreme (personal)	174.5 g	474.03	52.86	20.07	3.79	0.0	20.26	9.25	0.33	71.72
4	pizza hut	all meals	farmer`s pick (personal)	177 g	408.16	53.93	19.91	2.46	0.0	12.53	4.90	0.14	48.09
…	…	…	…	…	…	…	…	…	…	…	…	…	…
515	dominos	all meals	cheese & pepperoni regular	68.0	189.00	23.10	9.20	4.56	3.2	6.70	3.80	0.15	154.77
516	dominos	all meals	cheese & pepperoni medium	75.0	210.00	25.60	10.20	5.16	3.6	7.40	4.30	0.17	189.44
517	dominos	all meals	cheese & pepperoni large	97.0	272.30	33.20	13.20	6.60	4.1	9.60	5.50	0.22	318.72
518	dominos	all meals	garlic breadsticks	122.0	340.30	57.60	11.90	1.68	8.5	6.90	1.40	0.06	397.44
525	dominos	all meals	chicken wings	174.0	352.90	24.50	35.30	5.76	10.4	12.60	4.80	0.19	308.70

480 rows × 13 columns

In [135…
```
_,  axes = plt.subplots(2, 2, constrained_layout=True)
for col, ax in zip(num_df.columns, axes.flatten()):
```

```
    sns.kdeplot(data=df, x=col, ax=ax, hue='company', fill=True, alpha=0.1)
    ax.set_xlim(left=0, right=df[col].max())
```



From the graph above we can say that in average pizza hut meals have more calories. \ This becomes particularly evident noticing that the distribution of total fats has the highest mean above all companies.

Except for Mc an Starbucks we don't know the categories of other companies meals, but by common knowledge there is a marked differnce in the products sold by Starbucks.

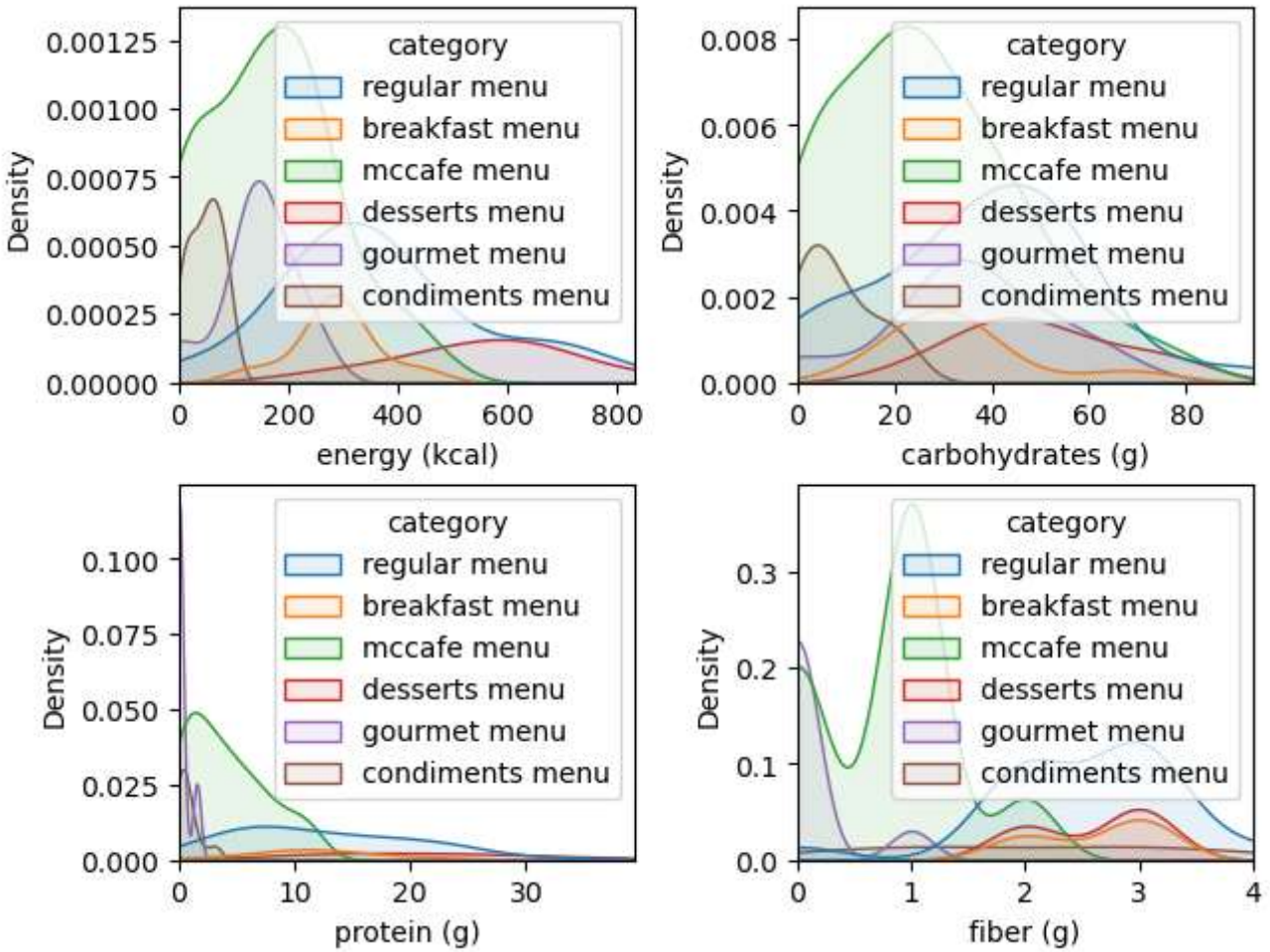
Let's have a look at meals from mc

In [136…

```
mc = df[(df.company == 'mcdonalds')]
```

In [137…

```
_, axes = plt.subplots(2, 2, constrained_layout=True)
for col, ax in zip(num_df.columns, axes.flatten()):
    sns.kdeplot(data=mc, x=col, ax=ax, hue='category', fill=True, alpha=0.1)
    ax.set_xlim(left=0, right=mc[col].max())
```



Desserts menu are the highest in kcals followed by regular menu, also mcmenus are fattier than breakfasts and mccafe menu. \
Unforunately we have too few data to extract other informations on every category.

What we can try to do now is to predict given only nutritional values what would be the company. \ Considering category would be like cheating because for starbucks and mcdonald we would have a perfect match. \ Also we must take into account that

Modeling

In [138…

```
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import GridSearchCV, cross_val_score, train_test_split, learning_curve, StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

In [139…

```
def plot_learning_curve(
    estimator,
    title,
    X,
    y,
    axes=None,
    ylim=None,
    cv=None,
    n_jobs=None,
    train_sizes=np.linspace(0.1, 1.0, 5),
    scoring='neg_mean_absolute_percentage_error'
):
    """
    Generate 3 plots: the test and training learning curve, the training
    samples vs fit times curve, the fit times vs score curve.

    Parameters
    ----------
    estimator : estimator instance
        An estimator instance implementing `fit` and `predict` methods which
        will be cloned for each validation.

    title : str
        Title for the chart.

    X : array-like of shape (n_samples, n_features)
        Training vector, where ``n_samples`` is the number of samples and
        ``n_features`` is the number of features.

    y : array-like of shape (n_samples) or (n_samples, n_features)
        Target relative to ``X`` for classification or regression;
        None for unsupervised learning.
```

```
    axes : array-like of shape (3,), default=None
        Axes to use for plotting the curves.

    ylim : tuple of shape (2,), default=None
        Defines minimum and maximum y-values plotted, e.g. (ymin, ymax).

    cv : int, cross-validation generator or an iterable, default=None
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:

          - None, to use the default 5-fold cross-validation,
          - integer, to specify the number of folds.
          - :term:`CV splitter`,
          - An iterable yielding (train, test) splits as arrays of indices.

        For integer/None inputs, if ``y`` is binary or multiclass,
        :class:`StratifiedKFold` used. If the estimator is not a classifier
        or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validators that can be used here.

    n_jobs : int or None, default=None
        Number of jobs to run in parallel.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.

    train_sizes : array-like of shape (n_ticks,)
        Relative or absolute numbers of training examples that will be used to
        generate the learning curve. If the ``dtype`` is float, it is regarded
        as a fraction of the maximum size of the training set (that is
        determined by the selected validation method), i.e. it has to be within
        (0, 1]. Otherwise it is interpreted as absolute sizes of the training
        sets. Note that for classification the number of samples usually have
        to be big enough to contain at least one sample from each class.
        (default: np.linspace(0.1, 1.0, 5))
    """
    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
```

```
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = learning_curve(
        estimator,
        X,
        y,
        cv=cv,
        n_jobs=n_jobs,
        train_sizes=train_sizes,
        return_times=True,
        scoring=scoring
    )
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(
        train_sizes,
        train_scores_mean - train_scores_std,
        train_scores_mean + train_scores_std,
        alpha=0.1,
        color="r",
    )
    axes[0].fill_between(
        train_sizes,
        test_scores_mean - test_scores_std,
        test_scores_mean + test_scores_std,
        alpha=0.1,
        color="g",
    )
    axes[0].plot(
        train_sizes, train_scores_mean, "o-", color="r", label="Training score"
    )
    axes[0].plot(
        train_sizes, test_scores_mean, "o-", color="g", label="Cross-validation score"
    )
    axes[0].legend(loc="best")

    # Plot n_samples vs fit_times
    axes[1].grid()
```

```
    axes[1].plot(train_sizes, fit_times_mean, "o-")
    axes[1].fill_between(
        train_sizes,
        fit_times_mean - fit_times_std,
        fit_times_mean + fit_times_std,
        alpha=0.1,
    )
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    fit_time_argsort = fit_times_mean.argsort()
    fit_time_sorted = fit_times_mean[fit_time_argsort]
    test_scores_mean_sorted = test_scores_mean[fit_time_argsort]
    test_scores_std_sorted = test_scores_std[fit_time_argsort]
    axes[2].grid()
    axes[2].plot(fit_time_sorted, test_scores_mean_sorted, "o-")
    axes[2].fill_between(
        fit_time_sorted,
        test_scores_mean_sorted - test_scores_std_sorted,
        test_scores_mean_sorted + test_scores_std_sorted,
        alpha=0.1,
    )
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")

    return plt
```

Data don't need much preprocessing, let's scale numerical columns, drop categoricals and at last encode company.

In [140…

```
from sklearn.preprocessing import StandardScaler

le = LabelEncoder()
X = df.iloc[:,1:]
y = le.fit_transform(df.iloc[:,0])

X_train, X_test, y_train, y_test = train_test_split(X, y)

preproc = ColumnTransformer(transformers=[
    ('dropper', 'drop', [0,1,2]),
    ('scaler', StandardScaler(), numeric_cols)],
    remainder='passthrough')
```

```
skf = StratifiedKFold(n_splits=10)
```

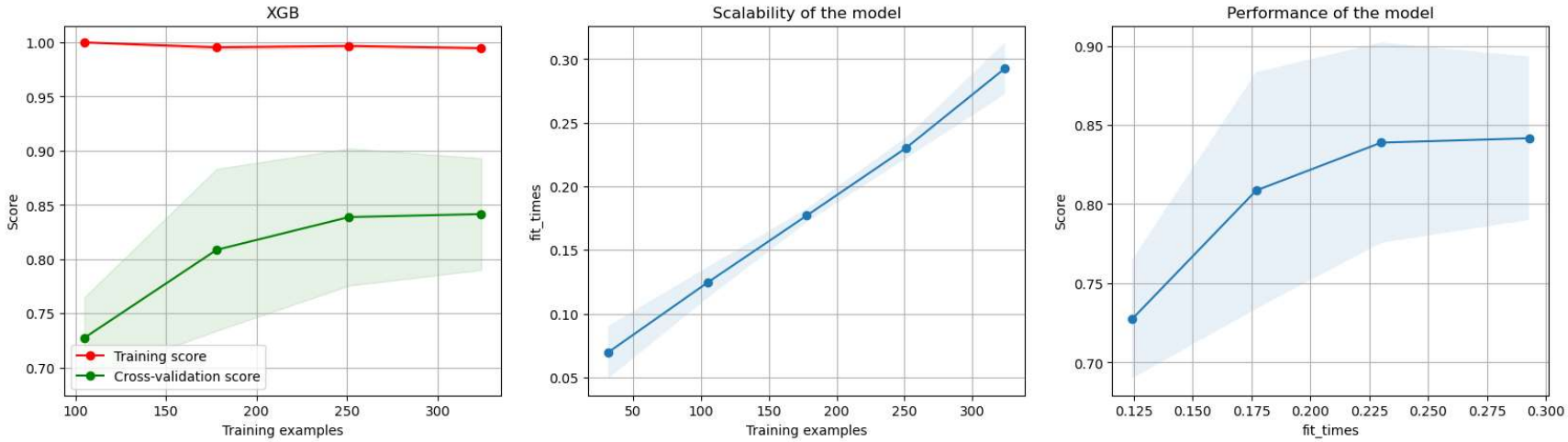
XGBoost

In [141…
```
pipe = Pipeline(steps=[('preprocessing', preproc),
                       ('model',XGBClassifier())])

print(cross_val_score(pipe, X_train, y_train, cv=skf, scoring='f1_weighted').mean())
```

```
0.8415817467133258
```

In [142…
```
plot_learning_curve(pipe, 'XGB', X_train, y_train, cv=skf, n_jobs=-1, scoring='f1_weighted')
```

Out[142]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\Dell\\anaconda3\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



The model is overfitting badly, we will see if we can reduce it.

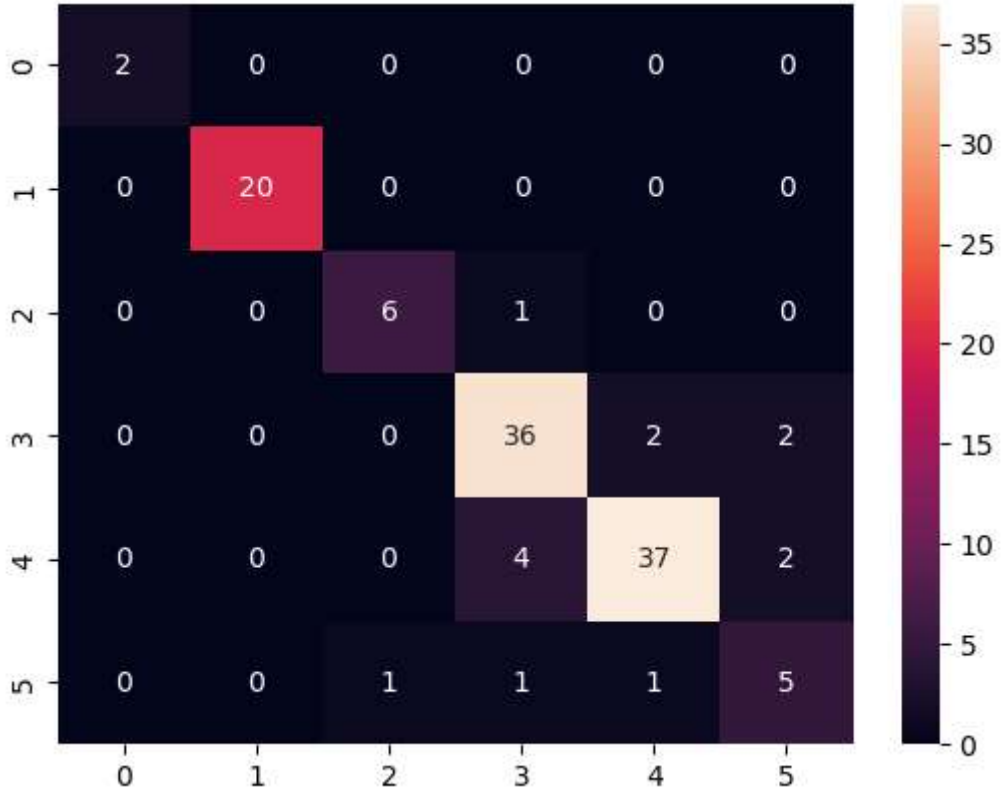
In [143…
```
pipe.fit(X_train, y_train)
pred = pipe.predict(X_test)
print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2
1	1.00	1.00	1.00	20
2	0.86	0.86	0.86	7
3	0.86	0.90	0.88	40
4	0.93	0.86	0.89	43
5	0.56	0.62	0.59	8
accuracy			0.88	120
macro avg	0.87	0.87	0.87	120
weighted avg	0.89	0.88	0.88	120

The recall on the target 5 is really low, but aside from that it looks promising. \ Let's see if we can achieve better with a bit of tuning

In [144…
```
cm = confusion_matrix(y_test, pred)
sns.heatmap(cm, annot=True)
```

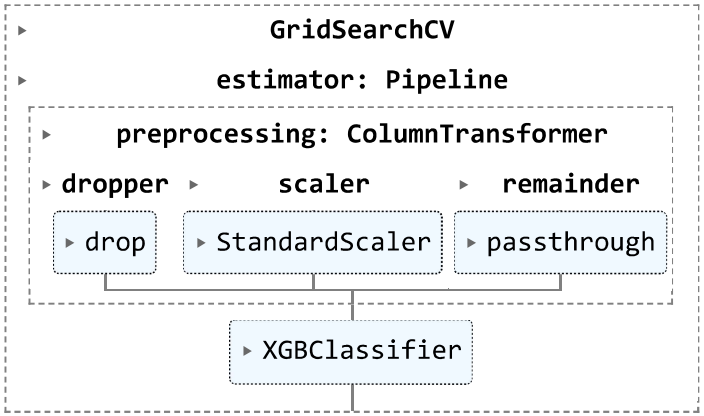
Out[144]:
```
<Axes: >
```



In [145…

```
xgb_params = {'model__max_depth': [i for i in range(2,4)],
              'model__learning_rate': [0.001, 0.005, 0.01],
              'model__n_estimators': [i for i in range(50, 101, 10)]
             }

xgb_grid = GridSearchCV(pipe,
                        param_grid=xgb_params,
                        cv = skf,
                        n_jobs=-1,
                        scoring='f1_weighted')

xgb_grid.fit(X_train, y_train)
```

Out[145]:



In [146…

```
print(xgb_grid.best_score_)
print(xgb_grid.best_params_)
```

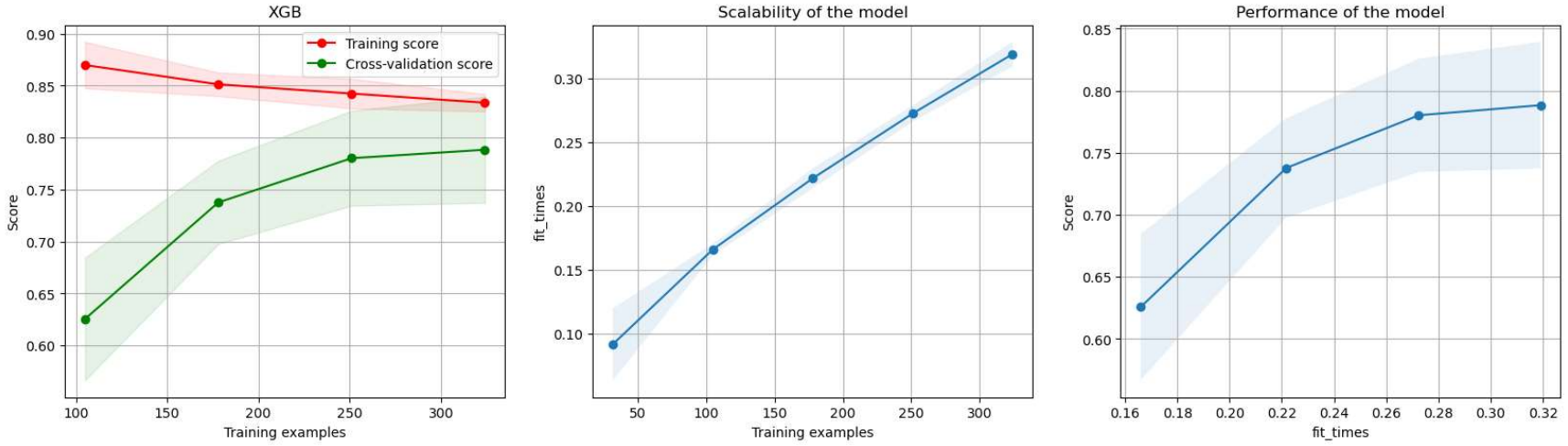
```
0.7882961948165242
{'model__learning_rate': 0.01, 'model__max_depth': 2, 'model__n_estimators': 100}
```

In [147…

```
plot_learning_curve(xgb_grid.best_estimator_, 'XGB', X_train, y_train, cv=skf, n_jobs=-1, scoring='f1_weighted')
```

Out[147]:

```
<module 'matplotlib.pyplot' from 'C:\\Users\\Dell\\anaconda3\\lib\\site-packages\\matplotlib\\pyplot.py'>
```

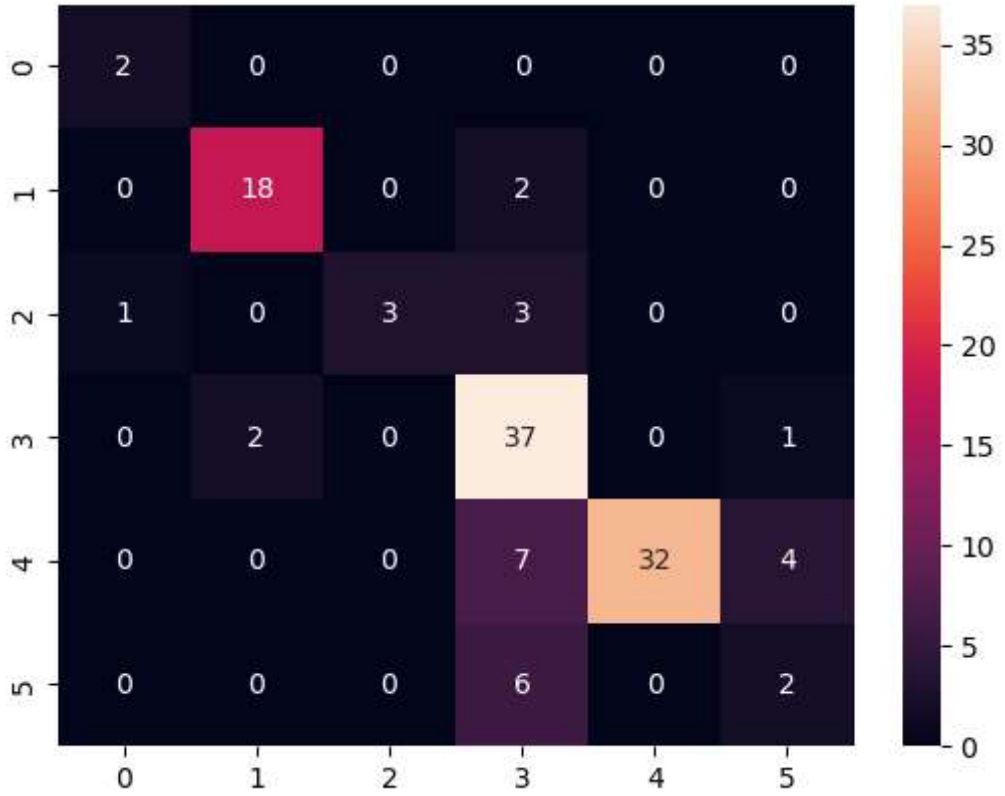


In [148…

```
pred = xgb_grid.predict(X_test)
print(classification_report(y_test, pred))
cm = confusion_matrix(y_test, pred)
sns.heatmap(cm, annot=True)
```

	precision	recall	f1-score	support
0	0.67	1.00	0.80	2
1	0.90	0.90	0.90	20
2	1.00	0.43	0.60	7
3	0.67	0.93	0.78	40
4	1.00	0.74	0.85	43
5	0.29	0.25	0.27	8
accuracy			0.78	120
macro avg	0.75	0.71	0.70	120
weighted avg	0.82	0.78	0.78	120

Out[148]: <Axes: >



We reduced overfitting and we achieved the same results, let's try a simpler model and see if it's better.

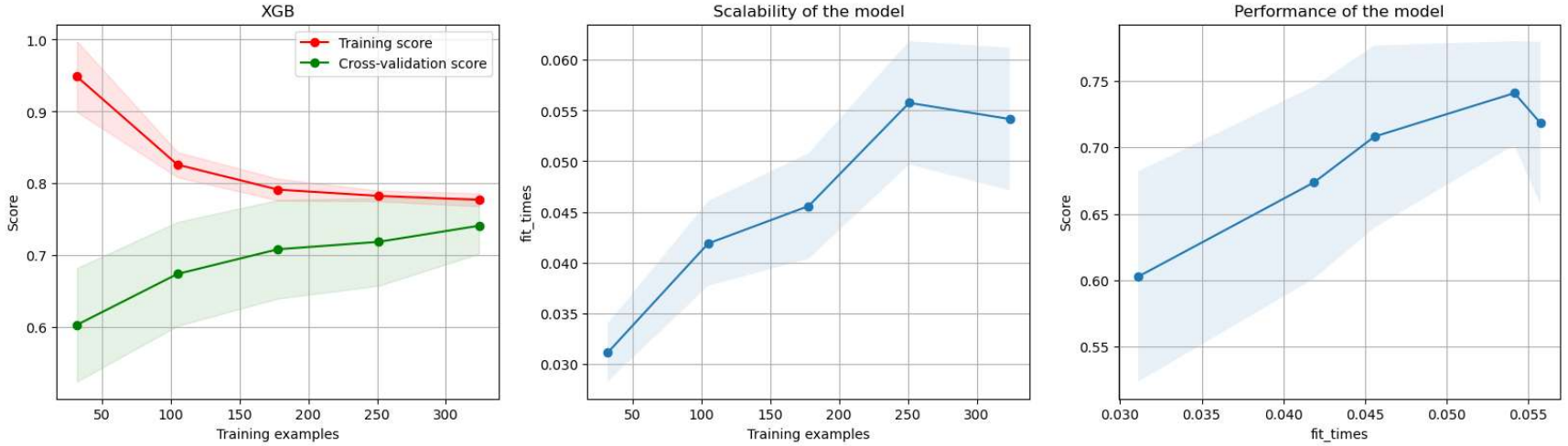
In [149…
```
from sklearn.linear_model import LogisticRegression
```

In [150…
```
pipe_lr = Pipeline(steps=[('preprocessing', preproc),
                          ('model',LogisticRegression())])

pipe_lr.fit(X_train, y_train)

print(cross_val_score(pipe_lr, X_train, y_train, cv=skf, scoring='f1_weighted').mean())
```

```
0.7407886508102439
```

In [151…
```
plot_learning_curve(pipe_lr, 'XGB', X_train, y_train, cv=skf, n_jobs=-1, scoring='f1_weighted')
```

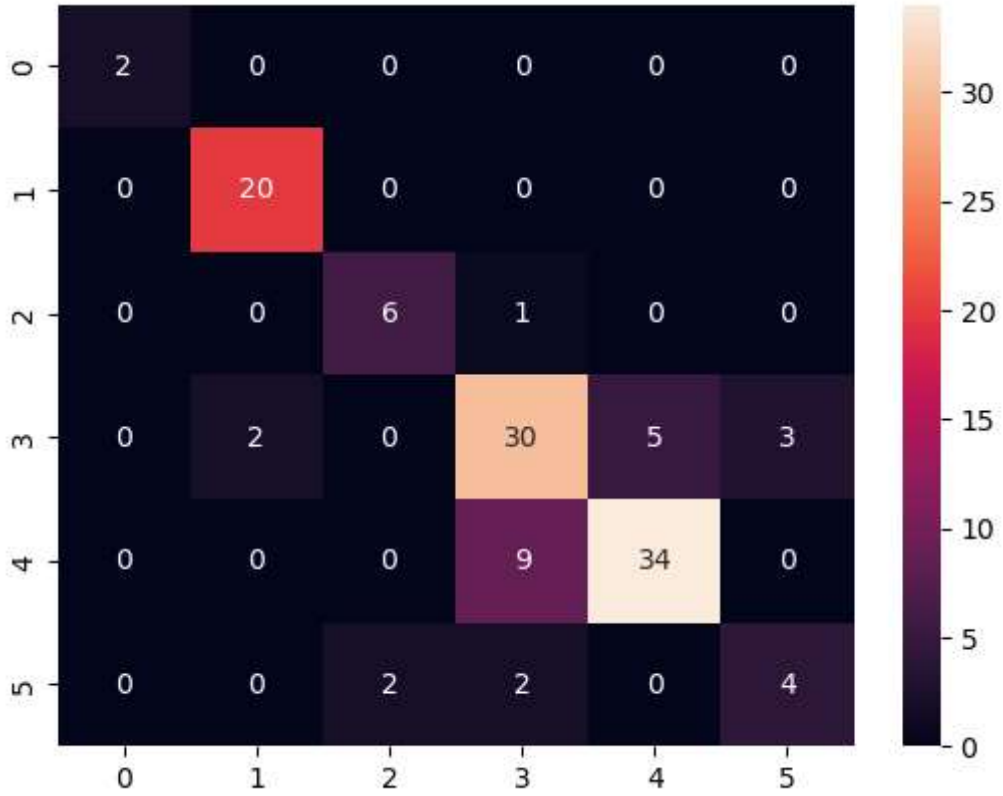
Out[151]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\Dell\\anaconda3\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



In [152…
```
pred = pipe_lr.predict(X_test)
print(classification_report(y_test, pred))
cm = confusion_matrix(y_test, pred)
sns.heatmap(cm, annot=True)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2
1	0.91	1.00	0.95	20
2	0.75	0.86	0.80	7
3	0.71	0.75	0.73	40
4	0.87	0.79	0.83	43
5	0.57	0.50	0.53	8
accuracy			0.80	120
macro avg	0.80	0.82	0.81	120
weighted avg	0.80	0.80	0.80	120

Out[152]: <Axes: >



Not really a good score and the model is underfitting. \ Let's try a bit of tuning

In [153…

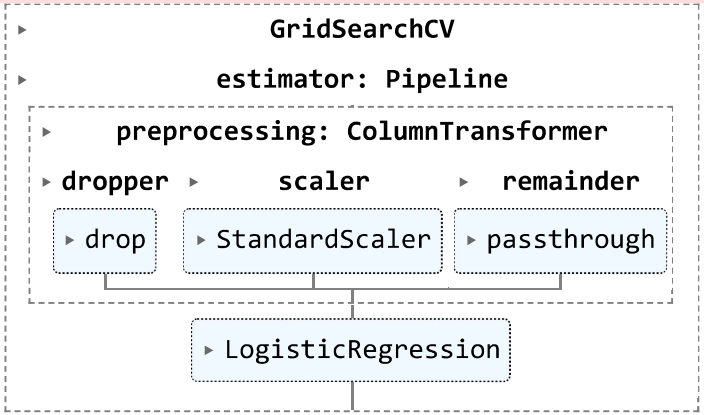
```python
# Define lr_params with the parameter values you want to search
lr_params = {
    'model__C': [0.1, 1, 2, 5, 10, 100, 500, 1000],
    'model__solver': ['lbfgs', 'liblinear', 'newton-cg', 'sag', 'saga']
}

# Rest of your code
```

In [154…

```python
r_params = {'model__C': [0.1, 1, 2 ,5, 10, 100, 500, 1000],
              'model__solver': ['lbfgs', 'liblinear', 'newton-cg', 'sag', 'saga']
             }

lr_grid = GridSearchCV(pipe_lr,
                       param_grid=lr_params,
                       cv=skf,
                       n_jobs=-1,
                       scoring='f1_weighted')

lr_grid.fit(X_train, y_train)
```

```
C:\Users\Dell\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to co
nverge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[154]:



In [155…

```
print(lr_grid.best_score_)
print(lr_grid.best_params_)
```

```
0.7661214701295243
{'model__C': 500, 'model__solver': 'lbfgs'}
```

In [ ]: