eda

August 6, 2022

1 EDA

[ ]:
```
# Import stuff
# !pip install pandas numpy seaborn sklearn nltk missingno tensorflow dask␣
 ↪hvplot
# !pip install xgboost --upgrade
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import missingno as msno
from sklearn import model_selection, preprocessing, ensemble, neighbors,␣
 ↪linear_model, svm, metrics
from nltk.util import ngrams
from imblearn import over_sampling, under_sampling, pipeline
from PIL import Image
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, SimpleImputer
from sklearn import metrics
from xgboost import XGBClassifier, XGBRegressor # 1.6.1
from sklearn.pipeline import Pipeline
import itertools
from sklearn.utils import class_weight
import plotly.io as pio
pio.renderers.default = "notebook+pdf"
```

First we'll define some useful functions for later:

[ ]:
```
# Define useful functions

"""
Returns dictionary of textual column values and their frequency in dataframe
"""
def get_column_values(df, exceptions=[]):
```

```
    n = len(df)
    text_columns = df.select_dtypes(exclude=[np.number])
    column_values = {}
    for column in text_columns:
        # Make frequencies dictionary and calculated percent of missing values
        if column not in exceptions:
            column_values[column] = df[column].value_counts().to_frame().
 ↪reset_index()
            column_values[column].columns = ['value', 'count']
            column_values[column]['pc'] = column_values[column]['count'] / n *␣
 ↪100
    return column_values


"""
Cleans text values in dataframe for EDA purposes:
0. Replaces spaces with underscore
1. Removes non-ASCII characters
2. Transforms to lowercase
3. Fills null values with "NA"
4. Removes punctuation characters (except underscore and .)
"""
from string import punctuation
def clean_text_values(df):
    # punc = [p for p in punctuation if p not in ['_']]
    column_values = get_column_values(df)
    for column in column_values:
        df[column].str.encode('ascii', 'ignore').str.decode('ascii')
        df[column] = df[column].str.lower()
        df[column].fillna('NA', inplace=True)
        if column != 'category':
            for p in punctuation:
                df[column] = df[column].str.replace(p, '', regex=False)
        df[column] = df[column].str.replace(r'\s+', ' ', regex=True)
    return df


"""
Recieves df, categories and column name as arguments and returns a Histogram␣
 ↪object of that column divided by the different categories, without
zeros and values above the 99th percentile to clear extreme values which mess␣
 ↪up the histogram (calculated after removing zeros).
"""
def create_column_histogram_by_category(df, categories, column):
    all_data = []
    num_99_percentile = 0
    for category in categories:
```

```
        data_no_zeros = df[(df['category'] == category) & (df[column] > 0)]
        data = data_no_zeros[(data_no_zeros[column] <= data_no_zeros[column].
 ↪quantile(0.99))]
        num_99_percentile = len(data_no_zeros) - len(data)
        all_data.append(go.Histogram(
            x=data[column],
            name=category,
            opacity=0.7
            # ,xbins={
            #     'start': 0,
            #     'end': data.max(),
            #     'size': data.max() / 500
            # }
            )
        )
    num_zeros = len(df[df[column] == 0])
    layout = go.Layout(
        barmode='overlay',
        title='{} histograms divided by class  |  No. of zeros: {}  |  No.␣
 ↪above 99th percentile: {}\n| Percent "bad" rows: {}%'.format(
            column.capitalize().replace('_', ' '), num_zeros,␣
 ↪num_99_percentile, round((num_zeros + num_99_percentile) / len(df) * 100, 3))
    )
    fig = go.Figure(data=all_data, layout=layout)
    return fig


def add_ngrams_columns(df, column, n, return_column=False):
    new_ngrams_column = '{}_{}gram'.format(column, n)
    df[new_ngrams_column] = df[df[column].notna()][column].apply(lambda row:␣
 ↪list(ngrams(row.split(' '), n)))
    if return_column:
        return df, new_ngrams_column
    return df


def get_ngrams(df, column, n, remove_column=False):
    df, new_ngrams_column = add_ngrams_columns(df, column, n,␣
 ↪return_column=True)
    categories = list(df['category'].unique())
    n_grams = {cat: {} for cat in categories}
    for category in categories:
        cat_df = df[df['category'] == category]
        for ngram_list in cat_df[cat_df[new_ngrams_column].
 ↪notna()][new_ngrams_column]:
            if type(ngram_list) == float:
```

```
                print(ngram_list)
            for n_gram in ngram_list:
                if n_gram in n_grams[category].keys():
                    n_grams[category][n_gram] += 1
                else:
                    n_grams[category][n_gram] = 1
    if remove_column:
        df = df.drop([new_ngrams_column], axis=1)
    return df, n_grams


def get_ngrams_top_k(df, column, n, k=1, remove_column=False):
    df, column_ngrams = get_ngrams(df, column, n, remove_column)
    categories = list(df['category'].unique())
    top_20_column_ngrams_by_category = {}
    for category in categories:
        top_column_ngrams = sorted(column_ngrams[category],␣
 ↪key=column_ngrams[category].get, reverse=True)[:k]
        top_20_column_ngrams = [(" ".join(key), column_ngrams[category][key])␣
 ↪for key in column_ngrams[category].keys() if key in top_column_ngrams]
        top_20_column_ngrams = pd.DataFrame(top_20_column_ngrams,␣
 ↪columns=['ngram', 'count']).sort_values('count')
        top_20_column_ngrams_by_category[category] = top_20_column_ngrams
    return df, top_20_column_ngrams_by_category


def get_dims(file):
    img = Image.open(file)
    try:
        h, w, d = np.array(img).shape
    except:
        h, w = np.array(img).shape
    img.close()
    return h, w
```

Let us load the data:

[ ]:
```
dataset = pd.read_csv('data/food_train.csv')
fact_nutrients = pd.read_csv('data/food_nutrients.csv')
dim_nutrients = pd.read_csv('data/nutrients.csv')
```

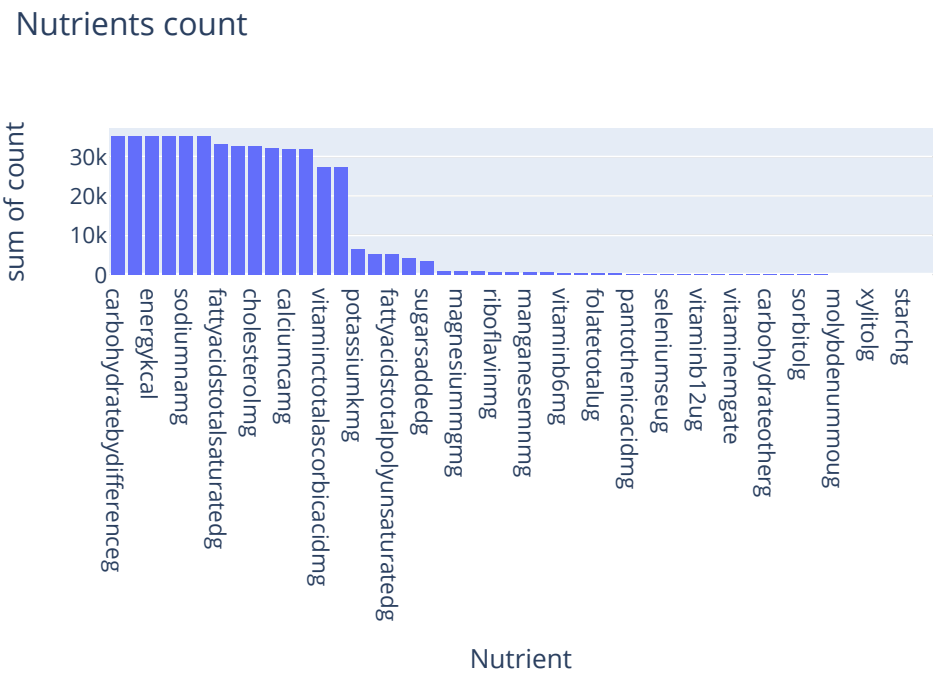
Then merge and clean it a bit:

[ ]:
```
# Join nutrients tables and clean it
nutrients = pd.merge(fact_nutrients, dim_nutrients, how='left',␣
 ↪on='nutrient_id')
nutrients = clean_text_values(nutrients)
```

```
nutrients['name'] = nutrients['name'].str.replace(' ', '_')
nutrients['name'] = nutrients['name'] + '_' + nutrients['unit_name']
nutrients_column_values = get_column_values(nutrients)
```

See how many records we have of each nutrient:

[ ]:
```
# Draw histogram of nutrients using plotly
num_nutrients = nutrients.groupby('name')['idx'].count().reset_index()
num_nutrients.columns = ['nutrient', 'count']
num_nutrients['pc'] = num_nutrients['count'] / len(dataset) * 100
num_nutrients = num_nutrients[num_nutrients['count'] > 0]
num_nutrients = clean_text_values(num_nutrients)
px.histogram(
    num_nutrients.sort_values('count', ascending=False), x='nutrient',␣
 ↪y='count', title='Nutrients count', width=600, height=400,
    labels={'nutrient':'Nutrient'}
)
```



Now let's left join the nutrients to our snacks dataset:

[ ]:
```
# Left join nutrients to dataset
joined_dataset = pd.merge(dataset, nutrients, how='left', on='idx').
 ↪drop(['nutrient_id', 'unit_name'], axis=1)
```

Good ol' pivot table and join:

[ ]:
```
# Pivot table - started by resetting index and rearranging columns but decided␣
 ↪to join again to avoid pandas index trouble
pivoted_dataset = pd.pivot_table(
    joined_dataset,
    values='amount',
    index='idx',
    columns='name')

dataset_w_nutrients = pd.merge(dataset, pivoted_dataset, how='left', on='idx')
```

Change categories names and encode them for convenience:

[ ]:
```
dataset_w_nutrients.loc[dataset_w_nutrients['category'] ==␣
 ↪'cakes_cupcakes_snack_cakes', 'category'] = 'cakes'
dataset_w_nutrients.loc[dataset_w_nutrients['category'] ==␣
 ↪'chips_pretzels_snacks', 'category'] = 'snacks'
dataset_w_nutrients.loc[dataset_w_nutrients['category'] == 'cookies_biscuits',␣
 ↪'category'] = 'cookies'
dataset_w_nutrients.loc[dataset_w_nutrients['category'] ==␣
 ↪'popcorn_peanuts_seeds_related_snacks', 'category'] = 'seeds'
le = preprocessing.LabelEncoder()
le.fit(dataset_w_nutrients['category'])
dataset_w_nutrients['category_enc'] = le.
 ↪transform(dataset_w_nutrients['category'])
le.classes_
```

[ ]:
```
array(['cakes', 'candy', 'chocolate', 'cookies', 'seeds', 'snacks'],
      dtype=object)
```

Now we can split our data for initial analysis!

[ ]:
```
# Split to perform EDA only on train data so conclusions won't later leak to␣
 ↪test set
X = dataset_w_nutrients.drop(['category'], axis=1)
y = dataset_w_nutrients['category']
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,␣
 ↪test_size=0.2, random_state=1337, stratify=y)
eda_df = pd.concat([X_train, y_train], axis=1)
```

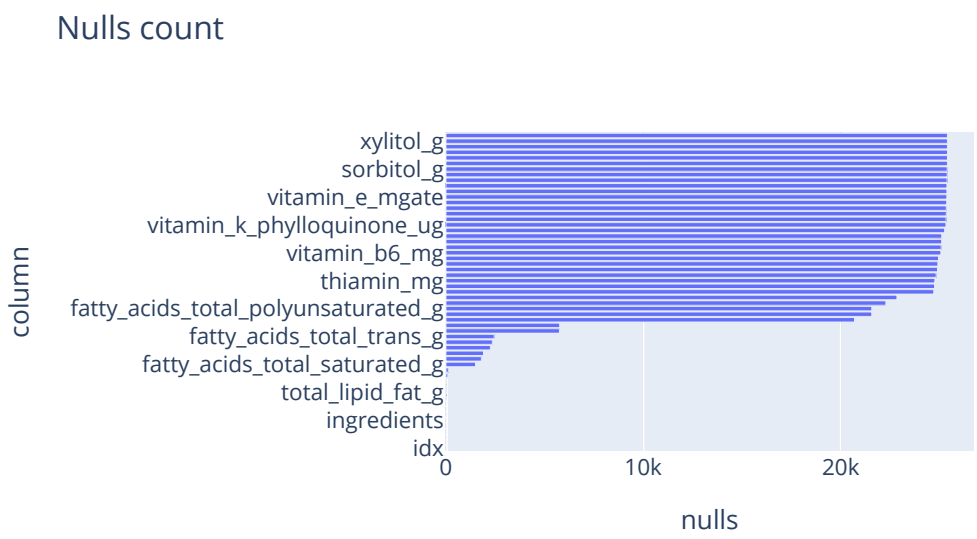
Clean text values:

[ ]:
```
eda_df = clean_text_values(eda_df)
```

Hajime!

First of all, let's check for nulls:

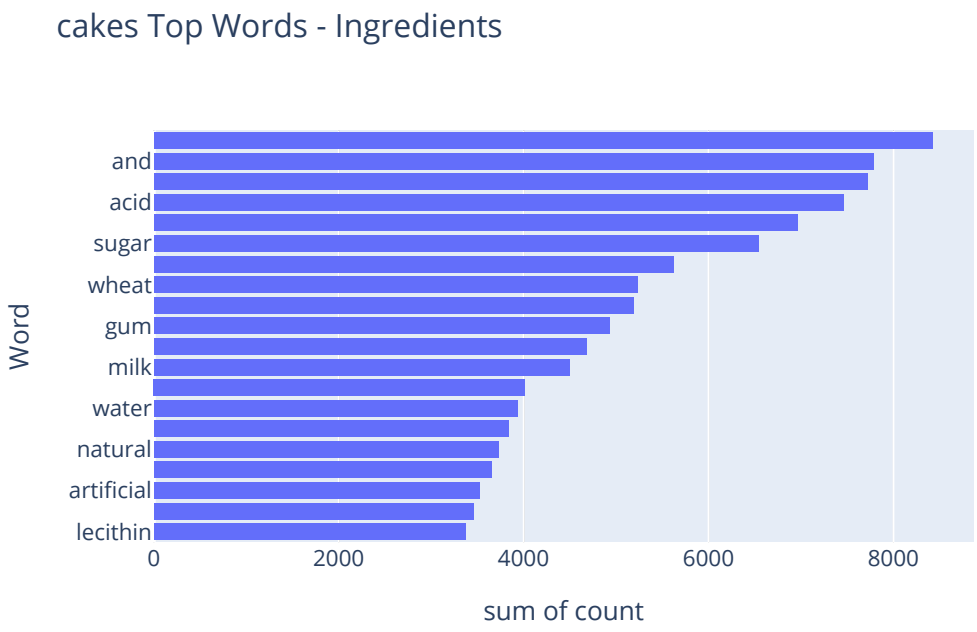
[ ]:
```
nulls = eda_df.isnull().sum().to_frame().reset_index()
nulls.columns = ['column', 'nulls']
nulls = nulls.sort_values('nulls')
px.bar(nulls, x='nulls', y='column', orientation='h', title='Nulls count',␣
↪width=600, height=350)
```

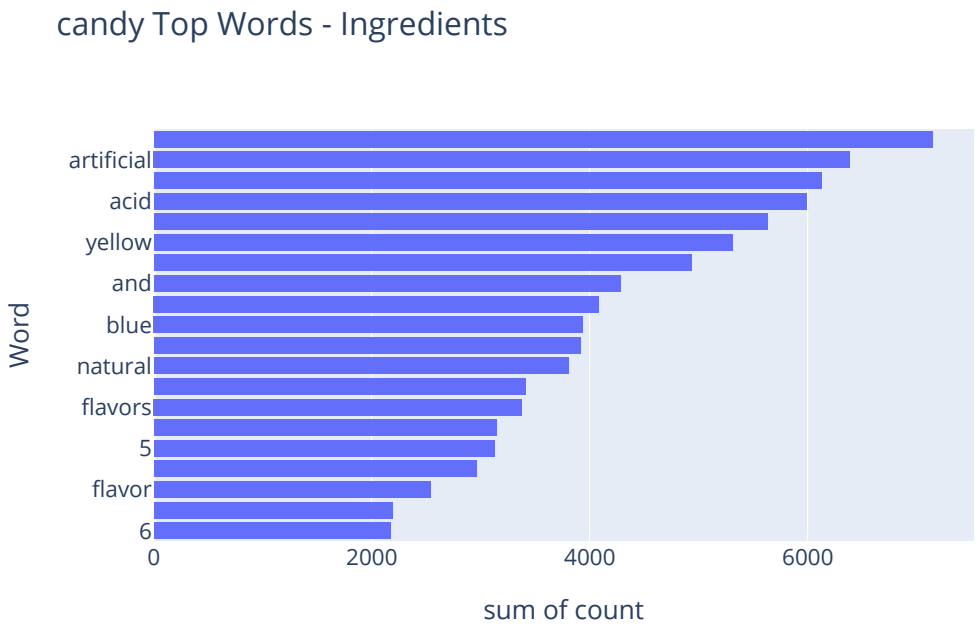


[ ]:
```
eda_df, top20_ingredients_words_by_category = get_ngrams_top_k(eda_df,␣
↪'ingredients', 1, 20)
```

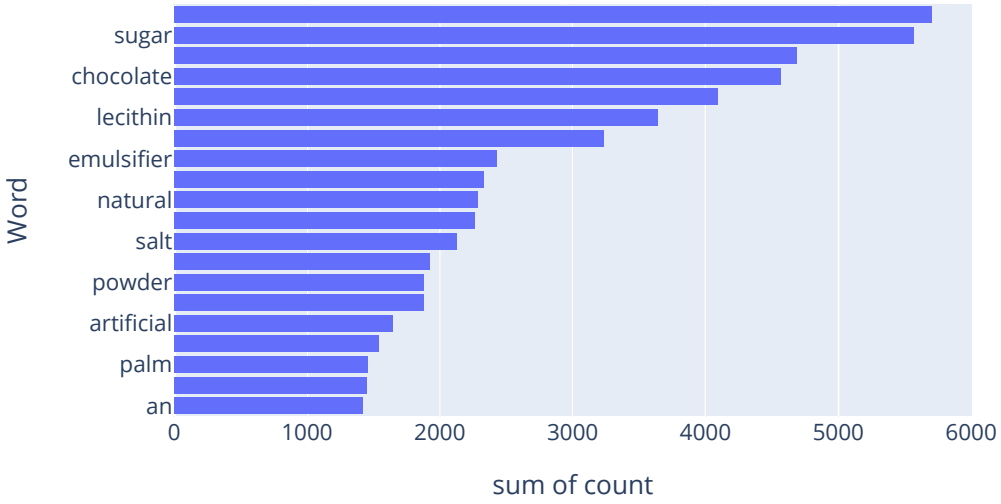
Let's have a look at some of the common words in the ingredients column:

[ ]:
```
# Get an idea of distinct words in 'ingredients' column
eda_df, top20_ingredients_words_by_category = get_ngrams_top_k(eda_df,␣
↪'ingredients', 1, 20)
for category in le.classes_:
    px.histogram(
        top20_ingredients_words_by_category[category], y='ngram', x='count',␣
↪orientation='h', width=600, height=400,
        labels={'ngram': 'Word'}, title='{} Top Words - Ingredients'.
↪format(category)
    ).show()
```

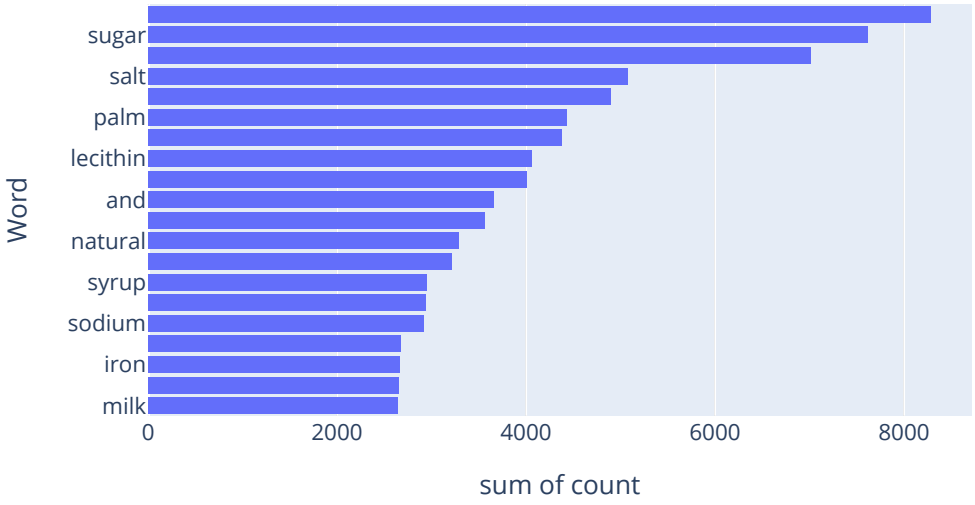

cakes Top Words - Ingredients
Word
and
acid
sugar
wheat
gum
milk
water
natural
artificial
lecithin
0
2000
4000
6000
8000
sum of count


candy Top Words - Ingredients
Word
artificial
acid
yellow
and
blue
natural
flavors
5
flavor
6
0
2000
4000
6000
sum of count

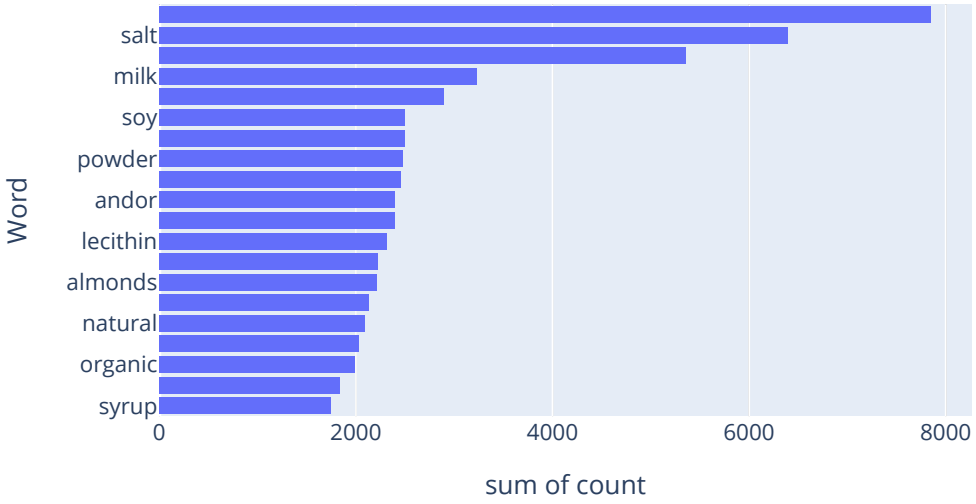
chocolate Top Words - Ingredients



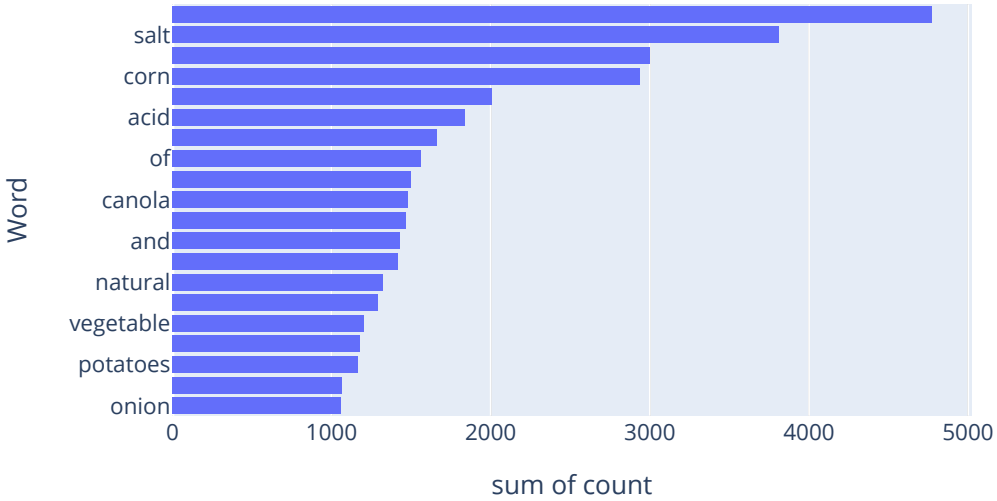
cookies Top Words - Ingredients



seeds Top Words - Ingredients

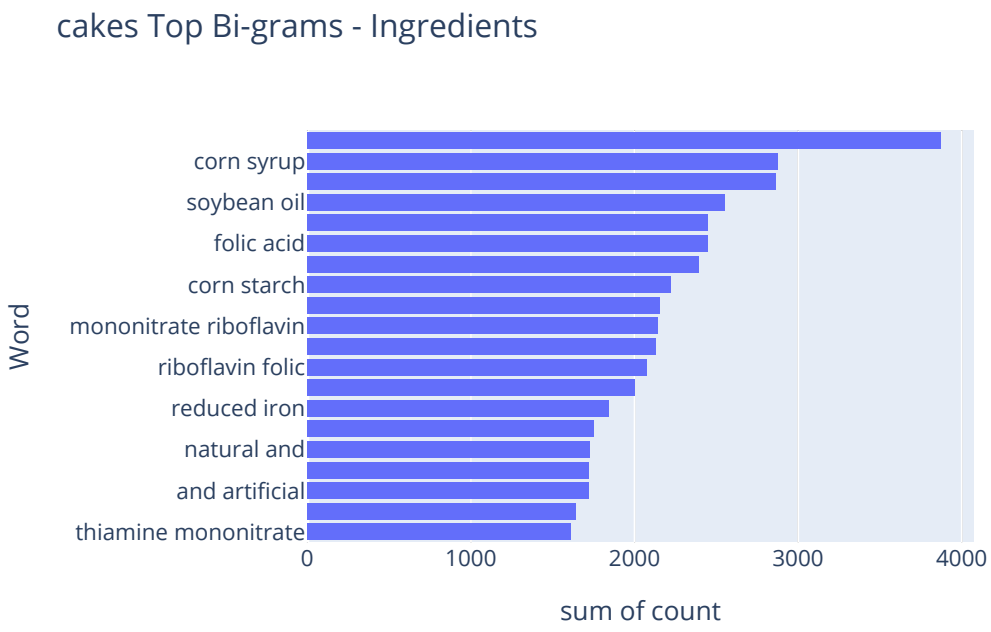


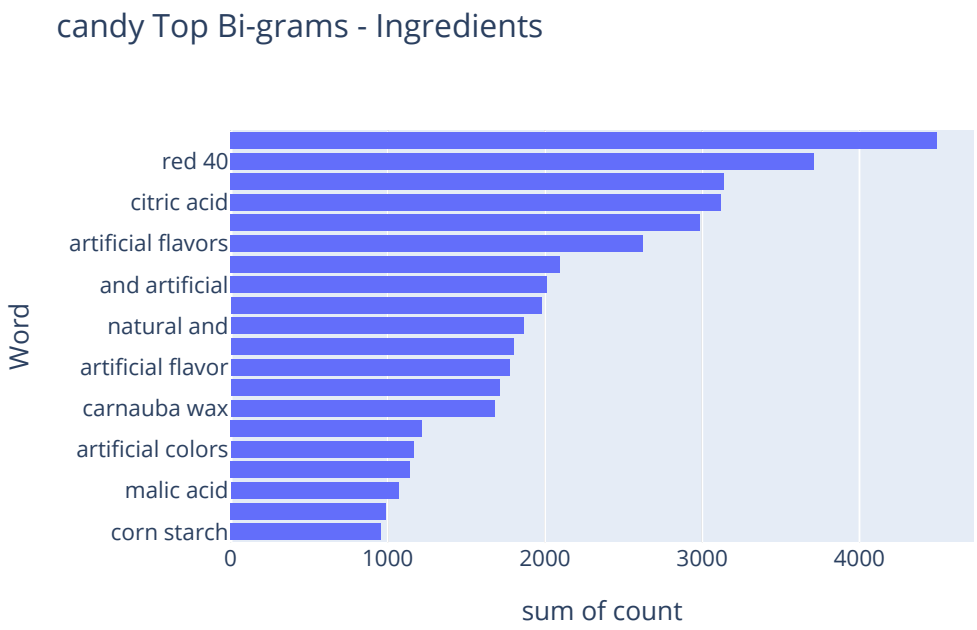
snacks Top Words - Ingredients

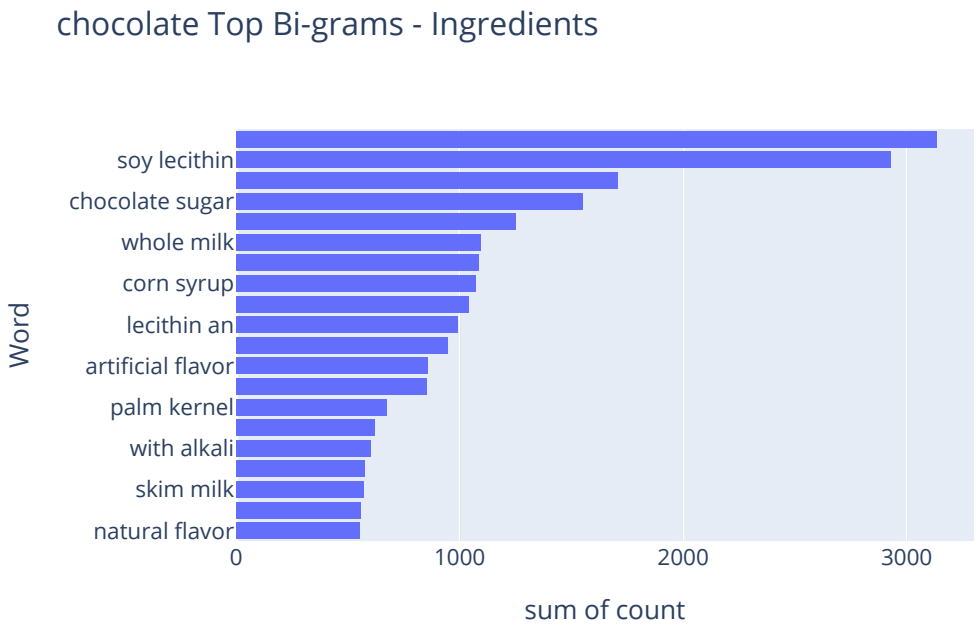


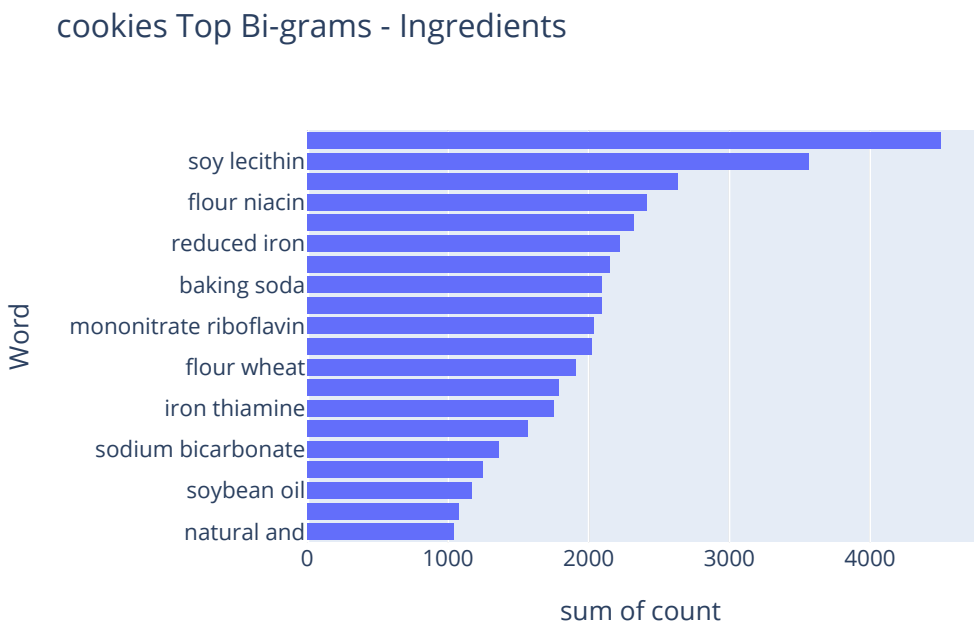
We can see we have a few words like 'and', 'or' in the top. Let us look at bigrams:

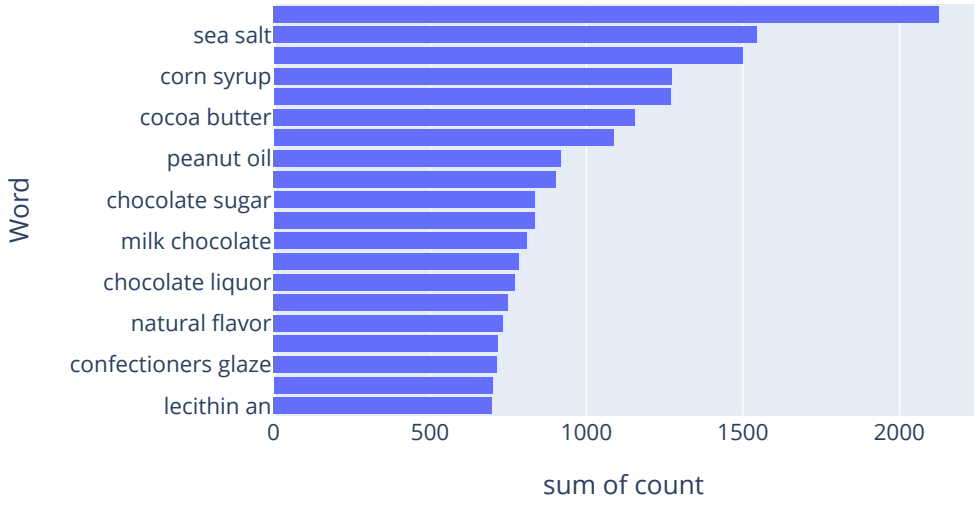
[ ]:
```
# Do same for bigrams
eda_df, top20_ingredients_bigrams_by_category = get_ngrams_top_k(eda_df,␣
 ↪'ingredients', 2, 20)
for category in le.classes_:
    px.histogram(
        top20_ingredients_bigrams_by_category[category], y='ngram', x='count',␣
 ↪orientation='h', width=600, height=400,
        labels={'ngram': 'Word'}, title='{} Top Bi-grams - Ingredients'.
 ↪format(category)
    ).show()
```

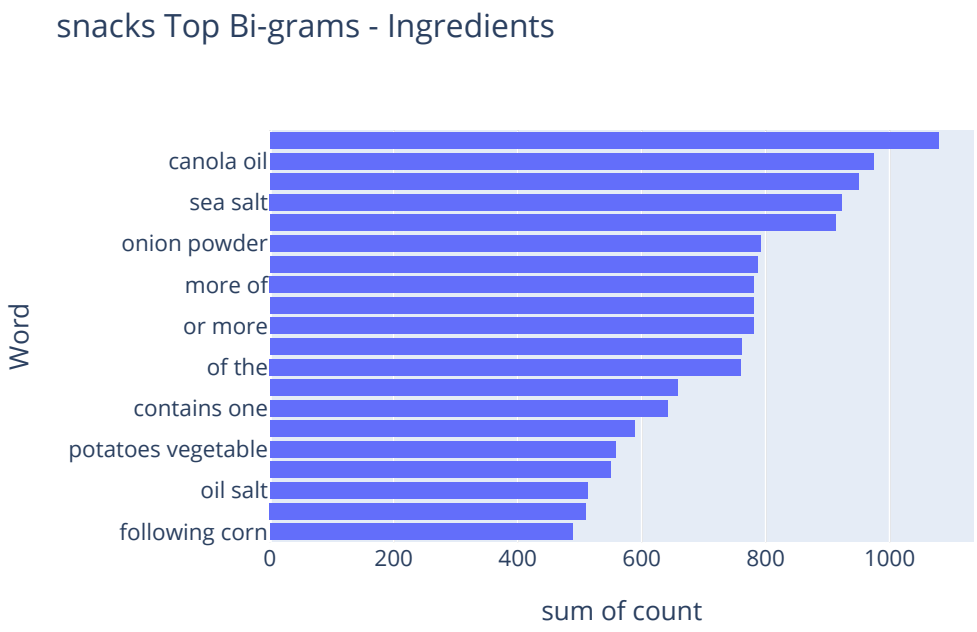



candy Top Bi-grams - Ingredients
red 40
citric acid
artificial flavors
and artificial
natural and
artificial flavor
carnauba wax
artificial colors
malic acid
corn starch
Word
0
1000
2000
3000
4000
sum of count


chocolate Top Bi-grams - Ingredients
soy lecithin
chocolate sugar
whole milk
corn syrup
lecithin an
artificial flavor
palm kernel
with alkali
skim milk
natural flavor
Word
0
1000
2000
3000
sum of count


cookies Top Bi-grams - Ingredients
soy lecithin
flour niacin
reduced iron
baking soda
mononitrate riboflavin
flour wheat
iron thiamine
sodium bicarbonate
soybean oil
natural and
Word
0
1000
2000
3000
4000
sum of count

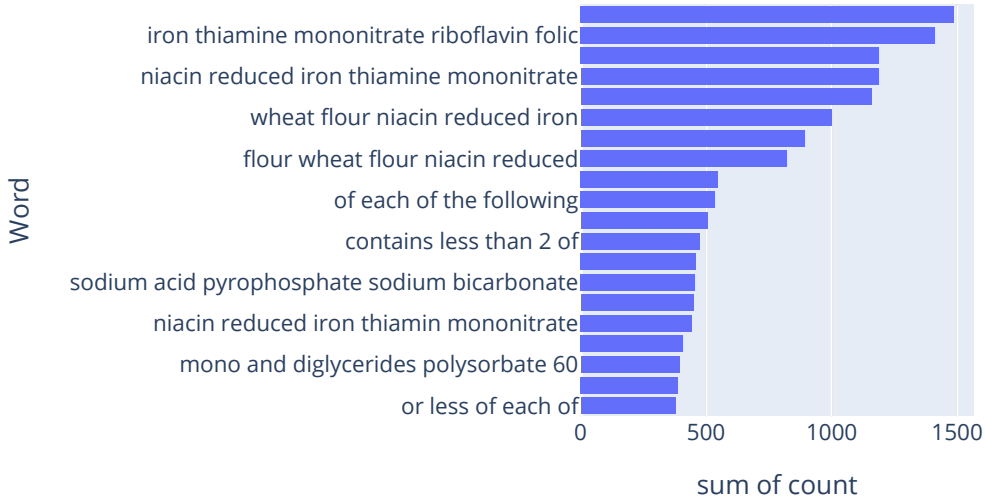

seeds Top Bi-grams - Ingredients
sea salt
corn syrup
cocoa butter
peanut oil
chocolate sugar
milk chocolate
chocolate liquor
natural flavor
confectioners glaze
lecithin an
Word
0
500
1000
1500
2000
sum of count



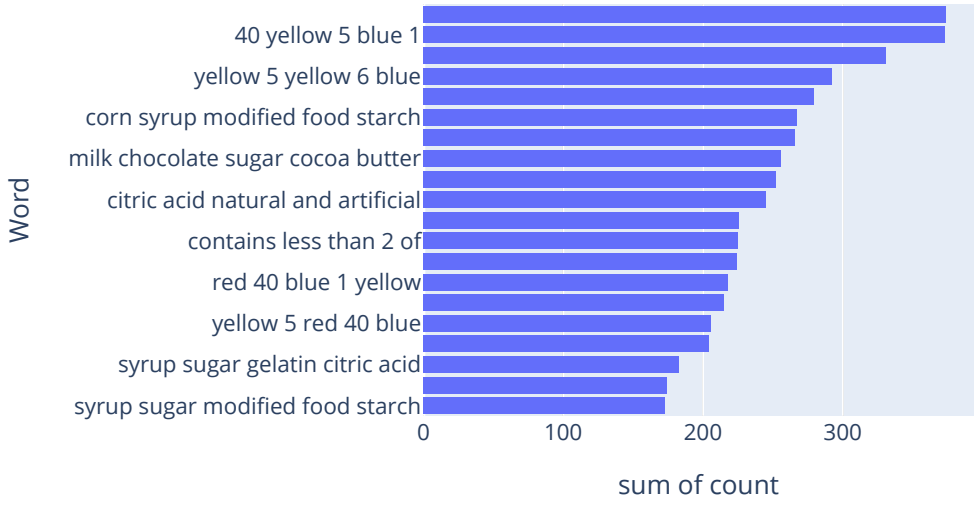
Seems we have some noise from the artificial colors and sentences. We'll keep those for now. Let us look for sentences by looking for common 5-grams:

[ ]:
```
eda_df, top20_ingredients_bigrams_by_category = get_ngrams_top_k(eda_df,␣
↪'ingredients', 5, 20)
for category in le.classes_:
    px.histogram(
        top20_ingredients_bigrams_by_category[category], y='ngram', x='count',␣
↪orientation='h', width=600, height=400,
        labels={'ngram': 'Word'}, title='{} Top Penta-grams - Ingredients'.
↪format(category)
    ).show()
```

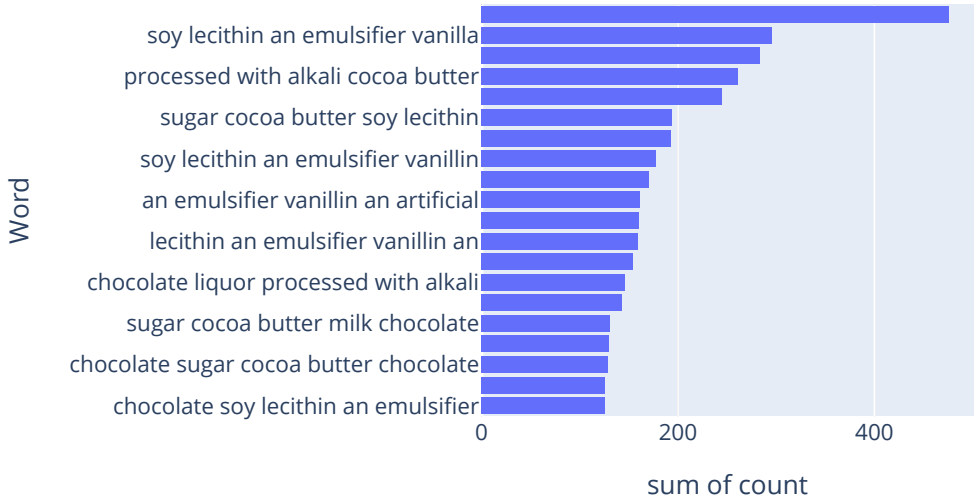
cakes Top Penta-grams - Ingredients



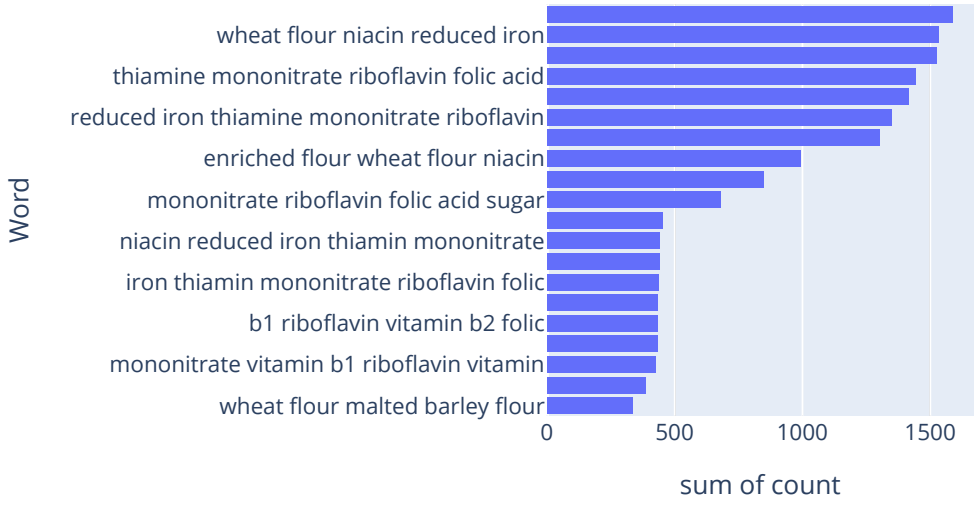
candy Top Penta-grams - Ingredients



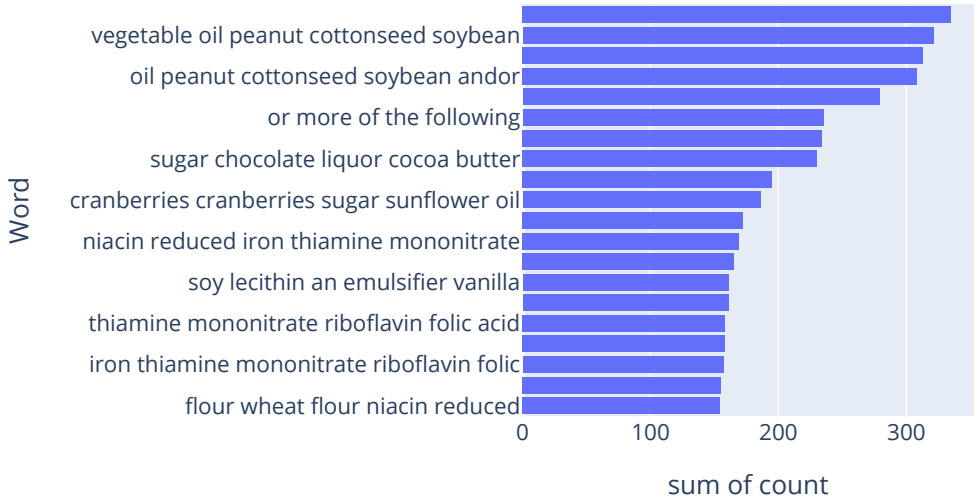
chocolate Top Penta-grams - Ingredients



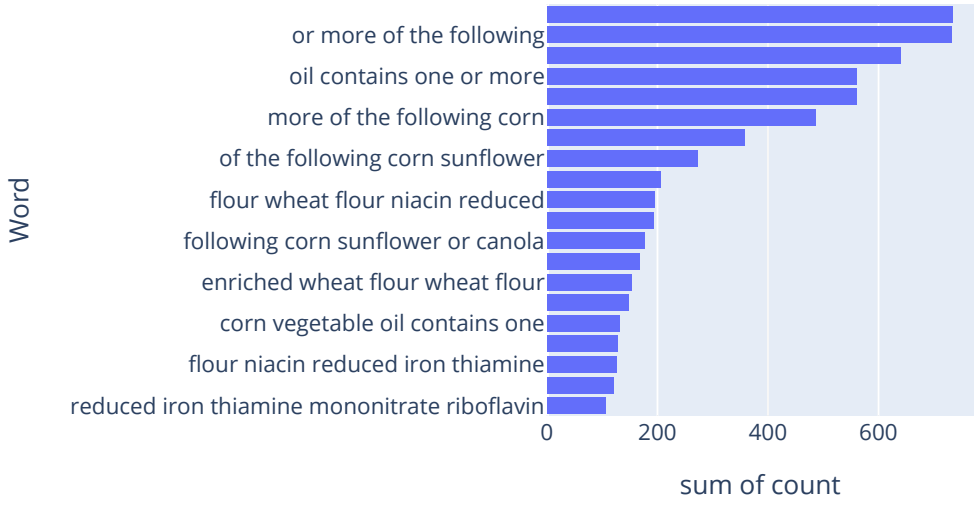
cookies Top Penta-grams - Ingredients



seeds Top Penta-grams - Ingredients



snacks Top Penta-grams - Ingredients



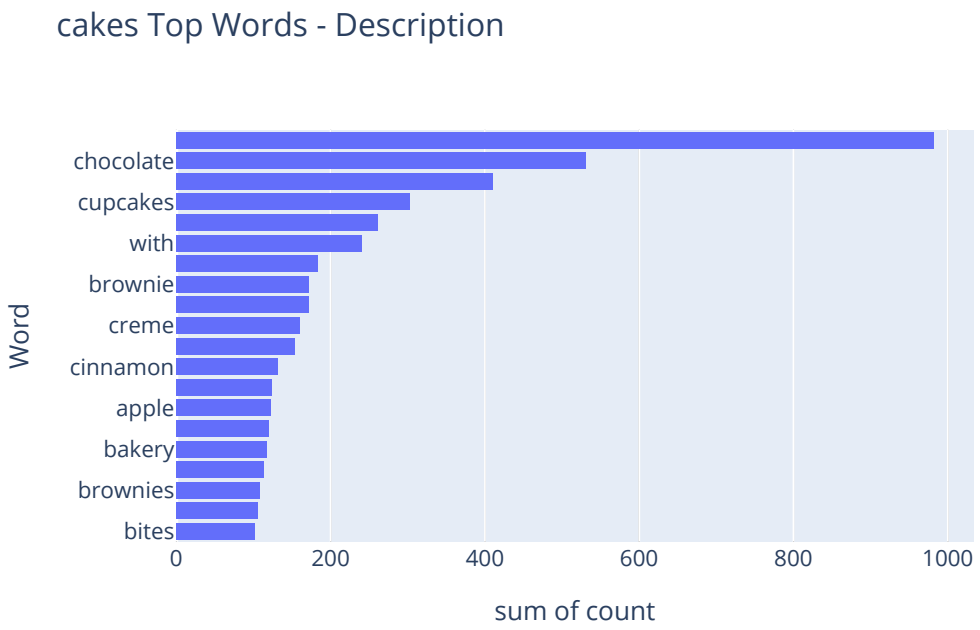
We see that we have some sentences and numbers in the ingredients column.

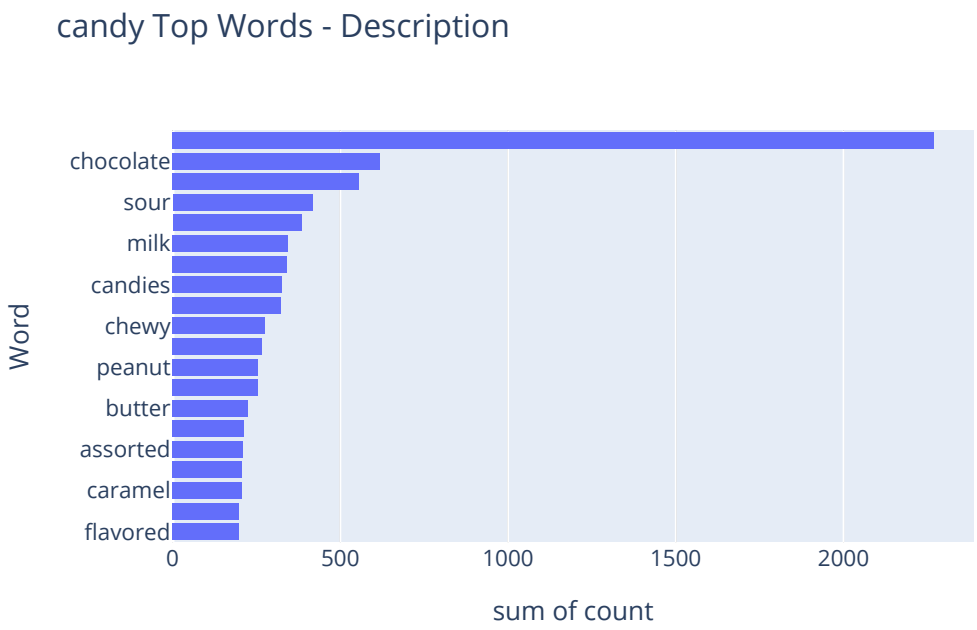
Let us do the same for the description column:

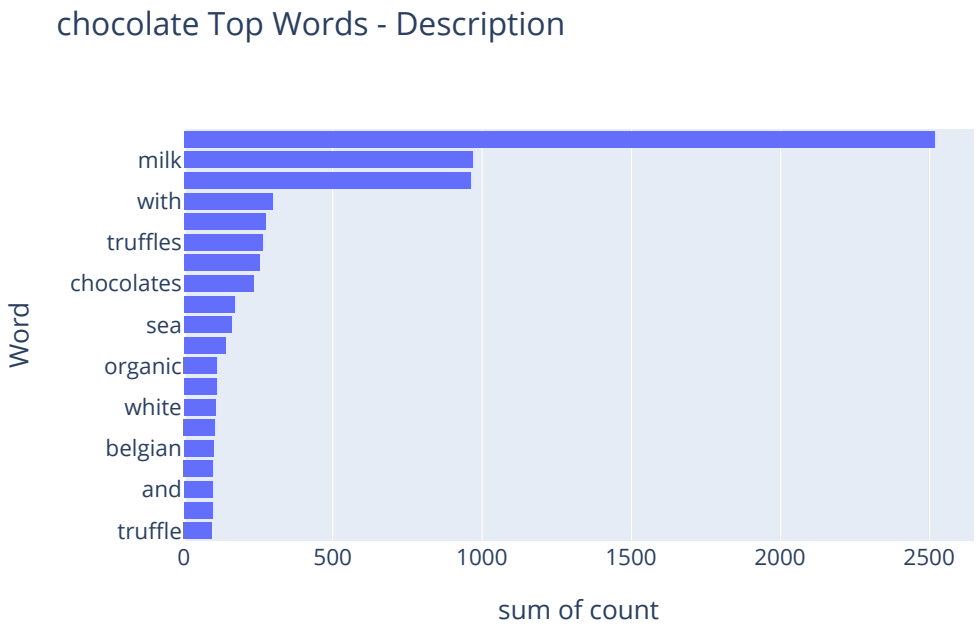
[ ]:
```
eda_df, top20_description_words_by_category = get_ngrams_top_k(eda_df,␣
 ↪'description', 1, 20)
eda_df, top20_description_bigrams_by_category = get_ngrams_top_k(eda_df,␣
 ↪'description', 2, 20)
eda_df, top20_description_trigrams_by_category = get_ngrams_top_k(eda_df,␣
 ↪'description', 3, 20)
```

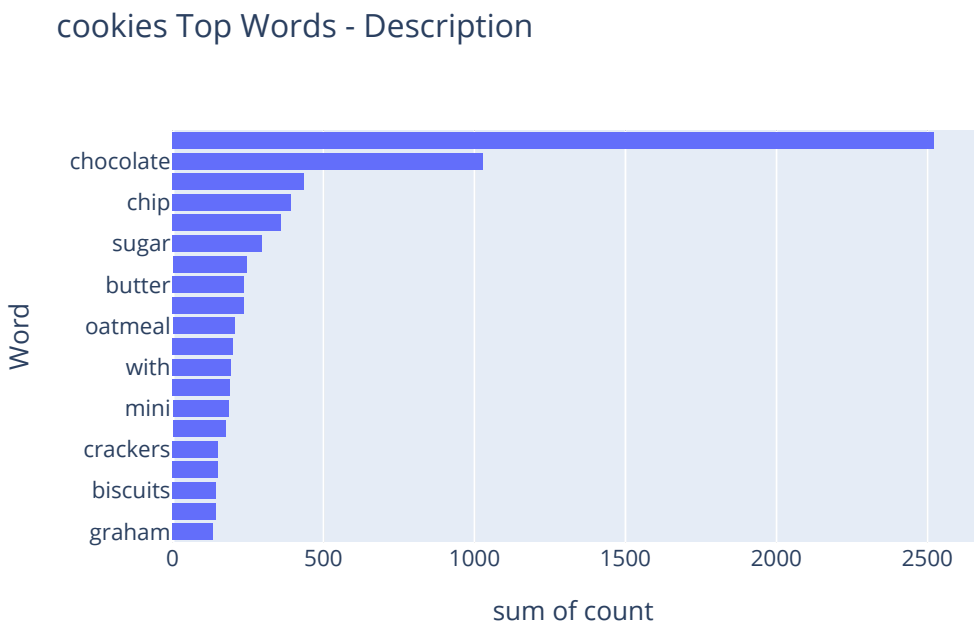
Top words:

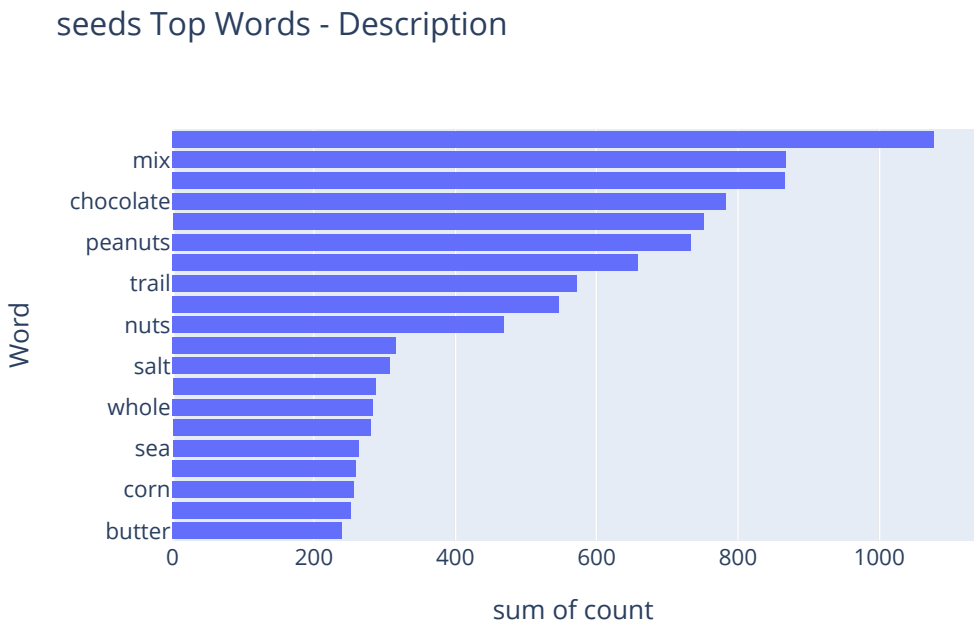
[ ]:
```
for category in le.classes_:
    px.histogram(
        top20_description_words_by_category[category], y='ngram', x='count',␣
 ↪orientation='h', width=600, height=400,
        labels={'ngram': 'Word'}, title='{} Top Words - Description'.
 ↪format(category)
    ).show()
```

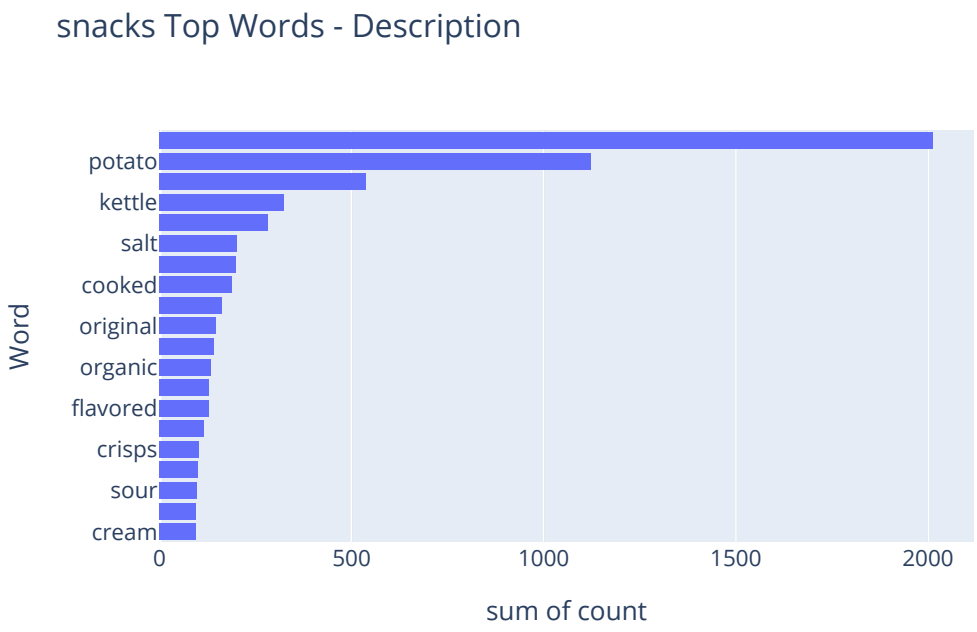



candy Top Words - Description
chocolate
sour
milk
candies
chewy
peanut
butter
assorted
caramel
flavored
Word
0
500
1000
1500
2000
sum of count


chocolate Top Words - Description
milk
with
truffles
chocolates
sea
organic
white
belgian
and
truffle
Word
0
500
1000
1500
2000
2500
sum of count

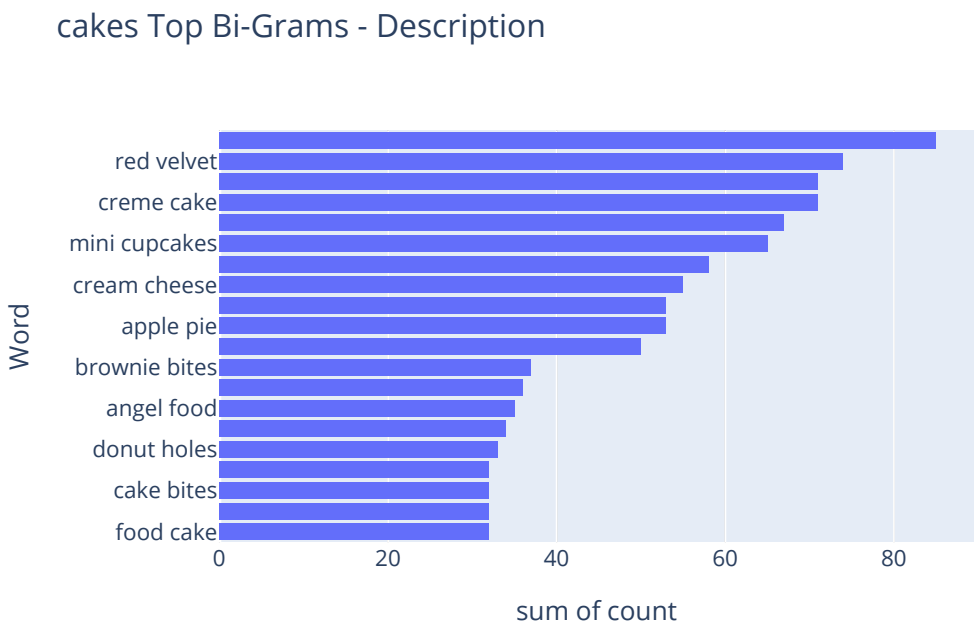

cookies Top Words - Description
chocolate
chip
sugar
butter
oatmeal
with
mini
crackers
biscuits
graham
Word
0
500
1000
1500
2000
2500
sum of count

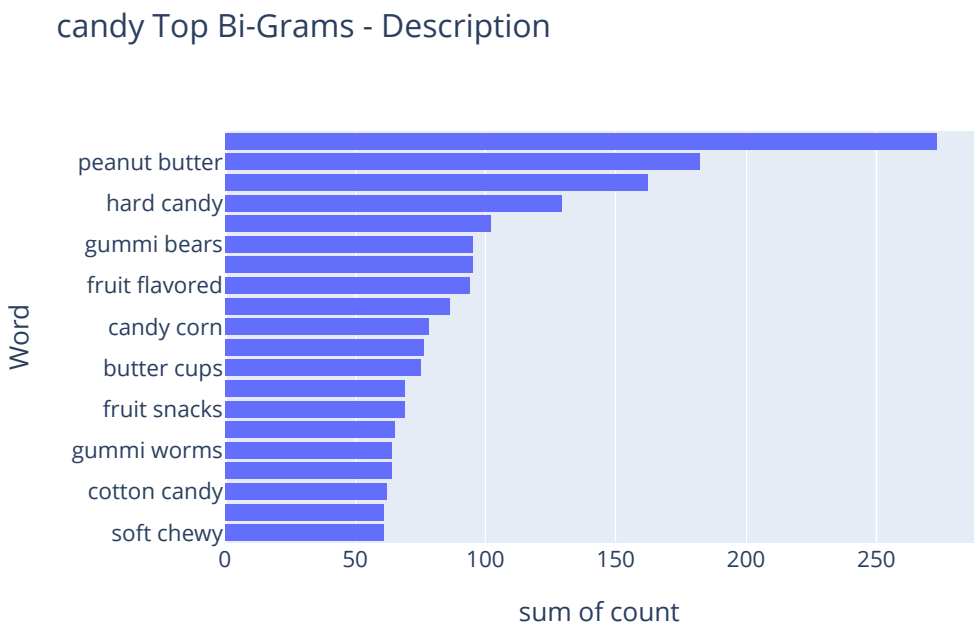

seeds Top Words - Description
mix
chocolate
peanuts
trail
nuts
salt
whole
sea
corn
butter
Word
0
200
400
600
800
1000
sum of count

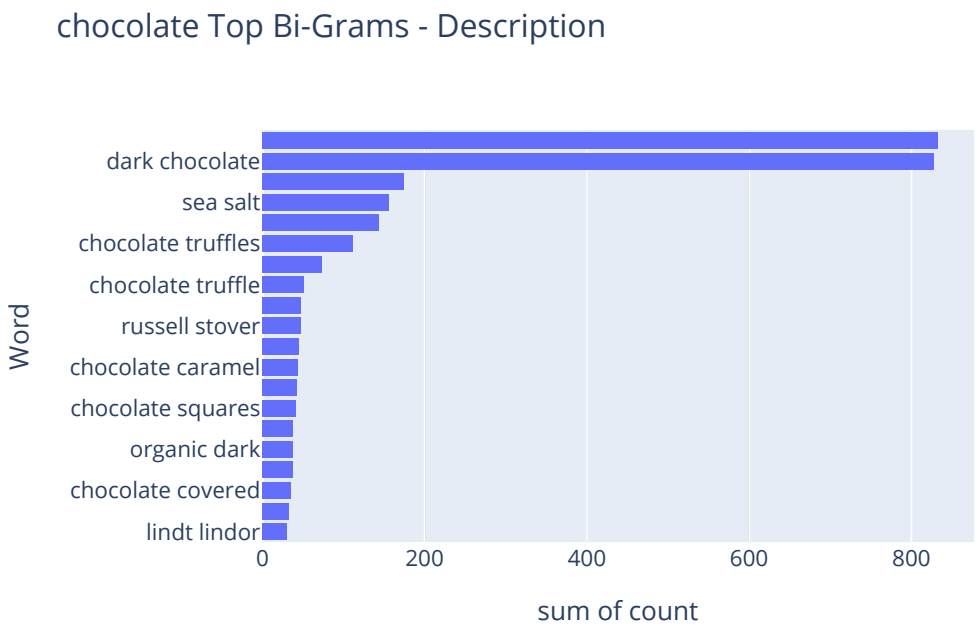


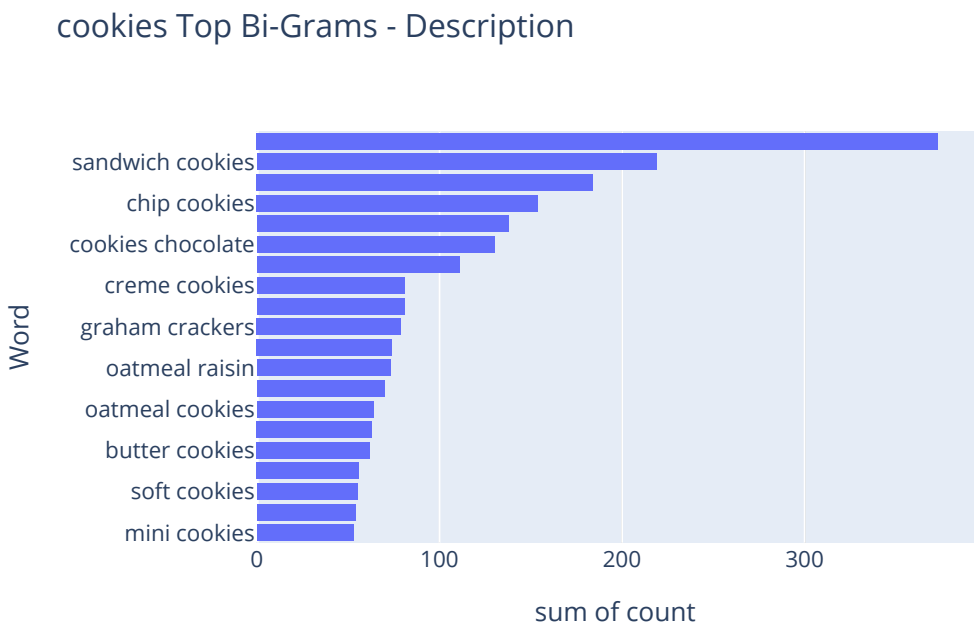
Top Bi-grams:

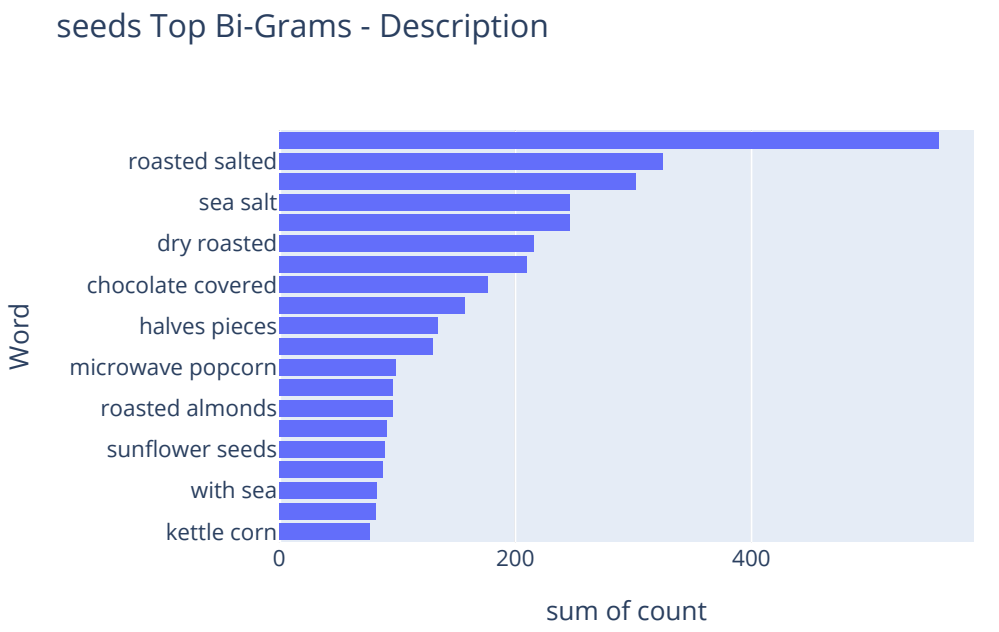
[ ]:
```
for category in le.classes_:
    px.histogram(
        top20_description_bigrams_by_category[category], y='ngram', x='count',␣
↪orientation='h', width=600, height=400,
        labels={'ngram': 'Word'}, title='{} Top Bi-Grams - Description'.
↪format(category)
    ).show()
```

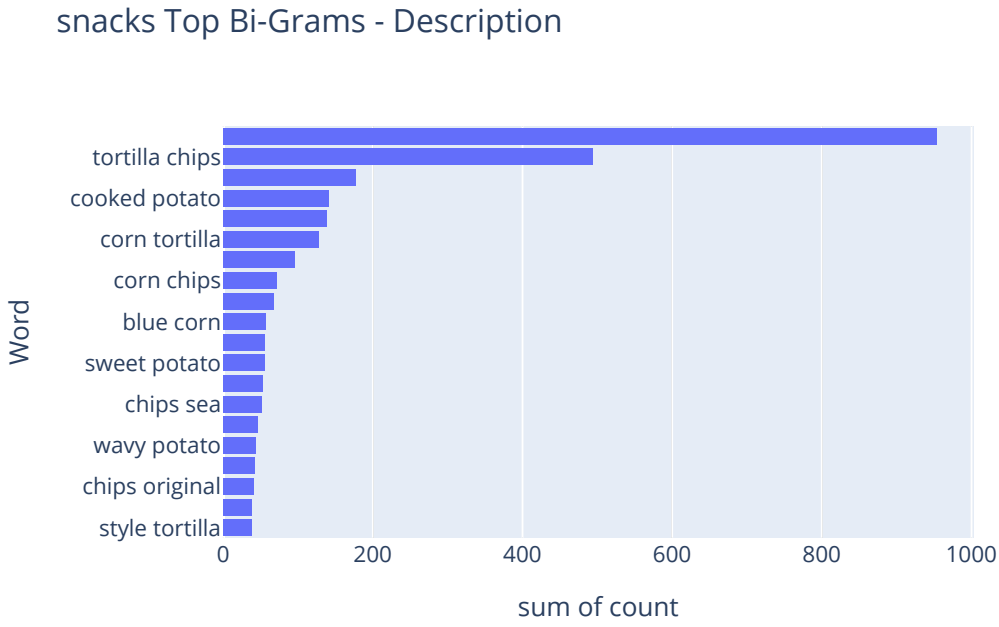

cakes Top Bi-Grams - Description
red velvet
creme cake
mini cupcakes
cream cheese
apple pie
brownie bites
angel food
donut holes
cake bites
food cake
Word
0
20
40
60
80
sum of count


candy Top Bi-Grams - Description
peanut butter
hard candy
gummi bears
fruit flavored
candy corn
butter cups
fruit snacks
gummi worms
cotton candy
soft chewy
Word
0
50
100
150
200
250
sum of count


chocolate Top Bi-Grams - Description
dark chocolate
sea salt
chocolate truffles
chocolate truffle
russell stover
chocolate caramel
chocolate squares
organic dark
chocolate covered
lindt lindor
Word
0
200
400
600
800
sum of count


cookies Top Bi-Grams - Description
sandwich cookies
chip cookies
cookies chocolate
creme cookies
graham crackers
oatmeal raisin
oatmeal cookies
butter cookies
soft cookies
mini cookies
Word
0
100
200
300
sum of count


seeds Top Bi-Grams - Description
roasted salted
sea salt
dry roasted
chocolate covered
halves pieces
microwave popcorn
roasted almonds
sunflower seeds
with sea
kettle corn
Word
0
200
400
sum of count


snacks Top Bi-Grams - Description
tortilla chips
cooked potato
corn tortilla
corn chips
blue corn
sweet potato
chips sea
wavy potato
chips original
style tortilla
Word
0
200
400
600
800
1000
sum of count

Now, let us move on to some analyses for the nutrients.

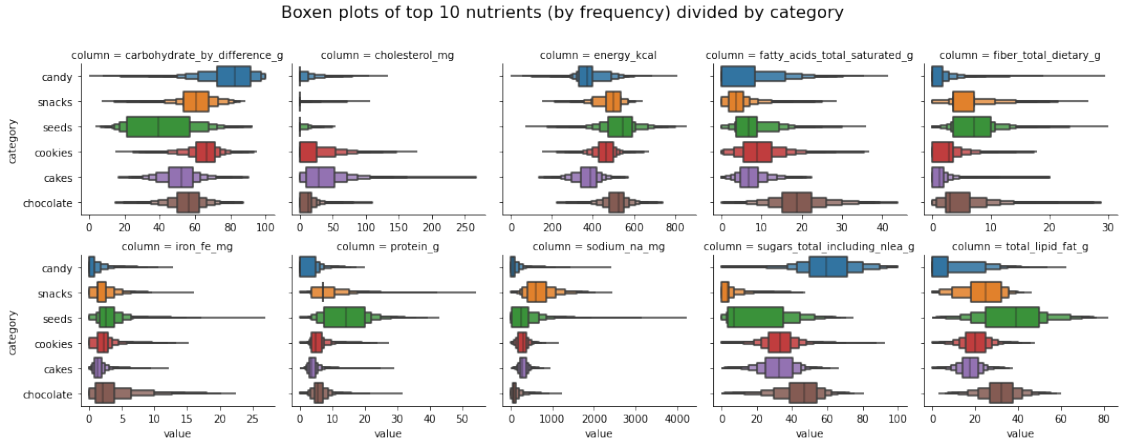
Let us first take the top 10 most common nutrients:

[ ]:
```
top_10_nutrients = nutrients_column_values['name'].head(10)['value']
```

Some boxen plots by category:

[ ]:
```
long = pd.melt(eda_df, id_vars=['idx', 'category', 'category_enc'],␣
 ↪var_name='column')
long = long[long['column'].isin(top_10_nutrients)].reset_index()
boxen_plots = sns.catplot(data=long, kind='boxen', x='value', y='category',␣
 ↪showfliers = False, col='column', col_wrap=5, height=3, sharex=False,␣
 ↪aspect=1,
orient='h', margin_titles=True)
boxen_plots.fig.subplots_adjust(top=0.85)
boxen_plots.fig.suptitle('Boxen plots of top 10 nutrients (by frequency)␣
 ↪divided by category', fontsize=16)
```

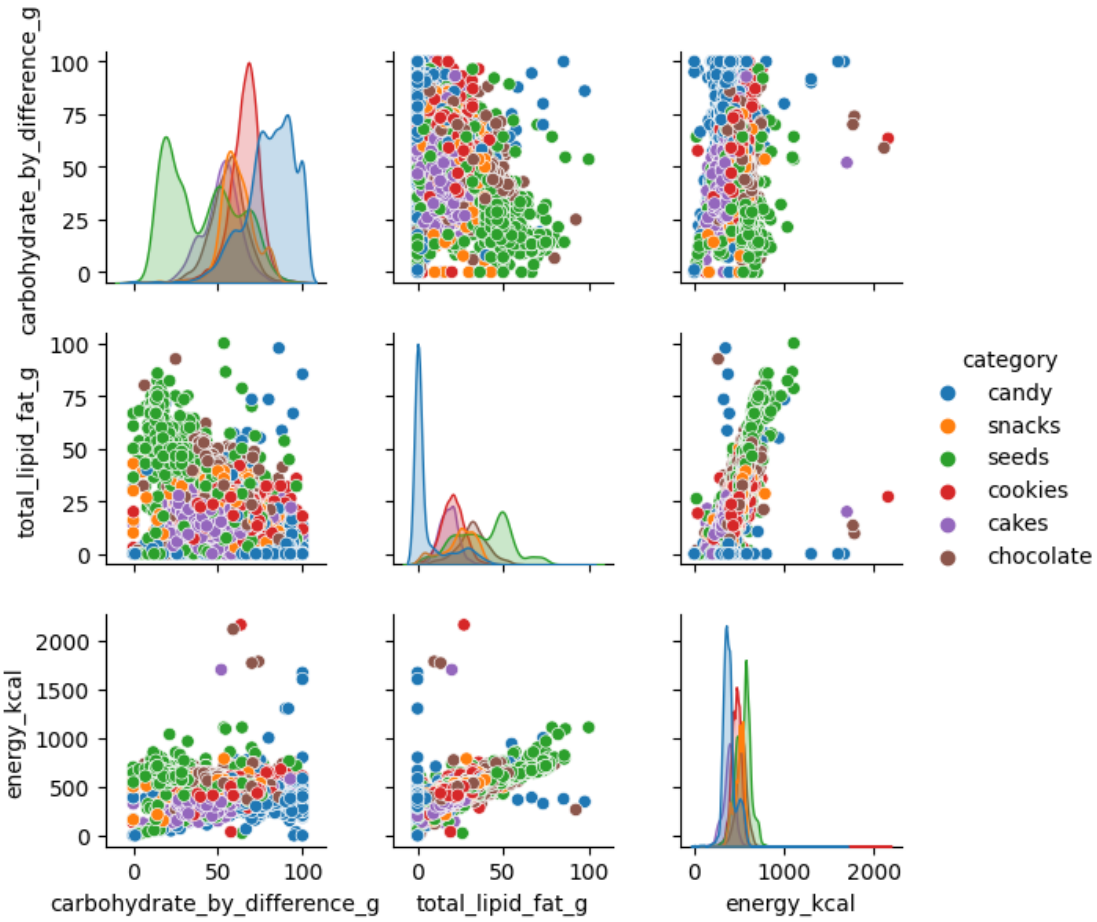
[ ]:
```
Text(0.5, 0.98, 'Boxen plots of top 10 nutrients (by frequency) divided by
category')
```



Pairwise plots by category for top 3 nutrients (to avoid a larger than necessary plot):

[ ]:
```
eda_df_nutrients = eda_df[list(top_10_nutrients.head(3)) + ['category']]
sns.pairplot(eda_df_nutrients, dropna=True, height=2, aspect=1, hue='category')
```

[ ]:
```
<seaborn.axisgrid.PairGrid at 0x21aad50e160>
```


carbohydrate_by_difference_g
total_lipid_fat_g
energy_kcal
0
25
50
75
100
0
500
1000
1500
2000
0
50
100
0
1000
2000
category
candy
snacks
seeds
cookies
cakes
chocolate