Market Basket Analysis using Apriori Algorithm

- python -m pip install mlxtend
- python -m pip install apyori
- python -m pip install squarify
- python -m pip install wordcloud
- conda install -c https://conda.anaconda.org/conda-forge (https://conda.anaconda.org/conda-forge) wordcloud

Import important libraries

In [2]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import squarify
import seaborn as sns
from wordcloud import WordCloud
import networkx as nx

from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder

import warnings
warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
%matplotlib inline
```

Reading the dataset

In [4]:
```python
data = pd.read_csv('C:/Users/Eric/Documents/Jupyter Notebook/practical/data/MBA.csv', header = None)
```

In [5]:
```python
# checking the head of the data

data.head()
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice	s
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

In [6]:
```python
# checkng the tail of the data

data.tail()
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
7496	butter	light mayo	fresh bread	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7497	burgers	frozen vegetables	eggs	french fries	magazines	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7498	chicken	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7499	escalope	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7500	eggs	frozen smoothie	yogurt cake	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [7]:

```python
# let's check the shape of the dataset
data.shape
```

Out[7]:

```
(7501, 20)
```

In [8]:

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7501 entries, 0 to 7500
Data columns (total 20 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       7501 non-null   object
 1   1       5747 non-null   object
 2   2       4389 non-null   object
 3   3       3345 non-null   object
 4   4       2529 non-null   object
 5   5       1864 non-null   object
 6   6       1369 non-null   object
 7   7       981 non-null    object
 8   8       654 non-null    object
 9   9       395 non-null    object
 10  10      256 non-null    object
 11  11      154 non-null    object
 12  12      87 non-null     object
 13  13      47 non-null     object
```

In [9]:

```
# let's describe the dataset

data.describe()
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1
count	7501	5747	4389	3345	2529	1864	1369	981	654	395	256	154	87	47	25	8	4	
unique	115	117	115	114	110	106	102	98	88	80	66	50	43	28	19	8	3	
top	mineral water	mineral water	mineral water	mineral water	green tea	french fries	green tea	green tea	green tea	green tea	low fat yogurt	green tea	green tea	green tea	magazines	salmon	frozen smoothie	prote ba
freq	577	484	375	201	153	107	96	67	57	31	22	15	8	4	3	1	2	

In [10]:

```
data.isnull().sum()
```

Out[10]:

```
0        0
1     1754
2     3112
3     4156
4     4972
5     5637
6     6132
7     6520
8     6847
9     7106
10    7245
11    7347
12    7414
13    7454
14    7476
15    7493
16    7497
17    7497
18    7498
```

In [11]:
```
data[0]
```

Out[11]:
```
0             shrimp
1            burgers
2            chutney
3             turkey
4      mineral water
           ...      
7496          butter
7497         burgers
7498         chicken
7499       escalope
7500            eggs
Name: 0, Length: 7501, dtype: object
```

EDA

Plotting most popular name of items

In [12]:

```
plt.rcParams['figure.figsize'] = (15, 15)
wordcloud = WordCloud(background_color = 'white', width = 1200,  height = 1200, max_words = 121).generate(str(dat
plt.imshow(wordcloud)
plt.axis('off')
plt.title('Most Popular Items',fontsize = 20)
plt.show()
```

Most Popular Items

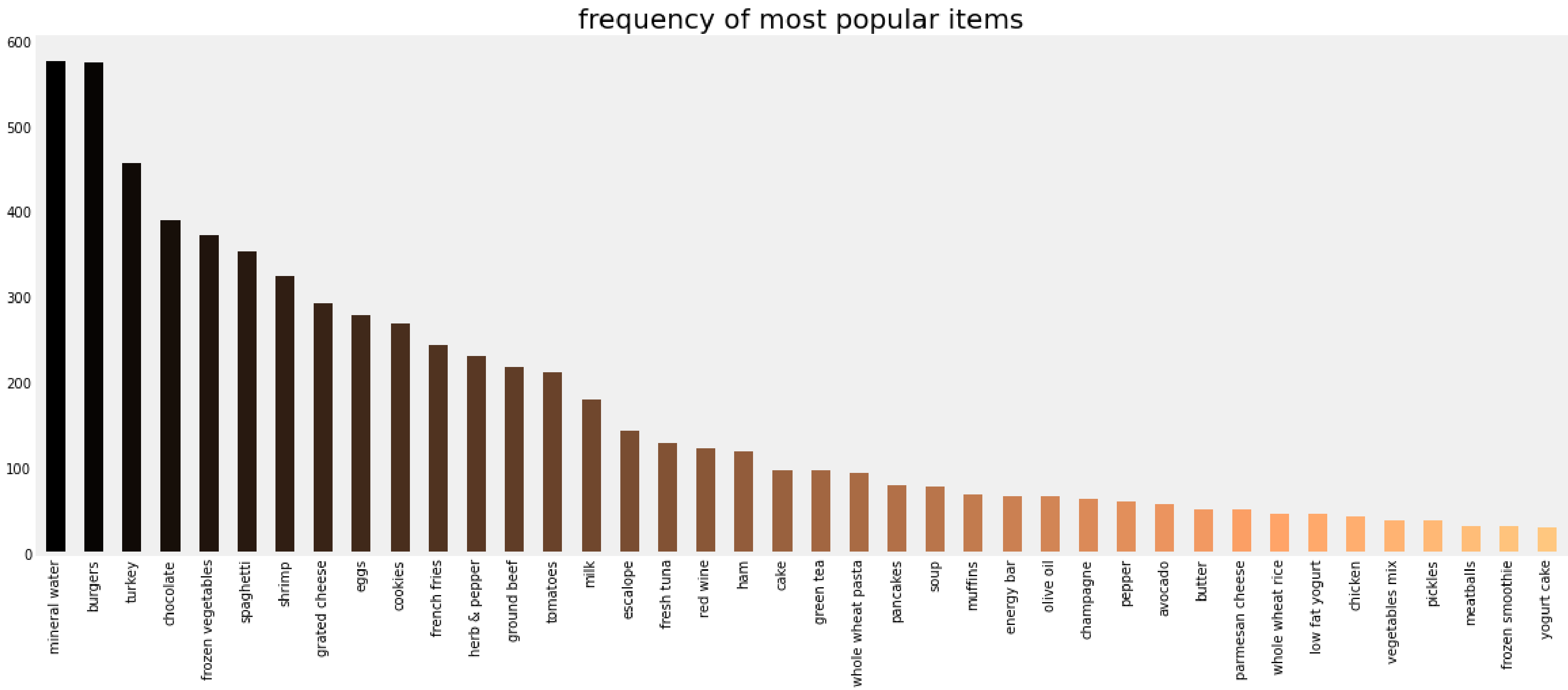
In [13]:
```
data[0].value_counts()
```

Out[13]:
```
mineral water        577
burgers              576
turkey               458
chocolate            391
frozen vegetables    373
                    ... 
cauliflower            1
ketchup                1
cream                  1
body spray             1
oatmeal                1
Name: 0, Length: 115, dtype: int64
```

Looking at the frequency of most popular items

In [14]:

```
plt.rcParams['figure.figsize'] = (18, 7)
color = plt.cm.copper(np.linspace(0, 1, 40))
data[0].value_counts().head(40).plot.bar(color = color)
plt.title('frequency of most popular items', fontsize = 20)
plt.xticks(rotation = 90 )
plt.grid()
plt.show()
```



In [15]:
```
data[0].value_counts().head(50)
```

Out[15]:
```
mineral water        577
burgers              576
turkey               458
chocolate            391
frozen vegetables    373
spaghetti            354
shrimp               325
grated cheese        293
eggs                 279
cookies              270
french fries         244
herb & pepper        232
ground beef          218
tomatoes             212
milk                 181
escalope             143
fresh tuna           129
red wine             123
ham                  120
```

In [16]:
```
y = data[0].value_counts().head(50).to_frame()
y
```

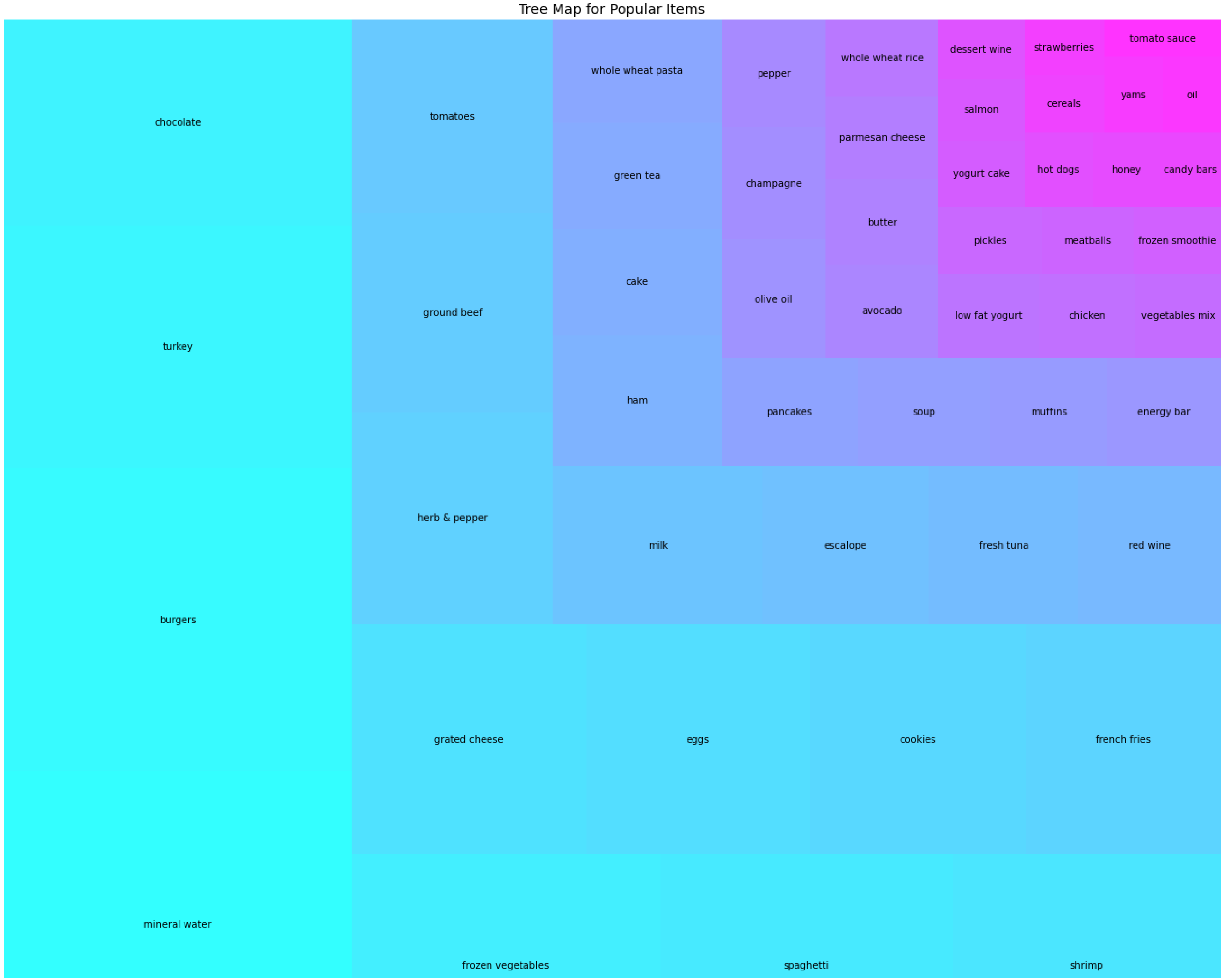
Out[16]:

	0
mineral water	577
burgers	576
turkey	458
chocolate	391
frozen vegetables	373
spaghetti	354
shrimp	325
grated cheese	293
eggs	279
cookies	270
french fries	244

Plotting a tree map

In [17]:

```
plt.rcParams['figure.figsize'] = (20, 20)
color = plt.cm.cool(np.linspace(0, 1, 50))
squarify.plot(sizes = y.values, label = y.index, alpha=.8, color = color)
plt.title('Tree Map for Popular Items')
plt.axis('off')
plt.show()
```


Tree Map for Popular Items
chocolate
turkey
burgers
mineral water
tomatoes
ground beef
herb & pepper
whole wheat pasta
green tea
cake
ham
pepper
champagne
olive oil
whole wheat rice
parmesan cheese
butter
avocado
dessert wine
salmon
yogurt cake
strawberries
cereals
tomato sauce
yams
oil
hot dogs
honey
candy bars
pickles
meatballs
frozen smoothie
low fat yogurt
chicken
vegetables mix
pancakes
soup
muffins
energy bar
milk
escalope
fresh tuna
red wine
grated cheese
eggs
cookies
french fries
frozen vegetables
spaghetti
shrimp

Syntax: DataFrame.truncate(before=None, after=None, axis=None, copy=True)

- Parameter :
- before : Truncate all rows before this index value.
- after : Truncate all rows after this index value.
- axis : Axis to truncate. Truncates the index (rows) by default.
- copy : Return a copy of the truncated section.
- Returns : The truncated Series or DataFrame.

In [18]:
```
data['food'] = 'Food'
food = data.truncate(before = -1, after = 15)
```

In [19]:
```
food
```

Out[19]:

	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	...	green tea	honey	salad	mineral water	salmon	antioxyd ju
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	N
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	N
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	N
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	N
5	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	N
6	whole wheat pasta	french fries	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	N

from_pandas_edgelist(df, source='source', target='target', edge_attr=None, create_using=None, edge_key=None)

Parameters

- df : Pandas DataFrame An edge list representation of a graph
- source : str or int A valid column name (string or integer) for the source nodes.
- target : str or int A valid column name (string or integer) for the target nodes.
- edge_attr : str or int, iterable, True, or None A valid column name (str or int) or iterable of column names that are used to retrieve items and add them to the graph as edge attributes. If True, all of the remaining columns will be added. If None, no edge attributes are added to the graph.
- create_using : NetworkX graph constructor, optional (default=nx.Graph) Graph type to create. If graph instance, then cleared before populated.
- edge_key : str or None, optional (default=None) A valid column name for the edge keys (for a MultiGraph). The values in this column are used for the edge keys when adding edges if create_using is a multigraph

Here Target is 0

In [20]:
```
food = nx.from_pandas_edgelist(food, source = 'food', target = 0, edge_attr = True)
```

In [21]:
```
pos = nx.spring_layout(food)
```

In [22]:

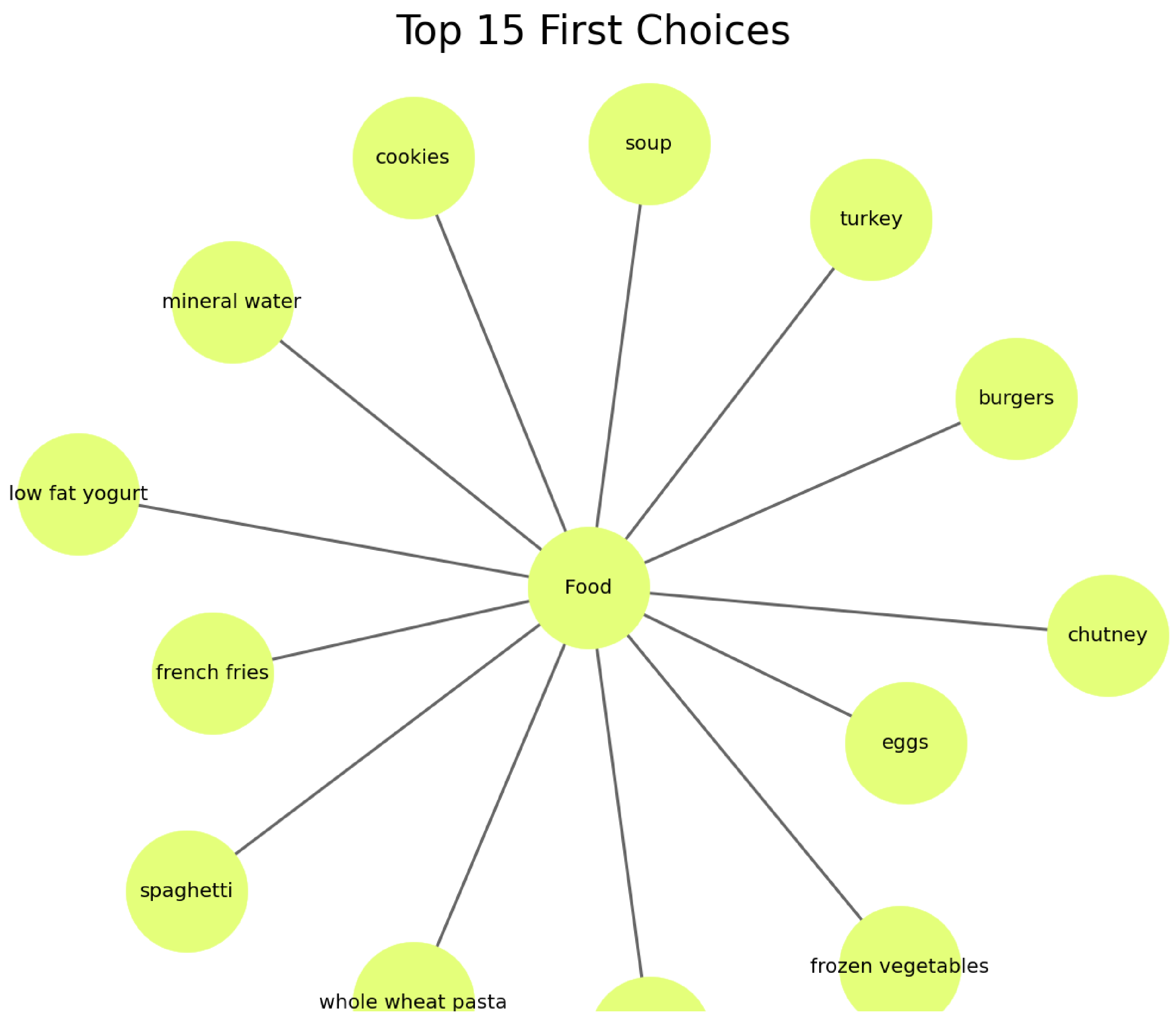
```
pos
```

Out[22]:

```
{'Food': array([0.00235065, 0.00337298]),
 'shrimp': array([ 0.12044205, -0.91478962]),
 'burgers': array([0.82326287, 0.38959108]),
 'chutney': array([ 1.        , -0.09331728]),
 'turkey': array([0.5452815 , 0.75414647]),
 'mineral water': array([-0.68300856,  0.58609314]),
 'low fat yogurt': array([-0.9785123 ,  0.19404559]),
 'whole wheat pasta': array([-0.33402417, -0.84198262]),
 'soup': array([0.11757481, 0.90901038]),
 'frozen vegetables': array([ 0.60071806, -0.76919842]),
 'french fries': array([-0.71956048, -0.17044391]),
 'eggs': array([ 0.61072909, -0.3120902 ]),
 'cookies': array([-0.33488785,  0.88109727]),
 'spaghetti': array([-0.77036567, -0.61553487])}
```

In [23]:

```
plt.rcParams['figure.figsize'] = (20, 20)
color = plt.cm.Wistia(np.linspace(0, 15, 1))
nx.draw_networkx_nodes(food, pos, node_size = 15000, node_color = color)
nx.draw_networkx_edges(food, pos, width = 3, alpha = 0.6, edge_color = 'black')
nx.draw_networkx_labels(food, pos, font_size = 20, font_family = 'sans-serif')
plt.axis('off')
plt.grid()
plt.title('Top 15 First Choices', fontsize = 40)
plt.show()
```



Top 15 First Choices
cookies
soup
turkey
mineral water
burgers
low fat yogurt
Food
chutney
french fries
eggs
spaghetti
frozen vegetables
whole wheat pasta

In [24]:
```python
data['secondchoice'] = 'Second Choice'
secondchoice = data.truncate(before = -1, after = 15)
secondchoice
```

Out[24]:

	0	1	2	3	4	5	6	7	8	9	...	12	13	14	15	16	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	...	honey	salad	mineral water	salmon	antioxydant juice	sn
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
5	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
	whole																

In [25]:

```python
secondchoice = nx.from_pandas_edgelist(secondchoice, source = 'food', target = 1, edge_attr = True) # Here Target
pos = nx.spring_layout(secondchoice)
pos
```

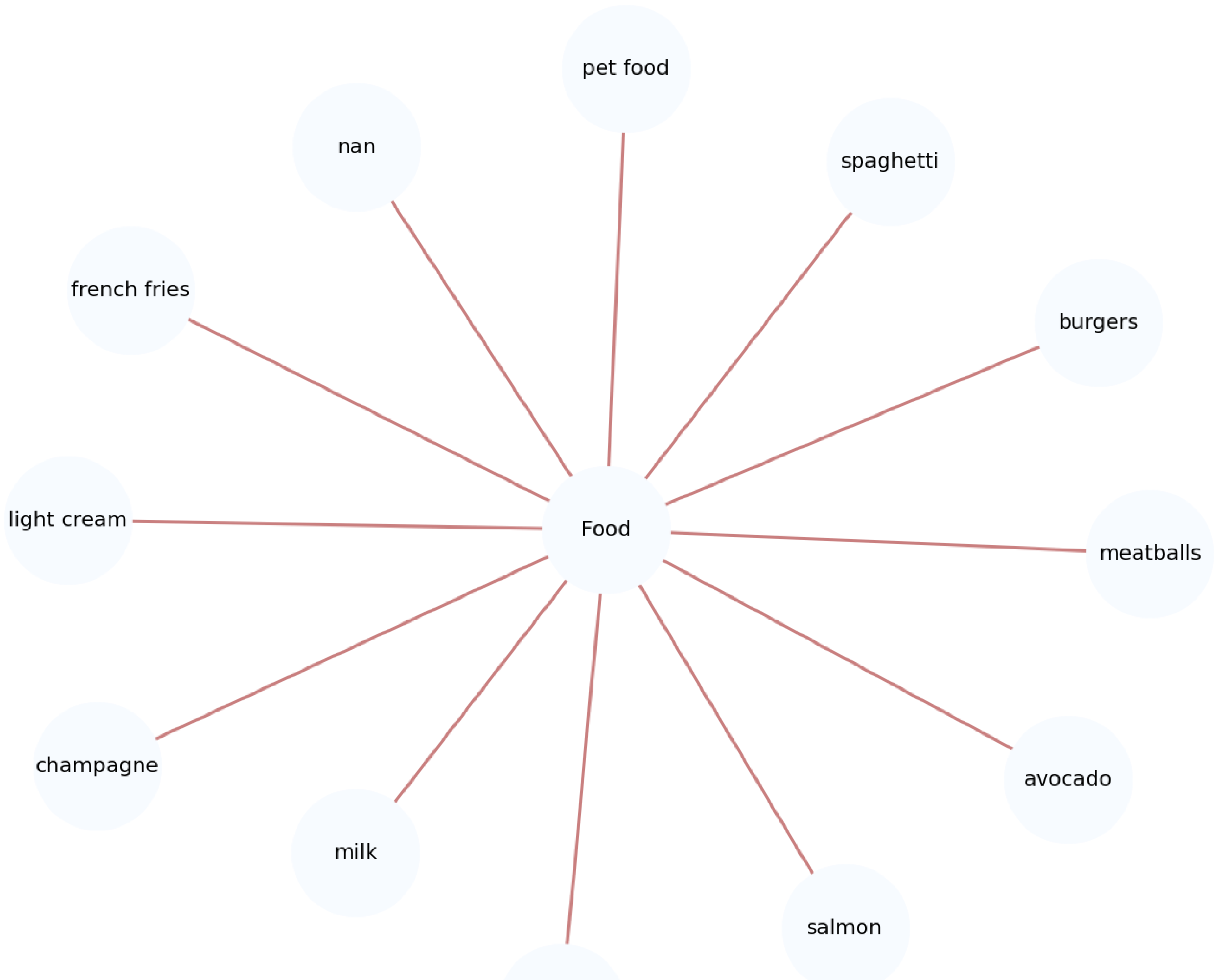
Out[25]:

```
{'Food': array([0.0036716 , 0.00661285]),
 'almonds': array([-0.07372186, -1.        ]),
 'meatballs': array([ 0.94079314, -0.04432206]),
 nan: array([-0.42740855,  0.81300829]),
 'avocado': array([ 0.80002361, -0.51825204]),
 'milk': array([-0.4279372 , -0.67362954]),
 'french fries': array([-0.81565396,  0.51057963]),
 'light cream': array([-0.92437157,  0.02626485]),
 'spaghetti': array([0.49354618, 0.78074858]),
 'pet food': array([0.03797973, 0.97753847]),
 'burgers': array([0.85123077, 0.44367474]),
 'champagne': array([-0.8735947 , -0.49105485]),
 'salmon': array([ 0.41544281, -0.83116893])}
```

In [26]:

```
plt.rcParams['figure.figsize'] = (20, 20)
color = plt.cm.Blues(np.linspace(0, 15, 1))
nx.draw_networkx_nodes(secondchoice, pos, node_size = 15000, node_color = color)
nx.draw_networkx_edges(secondchoice, pos, width = 3, alpha = 0.6, edge_color = 'brown')
nx.draw_networkx_labels(secondchoice, pos, font_size = 20, font_family = 'sans-serif')
plt.axis('off')
plt.grid()
plt.title('Top 15 Second Choices', fontsize = 40)
plt.show()
```

Top 15 Second Choices



almonds

In [27]:
```python
data['thirdchoice'] = 'Third Choice'
thirdchoice = data.truncate(before = -1, after = 10)
thirdchoice
```

Out[27]:

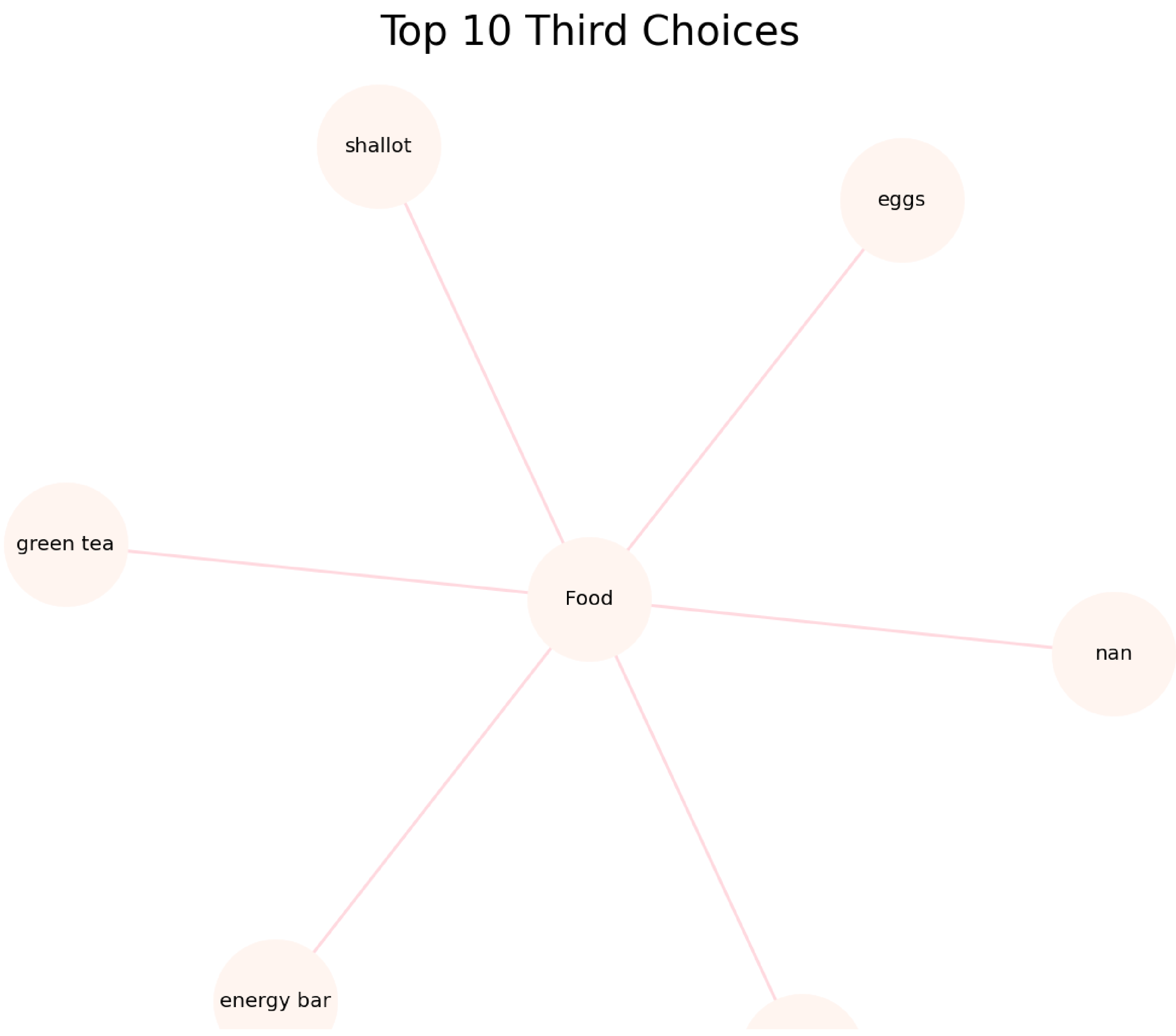
	0	1	2	3	4	5	6	7	8	9	...	13	14	15	16	17	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	...	salad	mineral water	salmon	antioxydant juice	frozen smoothie	sp
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
5	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
	whole																

In [28]:
```python
thirdchoice = nx.from_pandas_edgelist(thirdchoice, source = 'food', target = 2, edge_attr = True) # Here Target i
pos = nx.spring_layout(thirdchoice)
pos
```

Out[28]:
```
{'Food': array([-0.00038113,  0.00248743]),
 'avocado': array([ 0.40475569, -0.92399341]),
 'eggs': array([0.59651032, 0.81056829]),
 nan: array([ 1.        , -0.10904994]),
 'energy bar': array([-0.59883628, -0.81272252]),
 'shallot': array([-0.40249825,  0.92006306]),
 'green tea': array([-0.99955034,  0.11264709])}
```

In [29]:

```
plt.rcParams['figure.figsize'] = (20, 20)
color = plt.cm.Reds(np.linspace(0, 15, 1))
nx.draw_networkx_nodes(thirdchoice, pos, node_size = 15000, node_color = color)
nx.draw_networkx_edges(thirdchoice, pos, width = 3, alpha = 0.6, edge_color = 'pink')
nx.draw_networkx_labels(thirdchoice, pos, font_size = 20, font_family = 'sans-serif')
plt.axis('off')
plt.grid()
plt.title('Top 10 Third Choices', fontsize = 40)
plt.show()
```


Top 10 Third Choices
shallot
eggs
green tea
Food
nan
energy bar

Data Preprocessing

- The dataset contains the items bought by a customer i.e. each row represents one customer.
- Converting the dataframe into a list of lists, as required by the apriori algorithm.

In [30]:
```python
# making each customers shopping items an identical list
transactions = []
for i in range(0, 7501):
    transactions.append([str(data.values[i,j]) for j in range(0, 20)])

print(transactions)
```

```
[['shrimp', 'almonds', 'avocado', 'vegetables mix', 'green grapes', 'whole weat flour', 'yams', 'cottage cheese', 
'energy drink', 'tomato juice', 'low fat yogurt', 'green tea', 'honey', 'salad', 'mineral water', 'salmon', 'antio
xydant juice', 'frozen smoothie', 'spinach', 'olive oil'], ['burgers', 'meatballs', 'eggs', 'nan', 'nan', 'nan', 
'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan'], ['chutney', 'na
n', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'na
n', 'nan', 'nan'], ['turkey', 'avocado', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'na
n', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan'], ['mineral water', 'milk', 'energy bar', 'whole wheat rice', 
'green tea', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'na
n'], ['low fat yogurt', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 
'nan', 'nan', 'nan', 'nan', 'nan', 'nan'], ['whole wheat pasta', 'french fries', 'nan', 'nan', 'nan', 'nan', 'na
n', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan'], ['soup', 'light cr
eam', 'shallot', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 
'nan', 'nan', 'nan'], ['frozen vegetables', 'spaghetti', 'green tea', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'n
an', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan'], ['french fries', 'nan', 'nan', 'nan', 
'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan'], 
['eggs', 'pet food', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'n
an', 'nan', 'nan', 'nan', 'nan'], ['cookies', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'na
n', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan'], ['turkey', 'burgers', 'mineral water', 'eggs', 
'cooking oil', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 
```

In [31]:
```
print("First Transaction:\n")
print(transactions[:1])
print("\nSecond Transaction:\n")
print(transactions[1:2])
```

```
First Transaction:

[['shrimp', 'almonds', 'avocado', 'vegetables mix', 'green grapes', 'whole weat flour', 'yams', 'cottage cheese', 'e
nergy drink', 'tomato juice', 'low fat yogurt', 'green tea', 'honey', 'salad', 'mineral water', 'salmon', 'antioxyda
nt juice', 'frozen smoothie', 'spinach', 'olive oil']]

Second Transaction:

[['burgers', 'meatballs', 'eggs', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'na
n', 'nan', 'nan', 'nan', 'nan', 'nan']]
```

In [32]:
```
# conveting it into an numpy array
transactions = np.array(transactions)

# checking the shape of the array
print(transactions.shape)
```

```
(7501, 20)
```

In [33]:
```
transactions
```

Out[33]:
```
array([['shrimp', 'almonds', 'avocado', ..., 'frozen smoothie',
        'spinach', 'olive oil'],
       ['burgers', 'meatballs', 'eggs', ..., 'nan', 'nan', 'nan'],
       ['chutney', 'nan', 'nan', ..., 'nan', 'nan', 'nan'],
       ...,
       ['chicken', 'nan', 'nan', ..., 'nan', 'nan', 'nan'],
       ['escalope', 'green tea', 'nan', ..., 'nan', 'nan', 'nan'],
       ['eggs', 'frozen smoothie', 'yogurt cake', ..., 'nan', 'nan',
        'nan']], dtype='<U20')
```

In [34]:

```python
# create an encoder for apriori algorithm
encoder = TransactionEncoder()

# fit and Tranform encoder to transactions which will create a one-hot encoded(True, Flase) occurence matrix
data = encoder.fit_transform(transactions)

# convert array to pandas dataframe
data = pd.DataFrame(data, columns = encoder.columns_)

data.head(10)
```

Out[34]:

	asparagus	almonds	antioxydant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	...	turkey	vegetables mix	water spray	white wine
0	False	True	True	False	True	False	False	False	False	False	...	False	True	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False
3	False	False	False	False	True	False	False	False	False	False	...	True	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False
8	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False

In [35]:

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7501 entries, 0 to 7500
Columns: 121 entries,  asparagus to zucchini
dtypes: bool(121)
memory usage: 886.5 KB
```

In [36]:
```python
# getting the shape of the data
data.shape
```

Out[36]: (7501, 121)

In [37]:
```python
# getting correlations for 121 items would be messy
# so let's reduce the items from 121 to 40

data = data.loc[:, ['mineral water', 'burgers', 'turkey', 'chocolate', 'frozen vegetables', 'spaghetti',
                    'shrimp', 'grated cheese', 'eggs', 'cookies', 'french fries', 'herb & pepper', 'ground beef',
                    'tomatoes', 'milk', 'escalope', 'fresh tuna', 'red wine', 'ham', 'cake', 'green tea',
                    'whole wheat pasta', 'pancakes', 'soup', 'muffins', 'energy bar', 'olive oil', 'champagne',
                    'avocado', 'pepper', 'butter', 'parmesan cheese', 'whole wheat rice', 'low fat yogurt',
                    'chicken', 'vegetables mix', 'pickles', 'meatballs', 'frozen smoothie', 'yogurt cake']]

```

In [38]:
```
data
```

Out[38]:

	mineral water	burgers	turkey	chocolate	frozen vegetables	spaghetti	shrimp	grated cheese	eggs	cookies	...	butter	parmesan cheese	whole wheat rice	low fat yogurt	chicken
0	True	False	False	False	False	False	True	False	False	False	...	False	False	False	True	False
1	False	True	False	False	False	False	False	False	True	False	...	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
3	False	False	True	False	False	False	False	False	False	False	...	False	False	False	False	False
4	True	False	False	False	False	False	False	False	False	False	...	False	False	True	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7496	False	False	False	False	False	False	False	False	False	False	...	True	False	False	False	False
7497	False	True	False	False	True	False	False	False	True	False	...	False	False	False	False	False
7498	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	True
7499	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
7500	False	False	False	False	False	False	False	False	True	False	...	False	False	False	True	False

7501 rows × 40 columns

In [39]:
```
# checking the shape
data.shape
```

Out[39]: (7501, 40)

Model Training

In [40]:

```
#Now, let us return the items and itemsets with at least 3% support:
apriori(data, min_support = 0.03, use_colnames = True)
```

Out[40]:

	support	itemsets
0	0.238368	(mineral water)
1	0.087188	(burgers)
2	0.062525	(turkey)
3	0.163845	(chocolate)
4	0.095321	(frozen vegetables)
5	0.174110	(spaghetti)
6	0.071457	(shrimp)
7	0.052393	(grated cheese)
8	0.179709	(eggs)
9	0.080389	(cookies)
10	0.170911	(french fries)

In [41]:

```
#Now, let us return the items and itemsets with at least 5% support:
apriori(data, min_support = 0.05, use_colnames = True)
```

Out[41]:

	support	itemsets
0	0.238368	(mineral water)
1	0.087188	(burgers)
2	0.062525	(turkey)
3	0.163845	(chocolate)
4	0.095321	(frozen vegetables)
5	0.174110	(spaghetti)
6	0.071457	(shrimp)
7	0.052393	(grated cheese)
8	0.179709	(eggs)
9	0.080389	(cookies)
10	0.170911	(french fries)

In [42]:

```python
frequent_itemsets = apriori(data, min_support = 0.05, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```

Out[42]:

	support	itemsets	length
0	0.238368	(mineral water)	1
1	0.087188	(burgers)	1
2	0.062525	(turkey)	1
3	0.163845	(chocolate)	1
4	0.095321	(frozen vegetables)	1
5	0.174110	(spaghetti)	1
6	0.071457	(shrimp)	1
7	0.052393	(grated cheese)	1
8	0.179709	(eggs)	1
9	0.080389	(cookies)	1
10	0.170911	(french fries)	1

In [43]:
```
frequent_itemsets.sort_values('support', ascending = False)
```

Out[43]:

	support	itemsets	length
0	0.238368	(mineral water)	1
8	0.179709	(eggs)	1
5	0.174110	(spaghetti)	1
10	0.170911	(french fries)	1
3	0.163845	(chocolate)	1
16	0.132116	(green tea)	1
13	0.129583	(milk)	1
11	0.098254	(ground beef)	1
4	0.095321	(frozen vegetables)	1
17	0.095054	(pancakes)	1
1	0.087188	(burgers)	1

In [44]:
```
Asso_Rules = association_rules(frequent_itemsets, metric = "lift", min_threshold = 1)
Asso_Rules.sort_values('lift',ascending = False)
```

Out[44]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	(spaghetti)	(mineral water)	0.174110	0.238368	0.059725	0.343032	1.439085	0.018223	1.159314
3	(mineral water)	(spaghetti)	0.238368	0.174110	0.059725	0.250559	1.439085	0.018223	1.102008
1	(chocolate)	(mineral water)	0.163845	0.238368	0.052660	0.321400	1.348332	0.013604	1.122357
0	(mineral water)	(chocolate)	0.238368	0.163845	0.052660	0.220917	1.348332	0.013604	1.073256
4	(mineral water)	(eggs)	0.238368	0.179709	0.050927	0.213647	1.188845	0.008090	1.043158
5	(eggs)	(mineral water)	0.179709	0.238368	0.050927	0.283383	1.188845	0.008090	1.062815

In [45]:
```python
# getting the item sets with length = 2 and support more than 1%

frequent_itemsets[ (frequent_itemsets['length'] == 2) &
                   (frequent_itemsets['support'] >= 0.01) ]
```

Out[45]:

	support	itemsets	length
24	0.052660	(mineral water, chocolate)	2
25	0.059725	(spaghetti, mineral water)	2
26	0.050927	(mineral water, eggs)	2

In [46]:
```python
# getting the item sets with length = 1 and support more than 10%

frequent_itemsets[ (frequent_itemsets['length'] == 1) &
                   (frequent_itemsets['support'] >= 0.1) ]
```

Out[46]:

	support	itemsets	length
0	0.238368	(mineral water)	1
3	0.163845	(chocolate)	1
5	0.174110	(spaghetti)	1
8	0.179709	(eggs)	1
10	0.170911	(french fries)	1
13	0.129583	(milk)	1
16	0.132116	(green tea)	1

In [47]:
```python
frequent_itemsets[ frequent_itemsets['itemsets'] == {'eggs', 'mineral water'} ]
```

Out[47]:

	support	itemsets	length
26	0.050927	(mineral water, eggs)	2

In [48]:
```python
frequent_itemsets[ frequent_itemsets['itemsets'] == {'chocolate'} ]
```

Out[48]:

	support	itemsets	length
3	0.163845	(chocolate)	1

In [ ]:
```python

```