Importing Libraries

In [1]:
```
from nltk.corpus import wordnet as wn
import http.client
import json
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import numpy as np
from sklearn import preprocessing

import warnings
warnings.filterwarnings('ignore')
```

Task 1 : Data Identification : Identify one or more suitable web APIs¶

The API chosen for this assignment is **Food Calorie Data Search**. The API is available freely over Rapid API website. Therefore, we can easily download calorie data for a food item one at a time. The link to access the dataset is: https://rapidapi.com/kenpi04/api/food-calorie-data-search/endpoints (https://rapidapi.com/kenpi04/api/food-calorie-data-search/endpoints)
We need to login at Rapid API website which can give us api-key to play around the editor with API end-points.

Task 2 : Data Collection : Calling API and getting data for selective fruits

The API can get the data for each food item one at a time. Therefore, we need to call the values for one food item at a time. I chose to get the data of fruits.
For this I used a Fruit corpus available over nltk.corpus wordnet. From that corpus, I manually chose 12 different fruits and then made analysis over those values.
The data returned is of a particular food item contained in a group of meals. 10 records are returned per api call.

Taking Fruit data from corpus

In [2]:

```
# extracting fruit data from wordnet synset data
fruits = wn.synset('fruit.n.01')
# getting list of fruits from lemma
fruit_data = list(set([w for s in fruits.closure(lambda s:s.hyponyms()) for w
in s.lemma_names()]))

# for each record in the list remove special characters like ' and _ to get on
ly words
for i in fruit_data:
    if "'" in i:
        fruit_data.remove(i)
    if '_' in i:
        i = i.replace('_',' ')

#replace _ and ' with blank
fruit_data = [i.replace('_',' ') for i in fruit_data]
fruit_data = [i.replace("'",' ') for i in fruit_data]

# printing the fruit words
print(fruit_data)
```

['ketembilla', 'genip', 'schizocarp', 'marasca', 'honeydew melon', 'saskatoo
n', 'hog plum', 'May apple', 'quantong', 'sweet calabash', 'chincapin', 'acin
us', 'citrus fruit', 'cherimolla', 'dewberry', 'groundnut', 'mammee apple',
'coquilla nut', 'morello', 'beach plum', 'legume', 'Winesap', 'marmalade plu
m', 'mango', 'netted melon', 'kola nut', 'palm nut', 'Stayman Winesap', 'avoc
ado', 'caryopsis', 'chokecherry', 'seckel', 'sapodilla plum', 'jack', 'grai
n', 'cobnut', 'pond apple', 'currant', 'sour cherry', 'key lime', 'coffee',
'cumin seed', 'Pippin', 'bosc', 'shaddock', 'seckel pear', 'Seville orange',
'mangosteen', 'sunflower seed', 'gooseberry', 'algarobilla', 'hickory nut',
'vinifera grape', 'castor bean', 'cowpea', 'flame tokay', 'sweetsop', 'star f
ruit', 'kai apple', 'dika nut', 'Mexican black cherry', 'alligator pear', 'bo
ysenberry', 'pineapple guava', 'jujube', 'Baldwin', 'safflower seed', 'Chines
e gooseberry', 'cashew nut', 'wintergreen', 'edible seed', 'casaba', 'grapefr
uit', 'pseudocarp', 'linseed', 'rose hip', 'sapodilla', 'berry', 'pomelo', 'b
unya bunya', 'cranberry', 'aguacate', 'drupelet', 'coffee bean', 'ceriman',
'tonka bean', 'buckthorn berry', 'pea', 'scuppernong', 'date', 'West Indian c
herry', 'Northern Spy', 'crabapple', 'aggregate fruit', 'rambutan', 'souari n
ut', 'durian', 'black cherry', 'cantaloup', 'ear', 'lychee', 'persimmon', 'cu
min', 'anchovy pear', 'flaxseed', 'quince', 'blueberry', 'simple fruit', 'str
awberry', 'sapota', 'mombin', 'nicker seed', 'barbados cherry', 'kiwi fruit',
'teaberry', 'sultana', 'lansat', 'syconium', 'goober', 'longanberry', 'calaba
r bean', 'soy', 'litchee', 'almond', 'tangerine', 'jackfruit', 'eggfruit', 's
ugarberry', 'granadilla', 'emperor', 'papaw', 'rambotan', 'red currant', 'kum
quat', 'soybean', 'Newtown Wonder', 'green olive', 'yellow berry', 'pulasan',
'cob', 'rosehip', 'oil-rich seed', 'freestone', 'litchi', 'corn', 'slipskin g
rape', 'eating apple', 'cling', 'algarrobilla', 'tangelo', 'coffee berry', 'n
ative peach', 'blackberry', 'cashew', 'beechnut', 'citron', 'avocado pear',
'conker', 'seeded raisin', 'blue fig', 'Tokay', 'breadfruit', 'palm kernel',
'black-eyed pea', 'raspberry', 'damson', 'lichi', 'sorb apple', 'key', 'pumpk
in seed', 'Macoun', 'cohune nut', 'genipap fruit', 'ash-key', 'cocoa plum',
'citrus', 'melon ball', 'mammee', 'mountain cranberry', 'capitulum', 'shadber
ry', 'clingstone', 'blade apple', 'lingonberry', 'hazelnut', 'damson plum',
'garbanzo', 'areca nut', 'false fruit', 'greengage plum', 'McIntosh', 'leeche
e', 'muscat grape', 'tamarindo', 'syncarp', 'Barbados gooseberry', 'vegetable
ivory', 'edible fruit', 'net melon', 'stone fruit', 'grugru nut', 'banana',
'wheat berry', 'fox grape', 'loganberry', 'raisin', 'coumara nut', 'river pea
r', 'bacca', 'cowage', 'lanset', 'peanut', 'bell apple', 'medlar', 'passion f
ruit', 'Jonathan', 'acerola', 'mast', 'amaranth', 'nutlet', 'macadamia nut',
'sweet melon', 'mandarin orange', 'canistel', 'neem seed', 'casaba melon', 'l
oment', 'black currant', 'pistachio', 'Stayman', 'bilberry', 'chinkapin', 'Th
ompson Seedless', 'grape', 'fruitlet', 'divi-divi', 'black olive', 'pip', 'co
ttonseed', 'filbert', 'wild cherry', 'bartlett', 'barleycorn', 'horse chestnu
t', 'juneberry', 'pear', 'orange', 'fava bean', 'guava', 'Granny Smith', 'cra
b apple', 'Victoria plum', 'samara', 'prune', 'multiple fruit', 'yellow mombi
n', 'brazil', 'Valencia orange', 'carambola', 'oil nut', 'temple orange', 'cl
ementine', 'ivory nut', 'achene', 'garambulla', 'water lemon', 'pinon nut',
'soya bean', 'Red Delicious', 'guanabana', 'prickly pear', 'rye', 'ilama', 'c
owberry', 'sweet cup', 'elk nut', 'amarelle', 'capulin', 'English walnut', 'h
ip', 'prairie gourd', 'ribier', 'cherimoya', 'watermelon', 'bonduc nut', 'che
stnut', 'ordeal bean', 'walnut', 'cocoanut', 'Spanish lime', 'ugli fruit', 'p
apaya', 'melon', 'sugar apple', 'carissa plum', 'surinam cherry', 'cubeb', 's
our orange', 'pignolia', 'citrange', 'field pea', 'hackberry', 'akee', 'jabot
icaba', 'pomegranate', 'Cortland', 'candlenut', 'genipap', 'Concord grape',
'apple', 'monkey bread', 'cedar nut', 'mamey', 'windfall', 'Jaffa orange', 'l
ocust pod', 'pitahaya', 'dried fruit', 'gourd', 'lansa', 'honeydew', 'Chinese
date', 'goober pea', 'pawpaw', 'pecan', 'Yellow Delicious', 'monkey nut', 'qu
andang', 'Mexican jumping bean', 'betel nut', 'seedless raisin', 'lemon', 'Pe

```
armain', 'bean', 'muscatel', 'jumping seed', 'cherry', 'icaco', 'checkerberr
y', 'algarroba bean', 'satsuma', 'key fruit', 'kitambilla', 'heart cherry',
'spiceberry', 'muscadine', 'Catawba', 'Prima', 'butternut', 'apricot', 'buffa
lo nut', 'locust bean', 'carob', 'apple nut', 'yellow granadilla', 'nut', 'co
conut', 'mandarin', 'accessory fruit', 'mulberry', 'sorb', 'cola nut', 'boxbe
rry', 'kiwi', 'oxheart', 'ananas', 'bing cherry', 'navel orange', 'custard ap
ple', 'lime', 'buckeye', 'sweet orange', 'feijoa', 'serviceberry', 'ripe oliv
e', 'anjou', 'chickpea', 'muscat', 'pistachio nut', 'jak', 'acorn', 'Japanese
plum', 'brazil nut', 'marang', 'Rome Beauty', 'rose apple', 'mealie', 'nicker
nut', 'pyrene', 'kitembilla', 'bullace grape', 'lichee', 'huckleberry', 'caro
b bean', 'plum', 'fig', 'earthnut', 'seedpod', 'litchi nut', 'whortleberry',
'quandong nut', 'blackheart cherry', 'calabash', 'quandong', 'pod', 'juniper
berry', 'pyxis', 'winter melon', 'loquat', 'seed', 'Golden Delicious', 'pom
e', 'ackee', 'ugli', 'broad bean', 'nutmeg melon', 'jumping bean', 'screw bea
n', 'lowbush cranberry', 'water chinquapin', 'sweet cherry', 'baneberry', 'so
ursop', 'oilseed', 'lentil', 'monstera', 'lanseh', 'plumcot', 'rowanberry',
'nectarine', 'pyxidium', 'bitter orange', 'hagberry', 'peach', 'babassu nut',
'edible nut', 'Persian melon', 'Empire', 'muskmelon', 'European blueberry',
'rapeseed', 'horsebean', 'black walnut', 'wild plum', 'drupe', 'Delicious',
'bartlett pear', 'kernel', 'olive', 'coco plum', 'blackheart', 'pine nut', 's
loe', 'sapote', 'dessert apple', 'dried apricot', 'cembra nut', 'natal plum',
'annon', 'citrous fruit', 'elderberry', 'spike', 'cooking apple', 'Jamaica ap
ple', 'tamarind', 'pulassan', 'okra', 'greengage', 'garden pea', 'oxheart che
rry', 'pineapple', 'Jordan almond', 'algarroba', 'Chinese jujube', 'chinquapi
n', 'sour gourd', 'cantaloupe']
```

Note: Run entire notebook in one go if you face error like below:

TypeError: 'list' object is not callable

Connecting to API and getting data for selective fruits

In [3]:
```
# making connection request to the domain
conn = http.client.HTTPSConnection("food-calorie-data-search.p.rapidapi.com")

#setting headers with host and key values - provided after logging in rapidapi
headers = {
    'x-rapidapi-host': "food-calorie-data-search.p.rapidapi.com",
    'x-rapidapi-key': "5bf192c5cemsh6925a9e00884d04p102f48jsn05bc359e0d8f"
    }

#list to store results
results=[]

#list of selective fruits chosen from the Fruit corpus above
list=['hazelnut','cherry','muskmelon','bean','raspberry','chickpea','coffee be
an','litchi','peach','butternut','olive','pear','strawberry',\
      'sweet melon','melon']

#iterate through each word and call the food-calorie api for each
for str_ in list:
    #calling the api with each fruit value one by one
    conn.request("GET", "/api/search?keyword="+str_, headers=headers)
    res = conn.getresponse()
    #reading data from the response
    data = res.read()
    data = data.decode("utf-8")
    #removing all special characters and empty data and storing in json format
    if data not in '' and data not in '[]':
        data = json.loads(data)
        for i in data:
            i['fruit_name']=str_
        results.append(data)
```

Making list of Dataframes

The data we have now is in form of List of JSON objects, we need to convert each JSON object to dataframe. Thus, finally we get List of Dataframes.

In [4]:
```
df = pd.DataFrame()

list_of_df=[]

#for each JSON object, convert to data frame and store in list_of_df list
for i in results:
    df = pd.DataFrame(i)
    list_of_df.append(df)
```

Saving Data in CSV file

In [5]:
```
csv_data = pd.DataFrame()

#for each data frame in list, store in a common data frame, then store this fi
le to csv
for i in list_of_df:
    csv_data = csv_data.append(i, ignore_index=True)

csv_data.to_csv('fruits_data.csv')
```

Task 3 : Data preparation and analysis:

Pre-processing Data

We would have a list of dataframes where each dataframe looks like the following:

In [6]:
```
list_of_df[0]
```

Out[6]:

	id	ndb_no	shrt_desc	water	energ_kcal	protein	lipid_tot	ash
0	1353	4532	OIL,HAZELNUT	0.00	884	0.00	100.00	0.00
1	2527	8689	CEREALS,QUAKER,OATMEAL,REAL MEDLEYS,BLUEBERRY ...	7.47	386	9.81	9.80	3.46
2	2995	19125	CHOCOLATE-FLAVORED HAZELNUT SPRD	1.07	541	5.41	29.73	1.44
3	5105	35233	HAZELNUTS,BEAKED (NORTHERN PLAINS INDIANS)	5.92	628	14.89	52.99	3.22
4	6486	12120	HAZELNUTS OR FILBERTS	5.31	628	14.95	60.75	2.29
5	6487	12121	HAZELNUTS OR FILBERTS,BLANCHED	5.79	629	13.70	61.15	2.36
6	6488	12122	HAZELNUTS OR FILBERTS,DRY RSTD,WO/SALT	2.52	646	15.03	62.40	2.45
7	8248	16262	SILK HAZELNUT CREAMER	72.95	133	0.00	6.67	0.38

8 rows × 55 columns

Extracting specific columns from the dataset

The data we currently recieved have many irrelevant and unneccessay fields like - id, ndb_no, short description, gram weight etc. Those are difficult to understand, therefore, we would consider only specific fields which we can analyze and draw patterns from.

```
In [7]: # considered only the fields which makes sense and are important for us
for i in range(len(list_of_df)):
    fruit = list_of_df[i]
    list_of_df[i] = fruit[["fruit_name","calcium","carbohydrt","copper","energ_kcal","iron","magnesium","manganese","phosphorus",\
                    "protein","selenium","sodium","vit_b12","vit_b6","vit_c","vit_e","vit_k","water","zinc"]]
```

Now, we have a list of dataframes like the following:

```
In [8]: list_of_df[0]
```

Out[8]:

	fruit_name	calcium	carbohydrt	copper	energ_kcal	iron	magnesium	manganese	phospho
0	hazelnut	0	0.00	0.000	884	0.00	0	0.000	
1	hazelnut	64	69.47	0.000	386	3.08	98	0.000	
2	hazelnut	108	62.16	0.469	541	4.38	64	0.868	
3	hazelnut	441	22.98	1.200	628	3.12	235	7.600	
4	hazelnut	114	16.70	1.725	628	4.70	163	6.175	
5	hazelnut	149	17.00	1.600	629	3.30	160	12.650	
6	hazelnut	123	17.60	1.750	646	4.38	173	5.550	
7	hazelnut	0	20.00	0.000	133	0.00	0	0.000	

Normalize Data

As the values of different fields are in varied range, we need to scale this to match all columns equally.
For this I have performed Min Max scaling from Sklearn pre-processing library.
All values would be recuded to minimal after performing this step.

In [9]:
```
for fruits in list_of_df:
    for k in fruits:
        if k not in 'fruit_name':
            x = fruits[k].values.astype(float) #returns a numpy array
            x = x.reshape(-1, 1)
            min_max_scaler = preprocessing.MinMaxScaler()
            x_scaled = min_max_scaler.fit_transform(x)
            fruits[k] = pd.DataFrame(x_scaled)

list_of_df[0]
```

Out[9]:

	fruit_name	calcium	carbohydrt	copper	energ_kcal	iron	magnesium	manganese	p
0	hazelnut	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	
1	hazelnut	0.145125	1.000000	0.000000	0.336884	0.655319	0.417021	0.000000	
2	hazelnut	0.244898	0.894775	0.268000	0.543276	0.931915	0.272340	0.068617	
3	hazelnut	1.000000	0.330790	0.685714	0.659121	0.663830	1.000000	0.600791	
4	hazelnut	0.258503	0.240392	0.985714	0.659121	1.000000	0.693617	0.488142	
5	hazelnut	0.337868	0.244710	0.914286	0.660453	0.702128	0.680851	1.000000	
6	hazelnut	0.278912	0.253347	1.000000	0.683089	0.931915	0.736170	0.438735	
7	hazelnut	0.000000	0.287894	0.000000	0.000000	0.000000	0.000000	0.000000	

Check if any Missing Data

There is no data missing in any of the dataframes.

In [10]:
```
missing_values=0

#for each of the fruit data in list, check if there are any missing values
for fruits in list_of_df:
    missing_values += fruits.isnull().sum()

missing_values.sum()
```

Out[10]:
```
0
```

In [11]:
```
missing_values=0

#for each of the fruit data in list, check if there are any NaN values
for fruits in list_of_df:
    missing_values += fruits.dtypes.value_counts()

missing_values
```

Out[11]:
```
float64    216
object      12
dtype: int64
```

There is no NaN or empty strings found in the data. Therefore, the data is clean and good to use.

Analyse and summarise the cleaned dataset

For fruit 'Bean' plot between different elements

In [12]:

```
#extract the data of "Bean" fruit from the list of data we have
bean = pd.DataFrame()

#Store the data of 'bean' fruit in a seperate dataframe
for i in list_of_df:
    if 'bean' == i['fruit_name'][0]:
        bean = i

bean
```

Out[12]:

	fruit_name	calcium	carbohydrt	copper	energ_kcal	iron	magnesium	manganese	ph
0	bean	0.38	0.652284	1.000000	0.468750	0.733333	0.891892	0.833333	
1	bean	0.88	0.783418	1.000000	0.864583	0.761905	0.891892	0.833333	
2	bean	1.00	0.764805	1.000000	1.000000	1.000000	1.000000	1.000000	
3	bean	0.32	0.291878	0.533333	0.510417	0.571429	0.513514	0.483333	
4	bean	0.62	1.000000	0.000000	0.822917	0.647619	0.000000	0.000000	
5	bean	0.44	0.503384	0.273333	0.364583	0.276190	0.513514	0.380000	
6	bean	0.00	0.210660	0.000000	0.510417	0.000000	0.000000	0.000000	
7	bean	0.06	0.000000	0.516667	0.000000	0.019048	0.459459	0.416667	
8	bean	0.30	0.050761	0.506667	0.177083	0.019048	0.459459	0.408333	
9	bean	0.00	0.210660	0.000000	0.510417	0.000000	0.000000	0.000000	

Describing the data of Bean Fruit

The describe function gives the values of different agggregation functions as shown below.

In [13]:
```
bean.describe()
```
Out[13]:

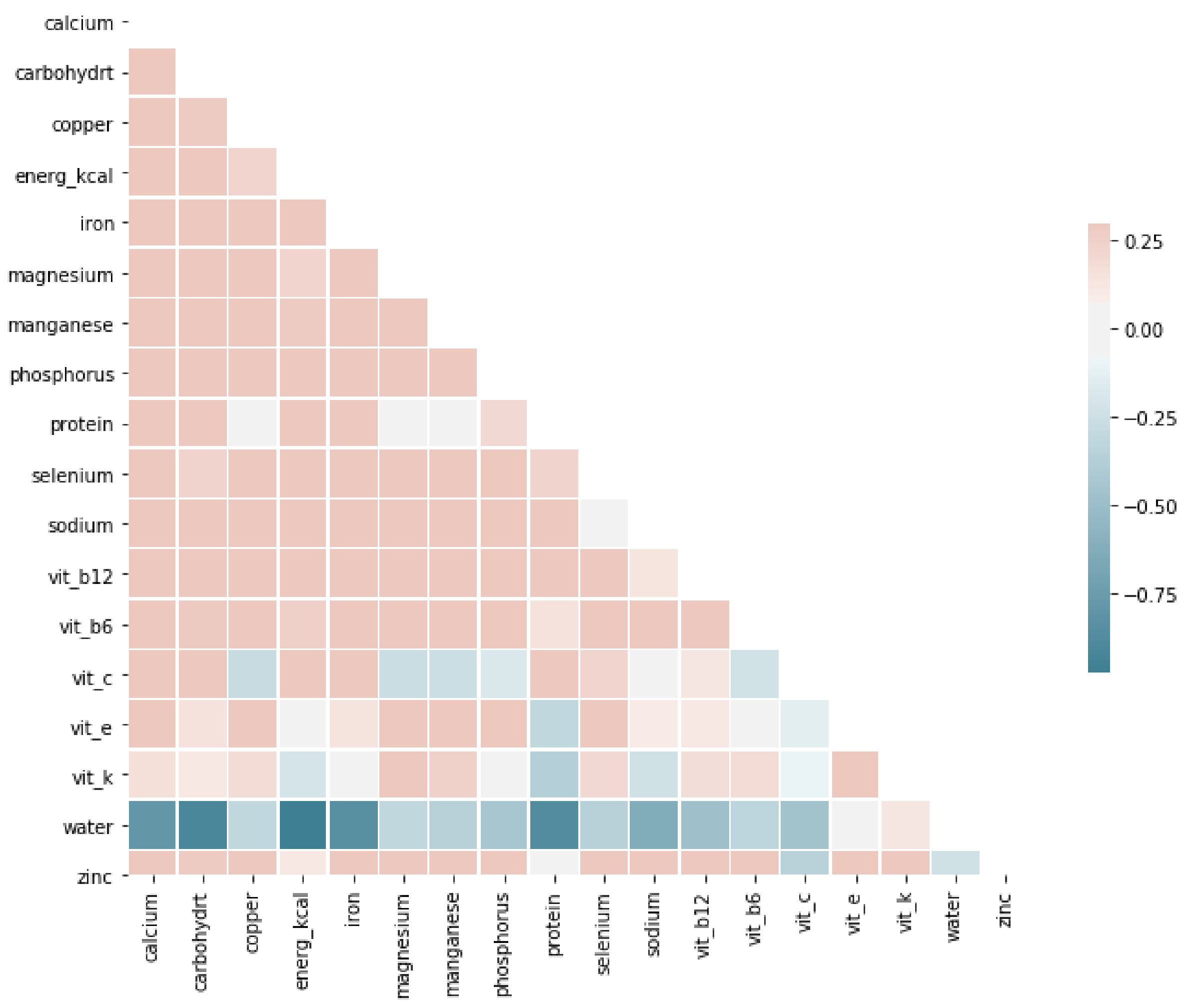
	calcium	carbohydrt	copper	energ_kcal	iron	magnesium	manganese	phospl
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.00
mean	0.400000	0.446785	0.483000	0.522917	0.402857	0.472973	0.435500	0.36
std	0.348457	0.342847	0.414986	0.308376	0.382722	0.380571	0.366764	0.35
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.120000	0.210660	0.068333	0.390625	0.019048	0.114865	0.095000	0.02
50%	0.350000	0.397631	0.511667	0.510417	0.423810	0.486486	0.412500	0.34
75%	0.575000	0.736675	0.883333	0.744792	0.711905	0.797297	0.745833	0.56
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00

Finding Correlation between elements of Bean

Construction of a heatmap which would help us find the approximate relation between data fields.
The more positive value of correlation coefficient represents direct proportionality and negative indicates inverse proportionality. The figure below represents cream color for positive correlation and blue shades for negative correlation. It is easier with the colors that we could tell that how much two fields are proportional to each other.

In [14]:
```
# variable to store correlation
corr = bean.corr()

# Hiding the upper triangle so that we get asymmetric triangle for data relati
on
mask = np.triu(np.ones_like(corr, dtype=np.bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)

# Drawing heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

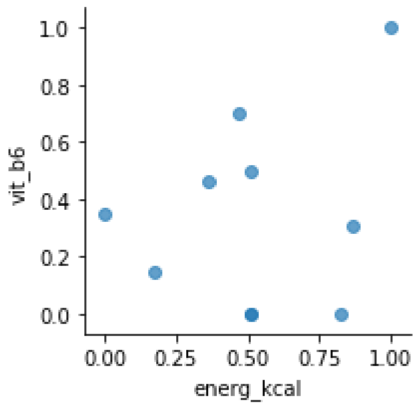
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1917f098198>



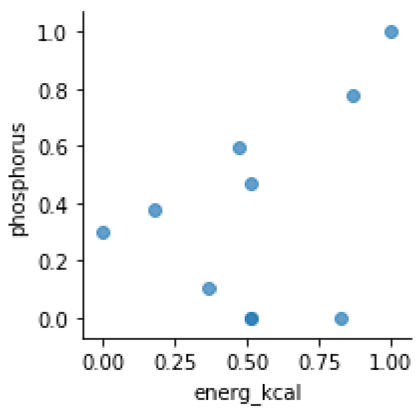
From the heatmap we can see that calcium, carbohydrate, copper, energy kcal, iron are related to many fields directly and have a quiet very positive relation.
Water is extremely inversely proportional to many fields like - Calcium, Carohydrate. Energy kilo-calorie, protein.

Finding relation between energy kilo-calories and different elements using Scatter Plots

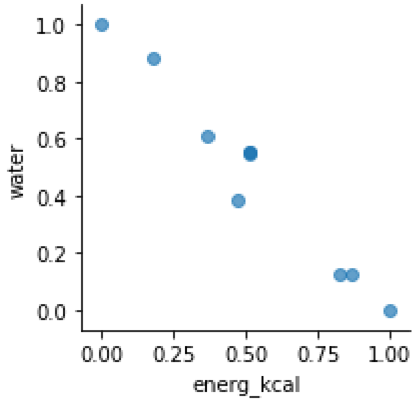
In [15]:
```
#plotting using seaborn Face Grid function
g = sns.FacetGrid(bean)
g.map(plt.scatter, "energ_kcal", "vit_b6", alpha=.7)
g.add_legend();
```



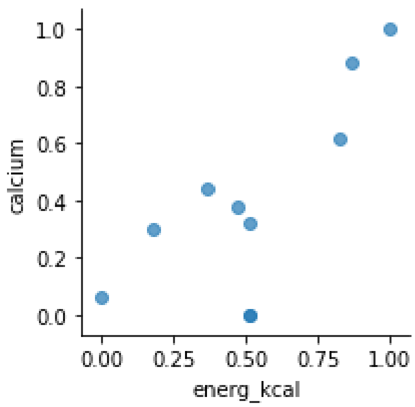
In [16]:
```
g = sns.FacetGrid(bean)
g.map(plt.scatter, "energ_kcal", "phosphorus", alpha=.7)
g.add_legend();
```



In [17]:
```
g = sns.FacetGrid(bean)
g.map(plt.scatter, "energ_kcal", "water", alpha=.7)
g.add_legend();
```

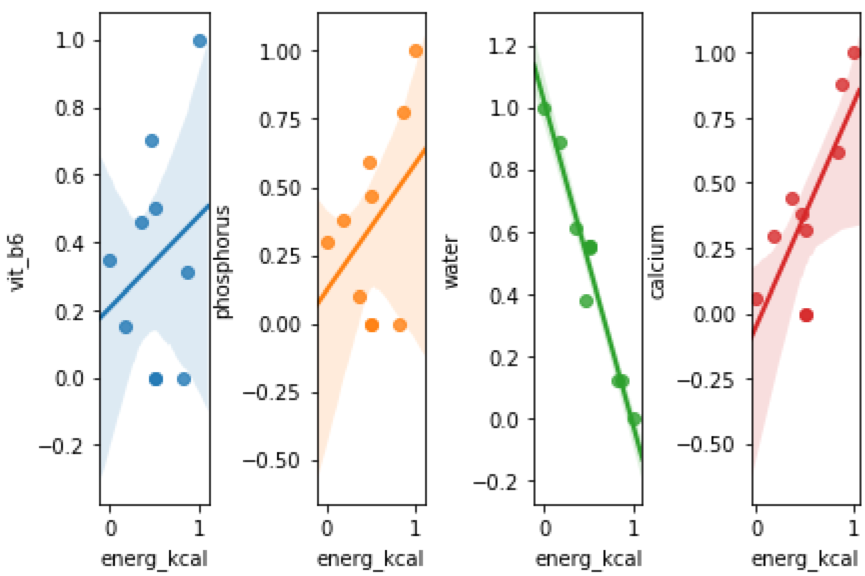


In [18]:
```
g = sns.FacetGrid(bean)
g.map(plt.scatter, "energ_kcal", "calcium", alpha=.7)
g.add_legend();
```



In [19]:
```
#finding relation comparison between Vitamin B6, Phosphorus, Water and Calcium
with Energy Kilo Calores
fig, axs = plt.subplots(ncols=4)
fig.tight_layout()
plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=1, hs
pace=None)
sns.regplot(x='energ_kcal', y='vit_b6', data=bean, ax=axs[0])
sns.regplot(x='energ_kcal', y='phosphorus', data=bean, ax=axs[1])
sns.regplot(x='energ_kcal',y='water', data=bean, ax=axs[2])
sns.regplot(x='energ_kcal',y='calcium', data=bean, ax=axs[3])
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1917f7e0828>

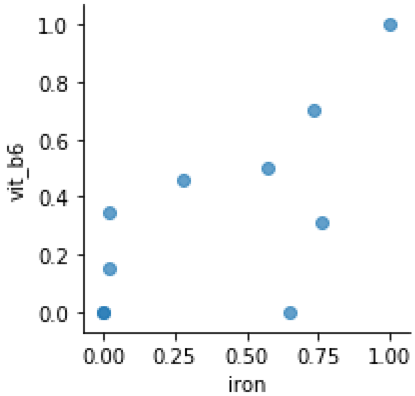


Plot Observations: Energy Kilo Calories is directly propoertional to Vitamin B6, Phosphorus, Calcium and inversely proportional to Water and Calcium when Bean is consumed with your meal.

Similarly, plotting for Iron vs other elements

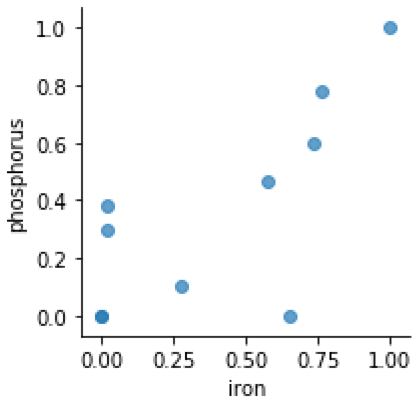
In [20]:

```
#plotting using seaborn Face Grid function
g = sns.FacetGrid(bean)
g.map(plt.scatter, "iron", "vit_b6", alpha=.7)
g.add_legend();
```



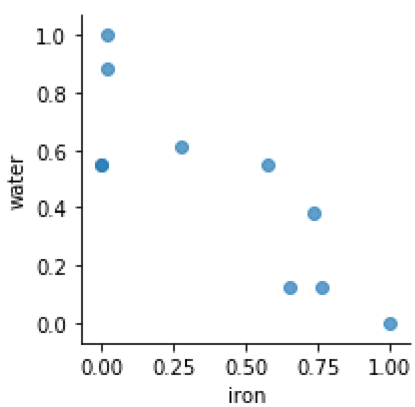
In [21]:

```
g = sns.FacetGrid(bean)
g.map(plt.scatter, "iron", "phosphorus", alpha=.7)
g.add_legend();
```

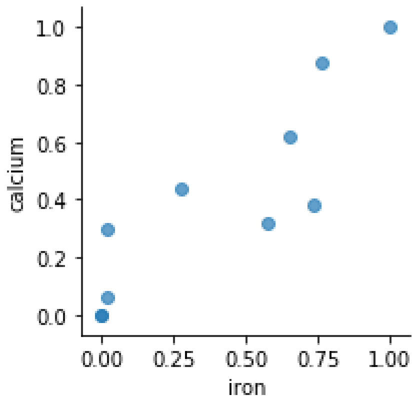


In [22]:

```
g = sns.FacetGrid(bean)
g.map(plt.scatter, "iron", "water", alpha=.7)
g.add_legend();
```

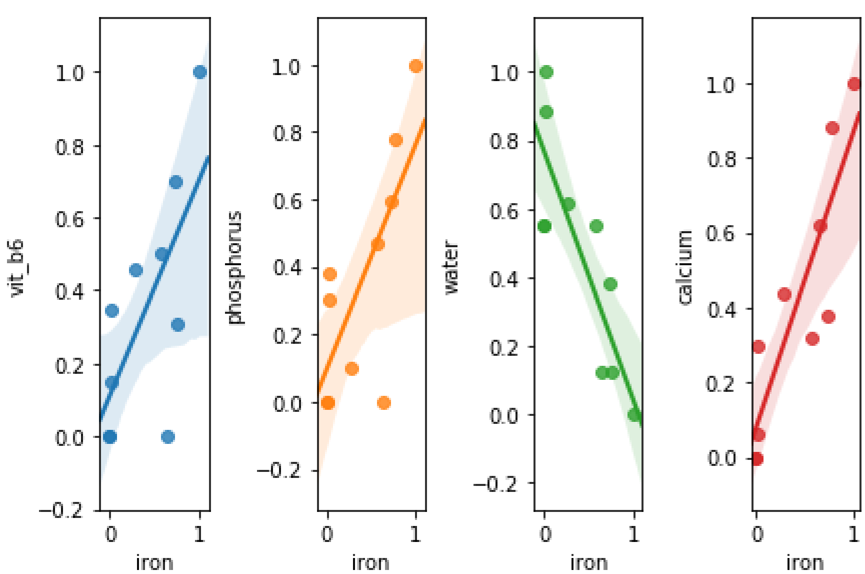


In [23]:
```
g = sns.FacetGrid(bean)
g.map(plt.scatter, "iron", "calcium", alpha=.7)
g.add_legend();
```



In [24]:
```
#finding relation comparison between Vitamin B6, Phosphorus, Water and Calcium with Energy Kilo Calores
fig, axs = plt.subplots(ncols=4)
fig.tight_layout()
plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=1, hspace=None)
sns.regplot(x='iron', y='vit_b6', data=bean, ax=axs[0])
sns.regplot(x='iron', y='phosphorus', data=bean, ax=axs[1])
sns.regplot(x='iron',y='water', data=bean, ax=axs[2])
sns.regplot(x='iron',y='calcium', data=bean, ax=axs[3])
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1917fad7ba8>
```



Plot Observations: Iron is directly proportional to Vitamin B6, Phosphorus and Calcium; and inversely proportional to Water when Bean is consumed in you meal.

For 'Peach' fruit plot between different elements

In [25]:
```
peach = pd.DataFrame()

for i in list_of_df:
    if 'peach' == i['fruit_name'][0]:
        peach = i

peach.describe()
```

Out[25]:

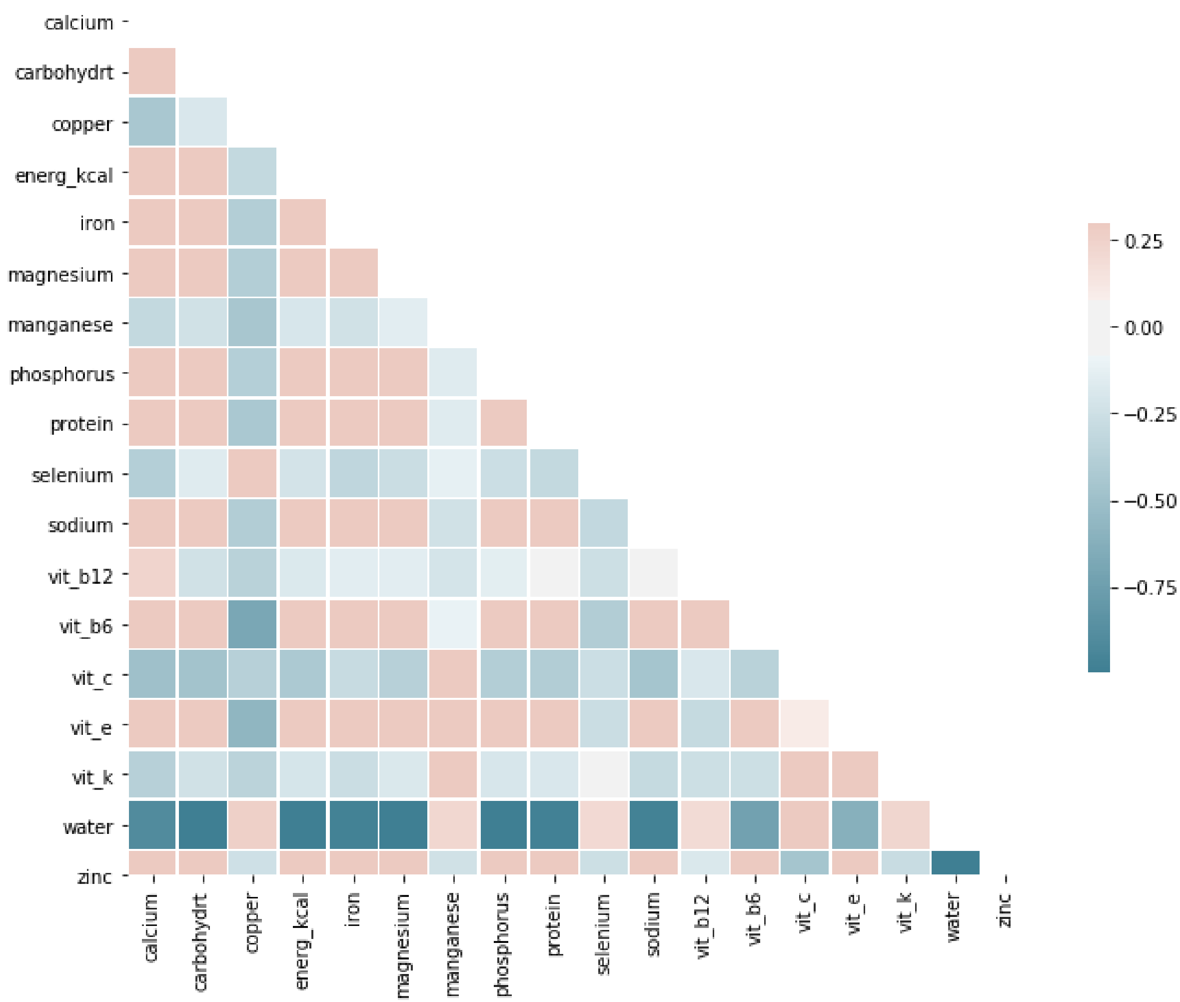
	calcium	carbohydrt	copper	energ_kcal	iron	magnesium	manganese	phosph
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.00
mean	0.158824	0.195612	0.442857	0.151429	0.148148	0.128696	0.248000	0.12
std	0.315991	0.287944	0.437758	0.299587	0.303433	0.307049	0.405539	0.30
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.009804	0.077403	0.153061	0.046429	0.023148	0.017391	0.000000	0.01
50%	0.019608	0.126212	0.170918	0.072857	0.048148	0.021739	0.000000	0.01
75%	0.075980	0.159655	0.906888	0.080000	0.082407	0.060870	0.555000	0.04
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00

Finding Correlation between elements of Peach

Construction of a heatmap which would help us find the approximate relation between data fields.
The more positive value of correlation coefficient represents direct proportionality and negative indicates inverse proportionality. The figure below represents cream color for positive correlation and blue shades for negative correlation. It is easier with the colors that we could tell that how much two fields are proportional to each other.

In [26]:
```python
corr = peach.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=np.bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x1917fbb30f0>



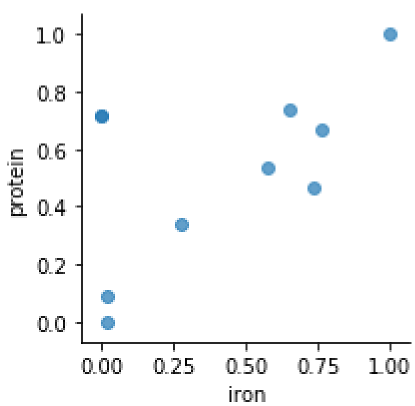
In [27]:
```
peach.describe()
```

Out[27]:

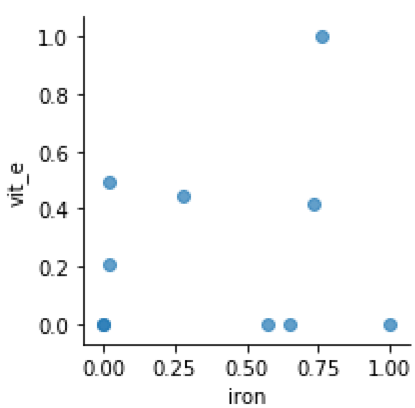
	calcium	carbohydrt	copper	energ_kcal	iron	magnesium	manganese	phospl
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.0(
mean	0.158824	0.195612	0.442857	0.151429	0.148148	0.128696	0.248000	0.12
std	0.315991	0.287944	0.437758	0.299587	0.303433	0.307049	0.405539	0.3(
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0(
25%	0.009804	0.077403	0.153061	0.046429	0.023148	0.017391	0.000000	0.01
50%	0.019608	0.126212	0.170918	0.072857	0.048148	0.021739	0.000000	0.01
75%	0.075980	0.159655	0.906888	0.080000	0.082407	0.060870	0.555000	0.04
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0(

Finding relation between iron and different elements using Scatter Plots

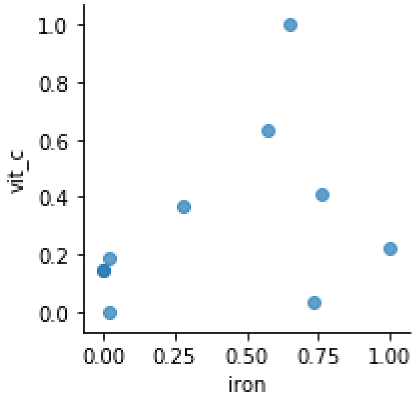
In [28]:
```
g = sns.FacetGrid(bean)
g.map(plt.scatter, "iron", "protein", alpha=.7)
g.add_legend();
```



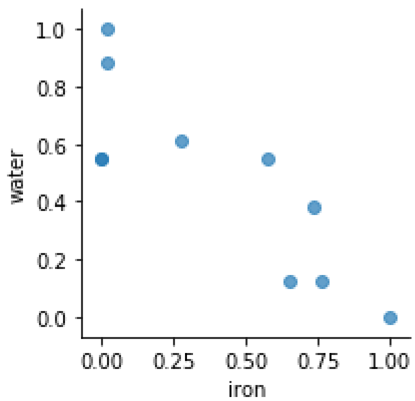
In [29]:
```
g = sns.FacetGrid(bean)
g.map(plt.scatter, "iron", "vit_e", alpha=.7)
g.add_legend();
```



In [30]:
```
g = sns.FacetGrid(bean)
g.map(plt.scatter, "iron", "vit_c", alpha=.7)
g.add_legend();
```

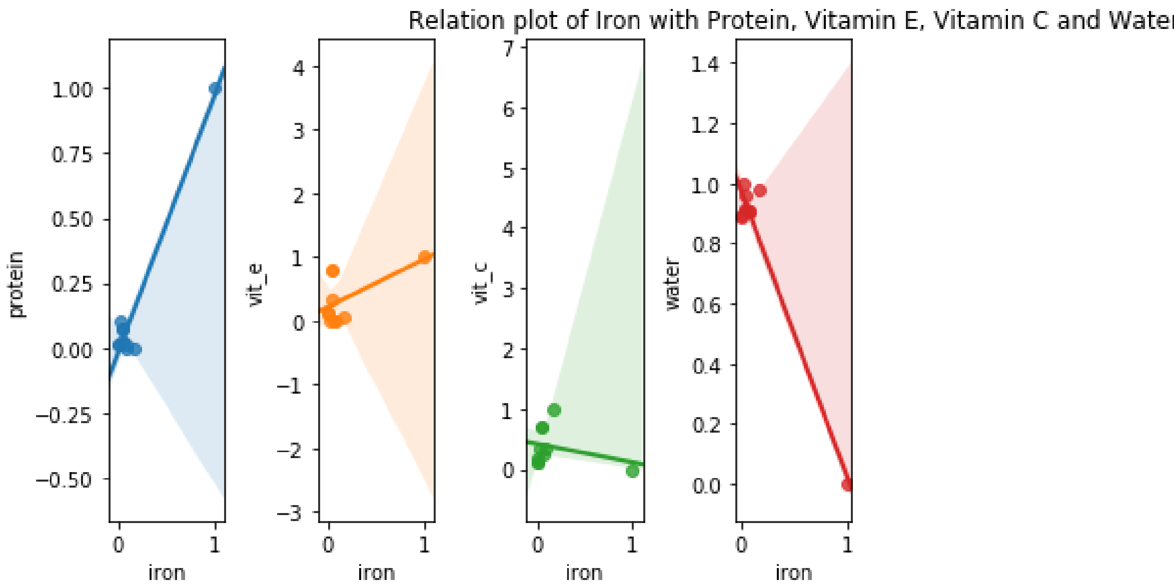


In [31]:
```
g = sns.FacetGrid(bean)
g.map(plt.scatter, "iron", "water", alpha=.7)
g.add_legend();
```



In [32]:
```
fig, axs = plt.subplots(ncols=4)
fig.tight_layout()
plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=0.8,
hspace=None)
sns.regplot(x='iron', y='protein', data=peach, ax=axs[0])
sns.regplot(x='iron', y='vit_e', data=peach, ax=axs[1])
sns.regplot(x='iron',y='vit_c', data=peach, ax=axs[2])
sns.regplot(x='iron',y='water', data=peach, ax=axs[3])
plt.title("Relation plot of Iron with Protein, Vitamin E, Vitamin C and Water"
)
```

Out[32]:
```
Text(0.5, 1, 'Relation plot of Iron with Protein, Vitamin E, Vitamin C and Wa
ter')
```

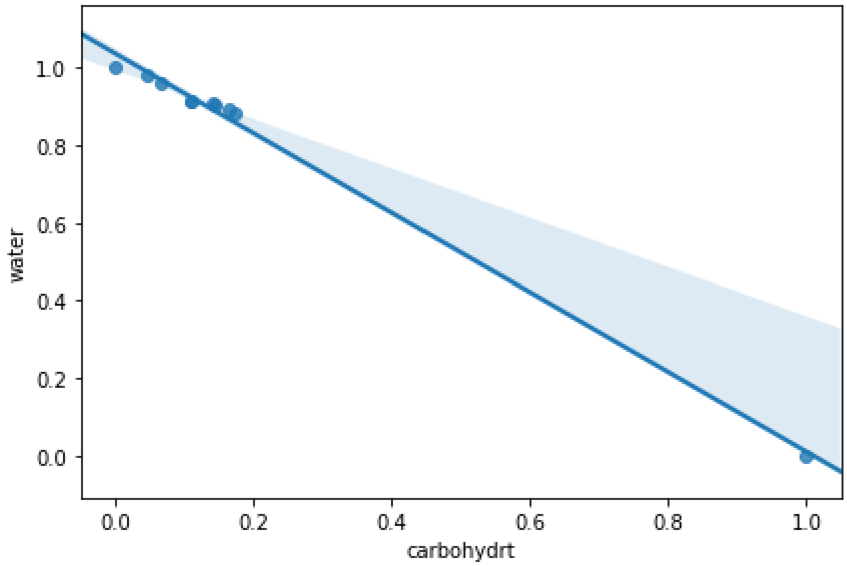


Plot observations: Direct proportionality of Iron can be seen with Protein and Vitamin E. It is slightly opposite in case of Vitamin C and completely opposite in case of Water.

Inverse Proportionality of Carbohydrate and Water: Carbohydrates and Water are oppositely proportional to each other. Water content is clearly inversely proportional to iron.

In [33]:
```
fig, axs = plt.subplots()
fig.tight_layout()
# plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=0.
8, hspace=None)
sns.regplot(x='carbohydrt', y='water', data=peach)
```

Out[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1917fddcfd0>
```



Finding which fruit has maximum given element

In [34]:
```python
# From overall data stored previously in csv_data dataframe is considered and
 pre processed again.
csv_data = csv_data[["fruit_name","calcium","carbohydrt","copper","energ_kcal"
,"iron","magnesium","manganese","phosphorus",\
                "protein","selenium","sodium","vit_b12","vit_b6","vit_c","vit_
e","vit_k","water","zinc"]]

for k in csv_data:
    if k not in ['fruit_name','s']:
        x = csv_data[k].values.astype(float) #returns a numpy array
        x = x.reshape(-1, 1)
        min_max_scaler = preprocessing.MinMaxScaler()
        x_scaled = min_max_scaler.fit_transform(x)
        csv_data[k] = pd.DataFrame(x_scaled)
csv_data
```

Out[34]:

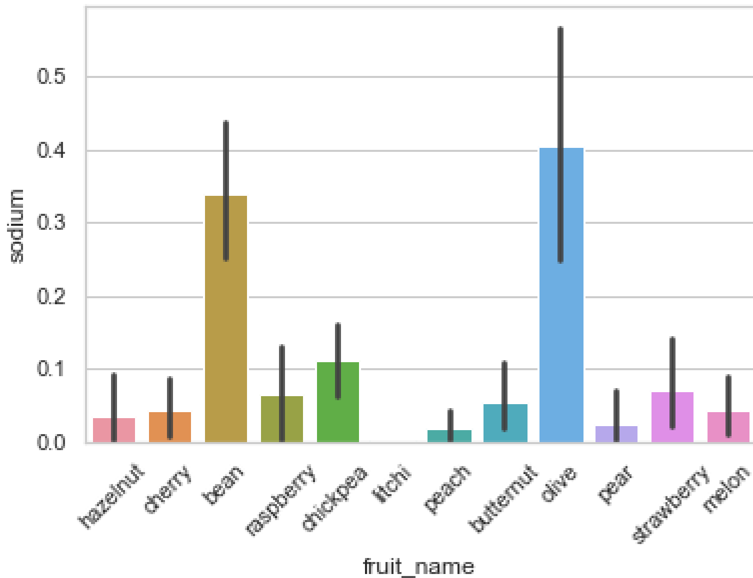
	fruit_name	calcium	carbohydrt	copper	energ_kcal	iron	magnesium	manganese
0	hazelnut	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
1	hazelnut	0.043011	0.849786	0.000000	0.434091	0.035212	0.162791	0.000000
2	hazelnut	0.072581	0.760367	0.268000	0.610227	0.050074	0.106312	0.040740
3	hazelnut	0.296371	0.281101	0.685714	0.709091	0.035669	0.390365	0.356707
4	hazelnut	0.076613	0.204281	0.985714	0.709091	0.053733	0.270764	0.289824
...	...	...	...	...	...	...	...	...
100	melon	0.006720	0.097125	0.034286	0.032955	0.003315	0.023256	0.001877
101	melon	0.008737	0.049664	0.020000	0.019318	0.002401	0.016611	0.001690
102	melon	0.004704	0.092355	0.024000	0.029545	0.002744	0.016611	0.001784
103	melon	0.008737	0.092477	0.011429	0.045455	0.012919	0.066445	0.001830
104	melon	0.012097	0.029969	0.012571	0.007955	0.004344	0.016611	0.002628

105 rows × 19 columns

From the overall data, we can see that Olives have maximum sodium element.

In [35]:
```
sns.set(style="whitegrid")
ax = sns.barplot(x="fruit_name", y="sodium", data=csv_data)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
```

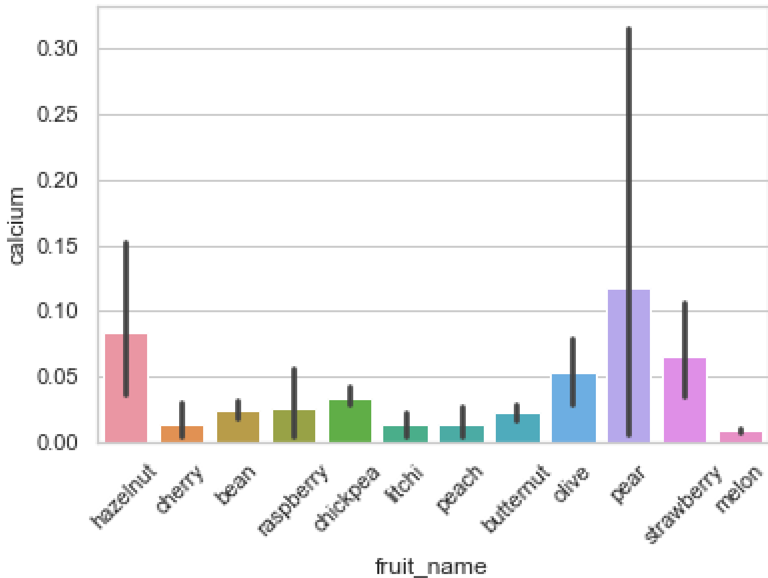
Out[35]:
```
[Text(0, 0, 'hazelnut'),
 Text(0, 0, 'cherry'),
 Text(0, 0, 'bean'),
 Text(0, 0, 'raspberry'),
 Text(0, 0, 'chickpea'),
 Text(0, 0, 'litchi'),
 Text(0, 0, 'peach'),
 Text(0, 0, 'butternut'),
 Text(0, 0, 'olive'),
 Text(0, 0, 'pear'),
 Text(0, 0, 'strawberry'),
 Text(0, 0, 'melon')]
```



From the overall data, we can see that Pear have maximum calcium.

In [36]:
```
sns.set(style="whitegrid")
ax = sns.barplot(x="fruit_name", y="calcium", data=csv_data)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
```

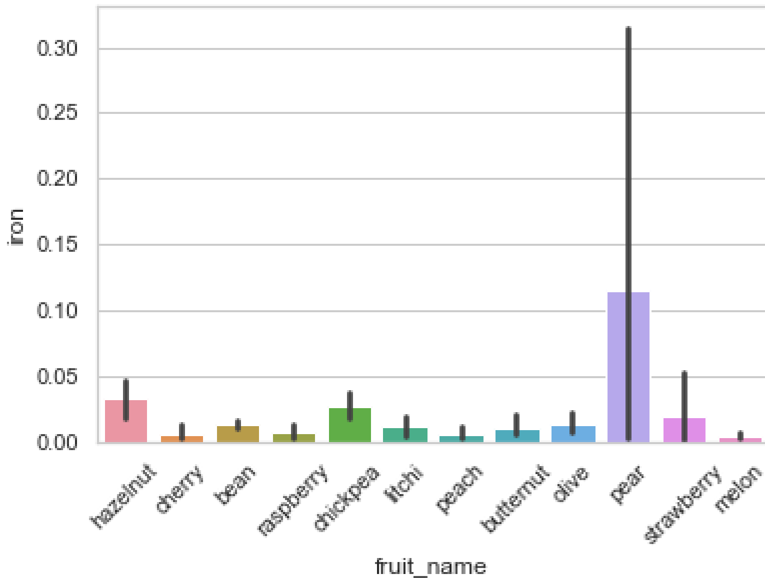
Out[36]:
```
[Text(0, 0, 'hazelnut'),
 Text(0, 0, 'cherry'),
 Text(0, 0, 'bean'),
 Text(0, 0, 'raspberry'),
 Text(0, 0, 'chickpea'),
 Text(0, 0, 'litchi'),
 Text(0, 0, 'peach'),
 Text(0, 0, 'butternut'),
 Text(0, 0, 'olive'),
 Text(0, 0, 'pear'),
 Text(0, 0, 'strawberry'),
 Text(0, 0, 'melon')]
```



From the overall data, Pear has maximum iron content. Iron is comparitively very less in other fruits in the given data.

In [37]:
```
sns.set(style="whitegrid")
ax = sns.barplot(x="fruit_name", y="iron", data=csv_data)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
```

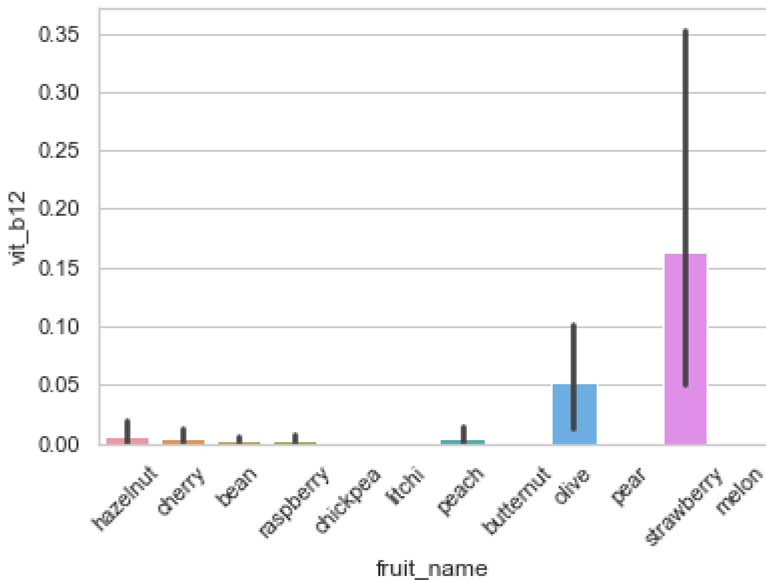
Out[37]:
```
[Text(0, 0, 'hazelnut'),
 Text(0, 0, 'cherry'),
 Text(0, 0, 'bean'),
 Text(0, 0, 'raspberry'),
 Text(0, 0, 'chickpea'),
 Text(0, 0, 'litchi'),
 Text(0, 0, 'peach'),
 Text(0, 0, 'butternut'),
 Text(0, 0, 'olive'),
 Text(0, 0, 'pear'),
 Text(0, 0, 'strawberry'),
 Text(0, 0, 'melon')]
```



From the considered fruits, Strawberry has maximum content of vitamin B12. This element is almost negligible in other given fruits.

In [38]:
```
sns.set(style="whitegrid")
ax = sns.barplot(x="fruit_name", y="vit_b12", data=csv_data)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
```

Out[38]:
```
[Text(0, 0, 'hazelnut'),
 Text(0, 0, 'cherry'),
 Text(0, 0, 'bean'),
 Text(0, 0, 'raspberry'),
 Text(0, 0, 'chickpea'),
 Text(0, 0, 'litchi'),
 Text(0, 0, 'peach'),
 Text(0, 0, 'butternut'),
 Text(0, 0, 'olive'),
 Text(0, 0, 'pear'),
 Text(0, 0, 'strawberry'),
 Text(0, 0, 'melon')]
```

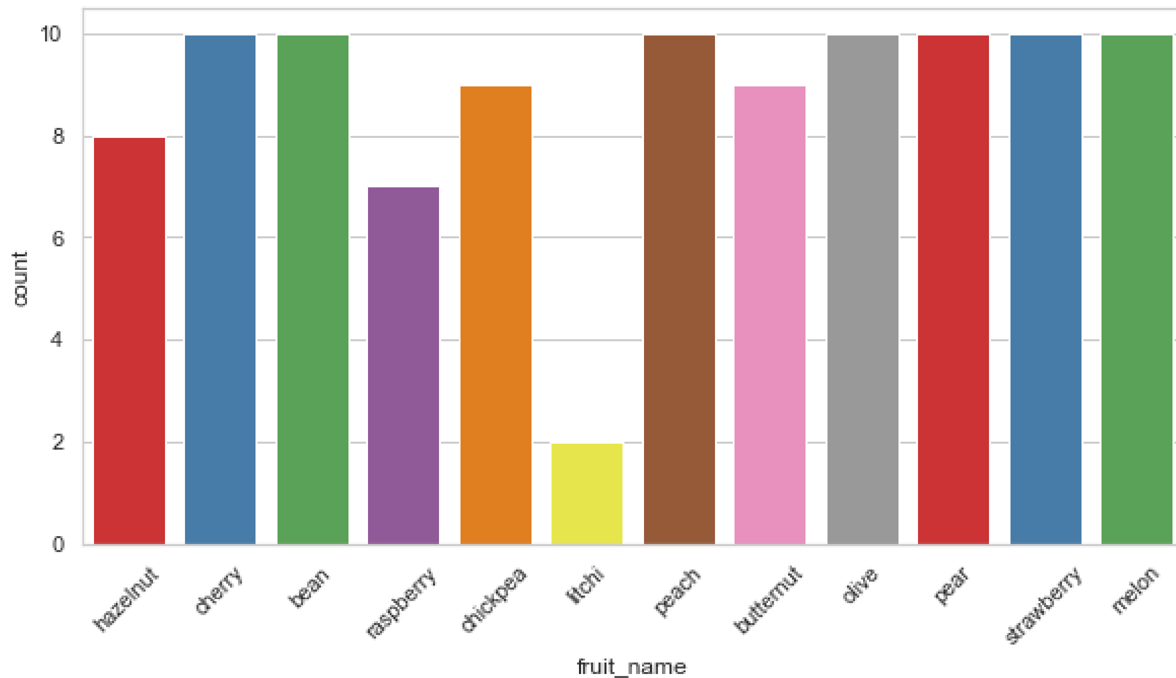


From the considered fruits, Pear has maximum content of Calcium.

As the given data varies over the amount of fruits, the maximum number of fruits given in dataset is as follows:

In [39]:
```
plt.figure(figsize=(10,5))
chart = sns.countplot(
    data=csv_data,
    x='fruit_name',
    palette='Set1'
)
chart.set_xticklabels(chart.get_xticklabels(), rotation=45)
```

Out[39]:
```
[Text(0, 0, 'hazelnut'),
 Text(0, 0, 'cherry'),
 Text(0, 0, 'bean'),
 Text(0, 0, 'raspberry'),
 Text(0, 0, 'chickpea'),
 Text(0, 0, 'litchi'),
 Text(0, 0, 'peach'),
 Text(0, 0, 'butternut'),
 Text(0, 0, 'olive'),
 Text(0, 0, 'pear'),
 Text(0, 0, 'strawberry'),
 Text(0, 0, 'melon')]
```



Litchi has minimum records and most of the fruits like Cherry, Bean, Peach etc have 10 records each meaning the element calculation can vary there.

Conclusion

In the analysis of different fruit elements, some things are particularily observed as given below:

1. Water is inversely proportional with most of the elements.
2. Pear has maximum Calcium and Iron elements in the provided data, similarily, sodium is maximum in olive, vitamin B12 in Strawberry.
3. Vitamin B12 and Iron are very less in other given fruit type meals.

Insights: As only limited fruits are considered, it is quite possible that there are other fruits having different values contain relative similarity with these fruits' elements. Each fruit has only 10 records given by the API so it is difficult to analyze much out of a single fruit value.

Future Scope: A meal selection system can be implemented to check the Body Analysis of a human and accordingly suggest him foods that which elements he/she should take intake of. Future visualizations-

1. To search - values of correlation in more than one fruit -> direct and indirect proportionality comparison between 2 elements in more than 2 fruits.
2. Plotting fruit against an element with some aggregation like - maximum or average

References:

1. https://pypi.org/project/pycorpora/ (https://pypi.org/project/pycorpora/)
2. https://rapidapi.com/kenpi04/api/food-calorie-data-search/endpoints (https://rapidapi.com/kenpi04/api/food-calorie-data-search/endpoints)
3. https://stackoverflow.com/questions/39689352/plotting-bar-plot-in-seaborn-python-with-rotated-xlabels/39689464 (https://stackoverflow.com/questions/39689352/plotting-bar-plot-in-seaborn-python-with-rotated-xlabels/39689464)
4. https://stackoverflow.com/questions/26414913/normalize-columns-of-pandas-data-frame/48651066 (https://stackoverflow.com/questions/26414913/normalize-columns-of-pandas-data-frame/48651066)