Load the dataset, and establish subsets

We removed the columns 'ref', 'Unnamed:0' and 'beans' (all have_beans) because they were irrelevant or redundant.

[14]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# load the chocodataset, dropping useless columns
df = pd.read_csv('chocolate.csv').drop(['ref', 'Unnamed: 0', 'beans'], axis=1)

#creating a separate indexed dataframe for just the ingredients
ing = df.iloc[:,8:14]
#sempre 6 colunas. location subject to changes in indexing order


#separate indexed column with the rating
rate = df.iloc[:,6] #subject to changes

#dataframe with all 4 tastes
tastes = df.iloc[:,14:18]

df.head()
```

[14]:

	company	company_location	review_date	country_of_bean_origin	specific_bean_origin_or_bar_name	cocoa_percent	rating	counts_of_ingredients	cocoa_butter	vanilla
0	5150	U.S.A	2019	Madagascar	Bejofo Estate, batch 1	76.0	3.75	3	have_cocoa_butter	have_not_vanila
1	5150	U.S.A	2019	Dominican republic	Zorzal, batch 1	76.0	3.50	3	have_cocoa_butter	have_not_vanila
2	5150	U.S.A	2019	Tanzania	Kokoa Kamili, batch 1	76.0	3.25	3	have_cocoa_butter	have_not_vanila
3	A. Morin	France	2012	Peru	Peru	63.0	3.75	4	have_cocoa_butter	have_not_vanila
4	A. Morin	France	2012	Bolivia	Bolivia	70.0	3.50	4	have_cocoa_butter	have_not_vanila

```
            lecithin           salt       sugar  \
0  have_not_lecithin  have_not_salt  have_sugar
1  have_not_lecithin  have_not_salt  have_sugar
2  have_not_lecithin  have_not_salt  have_sugar
3      have_lecithin  have_not_salt  have_sugar
4      have_lecithin  have_not_salt  have_sugar

            sweetener_without_sugar first_taste second_taste third_taste  \
0  have_not_sweetener_without_sugar       cocoa   blackberry   full body
1  have_not_sweetener_without_sugar       cocoa      vegetal      savory
2  have_not_sweetener_without_sugar  rich cocoa        fatty      bready
3  have_not_sweetener_without_sugar      fruity        melon      roasty
4  have_not_sweetener_without_sugar     vegetal        nutty         NaN

  fourth_taste
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN
```

[15]:
```
#created a function that transforms every dataframe that has a similar style to␣
 ↪have_/have_not into ones and zeroes

def ingredients_transformed(x):

   transformed = x.replace({"have_not*" :0, "have*": 1}, regex=True,␣
 ↪inplace=True)
   return transformed
```

[5]:
```
#only runs once, unless you restart and rerun all the outputs, from the␣
 ↪untransformed ing dataframe.
ingredients_transformed(ing)

#update the original dataset, with transformed ingredient columns, it works
df.update(ing)
```

1 Exploratory Analysis

[6]:
```
# Understanding the basic ground information of our chocodata

def all_about_my_data(df):
    print("Here is some Basic Ground Info about your Data:\n")
```

```
    # Shape of the chocodataframe
    print("Number of Instances:",df.shape[0])
    print("Number of Features:",df.shape[1])

    # Summary Stats
    print("\nSummary Stats:")
    print(df.describe())

    # Missing Value Inspection
    print("\nMissing Values:")
    print(df.isna().sum())

all_about_my_data(df)
```

```
Here is some Basic Ground Info about your Data:

Number of Instances: 2224
Number of Features: 18

Summary Stats:
       review_date  cocoa_percent       rating  counts_of_ingredients
count  2224.000000    2224.000000  2224.000000             2224.000000
mean   2013.857914      71.493930     3.198561                3.075989
std       3.582151       5.278253     0.434329                0.929875
min    2006.000000      42.000000     1.000000                1.000000
25%    2011.000000      70.000000     3.000000                2.000000
50%    2014.000000      70.000000     3.250000                3.000000
75%    2016.000000      74.000000     3.500000                4.000000
max    2020.000000     100.000000     4.000000                6.000000

Missing Values:
company                              0
company_location                     0
review_date                          0
country_of_bean_origin               0
specific_bean_origin_or_bar_name     0
cocoa_percent                        0
rating                               0
counts_of_ingredients                0
cocoa_butter                         0
vanilla                              0
lecithin                             0
salt                                 0
sugar                                0
sweetener_without_sugar              0
first_taste                          0
second_taste                        77
```

```
third_taste                    620
fourth_taste                  1982
dtype: int64
```

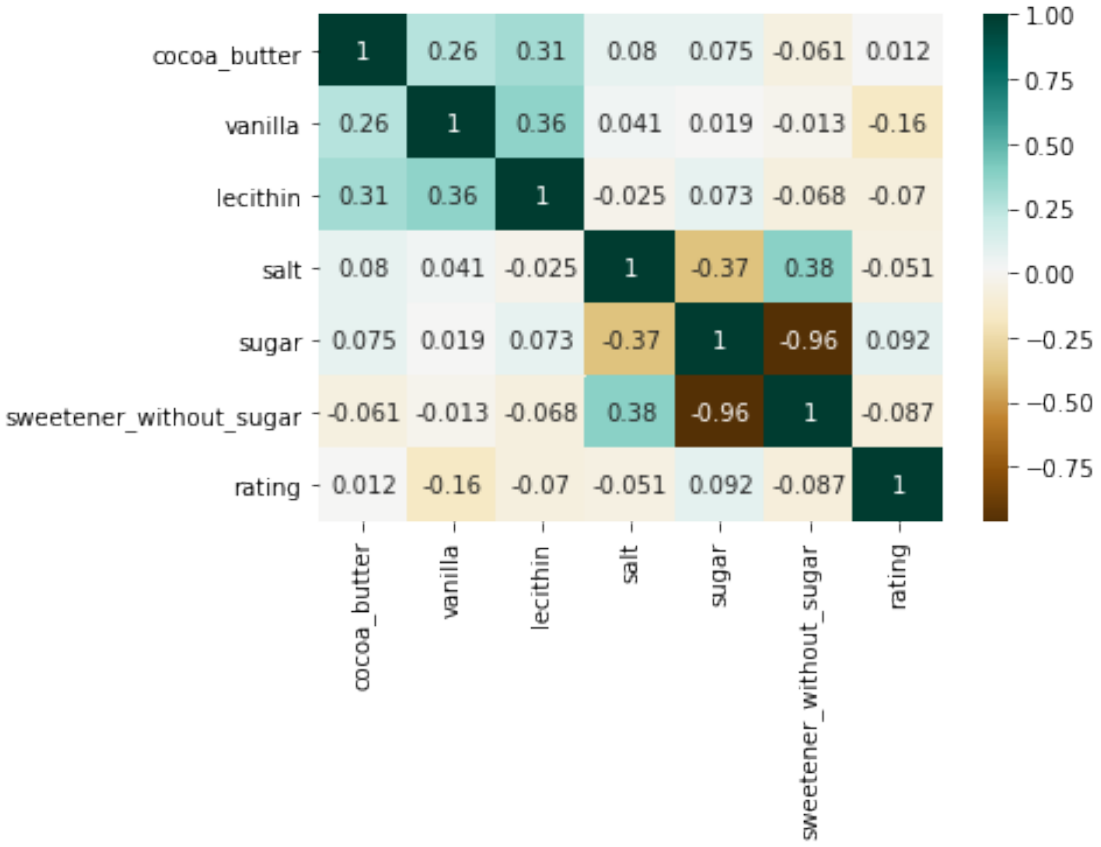
2 Who creates the best Chocolate bars?

2.1 What makes a good chocolate?

We tried to correlate the usage of all ingredients against each other, as represented in the heatmap. For example, salt+sugar, and sugar+sweetener without sugar are the least common combination of ingredients, strongly negatively correlated. Salt+sweetener without sugar, lecithin+vanilla/cocoa_butter, and cocoa butter+vanilla are some of the most common combos.

Finally, sugar and cocoa butter are certain amongst the better rated chocolates.

[7]:
```
import seaborn as sns
import matplotlib.pyplot as plt

#joining rating column with the ingredients dataframe
corr = ing.join(rate)
correlation_mat = corr.corr()
sns.heatmap(correlation_mat, annot = True, cmap="BrBG")
plt.show()
```



3 Exploring with pyspark.

[8]:
```
from pyspark.mllib.util import MLUtils
from pyspark.sql.types import *
from pyspark.ml.feature import CountVectorizer, CountVectorizerModel, Tokenizer,␣
 ↪RegexTokenizer, StopWordsRemover
from pyspark.sql.functions import mean, min, max
from pyspark import SparkContext
from pyspark.mllib.feature import HashingTF
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext, Row
from pyspark.mllib.linalg import Vector, Vectors
from __future__ import print_function
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.sql import functions as F


sparky = SparkSession.builder.appName('Choco_Data_Analysis').getOrCreate()


#If the csv file have a header (column names in the first row) then set␣
 ↪header=true. This will use the first row in the csv file as the dataframe's␣
 ↪column names. Setting
sp = (sparky.read
.option("inferSchema","true")
.option("header","true")
.csv('chocolate.csv')).na.drop()
```

3.1 Predict the number of ingredients based off the rating

[9]:
```
sqlContext = SQLContext(sparky)


#predicting  count of ingredients of the chocolate
FEATURES_COL = ['counts_of_ingredients']

#selecting from the "main" dataset for this query
sp2 = sp.select('rating', 'counts_of_ingredients').na.drop()
df = sp2.toDF('rating','counts_of_ingredients')


#we need all the features/vars to be of the same type
```

```
df = df.select(*(df[c].cast("float").alias(c) for c in df.columns[0:]))


vecAssembler = VectorAssembler(inputCols=FEATURES_COL,␣
 ↪outputCol='predicted_no_ingredient')
df_kmeans = vecAssembler.transform(df).select('rating',␣
 ↪'predicted_no_ingredient')

df_kmeans.printSchema()


# Train a k-means model.
kmeans = KMeans().setK(4).setSeed(1).setFeaturesCol('predicted_no_ingredient')
model = kmeans.fit(df_kmeans)
centers = model.clusterCenters()

print("Cluster Centers: ")
for center in centers:
    print(center)

# assigning the individual rows to the nearest cluster centroid. the transform␣
 ↪method,adds 'prediction' column to the dataframe.
transformed = model.transform(df_kmeans).select('rating',␣
 ↪'predicted_no_ingredient')
rows = transformed.collect()

from pyspark.sql.functions import col
df_pred = sqlContext.createDataFrame(rows)
df_pred = df_pred.join(df, 'rating')
df_pred.dropDuplicates(['rating','counts_of_ingredients']).filter(col("rating")␣
 ↪<= 4).orderBy('rating', ascending=False).show()
```

```
root
 |-- rating: float (nullable = true)
 |-- predicted_no_ingredient: vector (nullable = true)

Cluster Centers:
[3.99603175]
[2.]
[3.00574713]
[5.]
+------+-----------------------+---------------------+
|rating|predicted_no_ingredient|counts_of_ingredients|
+------+-----------------------+---------------------+
|   4.0|                  [4.0]|                 4.0|
|   4.0|                  [4.0]|                 2.0|
|   4.0|                  [4.0]|                 5.0|
```

```
|   4.0|                 [4.0]|                 3.0|
|  3.75|                 [4.0]|                 2.0|
|  3.75|                 [4.0]|                 3.0|
|  3.75|                 [4.0]|                 5.0|
|  3.75|                 [4.0]|                 4.0|
|   3.5|                 [4.0]|                 3.0|
|   3.5|                 [4.0]|                 4.0|
|   3.5|                 [4.0]|                 2.0|
|   3.5|                 [4.0]|                 5.0|
|  3.25|                 [3.0]|                 2.0|
|  3.25|                 [3.0]|                 4.0|
|  3.25|                 [3.0]|                 5.0|
|  3.25|                 [3.0]|                 3.0|
|   3.0|                 [2.0]|                 5.0|
|   3.0|                 [2.0]|                 4.0|
|   3.0|                 [2.0]|                 2.0|
|   3.0|                 [2.0]|                 3.0|
+------+----------------------+--------------------+
only showing top 20 rows
```

3.2 What is the rating of portuguese companies, and where do they get their beans?

[10]:
```
# describing portuguese companies, rating, and where do they get their beans

sp.select('rating', 'company_location', 'company', 'country_of_bean_origin').
 ↪filter(sp.company_location == 'Portugal').show(10)
```

```
+------+----------------+-------------+----------------------+
|rating|company_location|      company|country_of_bean_origin|
+------+----------------+-------------+----------------------+
|   2.5|        Portugal|Feitoria Cacao|             Venezuela|
+------+----------------+-------------+----------------------+
```

3.3 What are the top 5 worst rated countries of bean origin?

Venezuela, St. Lucia, and Belize

[11]:
```
from pyspark.sql.functions import col, avg

sp.select('rating', 'country_of_bean_origin').filter(col("rating") <= 4).
 ↪dropDuplicates(["country_of_bean_origin", 'rating']).orderBy('rating',␣
 ↪'country_of_bean_origin', ascending=True).show(5)
```

```
+------+----------------------+
|rating|country_of_bean_origin|
+------+----------------------+
```

```
|   2.0|             Venezuela|
|  2.25|             St. lucia|
|  2.25|             Venezuela|
|   2.5|                Belize|
|   2.5|                 Blend|
+------+----------------------+
only showing top 5 rows
```

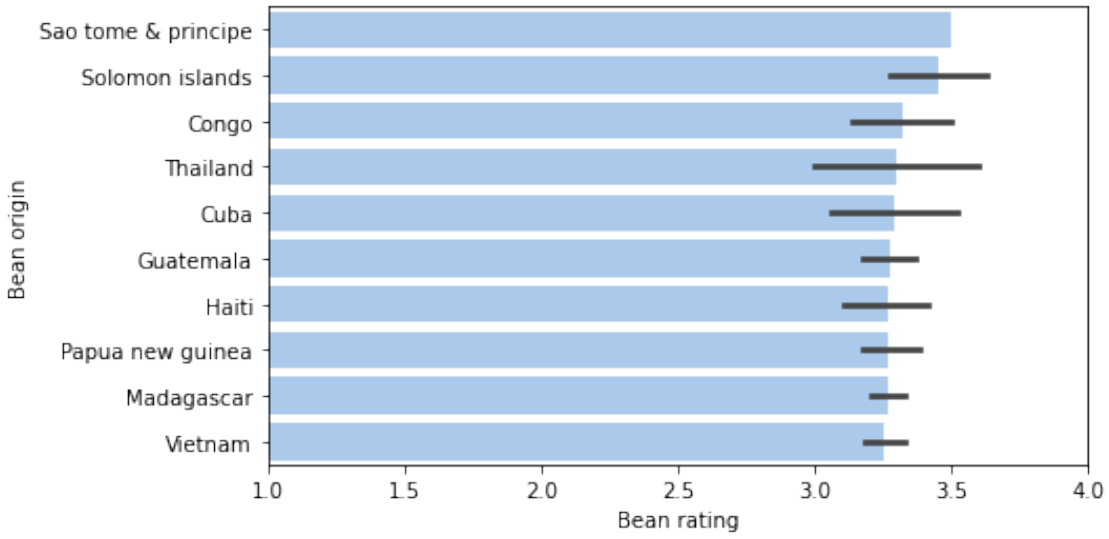
4 Plotting with pandas

4.1 Where are the best cocoa beans grown?

The best beans which create the highest rated chocolates are grown in Sao Tome e Principe, Solomon Islands, and Congo.

[16]:
```
import seaborn as sns
import matplotlib.pyplot as plt

#necessary to make the later plot show in descending order, and only show the␣
 ↪top 10 highest chocos
grp_order = df.groupby('country_of_bean_origin').rating.agg('mean').
 ↪sort_values(ascending=False).iloc[:10].index


# Initialize the matplotlib figure

f, ax = plt.subplots(figsize=(7, 4))
sns.set_color_codes('pastel')
sns.barplot(x='rating', y='country_of_bean_origin', data=df,
            label="Best cocoa beans", color="b",estimator=np.mean,␣
 ↪order=grp_order)

# Add a legend and informative axis and limit
ax.set(xlim=(1, 4), ylabel="Bean origin",
       xlabel="Bean rating")
```

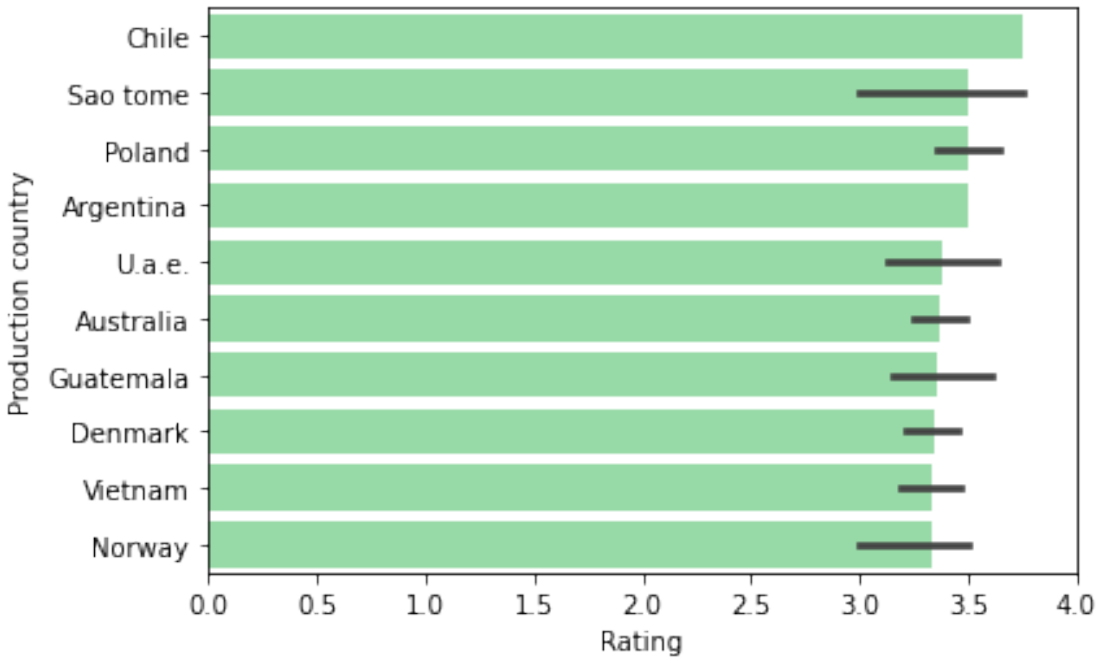
[16]:
```
[(1.0, 4.0), Text(0, 0.5, 'Bean origin'), Text(0.5, 0, 'Bean rating')]
```



4.2 Which countries produce the highest-rated bars?

[17]:
```
#necessary to make the later plot show in descending order, and only show the␣
 ↪top 10 highest chocos
grp_order = df.groupby('company_location').rating.agg('mean').
 ↪sort_values(ascending=False).iloc[:10].index


# Initialize the matplotlib figure

f, ax = plt.subplots(figsize=(6, 4))
sns.set_color_codes('pastel')
sns.barplot(x='rating', y='company_location', data=df,
            label="Best cocoa beans", color="g",estimator=np.mean,␣
 ↪order=grp_order)

# Add a legend and informative axis and limit
ax.set(xlim=(0, 4), ylabel="Production country",
       xlabel="Rating")
```

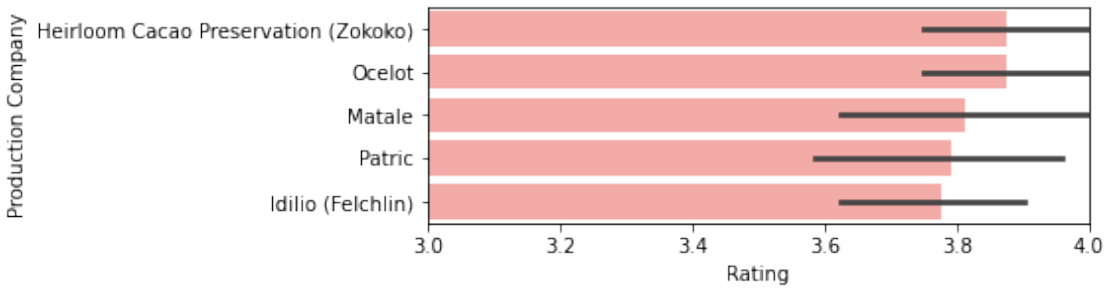
[17]:
```
[(0.0, 4.0), Text(0, 0.5, 'Production country'), Text(0.5, 0, 'Rating')]
```



4.3 Which company has the highest rating?

[18]:
```
#necessary to make the later plot show in descending order, and only show the␣
 ↪top 10 highest chocos
grp_order = df.groupby('company').rating.agg('mean').
 ↪sort_values(ascending=False).iloc[:5].index


# Initialize the matplotlib figure

f, ax = plt.subplots(figsize=(6, 2))
sns.set_color_codes('pastel')
sns.barplot(x='rating', y='company', data=df,
            label="Best cocoa beans", color="r",estimator=np.mean,␣
 ↪order=grp_order)

# Add a legend and informative axis and limit
ax.set(xlim=(3, 4), ylabel="Production Company",
       xlabel="Rating")
```

[18]:
```
[(3.0, 4.0), Text(0, 0.5, 'Production Company'), Text(0.5, 0, 'Rating')]
```



4.4 What are the most reported flavours in the first tastings?

[19]:
```
# Import the wordcloud library
from wordcloud import WordCloud

# Join the different processed titles together.
long_string = ','.join(list(tastes['first_taste'].values))

# Create a WordCloud object
wordcloud = WordCloud(background_color="white", max_words=5000, contour_width=3,␣
↪contour_color='steelblue')

# Generate a word cloud
wordcloud.generate(long_string)

# Visualize the word cloud
wordcloud.to_image()
```

[19]: