

Coffee Quality

In this analysis we have used a coffee quality dataset from Kaggle: [Coffee Quality Data \(CQI May-2023\)](#).

With this dataset and various machine learning methods we are trying to gain deeper insight into what determines coffee quality, discover patterns in the data and predict the quality based on the different features.

We will use a mixture of data exploration, different clustering methods and a random forest regression for our analysis and to investigate our problem domain about coffee quality.

We start by cleaning the dataset in the next page, so that we have a version that is both suitable for data exploration, clustering and regression.

In [281...]

```
%time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import zscore
```

CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.11 µs

Data Cleaning

In [282...]

```
df = pd.read_csv("../data/merged_data_cleaned.csv")
df.head()
```

Out[282...]

	Unnamed: 0	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mi
0	0	Arabica	metad plc	Ethiopia	metad plc	NaN	meta p
1	1	Arabica	metad plc	Ethiopia	metad plc	NaN	meta p
2	2	Arabica	grounds for health admin	Guatemala	san marcos barrancas "san cristobal cuch	NaN	Na
3	3	Arabica	yidnekachew dabessa	Ethiopia	yidnekachew dabessa coffee plantation	NaN	wolens
4	4	Arabica	metad plc	Ethiopia	metad plc	NaN	meta p

5 rows × 44 columns

We have a lot of columns with data that are irrelevant for our analysis. We'll drop them to reduce dimensionality of the dataset

In [283...]

```
df = df.drop(["Unnamed: 0", "Farm.Name", "Lot.Number", "Mill", "ICO.Numb
```

In [284...]

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1339 entries, 0 to 1338
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Species          1339 non-null    object  
 1   Country.of.Origin 1338 non-null    object  
 2   Variety           1113 non-null    object  
 3   Processing.Method 1169 non-null    object  
 4   Aroma              1339 non-null    float64 
 5   Flavor             1339 non-null    float64 
 6   Aftertaste         1339 non-null    float64 
 7   Acidity            1339 non-null    float64 
 8   Body               1339 non-null    float64 
 9   Balance            1339 non-null    float64 
 10  Uniformity         1339 non-null    float64 
 11  Clean.Cup          1339 non-null    float64 
 12  Sweetness           1339 non-null    float64 
 13  Copper.Points      1339 non-null    float64 
 14  Total.Cup.Points   1339 non-null    float64 
 15  Moisture            1339 non-null    float64 
 16  Category.One.Defects 1339 non-null    int64  
 17  Quakers            1338 non-null    float64 
 18  Color               1069 non-null    object  
 19  Category.Two.Defects 1339 non-null    int64  
 20  altitude_mean_meters 1109 non-null    float64 
dtypes: float64(14), int64(2), object(5)
memory usage: 219.8+ KB
```

In [285... df.isna().sum()

```
Out[285... Species          0
Country.of.Origin     1
Variety              226
Processing.Method     170
Aroma                0
Flavor               0
Aftertaste            0
Acidity              0
Body                 0
Balance              0
Uniformity           0
Clean.Cup             0
Sweetness             0
Copper.Points         0
Total.Cup.Points      0
Moisture              0
Category.One.Defects  0
Quakers              1
Color                 270
Category.Two.Defects  0
altitude_mean_meters 230
dtype: int64
```

We replace rows with a lot of missing nominal data with the mode of the column, to retain the variance of the rest of the row data

In [286... df = df.fillna({'Variety': df['Variety'].mode()[0]})
df = df.fillna({'Processing.Method': df['Processing.Method'].mode()[0]})

```
df = df.fillna({'Color': df['Color'].mode()[0]})
```

We drop the rows with a single row missing nominal data. Especially "Country of Origin", since we can not just put in a mode value, since it might create significant wrong data

```
In [287... df = df.dropna(how='any')
```

```
In [288... df.isna().sum()
```

```
Out[288... Species          0
Country.of.Origin      0
Variety               0
Processing.Method     0
Aroma                 0
Flavor                0
Aftertaste             0
Acidity               0
Body                  0
Balance               0
Uniformity            0
Clean.Cup              0
Sweetness              0
Copper.Points          0
Total.Cup.Points       0
Moisture               0
Category.One.Defects   0
Quakers               0
Color                 0
Category.Two.Defects   0
altitude_mean_meters    0
dtype: int64
```

No more missing values!

Outliers

```
In [289... df.describe()
```

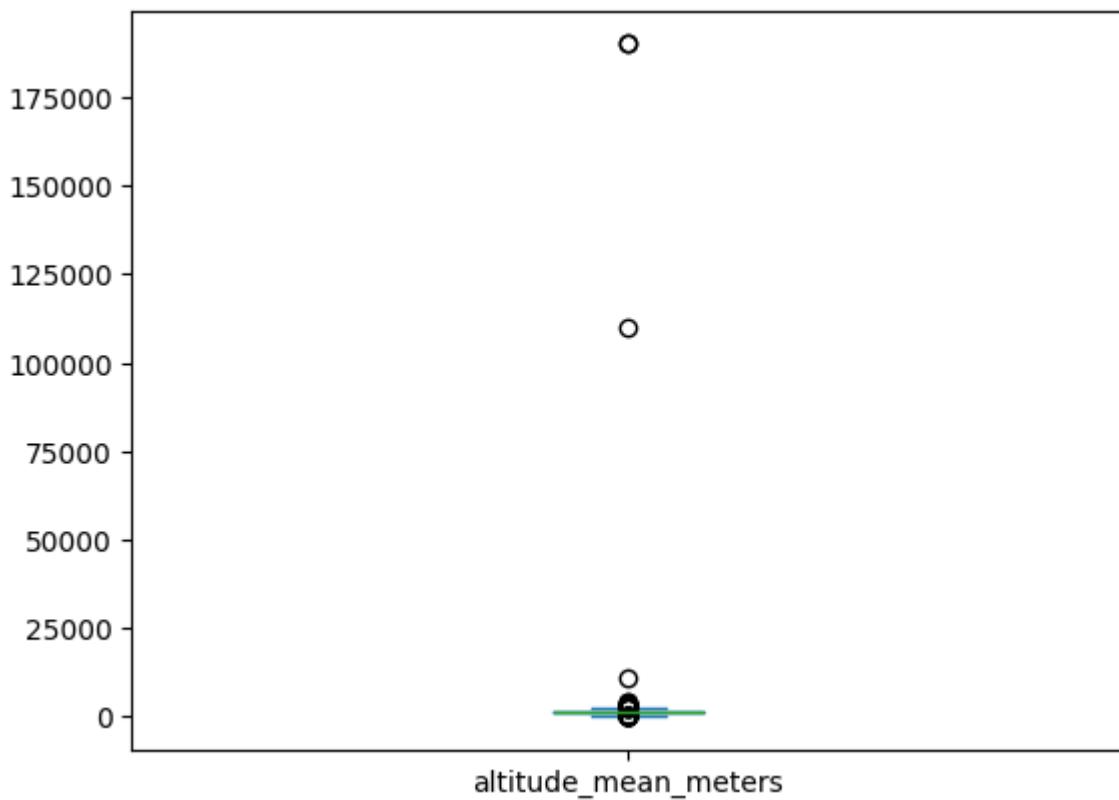
	Aroma	Flavor	Aftertaste	Acidity	Body	Balance
count	1108.000000	1108.000000	1108.000000	1108.000000	1108.000000	1108.000000
mean	7.570569	7.52056	7.394269	7.528953	7.506670	7.505542
std	0.383837	0.40059	0.405867	0.386075	0.366717	0.419311
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	7.420000	7.33000	7.250000	7.330000	7.330000	7.330000
50%	7.580000	7.58000	7.420000	7.500000	7.500000	7.500000
75%	7.750000	7.75000	7.580000	7.750000	7.670000	7.750000
max	8.750000	8.83000	8.670000	8.750000	8.580000	8.750000

We have some columns with very high standard deviations

```
In [290... fig = plt.figure()
```

```
df.altitude_mean_meters.plot.box()
```

Out[290... <Axes: >



The column contains significant outliers. Since its only a couple of rows, we'll drop them

```
In [291... # We'll remove the outlying rows based on z-score
df = df[np.abs(zscore(df['altitude_mean_meters'])) < 1]
```

In [292... df

Out[292...]

	Species	Country.of.Origin	Variety	Processing.Method	Aroma	Flavor	Aftertaste
0	Arabica	Ethiopia	Caturra	Washed / Wet	8.67	8.83	8.67
1	Arabica	Ethiopia	Other	Washed / Wet	8.75	8.67	8.50
2	Arabica	Guatemala	Bourbon	Washed / Wet	8.42	8.50	8.42
3	Arabica	Ethiopia	Caturra	Natural / Dry	8.17	8.58	8.42
4	Arabica	Ethiopia	Other	Washed / Wet	8.25	8.50	8.25
...
1331	Robusta	India	Caturra	Washed / Wet	7.67	7.67	7.50
1332	Robusta	India	Caturra	Natural / Dry	7.58	7.42	7.42
1333	Robusta	United States	Arusha	Natural / Dry	7.92	7.50	7.42
1335	Robusta	Ecuador	Caturra	Washed / Wet	7.50	7.67	7.75
1336	Robusta	United States	Caturra	Natural / Dry	7.33	7.33	7.15

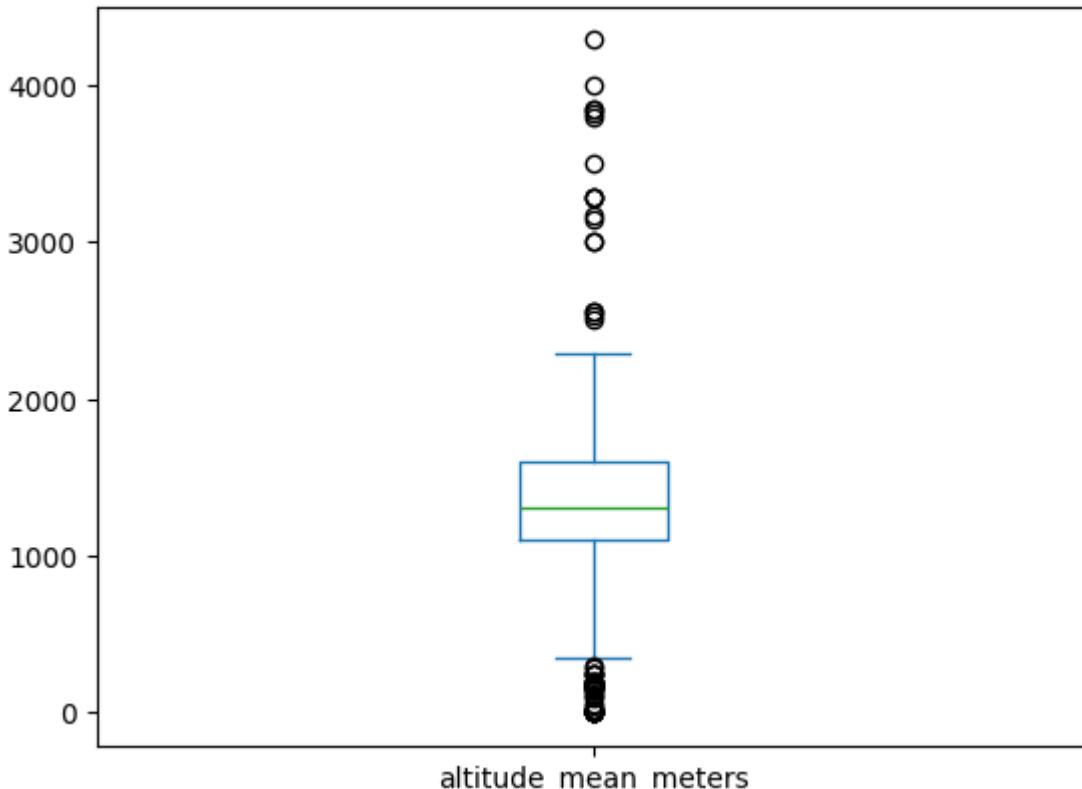
1104 rows × 21 columns

In [293...]

```
df.altitude_mean_meters.plot.box()
```

Out[293...]

```
<Axes: >
```

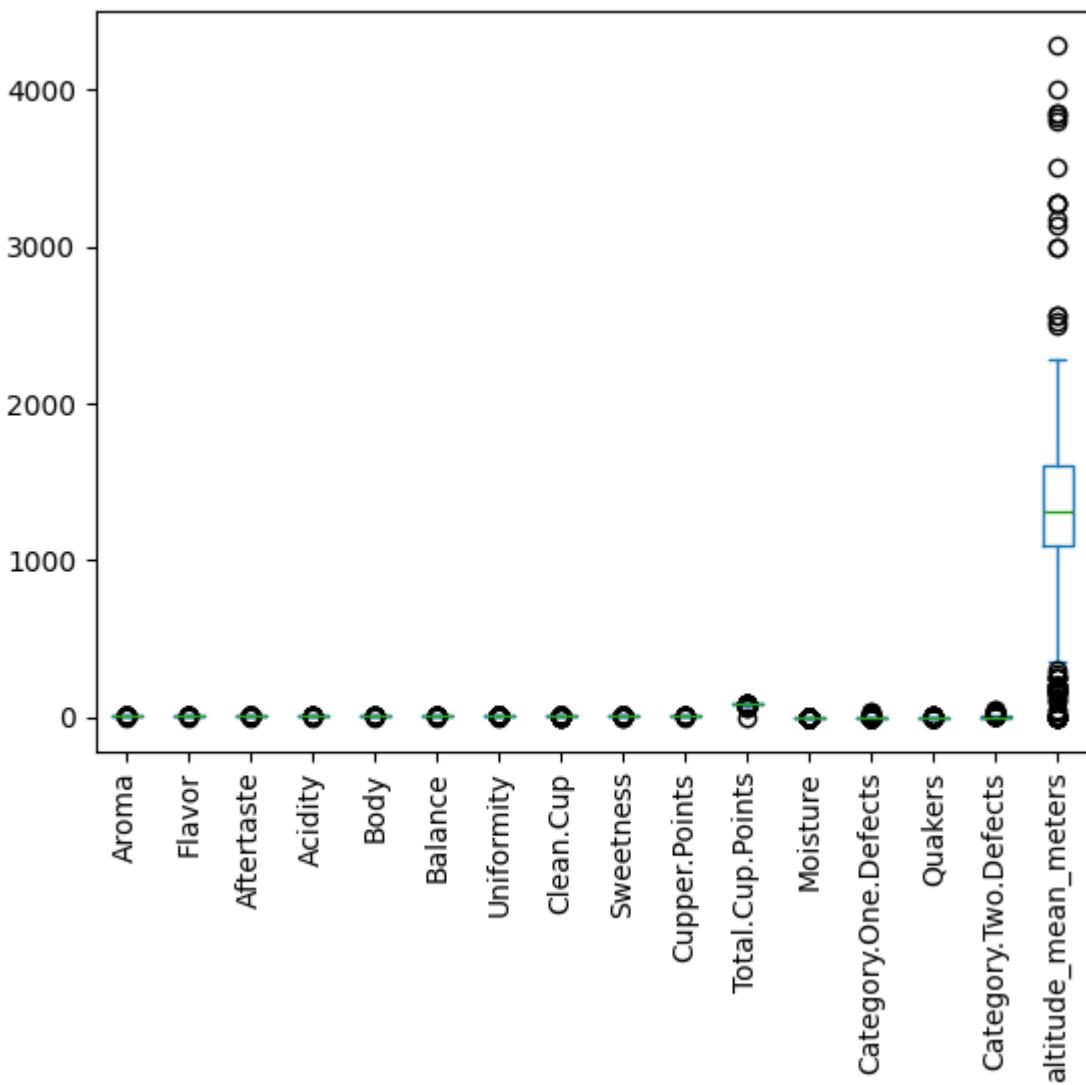


In [294...]

```
df.plot.box(rot=90)
```

Out[294...]

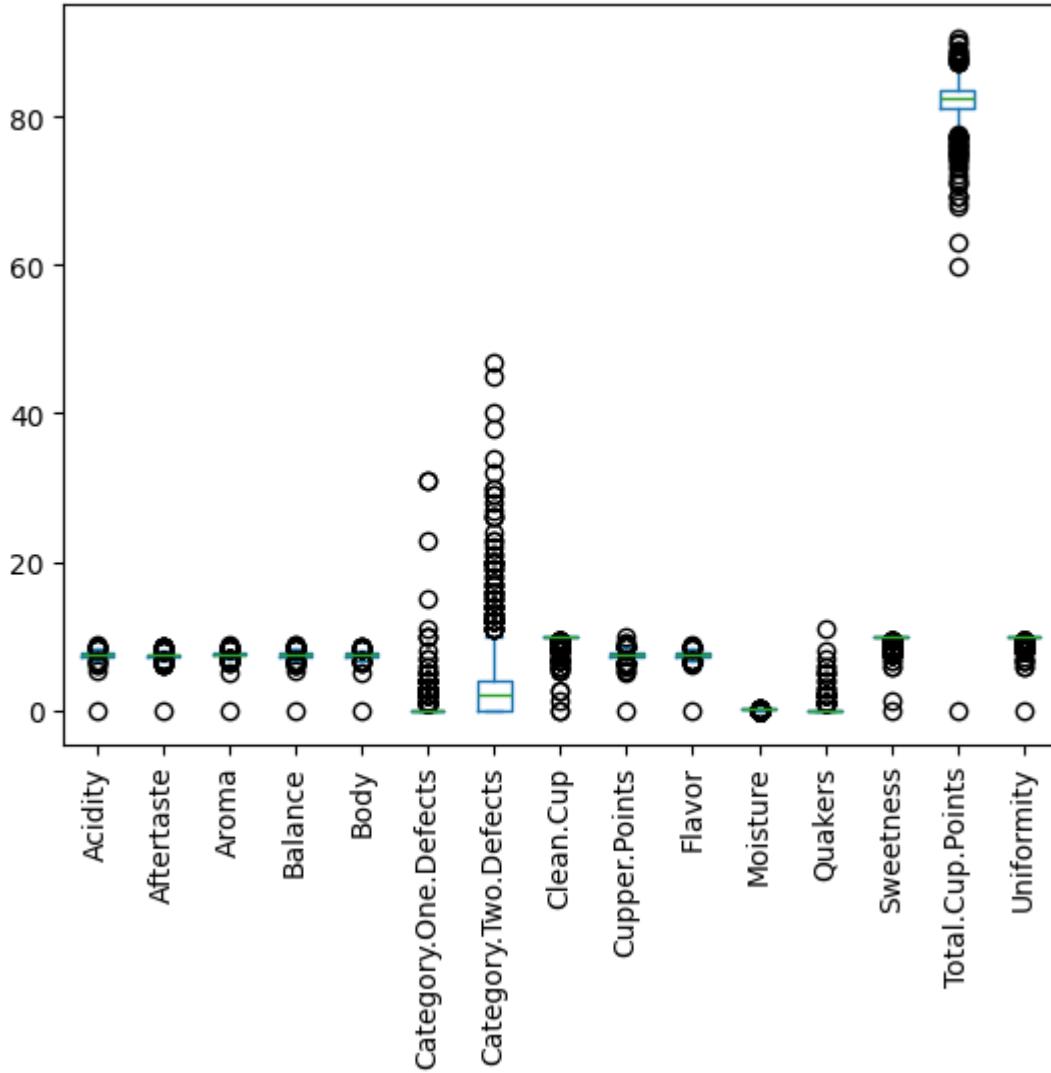
```
<Axes: >
```



Even after removing the worst outliers, the mean altitude still distributes over large values. We'll exclude it in the plot to identify other problematic features

```
In [295...]: df[df.columns.difference(['altitude_mean_meters'])].plot.box(rot=90)
```

```
Out[295...]: <Axes: >
```



It seems a lot of the features contain unnatural zero-values. We'll replace the zero values, with the median of the feature

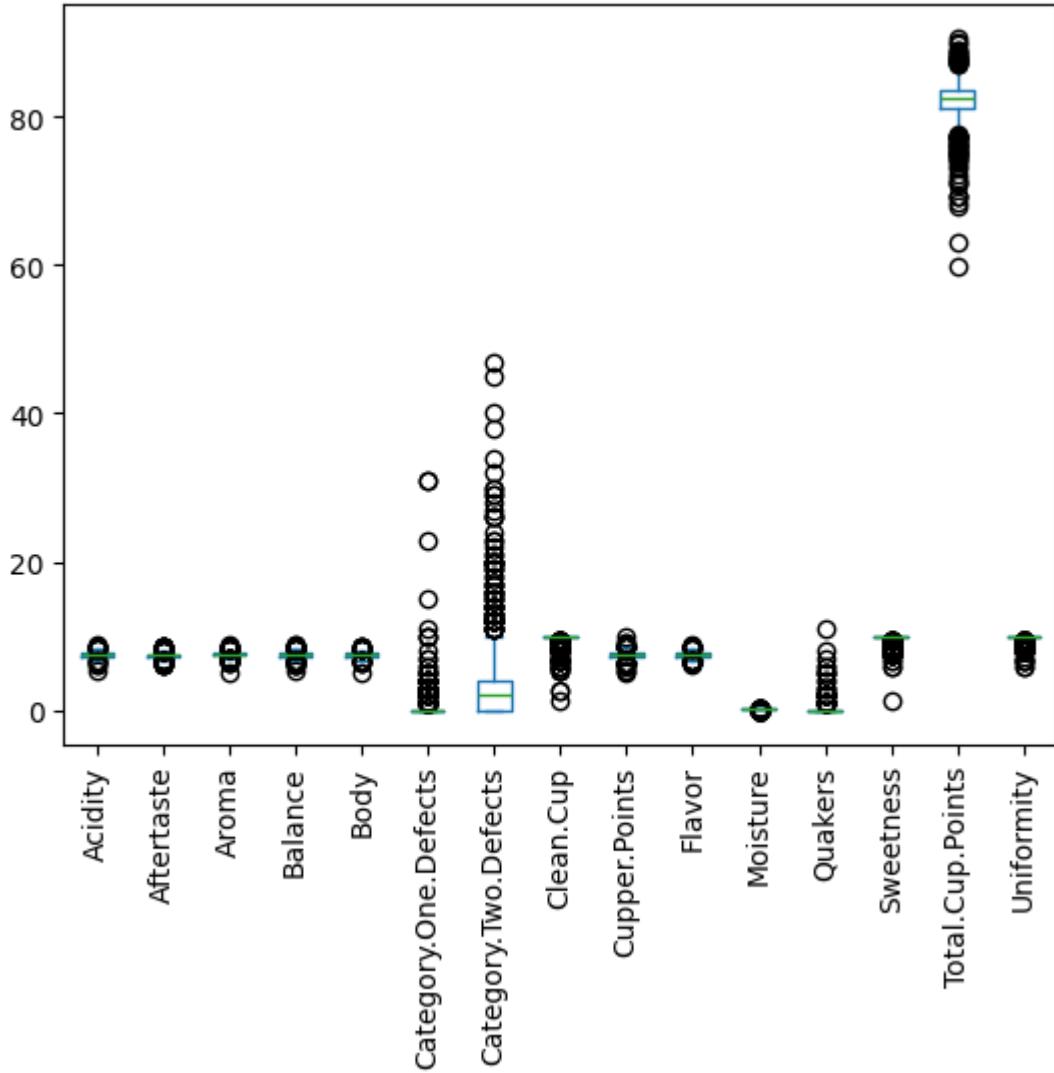
```
In [296...]: df['Acidity'] = df['Acidity'].replace(0, df['Acidity'].median())
df['Aftertaste'] = df['Aftertaste'].replace(0, df['Aftertaste'].median())
df['Aroma'] = df['Aroma'].replace(0, df['Aroma'].median())
df['Balance'] = df['Balance'].replace(0, df['Balance'].median())
df['Body'] = df['Body'].replace(0, df['Body'].median())
df['Clean.Cup'] = df['Clean.Cup'].replace(0, df['Clean.Cup'].median())
df['Cupper.Points'] = df['Cupper.Points'].replace(0, df['Cupper.Points'].median())
df['Flavor'] = df['Flavor'].replace(0, df['Flavor'].median())
df['Moisture'] = df['Moisture'].replace(0, df['Moisture'].median())
df['Sweetness'] = df['Sweetness'].replace(0, df['Sweetness'].median())
df['Uniformity'] = df['Uniformity'].replace(0, df['Uniformity'].median())
```

Except the Total cup points. We'll drop the row since it is our target value, and an unnatural zero might mess with correlations

```
In [297...]: df = df[df['Total.Cup.Points'] != 0]
```

```
In [298...]: df[df.columns.difference(['altitude_mean_meters'])].plot.box(rot=90)
```

```
Out[298...]: <Axes: >
```



```
In [299]: df.describe()
```

	Aroma	Flavor	Aftertaste	Acidity	Body	Balance
count	1103.000000	1103.000000	1103.000000	1103.000000	1103.000000	1103.000000
mean	7.578368	7.527824	7.401496	7.535739	7.513654	7.512661
std	0.309181	0.331268	0.340065	0.313192	0.289467	0.354021
min	5.080000	6.170000	6.170000	5.250000	5.170000	5.250000
25%	7.420000	7.330000	7.250000	7.330000	7.330000	7.330000
50%	7.580000	7.580000	7.420000	7.500000	7.500000	7.500000
75%	7.750000	7.750000	7.580000	7.750000	7.670000	7.750000
max	8.750000	8.830000	8.670000	8.750000	8.580000	8.750000

```
In [300]: df.to_csv('cleaned_dataset_no_zeros.csv', index=False)
```

The dataset is now ready for analysis!

In [47]:

```
%time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN
from scipy.stats import zscore
import scipy.cluster.hierarchy as ch
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import LabelEncoder
pd.options.display.max_columns = None
```

CPU times: user 0 ns, sys: 337 µs, total: 337 µs

Wall time: 4.65 ms

In [48]:

```
df = pd.read_csv("../data_cleaned/cleaned_dataset_no_zeros.csv")
df.head(20)
```

Out [48]:

	Species	Country.of.Origin	Variety	Processing.Method	Aroma	Flavor	Aftertaste
0	Arabica	Ethiopia	Caturra	Washed / Wet	8.67	8.83	8
1	Arabica	Ethiopia	Other	Washed / Wet	8.75	8.67	8
2	Arabica	Guatemala	Bourbon	Washed / Wet	8.42	8.50	8
3	Arabica	Ethiopia	Caturra	Natural / Dry	8.17	8.58	8
4	Arabica	Ethiopia	Other	Washed / Wet	8.25	8.50	8
5	Arabica	Ethiopia	Caturra	Washed / Wet	8.25	8.33	8
6	Arabica	Ethiopia	Caturra	Washed / Wet	8.67	8.67	8
7	Arabica	Ethiopia	Other	Natural / Dry	8.08	8.58	8
8	Arabica	Ethiopia	Caturra	Natural / Dry	8.17	8.67	8
9	Arabica	United States	Other	Washed / Wet	8.25	8.42	8
10	Arabica	United States	Other	Washed / Wet	8.08	8.67	8
11	Arabica	United States (Hawaii)	Caturra	Washed / Wet	8.33	8.42	8
12	Arabica	Ethiopia	Caturra	Washed / Wet	8.25	8.33	8
13	Arabica	United States	Other	Washed / Wet	8.00	8.50	8
14	Arabica	Indonesia	Caturra	Washed / Wet	8.33	8.25	7
15	Arabica	China	Catimor	Washed / Wet	8.42	8.25	8
16	Arabica	Ethiopia	Ethiopian Yirgacheffe	Natural / Dry	8.17	8.17	8
17	Arabica	United States	Other	Washed / Wet	8.00	8.25	8
18	Arabica	Costa Rica	Caturra	Washed / Wet	8.08	8.25	8
19	Arabica	Mexico	Other	Washed / Wet	8.17	8.25	8

```
In [49]: df.columns
```

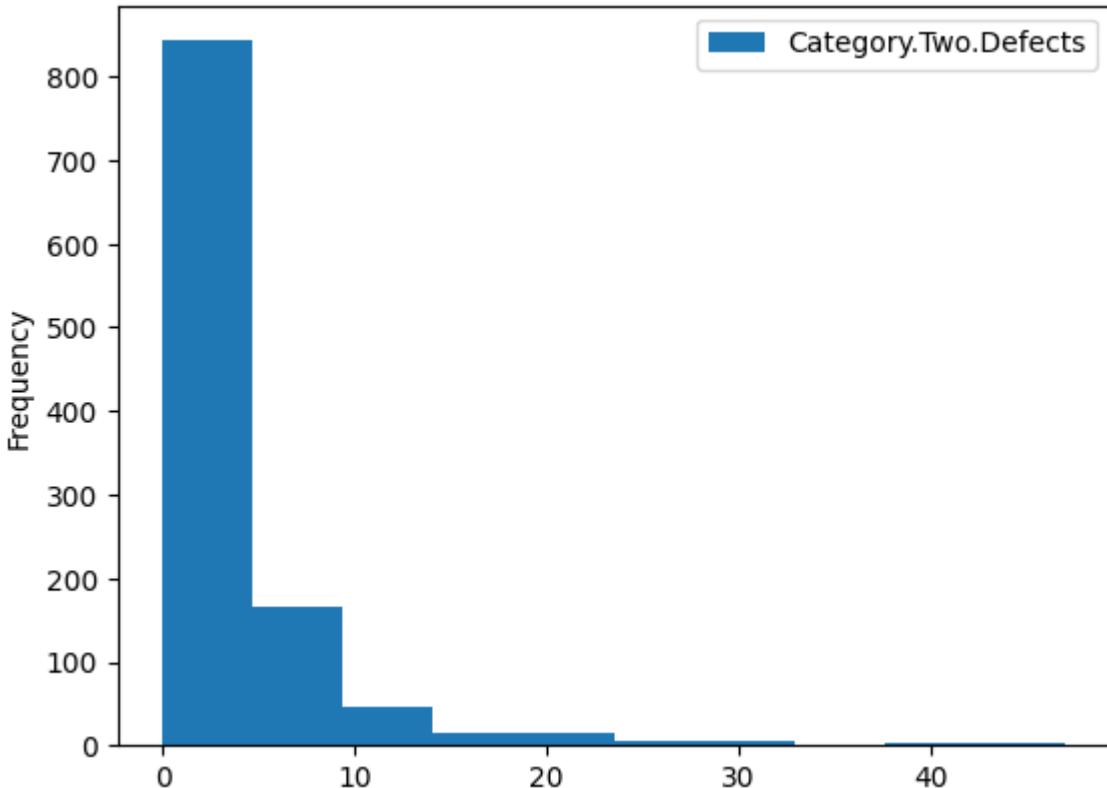
```
Out[49]: Index(['Species', 'Country.of.Origin', 'Variety', 'Processing.Method', 'Aroma',  
               'Flavor', 'Aftertaste', 'Acidity', 'Body', 'Balance', 'Uniformity',  
               'Clean.Cup', 'Sweetness', 'Cupper.Points', 'Total.Cup.Points',  
               'Moisture', 'Category.One.Defects', 'Quakers', 'Color',  
               'Category.Two.Defects', 'altitude_mean_meters'],  
              dtype='object')
```

```
In [50]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1103 entries, 0 to 1102  
Data columns (total 21 columns):  
 #   Column           Non-Null Count  Dtype     
 ---  --  
 0   Species          1103 non-null    object    
 1   Country.of.Origin 1103 non-null    object    
 2   Variety          1103 non-null    object    
 3   Processing.Method 1103 non-null    object    
 4   Aroma             1103 non-null    float64   
 5   Flavor            1103 non-null    float64   
 6   Aftertaste        1103 non-null    float64   
 7   Acidity           1103 non-null    float64   
 8   Body              1103 non-null    float64   
 9   Balance           1103 non-null    float64   
 10  Uniformity        1103 non-null    float64   
 11  Clean.Cup         1103 non-null    float64   
 12  Sweetness          1103 non-null    float64   
 13  Cupper.Points     1103 non-null    float64   
 14  Total.Cup.Points  1103 non-null    float64   
 15  Moisture          1103 non-null    float64   
 16  Category.One.Defects 1103 non-null    int64     
 17  Quakers           1103 non-null    float64   
 18  Color              1103 non-null    object    
 19  Category.Two.Defects 1103 non-null    int64     
 20  altitude_mean_meters 1103 non-null    float64  
dtypes: float64(14), int64(2), object(5)  
memory usage: 181.1+ KB
```

```
In [51]: df[['Category.Two.Defects']].plot.hist()
```

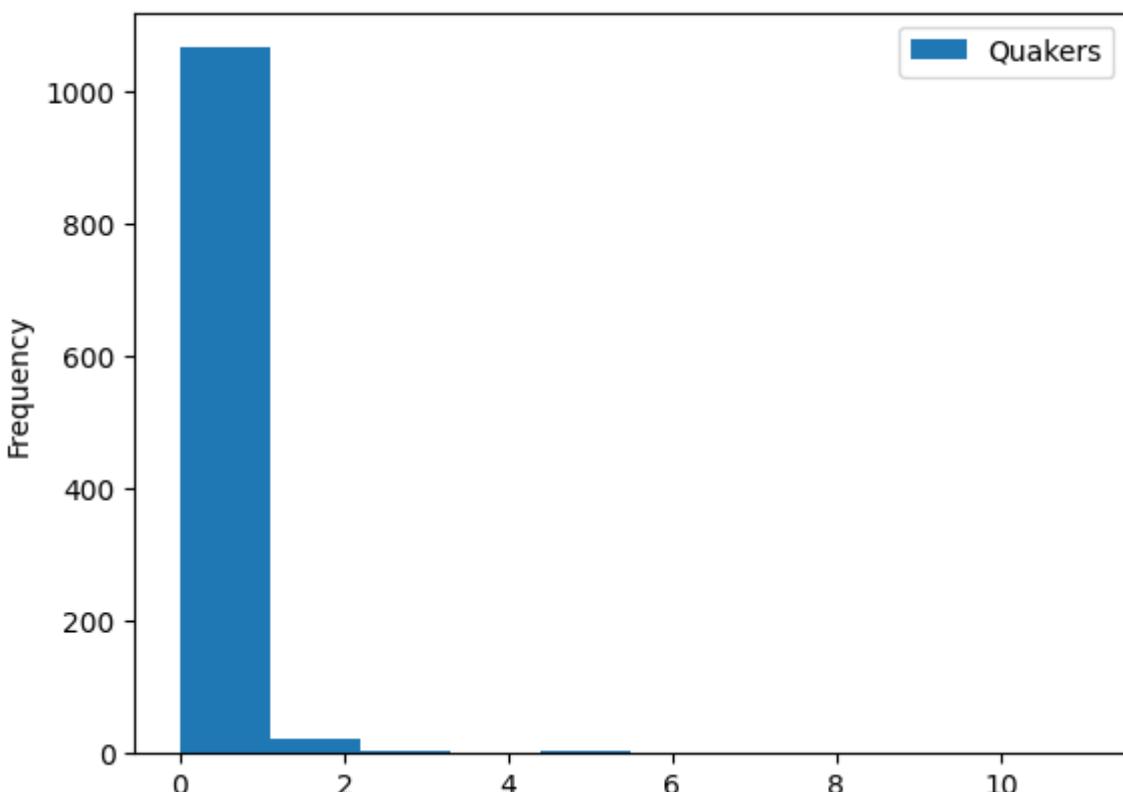
```
Out[51]: <Axes: ylabel='Frequency'>
```



We're gonna keep these outlying values, as they might have a high impact on the quality score

```
In [52]: df[['Quakers']].plot.hist()
```

```
Out[52]: <Axes: ylabel='Frequency'>
```



Data Exploration

Lets see which countries can boast the highest cup grades

```
In [53]: df.groupby('Country.of.Origin')['Total.Cup.Points'].mean().reset_index()
```

Out [53] :

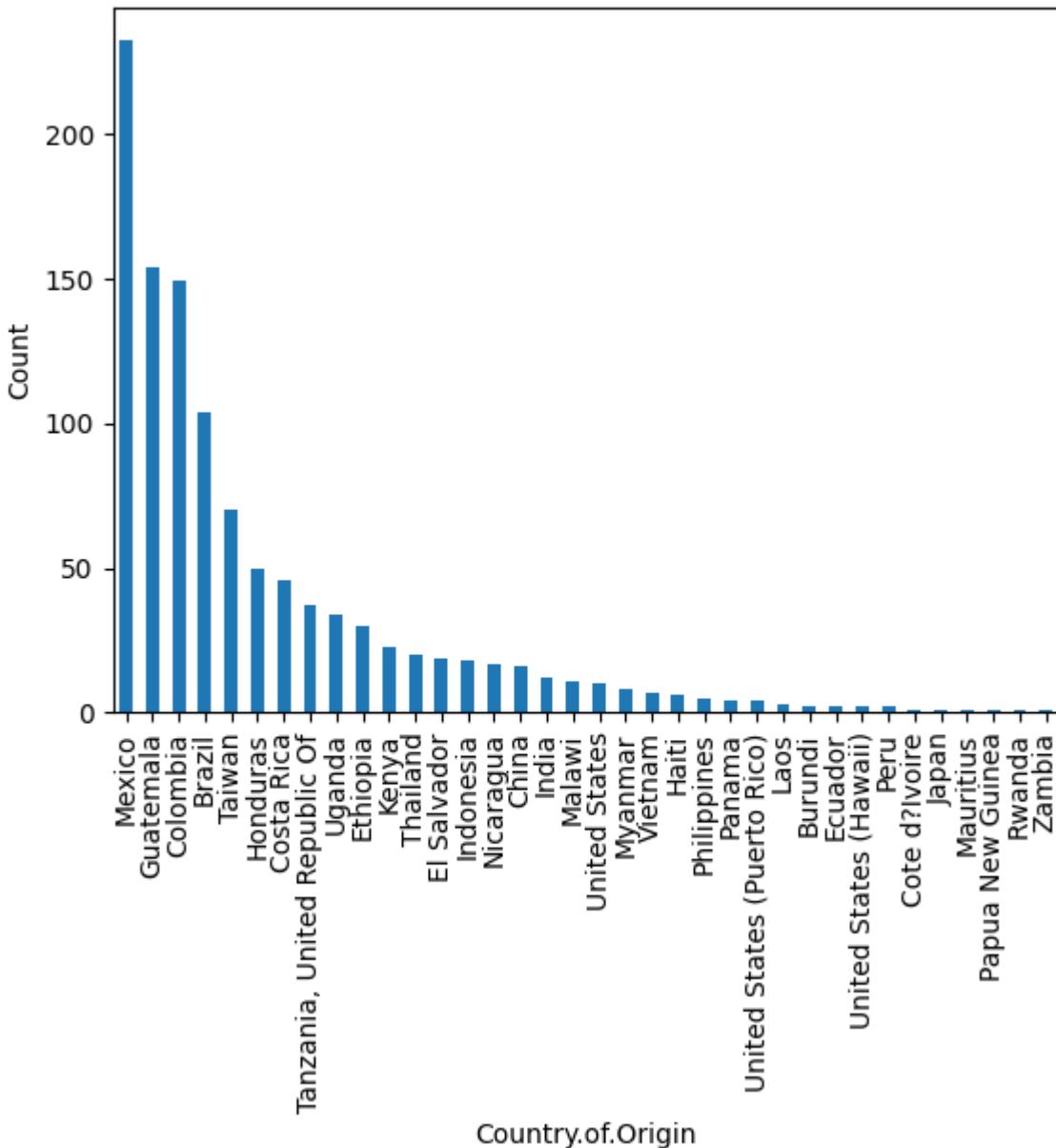
	Country.of.Origin	Total.Cup.Points
32	United States (Hawaii)	86.960000
8	Ethiopia	85.916333
23	Papua New Guinea	85.750000
14	Japan	84.670000
31	United States	84.433000
15	Kenya	84.271304
22	Panama	83.707500
30	Uganda	83.382941
3	Colombia	83.224832
7	El Salvador	83.115263
2	China	82.927500
26	Rwanda	82.830000
4	Costa Rica	82.800435
0	Brazil	82.711442
13	Indonesia	82.528333
29	Thailand	82.430000
28	Tanzania, United Republic Of	82.309459
34	Vietnam	82.274286
9	Guatemala	82.024221
27	Taiwan	81.947714
12	India	81.937500
35	Zambia	81.920000
16	Laos	81.833333
1	Burundi	81.830000
33	United States (Puerto Rico)	81.727500
17	Malawi	81.711818
6	Ecuador	80.955000
19	Mexico	80.863060
25	Philippines	80.834000
11	Honduras	80.832200
20	Myanmar	80.750000
18	Mauritius	80.500000
21	Nicaragua	80.010000
5	Cote d'Ivoire	79.330000
24	Peru	78.000000

	Country.of.Origin	Total.Cup.Points
10	Haiti	77.180000

Seems like a mixed bag from around the world. Quality doesn't seem to be linked regionally, but there is a small majority of african countries in the top half, and Hawaiian coffee seems to be a specialty!

```
In [54]: df.groupby('Country.of.Origin').size().sort_values(ascending=False).plot.
```

```
Out[54]: <Axes: xlabel='Country.of.Origin', ylabel='Count'>
```



Though we see that there is a very skewed representation of countries in the dataset. About 1/4 of the coffee in the dataset is from Mexico!

```
In [55]: df.groupby('Country.of.Origin')['Total.Cup.Points'].median().reset_index()
```

Out [55] :

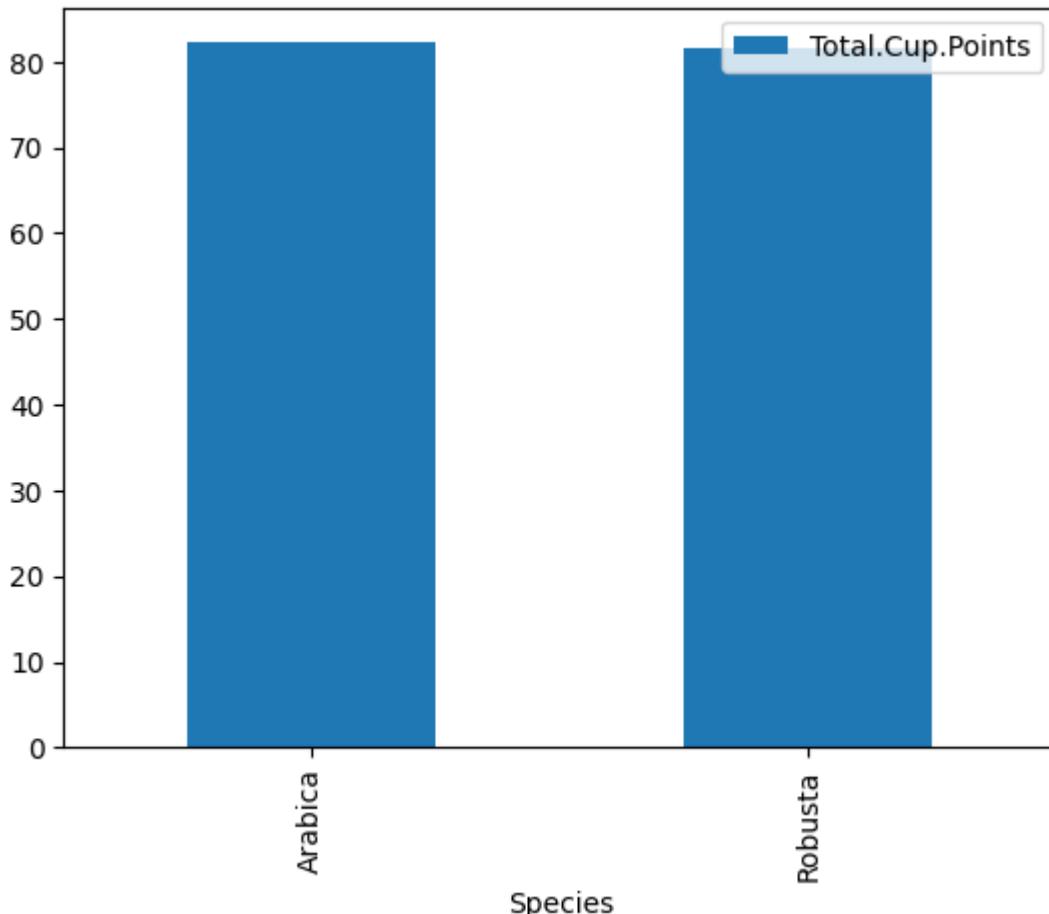
	Country.of.Origin	Total.Cup.Points
32	United States (Hawaii)	86.960
31	United States	86.625
23	Papua New Guinea	85.750
8	Ethiopia	85.250
14	Japan	84.670
15	Kenya	84.580
22	Panama	84.125
3	Colombia	83.250
2	China	83.170
4	Costa Rica	83.165
30	Uganda	83.085
7	El Salvador	82.920
34	Vietnam	82.830
26	Rwanda	82.830
13	Indonesia	82.665
29	Thailand	82.540
9	Guatemala	82.540
0	Brazil	82.500
28	Tanzania, United Republic Of	82.170
12	India	82.040
33	United States (Puerto Rico)	82.040
16	Laos	82.000
35	Zambia	81.920
27	Taiwan	81.875
1	Burundi	81.830
11	Honduras	81.625
17	Malawi	81.580
19	Mexico	81.580
25	Philippines	81.330
6	Ecuador	80.955
21	Nicaragua	80.920
20	Myanmar	80.625
18	Mauritius	80.500
5	Cote d'Ivoire	79.330
10	Haiti	79.000

Country.of.Origin	Total.Cup.Points	
24	Peru	78.000

Taking the median values, it seems roughly about the same distribution

```
In [56]: df.groupby('Species')['Total.Cup.Points'].mean().reset_index().sort_values
```

```
Out[56]: <Axes: xlabel='Species'>
```



Surprisingly, The Robusta species does not seem to have a much lower average score than Arabica, which we thought were far superior in taste

```
In [57]: corr_mat = df.corr(numeric_only=True)  
corr_mat
```

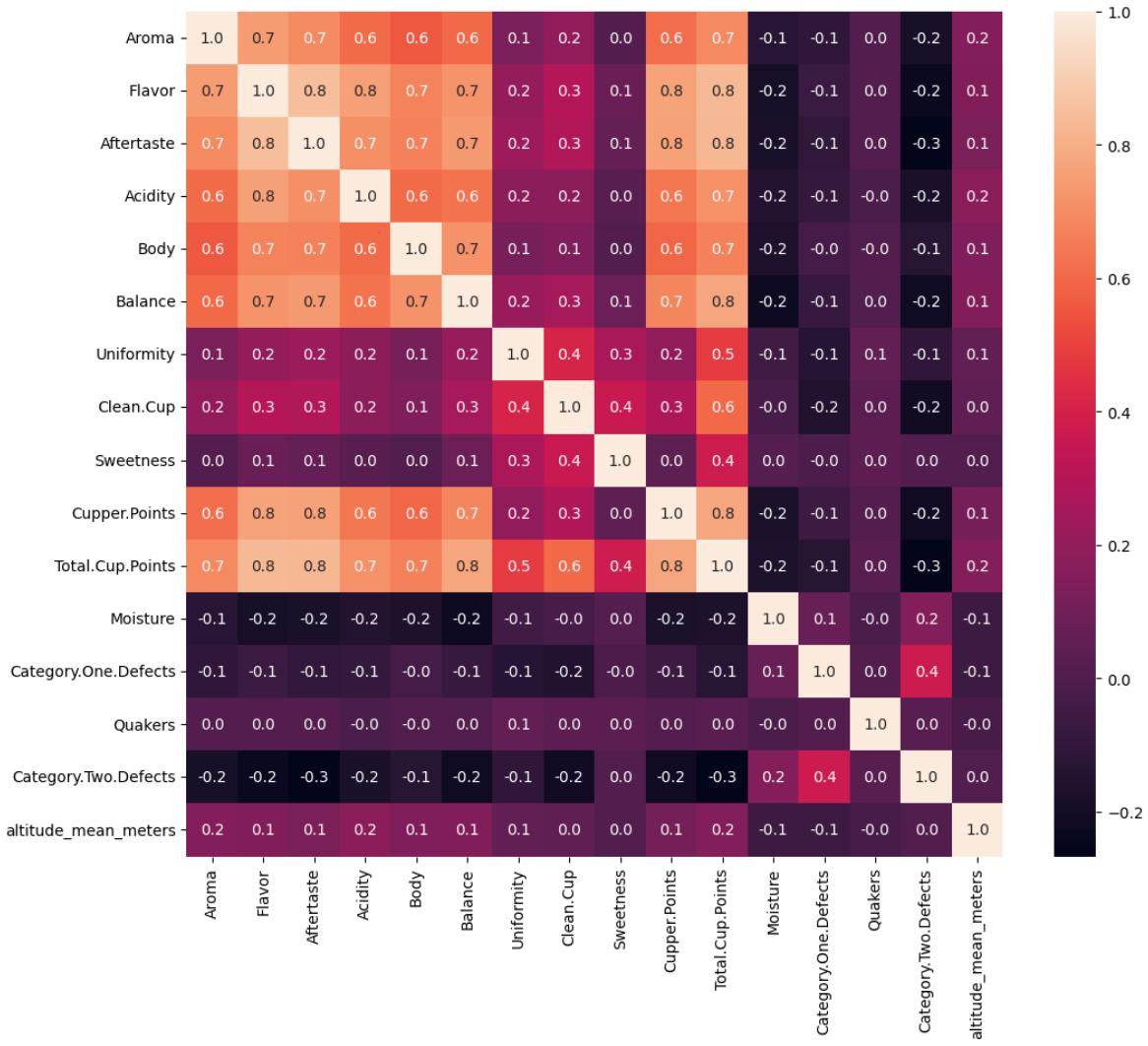
Out [57] :

	Aroma	Flavor	Aftertaste	Acidity	Body	Balance
Aroma	1.000000	0.743109	0.688130	0.606707	0.558217	0.602133
Flavor	0.743109	1.000000	0.849983	0.757027	0.668397	0.722149
Aftertaste	0.688130	0.849983	1.000000	0.705410	0.666111	0.737439
Acidity	0.606707	0.757027	0.705410	1.000000	0.604118	0.624788
Body	0.558217	0.668397	0.666111	0.604118	1.000000	0.715045
Balance	0.602133	0.722149	0.737439	0.624788	0.715045	1.000000
Uniformity	0.126962	0.207079	0.226183	0.178229	0.116480	0.215432
Clean.Cup	0.196811	0.297936	0.288698	0.176823	0.140021	0.254276
Sweetness	0.008310	0.077417	0.063372	0.022285	0.002498	0.083593
Cupper.Points	0.611088	0.764423	0.761057	0.642121	0.590999	0.679213
Total.Cup.Points	0.688725	0.836167	0.827814	0.708001	0.665436	0.775314
Moisture	-0.131316	-0.189600	-0.178685	-0.151101	-0.171327	-0.227089
Category.One.Defects	-0.104645	-0.069596	-0.103180	-0.090468	-0.035044	-0.079425
Quakers	0.007763	0.009010	0.007843	-0.017353	-0.002792	0.001002
Category.Two.Defects	-0.184304	-0.233813	-0.263210	-0.181162	-0.138092	-0.218804
altitude_mean_meters	0.155328	0.148021	0.133273	0.181499	0.142243	0.144477

In [58] :

```
plt.figure(figsize=(12, 10))
sns.heatmap(corr_mat, annot = True, fmt = ".1f")
```

Out [58] : <Axes: >



Unsurprisingly the different flavor parameters seems to correlate a lot with the Total Cup Points, and with each other. It seems that the main parameters that drags down the score is the amount of Category One and Two Defects, which we also suspected. Quakers doesn't seem to have a big impact on the overall Cup Point score. This might, of course, be biased by the low amount of values above zero

```
In [59]: df.dtypes
```

```
Out[59]: Species          object
Country.of.Origin      object
Variety                object
Processing.Method      object
Aroma                  float64
Flavor                 float64
Aftertaste              float64
Acidity                float64
Body                   float64
Balance                float64
Uniformity              float64
Clean.Cup               float64
Sweetness               float64
Cupper.Points           float64
Total.Cup.Points        float64
Moisture                float64
Category.One.Defects    int64
Quakers                float64
Color                  object
Category.Two.Defects    int64
altitude_mean_meters    float64
dtype: object
```

Lets label encode the categorical data into discrete values, so we can see how the nominal data affects the correlation score as well.

```
In [60]: le = LabelEncoder()
cols_to_enc = df.select_dtypes(include='object').columns
enc_df = df.copy()

for col in cols_to_enc:
    enc_df[col] = le.fit_transform(df[col]).astype(int)

enc_df
```

	Species	Country.of.Origin	Variety	Processing.Method	Aroma	Flavor	Aftertasi	
0	0		8	5	4	8.67	8.83	8.6
1	0		8	13	4	8.75	8.67	8.5
2	0		9	2	4	8.42	8.50	8.4
3	0		8	5	0	8.17	8.58	8.4
4	0		8	13	4	8.25	8.50	8.2
...
1098	1		12	5	4	7.67	7.67	7.5
1099	1		12	5	0	7.58	7.42	7.4
1100	1		31	0	0	7.92	7.50	7.4
1101	1		6	5	4	7.50	7.67	7.7
1102	1		31	5	0	7.33	7.33	7.1

1103 rows × 21 columns

```
In [61]: enc_df.to_csv("../data_cleaned/encoded_df.csv", index=False)
```

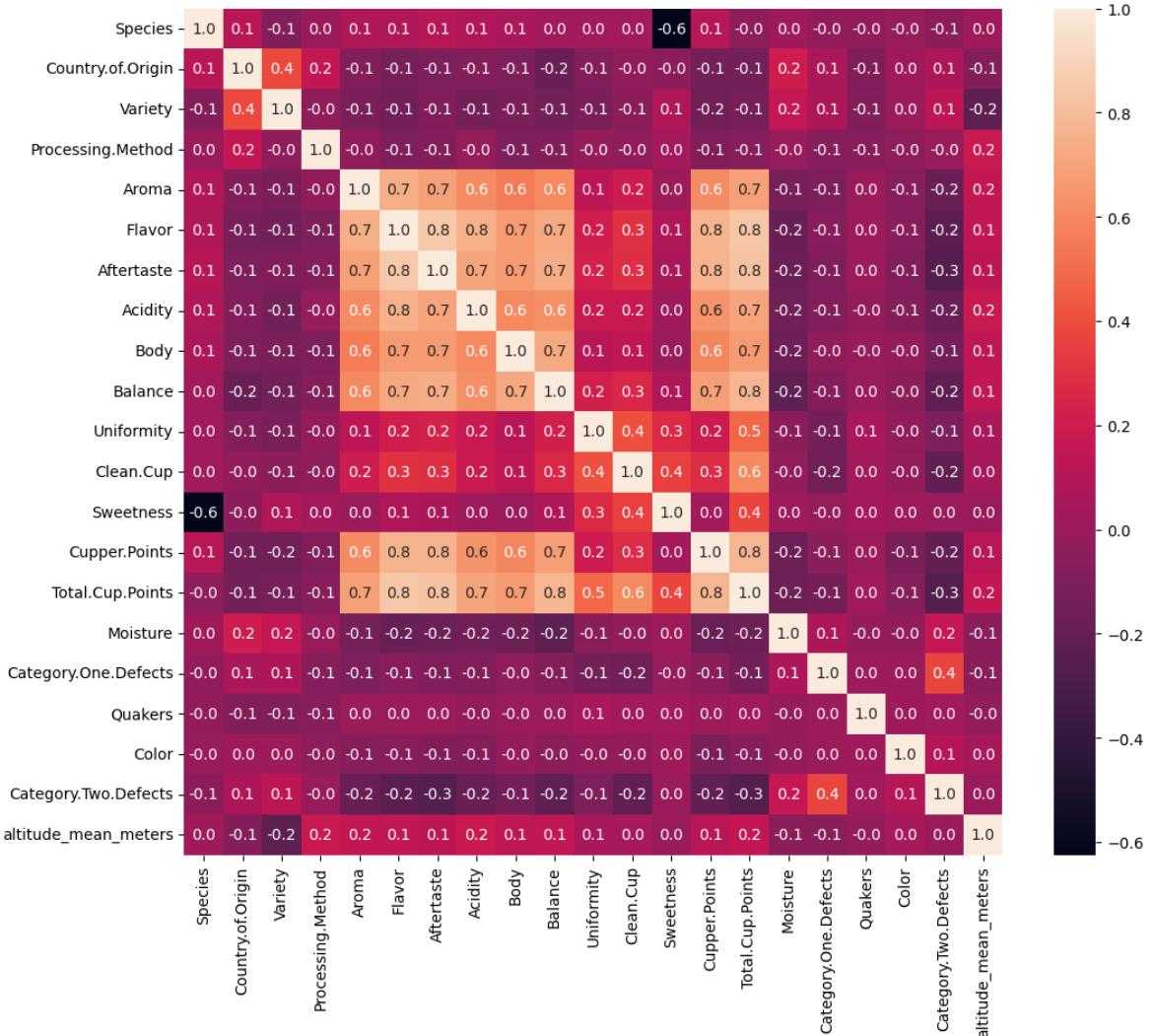
Lets see how much variance it give us

```
In [62]: df.describe()
```

	Aroma	Flavor	Aftertaste	Acidity	Body	Balance
count	1103.000000	1103.000000	1103.000000	1103.000000	1103.000000	1103.000000
mean	7.578368	7.527824	7.401496	7.535739	7.513654	7.512665
std	0.309181	0.331268	0.340065	0.313192	0.289467	0.354021
min	5.080000	6.170000	6.170000	5.250000	5.170000	5.250000
25%	7.420000	7.330000	7.250000	7.330000	7.330000	7.330000
50%	7.580000	7.580000	7.420000	7.500000	7.500000	7.500000
75%	7.750000	7.750000	7.580000	7.750000	7.670000	7.750000
max	8.750000	8.830000	8.670000	8.750000	8.580000	8.750000

```
In [63]: enc_corr_mat = enc_df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(enc_corr_mat, annot = True, fmt = ".1f")
```

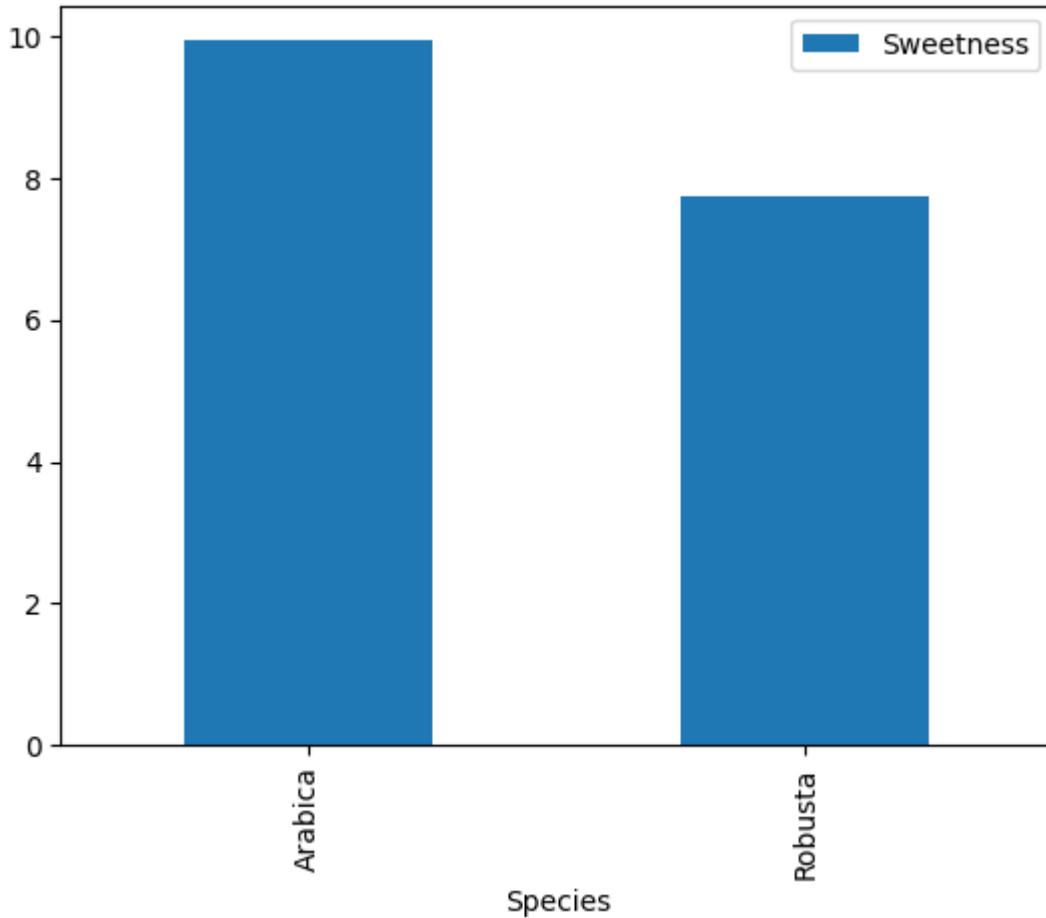
Out [63]: <Axes: >



The species of the coffee seems to have a significant impact on the sweetness

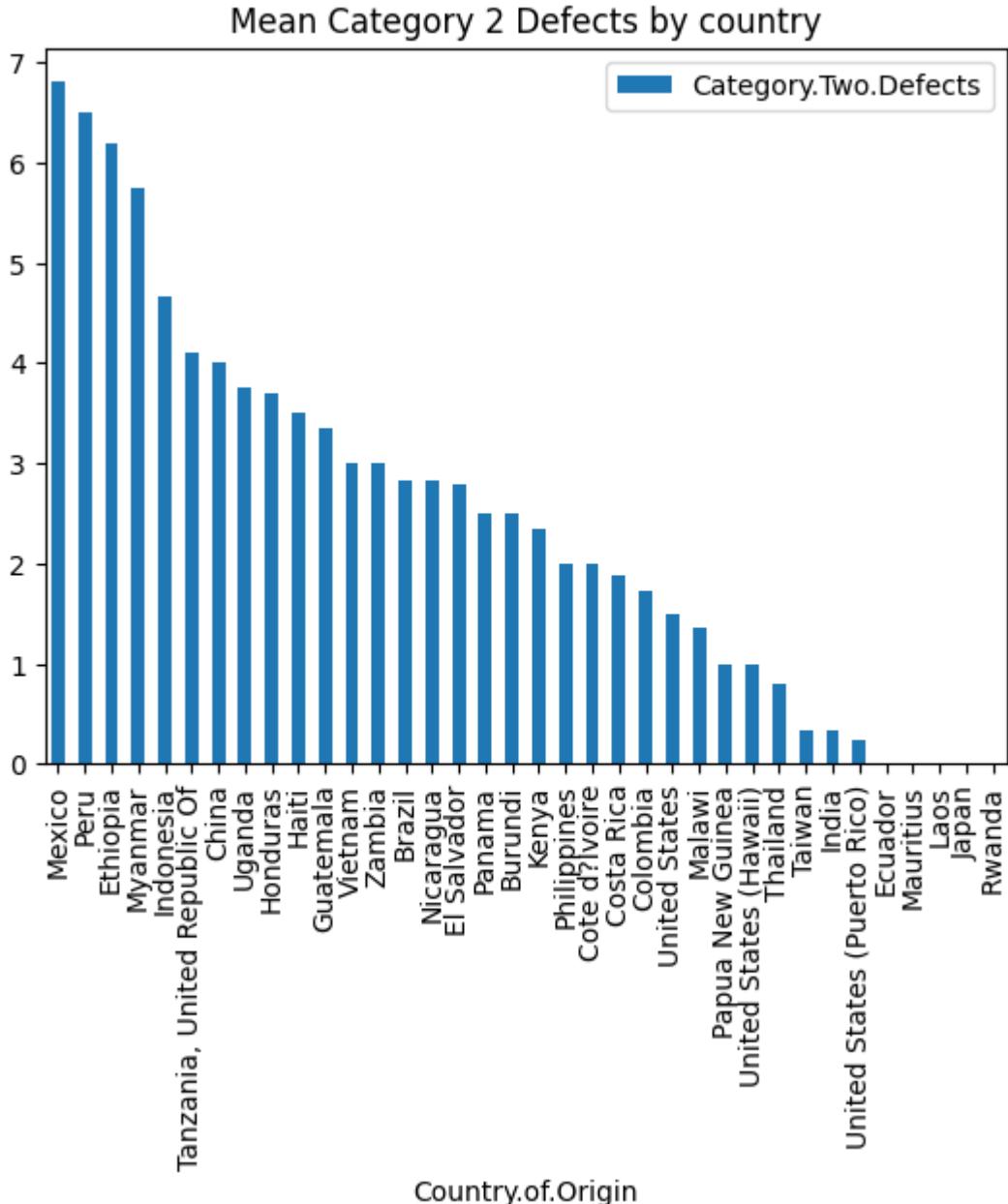
```
In [64]: # Sweetnes grouped by species
df.groupby('Species')['Sweetness'].mean().reset_index().plot.bar(x='Species')
```

Out [64]: <Axes: xlabel='Species'>



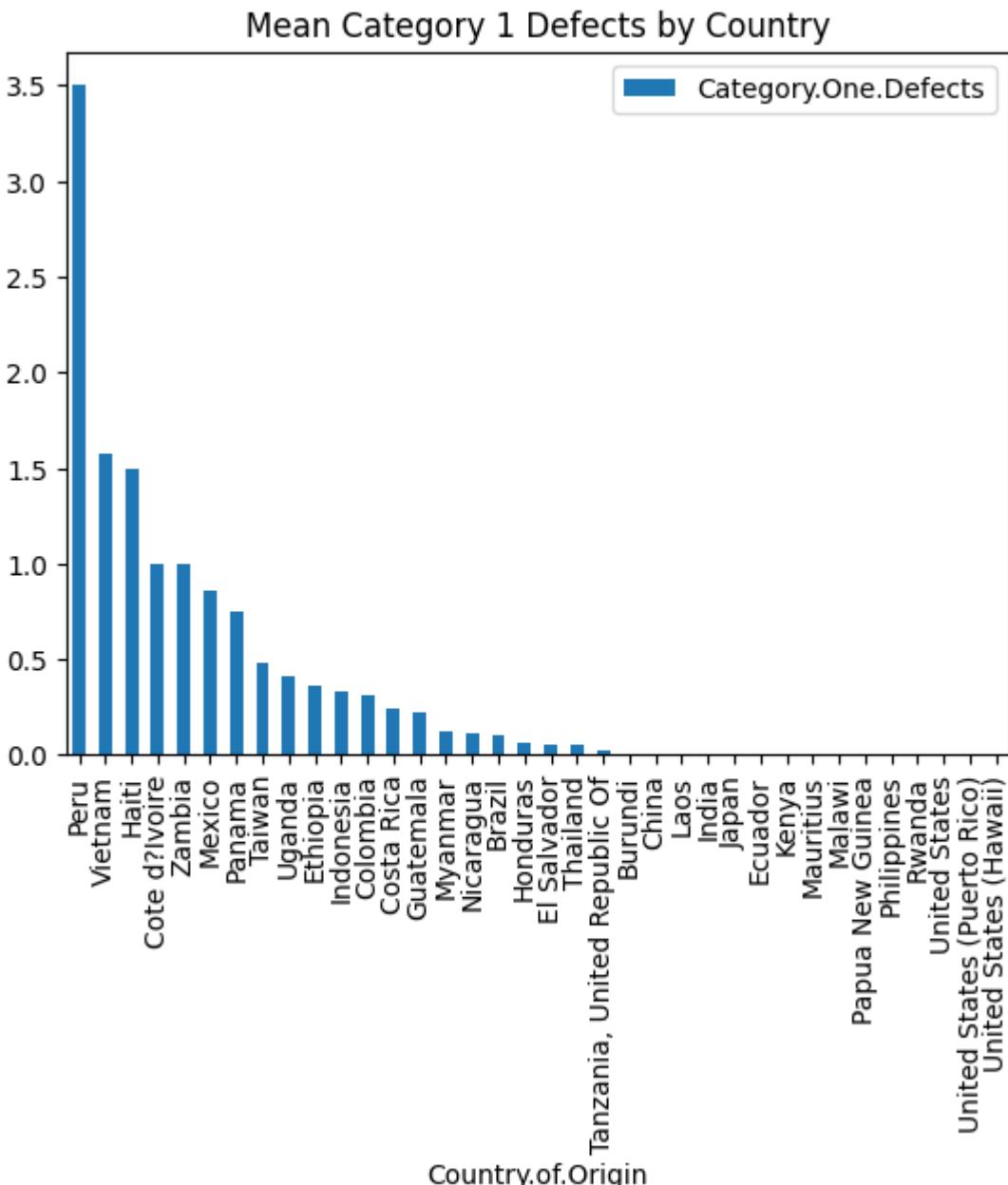
So this is where the Arabica beans stand out! It's on average around 20% sweeter than Robusta beans

```
In [ ]: df.groupby('Country.of-Origin')['Category.Two.Defects'].mean().reset_index()
Out[ ]: <Axes: title={'center': 'Mean Category 2 Defects by country'}, xlabel='Co
untry.of.Origin'>
```



```
In [106]: df.groupby('Country.of-Origin')[['Category.One.Defects']].mean().reset_index()
```

```
Out[106]: <Axes: title={'center': 'Mean Category 1 Defects by Country'}, xlabel='Country.of.Origin'>
```



Interestingly though, we also find Ethiopia among the countries with the highest mean in Category 2 defect parameters, even though it is the country with the 2nd averagely highest scoring coffee. This indicates that it takes rather large amounts of defects in the coffee to really make an impact on the overall score.

Clustering model training

We'll try to extract more information with clustering models. For this purpose we drop our previous target feature "Total Cup Points" and scale the feature values in the dataset, so we can do a principal component analysis

```
In [67]: df_cls = enc_df.drop('Total.Cup.Points', axis=1)
# df_cls = enc_df.copy()
```

```
In [68]: df_cls[df_cls.columns] = sk.preprocessing.StandardScaler().fit_transform(df)
```

```
In [69]: df_cls.sample(5)
```

	Species	Country.of.Origin	Variety	Processing.Method	Aroma	Flavor
708	-0.152286	0.605495	-0.958392	0.526081	0.814236	-0.597444
858	-0.152286	1.443464	1.635319	0.526081	-0.512451	-0.839050
1027	-0.152286	-1.175189	1.635319	0.526081	-0.803674	-1.080656
723	-0.152286	-0.441966	-0.958392	0.526081	-0.512451	-0.325637
409	-0.152286	-0.441966	-0.958392	0.526081	0.814236	0.429382

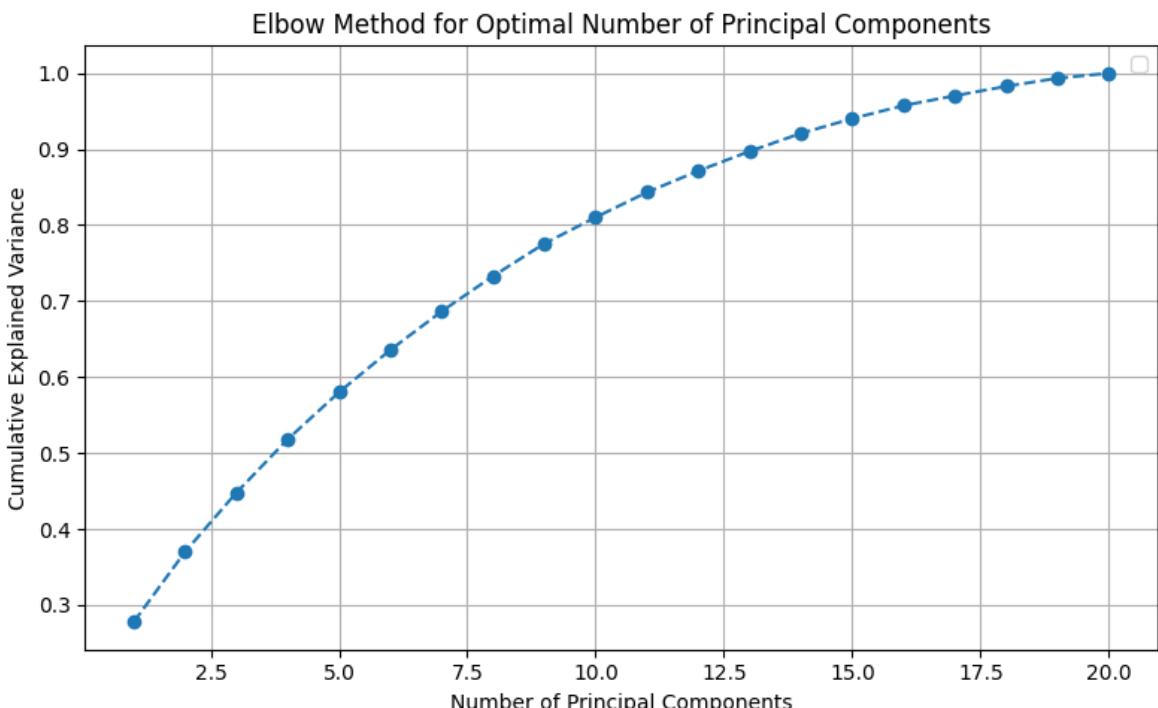
```
In [70]: pca = PCA()
pca.fit(df_cls)
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)
# 3 principal components
cumulative_variance[2]
```

Out [70]: np.float64(0.4478511071920876)

```
In [71]: plt.figure(figsize=(8, 5))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Elbow Method for Optimal Number of Principal Components')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

/tmp/ipykernel_2562/3103998374.py:7: UserWarning: No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.

plt.legend()



Not as much as we hoped for. For 3 principal components, we get just above 50% explained variance, which is not a lot. Lets try and extract high correlating features and

do the PCA again

```
In [72]: df_cls_ext = enc_df[["Aroma", "Flavor", "Acidity", "Body", "Balance", "Aftertaste"]]  
df_cls_ext[df_cls_ext.columns] = sk.preprocessing.StandardScaler().fit_transform(df_cls_ext)
```

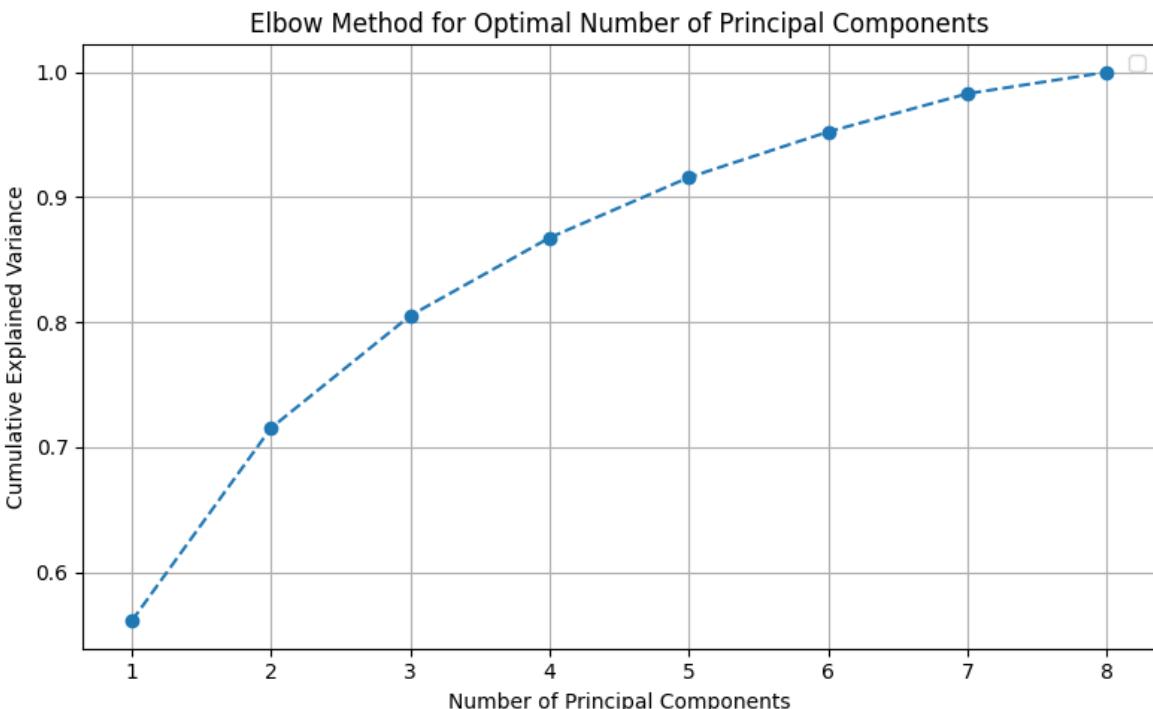
```
In [73]: pca = PCA()  
pca.fit(df_cls_ext)  
explained_variance = pca.explained_variance_ratio_  
cumulative_variance = np.cumsum(explained_variance)  
# 3 principal components  
cumulative_variance[2]
```

```
Out[73]: np.float64(0.8053004618615184)
```

```
In [74]: plt.figure(figsize=(8, 5))  
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o', linestyle='dashed')  
plt.xlabel('Number of Principal Components')  
plt.ylabel('Cumulative Explained Variance')  
plt.title('Elbow Method for Optimal Number of Principal Components')  
plt.grid(True)  
plt.legend()  
plt.tight_layout()  
plt.show()
```

/tmp/ipykernel_2562/3103998374.py:7: UserWarning: No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.

```
plt.legend()
```



Now we have an explained variance of above 70% with 2 principal components. Much better!

```
In [75]: pca_2 = PCA(n_components=2)  
pca_2_result = pca_2.fit_transform(df_cls_ext)  
dataset_pca = pd.DataFrame(abs(pca_2.components_), columns=df_cls_ext.columns)
```

	Aroma	Flavor	Acidity	Body	Balance	Aftertaste	Uniformity	Sweet
PC_1	0.381225	0.436044	0.392771	0.381476	0.404040	0.428197	0.128396	0.04
PC_2	0.103547	0.019839	0.057700	0.113975	0.009413	0.009358	0.659328	0.73

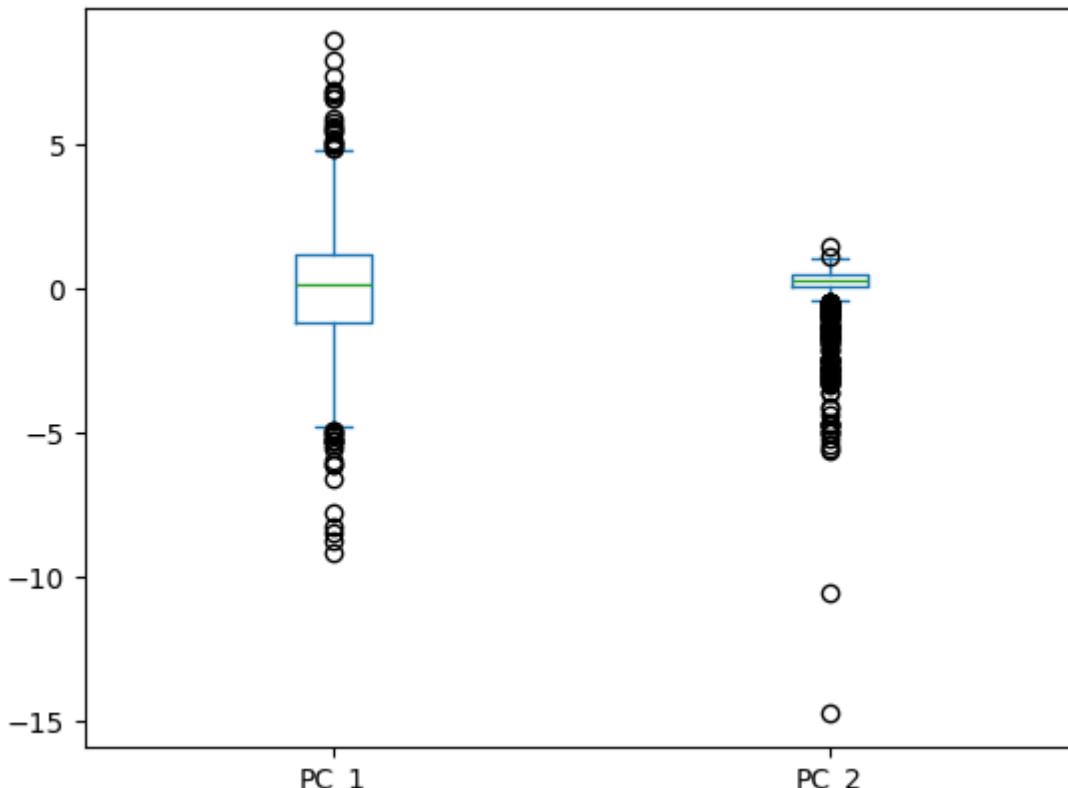
Here is our principal components. We see that PC1 is weighted across all the features, with a dive in Uniformity and Sweetness, while PC2 is mostly weighted by these.

```
In [76]: df_pca = pd.DataFrame(pca_2_result, columns=['PC_1', 'PC_2'])
df_pca.head()
```

	PC_1	PC_2
0	8.565309	-0.713374
1	7.920370	-0.663066
2	6.869235	-0.475166
3	6.696222	-0.467683
4	6.561643	-0.466118

```
In [77]: df_pca.plot.box()
```

```
Out[77]: <Axes: >
```

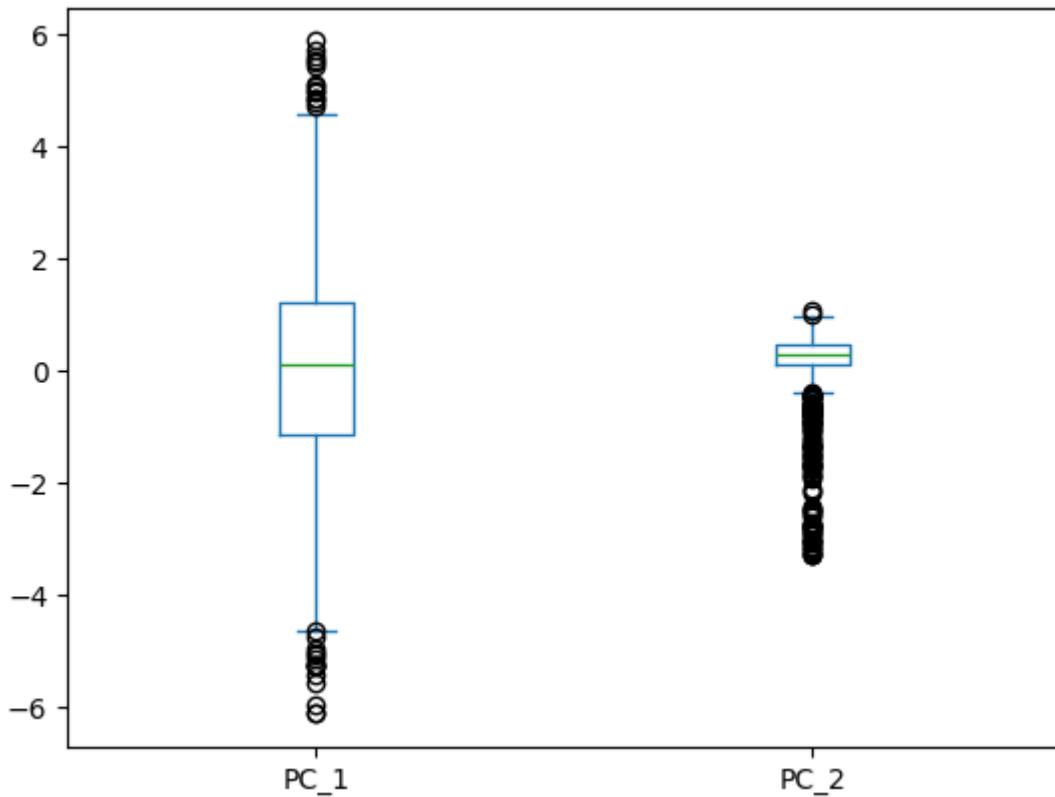


We have also introduced some serious outliers. Lets remove them

```
In [78]: z_scores = np.abs(zscore(df_pca))
df_pca = df_pca[(z_scores < 3).all(axis=1)]
```

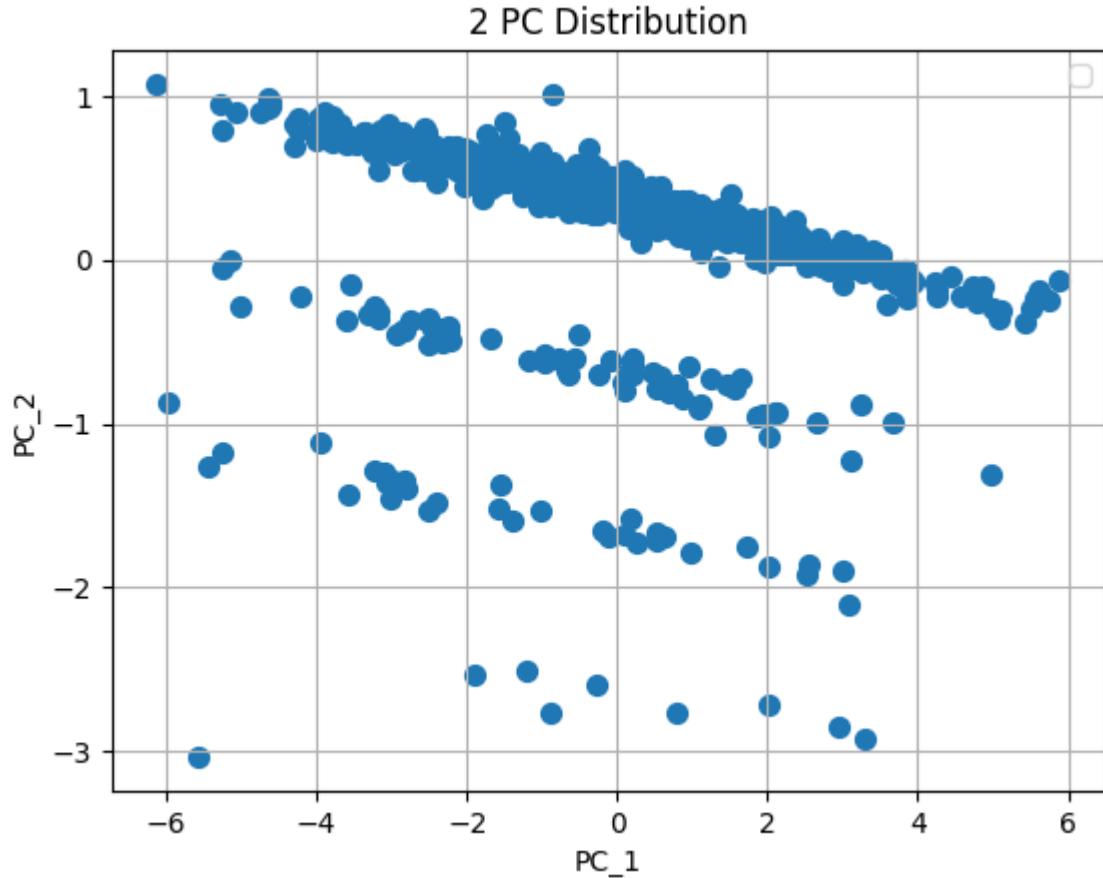
```
In [79]: df_pca.plot.box()
```

```
Out[79]: <Axes: >
```



```
In [109... fig = plt.figure()
plt.scatter(df_pca['PC_1'], df_pca['PC_2'], s=50)
plt.xlabel('PC_1')
plt.ylabel('PC_2')
plt.title("2 PC Distribution")
plt.grid(True)
plt.legend()
plt.show()
```

/tmp/ipykernel_2562/297791176.py:7: UserWarning: No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
plt.legend()



```
In [81]: # Calculating optimal number of clusters in a K-means algorithm using silhouette scores = []
K = range(2,10)
for k in K:
    model = KMeans(n_clusters=k, n_init=10)
    model.fit(df_pca)
    score = sk.metrics.silhouette_score(df_pca, model.labels_, metric='euclidean')
    print("\nNumber of clusters =", k)
    print("Silhouette score =", score)
    scores.append([k, score])
```

```
Number of clusters = 2
```

```
Silhouette score = 0.4864683885974626
```

```
Number of clusters = 3
```

```
Silhouette score = 0.4596864222774429
```

```
Number of clusters = 4
```

```
Silhouette score = 0.428968519554263
```

```
Number of clusters = 5
```

```
Silhouette score = 0.47649776806745087
```

```
Number of clusters = 6
```

```
Silhouette score = 0.4588818099987823
```

```
Number of clusters = 7
```

```
Silhouette score = 0.4704537161873324
```

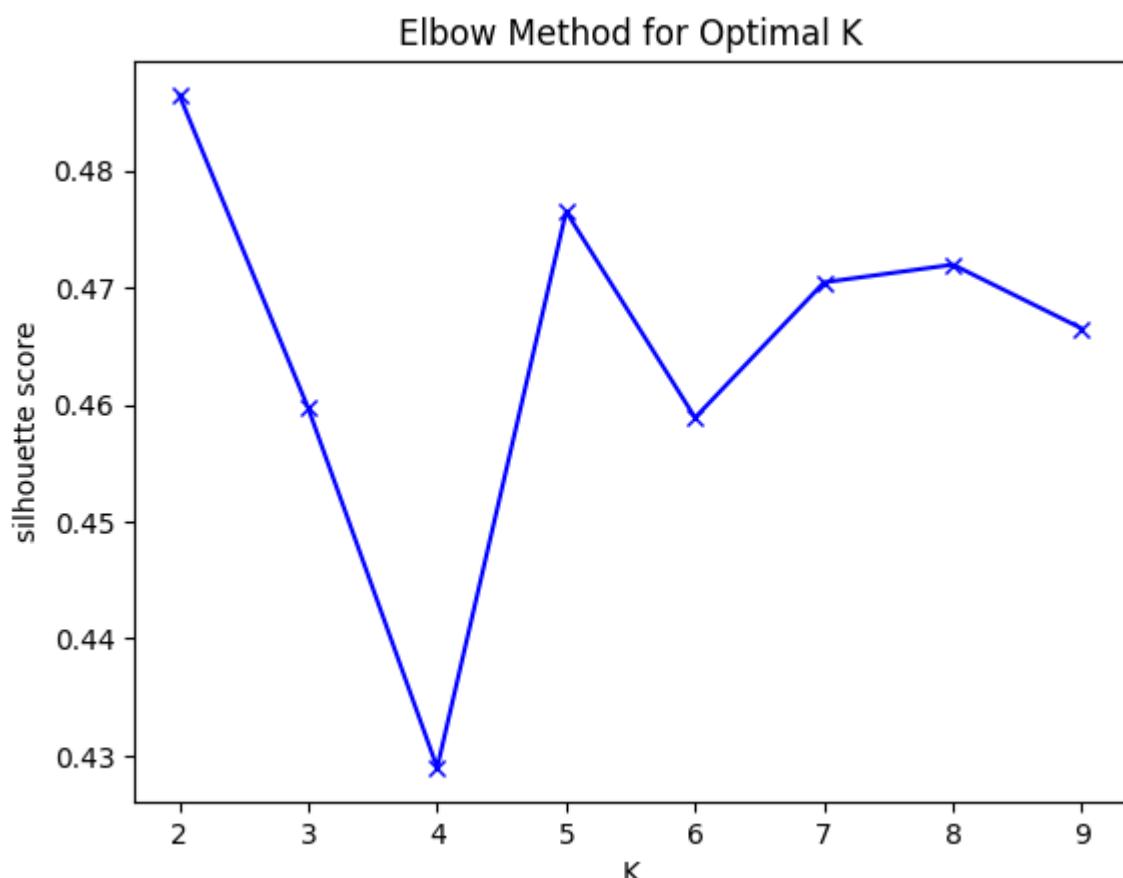
```
Number of clusters = 8
```

```
Silhouette score = 0.4719644914503527
```

```
Number of clusters = 9
```

```
Silhouette score = 0.46650507462084106
```

```
In [82]: score_df = pd.DataFrame(scores, columns=['k', 'scores'])
fig = plt.figure()
plt.title('Elbow Method for Optimal K')
plt.plot(score_df.k, score_df.scores, 'bx-')
plt.xlabel('K')
plt.ylabel('silhouette score')
plt.show()
```



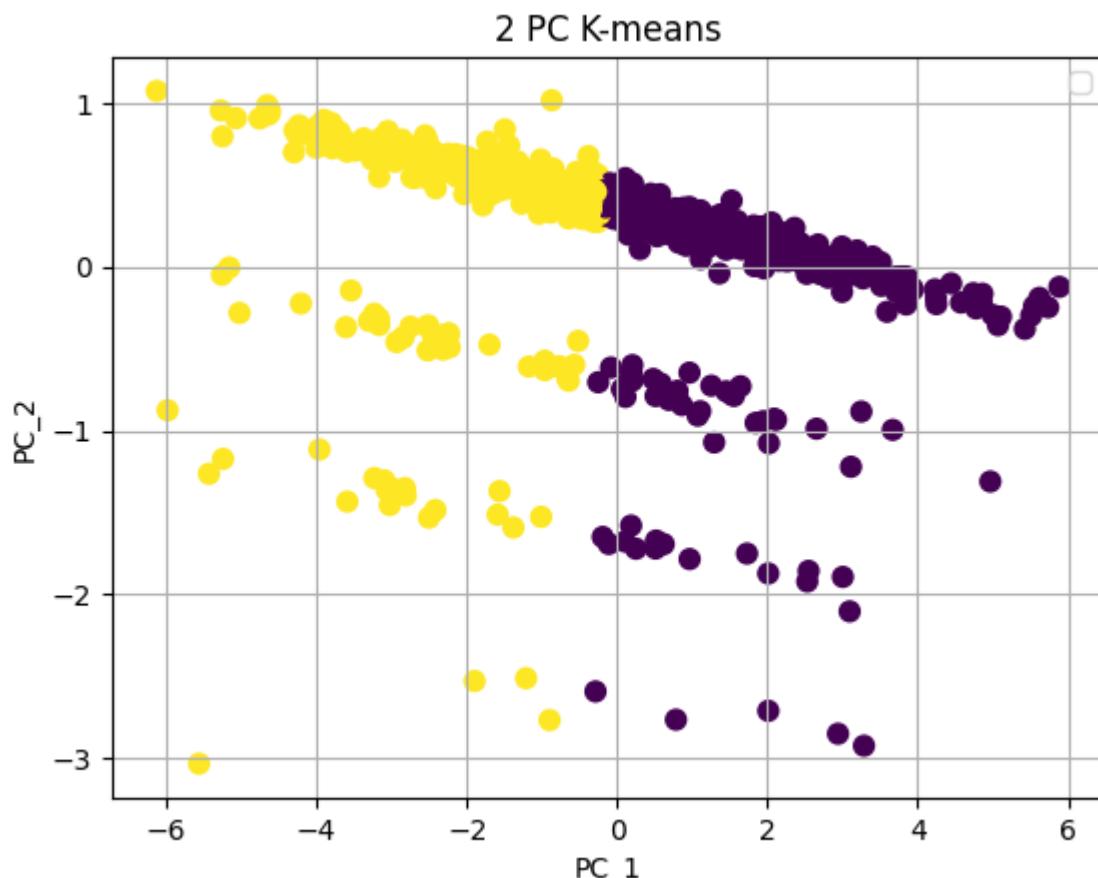
Using silhouette score for each amount of clusters, we see that we achieve a maximum score of 0,473 on 2 clusters. Let's try and visualize them

```
In [83]: kmeans = KMeans(init='k-means++', n_clusters=2, n_init=10)
prediction = kmeans.fit_predict(df_pca)
```

```
In [110]: fig = plt.figure()
plt.scatter(df_pca['PC_1'], df_pca['PC_2'], c=prediction, s=50, cmap='viridis')
plt.xlabel('PC_1')
plt.ylabel('PC_2')
plt.title("2 PC K-means")
plt.grid(True)
plt.legend()
plt.show()
```

/tmp/ipykernel_2562/1710067701.py:7: UserWarning: No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.

```
plt.legend()
```



Well, it doesn't seem like the K-means algorithm finds any clusters that makes visual sense. Lets try to the PCA again with an additional PC. The extra dimension and 10% added explained variance, might give additional insights

```
In [85]: pca_3 = PCA(n_components=3)
pca_3_result = pca_3.fit_transform(df_cls_ext)
dataset_pca = pd.DataFrame(abs(pca_3.components_), columns=df_cls_ext.columns)
dataset_pca
```

	Aroma	Flavor	Acidity	Body	Balance	Aftertaste	Uniformity	Sweet
PC_1	0.381225	0.436044	0.392771	0.381476	0.404040	0.428197	0.128396	0.04
PC_2	0.103547	0.019839	0.057700	0.113975	0.009413	0.009358	0.659328	0.73
PC_3	0.049970	0.040739	0.022757	0.067166	0.034935	0.002558	0.731547	0.67

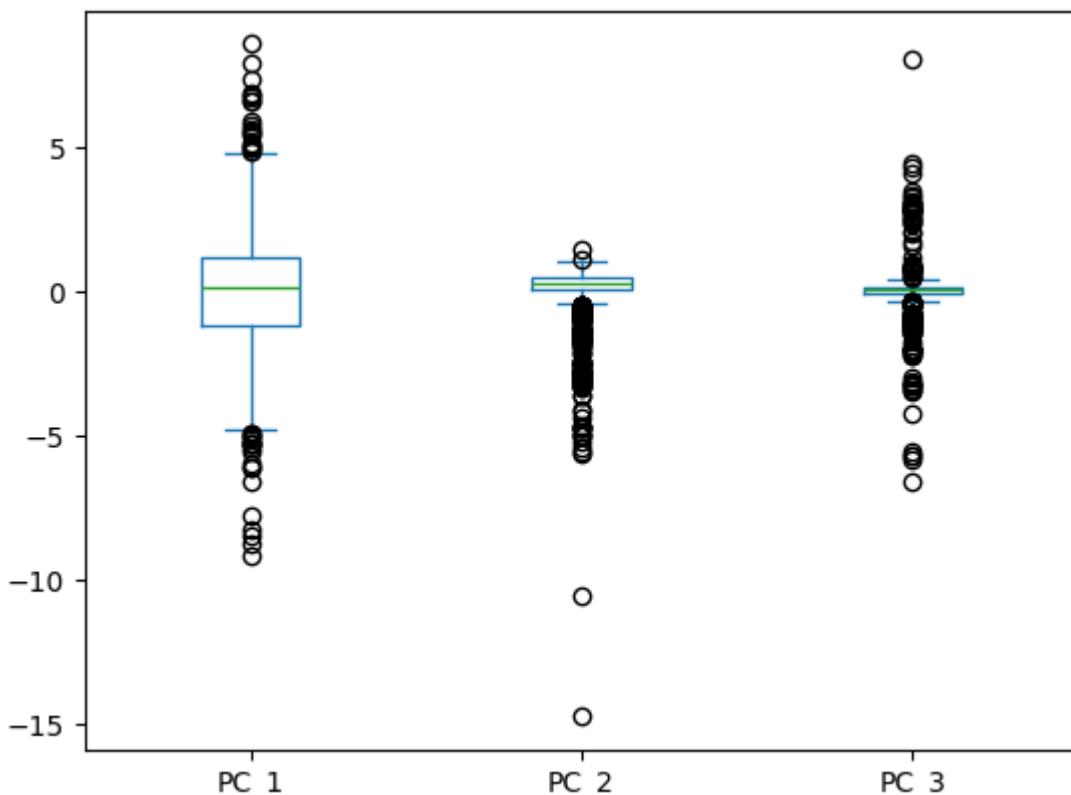
Here are our components. With PC1 weighted across most parameters, PC2 mostly weighted by Sweetness and Uniformity and PC3 weighted mainly by Uniformity

```
In [86]: df_pca = pd.DataFrame(pca_3_result, columns=['PC_1', 'PC_2', 'PC_3'])
df_pca.head()
```

	PC_1	PC_2	PC_3
0	8.565309	-0.713374	-0.498447
1	7.920370	-0.663066	-0.486764
2	6.869235	-0.475166	-0.403830
3	6.696222	-0.467683	-0.395929
4	6.561643	-0.466118	-0.383813

```
In [87]: df_pca.plot.box()
```

```
Out[87]: <Axes: >
```

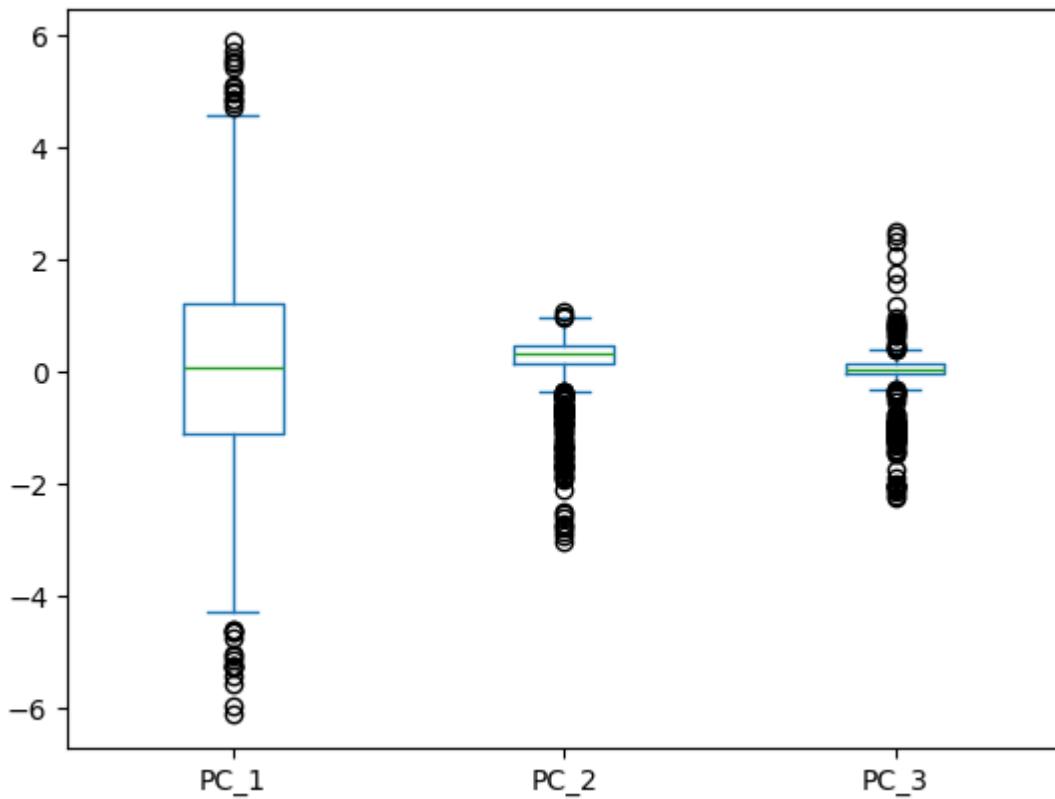


We'll remove the outliers again

```
In [88]: z_scores = np.abs(zscore(df_pca))
df_pca = df_pca[(z_scores < 3).all(axis=1)]
```

```
In [89]: df_pca.plot.box()
```

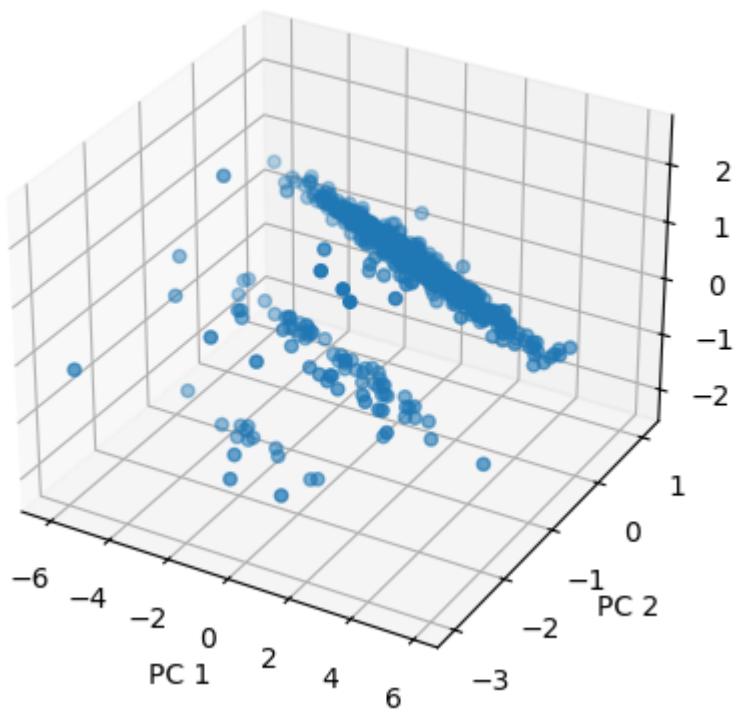
```
Out[89]: <Axes: >
```



```
In [90]: fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(df_pca['PC_1'], df_pca['PC_2'], df_pca['PC_3'])

ax.set_xlabel('PC 1')
ax.set_ylabel('PC 2')
ax.set_zlabel('PC 3')
```

```
Out[90]: Text(0.5, 0, 'PC 3')
```



Here is our 3D points. Let's start with doing a K-means clustering algorithm. First we need to find the optimal K (Number of clusters)

```
In [91]: # Calculating optimal number of clusters in a K-means algorithm using silhouette scores = []
K = range(2,10)
for k in K:
    model = KMeans(n_clusters=k, n_init=10)
    model.fit(df_pca)
    score = sk.metrics.silhouette_score(df_pca, model.labels_, metric='euclidean')
    print("\nNumber of clusters =", k)
    print("Silhouette score =", score)
    scores.append([k, score])
```

```
Number of clusters = 2
```

```
Silhouette score = 0.4898576695192006
```

```
Number of clusters = 3
```

```
Silhouette score = 0.4645804564710879
```

```
Number of clusters = 4
```

```
Silhouette score = 0.4464974447722136
```

```
Number of clusters = 5
```

```
Silhouette score = 0.4235502939312917
```

```
Number of clusters = 6
```

```
Silhouette score = 0.44154501142541325
```

```
Number of clusters = 7
```

```
Silhouette score = 0.46668632470795884
```

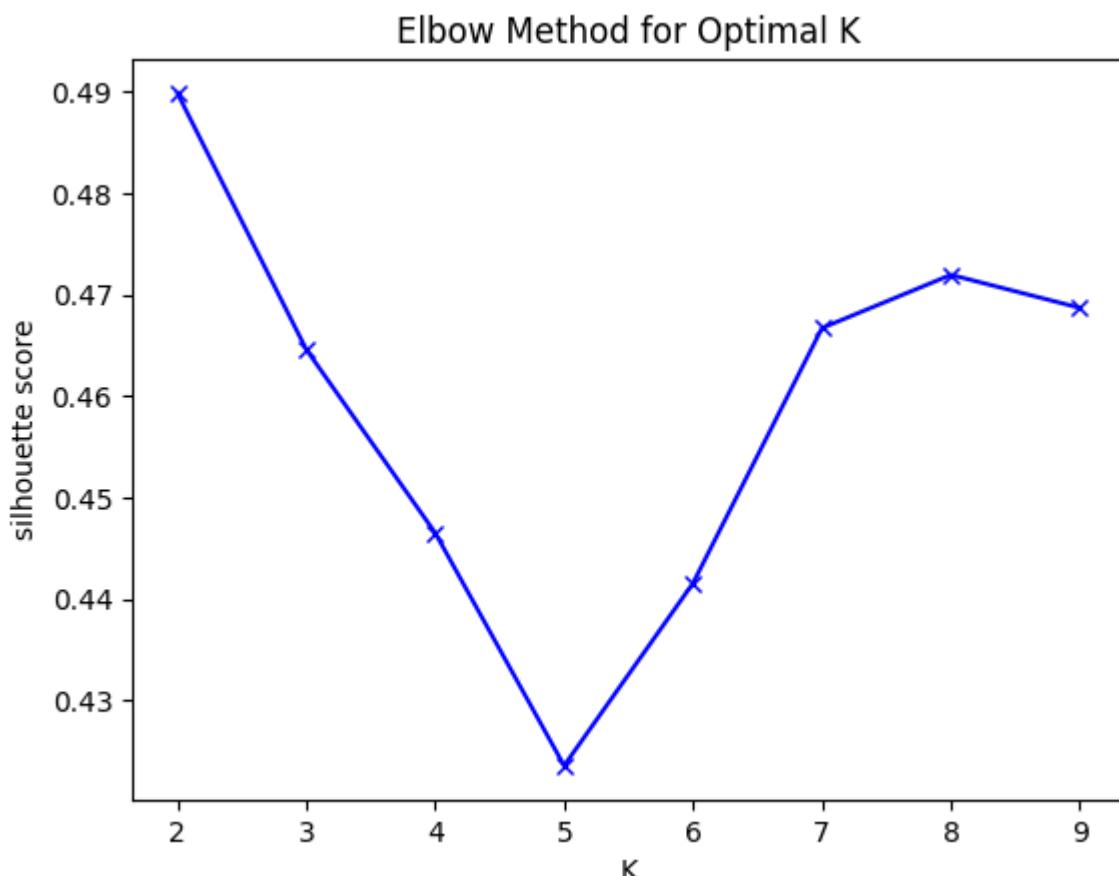
```
Number of clusters = 8
```

```
Silhouette score = 0.4719318976698181
```

```
Number of clusters = 9
```

```
Silhouette score = 0.4686778401053414
```

```
In [92]: score_df = pd.DataFrame(scores, columns=['k', 'scores'])
fig = plt.figure()
plt.title('Elbow Method for Optimal K')
plt.plot(score_df.k, score_df.scores, 'bx-')
plt.xlabel('K')
plt.ylabel('silhouette score')
plt.show()
```



```
In [93]: kmeans = KMeans(init='k-means++', n_clusters=2, n_init=10)
prediction = kmeans.fit_predict(df_pca)
```

```
In [94]: prediction
```

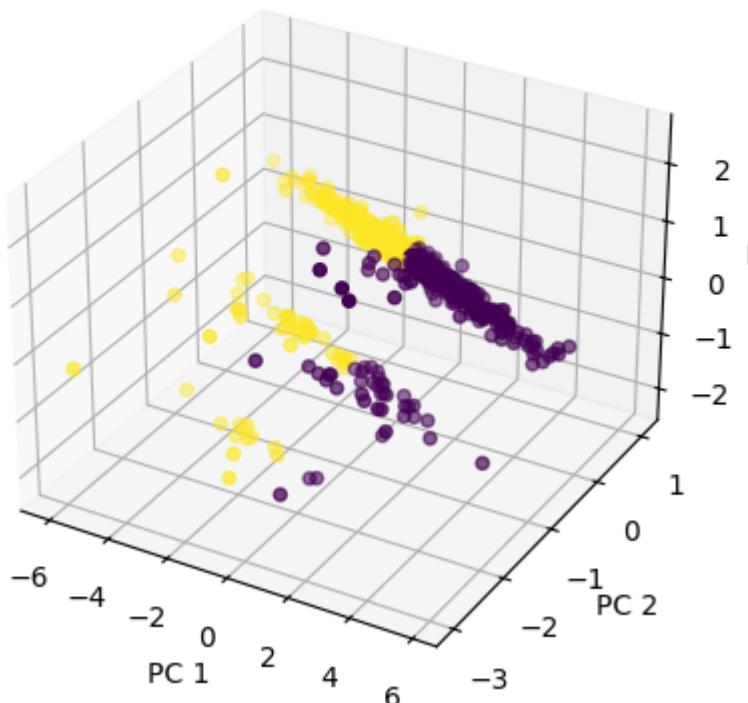
```
Out[94]: array([0, 0, 0, ..., 0, 0, 0], shape=(1042,), dtype=int32)
```

```
In [111... fig = plt.figure()
fig.suptitle("3 PC K-Means")
ax = fig.add_subplot(projection='3d')
ax.scatter(df_pca['PC_1'], df_pca['PC_2'], df_pca['PC_3'], c=prediction)

ax.set_xlabel('PC 1')
ax.set_ylabel('PC 2')
ax.set_zlabel('PC 3')
```

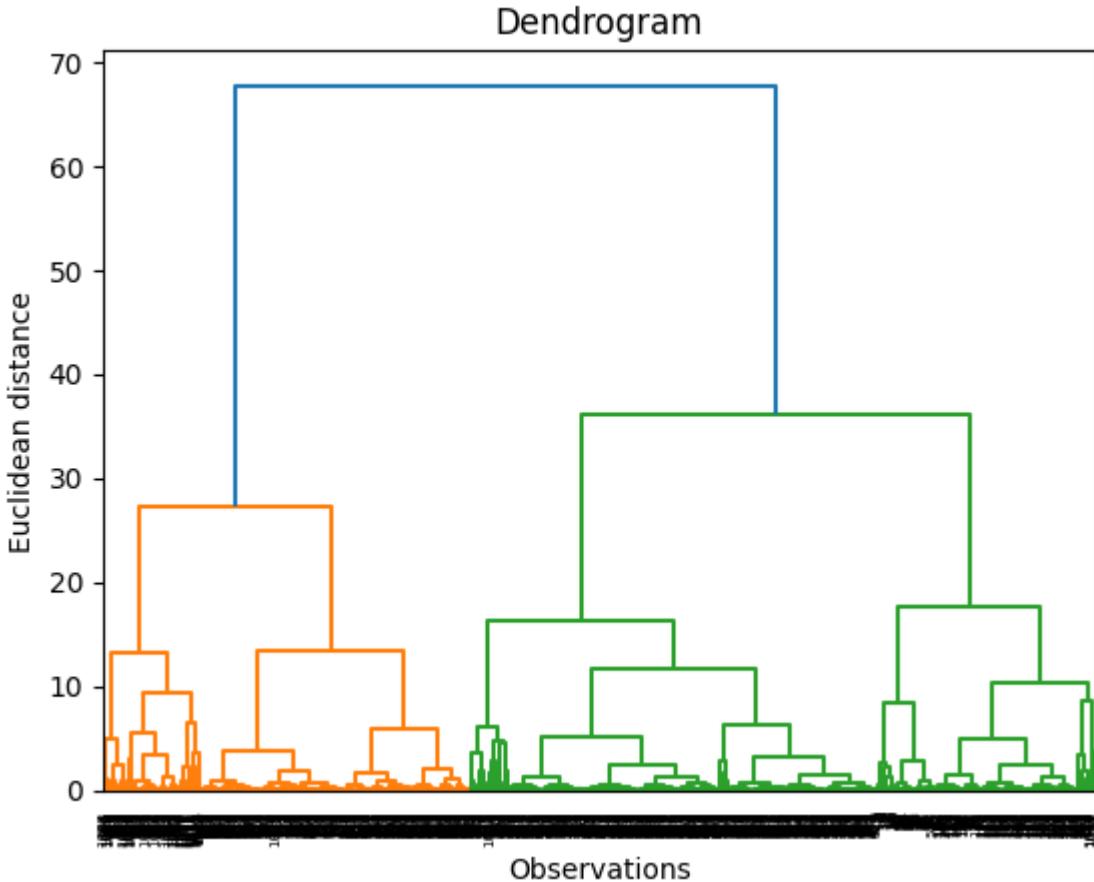
```
Out[111... Text(0.5, 0, 'PC 3')
```

3 PC K-Means



Well, the K-means method doesn't seem to agree with us. Lets try doing agglomerative clustering instead

```
In [96]: # Dendrogram for agglomerative clustering
plt.figure()
dendrogram = ch.dendrogram(ch.linkage(df_pca, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Observations')
plt.ylabel('Euclidean distance')
plt.show()
```



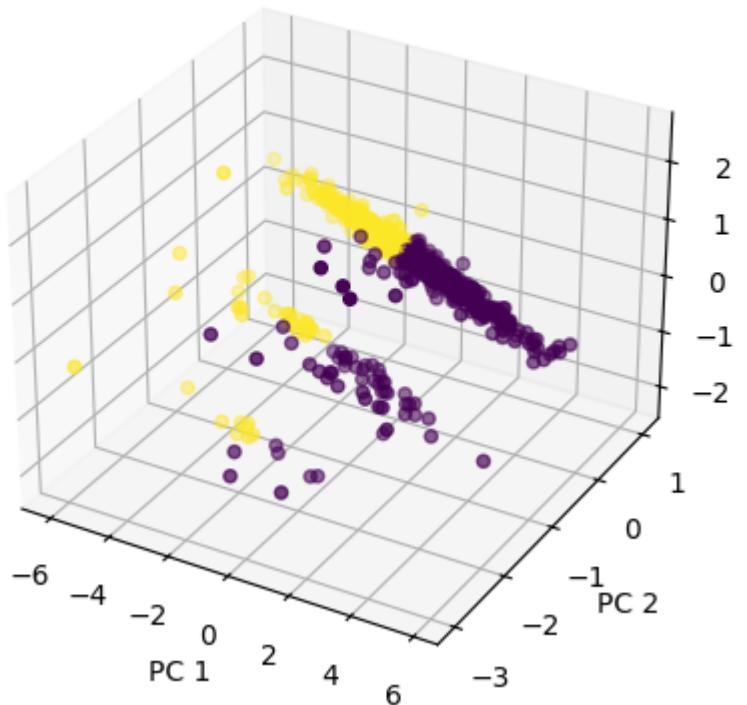
```
In [97]: model = AgglomerativeClustering(2, linkage = 'ward')
aggmodel_pred = model.fit_predict(df_pca)
```

```
In [98]: fig = plt.figure()
fig.suptitle("Agglomerative Clustering")
ax = fig.add_subplot(projection='3d')
ax.scatter(df_pca['PC_1'], df_pca['PC_2'], df_pca['PC_3'], c=aggmodel_pred)

ax.set_xlabel('PC 1')
ax.set_ylabel('PC 2')
ax.set_zlabel('PC 3')
```

```
Out[98]: Text(0.5, 0, 'PC 3')
```

Agglomerative Clustering



Well that doesn't seem to agree either. Lets try running a DBSCAN

```
In [99]: #DBSCAN for automatically determining amount of clusters
# Tried to play a little around with eps here
# for finding the optimal separation
dbscan = DBSCAN(eps=0.7, min_samples=10)
dbscan_pred = dbscan.fit_predict(df_pca)
dbscan_pred
```

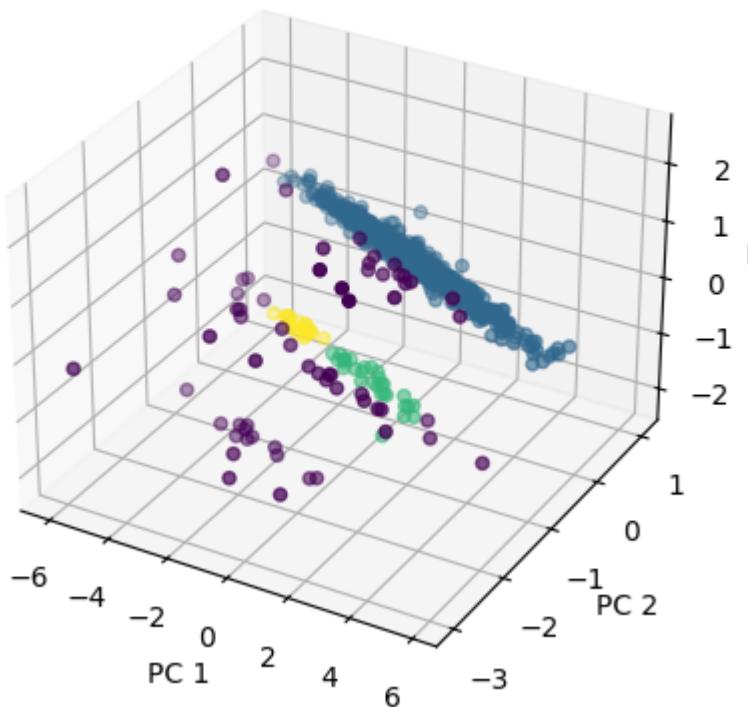
```
Out[99]: array([ 0,  0,  0, ..., -1, -1, -1], shape=(1042,))
```

```
In [100...]: fig = plt.figure()
fig.suptitle("DBSCAN")
ax = fig.add_subplot(projection='3d')
ax.scatter(df_pca['PC_1'], df_pca['PC_2'], df_pca['PC_3'], c=dbscan_pred)

ax.set_xlabel('PC 1')
ax.set_ylabel('PC 2')
ax.set_zlabel('PC 3')
```

```
Out[100...]: Text(0.5, 0, 'PC 3')
```

DBSCAN



The DBSCAN clustering seems to yield the best results so far. There's a clear large cluster, showing the correlation between PC1 and PC2. Two smaller clusters showing a concentration of data points in the same correlation between PC1 and PC2, but is offset on PC2 and PC3. Showing 2 potential subcategories of data along both axes of PC2 and PC3.

In this case we suspect the clusters mostly represent the bean species, since both PC2 and PC3 are mainly weighted by Uniformity and Sweetness. We found in earlier analysis, that the main difference between Arabica and Robusta beans seem to be in sweetness. This fits with a cluster split along PC2 and PC3 axes, along with the much smaller cluster size in terms of data points, since Robusta beans are less represented in the dataset.

In summary, what we can tell about the dataset is the following:

- The Main quality measures in the coffee seems to be the flavor parameters
- It takes a large amount of defects and quakers to actually worsen the overall quality
- Arabica beans tend to be sweeter than Robusta beans
- There is a very slight indication that higher quality coffee tends to come from Africa

Ensemble methods : Random forest

Preparation and extra cleaning

In [453...]

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import plot_tree
import shap
```

In [454...]

```
df = pd.read_csv("../data_cleaned/cleaned_dataset_no_zeros.csv")

df.head(10)
```

Out[454...]

	Species	Country.of.Origin	Variety	Processing.Method	Aroma	Flavor	Aftertaste
0	Arabica	Ethiopia	Caturra	Washed / Wet	8.67	8.83	8.67
1	Arabica	Ethiopia	Other	Washed / Wet	8.75	8.67	8.50
2	Arabica	Guatemala	Bourbon	Washed / Wet	8.42	8.50	8.42
3	Arabica	Ethiopia	Caturra	Natural / Dry	8.17	8.58	8.42
4	Arabica	Ethiopia	Other	Washed / Wet	8.25	8.50	8.25
5	Arabica	Ethiopia	Caturra	Washed / Wet	8.25	8.33	8.50
6	Arabica	Ethiopia	Caturra	Washed / Wet	8.67	8.67	8.58
7	Arabica	Ethiopia	Other	Natural / Dry	8.08	8.58	8.50
8	Arabica	Ethiopia	Caturra	Natural / Dry	8.17	8.67	8.25
9	Arabica	United States	Other	Washed / Wet	8.25	8.42	8.17

10 rows × 21 columns

In [455...]

```
# Extra cleaning in order to make a random forest regression model
# Species column should be numeric
df['Species'].unique()
df['Species_type'] = df['Species'].map({'Arabica': 0, 'Robusta': 1})
# Processing_method column should be numeric
df['Processing.Method'].unique()
df['Processing_method_type'] = df['Processing.Method'].map({'Washed / Wet': 0, 'Natural / Dry': 1, 'Honey': 2, 'Pulped Natural': 3, 'Ganxet': 4, 'Wet Process': 5})
# Extra columns that are not numeric should be dropped
columns_to_drop = ['Species', 'Country.of.Origin', 'Processing.Method', 'Variety']
df = df.drop(columns=columns_to_drop)
```

In [456...]

```
# Now we have a dataset that are all numeric that we can use for regression
df.head(5)
```

Out[456...]

	Aroma	Flavor	Aftertaste	Acidity	Body	Balance	Uniformity	Clean.Cup	Sweetnes
0	8.67	8.83	8.67	8.75	8.50	8.42	10.0	10.0	10.
1	8.75	8.67	8.50	8.58	8.42	8.42	10.0	10.0	10.
2	8.42	8.50	8.42	8.42	8.33	8.42	10.0	10.0	10.
3	8.17	8.58	8.42	8.42	8.50	8.25	10.0	10.0	10.
4	8.25	8.50	8.25	8.50	8.42	8.33	10.0	10.0	10.

In [457...]

```
# df_cleaned_random_forrest is what we are gonna use for the random forest regressor
df_cleaned_random_forrest = df
```

Random forest regression

We have decided to use the random forest regression model to predict coffee quality based on selected features. The reason we chose the random forest model was that first of all we wanted to predict coffee quality and our data is not time depending, so a traditional linear model might not be as effective. Random forest is known for its high accuracy and robustness against overfitting, as it takes the predictions of multiple decision trees to produce a more reliable output.

Source: <https://medium.com/@byanalytixlabs/random-forest-regression-how-it-helps-in-predictive-analytics-01c31897c1d4>

Setting target variable (y) to be the quality_score column and x is the rest of the columns as input variables

In [458...]

```
X = df_cleaned_random_forrest.drop(columns=["Total.Cup.Points"])
y = df_cleaned_random_forrest["Total.Cup.Points"]
```

Using the train_test_split from sklearn to split into training and test sets 80 % is training data and 30 % is test data. The training data is used to build the model and the test data is used to evaluate if the model performs well.

In [459...]

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Creates a random with a 100 trees and max depth of 15.

In [460...]

```
model = RandomForestRegressor(n_estimators=100, max_depth=15, random_state=42)
model.fit(X_train, y_train)
```

Out[460...]

```
▼      RandomForestRegressor
      i ? 
RandomForestRegressor(max_depth=15, random_state=42)
```

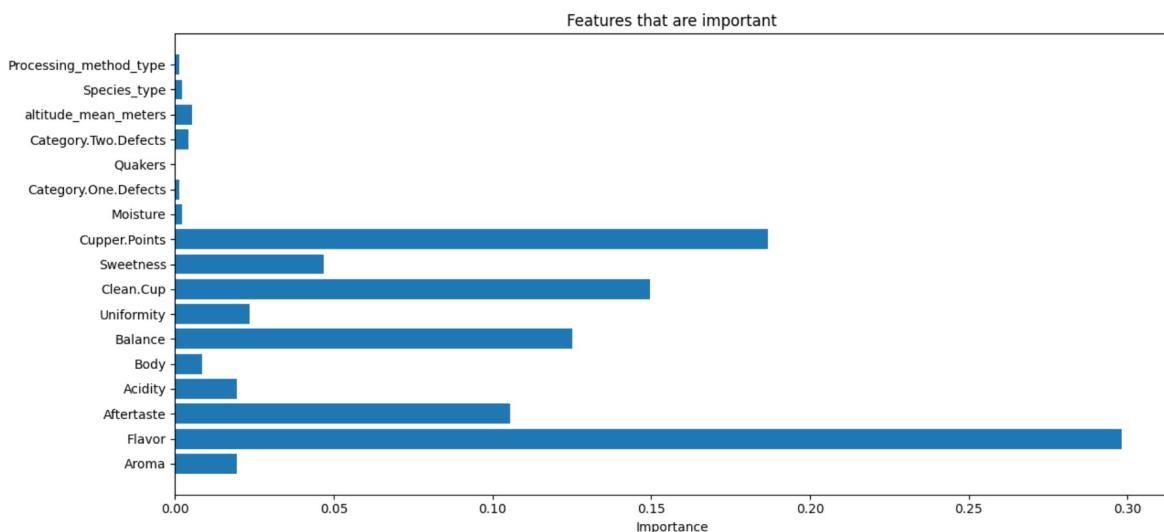
Predictions for the target variable (y) Total.Cup.Points column. It uses the test data on the model to predict Total.Cup.Points score.

```
In [461...]: y_pred = model.predict(X_test)
```

Plot that shows which features are most important for predicting the outcome of coffee_quality

```
In [462...]: importances = model.feature_importances_
feature_names = X.columns

plt.figure(figsize=(13, 6))
plt.barh(feature_names, importances)
plt.xlabel("Importance")
plt.title("Features that are important")
plt.tight_layout()
plt.show()
```



In the plot above we see a feature importance plot that shows us how important our features in the dataset is when it comes to reducing errors in measurements like R2 score and MSE. We see that features like Copper.Points, Flavor, etc are very important features for our model and features like species_type, processing_method_type is less important. It makes a lot of sense since features like Copper.Points, Flavor, Balance, etc all are features that has to do with the quality of the coffee and the taste, so it should have a great impact on the Total.Cup.Point variable.

We can also use the shap package to understand more about our features in the dataset and which are important for predicting the quality of the coffee. We can use shap to understand the individual predictions.

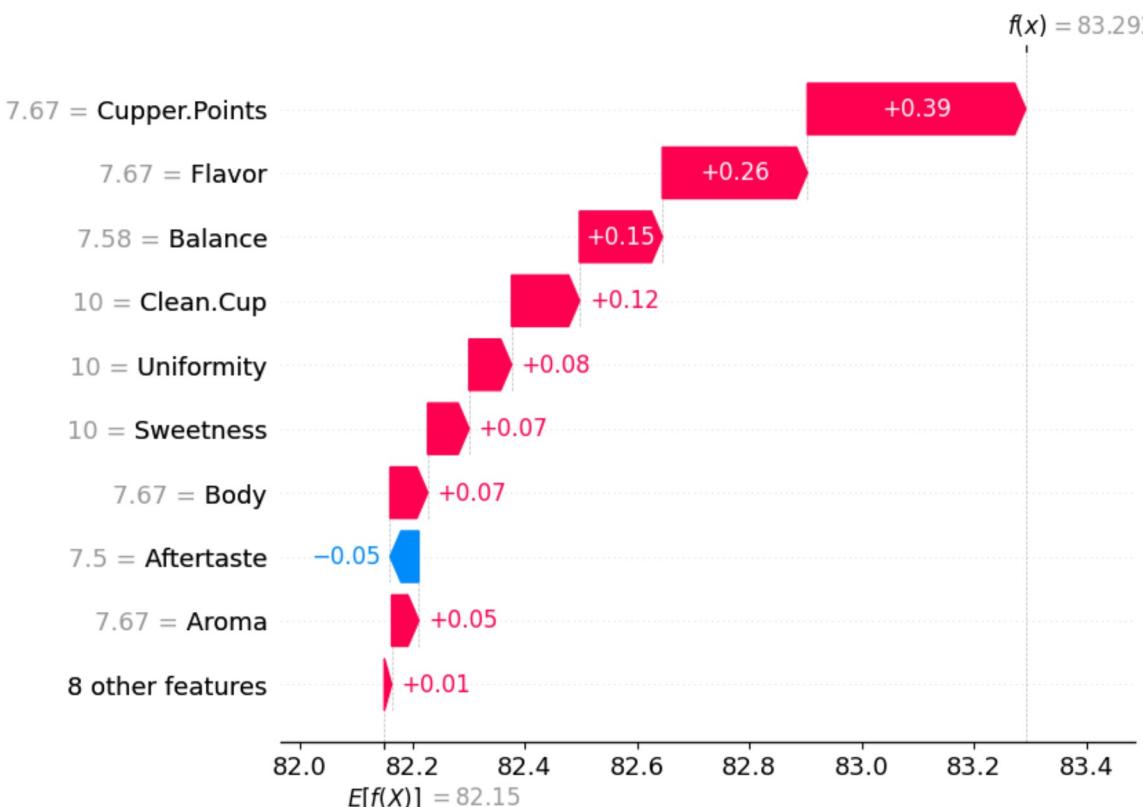
```
In [463...]: shap.initjs()

explainer = shap.TreeExplainer(model)

shap_values = explainer(X_test)
```



```
In [464...]: shap.plots.waterfall(shap_values[0])
```

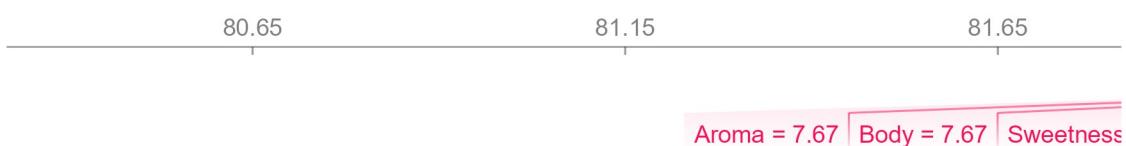


$E[f(x)]$ is the average predicted coffee quality across all samples. $f(x)$ is the predicted quality for this sample.

As we can see in the plot above Copper.Points, flavor and balance are very important for this specific sample as it increases the prediction quite much. The only feature in this sample that decreases the prediction is Aftertaste and it only decreases the prediction with -0.05.

```
In [465...]: shap.plots.force(shap_values[0])
```

```
Out[465...]
```



We can also visualize the SHAP values in a force plot as shown above but it can be a bit difficult to see. What we can see is that the baseline, which is the average value of coffee quality predicted without any features involved, is about 82.15. We see that most of the features increase the quality of the coffee from the baseline with only aftertaste decreasing the prediction.

Performance metrics

We can use different performance metrics to evaluate our model. For regression we typically use mean squared error (MSE) and R2 score. The MSE calculates the average squared difference between the predicted y (y_{pred}) and the actual value of y . So it

essentially tells us how far away our predicted value of y is away from the real value of y , in average. The smaller the value in y is, the smaller the difference is.

The r^2 score calculates how much of the variance in the data that can be explained. A score close to 1 in this case is good and closer to 0 means that the variance cannot be explained.

source: <https://medium.com/@wl8380/title-building-a-linear-regression-model-with-mse-and-r-squared-in-python-9df41472cf04>

```
In [466...]: # Performance metrics on test set
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R² Score: {r2:.2f}")
```

Mean Squared Error: 0.19

R² Score: 0.97

Our mean squared error score is 0.19 and the R² score is 0.97 which looks very good.

The performance metrics are so good that it would be a good idea to further investigate if we maybe have overfitted the model or there could be any other issues, just to evaluate that our model works well.

We will also calculate the MSE and R² for the training set. It is not usually done, but we have to make sure they are performing similar and the model doesn't just perform well on the test data.

```
In [467...]: # Performance metrics on train set
```

```
y_train_pred = model.predict(X_train)

mse = mean_squared_error(y_train, y_train_pred)
r2 = r2_score(y_train, y_train_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R² Score: {r2:.2f}")
```

Mean Squared Error: 0.11

R² Score: 0.98

The model is also performing very well on the training data with an MSE of 0.11 and R² score of 0.98, which is very similar to our test data. So it does indicate that the model works well on new data as well.

Plots

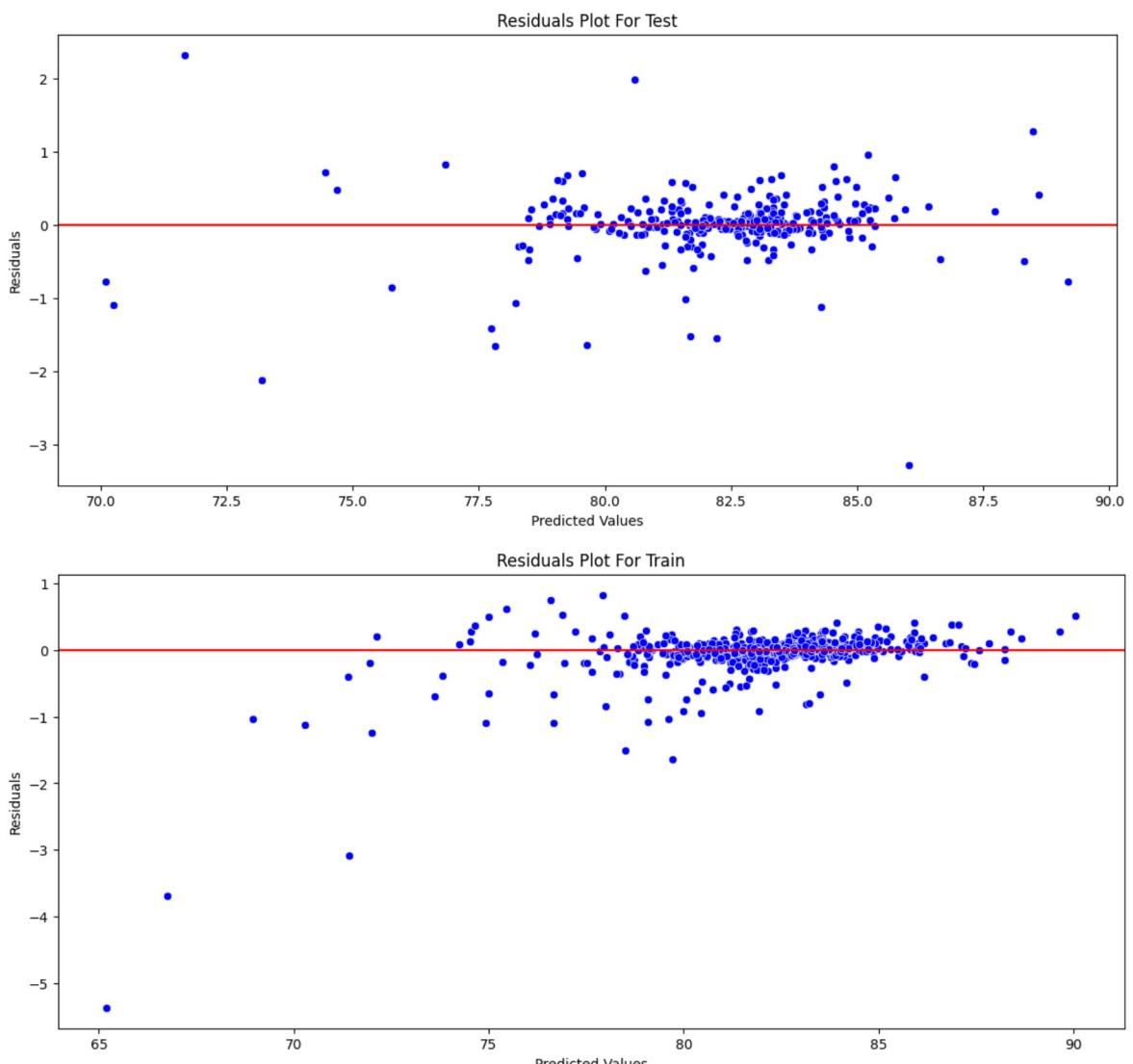
```
In [468...]: # Residual plots for test set and train set
```

```
residuals_test = y_test - y_pred

plt.figure(figsize=(14, 6))
```

```
sns.scatterplot(x=y_pred, y=residuals_test, color='blue')
plt.axhline(0, color='red')
plt.title('Residuals Plot For Test')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()

residuals_train = y_train - y_train_pred
plt.figure(figsize=(14, 6))
sns.scatterplot(x = y_train_pred, y = residuals_train, color='blue')
plt.axhline(0, color='red')
plt.title('Residuals Plot For Train')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```



In the above residual plots we do see that the points looks scattered around and there dosent seem to be any patterns. There is however a few points that are far away from 0 which maybe could indicate some outliers.

```
In [469]: # Actual vs predicted plot for both test and train
```

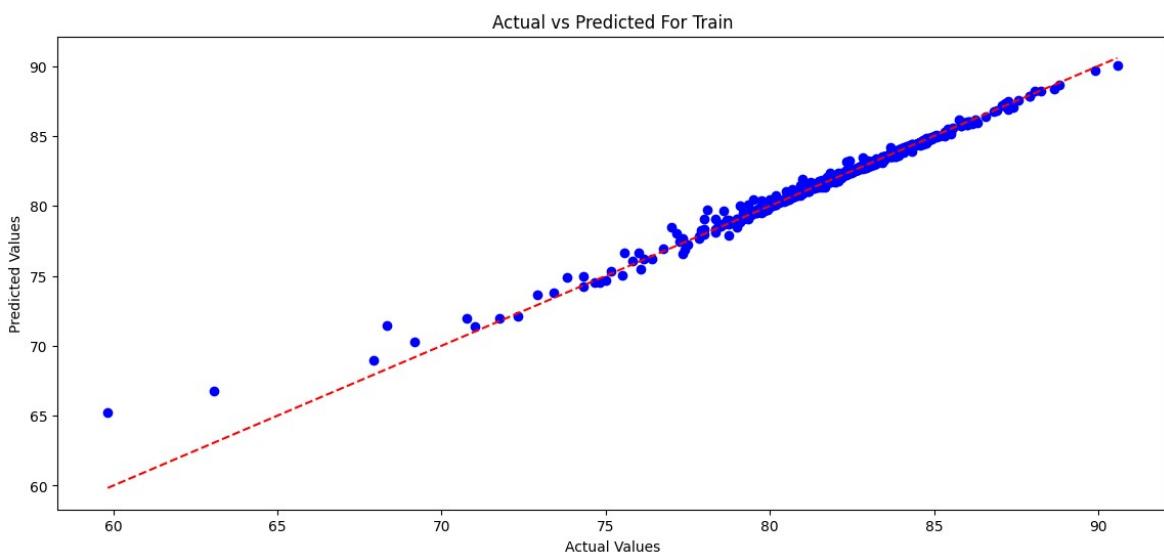
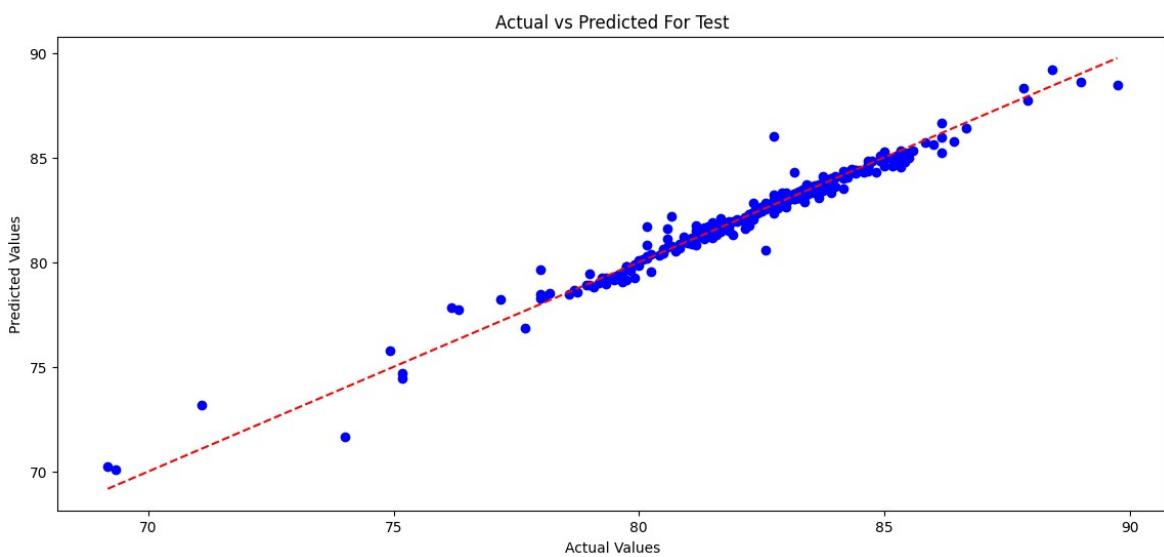
```
plt.figure(figsize=(14, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
```

```

plt.title('Actual vs Predicted For Test')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()

plt.figure(figsize=(14, 6))
plt.scatter(y_train, y_train_pred, color='blue')
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], 'r--')
plt.title('Actual vs Predicted For Train')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()

```



The predicted value is quite close in most cases, with the exception of a few outliers, to the actual value, which is good, because it tells us our model predicts well. It is however noticeable that it seems, from the actual vs predicted plots above, that the model predicts higher values better than lower. We have more scattered points between actual values of 70 - 77. It does indicate that the model might not perform as well when it comes to coffee with less quality.

Model evaluation

Our model seems to perform very well based on our metrics scores and the fact that the

model also works very well on the training set data. With an R2 score of 0.97 and MSE of 0.11 it shows us that it is a very good model.

The table below show how well the model performs when different metrics are changed.

Test Size	n_estimators	max_depth	R ² Score	MSE
0.2	50	10	0.97	0.23
0.2	100	10	0.96	0.24
0.2	120	10	0.96	0.24
0.2	50	15	0.97	0.22
0.2	100	15	0.97	0.23
0.2	120	15	0.96	0.24
0.3	50	10	0.96	0.21
0.3	100	10	0.97	0.20
0.3	120	10	0.97	0.20
0.3	50	15	0.97	0.20
0.3	100	15	0.97	0.19
0.3	120	15	0.97	0.19

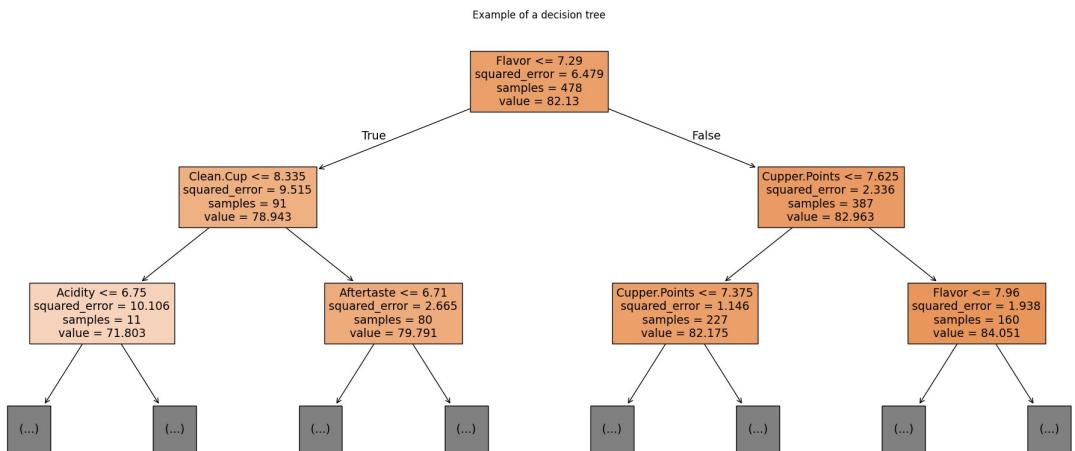
We do see overall that the R2 score is very good and close to 1. However we see that the MSE is a bit high maybe. MSE should be as close to 0 as possible and while our final model the MSE was 0.19 which is not low, but since the R2 score was so high, there could maybe be a potential issue, like some outliers we did not detect in the cleaning process. 0.19 is still closer to 0 as it should be but since the R2 score is 0.97 it is something to be aware of.

In [470...]:

```
estimator = model.estimators_[6]

plt.figure(figsize=(25, 10))
plot_tree(estimator,
          feature_names=X_train.columns,
          filled=True,
          max_depth=2)

plt.title("Example of a decision tree")
plt.show()
```



Above is just an example of how decisions are made and how it looks with a depth of 2 for better visibility.