Apple Quality Dataset

Presentation of Dataset Analysis

„Apples and Oranges … (Or Apples?) “

Analysis of the Dataset including Correlations, Regressions and Clustering

Assignment 2

CA259 - Data Analytics for Marketing Applications

Linus Lehr (22108475)

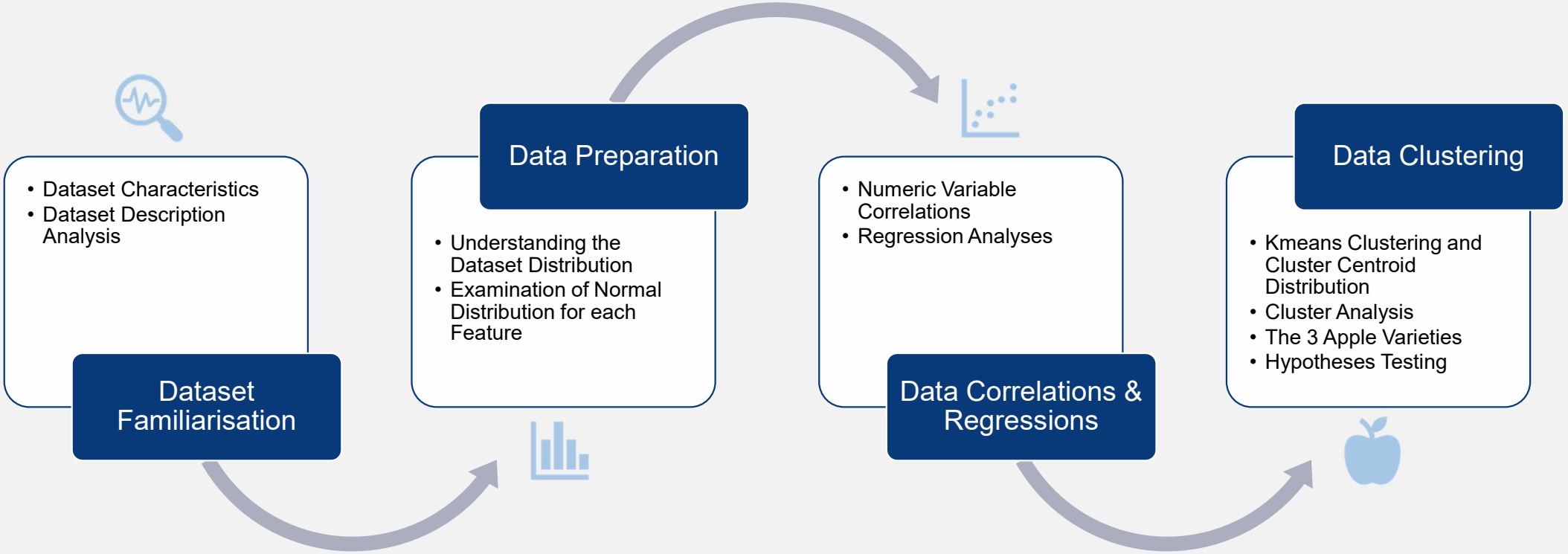
Caroline Ziegler (22108564)

Date: 20.02.2024

Place: Dublin, Ireland

The dataset “Apple Quality” has been examined in four distinct stages starting with dataset familiarisation until data clustering in the last phase

Table of Contents



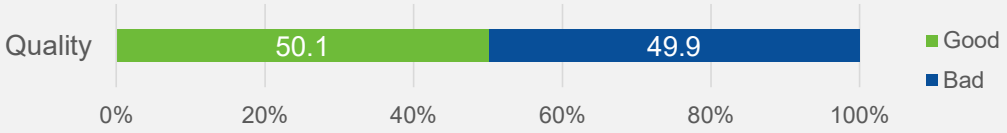
The Apple Quality Dataset is a transformed dataset excluding raw data where the transformation scale is unknown

Dataset Introduction

Dataset Description of Numeric Columns

	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity
count	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000
mean	-0.503015	-0.989547	-0.470479	0.985478	0.512118	0.498277	0.076877
std	1.928059	1.602507	1.943441	1.402757	1.930286	1.874427	2.110270
min	-7.151703	-7.149848	-6.894485	-6.055058	-5.961897	-5.864599	-7.010538
25%	-1.816765	-2.011770	-1.738425	0.062764	-0.801286	-0.771677	-1.377424
50%	-0.513703	-0.984736	-0.504758	0.998249	0.534219	0.503445	0.022609
75%	0.805526	0.030976	0.801922	1.894234	1.835976	1.766212	1.510493
max	6.406367	5.790714	6.374916	7.619852	7.364403	7.237837	7.404736

Dataset Description of Non-Numeric Column



Dataset Characteristics

- Consists of initially **4001 rows** and 9 columns (A_id was deleted as being identical to DataFrame index, the last row was deleted as it only contained the name of the author)
- Does **not contain any raw data** in the numeric columns - the data points in the data set have been transformed and therefore comprises no missing values[1]
- Contains data on the quality of apples described by 7 numeric and 1 non-numeric column
  - **Numeric columns** include size, weight, sweetness, crunchiness, juiciness, ripeness and acidity
  - **Non-numeric column** describes the quality (“good” vs. “bad”)

Dataset Description Analysis

- To prepare, clean and analyse the dataset, Python was used as analytics tool
- To understand the **numeric columns**, the *describe()* method was applied
  - The maximum values are between 5.8 and 7.6
  - The minimum values are between -5.9 and -7.2
  - The **means** are ranging between approximately -1 and 1
  - The **standard deviations** are laying in the range of 1.4 and 2.1
  - ➢ The normalisation of the raw data was **not done with a z-score transformation** since otherwise the mean would have been expected to be 0 and the standard deviation to be 1 and the min/max values should be closer to -/+ 3
- To examine the **non-numeric column**, the distribution was calculated showing an **even distribution** between good and bad apples in the dataset

Note: (1) The scale used for the transformation of the raw data is not known (there is no information about it either via Kaggle or in comments)

The data is overall normally distributed but no further data transformations were performed

Data Cleaning and Normal Distribution

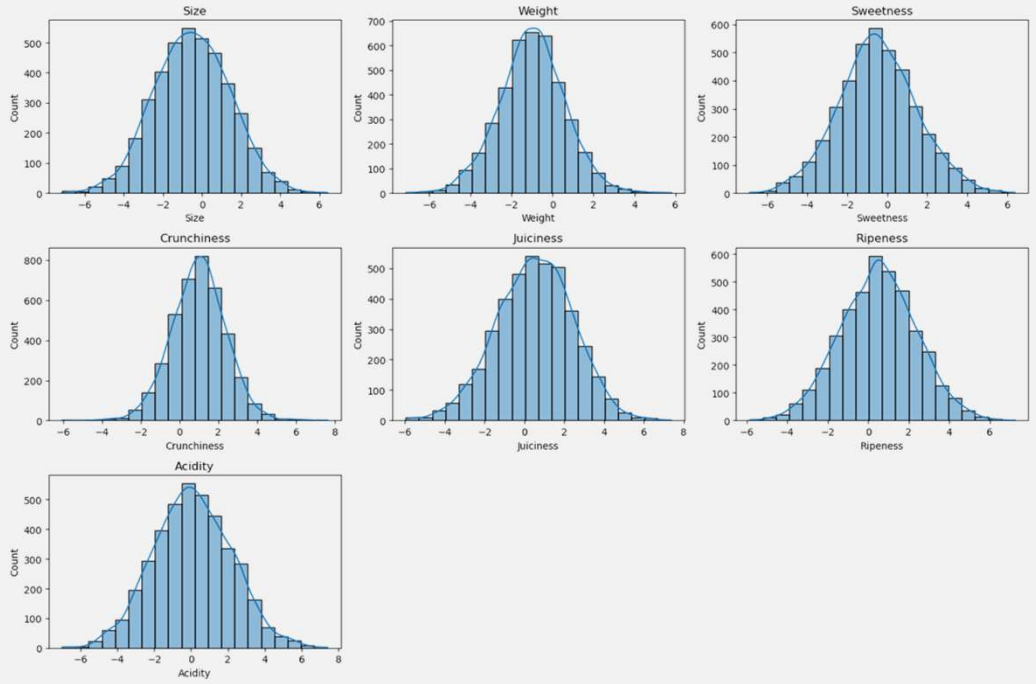
Understanding the Dataset Distribution

- To analyse whether the dataset is normally distributed, each numeric column was plotted as a combined graph of a histogram and line graph
- To further examine normal distribution, the skewness and kurtosis of each numeric column was calculated
- **Size:** widely distributed with a negative mean (skewness: 0.0024; kurtosis: -0.0833)
- **Weight:** narrowly distributed with a negative mean (skewness: 0.0031; kurtosis: 0.3590)
- **Sweetness:** more widely distributed with a negative mean (skewness: 0.0838; kurtosis: 0.01447)
- **Crunchiness:** the narrowest distributed column with a positive mean (skewness: 0.0002; kurtosis: 0.7220)
- **Juiciness:** widely distributed with a positive mean (skewness: -0.1134; kurtosis: 0.0287)
- **Ripeness**: more widely distributed with a positive mean (skewness: -0.0087; kurtosis: -0.0718)
- **Acidity:** more widely distributed with a positive mean (skewness: 0.0557; kurtosis: -0.0934)

➢ Overall, the data is normally distributed; however, as an unknown transformation was performed, another z-score transformation was rejected[1]

- For the categorical “quality” column, label encoding was used to transform bad to 0 and good to 1 to be able to use the data for regression analysis later

Examination of Normal Distribution for each Feature

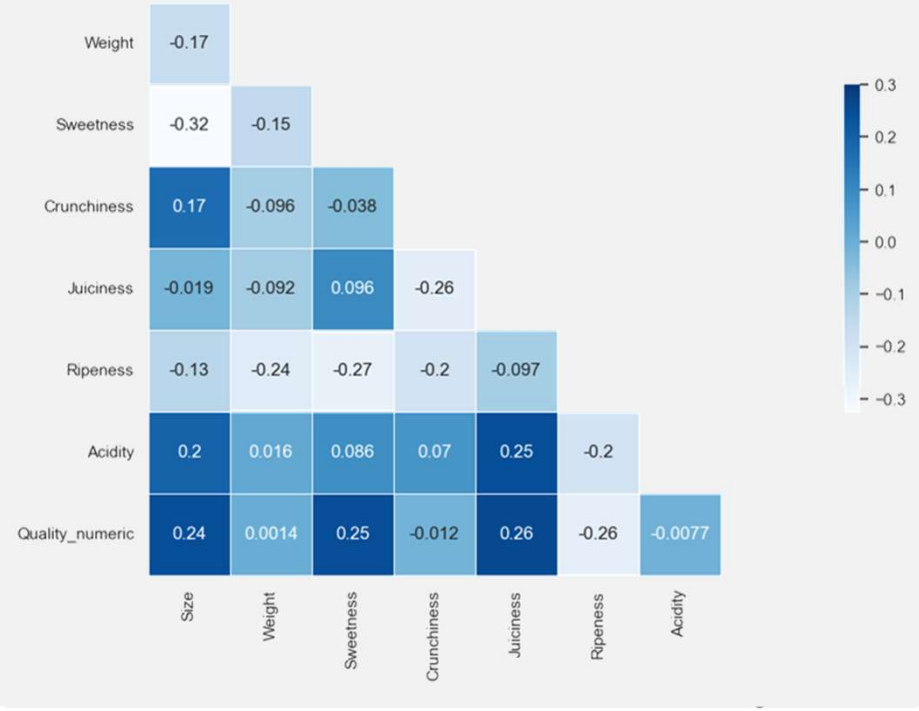


Note: (1) Another z-score transformation does not support the understanding of the raw data and a transformation on top of an unknown transformation can be seen as potentially problematic

Half of the correlations are categorized as medium or higher, enhancing comprehension of the attributes related to apple quality and their correlations

Numeric Variable Correlations

Correlation Matrix



Understanding Correlations

- Correlations in among variable pairs range from **-0.32 to 0.26**
- Positive as well as negative correlations are observed with **negative ones predominating** - 60.7% of all correlations are negative
- In-depth analysis of correlations with an **absolute value higher than 0.2** was conducted[1]
  - **Size:** the sweeter an apple, the smaller it is
  - **Weight:** the riper an apple, the more lightweight it is
  - **Sweetness:** the bigger and riper an apple is, the less sweet it is
  - **Crunchiness:** the juicier and riper an apple is, the less crunchy it is
  - **Juiciness:** the higher the acidity is, the juicier the apple is AND the crunchier an apple is, the less juicy it is
  - **Ripeness:** the heavier, the sweeter, the crunchier, the more acidic, the less ripe the apple is
  - **Acidity:** the juicier an apple, the higher the acidity, BUT the riper an apple is, the lower the acidity is
- The correlations enable a better understanding of how the apples' features relate to each other and what regressions to perform later
- The **encoded, numeric quality column** was not considered in the correlation analysis as it is categorical data for which correlations should not be calculated and cannot be interpreted

Note: (1) For each of the correlation pairs, the correlation coefficient was calculated calling the *.corr()* method on the DataFrame

Logistic regression offers predictive capability for determining apple quality - A good apple is big, sweet and juicy while not having a high degree of acidity

Regression Analyses

Dependent Variable	Regression Type	Independent Variable 1	Independent Variable 2	Independent Variable 3	Independent Variable 4	Independent Variable 5	Independent Variable 6	Independent Variable 7	Accuracy Score
Sweetness	Linear	Size	Ripeness						21.8%
Crunchiness	Linear	Juiciness	Ripeness						12.1%
Ripeness	Linear	Weight	Sweetness	Crunchiness	Acidity				25.4%
Quality Numeric	Logistic	Size	Sweetness	Juiciness	Ripeness	Weight	Acidity	Crunchiness	**72.8%**

Conclusion

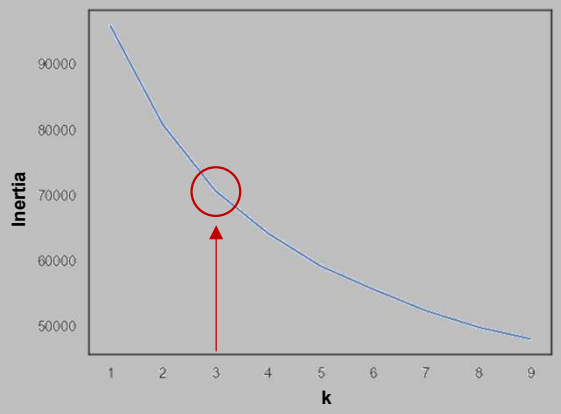
- The accuracy score for the linear regressions is the R^2 value, while the accuracy score for the logistic regression returns the mean accuracy; for both, a value of 1 would be a 100% prediction rate
- The apple natural features (all except the quality) cannot be used very well to predict each other as the maximum accuracy score is 25.4%
- However, the natural features can be used quite well to predict the apple quality (“good”=1 and “bad”=0) supported by an accuracy score of 72.8% - the highest coefficient with 0.66 is for size followed by 0.59 is sweetness, while the ripeness (-0.13) and acidity (-0.27) have a negative impact on the apple quality

Note: A linear regression analysis was performed with Sklearn for each feature for which there are at least two independent with a correlation of 20% or higher (absolute value)

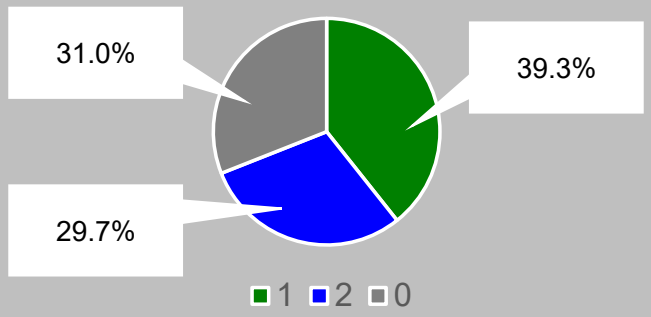
Three distinct apple varieties with different features were identified using Kmeans clustering

Kmeans Clustering and Cluster Centroid Distribution

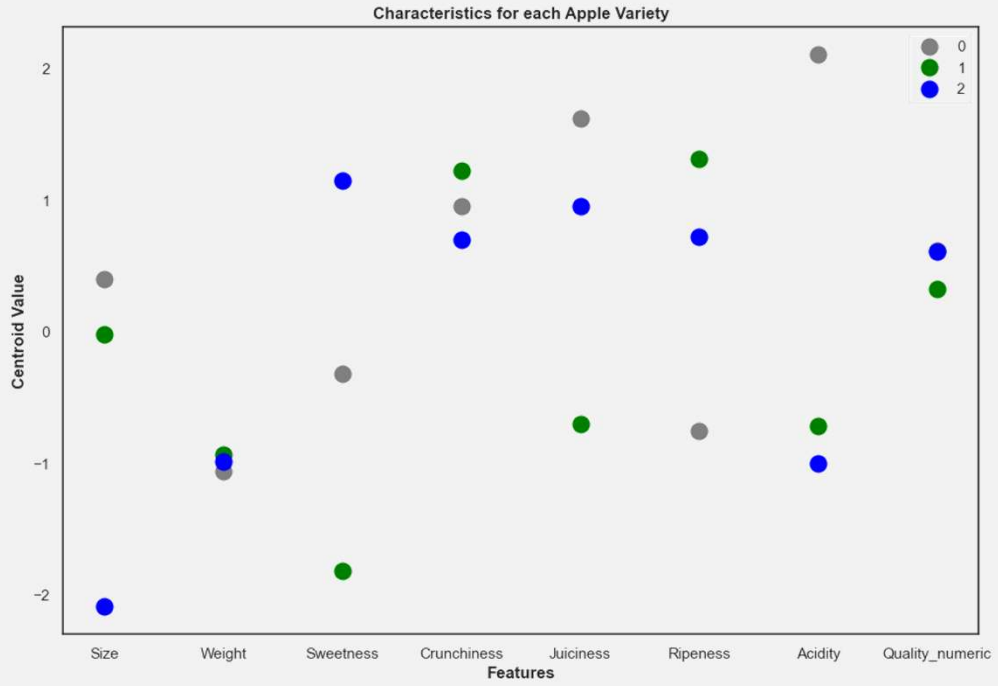
Elbow Method[1]



Label Distribution



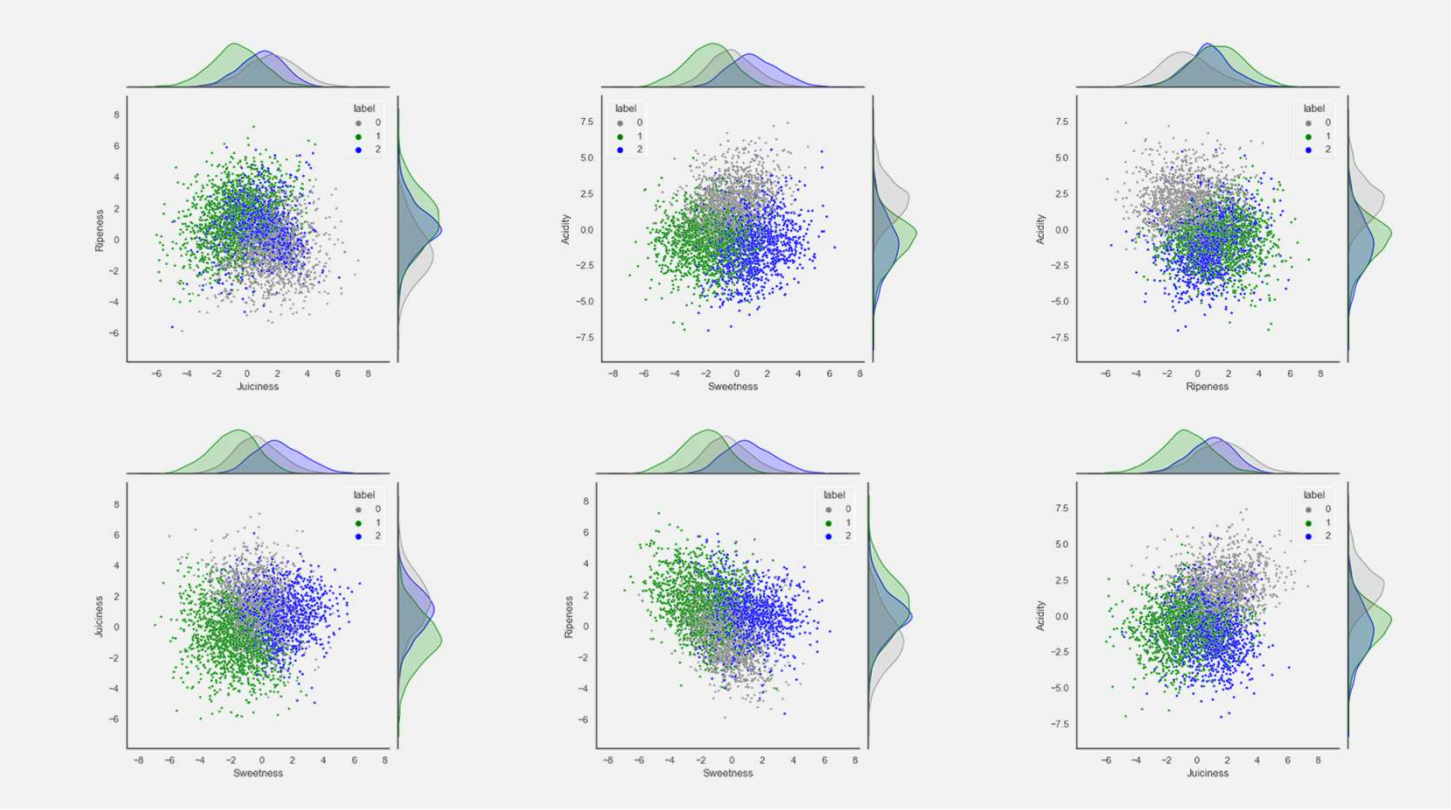
Cluster Centroid per Feature[2]



Note: (1) Using the elbow method is a standard approach to get the number of clusters, here k=3; (2) Centroids were used as they are the most representative datapoint for each cluster

The differences between the three clusters are more pronounced in some features than in others

Cluster Analysis



Note: The plots were crafted using the seaborn library, employing its versatile jointplot function

Understanding Apple Varieties

- The four features with the **highest range** between the **three centroids** are plotted in pairs against each other
  - Ripeness & Juiciness
  - Acidity & Sweetness
  - Acidity & Ripeness
  - Juiciness & Sweetness
  - Ripeness & Sweetness
  - Acidity & Juiciness
- The aim is to better understand **how similar and different the varieties are** in these features
- At the opposite of each axis, a density plot was added to get a better aggregate overview regarding the distribution in this feature
- In certain plots, such as Acidity vs. Sweetness, the disparities between clusters are more pronounced, while in others, like Juiciness vs. Ripeness, the clusters exhibit more overlap
- Jointly with the cluster centroid analysis these plots help to better understand and characterize the three different apple varieties…

Apples and Oranges … (Or Apples?) – The three identified apple varieties have distinct characteristics allowing for creative variety profiling

The 3 Apple Varieties

Apple 0

Core Features:

Size
- Very Big

Juiciness
- Very Juicy

Acidity
- Very High in Acidity

With its remarkable size, this variety promises a satisfying bite and generous portion. Renowned for its succulence, Apple 0 is a perfect candidate for the creation of apple juice or cider. Additionally, its notable acidity levels contribute to its tangy and refreshing taste profile, making it a distinct choice for those who appreciate a zesty flavor experience.

Apple 1

Core Features:

Size
- Big

Sweetness
- Low in Sweetness

Juiciness
- Not very juicy

The Apple 1 lacks robust flavor, with a taste profile that fails to excite the palate. Even though it's a big apple, expectations cannot be met. Additionally, its sourness registers as mediocre, lacking the bold tanginess that characterizes premium varieties. Furthermore, its lack of juiciness and sweetness diminishes its appeal, leaving a dry and unsatisfying texture.

Apple 2

Core Features:

Size
- Very small

Sweetness
- Very Sweet

Acidity
- Very low in acidity

Apple 2 stands out with its unique combination of core features, making it a delightful option. Its size is notably petite, which adds to its charm and makes it a convenient snack option. In terms of sweetness, it is exceptional, offering a delightful burst of natural sugars with every bite. Furthermore, its acidity levels are minimal, which contributes to its smooth and mellow flavor profile.

Note: The descriptive text of each apple variety presents a possible, humorous way of promotion

The differences between the apple varieties do not occur by chance but are systematic as proven by statistically significant independent two sample t-tests

Hypotheses Testing

Feature	Cluster Maximum	Cluster Minimum	H_0	P-Value	T-Statistics
Sweetness	Apple 2	Apple 1	Apple variety 2 is not sweeter than Apple variety 1.	**0.0**	50.54
Juiciness	Apple 0	Apple 1	Apple variety 0 is not juicer than Apple variety 1.	**$3.8 * 10^{-231}$**	35.76
Ripeness	Apple 1	Apple 0	Apple variety 1 is not riper than Apple variety 0.	**$1.8 * 10^{-191}$**	31.96
Acidity	Apple 0	Apple 2	Apple variety 0 is not more acidic than Apple variety 2.	**0.0**	46.53

Results

- The apple variety which has the **highest centroid value** for a specific feature was tested against the apple variety which has the **lowest centroid value** for the same specific feature
- The **features tested** are the same as in the clustering analysis and therefore the ones with the highest range between the three centroid values
- The hypotheses testing resulted in **4 statistically significant statements** to the significant level of 0.01:
  - Apple variety 2 is on average sweeter than apple variety 1
  - Apple variety 0 is on average juicier than apple variety 1
  - Apple variety 1 is on average riper than apple variety 0
  - Apple variety 0 is on average more acidic than apple variety 2

Note: The hypotheses testing was conducted with the stats module from the SciPy library

Conclusion

The quality of apples in the “Apple Quality” dataset can be predicted and three distinct apple varieties have been identified

Analysis Results

Data set contains data that has been transformed with an unknown transformation (no raw data)

No further transformations were conducted, even though data set is normally distributed

The major correlations were identified, and regression analysis conducted
With a logistic regression the quality of apples can be predicted with a high accuracy

With KMeans clustering three distinct apple varieties can be identified
The differences between the varieties in the features are systematic as proven by independent two-sample t tests

Note: Comprehensive details regarding the utilized code can be found in the appendix section

Thank you very much!

Appendix

Code used for the Dataset Familiarization Part

Dropping the last row

```
apples = apples.drop(4000)
apples
```

	A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality
0	0.0	-3.970049	-2.512336	5.346330	-1.012009	1.844900	0.329840	-0.491590483	good
1	1.0	-1.195217	-2.839257	3.664059	1.588232	0.853286	0.867530	-0.722809367	good
2	2.0	-0.292024	-1.351282	-1.738429	-0.342616	2.838636	-0.038033	2.621636473	bad
3	3.0	-0.657196	-2.271627	1.324874	-0.097875	3.637970	-3.413761	0.790723217	good
4	4.0	1.364217	-1.296612	-0.384658	-0.553006	3.030874	-1.303849	0.501984036	good
...	...	...	...	...	...	...	...	...	...
3995	3995.0	0.059386	-1.067408	-3.714549	0.473052	1.697986	2.244055	0.137784369	bad
3996	3996.0	-0.293118	1.949253	-0.204020	-0.640196	0.024523	-1.087900	1.854235285	good
3997	3997.0	-2.634515	-2.138247	-2.440461	0.657223	2.199709	4.763859	-1.334611391	bad
3998	3998.0	-4.008004	-1.779337	2.366397	-0.200329	2.161435	0.214488	-2.229719806	good
3999	3999.0	0.278540	-1.715505	0.121217	-1.154075	1.266677	-0.776571	1.599796456	good

4000 rows × 9 columns

Dropping column 'A_id'

```
apples.drop("A_id", axis=1, inplace = True)
apples
```

	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality
0	-3.970049	-2.512336	5.346330	-1.012009	1.844900	0.329840	-0.491590483	good
1	-1.195217	-2.839257	3.664059	1.588232	0.853286	0.867530	-0.722809367	good
2	-0.292024	-1.351282	-1.738429	-0.342616	2.838636	-0.038033	2.621636473	bad
3	-0.657196	-2.271627	1.324874	-0.097875	3.637970	-3.413761	0.790723217	good
4	1.364217	-1.296612	-0.384658	-0.553006	3.030874	-1.303849	0.501984036	good
...	...	...	...	...	...	...	...	...
3995	0.059386	-1.067408	-3.714549	0.473052	1.697986	2.244055	0.137784369	bad
3996	-0.293118	1.949253	-0.204020	-0.640196	0.024523	-1.087900	1.854235285	good
3997	-2.634515	-2.138247	-2.440461	0.657223	2.199709	4.763859	-1.334611391	bad
3998	-4.008004	-1.779337	2.366397	-0.200329	2.161435	0.214488	-2.229719806	good
3999	0.278540	-1.715505	0.121217	-1.154075	1.266677	-0.776571	1.599796456	good

4000 rows × 8 columns

Converting column 'Acidity' to float from string

```
type(apples.iloc[0,6])
```

```
str
```

```
apples["Acidity"] = apples["Acidity"].astype("float")
type(apples.iloc[0,6])
```

```
numpy.float64
```

```
apples.describe().style.background_gradient(axis=1, cmap='Blues')
```

	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity
count	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000
mean	-0.503015	-0.989547	-0.470479	0.985478	0.512118	0.498277	0.076877
std	1.928059	1.602507	1.943441	1.402757	1.930286	1.874427	2.110270
min	-7.151703	-7.149848	-6.894485	-6.055058	-5.961897	-5.864599	-7.010538
25%	-1.816765	-2.011770	-1.738425	0.062764	-0.801286	-0.771677	-1.377424
50%	-0.513703	-0.984736	-0.504758	0.998249	0.534219	0.503445	0.022609
75%	0.805526	0.030976	0.801922	1.894234	1.835976	1.766212	1.510493
max	6.406367	5.790714	6.374916	7.619852	7.364403	7.237837	7.404736

Generating the dataset description

```
apples.describe().style.background_gradient(axis=1, cmap='Blues')
```

	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness
count	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000
mean	-0.503015	-0.989547	-0.470479	0.985478	0.512118	0.498277
std	1.928059	1.602507	1.943441	1.402757	1.930286	1.874427
min	-7.151703	-7.149848	-6.894485	-6.055058	-5.961897	-5.864599
25%	-1.816765	-2.011770	-1.738425	0.062764	-0.801286	-0.771677
50%	-0.513703	-0.984736	-0.504758	0.998249	0.534219	0.503445
75%	0.805526	0.030976	0.801922	1.894234	1.835976	1.766212
max	6.406367	5.790714	6.374916	7.619852	7.364403	7.237837

Checking if there are any missing values

```
apples.isna().sum()
```

```
Size          0
Weight        0
Sweetness     0
Crunchiness   0
Juiciness     0
Ripeness      0
Acidity       0
Quality       0
dtype: int64
```

Generating a pie chart for the distribution of the 'Quality' variable

```
quality_counts = apples["Quality"].value_counts()

plt.pie(quality_counts, labels=quality_counts.index, autopct='%1.1f%%', textprops={'fontsize':10})
plt.title('Quality Distribution')
plt.show()
```

Code used for the Dataset Preparation Part

Code used to get grouped graphs to check for normal distribution

```
#gettig an overview over all histograms understanding normal distribution
numerical_columns = ['Size', 'Weight', 'Sweetness', 'Crunchiness', 'Juiciness', 'Ripeness', 'Acidity']

plt.figure(figsize=(15, 10))
sns.set_palette("tab10")


for i, column in enumerate(numerical_columns, 1):
    plt.subplot(3, 3, i)
    sns.histplot(data=apples, x=column, kde=True, bins=20)
    plt.title(column)

plt.tight_layout()
plt.show()
```

Code used to check for skewedness and kurtosis to determine normal distribution of data

```
print(apples["Acidity"].skew())
print(apples["Acidity"].kurtosis())
```

```
0.05578345335267586
-0.09345141613095986
```

Transforming ‘Quality’ into a binary, label encoded variable for further analysis

```
encoded_dict = {'good': 1, 'bad': 0}
apples["Quality_numeric"] = apples["Quality"].map(encoded_dict)
apples
```

	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality	Quality_numeric
0	-3.970049	-2.512336	5.346330	-1.012009	1.844900	0.329840	-0.491590	good	1
1	-1.195217	-2.839257	3.664059	1.588232	0.853286	0.867530	-0.722809	good	1
2	-0.292024	-1.351282	-1.738429	-0.342616	2.838636	-0.038033	2.621636	bad	0
3	-0.657196	-2.271627	1.324874	-0.097875	3.637970	-3.413761	0.790723	good	1
4	1.364217	-1.296612	-0.384658	-0.553006	3.030874	-1.303849	0.501984	good	1
...	...	...	...	...	...	...	...	...	...
3995	0.059386	-1.067408	-3.714549	0.473052	1.697986	2.244055	0.137784	bad	0
3996	-0.293118	1.949253	-0.204020	-0.640196	0.024523	-1.087900	1.854235	good	1
3997	-2.634515	-2.138247	-2.440461	0.657223	2.199709	4.763859	-1.334611	bad	0
3998	-4.008004	-1.779337	2.366397	-0.200329	2.161435	0.214488	-2.229720	good	1
3999	0.278540	-1.715505	0.121217	-1.154075	1.266677	-0.776571	1.599796	good	1

Code used for the Data Correlations & Regressions Part

Generating a correlation table

```
apples_corr = apples.corr()
apples_corr
```

```
C:\Users\CZ\AppData\Local\Temp\ipykernel_19336\3231808911.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  apples_corr = apples.corr()
```

	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality_numeric
Size	1.000000	-0.170702	-0.324680	0.169868	-0.018892	-0.134773	0.196218	0.244007
Weight	-0.170702	1.000000	-0.154246	-0.095882	-0.092263	-0.243824	0.016414	0.001421
Sweetness	-0.324680	-0.154246	1.000000	-0.037552	0.095882	-0.273800	0.085999	0.250998
Crunchiness	0.169868	-0.095882	-0.037552	1.000000	-0.259607	-0.201982	0.069943	-0.012376
Juiciness	-0.018892	-0.092263	0.095882	-0.259607	1.000000	-0.097144	0.248714	0.260223
Ripeness	-0.134773	-0.243824	-0.273800	-0.201982	-0.097144	1.000000	-0.202669	-0.264315
Acidity	0.196218	0.016414	0.085999	0.069943	0.248714	-0.202669	1.000000	-0.007697
Quality_numeric	0.244007	0.001421	0.250998	-0.012376	0.260223	-0.264315	-0.007697	1.000000

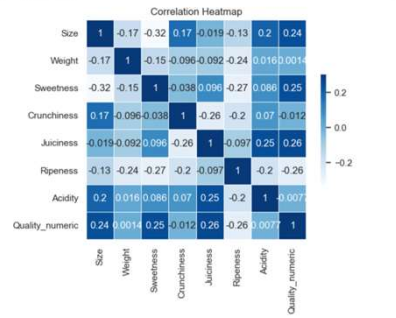
Filtering for correlations with a coefficient >= | 0.2 |

```
#Analysing the correlations pairs with a correlation higher or lower and equal to 0.2 and -0.2
tpllst = []
for i in range(0,8):
    for j in range(0,8):
        if apples_corr.iloc[i,j]>=0.2 and apples_corr.iloc[i,j]<1:
            tpllst.append((apples_corr.index[i], apples_corr.index[j], apples_corr.iloc[i,j]))
        elif apples_corr.iloc[i,j]<= -0.2 and apples_corr.iloc[i,j]>-1:
            tpllst.append((apples_corr.index[i], apples_corr.index[j], apples_corr.iloc[i,j]))

dltlst = [] #new list to save only those items where the quality_numeric is not at the second place underlining that this
for i in range(len(tpllst)):
    if tpllst[i][1] != "Quality_numeric":
        dltlst.append(tpllst[i])
dltlst
```

```
[('Size', 'Sweetness', -0.32468024956214525),
 ('Weight', 'Ripeness', -0.24382426555524034),
 ('Sweetness', 'Size', -0.32468024956214525),
 ('Sweetness', 'Ripeness', -0.27380006756070097),
 ('Crunchiness', 'Juiciness', -0.2596070855403657),
 ('Crunchiness', 'Ripeness', -0.2019816301087344),
 ('Juiciness', 'Crunchiness', -0.2596070855403657),
 ('Juiciness', 'Acidity', 0.24871388355218035),
 ('Ripeness', 'Weight', -0.24382426555524034),
 ('Ripeness', 'Sweetness', -0.27380006756070097),
 ('Ripeness', 'Crunchiness', -0.2019816301087344),
 ('Ripeness', 'Acidity', -0.20266908286974356),
 ('Acidity', 'Juiciness', 0.24871388355218035),
 ('Acidity', 'Ripeness', -0.20266908286974356),
 ('Quality_numeric', 'Size', 0.24400717867851435),
 ('Quality_numeric', 'Sweetness', 0.2509984588365l9),
 ('Quality_numeric', 'Juiciness', 0.26022254529522404),
 ('Quality_numeric', 'Ripeness', -0.264314879245958)]
```

Turning the correlation table into a heatmap and showing only one half

```
sns.heatmap(apples_corr, cmap= "Blues", vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)

plt.title('Correlation Heatmap')
plt.show()
```




```
mask = np.triu(np.ones_like(apples_corr, dtype=bool))

plt.figure(figsize=(10, 8))

sns.heatmap(apples_corr, mask=mask, cmap= "Blues", vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)

plt.title('Correlation Heatmap')
plt.show()
```

Code used for linear regression

```
#Linear regressions for variables with 2 or more correlations above/equal 0.2 or below/equal -0.2

# import libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
```

```
# Split the data columnwise into the independt variables (x) and the dependent variable (y)
xDf = apples[['Size','Ripeness']]
yDf = apples['Sweetness']

# Split between train and test set
X_train, X_test, y_train, y_test = train_test_split(xDf, yDf, test_size=0.3, random_state=0)

#fitting the model
reg_lin_sweet = LinearRegression().fit(X_train, y_train)

#getting the accuracy
print(reg_lin_sweet.score(X_test, y_test))
print(reg_lin_sweet.coef_)
```

```
0.21798845545811663
[-0.37174626 -0.33195112]
```

Code used for logistic regression

```
from sklearn.linear_model import LogisticRegression
xDf = apples[['Size','Sweetness','Juiciness','Ripeness', 'Weight', 'Acidity', 'Crunchiness']]
yDf = apples['Quality_numeric']

# Split between train and test set
X_train, X_test, y_train, y_test = train_test_split(xDf, yDf, test_size=0.3, random_state=0)

logreg2 = LogisticRegression()

# Fit the model to the training data
logreg2.fit(X_train, y_train)


# Evaluate the accuracy of the model
print(logreg2.score(X_test, y_test))
print(logreg2.coef_)
```

```
0.7275
[[ 0.65733981  0.59473852  0.46334712 -0.13444247  0.26102701 -0.2786047
   0.05296088]]
```

Code used for the Data Clustering Part 1

Employing the KMeans algorithm for several k values and plotting the data to identify the best k value

```
import random

random.seed(42)

#optional
xDf = apples.drop(columns='Quality')
#correct import
from sklearn.cluster import KMeans

#initialize inertia list and get the k and the inertia per k (or WCSS per k)
inertiaLst = []
for kVal in range(1, 10):
    kmeans = KMeans(n_clusters=kVal)
    kmeans.fit(xDf)
    inertiaLst.append([kVal, kmeans.inertia_])

#transposing and plotting the inertia list
inertiaArr = np.array(inertiaLst).transpose()
plt.plot(inertiaArr[0], inertiaArr[1])
plt.xlabel('k')
plt.ylabel('Inertia')
plt.show()
```

```
line_colors = ['blue','green', 'gray']
label_counts = apples["label"].value_counts()

plt.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%', textprops={'fontsize':10}, colors = line_colors)
plt.title('Label Distribution')
plt.show()
```

Generating the cluster labels for k = 3

```
#read kVal at the "elbow"
kVal = 3

#set the k for the KMeans to kVal
kmeans = KMeans(n_clusters=kVal)

#fit the model
kmeans.fit(xDf)

#get the label from the kmeans and add it as a column to the df
apples['label'] = kmeans.labels_
```

```
C:\Users\CZ\AppData\Local\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:
n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the w
  warnings.warn(
```

```
apples
```

	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality	Quality_numeric	label
0	-3.970049	-2.512336	5.346330	-1.012009	1.844900	0.329840	-0.491590	good	1	2
1	-1.195217	-2.839257	3.664059	1.588232	0.853286	0.867530	-0.722809	good	1	2
2	-0.292024	-1.351282	-1.738429	-0.342616	2.838636	-0.038033	2.621636	bad	0	0
3	-0.657196	-2.271627	1.324874	-0.097875	3.637970	-3.413761	0.790723	good	1	0
4	1.364217	-1.296612	-0.384658	-0.553006	3.030874	-1.303849	0.501984	good	1	0
...	...	...	...	...	...	...	...	...	...	...
3995	0.059386	-1.067408	-3.714549	0.473052	1.697986	2.244055	0.137784	bad	0	1
3996	-0.293118	1.949253	-0.204020	-0.640196	0.024523	-1.087900	1.854235	good	1	0
3997	-2.634515	-2.138247	-2.440461	0.657223	2.199709	4.763859	-1.334611	bad	0	1
3998	-4.008004	-1.779337	2.366397	-0.200329	2.161435	0.214488	-2.229720	good	1	2
3999	0.278540	-1.715505	0.121217	-1.154075	1.266677	-0.776571	1.599796	good	1	0

Generating the mean values for each cluster and getting the centroid value coordinates

```
apples.drop(columns = "Quality").groupby(by = "label").mean()
```

label	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality_numeric
0	0.393227	-1.061550	-0.311425	0.953723	1.623771	-0.754048	2.111096	0.614827
1	-0.017902	-0.932814	-1.818188	1.224925	-0.696652	1.320929	-0.709724	0.325064
2	-2.082487	-0.989401	1.148067	0.701567	0.950724	0.718099	-1.008151	0.614996

```
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[ 0.3994883  -1.06087439 -0.31465228  0.95336606  1.62516799 -0.75549405
   2.1089665   0.61581921]
 [-0.01696957 -0.93399311 -1.81602336  1.22532892 -0.69796081  1.31704442
  -0.71282752  0.32485696]
 [-2.08782104 -0.9887135   1.14860676  0.70138778  0.95351944  0.72176428
  -0.99682038  0.61447811]]
```

```
centroids = kmeans.cluster_centers_
feature_names = ['Size', 'Weight', 'Sweetness' , 'Crunchiness', 'Juiciness', 'Ripeness', 'Acidity' ,'Quality_numeric']

# Create a DataFrame to store the centroids and feature names
centroid_table = pd.DataFrame(centroids, columns=feature_names)

centroid_table
```

	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality_numeric
0	0.399488	-1.060874	-0.314652	0.953366	1.625168	-0.755494	2.108966	0.615819
1	-0.016970	-0.933993	-1.816023	1.225329	-0.697961	1.317044	-0.712828	0.324857
2	-2.087821	-0.988714	1.148607	0.701388	0.953519	0.721764	-0.996820	0.614478

Code used for the Data Clustering Part 2

Transposing and plotting the centroid coordinate values to get the plot per feature

```
transposed_centroid = centroid_table.transpose()
transposed_centroid
```

	0	1	2
Size	0.399488	-0.016970	-2.087821
Weight	-1.060874	-0.933993	-0.988714
Sweetness	-0.314652	-1.816023	1.148607
Crunchiness	0.953366	1.225329	0.701388
Juiciness	1.625168	-0.697961	0.953519
Ripeness	-0.755494	1.317044	0.721764
Acidity	2.108966	-0.712828	-0.996820
Quality_numeric	0.615819	0.324857	0.614478

```
line_colors = ['gray', 'green', 'blue']
fig = plt.figure(figsize=(12, 8))
for i, column in enumerate(transposed_centroid.columns):
    plt.plot(transposed_centroid[column], marker='o', linestyle='', markersize=12, color=line_colors[i])
plt.title('Characteristics for each Apple Variety', fontweight='bold')
plt.xlabel("Features", fontweight='bold')
plt.ylabel("Centroid Value", fontweight='bold')
plt.legend(transposed_centroid.columns)
```

Selecting all apples of one variety (in this case variety 0)

```
apples_0 = apples[apples["label"] == 0]
apples_0
```

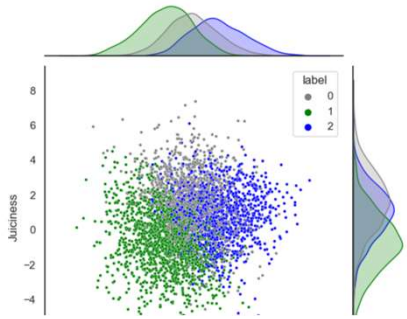
	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality	Quality_numeric	label
2	-0.292024	-1.351282	-1.738429	-0.342616	2.838636	-0.038033	2.621636	bad	0	0
3	-0.657196	-2.271627	1.324874	-0.097875	3.637970	-3.413761	0.790723	good	1	0
4	1.364217	-1.296612	-0.384658	-0.553006	3.030874	-1.303849	0.501984	good	1	0
6	1.331606	1.635956	0.875974	-1.677798	3.106344	-1.847417	2.414171	good	1	0
17	-0.074370	-4.714750	0.249768	2.935319	1.409755	-2.643810	1.250970	good	1	0
...	...	...	...	...	...	...	...	...	...	...
3969	3.300382	-1.662606	-1.724308	-0.074672	3.088414	0.720912	2.870162	good	1	0
3976	-0.524415	0.897921	-1.377694	1.579954	0.994690	-0.824029	2.422747	bad	0	0
3985	-0.230550	-0.669956	-1.896049	0.657545	1.843634	0.473194	1.461085	bad	0	0
3996	-0.293118	1.949253	-0.204020	-0.640196	0.024523	-1.087900	1.854235	good	1	0
3999	0.278540	-1.715505	0.121217	-1.154075	1.266677	-0.776571	1.599796	good	1	0

1241 rows × 10 columns

Plotting two features with coloring by label and a distribution plot as a joint plot

```
plt.figure(figsize=(15, 10))
sns.set(style="white")

sns.jointplot(x='Sweetness', y='Juiciness', hue='label', data=apples, palette=line_colors, s=9)
```

```
<seaborn.axisgrid.JointGrid at 0x229bb5e1450>

<Figure size 1500x1000 with 0 Axes>
```



Conducting independent two sample t tests for two varieties and one feature

```
from scipy import stats
```

```
# Sample data
sweet_1 = apples_1["Sweetness"]
sweet_2 = apples_2["Sweetness"]

# Perform independent t-test
t_statistic, p_value = stats.ttest_ind(sweet_2, sweet_1)

# Print the results
print("t-statistic:", t_statistic)
print("p-value:", p_value)
```

```
t-statistic: 50.538669333001174
p-value: 0.0
```