Assignment 1

Objectives

1. A descriptive analysis of the additives and graphical analysis
2. A clustering to determine the distinctive number of formulations present in data

Missing values: None

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   a       214 non-null    float64
 1   b       214 non-null    float64
 2   c       214 non-null    float64
 3   d       214 non-null    float64
 4   e       214 non-null    float64
 5   f       214 non-null    float64
 6   g       214 non-null    float64
 7   h       214 non-null    float64
 8   i       214 non-null    float64
```

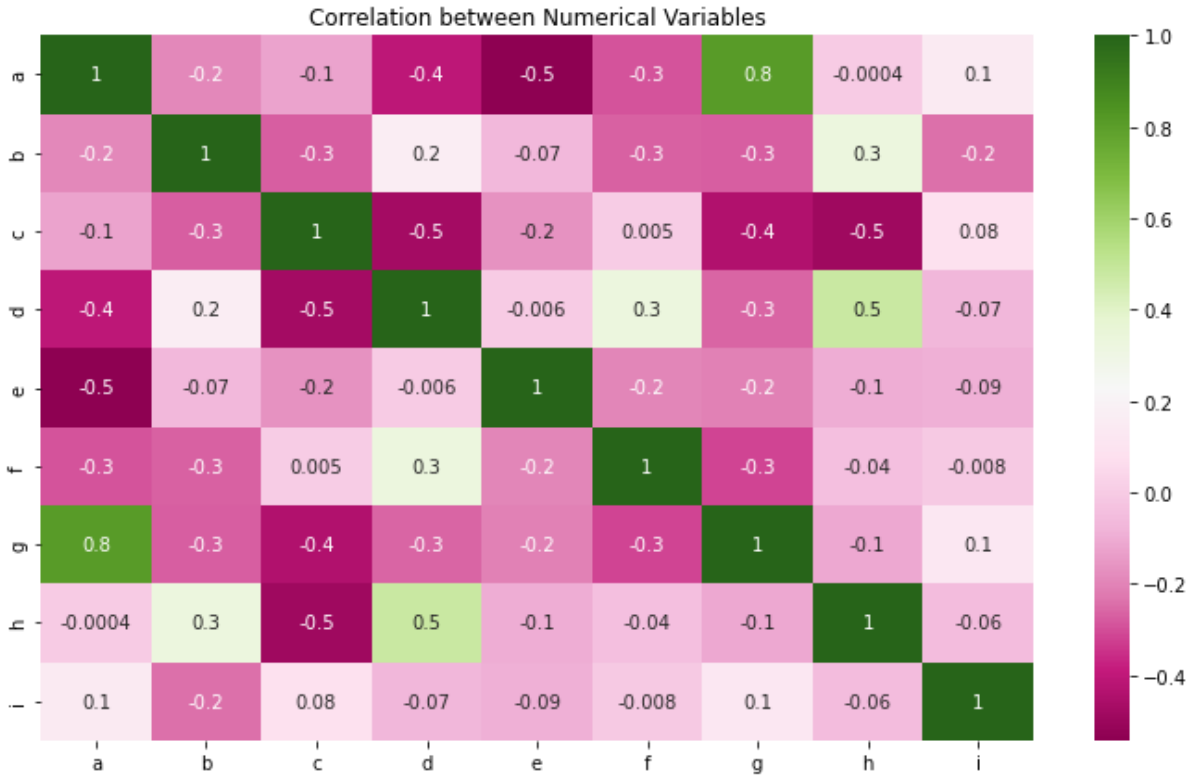
Major steps to be performed

1. Load data
2. Exploratory data analysis
3. Cluster and exclude outliers using DBSCAN
4. Segregate clean data into distinct formulations
5. Hypothesis testing on distinct formulations

Descriptive analysis:

	a	b	c	d	e	f	g	h	i
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000
mean	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	8.956963	0.175047	0.057009
std	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.423153	0.497219	0.097439
min	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.000000	0.000000
25%	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.240000	0.000000	0.000000
50%	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.600000	0.000000	0.000000
75%	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.172500	0.000000	0.100000
max	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.190000	3.150000	0.510000

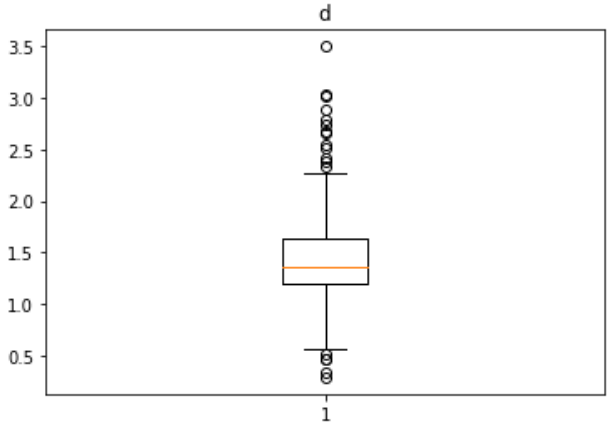
Correlation between the variables:

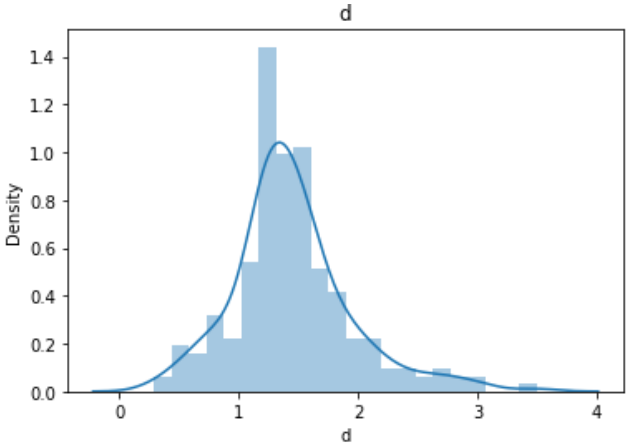


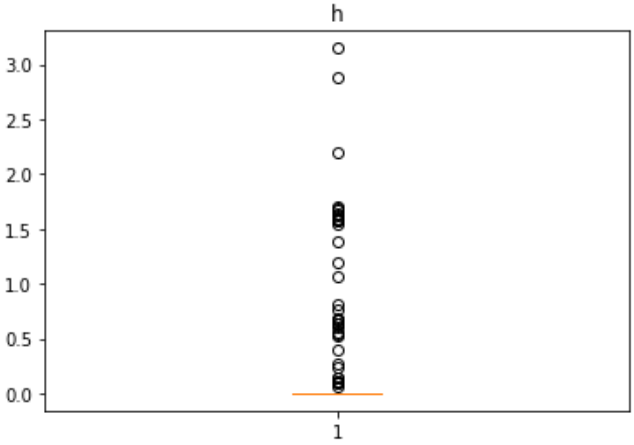
Significant correlation:

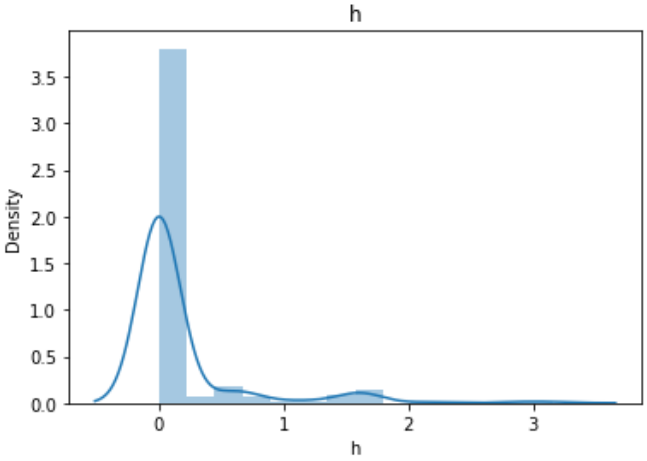
- a is strongly correlated with g (0.8 correlation)
- when h increase, d increase (0.5 correlation)
- when c increase, d decrease (-0.5 correlation)

Some highlights on boxplots and distribution plot:









```
df.skew()
```

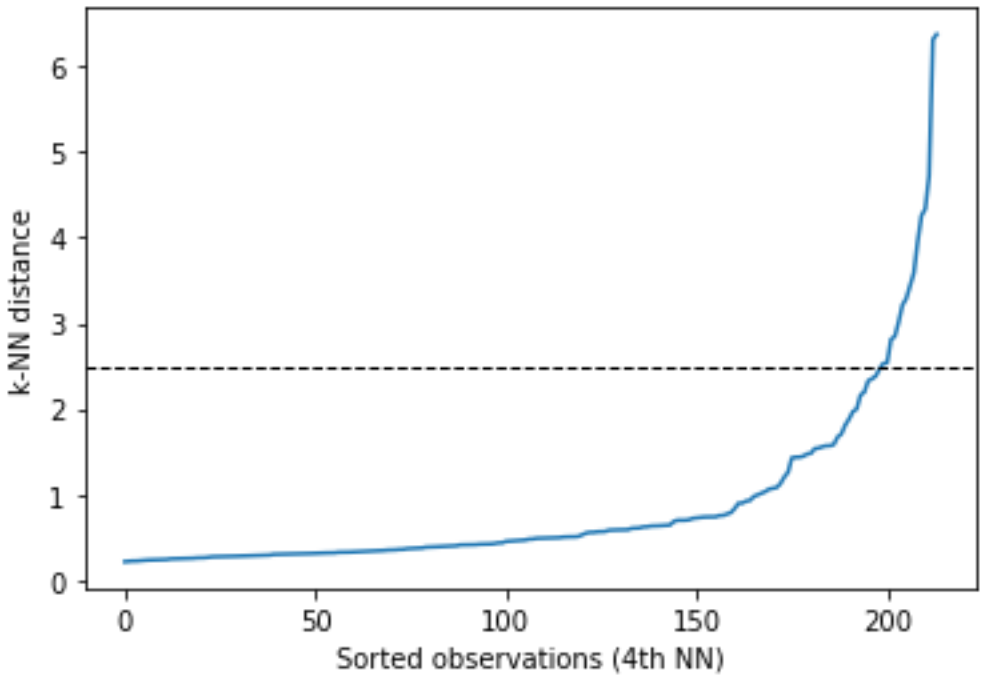
```
a    1.625431
b    0.454181
c   -1.152559
d    0.907290
e   -0.730447
f    2.541076
g    2.047054
h    2.775887
i    1.552263
```

As we can see, all the variables are extremely skewed.

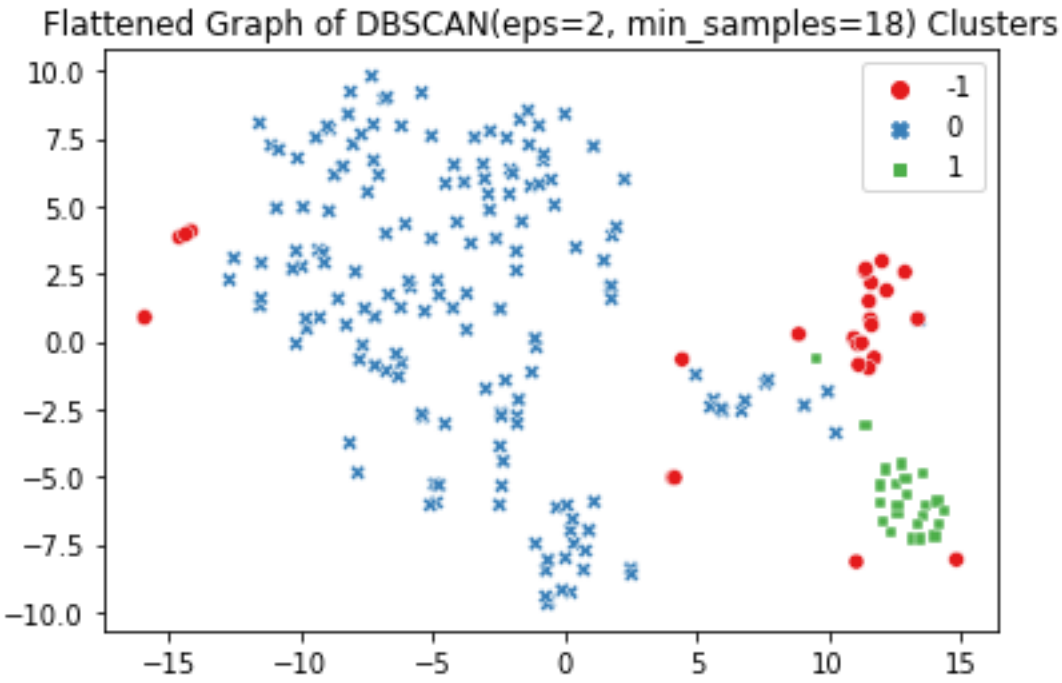
DBSCAN for outliers

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) method is used to cluster the outliers. It takes multi-dimensional data as inputs and clusters them according to the model parameters, epsilon and minimum samples. Based on these parameters, the algorithm determines whether certain values in the dataset are outliers or not.

'Knee' of the curve from KNN distance plot is used to identify epsilon value:



Minimum samples = number of features x 2



There are 27 points of value -1 which represents noisy points that could not assigned to any cluster, hence the outliers.

Data without outliers are collected into a new dataset:

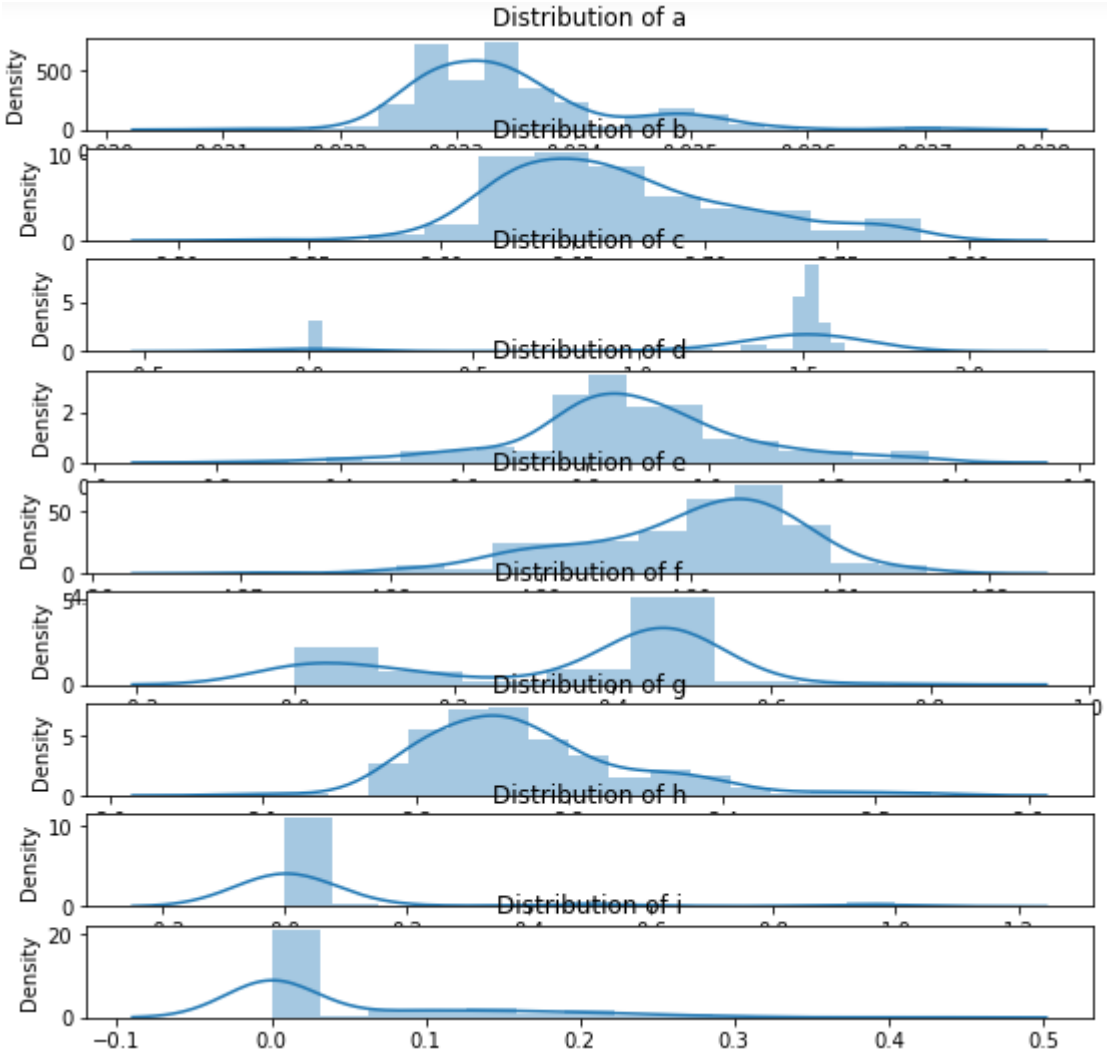
```
clean_data = df_new[(df_new['Cluster'] != -1)]
```

K-Means Clustering to Segregate Distinct Formulations

K-means uses Euclidean distance as a distance metric to calculate the distance between each point and the centroid.

Since K-means gives the best result when data's distribution is not skewed and data is standardised (i.e. mean of 0 and standard deviation of 1), I use log transformation to further reduce skewness of data. I added a small constant as log transformation demands all the values to be positive.

After log transform:

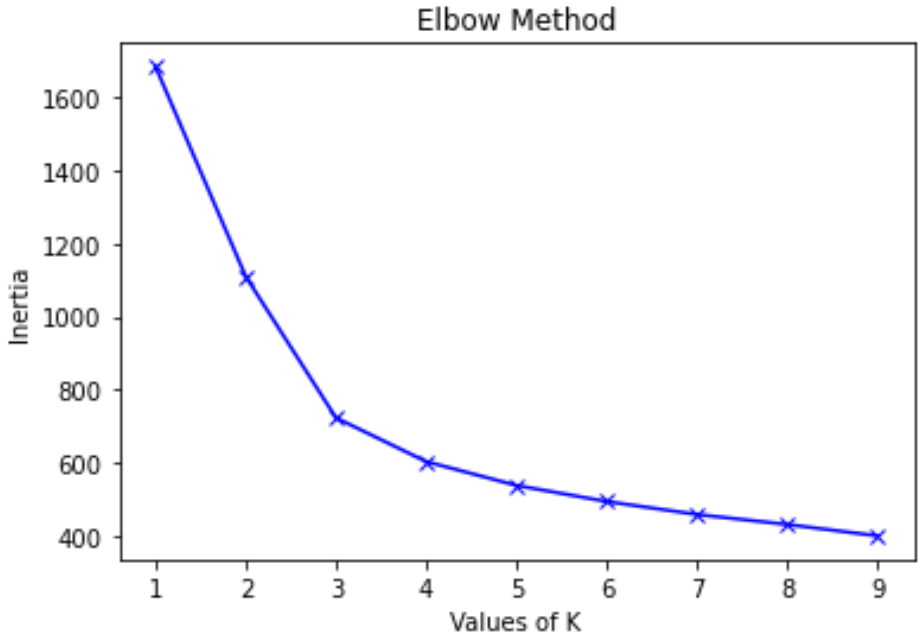


Once the skewness is reduced, I standardised the data by centring and scaling.

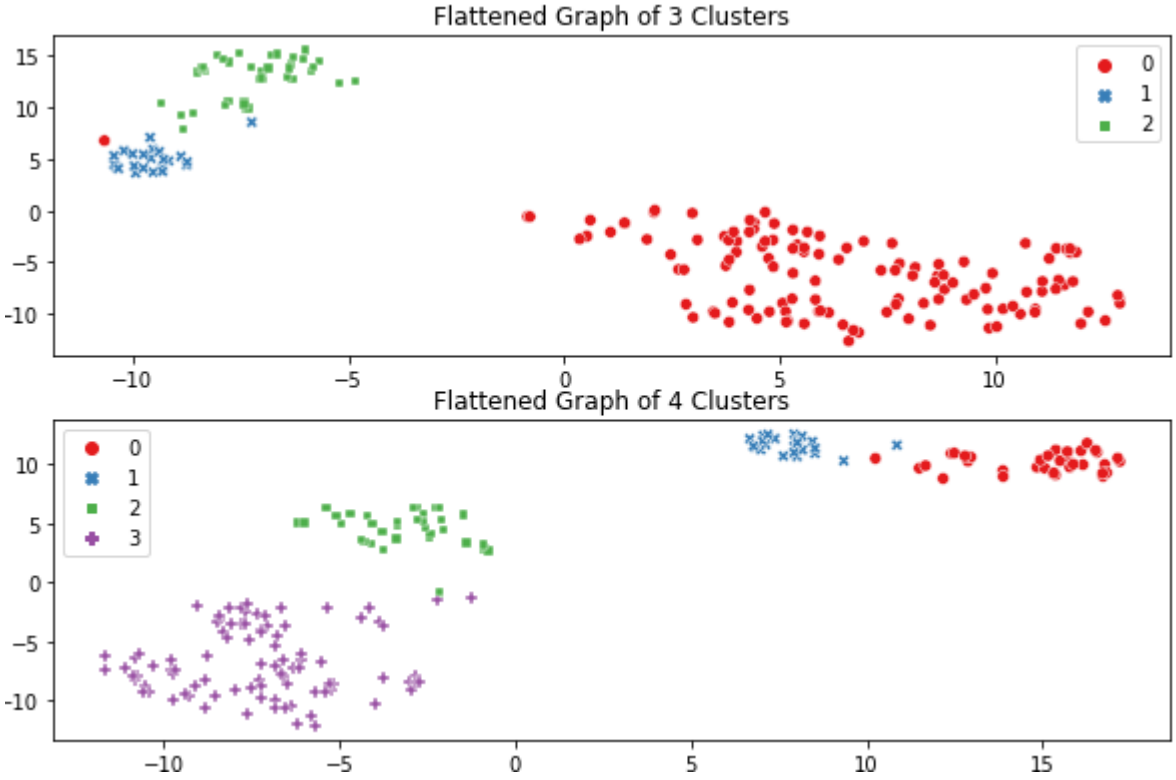
```
scaler = StandardScaler()
scaler.fit(df_rfm_log)
Table_scaled = scaler.transform(df_rfm_log)
Table_scaled
```

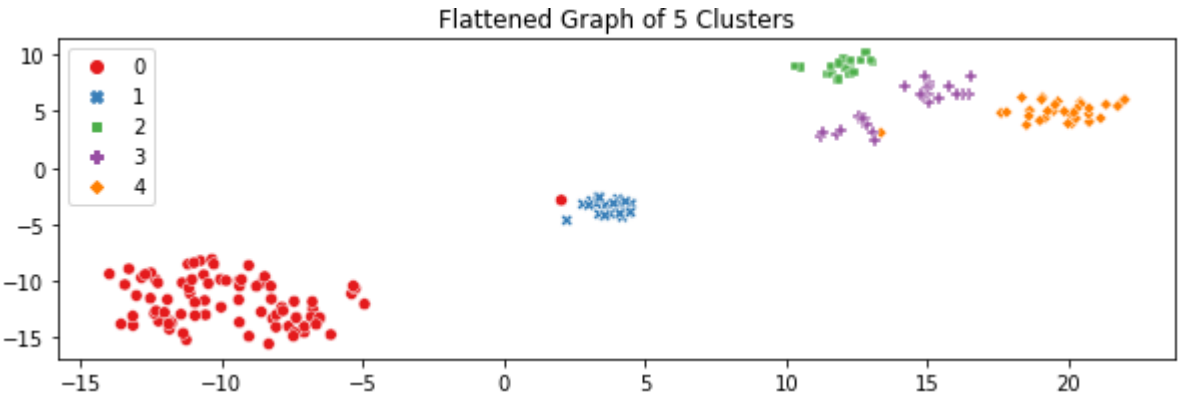
	a	b	c	d	e	f	g	h	i
0	-0.308061	-0.606663	0.441220	0.653893	0.075458	0.486304	-0.397642	-0.378769	0.174823
1	2.296027	-0.151176	0.458623	-1.504733	-1.268337	-1.156845	1.978154	-0.378769	-0.272150
2	-0.155804	-1.348848	0.432461	0.092282	0.832165	0.585852	-0.248868	-0.378769	-0.618923
3	0.051789	0.034606	0.624735	-0.507543	-0.358884	0.618613	-0.838168	-0.378769	-0.618923
4	-0.372662	-0.845970	0.414826	-0.139893	0.634395	0.552883	-0.412606	-0.378769	-0.618923

I apply Elbow Method to identify optimal number of clusters. Elbow Method simply tells the optimal cluster number for optimal inertia.



Here it looks like 3 is the optimal one. I tried my analysis with 3,4 and 5 clusters to make decision:





The technique for flattening high dimensional graph and visualising it in a two-dimensional format is known as t-Distributed Stochastic Neighbor Embedding (t-SNE). I decided to go with 4 clusters.

Interpretation of the clusters formed using k-means using median. I used median because it is unaffected by the skewed data.

Cluster	a	b	c	d	e	f	g	h	i
Formulation 1	1.517355	13.205	3.525	1.365	72.845	0.59	8.405	0.00	0.000
Formulation 2	1.517720	12.960	3.530	1.285	72.915	0.58	8.480	0.00	0.215
Formulation 3	1.521520	13.795	3.650	0.900	71.990	0.11	9.630	0.00	0.000
Formulation 4	1.516400	14.560	0.000	2.250	73.230	0.00	8.670	0.76	0.000

Hypothesis Testing on Formulations

From the problem statement, third party certification organisation would like to verify if the formulations are significantly different. So, I have to decide whether to use one-way ANOVA or Kruskal–Wallis test, which is a nonparametric approach to the one-way ANOVA.

First, I performed Shapiro test to test for normality:

```
# Normality Test
print(stats.shapiro(result['Formulation 1'])) # Shapiro Test
print(stats.shapiro(result['Formulation 2']))
print(stats.shapiro(result['Formulation 3']))
print(stats.shapiro(result['Formulation 4']))
```

```
ShapiroResult(statistic=0.5408061146736145, pvalue=1.9301562133478e-05)
ShapiroResult(statistic=0.5384564399719238, pvalue=1.8112325051333755e-05)
ShapiroResult(statistic=0.553828239440918, pvalue=2.7454823793959804e-05)
ShapiroResult(statistic=0.5456883311271667, pvalue=2.2027727027307265e-05)
```

All formulation failed the normality test since p-value are less than 0.05.

Then I proceed with variance test to check that variances are equal for all samples:

```
# Variance test
scipy.stats.levene(result['Formulation 1'], result['Formulation 2'],
                   result['Formulation 3'],result['Formulation 4'])
```

```
LeveneResult(statistic=0.0003066971882176188, pvalue=0.9999924071318681)
```

Since p-value is more than 0.05, variances are equal for all formulations.

One- way ANOVA assumptions are:
The populations must be normally distributed and the populations must have the same variance.

I proceeded with Kruskal–Wallis test:

```
# Conduct the Kruskal-Wallis Test
test = stats.kruskal(result['Formulation 1'], result['Formulation 2'],
                     result['Formulation 3'],result['Formulation 4'])

# Print the result
print(test)
```

```
KruskalResult(statistic=0.08854482898199191, pvalue=0.9931757256816306)
```

Conclusion:
Null hypothesis is accepted since p-value more than 0.05. Formulations are equal.

Assignment 2: Analysis on how external factors influence fresh fruit bunch (FFB) yield

Objective:

To predict fresh fruit bunch (FFB) yield of oil palm trees. This is an attempt to use Random Forest Regressor to find out important features in the prediction.

Major Steps to be Performed:

1. Load data
2. Setup libraries
3. Perform descriptive analysis (EDA)
4. Feature engineering
5. Split train and test set
6. Train model
7. Predict

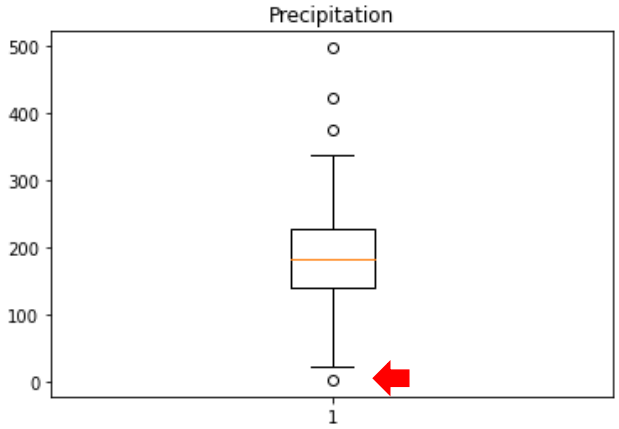
Loading dataset and setting up all libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display
from statistics import mode
from scipy.special import binom
from scipy.stats import iqr
pd.options.display.max_columns = None

df = pd.read_csv('palm_ffb.csv')

df.head()
```

Below is the sample outcome from my Jupyter notebook.

	Date	SoilMoisture	Average_Temp	Min_Temp	Max_Temp	Precipitation	Working_days	HA_Harvested	FFB_Yield
0	01.01.2008	616.4	25.306452	21.3	32.2	184.4	25	777778.3951	1.62
1	01.02.2008	568.9	26.165517	20.9	35.1	140.2	23	767988.2759	1.45
2	01.03.2008	577.6	25.448387	21.3	32.9	280.4	25	783951.9231	1.56
3	01.04.2008	581.1	26.903333	20.6	34.8	173.3	25	788987.0504	1.39
4	01.05.2008	545.4	27.241935	20.9	35.0	140.6	25	813659.7222	1.44

There is one outlier which turns out to be a mistyped data found in ‘precipitation’ variable:

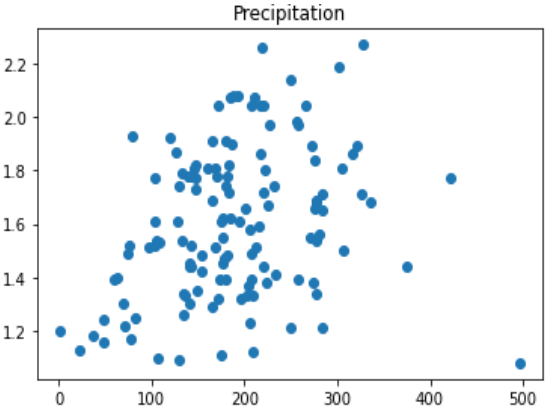


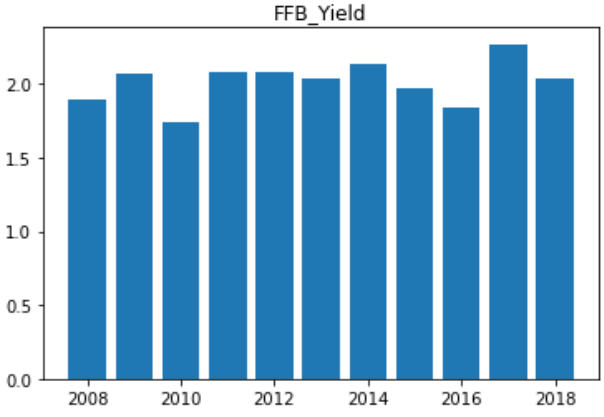
	SoilMoisture	Average_Temp	Min_Temp	Max_Temp	Precipitation	Working_days	HA_Harvested	FFB_Yield	year	month	dayofweek
count	130.000000	130.000000	130.000000	130.000000	130.000000	130.000000	130.000000	130.000000	130.000000	130.000000	130.0
mean	527.646923	26.849918	21.379231	33.851538	188.980769	24.753846	793404.491565	1.602231	2012.923077	6.423077	1.0
std	57.367844	0.651413	0.688971	1.079638	80.237210	1.239289	34440.893854	0.281751	3.136718	3.434583	0.0
min	380.700000	25.158065	18.900000	31.100000	2.000000	21.000000	683431.944400	1.080000	2008.000000	1.000000	1.0
25%	488.625000	26.442285	21.000000	33.100000	140.300000	24.000000	768966.949100	1.390000	2010.000000	3.250000	1.0
50%	538.300000	26.930645	21.500000	33.900000	182.150000	25.000000	790036.158050	1.585000	2013.000000	6.000000	1.0
75%	571.025000	27.270726	21.800000	34.600000	226.100000	26.000000	821989.235250	1.807500	2016.000000	9.000000	1.0
max	647.300000	28.580000	22.600000	36.000000	496.100000	27.000000	882254.225400	2.270000	2018.000000	12.000000	1.0

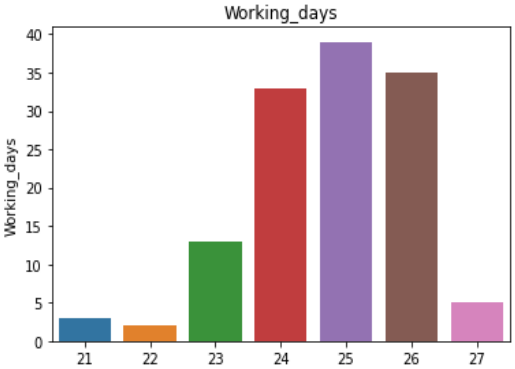
Value is replaced into 22mm.

Exploratory Data Analysis

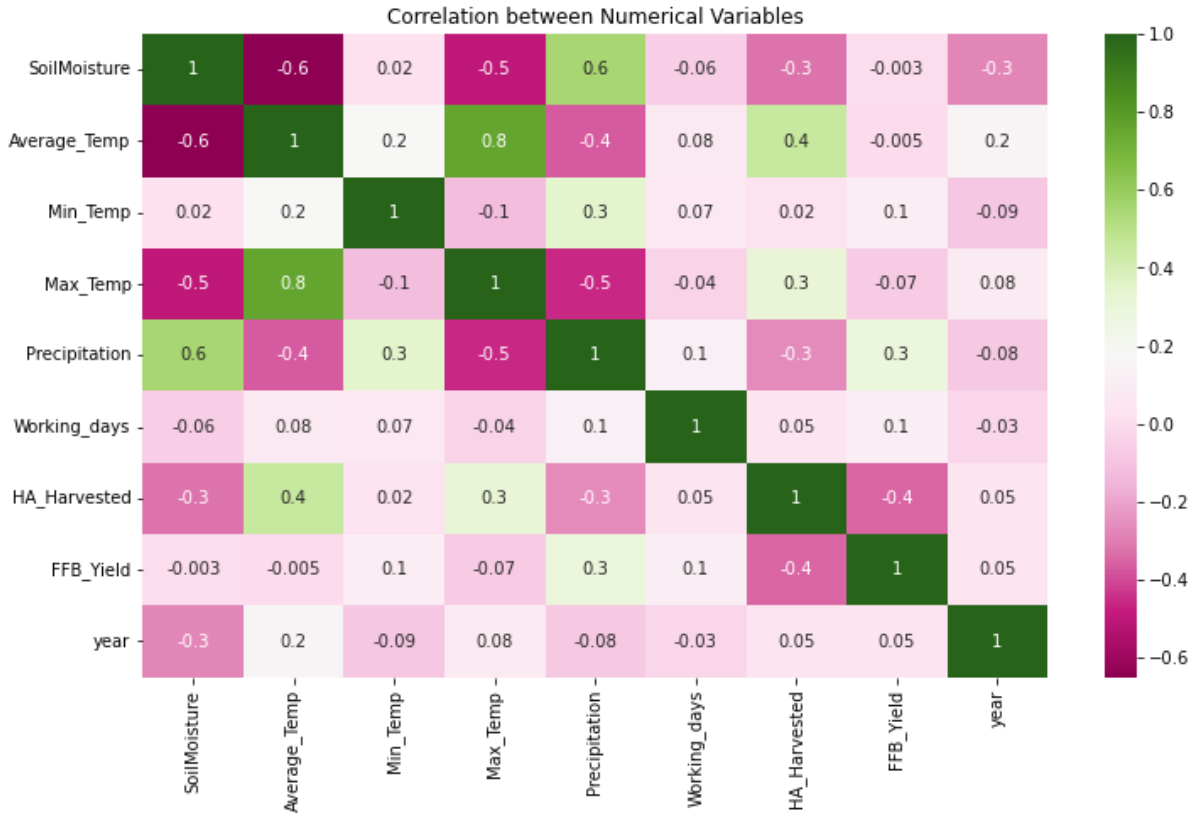
Below are a few highlights on visualizations performed to understand data:







Identify the correlation between all features using heatmap

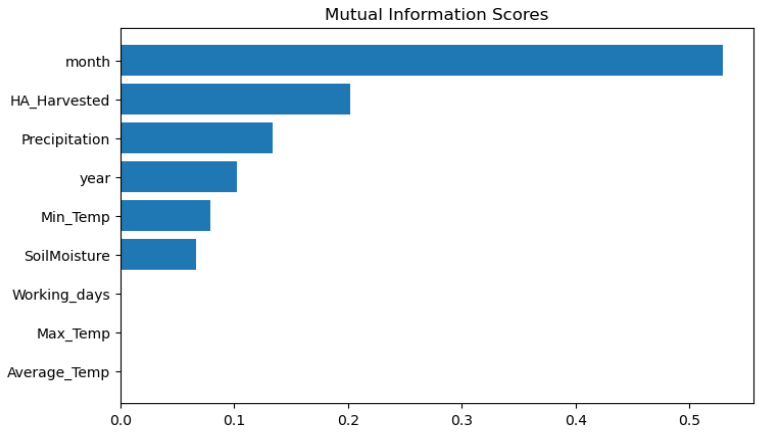


Insight from heatmap:

1. Month is very strongly correlated with yield (correlation 0.7)
2. Increase of Avg Temp cause lower soil moisture (correlation -0.6)
3. Higher precipitation causes higher soil moisture (correlation 0.6)
4. Max Temp very strong correlation with Avg Temp (correlation 0.8)
5. When Avg Temp increase, HA Harvested increase (correlation 0.4)
6. High precipitation causes lower Avg Temp (correlation 0.4)
7. High HA Harvested cause lower FFB Yield (correlation -0.4)

Feature Engineering

I construct a ranking with a feature utility metric, a function measuring associations between a feature and the target.



I used Random Forest Regressor to establish a baseline score to help my determine whether my new features are actually useful.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score

def score_dataset(X, y,
model=RandomForestRegressor(criterion='absolute_error',
random_state=0)):

    for colname in X.select_dtypes(["category"]):
        X[colname] = X[colname].cat.codes
    # Metric is MAE
    score = cross_val_score(
    model, X, y, cv=5, scoring="neg_mean_absolute_error"
    )
    score = -1 * score.mean()
```

```
X = df.copy()
y = X.pop("FFB_Yield")

baseline_score = score_dataset(X, y)
print(f"Baseline score: {baseline_score:.5f} MAE")
```

My baseline score is 0.15897 MAE

Steps I used for feature engineering:

```
# 1. Maximum and minimum temperature ratio
X['Temp_Ratio'] = X['Min_Temp']/X['Max_Temp']

# 2. Drop Max Temp
X = X.drop(['Max_Temp'],axis =1 )

# 3. Create a new feature using K-Means Clustering
features = ["month", "Precipitation",'HA_Harvested']
# Standardize
X_scaled = X.loc[:, features]
X_scaled = (X_scaled - X_scaled.mean(axis=0)) /
X_scaled.std(axis=0)

kmeans = KMeans(n_clusters=10, n_init=10, random_state=0)
```

Final baseline error: 0.1515

Train and test model

Models used for predicting test set:

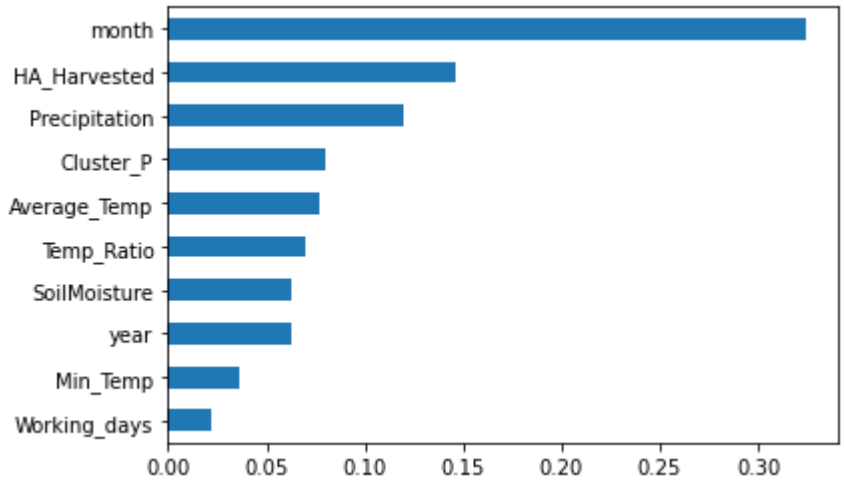
```
# 1. Random Forest Regressor with hyperparameter tuning using
GridSearchCV

from sklearn.model_selection import GridSearchCV
parameters = {'n_estimators':range(10,300,10),
'criterion':('mse','mae'), 'max_features':('auto','sqrt','log2')}

gs =
GridSearchCV(rf,parameters,scoring='neg_mean_absolute_error',cv=3
)
gs.fit(X_train,Y_train)

gs.best_score_
gs.best_estimator_

tpred_rf = gs.best_estimator_.predict(X_test)
print('mae:',mean_absolute_error(Y_test,tpred_rf))

# 2. Linear Regressor
lm = LinearRegression()
lm.fit(X_train, Y_train)
tpred_lm = lm.predict(X_test)

# 3. Lasso Regressor
lm_l = Lasso(alpha=.13)
lm_l.fit(X_train,Y_train)
tpred_lml = lm_l.predict(X_test)

# 4. Gradient Boosting Regressor with hyperparameter tuning
gbm_model = GradientBoostingRegressor().fit(X_train, Y_train)

parameters = {'learning_rate':[0.1, 0.5, 1, 1.5],
'max_depth':[1,3,5,7], 'loss':('squared_error', 'absolute_error',
'huber', 'quantile'),
             'criterion':('friedman_mse', 'squared_error',
'mse'),}

gs2 =
GridSearchCV(gbm_model,parameters,scoring='neg_mean_absolute_erro
r',cv=3)
gs2.fit(X_train,Y_train)
gs2.best_score_
gs2.best_estimator_

tpred_gbm = gs2.best_estimator_.predict(X_test)
```

MAE score by each model:

1. Random forest regressor: 0.15337948717948727
2. Linear regressor: 0.15997919223199422
3. Lasso regressor: 0.16673449990578718
4. Gradient boosting regressor: 0.158732709093196

Model chosen: Random Forest Regressor

Feature Importance:



Insight:

Main features that determine FFB Yield are month, harvested area, precipitation amount and average temperature.

Assignment 3:

a. What is the probability of the word “data” occurring in each line?
Steps:
i. Count occurrence of word ‘data’ in each line
ii. Count occurrence of all words in each line
iii. Divide step i by step ii

Code snippet:

```
# Q1: count probability of the word 'data' occuring in each line
# count occurrence of 'data' in each line
nm = 'data'
occurences = [item.count(nm) for item in  lines_list2]
count = sum(occurences)
print(occurences) # This will print out all occurences of input string
print('count = ', count)
# count total words in each line
total_word = []
for x,word in enumerate(lines_list2):
    num = (len(word.split()))
    total_word.append(num)
# probability of word 'data' occuring in each line
prob = list(map(truediv, occurences, total_word))
print ("The probability of of the word “data” occurring for each line is : " + str(prob))
```

b. What is the distribution of distinct word counts across all the lines?
Steps: Use nltk to count frequency of unique words then plot frequency distribution

Code snippet:

```
# Q2: Frequency distribution of unique words
import nltk
# join into paragraph
paragraph = " ".join(lines_list2)
nltk.download('punkt')
words = nltk.tokenize.word_tokenize(paragraph)
fdist1 = nltk.FreqDist(words)
filtered_word_freq = dict((word, freq) for word, freq in fdist1.items() if not word.isdigit())
print(filtered_word_freq)
# frequency distribution of words
import matplotlib.pyplot as plt
plt.barh([ str(i) for i in filtered_word_freq.keys()], filtered_word_freq.values(), color='g')
```

c. What is the probability of the word “analytics” occurring after the word “data”?
Step:
i. Use conditional probability to calculate probability of word ‘analytics’ occurring given probability of word ‘data’ occurred.

```
# Q3: probability of 'analytics' occurring given 'data'occurred in paragraph
# Bayes theorem --> conditional probability
paragraph = str.split(paragraph)
tot_word = 0
count_d = 0
count_a = 0
for lines in  paragraph:
    tot_word = tot_word + 1
    if lines == 'data':
        count_d = count_d + 1
    if lines == 'analytics':
        count_a = count_a + 1
prob_data = count_d / tot_word
prob_analytics = count_a / tot_word
print('P(analytics_given_data):',(prob_data*prob_analytics)/prob_data)
```