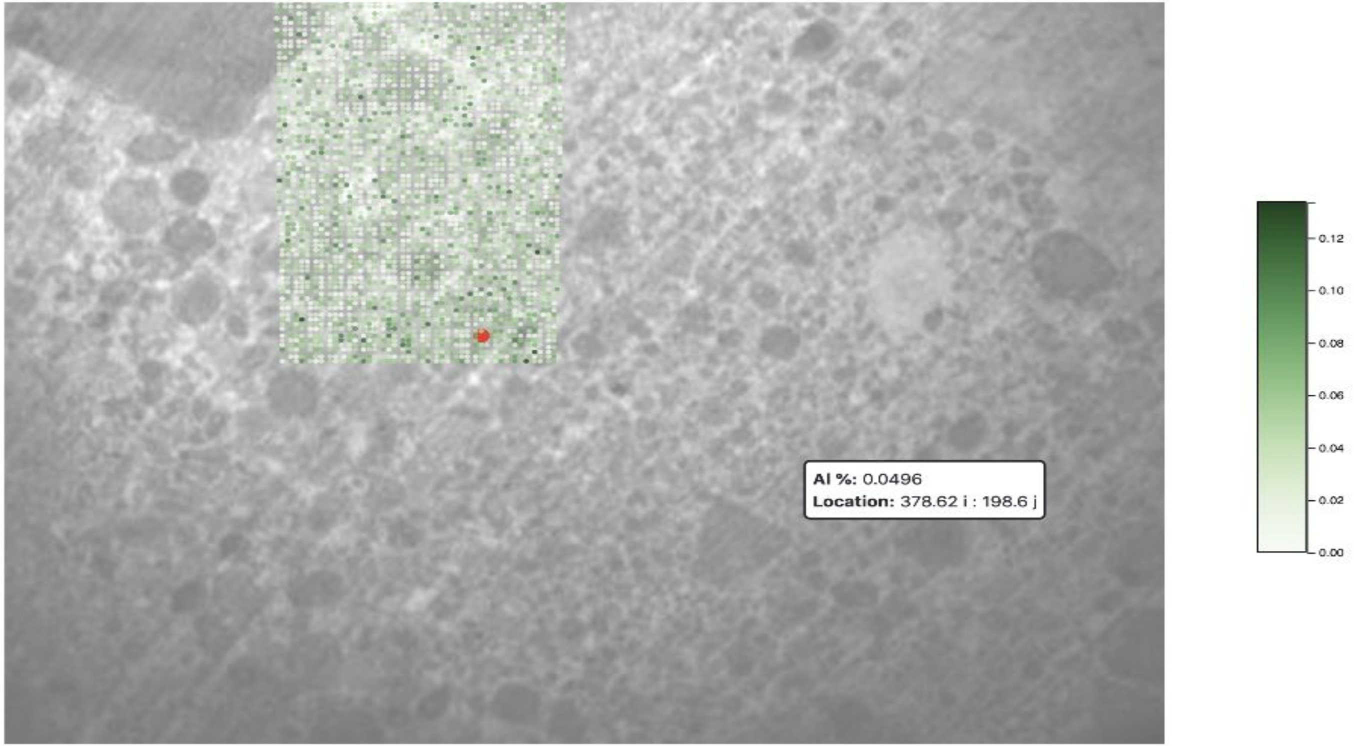
Mars Rover Mini Project

Frank Serdenia, Kaylee Pham, Nafisa Habib, Michael Castillo

Rimma Hämäläinen

Dr. Li Liu



Detector A Detector B

Mars Rover Dataset

In [1]:

```python
# Load libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

data=pd.read_csv("dataset_Mars.csv") # Load dataset
data
```

Out[1]:

	PMC	Detector	Mg_%	Al_%	Ca_%	Ti_%	Fe_%	Si_%	Mg_int	Al_int	Ca_int	Ti_int	Fe_int	Si_int	image_i	image_j
0	7	A	0.4605	0.0305	22.9336	0.1140	1.0066	0.0067	31.9	8.0	50624.2	152.2	2015.6	5.0	409.04	416.18
1	7	B	0.1968	0.0735	22.4898	0.1392	1.1196	0.0196	13.8	19.8	50480.9	188.8	2270.2	14.9	409.04	416.18
2	8	A	0.5142	0.0000	22.6415	0.1949	1.0006	0.0130	35.7	0.0	50074.6	261.4	2009.1	9.7	408.99	413.59
3	8	B	0.4354	0.0271	21.9504	0.2441	1.0346	0.0150	30.8	7.3	49673.5	335.7	2123.7	11.4	408.99	413.59
4	9	A	0.3532	0.0570	26.5924	0.1641	0.3400	0.0179	24.7	15.2	57985.5	211.6	665.5	13.5	408.94	410.99
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8050	4037	A	0.2774	0.0165	20.7999	0.0357	0.6332	0.0000	19.5	4.4	46253.5	47.3	1267.2	0.0	304.80	414.02
8051	4037	B	0.5039	0.0297	22.1842	0.0652	0.7391	0.0070	36.2	8.0	50318.6	88.8	1520.9	5.4	304.80	414.02
8052	4038	A	0.3707	0.0599	22.2514	0.0329	0.9844	0.0326	25.6	15.7	48856.2	43.6	1963.0	24.2	304.76	416.61
8053	4038	B	0.2642	0.0576	21.6834	0.0768	0.9042	0.0106	18.4	15.3	47978.6	102.1	1808.3	8.0	304.76	416.61
8054	3651	B	0.2766	0.0098	19.9610	0.0847	1.1792	0.0071	19.3	2.6	44915.8	115.5	2403.7	5.4	315.01	408.78

8055 rows × 16 columns

Goal

- Explore and analyze the relationships between the element quantifications
- Find underlying patterns using different methods
- Use visuals to better understand and conceptualize the data set
- Algorithms:
    - K-Means Clustering
    - Hierarchical Clustering
    - Principal Component Analysis

K-Means Algorithm in Clustering

- Unsupervised learning
- Fast and easier to understand
- Good for well separated data
- ★ Steps:
  - Input K number of cluster
  - Initialize centroids using random K data points for the centroids with no replacement
  - Keep iterating until there is no change to the centroids

Source: K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks
https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

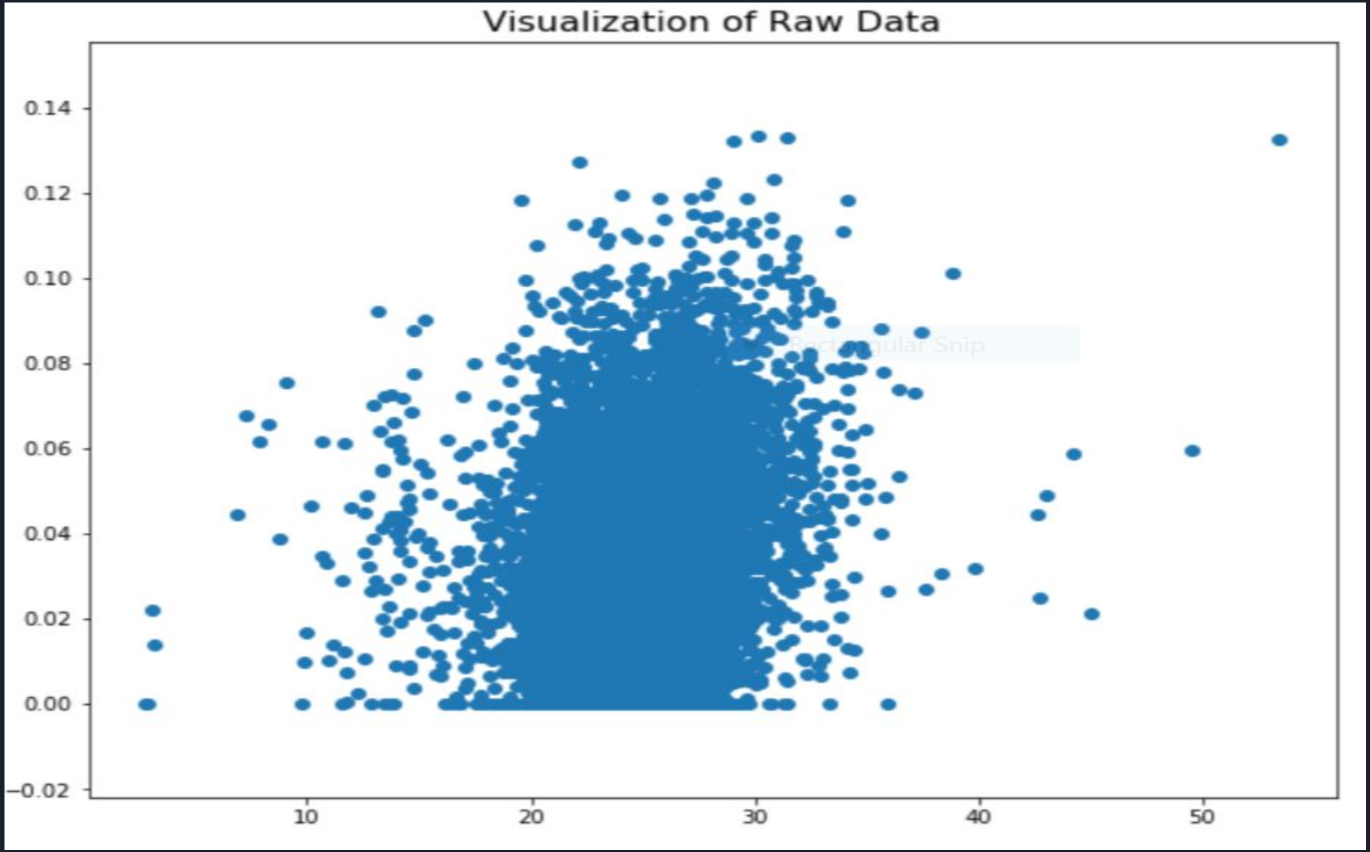
In [2]:

```python
# Select a subset %_value of data
feature_cols = data.drop(columns=["PMC","Detector","Mg_int","Al_int", "Ca_int",
                                  "Ti_int", "Fe_int", "Si_int", "image_i", "image_j"])
print(feature_cols)

features=np.array(feature_cols)
#print("\nArray of Features\n")
#print(features)

# Plot subset %_value
plt.scatter(features[:,2],features[:,1])
plt.gcf().set_size_inches(10,8)
plt.title('Visualization of Raw Data',fontsize=16)
```

	Mg_%	Al_%	Ca_%	Ti_%	Fe_%	Si_%
0	0.4605	0.0305	22.9336	0.1140	1.0066	0.0067
1	0.1968	0.0735	22.4898	0.1392	1.1196	0.0196
2	0.5142	0.0000	22.6415	0.1949	1.0006	0.0130
3	0.4354	0.0271	21.9504	0.2441	1.0346	0.0150
4	0.3532	0.0570	26.5924	0.1641	0.3400	0.0179
...	...	...	...	...	...	...
8050	0.2774	0.0165	20.7999	0.0357	0.6332	0.0000
8051	0.5039	0.0297	22.1842	0.0652	0.7391	0.0070
8052	0.3707	0.0599	22.2514	0.0329	0.9844	0.0326
8053	0.2642	0.0576	21.6834	0.0768	0.9042	0.0106
8054	0.2766	0.0098	19.9610	0.0847	1.1792	0.0071

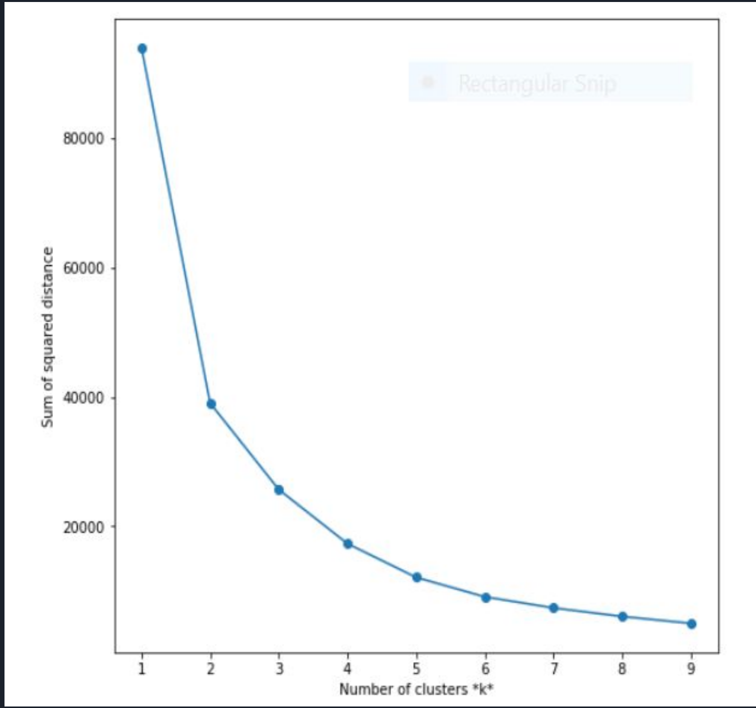
[8055 rows x 6 columns]

Al_%


Visualization of Raw Data
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
10
20
30
40
50

Ca_%

```
In [3]:
# Elbow Method to find a good k number
# Run the Kmeans algorithm and get the index of data points clusters
sse = []
list_k = list(range(1, 10))

for k in list_k:
    km = KMeans(n_clusters=k)
    km.fit(features)
    sse.append(km.inertia_)

# Plot sse against k
plt.figure(figsize=(8, 8))
plt.plot(list_k, sse, '-o')
plt.xlabel(r'Number of clusters *k*')
plt.ylabel('Sum of squared distance')
```

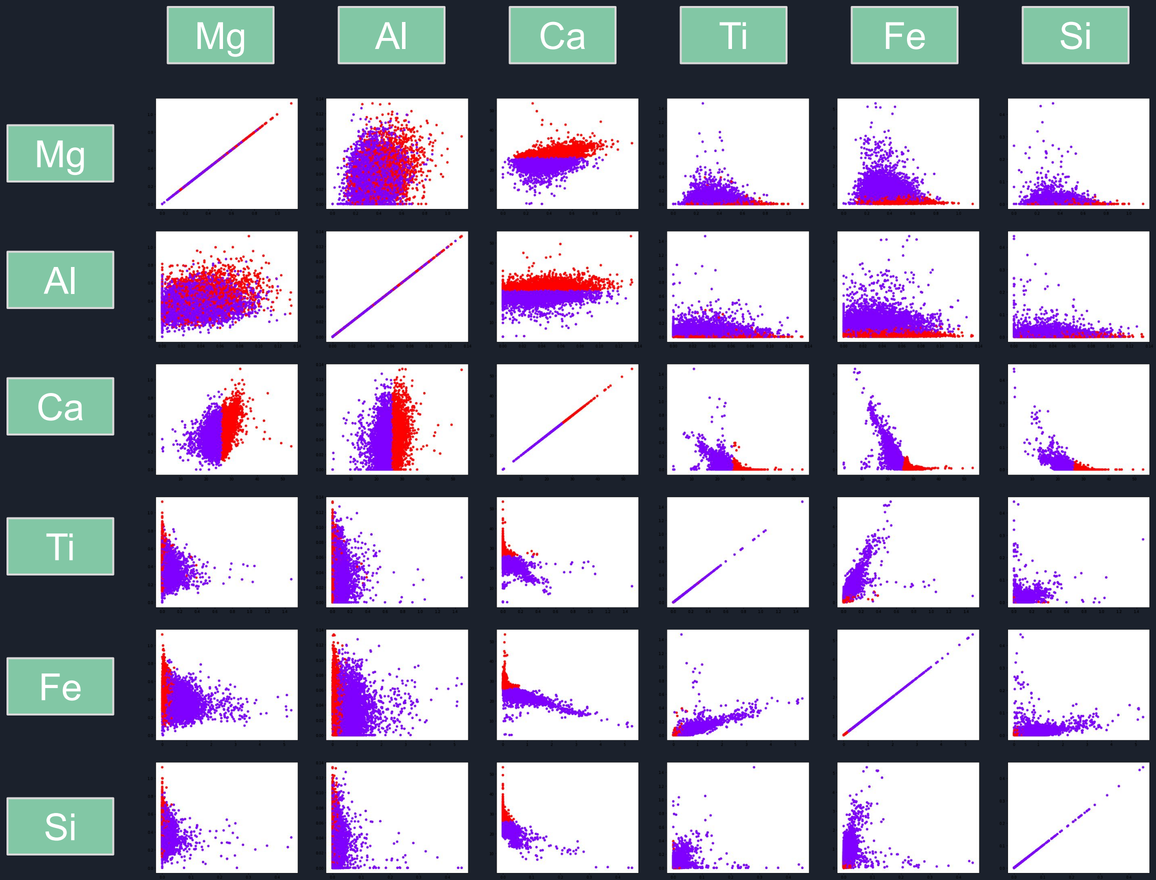


```
In [4]:
# Instantiate k-means algorithm
kmeans = KMeans(n_clusters=2)

# Fit the algorithm to the features
kmeans.fit(features)

# Finding the centroid
centroids = kmeans.cluster_centers_

# Compute the silhouette score
kmeans_silhouette = silhouette_score(
features, kmeans.labels_).round(2)
```

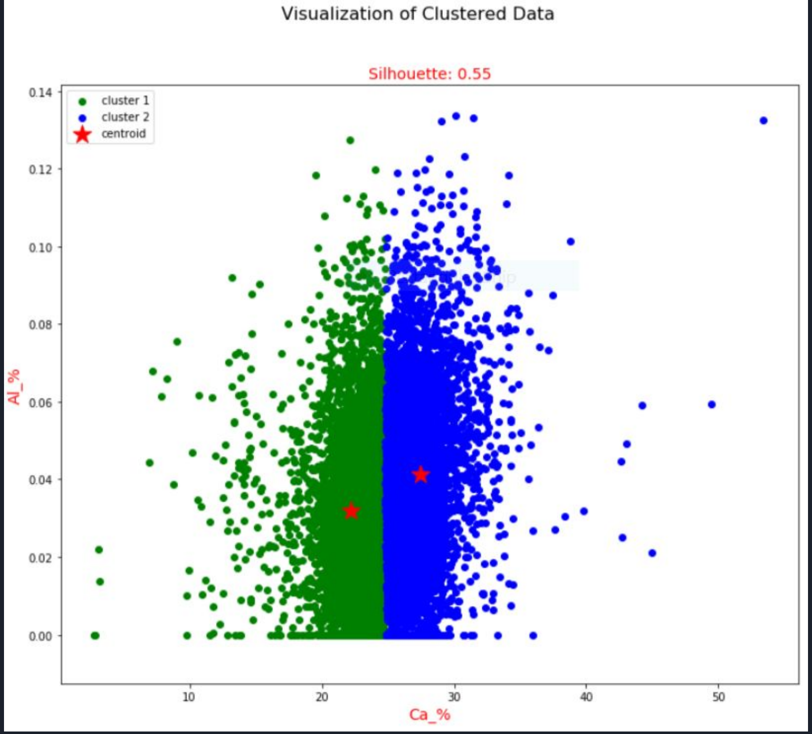
```
In [5]:
kmeans_silhouette  # between -1 and 1, closer to 1 is more accurate
```

Out[5]: 0.55

Clusters = 2



```python
# Plot the clustered data
fig,ax = plt.subplots(1, figsize=(12, 10), sharex=True, sharey=True)
fig.suptitle("Visualization of Clustered Data", fontsize=16)

ax.scatter(features[kmeans.labels_ == 0, 2],
           features[kmeans.labels_ == 0, 1],c='green', label='cluster 1')
ax.scatter(features[kmeans.labels_ == 1, 2],
           features[kmeans.labels_ == 1, 1],c='blue', label='cluster 2')

ax.scatter(centroids[:, 2], centroids[:, 1], marker='*', s=300,c='r', label='centroid')
ax.legend()
ax.set_title(f"Silhouette: {kmeans_silhouette}", size=14,color='red')
ax.set_xlabel('Ca_%',size=14, color='red')
ax.set_ylabel('Al_%',size=14, color='red')
```

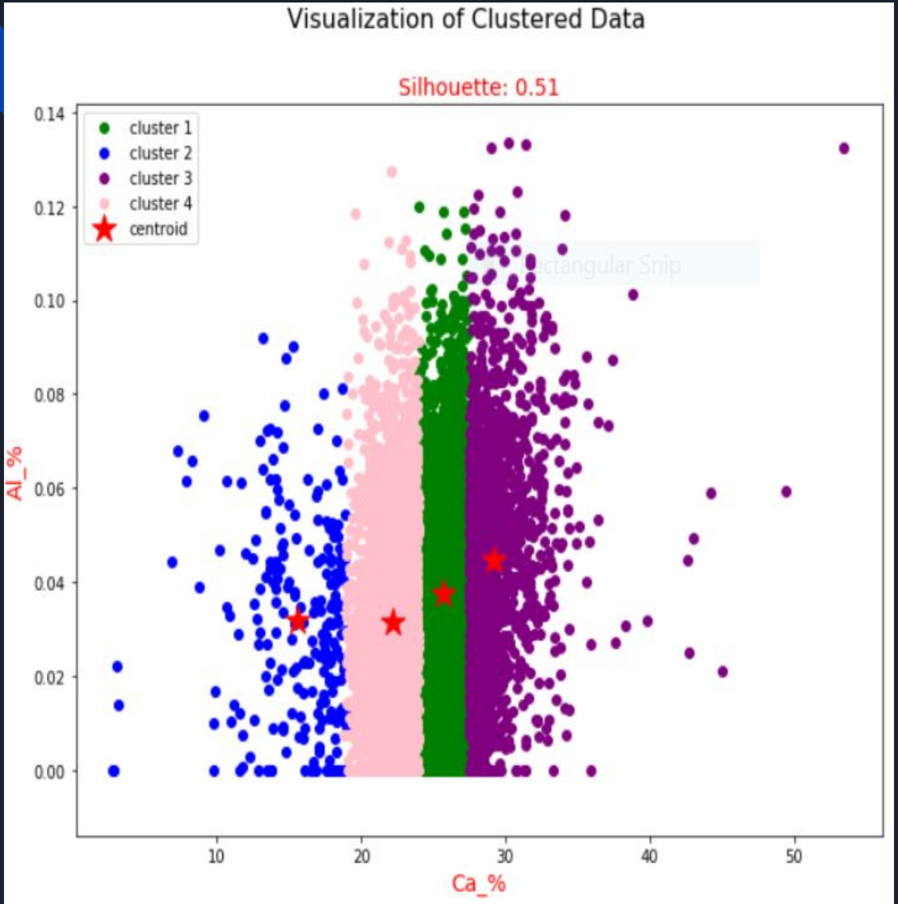


```
In [10]: cols_name=list(data.columns)[2:8]
         print("\nSUM - Min - Max of each column in Clustered data\n")
         for i in range(6):
             print(cols_name[i], "\tSum:",round(features[kmeans.labels_ == 0,i].sum(),2),
                   "\t Min:", round(features[kmeans.labels_ == 0,i].min(),2),
                   "\t Max:", round(features[kmeans.labels_ == 0,i].max(),2))
```

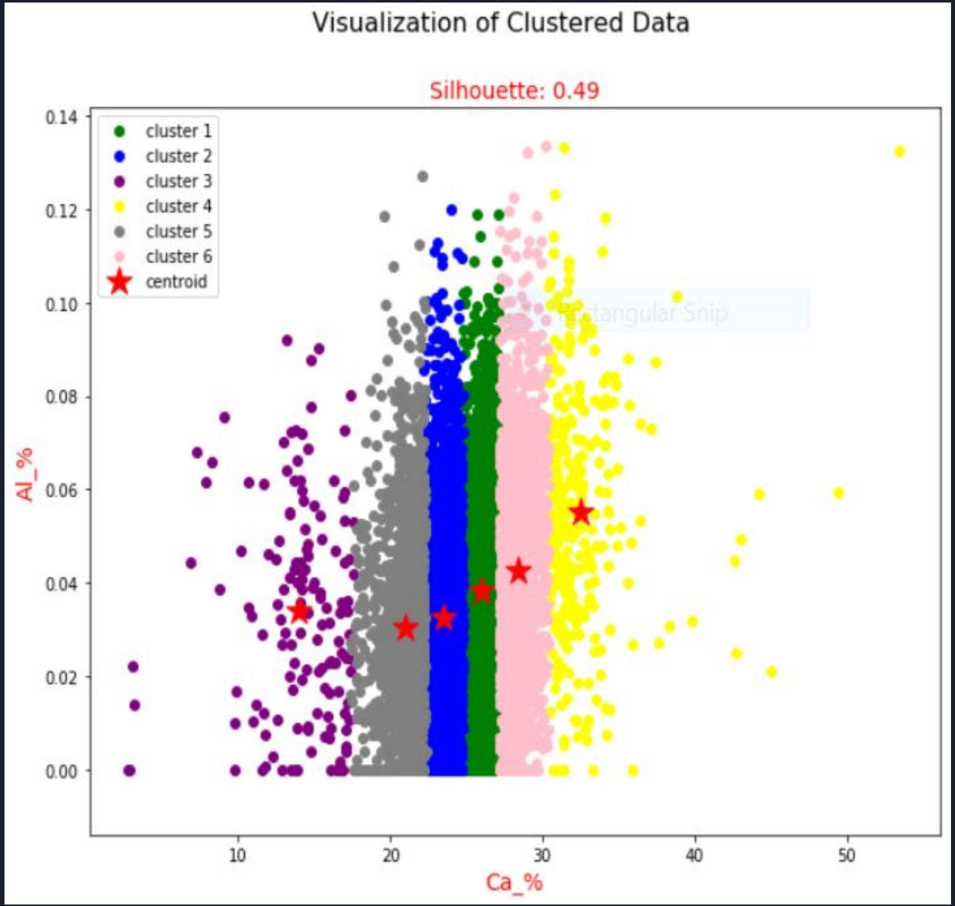
```
SUM - Min - Max of each column in Clustered data

Mg_%    Sum: 1739.34     Min: 0.04     Max: 1.12
Al_%    Sum: 171.71      Min: 0.0      Max: 0.13
Ca_%    Sum: 114216.53   Min: 24.78    Max: 53.43
Ti_%    Sum: 51.1        Min: 0.0      Max: 0.39
Fe_%    Sum: 787.53      Min: 0.0      Max: 1.49
Si_%    Sum: 10.93       Min: 0.0      Max: 0.05
```

```
In [11]: print("\nSUM - Min - Max of each column in Raw data\n")
         for i in range(6):
             print(cols_name[i], "\tSum:",round(features[:,i].sum(),2),
                   "\t Min:", round(features[:,i].min(),2),"\t Max:", round(features[:,i].max(),2))
```

```
SUM - Min - Max of each column in Raw data

Mg_%    Sum: 3095.58     Min: 0.0      Max: 1.12
Al_%    Sum: 295.38      Min: 0.0      Max: 0.13
Ca_%    Sum: 200669.84   Min: 2.78     Max: 53.43
Ti_%    Sum: 366.6       Min: 0.0      Max: 1.47
Fe_%    Sum: 4014.11     Min: 0.0      Max: 5.3
Si_%    Sum: 68.81       Min: 0.0      Max: 0.45
```



Visualization of Clustered Data
Silhouette: 0.51
cluster 1
cluster 2
cluster 3
cluster 4
centroid
Al_%
Ca_%
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
10
20
30
40
50



Visualization of Clustered Data
Silhouette: 0.49
cluster 1
cluster 2
cluster 3
cluster 4
cluster 5
cluster 6
centroid
Al_%
Ca_%
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
10
20
30
40
50

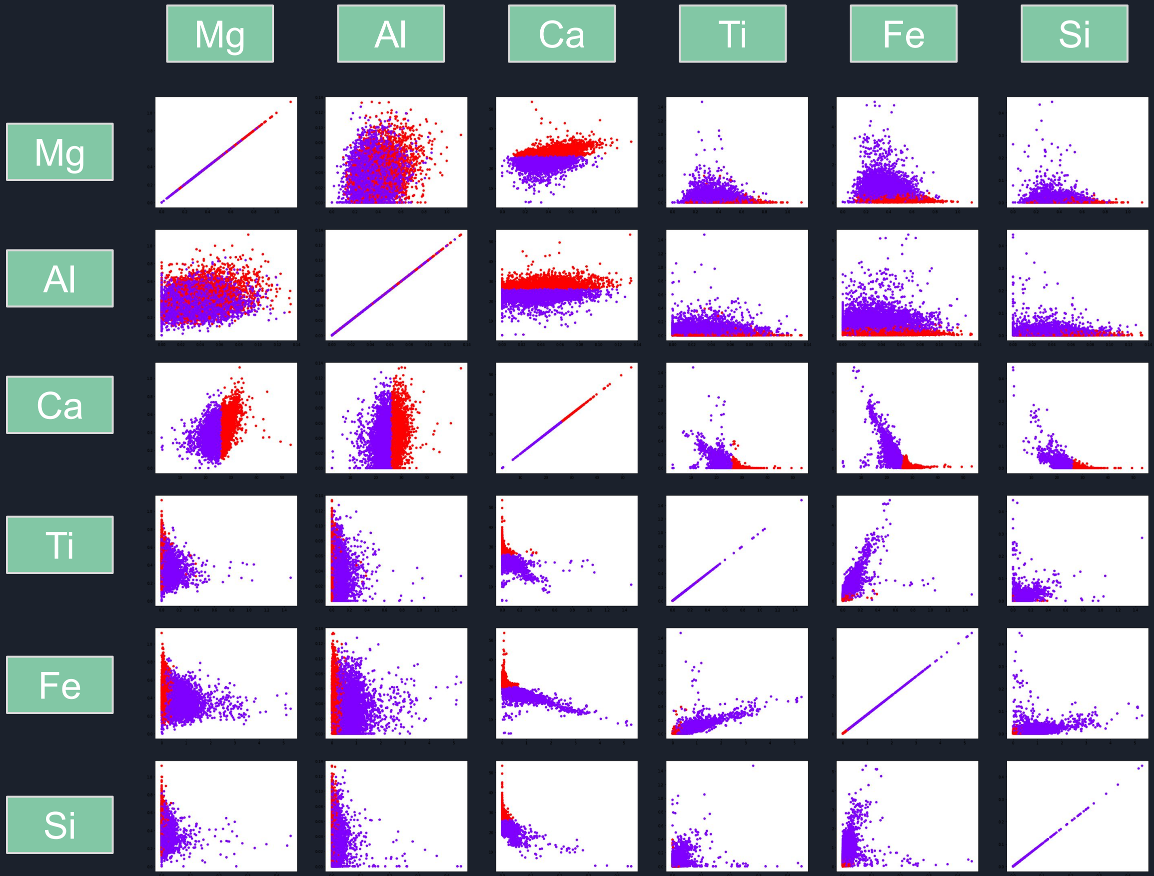
Hierarchical Clustering

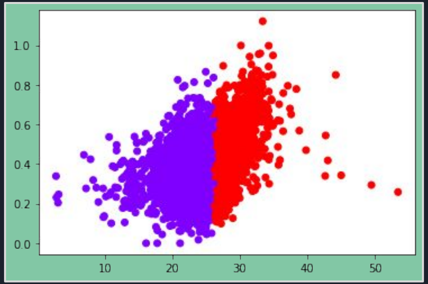
- Also known as hierarchical cluster analysis
- An algorithm that groups similar objects into groups called clusters.
- The endpoint is a set of clusters, and the objects within each cluster are broadly similar to each other.
- It identify the two clusters that are closest together and merge the two most similar clusters.
- There are two hierarchical clustering
    - Agglomerative clustering
    - Divisive clustering

Agglomerative Clustering

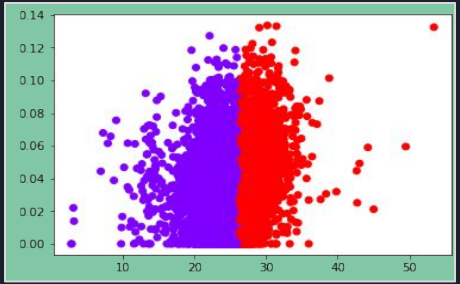
```
In [5]:  data_array = np.array(dataset)
         data_array.shape

Out[5]:  (8055, 6)

In [10]: #Hierarchical Clustering
         cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
         cluster.fit_predict(data_array)

Out[10]: array([0, 0, 0, ..., 0, 0, 0], dtype=int32)

In [11]: print(cluster.labels_)

         [0 0 0 ... 0 0 0]

In [12]: fig, ax = plt.subplots(6,6)
         for i in range(6):
             for j in range(6):
                 ax[i,j].scatter(data_array[:,i],data_array[:,j], c=cluster.labels_, cmap='rainbow')
         fig.set_size_inches(50,40)
```

Clusters = 2

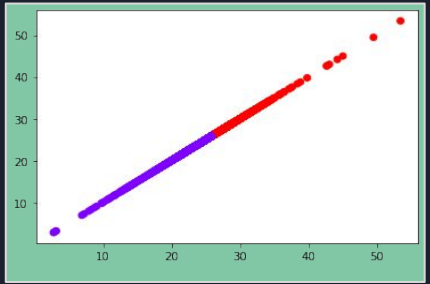




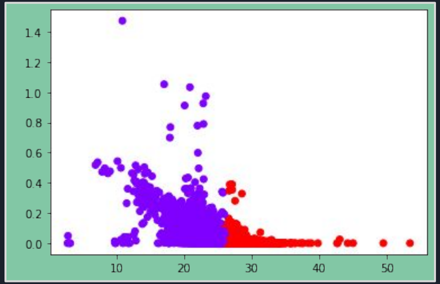
Ca vs. Mg



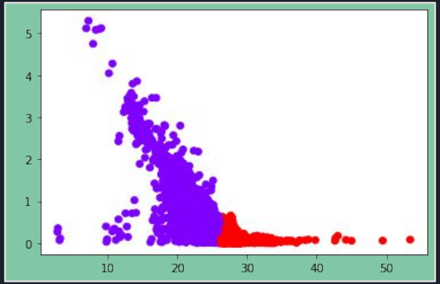
Ca vs. Al



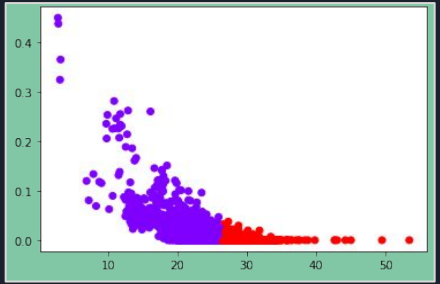
Ca vs. Ca



Ca vs. Ti



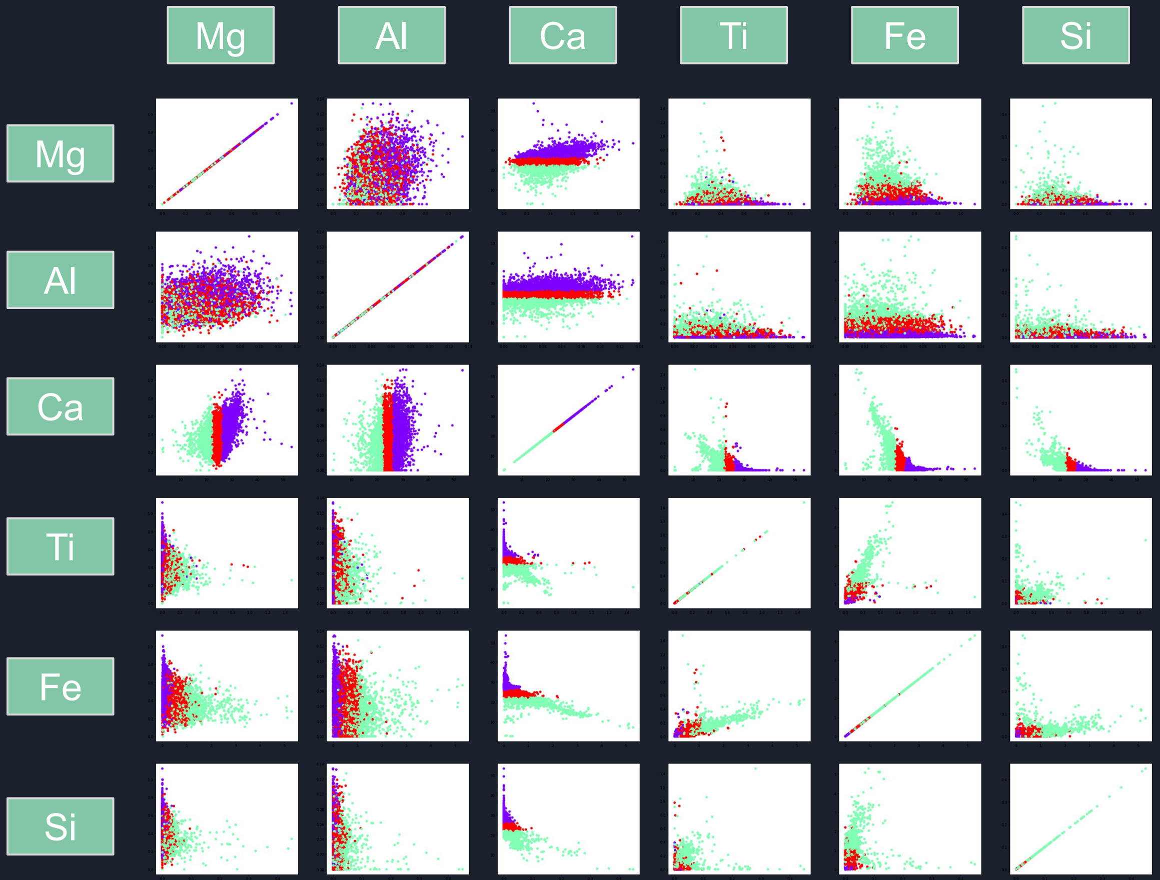
Ca vs. Fe



Ca vs. Si

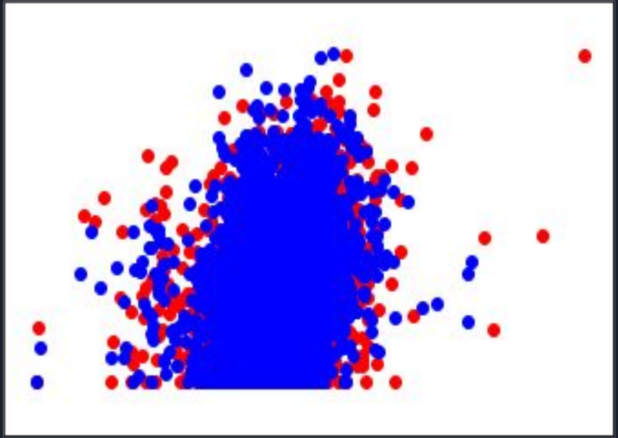
```
In [13]: data_array[cluster.labels_ == 0, 2].max()

Out[13]: 26.1842
```

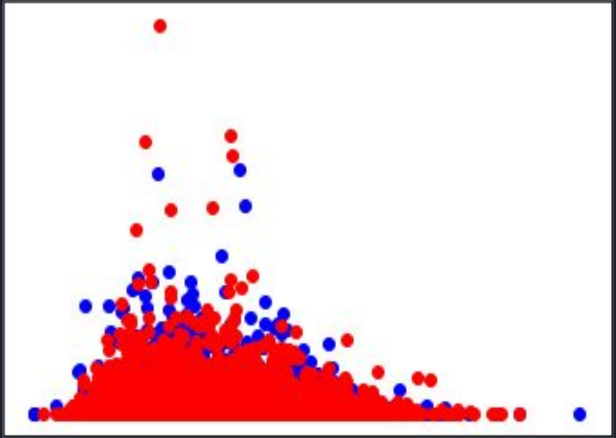
Clusters = 3

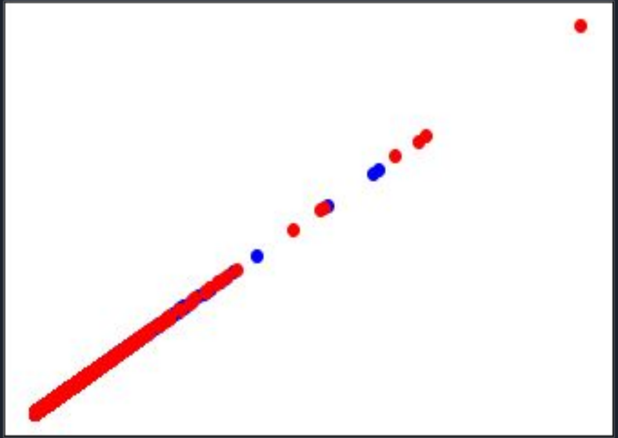


Mars_Data

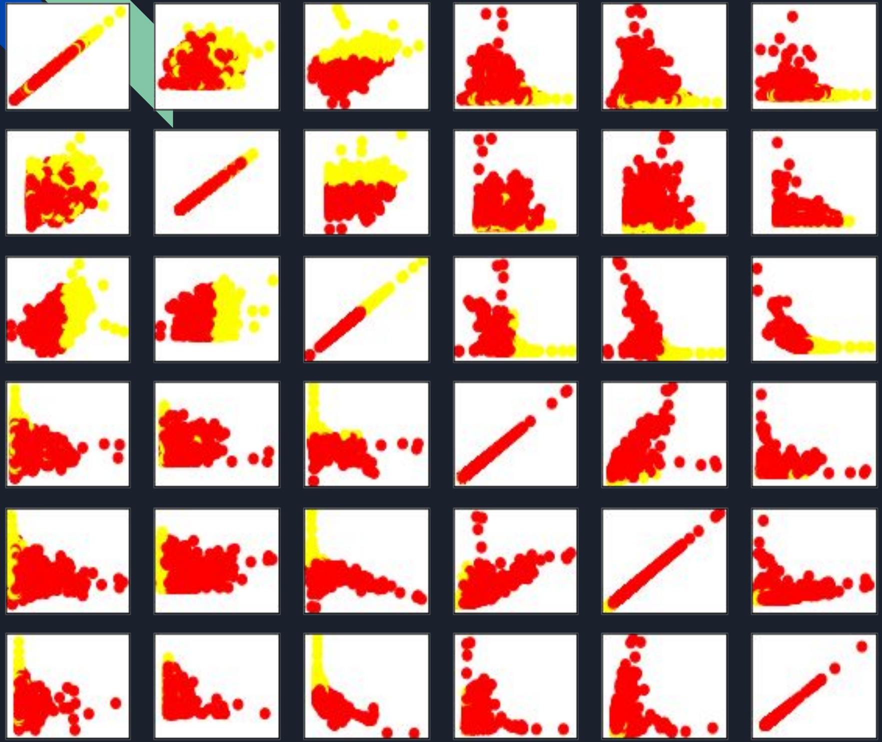
PMC	Detector	Mg_%	Al_%	Ca_%	Ti_%	Fe_%	Si_%	Mg_int	Al_int	Ca_int	Ti_int	Fe_int	Si_int	image_i	image_j
7	A	0.4605	0.0305	22.9336	0.114	1.0066	0.0067	31.9	8	50624.2	152.2	2015.6	5	409.04	416.18
7	B	0.1968	0.0735	22.4898	0.1392	1.1196	0.0196	13.8	19.8	50480.9	188.8	2270.2	14.9	409.04	416.18



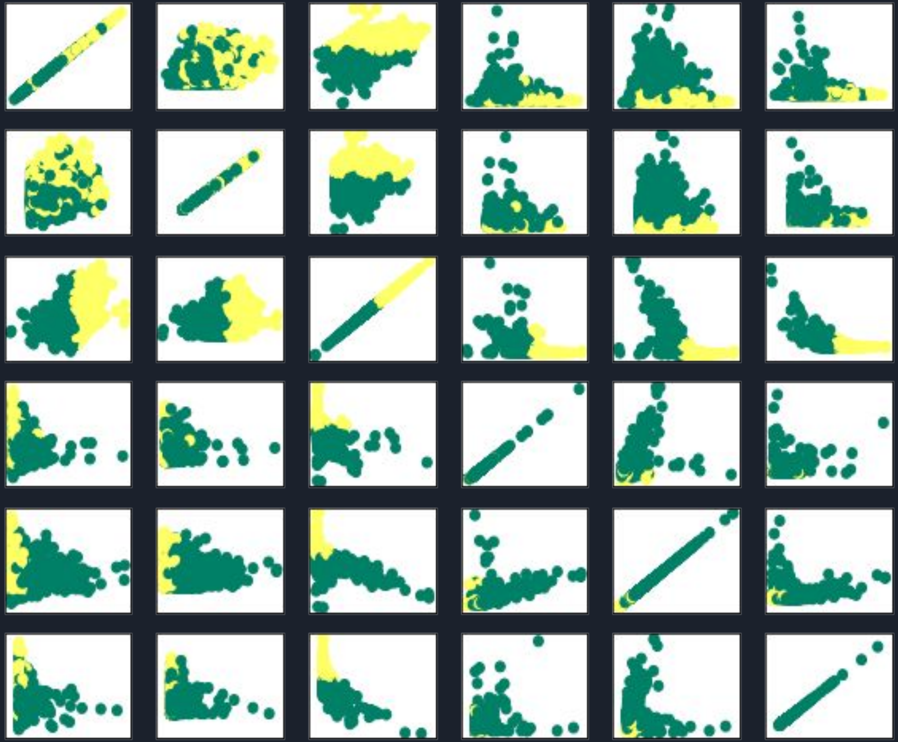




Detectors



A



B

```python
DF = pd.read_csv('Mars_Rover_Mini_Project_data.csv')
x = DF[['Mg_%', 'Al_%', 'Ca_%', 'Ti_%', 'Fe_%', 'Si_%']]
print(x)
plot_corr(x,size=6)
```

	Mg_%	Al_%	Ca_%	Ti_%	Fe_%	Si_%
0	0.4605	0.0305	22.9336	0.1140	1.0066	0.0067
1	0.1968	0.0735	22.4898	0.1392	1.1196	0.0196
2	0.5142	0.0000	22.6415	0.1949	1.0006	0.0130
3	0.4354	0.0271	21.9504	0.2441	1.0346	0.0150
4	0.3532	0.0570	26.5924	0.1641	0.3400	0.0179
...	...	...	...	...	...	...
8050	0.2774	0.0165	20.7999	0.0357	0.6332	0.0000
8051	0.5039	0.0297	22.1842	0.0652	0.7391	0.0070
8052	0.3707	0.0599	22.2514	0.0329	0.9844	0.0326
8053	0.2642	0.0576	21.6834	0.0768	0.9042	0.0106
8054	0.2766	0.0098	19.9610	0.0847	1.1792	0.0071

[8055 rows x 6 columns]

Dimensionality Reduction

Dimensionality reduction, or dimension reduction, is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension.

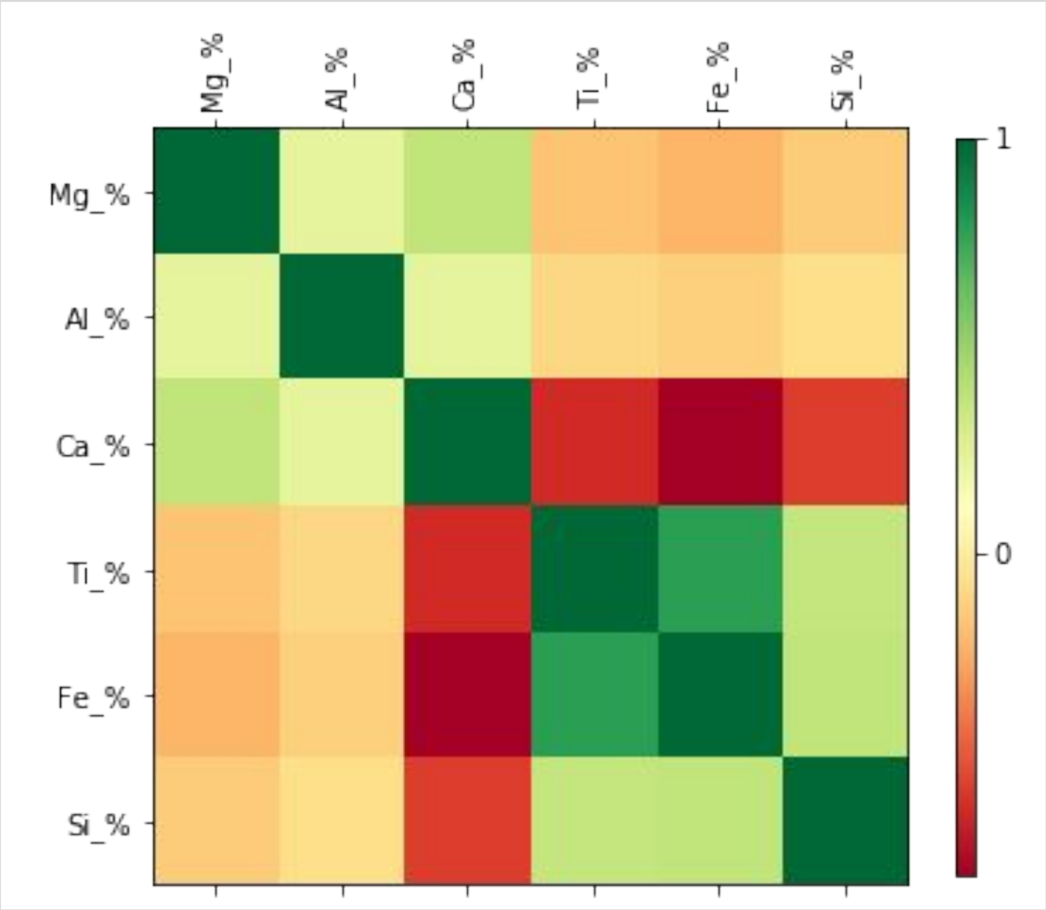
Source: https://en.wikipedia.org/wiki/Dimensionality_reduction

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a **linear dimensionality reduction** technique that can be utilized for extracting information from a high-dimensional space by projecting it into a lower-dimensional subspace. It tries to preserve the essential parts that have more variation of the data and remove the non-essential parts with fewer variation.

Source:

https://towardsdatascience.com/principal-component-analysis-for-dimensionality-reduction-115a3d157bad


Mg_%
Al_%
Ca_%
Ti_%
Fe_%
Si_%
1
0

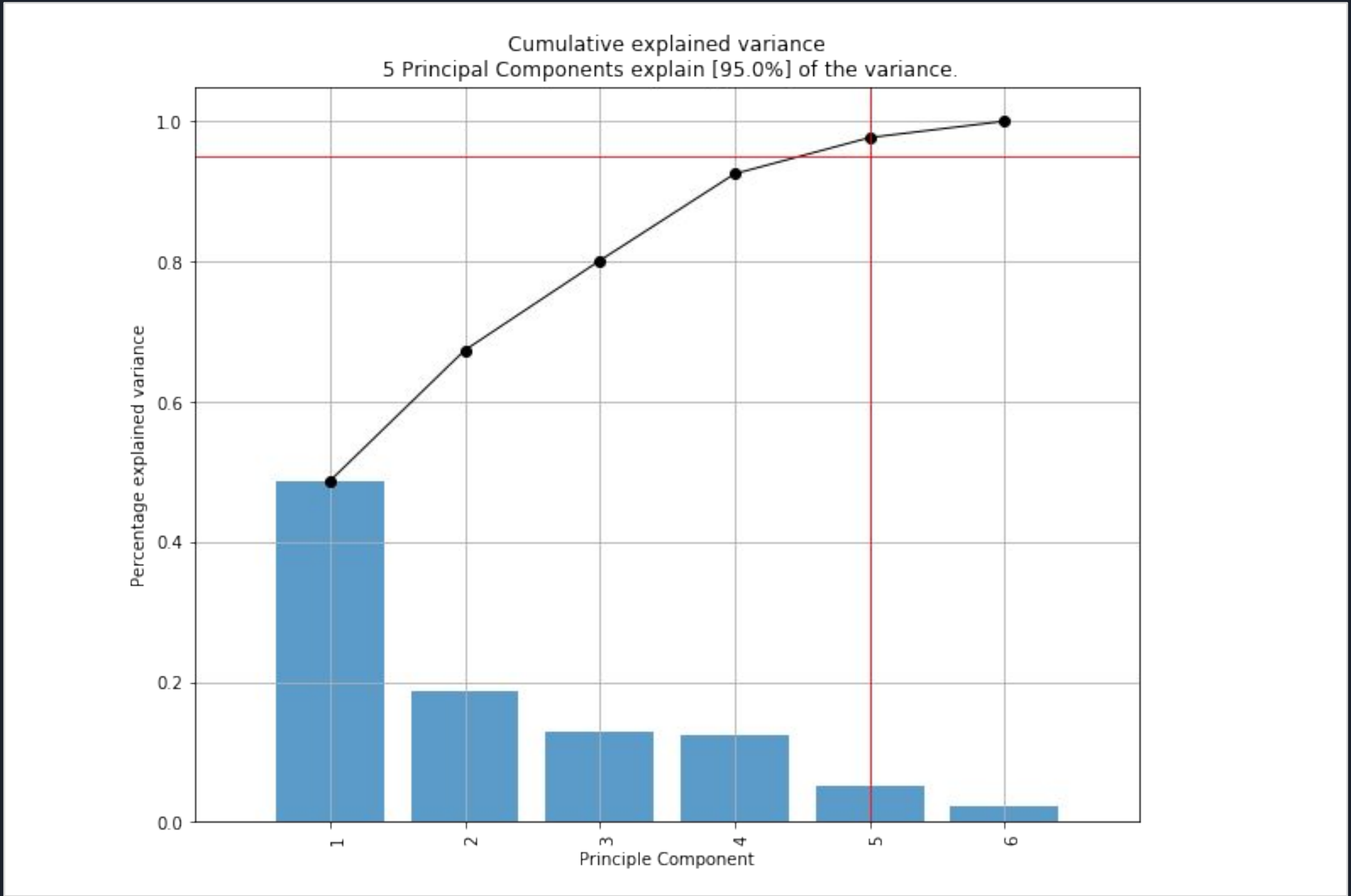
In [5]:

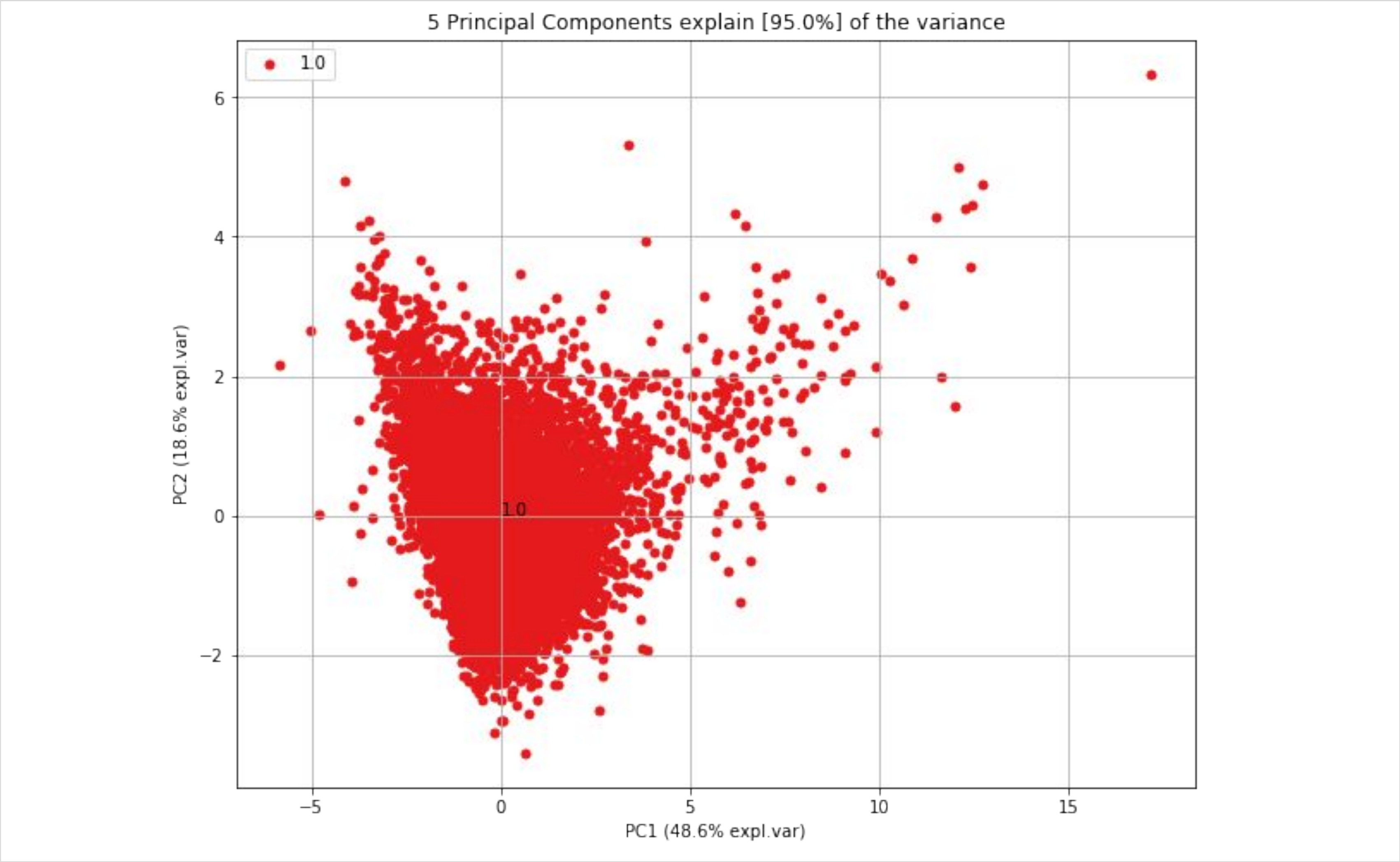
```python
from sklearn.decomposition import PCA
pca = PCA()
#pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents)

print(principalDf)
```

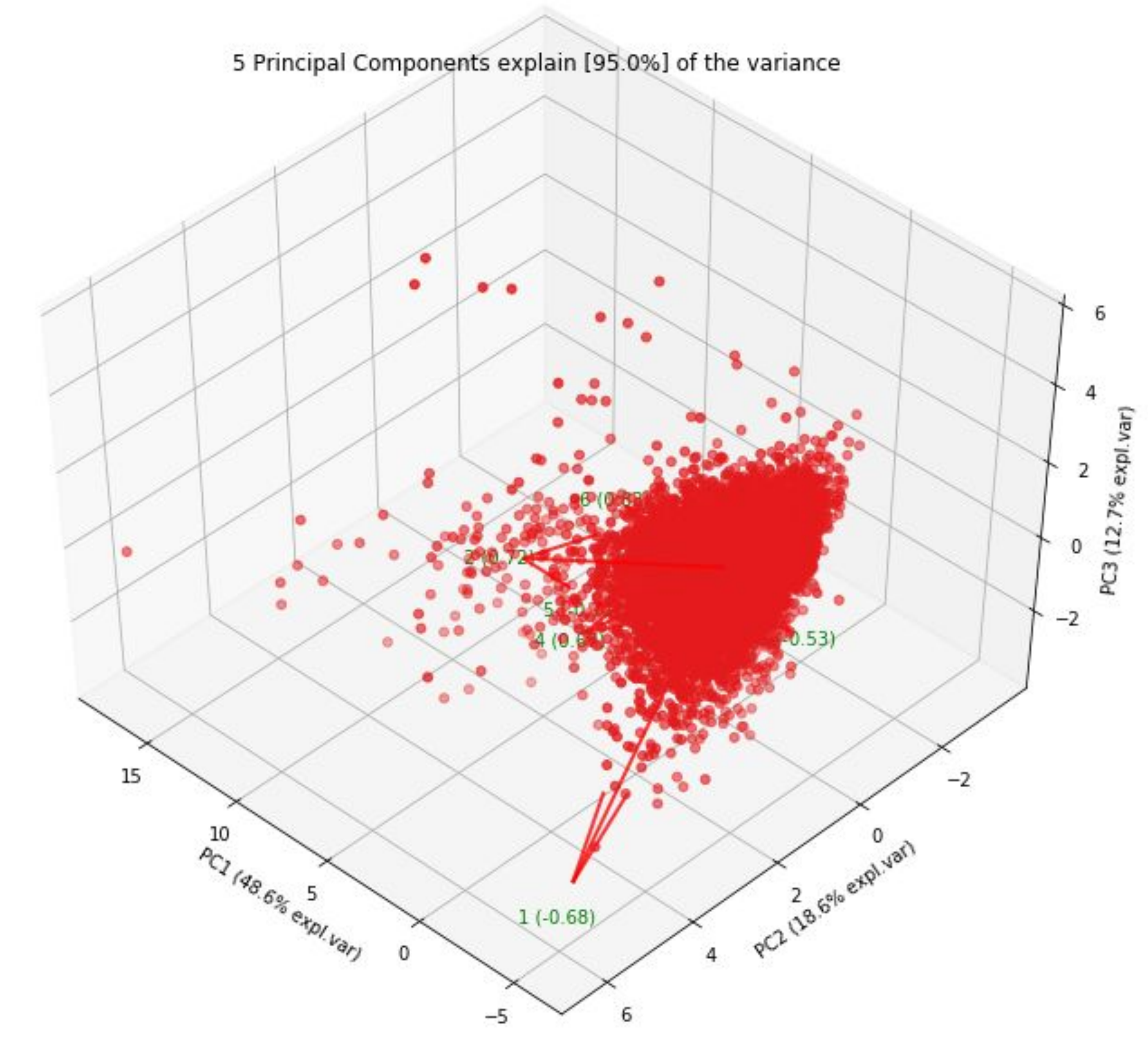
```
             0         1         2         3         4         5
0     1.162403  0.573699 -0.944241 -0.438817 -0.094415  0.008564
1     1.983244  0.881529  1.713346 -1.021524  0.422812 -0.197341
2     1.978455  0.211909 -2.229534 -0.122304  0.726635  0.365481
3     2.462980  0.869971 -1.188093 -0.690295  1.114537  0.646480
4     0.477140  0.917790  0.494050 -0.327358  1.631186  0.427602
...        ...       ...       ...       ...       ...       ...
8050  0.908525 -1.215567 -0.010230 -0.363731 -0.798500  0.421284
8051  0.599055  0.485408 -0.947020  0.065388 -0.609519  0.315492
8052  1.157216  0.983152  0.933572  0.594519 -0.620962 -0.485513
8053  1.271200  0.334120  1.079959 -0.690663 -0.366959  0.057080
8054  2.119894 -0.996864 -0.413484 -0.529650 -0.669290 -0.061324

[8055 rows x 6 columns]
```

pca.explained_variance_ratio_

```
array(  [0.48655327,
         0.18663207,
         0.1278593 ,
         0.12383701,
         0.05166757,
          0.02345078])
```


Cumulative explained variance
5 Principal Components explain [95.0%] of the variance.
Percentage explained variance
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
6
Principle Component


5 Principal Components explain [95.0%] of the variance
1.0
PC2 (18.6% expl.var)
PC1 (48.6% expl.var)
6
4
2
0
−2
−5
0
5
10
15
1.0



5 Principal Components explain [95.0%] of the variance
PC1 (48.6% expl.var)
PC2 (18.6% expl.var)
PC3 (12.7% expl.var)
15
10
5
0
−5
−2
0
2
4
6
6
4
2
0
−2
1 (-0.68)
-0.53)

	PC	feature	loading	type
0	PC1	3	-0.533718	best
1	PC2	2	0.719965	best
2	PC3	1	-0.682664	best
3	PC4	6	0.829943	best
4	PC5	4	0.674966	best
5	PC6	5	-0.710862	best

Thank You