## Suggestion/Homework:

To help improve your programming skills and your depth of understanding,

1. Implement Lloyd's algorithm from scratch
2. Implment KMeans++ from scratch

## Retail Customer Segmentation: Acquire Data

```
# https://drive.google.com/file/d/1lEccW5Y5_2z00VRtLGOAJOAU6YA9fl6W/view?usp=sharin
id = "1lEccW5Y5_2z00VRtLGOAJOAU6YA9fl6W"
print("https://drive.google.com/uc?export=download&id=" + id)
```

```
https://drive.google.com/uc?export=download&id=1lEccW5Y5_2z00VRtLGOAJOAU6YA9fl
```

```
!wget "https://drive.google.com/uc?export=download&id=1lEccW5Y5_2z00VRtLGOAJOAU6YA9
```

```
--2022-06-01 13:27:48--  https://drive.google.com/uc?export=download&id=1lEccW
Resolving drive.google.com (drive.google.com)... 172.253.62.100, 172.253.62.10
Connecting to drive.google.com (drive.google.com)|172.253.62.100|:443... conne
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-10-64-docs.googleusercontent.com/docs/securesc/ha0ro937g
Warning: wildcards not supported in HTTP.
--2022-06-01 13:27:48--  https://doc-10-64-docs.googleusercontent.com/docs/sec
Resolving doc-10-64-docs.googleusercontent.com (doc-10-64-docs.googleuserconte
Connecting to doc-10-64-docs.googleusercontent.com (doc-10-64-docs.googleuserc
HTTP request sent, awaiting response... 200 OK
Length: 139827 (137K) [text/csv]
Saving to: ‘E-commerce.csv’

E-commerce.csv      100%[===================>] 136.55K  --.-KB/s    in 0.001s

2022-06-01 13:27:48 (99.9 MB/s) - ‘E-commerce.csv’ saved [139827/139827]
```

```
!head ./E-commerce.csv
```

```
ID,n_clicks,n_visits,amount_spent,amount_discount,days_since_registration,prof
1476,130,65,213.90583071577163,31.600750627904915,233,235
1535,543,46,639.2230037736391,5.6891747173479414,228,170
1807,520,102,1157.4027626541078,844.3216058194998,247,409
1727,702,83,1195.903633609631,850.0417570033645,148,200
1324,221,84,180.75461615086704,64.2833000293408,243,259
1793,971,167,1700.9096451005262,1257.4171176204811,205,229
646,345,77,1314.029384121076,12.095727427593667,230,217
416,222,61,3869.409085656883,117.49933081401785,257,296
232,451,74,2598.1462935566806,103.64066379042382,65,102
```

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
```

```
df = pd.read_csv('./E-commerce.csv')
```

```
df.head()
```

|  | ID | n_clicks | n_visits | amount_spent | amount_discount | days_since_registra |
|---|---|---|---|---|---|---|
| **0** | 1476 | 130 | 65 | 213.905831 | 31.600751 |  |
| **1** | 1535 | 543 | 46 | 639.223004 | 5.689175 |  |
| **2** | 1807 | 520 | 102 | 1157.402763 | 844.321606 |  |
| **3** | 1727 | 702 | 83 | 1195.903634 | 850.041757 |  |
| **4** | 1324 | 221 | 84 | 180.754616 | 64.283300 |  |

```
## We do not need 5 decimal places, so we round it to 2 places instead.
```

```
df['amount_spent'].round(decimals=2)
df['amount_discount'].round(decimals=2)
```

```
0         31.60
1          5.69
2        844.32
3        850.04
4         64.28
         ...
2495     373.41
2496     122.64
2497       0.00
2498      78.13
2499    1065.42
Name: amount_discount, Length: 2500, dtype: float64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   ID                      2500 non-null   int64
 1   n_clicks                2500 non-null   int64
 2   n_visits                2500 non-null   int64
 3   amount_spent            2500 non-null   float64
 4   amount_discount         2500 non-null   float64
 5   days_since_registration  2500 non-null   int64
 6   profile_information     2500 non-null   int64
dtypes: float64(2), int64(5)
memory usage: 136.8 KB
```

## Cleaning + Preprocessing

```
X=df.drop("ID",axis=1)
X
```

| | n_clicks | n_visits | amount_spent | amount_discount | days_since_registrati |
|---|---|---|---|---|---|
| **0** | 130 | 65 | 213.905831 | 31.600751 | 2 |
| **1** | 543 | 46 | 639.223004 | 5.689175 | 2 |
| **2** | 520 | 102 | 1157.402763 | 844.321606 | 2 |
| **3** | 702 | 83 | 1195.903634 | 850.041757 | 1 |
| **4** | 221 | 84 | 180.754616 | 64.283300 | 2 |
| **...** | ... | ... | ... | ... | |
| **2495** | 804 | 120 | 502.643798 | 373.413462 | 3 |
| **2496** | 482 | 60 | 530.014805 | 122.639755 | 1 |
| **2497** | 375 | 111 | 0.000000 | 0.000000 | |
| **2498** | 271 | 32 | 3190.499018 | 78.133067 | 1 |
| **2499** | 814 | 123 | 1394.589041 | 1065.415902 | 2 |

2500 rows × 6 columns

```
X.describe()
```

| | n_clicks | n_visits | amount_spent | amount_discount | days_since_registr |
|---|---|---|---|---|---|
| **count** | 2500.00000 | 2500.000000 | 2500.000000 | 2500.000000 | 2500.0 |
| **mean** | 408.68000 | 94.475600 | 1445.090745 | 388.508637 | 200.9 |
| **std** | 186.41409 | 38.866356 | 1167.663473 | 487.143968 | 99.1 |
| **min** | 50.00000 | 10.000000 | 0.000000 | 0.000000 | 0.0 |
| **25%** | 274.75000 | 67.000000 | 609.618538 | 56.298615 | 130.0 |
| **50%** | 378.00000 | 92.000000 | 1036.189112 | 137.454623 | 200.0 |
| **75%** | 522.00000 | 119.000000 | 1949.270949 | 679.540536 | 268.0 |
| **max** | 1246.00000 | 259.000000 | 6567.402267 | 2428.406527 | 514.0 |

```
# Do we need scaling at all?
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
scaler.fit(X)
```

```
X
```

```
array([[-1.49525046, -0.75853514, -1.05461141, -0.73280039,  0.32311781,
         0.33919174],
       [ 0.72069055, -1.24748762, -0.69029218, -0.78600183,  0.27267227,
        -0.31003449],
       [ 0.59728463,  0.19363547, -0.24642848,  0.93587151,  0.46436533,
         2.07712043],
       ...,
       [-0.18070918,  0.42524454, -1.2378394 , -0.7976828 , -1.18015931,
         2.52658475],
       [-0.73871854, -1.60776839,  1.49508613, -0.63726061, -0.90775339,
         0.13942982],
       [ 2.17473416,  0.73405663, -0.04325887,  1.38982053,  0.47445444,
        -0.51978451]])
```

```
from sklearn.cluster import KMeans

k = 4 ## arbitrary value
kmeans = KMeans(n_clusters=k)
y_pred = kmeans.fit_predict(X)
```

```
## what are learned labels(cluster #)
y_pred
```

```
array([3, 3, 0, ..., 1, 2, 0], dtype=int32)
```

```
##coordinates of the cluster centers
kmeans.cluster_centers_
```

```
array([[ 1.0387607 ,  0.87726845, -0.0308178 ,  1.51719781, -0.01890781,
        -0.02259216],
       [-0.10975713, -0.22690132, -0.62750954, -0.4811005 , -0.79076605,
        -0.0765108 ],
       [-0.84848342, -0.69485355,  1.62272465, -0.62820588, -0.03079898,
         0.13822599],
       [-0.18505778, -0.02992418, -0.6241805 , -0.42574709,  0.89260056,
        -0.00923737]])
```

```
y_pred is kmeans.labels_
```

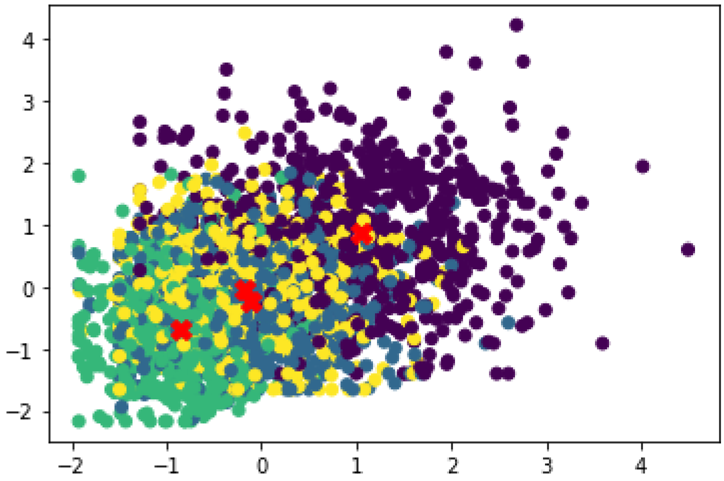
```
True
```

```
#Visualize clusters
clusters = pd.DataFrame(X, columns=df.drop("ID",axis=1).columns)
clusters['label'] = kmeans.labels_
clusters
```

| | n_clicks | n_visits | amount_spent | amount_discount | days_since_registrati |
|---|---|---|---|---|---|
| **0** | -1.495250 | -0.758535 | -1.054611 | -0.732800 | 0.3231 |
| **1** | 0.720691 | -1.247488 | -0.690292 | -0.786002 | 0.2726 |
| **2** | 0.597285 | 0.193635 | -0.246428 | 0.935872 | 0.4643 |
| **3** | 1.573801 | -0.295317 | -0.213449 | 0.947616 | -0.5344 |
| **4** | -1.006992 | -0.269583 | -1.083008 | -0.665697 | 0.4240 |
| **...** | ... | ... | ... | ... | |
| **2495** | 2.121079 | 0.656854 | -0.807284 | -0.030993 | 1.6347 |
| **2496** | 0.393397 | -0.887207 | -0.783838 | -0.545880 | -0.8976 |
| **2497** | -0.180709 | 0.425245 | -1.237839 | -0.797683 | -1.1801 |
| **2498** | -0.738719 | -1.607768 | 1.495086 | -0.637261 | -0.9077 |

```
def viz_clusters(kmeans):
    plt.scatter(clusters['n_clicks'], clusters['n_visits'], c=clusters['label'])
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
                color="red",
                marker="X",
                s=100)

viz_clusters(kmeans)
```



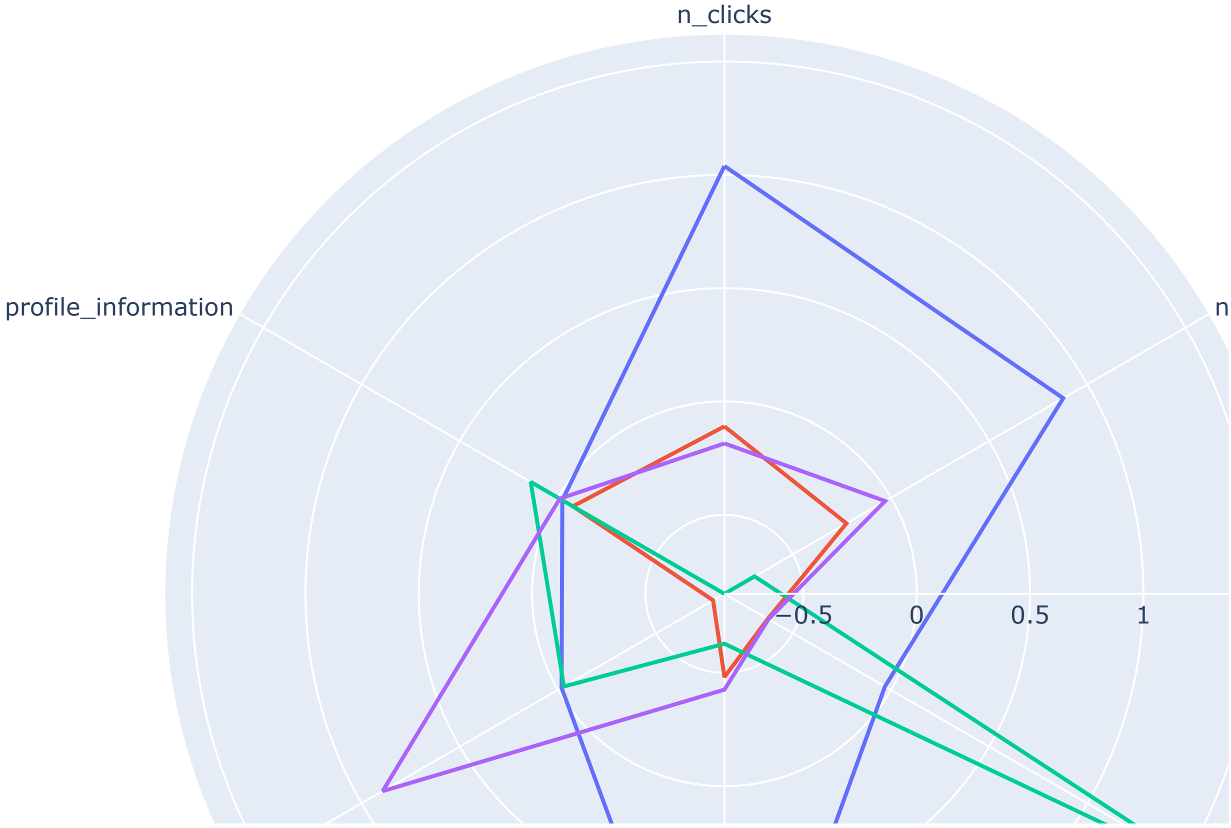
```
#polar plot

polar = clusters.groupby("label").mean().reset_index()
polar = pd.melt(polar, id_vars=["label"])
polar
```

| | label | variable | value |
|---|---|---|---|
| **0** | 0 | n_clicks | 1.038761 |
| **1** | 1 | n_clicks | -0.109757 |
| **2** | 2 | n_clicks | -0.848483 |
| **3** | 3 | n_clicks | -0.185058 |
| **4** | 0 | n_visits | 0.877268 |
| **5** | 1 | n_visits | -0.226901 |
| **6** | 2 | n_visits | -0.694854 |
| **7** | 3 | n_visits | -0.029924 |
| **8** | 0 | amount_spent | -0.030818 |
| **9** | 1 | amount_spent | -0.627510 |
| **10** | 2 | amount_spent | 1.622725 |
| **11** | 3 | amount_spent | -0.624181 |
| **12** | 0 | amount_discount | 1.517198 |
| **13** | 1 | amount_discount | -0.481101 |
| **14** | 2 | amount_discount | -0.628206 |
| **15** | 3 | amount_discount | -0.425747 |
| **16** | 0 | days_since_registration | -0.018908 |
| **17** | 1 | days_since_registration | -0.790766 |
| **18** | 2 | days_since_registration | -0.030799 |
| **19** | 3 | days_since_registration | 0.892601 |
| **20** | 0 | profile_information | -0.022592 |

```
import plotly.express as px

fig = px.line_polar(polar, r="value", theta="variable", color="label", line_close=T
fig.show()
```



Insights

1. The plot is read and interpreted radially - values increase as we move away from the center showing the influence of a feature on that label.
2. Many overlapping lines - green(2) & red(1) and blue and overlap on all the features except one. Looking at this plot, we have different customer segments:
3. Bargain shoppers(label 0) - people who buy heavily discounted items. Action: show them more discounted items.
4. Inactive old users (label 1) - people who have been a long time user of the app but have not shown much interest now. Action: start sending notifications, emails, etc - get them back on the platform.
5. New and inactive users - Users who have recently joined but haven't bought much and are not that actively looking for items. Somewhat similar to label 1.
6. Premium shopper (label 3) - Heavy spenders who like to buy items. Action: Show them more quality

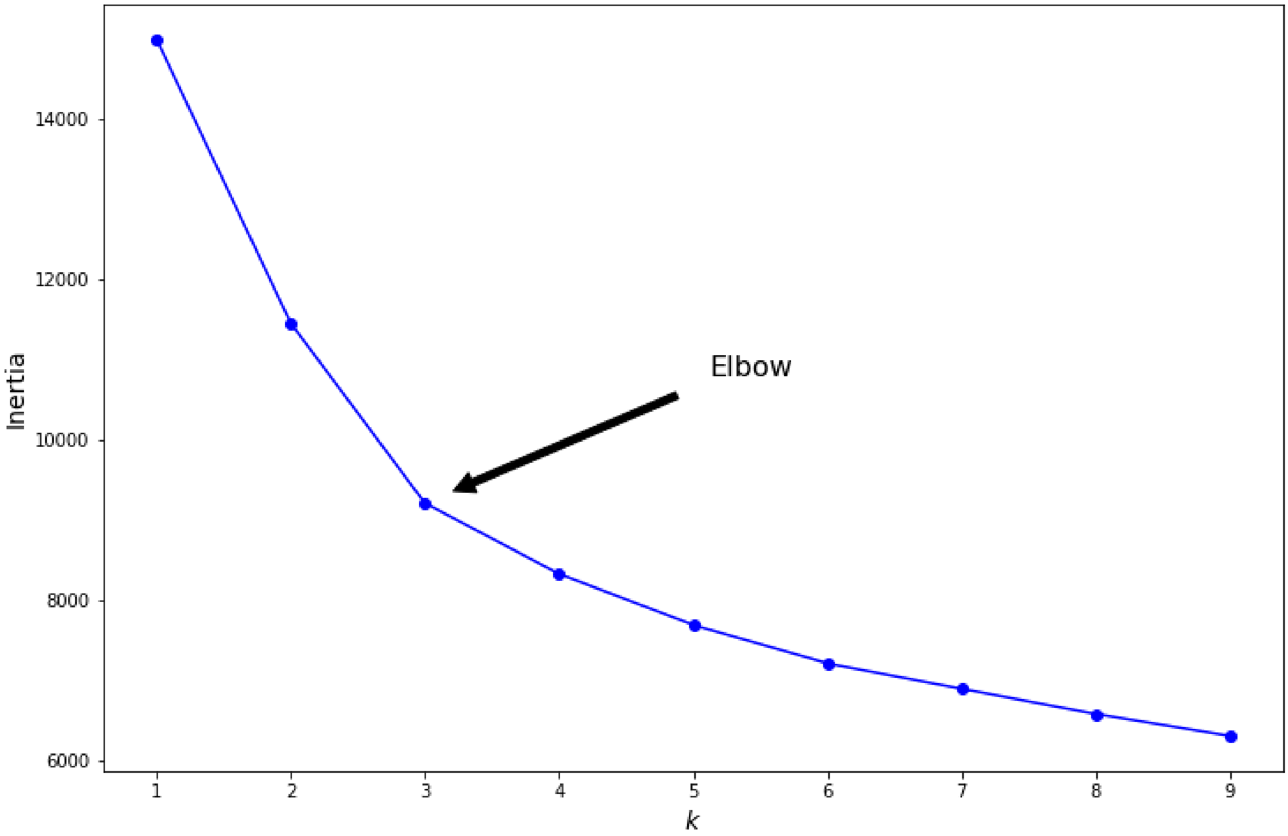
## Finding the best K

```
# Inertia = Within Cluster Sum of Squares
kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(X)
```

```
                for k in range(1, 10)]

inertias = [model.inertia_ for model in kmeans_per_k]


plt.figure(figsize=(12, 8))
plt.plot(range(1, 10), inertias, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inertia", fontsize=14)
plt.annotate('Elbow',
             xy=(3, inertias[2]),
             xytext=(0.55, 0.55),
             textcoords='figure fraction',
             fontsize=16,
             arrowprops=dict(facecolor='black', shrink=0.1)
            )
plt.show()
```



## Silhouette score

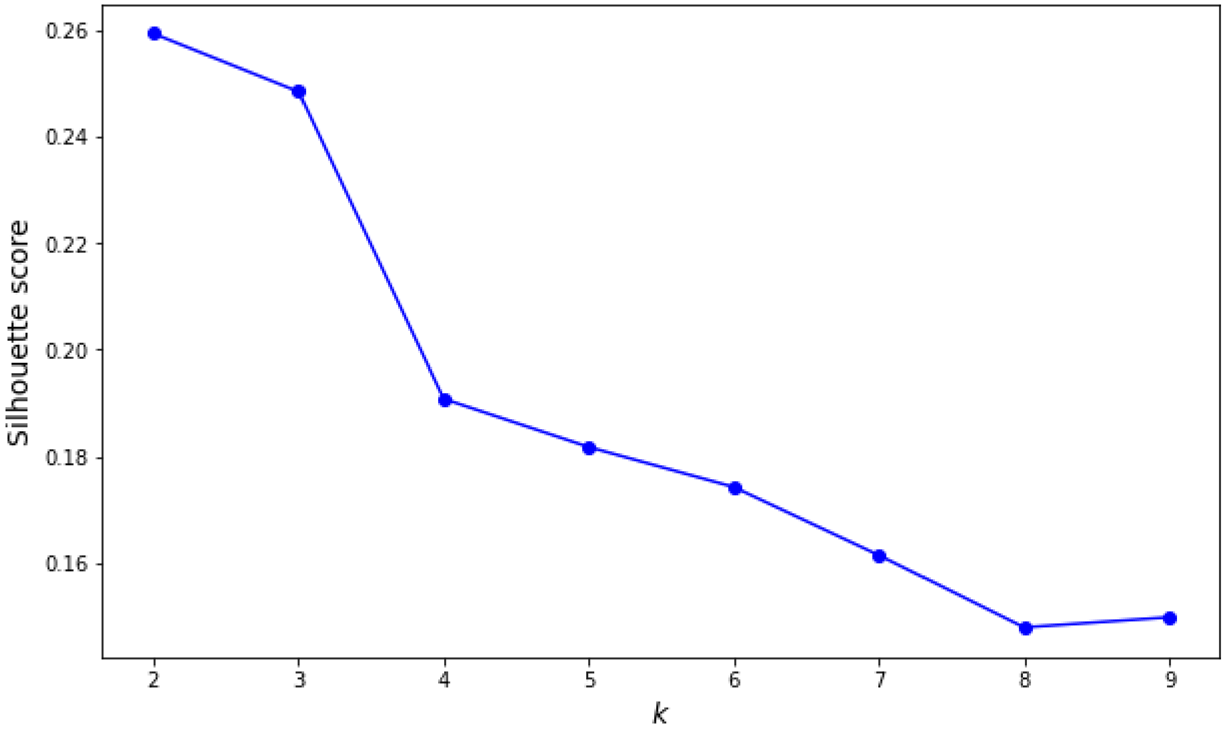
1. Silhouette Coefficient of X_i = (b – a) / max(a, b)
2. Diagram: https://www.researchgate.net/figure/Derivation-of-the-Overall-Silhouette-Coefficient-OverallSil_fig1_221570710
3. Range: Worst (-1) to Best(+1)

4. Average Silhouette Coefficients of all points.

```
from sklearn.metrics import silhouette_score

silhouette_scores = [silhouette_score(X, model.labels_)
                     for model in kmeans_per_k[1:]]


plt.figure(figsize=(10, 6))
plt.plot(range(2, 10), silhouette_scores, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Silhouette score", fontsize=14)
plt.show()
```



✓ 0s completed at 19:28 ● ×