# Project Description

Pada proyek ini, saya akan menganalisis data Bank Customer untuk memahami pola dan perilaku pelanggan didasarkan pada tiga faktor penting: Recency (seberapa baru pelanggan terakhir kali bertransaksi), Monetary (nilai total transaksi yang dilakukan oleh pelanggan), dan Frequency (frekuensi transaksi yang dilakukan oleh pelanggan). Saya akan menggunakan pendekatan algoritma K-Means, sebuah algoritma unsupervised learning, untuk mengelompokkan pelanggan menjadi segmen yang berbeda berdasarkan kombinasi dari tiga faktor RMF tersebut. Metode K-Means akan membantu Anda mengidentifikasi kelompok-kelompok pelanggan yang memiliki karakteristik yang serupa dalam hal perilaku. Dengan melakukan segmentasi pelanggan, diharapkan akan mendapatkan wawasan yang berharga mengenai preferensi dan kebiasaan pelanggan. Hal ini dapat membantu dalam merancang strategi pemasaran yang lebih efektif, menargetkan kelompok pelanggan yang tepat, serta meningkatkan kepuasan dan retensi pelanggan.

# Data Understanding

Dataset ini terdiri dari 1000 transaksi pada suatu bank di India. Data ini berisi informasi seperti – customer age (DOB), location, gender, account balance, transaction details, transaction amount, dll.

|  |  |  |
| --- | --- | --- |
| Column | Description | Data Type |
| TransactionID | ID unik untuk setiap transaksi | Object |
| CustomerID | ID unik untuk setiap pelanggan | Object |
| CustomerDOB | Tanggal lahir pelanggan | Object |
| CustGender | Jenis kelamin pelanggan | Object |
| CustLocation | Daerah pelanggan berasal | Object |
| CustAccountBalance | Saldo rekening pelanggan | Float |
| TransactionDate | Tanggal transaksi | Object |
| TransactionTime | Waktu transaksi | Integer |
| TransactionAmount (INR) | Jumlah nominal transaksi | Float |

# Data Preparation

Proses persiapan data melibatkan pengambilan nilai-nilai yang relevan dari data Bank Customer untuk pembuatan data RMF (Recency, Frequency, Monetary). RMF adalah metode yang digunakan untuk menganalisis perilaku pelanggan berdasarkan tiga aspek utama: kebaruan (recency) transaksi, frekuensi (frequency) transaksi, dan nilai moneter (monetary) transaksi. Dalam proses ini, data Bank Customer akan diproses dan diekstraksi untuk mendapatkan informasi yang dibutuhkan, seperti tanggal terakhir transaksi pelanggan, jumlah transaksi yang dilakukan, dan total nilai transaksi yang dilakukan oleh masing-masing pelanggan. Data RMF yang dihasilkan akan digunakan untuk memahami pola dan profil pelanggan yang nantinya akan dilakukan proses segmentasi pada tahapan berikutnya.

## Convert to Datetime Data Type

Pada kolom CustomerDOB dan TransactionDate akan diubah tipe datanya dari object menjadi datetime menggunakan library Pandas. Pada proses ini juga akan dilakukan penghapusan tanggal CustomerDOB yang anomali, yaitu tanggal 1/1/1800.

df = df.loc[~(df['CustomerDOB'] == '1/1/1800')]

df['CustomerDOB'] = pd.to\_datetime(df['CustomerDOB'], format = '%d/%m/%y')

df['TransactionDate'] = pd.to\_datetime(df['TransactionDate'], format = '%d/%m/%y')

## Creating First and Last Transaction Columns

Untuk mendapatkan nilai Recency dibutuhkan tanggal transaksi pertama dan tanggal transaksi terakhir dari pelanggan. Sehingga, kita akan membuat kolom baru yaitu TransactionDateFirst dan TransactionDateLast berdasarkan kolom TransactionDate. Kita juga akan membuat kolom baru yaitu CustomerAge berdasarkan selisih tahun pada TransactionDate dan CustomerDOB.

df['TransactionDateFirst'] = df['TransactionDate'] # first transaction

df['TransactionDateLast'] = df['TransactionDate'] # last transaction

df['CustomerAge'] = df['TransactionDate'].dt.year - df['CustomerDOB'].dt.year

## Creating RMF Dataframe

Pada tahapan ini kita akan membuat dataframe RMF dari dataframe sebelumnya yang sudah diagregasi berdasarkan CustomerID.

#Creating RMF Dataframe

RMF\_df = df.groupby("CustomerID").agg({

    "TransactionID" : "count",

    "CustGender" : "first",

    "CustLocation":"first",

    "CustAccountBalance"  : "mean",

    "TransactionAmount (INR)" : "mean",

    "CustomerAge" : "median",

    "TransactionDateFirst":"min",

    "TransactionDateLast":"max",

    "TransactionDate":"median"

    })

RMF\_df = RMF\_df.reset\_index()

## Creating Frequency and Recency Columns

Langkah selanjutnya adalah membuat kolom Frequency dan Recency pada dataframe RMF. Kolom Frequency berdasarkan kolom TransactionID, kolom Recency berdasarkan selisih antara TransactionDateLast dan TransactionDateFirst. Pada kolom Recency juga akan dilakukan beberapa preparation seperti melakukan convert dari datetime ke string, mengambil angka hari dari string tersebut, lalu convert lagi string ke integer, dan akan mengubah nilai 0 menjadi 1 karena jika 0 berarti pelanggan sebenarnya telah melakukan transaksi satu kali pada tanggal yang sama.

RMF\_df.rename(columns={"TransactionID":"Frequency"},inplace=True)

RMF\_df['Recency']=RMF\_df['TransactionDateLast']-RMF\_df['TransactionDateFirst']

RMF\_df['Recency']=RMF\_df['Recency'].astype(str)

RMF\_df['Recency']=RMF\_df['Recency'].apply(lambda x :re.search('\d+',x).group())

RMF\_df['Recency']=RMF\_df['Recency'].astype(int)

def rep\_0(i):

    if i==0:

        return 1 # 0 days mean that a customer has done transaction recently one time by logic so I will convert 0 to 1

    else:

        return i

RMF\_df['Recency'] = RMF\_df['Recency'].apply(rep\_0)

## Dropping Unnecessary Columns

Pada tahapan ini kita akan menghapus kolom yang tidak dibutuhkan pada tahapan selanjutnya, yaitu kolom TransactionDateFirst, TransactionDateLast, CustomerID, CustLocation, dan TranscationDate.

RMF\_df.drop(columns=["TransactionDateFirst",

                     "TransactionDateLast",

                     "CustomerID",

                      "CustLocation",

                      "TransactionDate",

                      ],

                     inplace=True)

RMF\_df = RMF\_df.reset\_index(drop=True)

# Machine Learning Model Development

## Import Libraries

Terdapat bebarapa libraries yang digunakan di proyek ini:

### Pandas

Pandas adalah sebuah library open-source yang populer digunakan dalam analisis dan manipulasi data pada bahasa pemrograman Python.

### Matplotlib

Matplotlib adalah library visualisasi data yang populer dalam bahasa pemrograman Python. Library ini menyediakan alat untuk membuat berbagai jenis grafik dan visualisasi, termasuk grafik garis, grafik batang, grafik scatter, histogram, heatmap, dan banyak lagi.

### Plotly

Plotly adalah sebuah library visualisasi data interaktif yang populer dalam bahasa pemrograman Python. Library ini menyediakan alat yang kuat untuk membuat grafik interaktif, visualisasi 3D, dan dashboard interaktif yang menarik.

### Yellowbrick

Yellowbrick adalah library Python yang dirancang khusus untuk visualisasi dan pemahaman model machine learning. Pada kasus ini, Yellowbrick akan digunakan untuk melakukan visualisasi K-Elbow yang akan digunakan untuk menentukan nilai K pada model K-Means.

### Sklearn

Scikit-learn, yang juga dikenal sebagai sklearn, adalah library machine learning yang populer dalam bahasa pemrograman Python. Dalam kasus ini kita akan menggunakan model machine learning K-Means.

### Re

Library "re" dalam Python menyediakan dukungan untuk bekerja dengan ekspresi reguler, yang merupakan pola kuat yang digunakan untuk mencocokkan dan memanipulasi teks.

## Data Preprocessing

### Handling Missing Values

Pada tahapan ini kita akan menangani nilai-nilai yang kosong (missing values) dengan cara imputasi atau mengisi nilai-nilai kosong tersebut (missing values). Untuk data dengan tipe numerik akan kita imputasi dengan nilai mediannya, sedangkan untuk data dengan tipe kategorikal akan kita imputasi dengan nilai modusnya.

RMF\_df["CustGender"].fillna(RMF\_df["CustGender"].mode()[0], inplace=True)

RMF\_df["CustomerAge"].fillna(RMF\_df["CustomerAge"].median(), inplace=True)

RMF\_df["CustAccountBalance"].fillna(RMF\_df["CustAccountBalance"].median(), inplace=True)

### Encode Categorical Data

Pada tahap ini kita akan melakukan transformasi berupa encoding data kategori ke dalam bentuk numerik. Pada kasus ini kita akan melakukan encoding pada kolom CustGender, dimana jika bernilai ‘M’ akan diencode menjadi 1 dan jika ‘F’ akan diencode menjadi 0.

RMF\_df['CustGender']=RMF\_df['CustGender'].map({'M':1,'F':0})

### Handling Negative Values

Pada tahapan ini kita akan menghapus nilai negative dari dataframe, karena nilai yang negative dapat dianggap suatu anomaly.

def remove\_negative\_values(dataframe):

    # Mengambil semua kolom dalam DataFrame

    columns = dataframe.columns

    # Menghapus nilai negatif dalam setiap kolom

    for column in columns:

        dataframe[column] = dataframe[column].apply(lambda x: max(x, 0))

    return dataframe

RMF\_df = remove\_negative\_values(RMF\_df)

### Standardization

Pada tahapan ini dilakukan standarisasi atau normalisasi data. Standarisasi adalah proses mengubah data sehingga memiliki rata-rata nol dan standar deviasi satu. Dalam konteks standard scaler tujuan utamanya adala menghilangkan perbedaan skala antara fitur-fitur dalam dataset. Banyak algoritma Machine Learning, seperti K-Means Clustering, sensitive terhadap perbedaan skala antar fitur. Menstandarisasi data memastikan bahwa tidak ada fitur yang dominan hanya karena skala yang lebih besar dan memungkinkan algoritma untuk lebih focus pada pola dan informasi yang relevan.

# data scaling

df\_scaled=StandardScaler().fit\_transform(RMF\_df)

df\_scaled=pd.DataFrame(df\_scaled,columns=RMF\_df.columns)

df\_scaled.head()

## K-Means Model

K-Means adalah algoritma klastering yang populer dan sederhana. Tujuan utama K-Means adalah mempartisi himpunan data menjadi beberapa kelompok (klaster) berdasarkan kemiripan antara data. Algoritma ini bekerja dengan menentukan pusat klaster (centroid) dan mengelompokkan data berdasarkan jarak terdekat dengan pusat klaster. K-Means berusaha meminimalkan varians dalam setiap klaster dan mencapai konvergensi ketika tidak ada lagi perubahan dalam penempatan data. Algoritma ini memerlukan pengaturan jumlah klaster (K) sebelumnya.

# instatiate the clustering model and visualizer

cluster\_model = KMeans(random\_state=42)

### Elbow Method

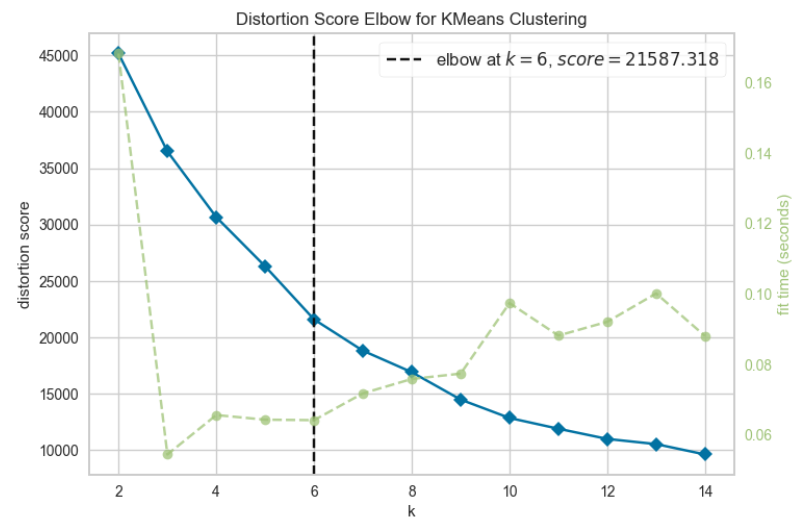
Metode Elbow digunakan untuk membantu menentukan jumlah optimal dari klaster (K) dalam algoritma K-Means. Metode ini mengukur varian dalam setiap klaster saat jumlah klaster berubah. Plot grafik antara jumlah klaster versus varian (inertia) membentuk kurva yang menyerupai bentuk siku (elbow). Jumlah klaster yang optimal dapat ditemukan pada titik "siku" tersebut, di mana penambahan klaster tidak memberikan penurunan varian yang signifikan lagi.

visualizer = KElbowVisualizer(cluster\_model, k=(2, 15))

visualizer.fit(df\_scaled)  # fit the data to the visualizer

visualizer.show()  # finalize and render the figure

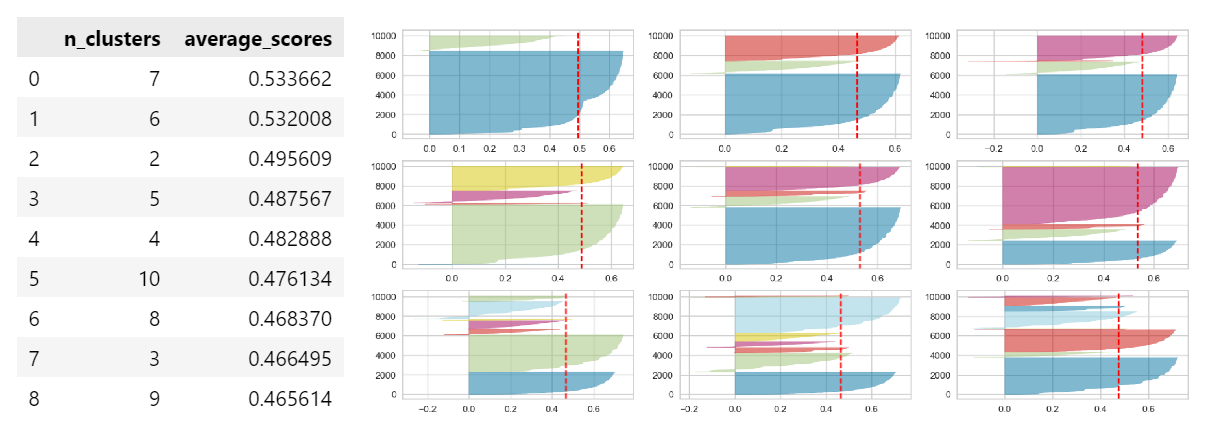
plt.show()



Berdasarkan hasil visualisasi Elbow Method di atas, maka ditentukan nilai K yang optimal adalah 6.

### Silhoutte Score

Silhouette adalah metrik evaluasi yang digunakan untuk mengukur seberapa baik pengelompokan dalam algoritma klastering. Metrik ini menggabungkan kualitas pengelompokan dan jarak antar klaster. Nilai Silhouette berkisar antara -1 hingga 1, di mana nilai positif yang lebih tinggi menunjukkan pengelompokan yang lebih baik. Nilai Silhouette yang mendekati 1 menunjukkan bahwa pengelompokan baik dengan kelompok yang saling terpisah dengan baik, sedangkan nilai yang mendekati -1 menunjukkan adanya campuran antar kelompok. Nilai Silhouette yang mendekati 0 menunjukkan adanya tumpang tindih atau kesulitan dalam mengklasifikasikan data.



from sklearn.metrics import silhouette\_score

from yellowbrick.cluster import SilhouetteVisualizer

k\_range = list(range(2, 11))

lst\_scores = []

fig, axs = plt.subplots(3, 3, figsize=(15, 8))

for k, ax\_i in zip(

    k\_range, [[0, 0], [0, 1], [0, 2], [1, 0], [1, 1], [1, 2], [2, 0], [2, 1], [2, 2]]

):

    """

    Create KMeans instances for different number of clusters

    """

    cluster\_model = KMeans(n\_clusters=k, random\_state=42)

    """

    Print the Silhoutte Score

    """

    preds = cluster\_model.fit\_predict(df\_scaled)

    score = silhouette\_score(df\_scaled, preds)

    lst\_scores.append(score)

    """

    Create SilhouetteVisualizer instance with KMeans instance

    Fit the visualizer

    """

    q = ax\_i[0]

    mod = ax\_i[1]

    visualizer = SilhouetteVisualizer(

        cluster\_model, colors="yellowbrick", ax=axs[q][mod]

    )

    visualizer.fit(df\_scaled)

df\_sillhoutte = pd.DataFrame(

    list(zip(k\_range, lst\_scores)), columns=["n\_clusters", "average\_scores"]

)

df\_sillhoutte.sort\_values("average\_scores", inplace=True, ascending=False)

df\_sillhoutte.reset\_index(drop=True)

Terlihat bahwa nilai Silhoutte untuk K=6 salah satu yang memiliki nilai yang tertinggi. Sehingga kita akan lakukan model training K-Means untuk K=6.

kmeans = KMeans(n\_clusters=6)

kmeans.fit(df\_scaled)

df\_scaled['Label']=kmeans.labels\_

df\_scaled.head()

# Cluster’s Characteristics

A picture containing diagram, screenshot, line, circle

Description automatically generated

A picture containing screenshot, circle, diagram, text

Description automatically generated

A picture containing text, circle, diagram, screenshot

Description automatically generated

A picture containing diagram, line, circle, text

Description automatically generated

A picture containing diagram, text, line, screenshot

Description automatically generated

A picture containing text, screenshot, circle, diagram

Description automatically generated

# Summaries

* Klaster 1: mempunyai pelanggan dengan nilai CustGender dan CustomerAge yang tinggi
* Klaster 2: mempunyai pelanggan dengan nilai Frequency yang tinggi
* Klaster 3: mempunyai pelanggan dengan nilai CustAccountBalance yang tinggi
* Klaster 6: mempunyai pelanggan dengan nilai TransactionAmount yang tinggi
* Klaster 4 dan 5: tidak mempunyai satu ada dua kolom yang mempunyai nilai yang dominan.

# Evaluasi Model

Dalam proses operasionalisasi model clustering ini, dapat dilakukan evaluasi model dalam kurun waktu tertentu untuk mengukur kinerja dan efektivitas model tersebut dengan mempertimbangkan jumlah dan isi klaster sebagai berikut:

* **Apakah hasil clusteringnya bagus?**

Inspeksi secara manual dengan mengunduh data dan dianalisa menggunakan Excel Spreadsheet. Setelah inspeksi, dapat melakukan corrective action sebagai berikut:

* Scale data
* Cek apakah ada outliers
* Correct similarity measure
* **Apakah similarity measure-nya sudah bagus?**

Dapat dilakukan pengecekan berdasarkan data sebelumnya. Berikut adalah script untuk menghitung rata-rata nilai similarity measure setiap cluster:

# Import libraries

import pandas as pd

import numpy as np

from sklearn.metrics.pairwise import euclidean\_distances

# Import dataset

df\_clusters = pd.read\_csv('../data/output.csv')

df\_clusters['CustGender'] = df\_clusters['CustGender'].map({'M':1,'F':0})

# Calculate the similarity measure

cluster\_ids = df\_clusters['Label'].unique()

similarity\_measures = []  # List untuk menyimpan similarity measure tiap cluster

for cluster\_id in cluster\_ids:

    cluster\_data = df\_clusters[df\_clusters['Label'] == cluster\_id]

    cluster\_data = cluster\_data.loc[:,:'Recency']

    similarity\_matrix = euclidean\_distances(cluster\_data)

    similarity\_measure = similarity\_matrix.mean()

    similarity\_measure = 1 - (similarity\_measure / similarity\_matrix.max())

    similarity\_measure = round(similarity\_measure, 2)

    similarity\_measures.append({'Cluster': cluster\_id, 'Similarity Measure': similarity\_measure})

# Make dataframe from similarity measures

similarity\_df = pd.DataFrame(similarity\_measures)

similarity\_df.sort\_values('Cluster', inplace=True,ascending=True)

similarity\_df.reset\_index(inplace=True, drop=True)

* **Apakah optimum number of cluster sudah tepat?**

Cek apakah dalam suatu cluster berisikan 5-30% dari jumlah data. Jumlah cluster yang terlalu banyak atau terlalu sedikit tidak dapat menghasilkan insight yang baik untuk bisnis.

# Data Preprocessing Full Script

# Import libraries

import pandas as pd

import re

from sklearn. preprocessing import StandardScaler

import os

import datetime

# Import Dataset

df = pd.read\_csv(os.getcwd() + '\data\dataset.csv')

"""

Data Preparation Steps

"""

# Convert to datetime

df = df.loc[~(df['CustomerDOB'] == '1/1/1800')]

df['CustomerDOB'] = pd.to\_datetime(df['CustomerDOB'], format = '%d/%m/%y')

df['TransactionDate'] = pd.to\_datetime(df['TransactionDate'], format = '%d/%m/%y')

# Creating first and last transaction columns

df['TransactionDateFirst'] = df['TransactionDate'] # to calculate the minimum (first transaction)

df['TransactionDateLast'] = df['TransactionDate'] # to calculate the maximum (last transaction)

df['CustAge'] = datetime.datetime.now().year - df['CustomerDOB'].dt.year

# Creating MRF dataframe

MRF\_df = df.groupby("CustomerID").agg({

    "TransactionID" : "count",

    "CustGender" : "first",

    "CustLocation":"first",

    "CustAccountBalance"  : "mean",

    "TransactionAmount (INR)" : "mean",

    "CustAge" : "median",

    "TransactionDateFirst":"min",

    "TransactionDateLast":"max",

    "TransactionDate":"median"

    })

MRF\_df = MRF\_df.reset\_index()

# Creating frequency and recency columns

MRF\_df.rename(columns={"TransactionID":"Frequency"},inplace=True)

MRF\_df['Recency']=MRF\_df['TransactionDateLast']-MRF\_df['TransactionDateFirst']

MRF\_df['Recency']=MRF\_df['Recency'].astype(str)

MRF\_df['Recency']=MRF\_df['Recency'].apply(lambda x :re.search('\d+',x).group())

MRF\_df['Recency']=MRF\_df['Recency'].astype(int)

def rep\_0(i):

    if i==0:

        return 1 # 0 days mean that a customer has done transaction recently one time by logic so I will convert 0 to 1

    else:

        return i

MRF\_df['Recency']=MRF\_df['Recency'].apply(rep\_0)

# Dropping unnecessary columns

MRF\_df.drop(columns=["TransactionDateFirst",

                     "TransactionDateLast",

                     "CustomerID",

                      "CustLocation",

                      "TransactionDate",

                      ],

                     inplace=True)

MRF\_df = MRF\_df.reset\_index(drop=True)

"""

Data Preprocessing Steps

"""

# Handling missing values

MRF\_df["CustGender"].fillna(MRF\_df["CustGender"].mode()[0], inplace=True)

MRF\_df["CustAge"].fillna(MRF\_df["CustAge"].median(), inplace=True)

MRF\_df["CustAccountBalance"].fillna(MRF\_df["CustAccountBalance"].median(), inplace=True)

# Encode categorical data

MRF\_df['CustGender']=MRF\_df['CustGender'].map({'M':1,'F':0})

# Handling negative values

def remove\_negative\_values(dataframe):

    # Mengambil semua kolom dalam DataFrame

    columns = dataframe.columns

    # Menghapus nilai negatif dalam setiap kolom

    for column in columns:

        dataframe[column] = dataframe[column].apply(lambda x: max(x, 0))

    return dataframe

MRF\_df = remove\_negative\_values(MRF\_df)

# data scaling

df\_scaled = StandardScaler().fit\_transform(MRF\_df)

df\_scaled=pd.DataFrame(df\_scaled,columns=MRF\_df.columns)

df\_scaled.head()

# remap gender column

MRF\_df['CustGender'] = MRF\_df['CustGender'].map({1:'M',0:'F'})

"""

DF to CSV

"""

# Create a CSV file

MRF\_df.to\_csv(os.getcwd() + '\data\cleaned\_data.csv', index=False)

df\_scaled.to\_csv(os.getcwd() + '\data\standardized\_data.csv', index=False)

# Clustering (K-Means) Training Full Script

# Import Libraries

from sklearn.cluster import KMeans

import plotly.express as px

import pandas as pd

import pickle

import os

import warnings

warnings.filterwarnings("ignore")

# Import Cleaned Dataset

df\_scaled = pd.read\_csv(os.getcwd() + '\data\standardized\_data.csv')

df\_cleaned = pd.read\_csv(os.getcwd() + '\data\cleaned\_data.csv')

# K-Means with K=6

kmeans = KMeans(n\_clusters=6, random\_state=0)

preds = kmeans.fit\_predict(df\_scaled)

df\_scaled['Label'] = preds

# Save the model

pickle.dump(preds, open(os.getcwd() + '\models\clustering\_kmeans.pkl', 'wb'))

# Visualize cluster charateristics

def plot\_cluster(n\_cluster):

    cluster = df\_scaled[df\_scaled['Label']==n\_cluster].loc[:,:"Recency"]

    fig = px.line\_polar(cluster,

                        r = cluster.mean().tolist(),

                        theta = cluster.columns.tolist(),

                        line\_close = True)

    fig.update\_layout(

        title="Cluster {}".format(n\_cluster+1),

    )

    fig.show()

for i in range(0,6):

    plot\_cluster(i)

# add clusters label column to the RMF df

df\_cleaned['Label'] = kmeans.labels\_

# change the cluster label order

change\_order = {0:1, 1:2, 2:3, 3:4, 4:5, 5:6}

df\_cleaned['Label'] = df\_cleaned['Label'].replace(change\_order)

# saving cluster df to csv

df\_cleaned.to\_csv(os.getcwd() + "\\data\output.csv" , index=False)

output\_directory = os.getcwd() + "\\data\clusters\\"

for label in range(1, 7):

    cluster = df\_cleaned[df\_cleaned['Label'] == label]

    file\_path = f"{output\_directory}cluster\_{label}.csv"

    cluster.to\_csv(file\_path, index=False)

    print(f"Saved cluster {label} to {file\_path}")

# Github Repo

<https://github.com/adrn-mm/customer_segmentation>