cabify

Rider Growth Analysis

Business Plan 2011

Diego Beteta

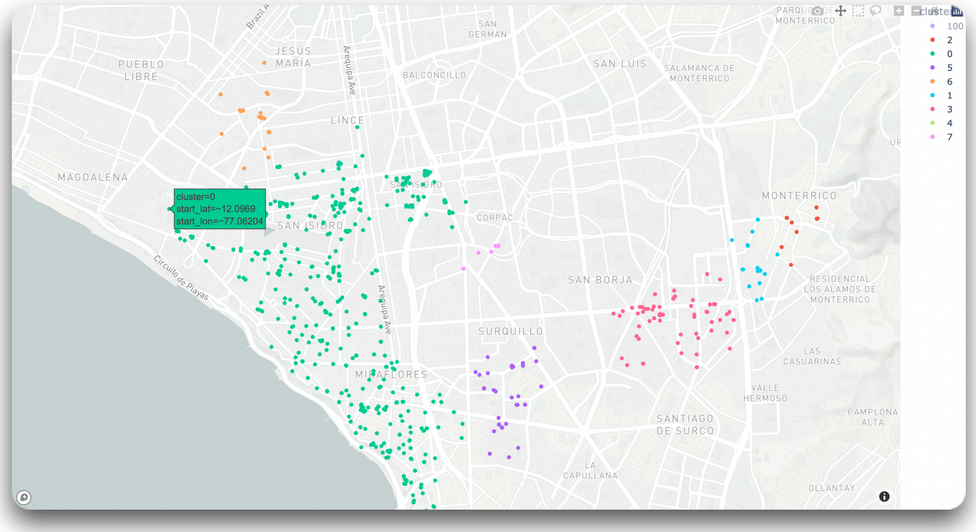
1. Go where the money is!

- Barranco, Miraflores, San Borja and San Isidro are the districts where there are more cases of software failures, driver not found, driver not show or driver cancel.
- To meet the demand, we must put more cabs in these districts.

1. Go where the money is!

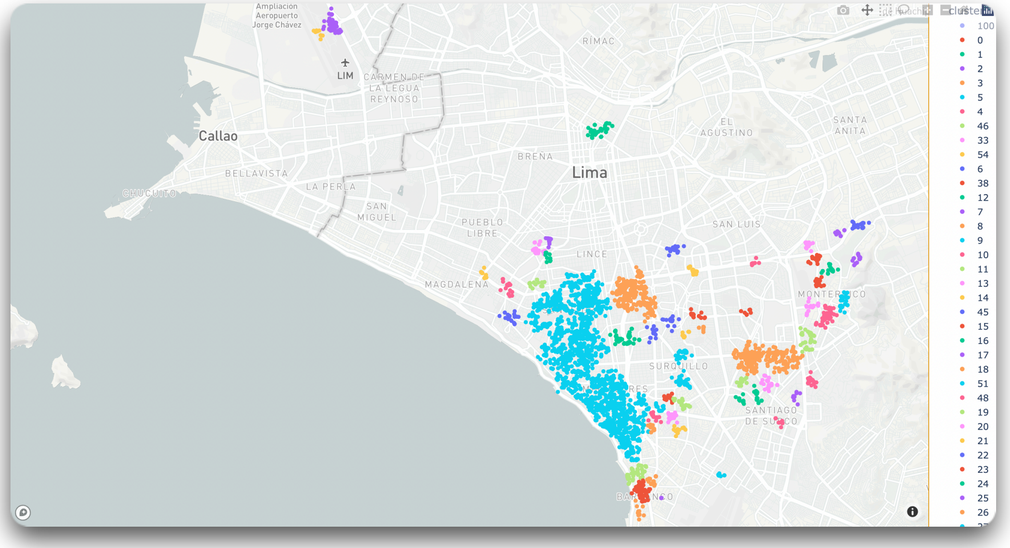
```python
# Districts where there are more cases of end_at with failuer, not found, no show and driver cancel values
df_not_found = df_not_found[df_not_found['end_state'].isin(['failure', 'not found', 'no show', 'driver cancel'])]
X = df_not_found[['start_lat','start_lon']].values

# Normalizing
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
Clus_dataSet = StandardScaler().fit_transform(X)

# Clusterizar zonas donde al menos hay 20 solicitudes de servicio
clustering = DBSCAN(eps=0.10, min_samples=20).fit(Clus_dataSet)
#clustering = KMeans(n_clusters=5).fit(Clus_dataSet)
df_not_found['cluster']= clustering.labels_
df_not_found['cluster'] = df_not_found['cluster'].astype('str')
df_not_found.loc[df_not_found['cluster']=='-1','cluster']='100'

draw_cluster(df_not_found)
```

✓ 0.1s




```python
# Districts where ther are more cases of end_at as drop_off (executed services)
X = df_drop_off[['start_lat','start_lon']].values

# Normalizing
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
Clus_dataSet = StandardScaler().fit_transform(X)

# Clusterizar zonas donde al menos hay 20 solicitudes de servicio
clustering = DBSCAN(eps=0.05, min_samples=20).fit(Clus_dataSet)
#clustering = KMeans(n_clusters=5).fit(Clus_dataSet)
df_drop_off['cluster']= clustering.labels_
df_drop_off['cluster'] = df_drop_off['cluster'].astype('str')
df_drop_off.loc[df_drop_off['cluster']=='-1','cluster']='100'

draw_cluster(df_drop_off)
```

✓ ✓ 0.3s



2. It’s work o’clock!

- In Lima Metropolitan Area, on Mondays from 6:00 to 8:00 a.m. and from 4:00 to 6:00 p.m., our clients need us more often to get around!
- It is more likely that on Mondays and Tuesdays, at midnight, at 8:00 a.m. and close to 8:00 p.m., our customers will have trouble finding cabs due to excessive traffic, platform errors or driver canceling the service or not showing up.

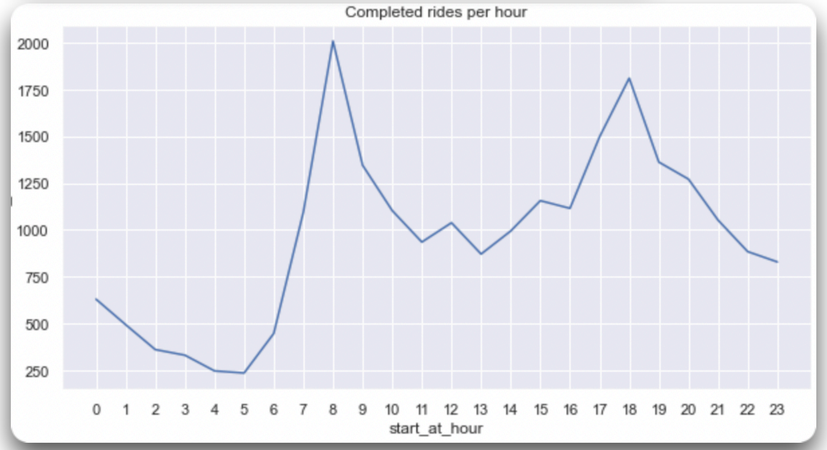
2. It’s work o’clock!

```python
# Extract hour of each start_at
def hr_func(ts):
    return ts.hour
df['start_at_hour'] = df.start_at.apply(hr_func)
```

```python
# Set start_at as index
df_drop_off = df.copy()
df_drop_off.set_index('start_at', inplace=True)
df_drop_off.sort_index(ascending=False)
```

```python
# Users per hour along all 2010
df_hour_traffic = df_drop_off[['user_id', 'start_at_hour']].groupby(['start_at_hour']).count()
df_hour_traffic
```

```python
# Completed rides per hour
sns.set()
plt.figure(figsize=(10,5))
x = np.arange(0, 24, 1)
sns.lineplot(x='start_at_hour', y='user_id', data=df_hour_traffic)
plt.title('Completed rides per hour')
plt.xticks(x)
plt.show()
```

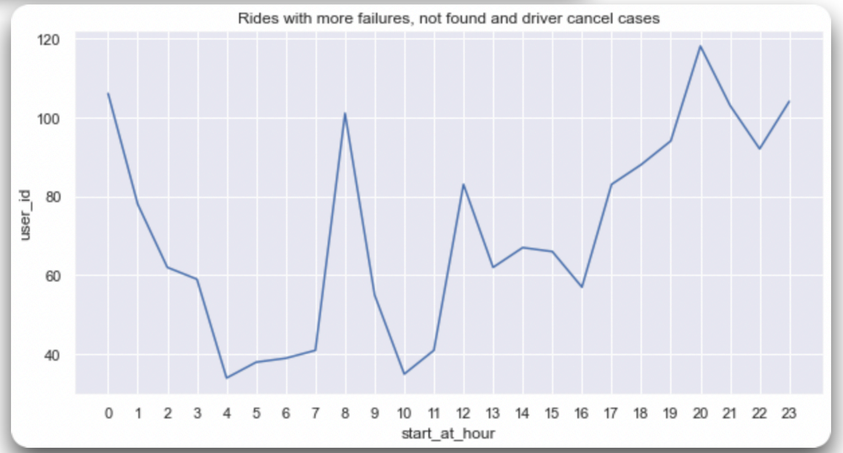



```python
# Filter end_state with 'not found', 'no show' values
df_not_found = df.copy()
df_not_found = df_not_found[df_not_found['end_state'].isin(['failure', 'not found', 'no show', 'driver cancel'])]
print(df_not_found['end_state'].count())

columns = ['user_id', 'Day of Week - Start at']
notfound_per_day_of_week = df_not_found[columns].groupby(['Day of Week - Start at']).count().reindex(order_dayweek)
notfound_per_day_of_week
```

Day of Week - Start at	user_id
Monday	269
Tuesday	374
Wednesday	256
Thursday	165
Friday	192
Saturday	227
Sunday	223

```python
notfound_hour_traffic = df_not_found[['user_id', 'start_at_hour']].groupby(['start_at_hour']).count()
notfound_hour_traffic
```

```python
# Rides with more failures, not found and driver cancel cases
sns.set()
plt.figure(figsize=(10,5))
x = np.arange(0, 24, 1)
sns.lineplot(x='start_at_hour', y='user_id', data=notfound_hour_traffic)
plt.title('Rides with more failures, not found and driver cancel cases')
plt.xticks(x)
plt.show()
```



3. Time is money!

- About 12 minutes, on average, is how long it takes our drivers to get to the pick-up point once the service is accepted.
- Throughout 2010, our rides have an average duration of 40 minutes within Lima Metropolitan Area.

3. Time is money!

```
# Time (in minutes) between Start and Arrived
df['Arrived-Start_duration_min'] = (df.arrived_at  - df.start_at).dt.total_seconds()/60
```

```
# Apply filters to determine ride times of completed rides
df_drop_off = df_drop_off[(df_drop_off['arrived_at'] > df_drop_off.index) &
                          (df_drop_off['end_at'] > df_drop_off['arrived_at']) &
                          (df_drop_off['arrived_at'].notnull()) &
                          (df_drop_off['end_at'].notnull())]
df_drop_off
```

```
# Describe Arrived-Start_duration_min
df_drop_off['Arrived-Start_duration_min'].describe()
```

```
count    12156.000000
mean        12.094248
std         14.643811
min          0.016667
25%          4.666667
50%          9.950000
75%         16.450000
max        626.666667
Name: Arrived-Start_duration_min, dtype: float64
```

```
# Exclude Arrived-Start_duration_min > 3 hours (180 min) to avoid outliers
df_drop_off[df_drop_off['Arrived-Start_duration_min'] < 180]['Arrived-Start_duration_min'].describe()
```

```
count    12151.000000
mean        11.896522
std         10.560044
min          0.016667
25%          4.658333
50%          9.933333
75%         16.433333
max        153.900000
Name: Arrived-Start_duration_min, dtype: float64
```

```
# Time (in minutes) between Arrived and End
df['End-Arrived_duration_min'] = (df.end_at  - df.start_at).dt.total_seconds()/60
```

```
# Apply filters to determine ride times of completed rides
df_drop_off = df_drop_off[(df_drop_off['arrived_at'] > df_drop_off.index) &
                          (df_drop_off['end_at'] > df_drop_off['arrived_at']) &
                          (df_drop_off['arrived_at'].notnull()) &
                          (df_drop_off['end_at'].notnull())]
df_drop_off
```

```
# Describe End-Arrived_duration_min
df_drop_off['End-Arrived_duration_min'].describe()
```

```
count    12156.000000
mean        43.937941
std         42.783821
min          0.083333
25%         25.883333
50%         36.533333
75%         51.483333
max       1417.083333
Name: End-Arrived_duration_min, dtype: float64
```

```
# Exclude End-Arrived_duration_min > 3 hours (180 min)
df_drop_off[df_drop_off['End-Arrived_duration_min'] < 180]['End-Arrived_duration_min'].describe()
```

```
count    12010.000000
mean        40.698719
std         24.948904
min          0.083333
25%         25.750000
50%         36.233333
75%         50.716667
max        179.850000
Name: End-Arrived_duration_min, dtype: float64
```

4. Know your customer!

- This is possible applying a K-Means clustering model based on RFM metrics (Recency, Frecuency and MonetaryValue).
- The results were three clusters labelled as ‘profitable customers’, ‘potential profitable customers’ and ‘desinterested customers’
- We have 242 profitable customers have monetary value of 200,000 PEN, with 57 times that purchase our services again and with 21 days have passed since their last purchase, on approximate average!

4. Know your customer!

RFM Segmentation (Recency, Frequency and Monetary)

Segmentación de conducta de cliente basado en 03 métricas:

- Recency: Mide la cantidad de días que han pasado desde la última compra que hizo el cliente durante los últimos 12 meses.
- Frequency: Mide la cantidad acumulada de las veces que el cliente compró durante los últimos 12 meses.
- MonetaryValue: Mide la cantidad acumulada de dinero que el cliente ha gasto en nuestros productos en los últimos 12 meses.

```
# Crear snapshot_date para seleccionar la fecha más reciente de todo el dataset, sumar 1 para simular que es el día que hacemos el análisis
snapshot_date = max(df.start_at) + dt.timedelta(days=1)

# Agrupar los datos por user_id
datamart = df.groupby(['user_id']).agg({
    'start_at': lambda x: (snapshot_date - x.max()).days, # Fecha del día del análisis - Fecha del start_at más reciente por cliente
    'journey_id': 'count',
    'price': 'sum'})

# Renombar columnas para una fácil interpretación
datamart.rename(columns = {'start_at': 'Recency',
                           'journey_id': 'Frequency',
                           'price': 'MonetaryValue'}, inplace=True)

# Datamart es una tabla donde las filas son cada cliente con su antiguedad, frecuencia y valor monetario al día de hoy.
datamart.head()
```

user_id	Recency	Frequency	MonetaryValue
00cb8ad6a0f7214d002e3ded6b7c9b80	123	7	31587.0
00cb8ad6a0f7214d002e3ded6b7de012	101	5	14056.0
00cb8ad6a0f7214d002e3ded6b7eb91b	317	1	5368.0
0194b4a5c9e41bfd35f9168423fa2857	137	23	50935.0
0221e01fd79a9e76808595048df20efd	13	15	59668.0

Supuestos claves de k-means clustering respecto a las variables (R/F/M)

1. Las variables deben tener distribuciones simétricas (Las variables sesgadas se gestionan con transformación logarítmica)
2. Las variables deben tener el mismo promedio (Para garantizar que se asigne la misma importancia a cada variable)
3. Las varialbes deben tener la misma varianza (Para garantizar que se asigne la misma importancia a cada variable)

3.1. Explorar datos para k-means clustering

Pasos

1. Explorar variables con distribuciones asimétricas - aplicarles transformación logarítimica
2. Normalizar/Estandarizar las variables con el mismo promedio
3. Normalizar/Estandarizar las variables con la misma varianza
4. Almacenar como un 'array' separado para ser usado posteriormente para el clustering

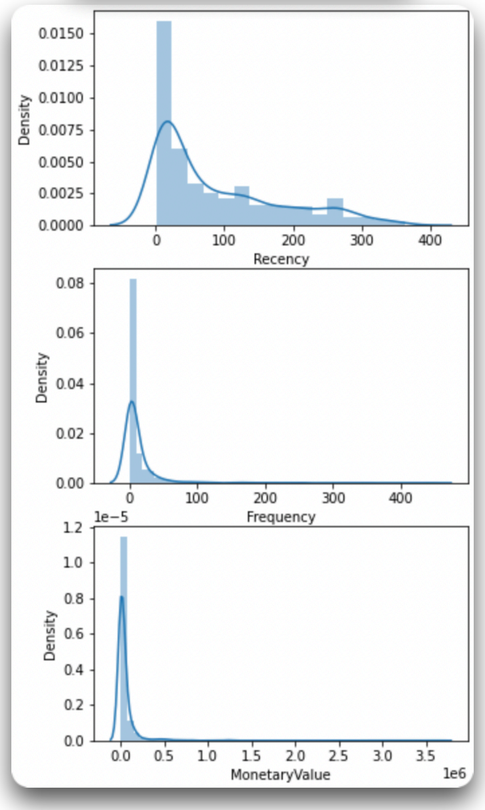
```
# Filtrar columnas CustomerID, Recency, Frequency, MonetaryValue en un nuevo DataFrame
datamart_rfm = datamart[['Recency', 'Frequency', 'MonetaryValue']]
datamart_rfm.describe()
```

	Recency	Frequency	MonetaryValue
count	1217.000000	1217.000000	1.217000e+03
mean	88.062449	14.284306	5.095065e+04
std	91.968759	39.010875	1.564963e+05
min	1.000000	1.000000	1.300000e+03
25%	13.000000	1.000000	4.870000e+03
50%	47.000000	3.000000	1.254000e+04
75%	138.000000	11.000000	3.800500e+04
max	364.000000	446.000000	3.674821e+06

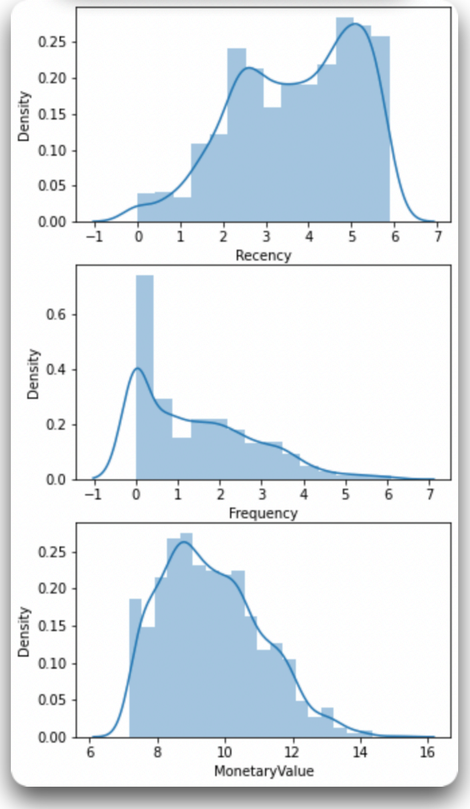
4. Know your customer!

Transform data to log, then StandardNormalized

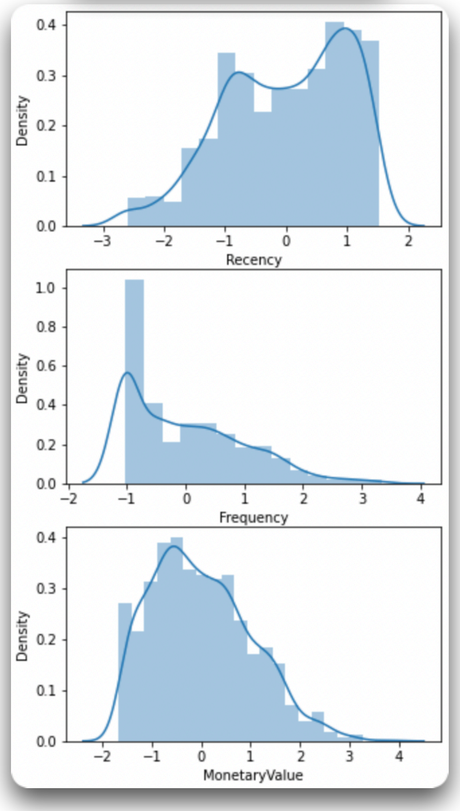
	Recency	Frequency	MonetaryValue
count	1217.000000	1217.000000	1.217000e+03
mean	88.062449	14.284306	5.095065e+04
std	91.968759	39.010875	1.564963e+05
min	1.000000	1.000000	1.300000e+03
25%	13.000000	1.000000	4.870000e+03
50%	47.000000	3.000000	1.254000e+04
75%	138.000000	11.000000	3.800500e+04
max	364.000000	446.000000	3.674821e+06



	Recency	Frequency	MonetaryValue
count	1217.000000	1217.000000	1217.000000
mean	3.714110	1.433896	9.605137
std	1.428372	1.399261	1.455350
min	0.000000	0.000000	7.170120
25%	2.564949	0.000000	8.490849
50%	3.850148	1.098612	9.436679
75%	4.927254	2.397895	10.545473
max	5.897154	6.100319	15.117015



	Recency	Frequency	MonetaryValue
count	1217.00	1217.00	1217.00
mean	-0.00	-0.00	0.00
std	1.00	1.00	1.00
min	-2.60	-1.03	-1.67
25%	-0.80	-1.03	-0.77
50%	0.10	-0.24	-0.12
75%	0.85	0.69	0.65
max	1.53	3.34	3.79



4. Know your customer!

Transform data to log, then StandardNormalized

Definir el número óptimo de clusterings

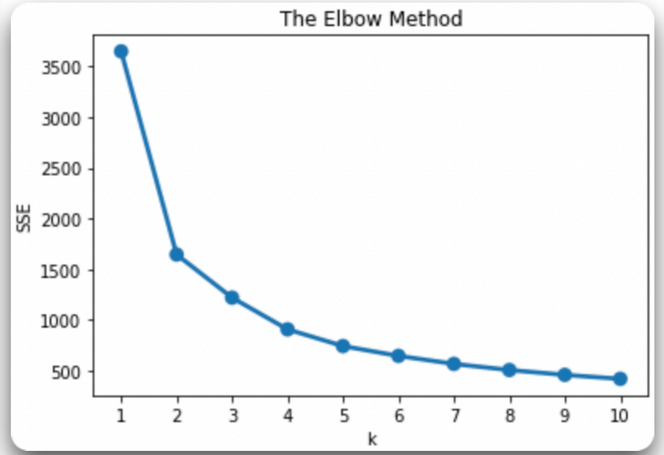
Mediante el método 'Elbow criterion': Fácil de interpretar y brinda una buena estimación

Pasos

- Graficar el número de clusters contra la suma de errores al cuadrado (SSE) dentro del cluster.
- Identificar el "codo" en el gráfico: Punto donde disminuye la disminución de SSE y se vuelve algo marginal.
- Definir el punto "codo" como k-means

(*)SSE: Suma de distancias al cuadrado desde cada punto de datos hasta su centro de clúster.

```
# Importar librerías
from sklearn.cluster import KMeans
import seaborn as sns
from matplotlib import pyplot as plt

# Entrenar el KMeans y calcular el SSE para cada *k*
sse = {}
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=1)
    kmeans.fit(datamart_rfm_normalized)
    sse[k] = kmeans.inertia_ # sum of squared distances to closest cluster center

# Graficar SSE para cada *k*
plt.title('The Elbow Method')
plt.xlabel('k')
plt.ylabel('SSE')
sns.pointplot(x=list(sse.keys()), y=list(sse.values()))
plt.show()
```



Ejecutar el K-Means clustering

- El gráfico The Elbow Method nos sugiere que los números óptimos de clustering son 2 y 3.
- En esta ocasión, escogeré k-means=3 para una mejor flexibilidad en las estrategias comerciales.

```
# El gráfico The Elbow Method nos sugiere que
# los números óptimos de clustering son 2 y 3.

# En esta ocasión, escogeré k-means=3 para
# una mejor flexibilidad en las estrategias comerciales
# y para incrementar la información en el resumen estadístico.
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=1)

# Calcular el k-means clustering sobre la data pre-procesada
kmeans.fit(datamart_rfm_normalized)

# Extraer etiquetas de cluster desde el atributo labels_
cluster_labels = kmeans.labels_
```

4. Know your customer!

Transform data to log, then StandardNormalized

Analizar y visualizar resultados

Enfoques para construir perfiles de clientes:

1. Resúmenes estadísticos para cada clúster
   - Ejecutar la segmentación de k-means para varios valores de k alrededor del valor recomendado
   - Crear una columna de etiqueta de cluster en el datamart_rfm
   - Comparar los valores promedio de RFM de cada solución de agrupamiento
2. Snake plots
3. Importancia relativa de los atributos del segmento en la comparación con la población

Resúmenes estadísticos para cada clúster

```
# Crear una columna de etiquetas de clúster en datamart_rfm
datamart_rfm_k3 = datamart_rfm.assign(Cluster = cluster_labels)

# Calcular el promedio de los valores RFM
# y la cantidad de clientes por cada cluster
datamart_rfm_k3.groupby(['Cluster']).agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'MonetaryValue': ['mean', 'count']
}).round(0)
```

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
Cluster				
0	148.0	2.0	5816.0	552
1	21.0	57.0	199674.0	242
2	49.0	7.0	24764.0	423

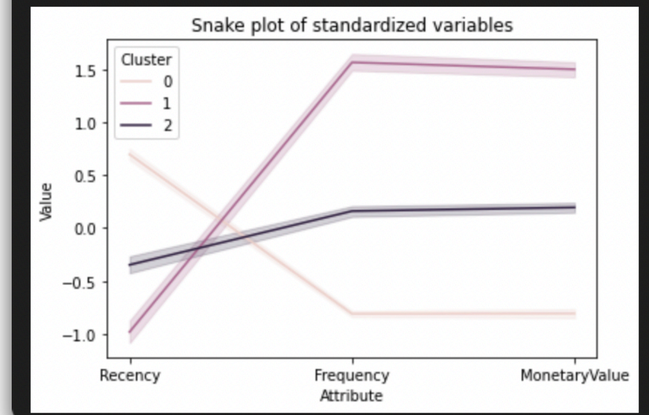
Snake Plots

```
# Preparar la data para el snake plot
# Transformar datamart_normalized como DataFrame y agregar una columna 'Cluster'
datamart_rfm_normalized = pd.DataFrame(datamart_rfm_normalized,
                                       index=datamart_rfm.index,
                                       columns=datamart_rfm.columns)
datamart_rfm_normalized['Cluster'] = datamart_rfm_k3['Cluster']

# Moldear los datos en un formato largo para que los valores RFM
#  y los nombres de las métricas se almacenan en una sóla columna
datamart_melt = pd.melt(datamart_rfm_normalized.reset_index(),
                        # Asignar CustomerID y Cluster como varialbles ID
                        id_vars=['user_id', 'Cluster'],
                        # Asignar RFM_Values como value_vars
                        value_vars=['Recency', 'Frequency', 'MonetaryValue'],
                        var_name='Attribute',
                        value_name='Value')
datamart_melt
```

	user_id	Cluster	Attribute	Value
0	00cb8ad6a0f7214d002e3ded6b7c9b80	2	Recency	0.769075
1	00cb8ad6a0f7214d002e3ded6b7de012	2	Recency	0.631054
2	00cb8ad6a0f7214d002e3ded6b7eb91b	0	Recency	1.432142
3	0194b4a5c9e41bfd35f9168423fa2857	2	Recency	0.844574
4	0221e01fd79a9e76808595048df20efd	1	Recency	-0.804856
...	...	...	...	...
3646	ff1cb10b9edf79369742a37ce5d8d6f6	0	MonetaryValue	-1.489432
3647	ff5c924e0b630fd7c019a4234053c385	0	MonetaryValue	-0.829353
3648	ff5c924e0b630fd7c019a42340593ee2	0	MonetaryValue	-1.489432
3649	ff5c924e0b630fd7c019a42340594b55	1	MonetaryValue	1.093020
3650	ff5c924e0b630fd7c019a42340596fb0	0	MonetaryValue	-1.224326

3651 rows × 4 columns

```
# Graficar el snake plot
plt.title('Snake plot of standardized variables')
sns.lineplot(x='Attribute', y='Value', hue='Cluster',
             data=datamart_melt)

# Presentar gráfico
plt.show()
```



5. Wrap up!

- We have 4x our customer accounts in 2010!
- We have to focus on Barranco, Miraflores, San Borja and San Isidro markets, solve technical problems and review drivers problem causes.
- With 3 cluster of customers we can make marketing campaigns more efficient. Focus on potential profitable customers, not let them go with our competition.

start_at	user_id
2009-12-31	2
2010-01-31	578
2010-02-28	772
2010-03-31	986
2010-04-30	1464
2010-05-31	1595
2010-06-30	1915
2010-07-31	1521
2010-08-31	2119
2010-09-30	2600
2010-10-31	3268
2010-11-30	3018
2010-12-31	3273