



UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA

DIRECCIÓN DE POSGRADO



DIPLOMADO ESTADÍSTICA APLICADA A LA TOMA DE DECISIONES

SEGUNDA VERSIÓN

Laboratorio Spark Machine Learning

NOMBRE : HEBERT JUAN DE DIOS DELGADILLO FERNANDEZ
CARLOS ALFREDO ORIHUELA BERRIOS

DOCENTE : DANNY LUIS HUANCA SEVILLA

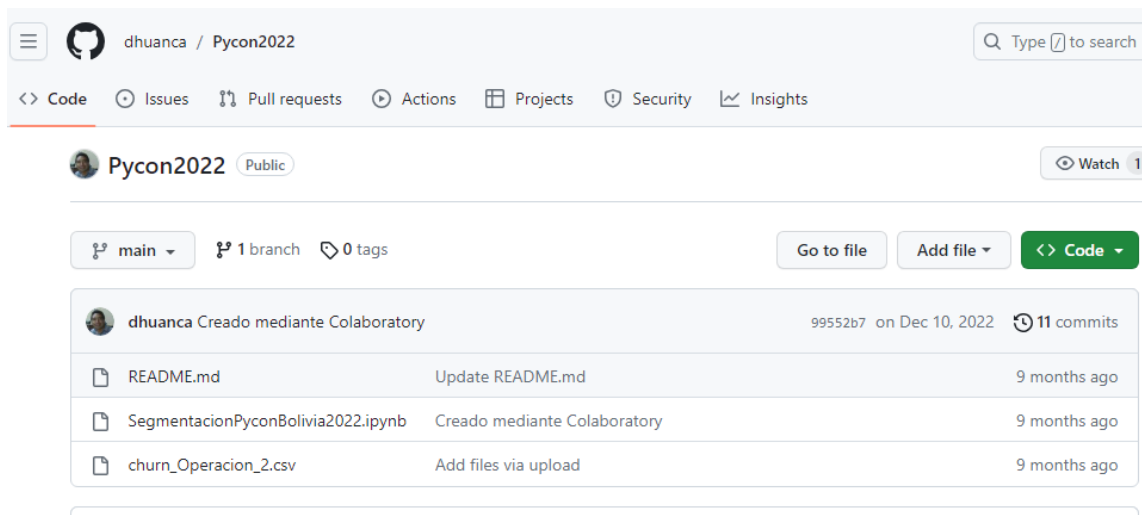
Cochabamba – Bolivia

2023

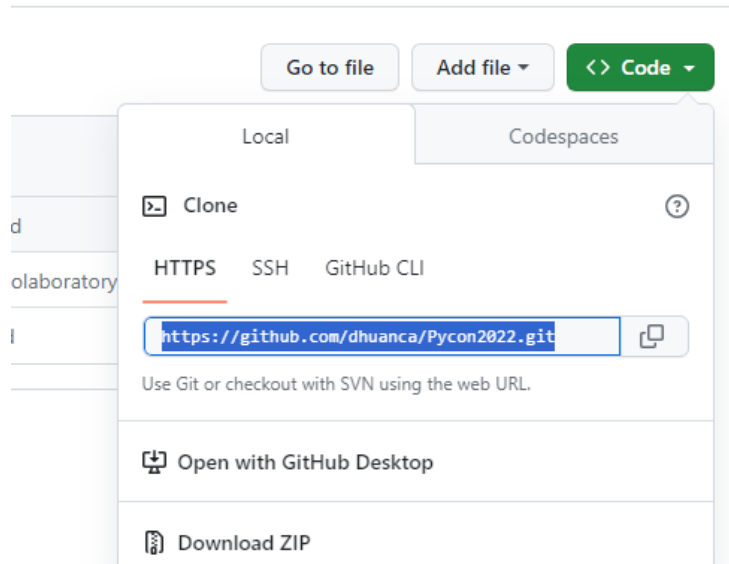
1. Clonar el siguiente repositorio en github. Pueden utilizar Colab, Databricks o la máquina virtual proporcionada. La dirección es la siguiente:

<https://github.com/dhuanca/Pycon2022.git>

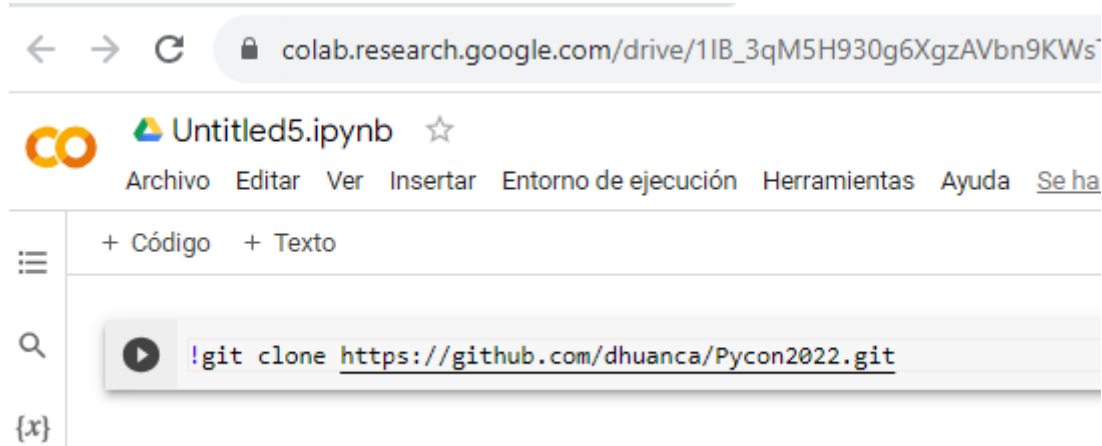
Ingresamos al repositorio GIT



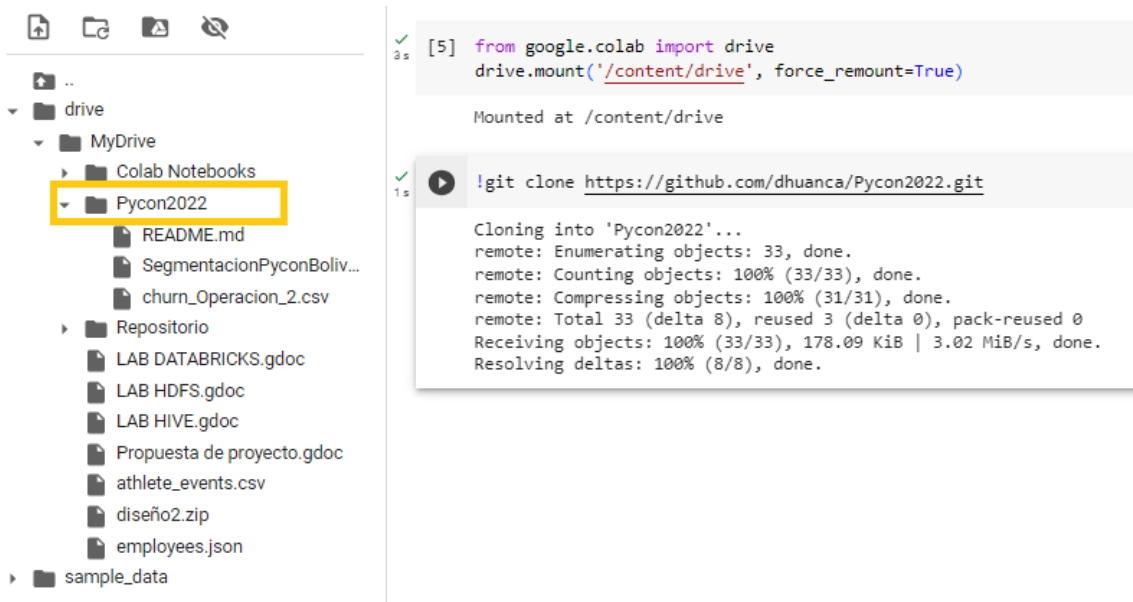
copiamos el link para clonarlo en COLAB



Con el comando `!git clone https://github.com/dhuanca/Pycon2022.git` para clonar el repositorio de gitHub



Y al finalizar todos los archivos del repositorio están clonados en COLAB.



2. Ejecutar el laboratorio y obtener capturas de pantalla por cada ejecución generada

Instalamos la librería `pyspark`, para luego hacer uso `SparkSession` y trabajar con `dataframe`, en nuestro análisis de segmentación.

▼ Modelo Segmentación cliente

```
[1] !pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.4.1.tar.gz (310.8 MB)
    310.8/310.8 MB 3.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.4.1-py2.py3-none-any.whl size=311285387 sha256=8e849c6e1eabb540a3c8979:
  Stored in directory: /root/.cache/pip/wheels/0d/77/a3/ff2f74cc9ab41f8f594dabf0579c2a7c6de920d584206e0834
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.4.1
```

```
from pyspark.sql import SparkSession
import pandas as pd
import matplotlib.pyplot as plt
```

El método `builder` es el constructor de instancias `SparkSession`

```
[3] spark = SparkSession.builder.appName("Segmentacion").getOrCreate()
```

convertimos nuestro dataset en un dataframe de Spark.

```
[9] df_telco = spark.read.csv('drive/MyDrive/Pycon2022/churn_Operacion_2.csv', header=True, inferSchema=True)
```

```
[10] df_telco.show(1)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|state|account|area_code|phone_number|international_plan|voice_mail_plan|number_vmail_messages|total_day_minutes|total_day_calls|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|KS|128.0|415.0|382-4657|no|yes|25.0|265.1|110.0|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 1 row
```

Revisamos los metadatos del Dataframe

```
✓ [11] df_telco.printSchema()
0s

root
|-- state: string (nullable = true)
|-- account: double (nullable = true)
|-- area_code: double (nullable = true)
|-- phone_number: string (nullable = true)
|-- international_plan: string (nullable = true)
|-- voice_mail_plan: string (nullable = true)
|-- number_vmail_messages: double (nullable = true)
|-- total_day_minutes: double (nullable = true)
|-- total_day_calls: double (nullable = true)
|-- total_day_charge: double (nullable = true)
|-- total_eve_minutes: double (nullable = true)
|-- total_eve_calls: double (nullable = true)
|-- total_eve_charge: double (nullable = true)
|-- total_night_minutes: double (nullable = true)
|-- total_night_calls: double (nullable = true)
|-- total_night_charge: double (nullable = true)
|-- total_intl_minutes: double (nullable = true)
|-- total_intl_calls: double (nullable = true)
|-- total_intl_charge: double (nullable = true)
|-- number_customer_service_calls: double (nullable = true)
|-- churn.: string (nullable = true)
```

Definimos un schema para que el modelo identifique a las variables cuantitativas y cualitativas.

```
✓ [12] from pyspark.sql.types import StructType, StructField, StringType, LongType, IntegerType, DoubleType, FloatType
0s

✓ [13] Mischema = StructType(
0s
    [
        StructField('state', StringType(), True),
        StructField('account', StringType(), True),
        StructField('area_code', StringType(), True),
        StructField('phone_number', StringType(), True),
        StructField('international_plan', StringType(), True),
        StructField('voice_mail_plan', StringType(), True),
        StructField("number_vmail_messages", DoubleType(), True),
        StructField('total_day_minutes', DoubleType(), True),
        StructField('total_day_calls', DoubleType(), True),
        StructField('total_day_charge', DoubleType(), True),
        StructField('total_eve_minutes', DoubleType(), True),
        StructField('total_eve_calls', DoubleType(), True),
        StructField('total_eve_charge', DoubleType(), True),
        StructField('total_night_minutes', DoubleType(), True),
        StructField('total_night_calls', DoubleType(), True),
        StructField('total_night_charge', DoubleType(), True),
        StructField('total_intl_minutes', DoubleType(), True),
        StructField('total_intl_calls', DoubleType(), True),
        StructField('total_intl_charge', DoubleType(), True),
        StructField('number_customer_service_calls', DoubleType(), True),
        StructField('churn.', StringType(), True)
    ]
)
```

Explorando los datos , por ejemplo: las 5 personas que menos reclamos y las 5 personas que más reclaman?

```
0s df_telco.sort('number_customer_service_calls')[['number_customer_service_calls', 'phone_number']].show(5)
```

```
+-----+-----+
|number_customer_service_calls|phone_number|
+-----+-----+
|0.0|396-5800|
|0.0|395-2854|
|0.0|358-1958|
|0.0|358-1921|
|0.0|343-4696|
+-----+-----+
only showing top 5 rows
```

```
0s [18] from pyspark.sql import functions as F
```

```
0s [19] df_telco.sort(F.desc('number_customer_service_calls'))[['number_customer_service_calls', 'phone_number']].show(5)
```

```
+-----+-----+
|number_customer_service_calls|phone_number|
+-----+-----+
|9.0|416-2778|
|9.0|416-6886|
|8.0|333-8822|
|8.0|371-1727|
|7.0|397-9184|
+-----+-----+
only showing top 5 rows
```

Para verificar el dominio de variables cualitativas se usan funciones de agrupación por ejemplo:

¿Cuál es la distribución por área?

```
2s [20] df_telco.groupBy(F.col('area_code')).count().show()
```

```
+-----+-----+
|area_code|count|
+-----+-----+
|408|838|
|510|840|
|415|1655|
+-----+-----+
```

¿Cuál es la distribución de la tenencia de un plan de llamadas internacionales?

```
1s [21] df_telco.groupBy(F.col('international_plan')).count().show()
```

```
+-----+-----+
|international_plan|count|
+-----+-----+
|no|3010|
|yes|323|
+-----+-----+
```

¿Cuál es la media de mensajes de texto y llamadas al call center por estado?

```
[22] df_telco[['state', 'number_vmail_messages', 'number_customer_service_calls']].groupBy(F.col('state')).avg().show()
```

state	avg(number_vmail_messages)	avg(number_customer_service_calls)
AZ	9.46875	1.59375
SC	8.683333333333334	1.5833333333333333
LA	7.901960784313726	1.5686274509803921
MN	8.261904761904763	1.5476190476190477
NJ	9.191176470588236	1.6764705882352942
DC	9.74074074074074	1.3518518518518519
OR	6.17948717948718	1.7307692307692308
VA	7.038961038961039	1.5974025974025974
RI	7.323076923076923	1.4153846153846155
WY	6.51948051948052	1.4415584415584415
KY	7.0	1.7118644067796611
NH	8.267857142857142	1.5178571428571428
MI	8.452054794520548	1.63013698630137
NV	6.803030303030303	1.5909090909090908
WI	8.282051282051283	1.4230769230769231
ID	10.219178082191782	1.6712328767123288
CA	9.0	1.4705882352941178
NE	8.557377049180328	1.459016393442623
CT	9.067567567567568	1.5135135135135136
MT	9.338235294117647	1.6323529411764706

only showing top 20 rows

Para graficar con Spark lo convertimos en dataframe de Pandas y usar la librería de graficación. usando el método **toPandas**

```
[24] df_telco_pandas = df_telco[['churn', 'number_customer_service_calls']].groupBy(F.col('churn')).avg().toPandas()
```

```
[25] type(df_telco_pandas)

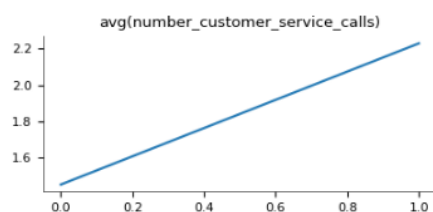
pandas.core.frame.DataFrame
```

```
[26] type(df_telco)
```

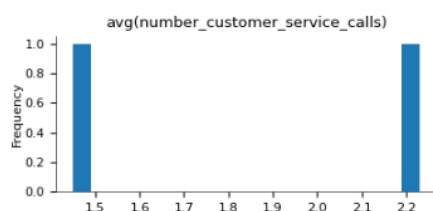
```
[27] df_telco_pandas
```

churn	avg(number_customer_service_calls)
0 False.	1.449825
1 True.	2.229814

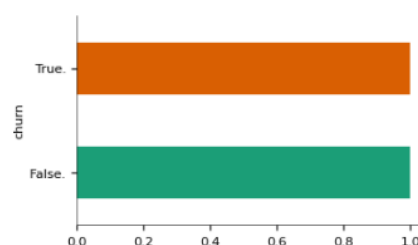
Values



Distributions



Categorical distributions



Faceted distributions



Transformaciones para refinar datos

La variable churn. contiene un punto al final que podemos eliminarlo

```
df_telco.columns
```

```
['state',  
 'account',  
 'area_code',  
 'phone_number',  
 'international_plan',  
 'voice_mail_plan',  
 'number_vmail_messages',  
 'total_day_minutes',  
 'total_day_calls',  
 'total_day_charge',  
 'total_eve_minutes',  
 'total_eve_calls',  
 'total_eve_charge',  
 'total_night_minutes',  
 'total_night_calls',  
 'total_night_charge',  
 'total_intl_minutes',  
 'total_intl_calls',  
 'total_intl_charge',  
 'number_customer_service_calls',  
 'churn']
```

Eliminación de valores nulos

```
[32] df_telco_describe = df_telco.describe().toPandas()
```

```
[33] df_telco_describe
```

	summary	state	account	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_
0	count	3333	3333	3333	3333	3333	3333	3333	
1	mean	None	101.06480648064806	437.18241824182417	None	None	None	8.099009900990099	179.7750975
2	stddev	None	39.822105928595676	42.37129048560661	None	None	None	13.688365372038598	54.4673892
3	min	AK	1	408	327-1058	no	no	0.0	
4	max	WY	99	510	422-9964	yes	yes	51.0	

5 rows x 22 columns

Borrar posibles espacios vacios. Se usa la funcion trim

```
✓ [37] final_df.groupBy(F.col('churn')).count().show()  
0s
```

```
+-----+-----+  
| churn|count|  
+-----+-----+  
| False.| 2850|  
| True. |  483|  
+-----+-----+
```

```
✓ [38] final_df = final_df.withColumn('churn', F.trim(final_df.churn))  
0s
```

```
✓ [39] final_df.groupBy(F.col('churn')).count().show()  
0s
```


```
+-----+-----+  
| churn|count|  
+-----+-----+  
| True. |  483|  
| False.| 2850|  
+-----+-----+
```

Feature engineering

Se realiza una transformación de variables cualitativas.

Los modelos de ML no interpretan palabras o textos, estos deben ser convertidos a números.

Una de estas transformaciones es el one-hot-encoding

International_plan		international_plan_no	international_plan_yes
no		1	0
yes		0	1

```
✓ 0s [40] df_telco_final = final_df.select([
    'state',
    'area_code',
    'international_plan',
    'voice_mail_plan',
    'number_vmail_messages',
    'total_day_minutes',
    'total_day_calls',
    'total_day_charge',
    'total_eve_minutes',
    'total_eve_calls',
    'total_eve_charge',
    'total_night_minutes',
    'total_night_calls',
    'total_night_charge',
    'total_intl_minutes',
    'total_intl_calls',
    'total_intl_charge',
    'number_customer_service_calls',
    "churn"])
```

Aplicando one hot encoding a **international_plan**

```
✓ 0s [42] international_plan_indexer = StringIndexer(inputCol='international_plan', outputCol= 'international_planIndex')

✓ [43] type(international_plan_indexer)

✓ 0s [44] international_plan_encoder = OneHotEncoder(inputCol = 'international_planIndex', outputCol= 'international_planVec')

✓ 0s [45] type(international_plan_encoder)

pyspark.ml.feature.OneHotEncoder
```

Aplicando one hot encoding a **area_code**

```
✓ 0s [48] area_code_indexer = StringIndexer(inputCol='area_code', outputCol= 'area_codeIndex')

✓ 0s [49] area_code_encoder = OneHotEncoder(inputCol = 'area_codeIndex', outputCol= 'area_codeVec')
```

Aplicando one hot encoding a **voice_mail_plan**

```
✓ [50] voice_mail_plan_indexer = StringIndexer(inputCol='voice_mail_plan', outputCol= 'voice_mail_planIndex')
0s

✓ [51] voice_mail_plan_encoder = OneHotEncoder(inputCol = 'voice_mail_planIndex', outputCol= 'voice_mail_planVec')
0s

✓ [52] churn_indexer = StringIndexer(inputCol= 'churn', outputCol= 'churnIndex')
0s

✓ [53] # 'stateVec',
0s assembler = VectorAssembler(inputCols = [ 'area_codeVec', 'international_planVec', 'voice_mail_planVec',
                                             'number_vmail_messages',
                                             'total_day_minutes',
                                             'total_day_calls',
                                             'total_day_charge',
                                             'total_eve_minutes',
                                             'total_eve_calls',
                                             'total_eve_charge',
                                             'total_night_minutes',
                                             'total_night_calls',
                                             'total_night_charge',
                                             'total_intl_minutes',
                                             'total_intl_calls',
                                             'total_intl_charge',
                                             'number_customer_service_calls'
                                             ], outputCol= 'features')
```

Seleccionando las variables que se usarán en el análisis cluster

```
✓ [54] df_telco_final.columns
0s

['state',
 'area_code',
 'international_plan',
 'voice_mail_plan',
 'number_vmail_messages',
 'total_day_minutes',
 'total_day_calls',
 'total_day_charge',
 'total_eve_minutes',
 'total_eve_calls',
 'total_eve_charge',
 'total_night_minutes',
 'total_night_calls',
 'total_night_charge',
 'total_intl_minutes',
 'total_intl_calls',
 'total_intl_charge',
 'number_customer_service_calls',
 'churn']
```

Quitando la variable churn que sirve para un modelo de aprendizaje supervisado

```
df_telco_col_seg = df_telco_final.select([
    # 'state',
    'area_code',
    'international_plan',
    'voice_mail_plan',
    'number_vmail_messages',
    'total_day_minutes',
    'total_day_calls',
    'total_day_charge',
    'total_eve_minutes',
    'total_eve_calls',
    'total_eve_charge',
    'total_night_minutes',
    'total_night_calls',
    'total_night_charge',
    'total_intl_minutes',
    'total_intl_calls',
    'total_intl_charge',
    'number_customer_service_calls'])
```

Modelado

Debido a que el problema de negocio tiene que ver con la identificación de grupos, se usará un algoritmo de clustering que se basa en distancias con la finalidad de crear grupos que se convertirán en segmentos.

Para ello importamos la librería del Modelo de Machine learning **KMeans** y su evaluador **ClusteringEvaluator**

```
✓ [56] from pyspark.ml.clustering import KMeans
0s      from pyspark.ml.evaluation import ClusteringEvaluator
```

```
✓ [57] from pyspark.ml import Pipeline
0s
```

```
✓ [58] kmeans = KMeans().setK(5).setSeed(1)
0s
```

```
✓ [59] pipeline = Pipeline(stages= [
0s      # state_indexer,
      area_code_indexer,
      international_plan_indexer,
      voice_mail_plan_indexer,
      # state_encoder,
      area_code_encoder,
      international_plan_encoder,
      voice_mail_plan_encoder,
      assembler,
      kmeans])
```

dividimos los datos para el entrenamiento y el test 70% y 30% respectivamante

```
✓ [60] train_data, test_data = df_telco_col_seg.randomSplit([0.7,0.3])
```

```
✓ [61] fit_model = pipeline.fit(train_data)
```

```
✓ [62] type(fit_model)
```

pyspark.ml.pipeline.PipelineModel

Comenzamos con las **clusterizaciones**

```
✓ [63] predictions = fit_model.transform(test_data)
```

```
✓ [64] predictions.show(10)
```

al_night_minutes	total_night_calls	total_night_charge	total_intl_minutes	total_intl_calls	total_intl_charge	number_customer_service_calls
203.1	82.0	9.14	10.6	6.0	2.86	1.0
185.8	90.0	8.36	10.0	6.0	2.7	0.0
203.0	97.0	9.14	12.1	13.0	3.27	1.0
223.5	115.0	10.06	10.1	3.0	2.73	3.0
111.2	110.0	5.0	12.1	3.0	3.27	2.0
236.0	113.0	10.62	13.8	1.0	3.73	2.0
230.8	125.0	10.39	9.5	1.0	2.57	6.0
192.2	101.0	8.65	9.8	7.0	2.65	3.0
210.5	66.0	9.47	7.5	5.0	2.03	2.0
98.6	109.0	4.44	8.9	4.0	2.4	1.0

```
✓ [65] predictions.sort(F.desc('voice_mail_plan')).show(5)
```

area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes
408	no	yes	13.0	207.6	65.0	35.29	152.7
408	no	yes	19.0	155.7	104.0	26.47	185.4
408	no	yes	15.0	165.1	85.0	28.07	267.0
408	no	yes	16.0	118.9	112.0	20.21	228.3
408	no	yes	17.0	204.9	84.0	34.83	201.0

only showing top 5 rows

Otra forma de observar sería obteniendo una muestra y convertirlo en un dataframe Pandas

```
✓ [66] predictions_pandas = predictions.sample(fraction=0.5).toPandas()
✓ [67] predictions_pandas.head()
```

ssages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes	total_eve_calls	total_eve_charge	...	total_intl_charge	number_customer_service_calls	area_codeIndex	international_planIndex
0.0	17.8	121.0	2.99	161.7	125.0	13.74	...	2.88	1.0	2.0	0.0
0.0	39.5	78.0	6.72	264.3	106.0	22.47	...	2.70	0.0	2.0	0.0
0.0	51.9	108.0	8.82	162.0	83.0	13.77	...	2.73	3.0	2.0	0.0
0.0	70.8	94.0	12.04	215.6	102.0	18.33	...	2.57	6.0	2.0	0.0
0.0	74.5	117.0	12.07	200.8	98.0	17.07	...	2.85	3.0	2.0	0.0

```
✓ [68] predictions.groupBy(F.col('prediction')).count().show()
```

```
+-----+-----+
|prediction|count|
+-----+-----+
|          1| 182|
|          3| 222|
|          4| 161|
|          2| 176|
|          0| 209|
+-----+-----+
```

```
✓ [69] predictions_pandas['prediction'].value_counts()
```

```
3    119
0    115
1     99
2     95
4     77
Name: prediction, dtype: int64
```

Evaluando con el coeficiente de Silhouette

El coeficiente Silhouette se encuentra entre - 1 y 1 siendo lo mejor 1 y lo peor -1, el valor 0 indica que los clusters se están sobreponiendo.

```
✓ [70] evaluador = ClusteringEvaluator()
```

```
✓ [71] silhouette = evaluador.evaluate(predictions)
      print("El coeficiente Silhouette usando distancias euclidianas al cuadrado es = " + str(silhouette))
```

```
El coeficiente Silhouette usando distancias euclidianas al cuadrado es = 0.27846377025438307
```

3. En la última parte de clustering como vimos en clases identificar las características de cada grupo y asignar un nombre a cada grupo, por ejemplo un grupo podría ser: “las personas que tienen mayor cantidad de llamadas de día y también internacionales, por lo que se les puede ofrecer un paquete de datos en esos horarios y para cuando viajen”

teniendo los clustering

```
✓ [74] predictions.groupBy(F.col('prediction')).count().show()
0s
```

prediction	count
1	182
3	222
4	161
2	176
0	209

luego agrupamos las variables, con la columna prediction

```
✓ [78] predictions.columns
0s
```

```
['area_code',
 'international_plan',
 'voice_mail_plan',
 'number_vmail_messages',
 'total_day_minutes',
 'total_day_calls',
 'total_day_charge',
 'total_eve_minutes',
 'total_eve_calls',
 'total_eve_charge',
 'total_night_minutes',
 'total_night_calls',
 'total_night_charge',
 'total_intl_minutes',
 'total_intl_calls',
 'total_intl_charge',
 'number_customer_service_calls',
 'area_codeIndex',
 'international_planIndex',
 'voice_mail_planIndex',
 'area_codeVec',
 'international_planVec',
 'voice_mail_planVec',
 'features',
 'prediction']
```

el grupo 0: es el grupo promedio

el grupo 1: personas que realizan más llamadas y recargas por las mañanas

el grupo 2: personas que realizan menos recargas por las mañanas y menos llamadas internacionales.

el grupo 3: personas que realizan más llamadas por las tardes y por las noches.

el grupo 4: personas que realizan más llamadas internacionales y menos llamadas por las noches.

```
✓ [87] predictions[['prediction', 'total_day_minutes', 'total_eve_calls', 'total_day_charge', 'total_night_minutes', 'total_intl_calls', 'total_night_charge']] \
18 .groupBy(F.col('prediction')).avg().show()
```

prediction	avg(prediction)	avg(total_day_minutes)	avg(total_eve_calls)	avg(total_day_charge)	avg(total_night_minutes)	avg(total_intl_calls)	avg(total_night_charge)
1	1.0	236.38791208791204	97.91208791208791	40.18620879120882	166.37032967032977	4.6208791208791204	7.486868131868131
3	3.0	210.19459459459475	100.82432432432432	35.73391891891893	254.89504504504504	4.635135135135135	11.470450450450444
4	4.0	189.66273291925467	99.66459627329192	32.24285714285715	165.16894409937905	4.6521739130434785	7.432608695652174
2	2.0	121.80738636363644	99.73295454545455	20.70823863636364	228.55909090909091	4.420454545454546	10.285568181818181
0	0.0	139.37368421052625	98.17703349282297	23.694066985645925	180.84449760765546	4.502392344497608	8.138181818181818