

SUBTEMA 1.1: Análisis estadístico numérico

El análisis exploratorio de datos permite ver el tamaño y las características de los datos. Si los datos son muy variables, todas nuestras suposiciones de normalidad ya no se aplican. O si los datos tienen muchos valores extremos, estos pueden dominar al resto de los valores normales y hacer que los modelos sean muy inexactos. Ahora veremos cómo realizar un análisis exploratorio de datos en pySpark.

Iniciamos con la estadística descriptiva. Ya has visto este tema en otros contextos, **por ejemplo** Tableau para visualización de datos, tal vez cuando aprendiste el lenguaje R o tal vez cuando aprendiste Python y Scikit-Learn. En este tema no vamos a repetir los conceptos, pero te diremos cómo hacerlo en pySpark.

Vamos a suponer que has instalado Spark en un cuaderno de Google Colab y que lo has detectado y que has iniciado una sesión Spark. Vamos a suponer también que has montado el sistema de archivos del Google Drive.

Ahora leamos el archivo de datos “***daily_weather.csv***”.

NOTA: El código con comentarios (#...) ponemos varias formas de llevar a cabo esta lectura:

```
path_csv="/content/daily_weather.csv"

df = sqlContext.read.csv(path_csv,header=True,inferSchema=True)

#df = spark.read.csv(path_csv,header=True,inferSchema=True).coalesce(2)

#df=spark.read.format("csv").option("header","true").option("inferSchema",
a"

,"true").option("minPartitions",2).load(path_csv)

df.show(10)
```

Recuerda que para ver las características de la base datos contenida en el dataframe puedes usar los siguientes comandos:

Comando	Información
df.count()	Número de renglones
len(df.columns)	Número de columnas
df.columns	Nombres de columnas
df.printSchema()	Estructura con tipos de variables

```
df.describe("number", "air_pressure_9am", "air_temp_9am", "avg_wind_direct
ion_9am", "avg_wind_speed_9am", "max_wind_direction_9am").show()
#df.describe()
```

```
+-----+-----+-----+-----+
|summary|          number| air_pressure_9am|    air_temp_9am|
+-----+-----+-----+-----+
|  count|          1095|          1092|          1090|
```

```
|    mean|          547.0|918.8825513138094| 64.93300141287072|
| stddev|316.24357700987383|3.184161180386833|11.175514003175877|
|    min|          0|907.9900000000024|36.752000000000685|
|    max|        1094|929.3200000000012| 98.90599999999992|
+-----+-----+-----+-----+
```

La información es suficientemente pequeña como para pasarlo al nodo coordinador y usar Pandas para un desplegado más atractivo y comprensibles:

```
df.describe().toPandas().transpose()
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
number	1095	547.0	316.24357700987383	0	1094
air_pressure_9am	1092	918.8825513138094	3.184161180386833	907.9900000000024	929.3200000000012
air_temp_9am	1090	64.93300141287072	11.175514003175877	36.752000000000685	98.90599999999992
avg_wind_direction_9am	1091	142.2355107005759	69.13785928889189	15.500000000000046	343.4
avg_wind_speed_9am	1092	5.50828424225493	4.5528134655317185	0.69345139999974	23.5549781999999763
max_wind_direction_9am	1092	148.95351796516923	67.23801294602953	28.899999999999991	312.19999999999993
max_wind_speed_9am	1091	7.019513529175272	5.598209170780958	1.1855782000000479	29.840779599999996
rain_accumulation_9am	1089	0.20307895225211126	1.5939521253574893	0.0	24.019999999999907
rain_duration_9am	1092	294.1080522756142	1598.0787786601481	0.0	17704.0
relative_humidity_9am	1095	34.24140205923536	25.472066802250055	6.090000000000012	92.62000000000002
relative_humidity_3pm	1095	35.34472714825898	22.524079453587273	5.30000000000006855	92.25000000000003

Nótese que en la columna “count” no todas las variables tienen la misma cantidad de valores, por lo que será necesario eliminar valores vacíos y nulos.

Para ver los cuantiles se usa el comando `approxQuantile`. Recuerde que los datos se encuentran distribuidos y la cantidad de ellos puede ser muy grande. Spark utiliza un método estadístico, por lo que los cuantiles son aproximados, no exactos. El siguiente ejemplo nos muestra los cuantiles de la variable **air_pressure_9am**:

```
quantiles_air_pressure_9am =  
df.approxQuantile("air_pressure_9am", [0.25, 0.5, 0.75], 0)  
quantile_25 = quantiles_air_pressure_9am [0]  
quantile_50 = quantiles_air_pressure_9am [1]  
quantile_75 = quantiles_air_pressure_9am [2]  
print('quantile_25: '+str(quantile_25))  
print('quantile_50: '+str(quantile_50))  
print('quantile_75: '+str(quantile_75))
```

```
quantile_25: 916.5500000000009  
quantile_50: 918.9020905167166  
quantile_75: 921.16000000000036
```

El siguiente ejemplo muestra los cuantiles de todas las variables:

```
quantiles = df.approxQuantile(df.columns, [0.25, 0.5, 0.75], 0)  
for i in range(len(df.columns)):  
    print("Cuantiles " + str(df.columns[i]) + ": " + str(quantile[0]))  
  
Cuantiles number: [273.0, 547.0, 821.0]  
Cuantiles air_pressure_9am: [273.0, 547.0, 821.0]  
Cuantiles air_temp_9am: [273.0, 547.0, 821.0]  
Cuantiles avg_wind_direction_9am: [273.0, 547.0, 821.0]  
Cuantiles avg_wind_speed_9am: [273.0, 547.0, 821.0]  
Cuantiles max_wind_direction_9am: [273.0, 547.0, 821.0]  
Cuantiles max_wind_speed_9am: [273.0, 547.0, 821.0]  
Cuantiles rain_accumulation_9am: [273.0, 547.0, 821.0]  
Cuantiles rain_duration_9am: [273.0, 547.0, 821.0]  
Cuantiles relative_humidity_9am: [273.0, 547.0, 821.0]  
Cuantiles relative_humidity_3pm: [273.0, 547.0, 821.0]
```

Esta cantidad de información también es suficientemente pequeña para pasarla a Pandas como muestra el siguiente ejemplo:

```
pd.DataFrame(data=quantiles, index=df.columns, columns=[0.25, 0.5, 0.75])
```

	0.25	0.50	0.75
number	273.000000	547.000000	821.000000
air_pressure_9am	916.550000	918.902091	921.160000
air_temp_9am	57.272000	65.696000	73.454000
avg_wind_direction_9am	65.880616	166.000000	191.100000
avg_wind_speed_9am	2.248279	3.869906	7.337163
max_wind_direction_9am	76.456663	177.100000	201.200000
max_wind_speed_9am	3.064608	4.943637	8.970129
rain_accumulation_9am	0.000000	0.000000	0.000000
rain_duration_9am	0.000000	0.000000	0.000000
relative_humidity_9am	15.090000	23.179259	45.590000
relative_humidity_3pm	17.390000	24.380000	52.070000

Para encontrar las correlaciones se usa la función “Correlation” dentro MLIB Spark subpaquete `pyspark.ml.stat`. Sin embargo, requiere que se alimente una columna de tipo vector, por lo que necesario convertir las columnas del dataframe a vectores primero usando la función “VectorAssembler” y después de esto calcular las correlaciones. El resultado es un vector lineal, por lo que hay que cambiarlo a formato matrix. Sin embargo, la matrix de correlaciones no es muy grande, por lo que se pueden recolectar en el nodo central y luego manejarlo todo con Numpy y Pandas.

Ahora, la función “VectorAssembler” es muy importante porque se usa para los algoritmos de machine learning, tanto supervisada como la regresión y la clasificación, como no supervisada. Así es que es afortunado que podamos ver esta función tan importante en un contexto mas bien sencillo.

En el siguiente ejemplo se encuentran las correlaciones de Pearson de nuestro Spark Dataframe. Primero importamos nuestros paquetes y eliminamos valores nulos:

```
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler
```

```
df_dropped = df.na.drop()
df_dropped = df_dropped.drop("number")
df_columns2 = df_dropped.columns
```

Ahora ensamblamos el vector de columnas:

```
# convert to vector column first
vector_col = "corr_features"
assembler = VectorAssembler(inputCols=df_dropped.columns,
outputCol=vector_col)
df_vector = assembler.transform(df_dropped).select(vector_col)
```

Finalmente obtenemos la correlación entre variables:

```
r1 = Correlation.corr(df_vector, vector_col)
print("Pearson Corr
Matrix\n",r1.collect()[0]["pearson({})".format(vector_col)].values)
```

Pearson Corr Matrix

```
[ 1.00e+00 -5.73e-02 -3.13e-01  2.36e-01 -2.87e-01  2.52e-01 -8.54e-02
-1.00e-01 -4.31e-01 -4.80e-01 -5.73e-02  1.00e+00 -1.51e-02 -2.84e-01
-7.95e-02 -2.82e-01 -1.99e-01 -2.59e-01 -5.82e-01 -4.91e-01 -3.13e-01
-1.51e-02  1.00e+00 -4.41e-01  8.69e-01 -4.60e-01  1.18e-01  1.27e-01
 3.47e-01  3.85e-01  2.36e-01 -2.84e-01 -4.41e-01  1.00e+00 -3.64e-01
 9.95e-01  2.32e-03  1.21e-02 -9.55e-02 -1.54e-01 -2.87e-01 -7.95e-02
 8.69e-01 -3.64e-01  1.00e+00 -3.83e-01  9.10e-02  8.26e-02  3.69e-01
 4.06e-01  2.52e-01 -2.82e-01 -4.60e-01  9.95e-01 -3.83e-01  1.00e+00
-3.34e-04  1.19e-02 -1.09e-01 -1.74e-01 -8.54e-02 -1.99e-01  1.18e-01
 2.32e-03  9.10e-02 -3.34e-04  1.00e+00  7.36e-01  2.29e-01  1.82e-01
-1.00e-01 -2.59e-01  1.27e-01  1.21e-02  8.26e-02  1.19e-02  7.36e-01
 1.00e+00  3.05e-01  2.63e-01 -4.31e-01 -5.82e-01  3.47e-01 -9.55e-02
 3.69e-01 -1.09e-01  2.29e-01  3.05e-01  1.00e+00  8.81e-01 -4.80e-01
-4.91e-01  3.85e-01 -1.54e-01  4.06e-01 -1.74e-01  1.82e-01  2.63e-01
 8.81e-01  1.00e+00]
```

Si cambiamos “pearson” por “spearman” se obtiene la correlación Spearman. El resultado es un vector unidimensional. Vamos a recolectarlo y luego pasarlo a una arreglo Numpy y luego un dataframe Pandas:

```
pearson_corr_arr =
np.matrix(r1.collect()[0]["pearson({})".format(vector_col)].values).reshape(len(df.columns)-1, len(df.columns)-1)
pearson_corr_df = pd.DataFrame(data=pearson_corr_arr,
index=df_columns2,columns=df_columns2)
pearson_corr_df
```

	air_ pres sure _9am	air _te mp_ 9am	avg_win d_direc tion_9a m	avg_w ind_s peed_ 9am	max_win d_direc tion_9a m	max_w ind_s peed_ 9am	rain_a ccumul ation_ 9am	rain_ durat ion_9 am	relati ve_hum idity_ 9am	relati ve_hum idity_ 3pm
air_pre ssure_9 am	1.00 0000	-0. 057 329	-0.3133 60	0.235 826	-0.2872 80	0.252 479	-0.085 434	-0.10 0485	-0.431 126	-0.480 117
air_tem p_9am	-0.0 5732 9	1.0 000 00	-0.0150 69	-0.28 4327	-0.0795 20	-0.28 2483	-0.199 102	-0.25 9103	-0.582 318	-0.491 068
avg_win d_direc tion_9a m	-0.3 1336 0	-0. 015 069	1.00000 0	-0.44 1441	0.86883 7	-0.46 0122	0.1176 34	0.127 367	0.3468 65	0.3848 85
avg_win d_speed _9am	0.23 5826	-0. 284 327	-0.4414 41	1.000 000	-0.3643 38	0.995 412	0.0023 22	0.012 057	-0.095 472	-0.154 422
max_win d_direc tion_9a m	-0.2 8728 0	-0. 079 520	0.86883 7	-0.36 4338	1.00000 0	-0.38 3025	0.0910 03	0.082 574	0.3695 00	0.4059 46
max_win d_speed _9am	0.25 2479	-0. 282 483	-0.4601 22	0.995 412	-0.3830 25	1.000 000	-0.000 334	0.011 890	-0.108 673	-0.173 720
rain_ac cumulat ion_9am	-0.0 8543 4	-0. 199 102	0.11763 4	0.002 322	0.09100 3	-0.00 0334	1.0000 00	0.735 737	0.2289 51	0.1818 87
rain_du ration_ 9am	-0.1 0048 5	-0. 259 103	0.12736 7	0.012 057	0.08257 4	0.011 890	0.7357 37	1.000 000	0.3048 28	0.2632 83

relativ e_humid ity_9am	-0.4 3112 6	-0. 582 318	0.34686 5	-0.09 5472	0.36950 0	-0.10 8673	0.2289 51	0.304 828	1.0000 00	0.8813 01
relativ e_humid ity_3pm	-0.4 8011 7	-0. 491 068	0.38488 5	-0.15 4422	0.40594 6	-0.17 3720	0.1818 87	0.263 283	0.8813 01	1.0000 00

Otra forma de hacerlo es con el paquete Statistics de MLIB. Primero eliminamos la column "number" y luego los valores nulos:

```
from pyspark.mllib.stat import Statistics
```

```
df_features = df.drop("number")
```

```
df_features = df.select(df_features.columns).dropna()
```

Luego creamos una table RDD para almacenar las correlaciones:

```
rdd_table = df_features.rdd.map(lambda row: row[0:])
```

Finalmente llamamos "Statistics.corr" que nos generará una verdadera matriz:

```
corr_mat=Statistics.corr(rdd_table, method="pearson")
print(corr_mat)
```

```
[ [ 1.00e+00 -5.73e-02 -3.13e-01  2.36e-01 -2.87e-01  2.52e-01 -8.54e-02
   -1.00e-01 -4.31e-01 -4.80e-01]
  [-5.73e-02  1.00e+00 -1.51e-02 -2.84e-01 -7.95e-02 -2.82e-01 -1.99e-01
   -2.59e-01 -5.82e-01 -4.91e-01]
  [-3.13e-01 -1.51e-02  1.00e+00 -4.41e-01  8.69e-01 -4.60e-01  1.18e-01
   1.27e-01  3.47e-01  3.85e-01]
  [ 2.36e-01 -2.84e-01 -4.41e-01  1.00e+00 -3.64e-01  9.95e-01  2.32e-03
   1.21e-02 -9.55e-02 -1.54e-01]
  [-2.87e-01 -7.95e-02  8.69e-01 -3.64e-01  1.00e+00 -3.83e-01  9.10e-02
   8.26e-02  3.69e-01  4.06e-01]
  [ 2.52e-01 -2.82e-01 -4.60e-01  9.95e-01 -3.83e-01  1.00e+00 -3.34e-04
   1.19e-02 -1.09e-01 -1.74e-01]
  [-8.54e-02 -1.99e-01  1.18e-01  2.32e-03  9.10e-02 -3.34e-04  1.00e+00
   7.36e-01  2.29e-01  1.82e-01]
  [-1.00e-01 -2.59e-01  1.27e-01  1.21e-02  8.26e-02  1.19e-02  7.36e-01
```



```

1.00e+00  3.05e-01  2.63e-01]
[-4.31e-01 -5.82e-01  3.47e-01 -9.55e-02  3.69e-01 -1.09e-01  2.29e-01
 3.05e-01  1.00e+00  8.81e-01]
[-4.80e-01 -4.91e-01  3.85e-01 -1.54e-01  4.06e-01 -1.74e-01  1.82e-01
 2.63e-01  8.81e-01  1.00e+00]]

```

La cual también podemos pasar a un dataframe de Pandas para mejorar la visualización:

	air_ pres sure_ _9am	air_ _te mp_ _9am	avg_win d_direc tion_9a m	avg_w ind_s peed_ _9am	max_win d_direc tion_9a m	max_w ind_s peed_ _9am	rain_a ccumul ation_ _9am	rain_ durat ion_9 am	relati ve_hum idity_ _9am	relati ve_hum idity_ _3pm
air_pre ssure_9 am	1.00 0000	-0. 057 329	-0.3133 60	0.235 826	-0.2872 80	0.252 479	-0.085 434	-0.10 0485	-0.431 126	-0.480 117
air_tem p_9am	-0.0 5732 9	1.0 000 00	-0.0150 69	-0.28 4327	-0.0795 20	-0.28 2483	-0.199 102	-0.25 9103	-0.582 318	-0.491 068
avg_win d_direc tion_9a m	-0.3 1336 0	-0. 015 069	1.00000 0	-0.44 1441	0.86883 7	-0.46 0122	0.1176 34	0.127 367	0.3468 65	0.3848 85
avg_win d_speed _9am	0.23 5826	-0. 284 327	-0.4414 41	1.000 000	-0.3643 38	0.995 412	0.0023 22	0.012 057	-0.095 472	-0.154 422
max_win d_direc tion_9a m	-0.2 8728 0	-0. 079 520	0.86883 7	-0.36 4338	1.00000 0	-0.38 3025	0.0910 03	0.082 574	0.3695 00	0.4059 46
max_win d_speed _9am	0.25 2479	-0. 282 483	-0.4601 22	0.995 412	-0.3830 25	1.000 000	-0.000 334	0.011 890	-0.108 673	-0.173 720
rain_ac cumulat ion_9am	-0.0 8543 4	-0. 199 102	0.11763 4	0.002 322	0.09100 3	-0.00 0334	1.0000 00	0.735 737	0.2289 51	0.1818 87
rain_du ration_ _9am	-0.1 0048 5	-0. 259 103	0.12736 7	0.012 057	0.08257 4	0.011 890	0.7357 37	1.000 000	0.3048 28	0.2632 83
relativ e_humid ity_9am	-0.4 3112 6	-0. 582 318	0.34686 5	-0.09 5472	0.36950 0	-0.10 8673	0.2289 51	0.304 828	1.0000 00	0.8813 01

relative_humidity_3pm	-0.480117	-0.491068	0.384885	-0.154422	0.405946	-0.173720	0.181887	0.263283	0.881301	1.000000
-----------------------	-----------	-----------	----------	-----------	----------	-----------	----------	----------	----------	----------

Lo mismo podemos hacer para obtener la correlación Spearman:

```
corr_mat_sp=Statistics.corr(rdd_table, method="spearman")
corr_mat_pd_sp = pd.DataFrame(data=corr_mat_sp,
index=df_features.columns,columns=df_features.columns)
corr_mat_pd_sp
```

	air_pressure_9am	air_temperature_9am	avg_wind_direction_9am	avg_wind_speed_9am	max_wind_direction_9am	max_wind_speed_9am	rain_accumulation_9am	rain_duration_9am	relative_humidity_9am	relative_humidity_3pm
air_pressure_9am	1.000000	-0.117418	-0.279479	0.232753	-0.243158	0.248009	-0.063062	0.000697	-0.472377	-0.527897
air_temperature_9am	-0.117418	1.000000	0.021986	-0.360481	-0.040398	-0.364843	-0.227157	-0.286164	-0.424956	-0.359926
avg_wind_direction_9am	-0.279479	0.021986	1.000000	-0.377934	0.859601	-0.392063	0.142616	0.039965	0.335965	0.408805
avg_wind_speed_9am	0.232753	-0.360481	-0.377934	1.000000	-0.299334	0.993435	0.082594	0.210421	-0.105392	-0.205306
max_wind_direction_9am	-0.243158	-0.040398	0.859601	-0.299334	1.000000	-0.309779	0.129086	0.043627	0.336457	0.394171
max_wind_speed_9am	0.248009	-0.364843	-0.392063	0.993435	-0.309779	1.000000	0.080661	0.212161	-0.112784	-0.218481
rain_accumulation_9am	-0.063062	-0.227157	0.142616	0.082594	0.129086	0.080661	1.000000	0.846628	0.286516	0.238172

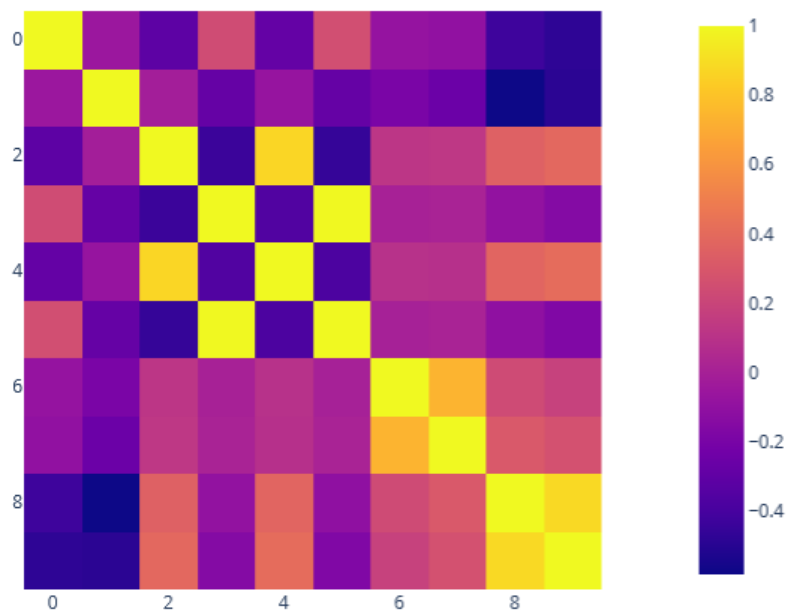
rain_du ration_ 9am	0.00 0697	-0. 286 164	0.03996 5	0.210 421	0.04362 7	0.212 161	0.8466 28	1.000 000	0.2063 81	0.1500 08
relativ e_humid ity_9am	-0.4 7237 7	-0. 424 956	0.33596 5	-0.10 5392	0.33645 7	-0.11 2784	0.2865 16	0.206 381	1.0000 00	0.8667 84
relativ e_humid ity_3pm	-0.5 2789 7	-0. 359 926	0.40880 5	-0.20 5306	0.39417 1	-0.21 8481	0.2381 72	0.150 008	0.8667 84	1.0000 00

Para visualizar las correlaciones correctamente podemos usar `imshow` de `plotly.express` que mostrará las correlaciones como un mapa de calor, el paquete `seaborn` que también usará colores o `corrplot` del paquete `heatmapz` que creará un mapa de color más fácil de leer.

Usando `imshow` y el arreglo de correlaciones:

```
import plotly.express as px

fig = px.imshow(pearson_corr_arr)
fig.show()
```



Nótese cómo es posible ver correlaciones significativas entre varias variables (pintadas de amarillo y naranja).

Ahora usando `seaborn` y el dataframe de correlaciones en vez del arreglo:

```

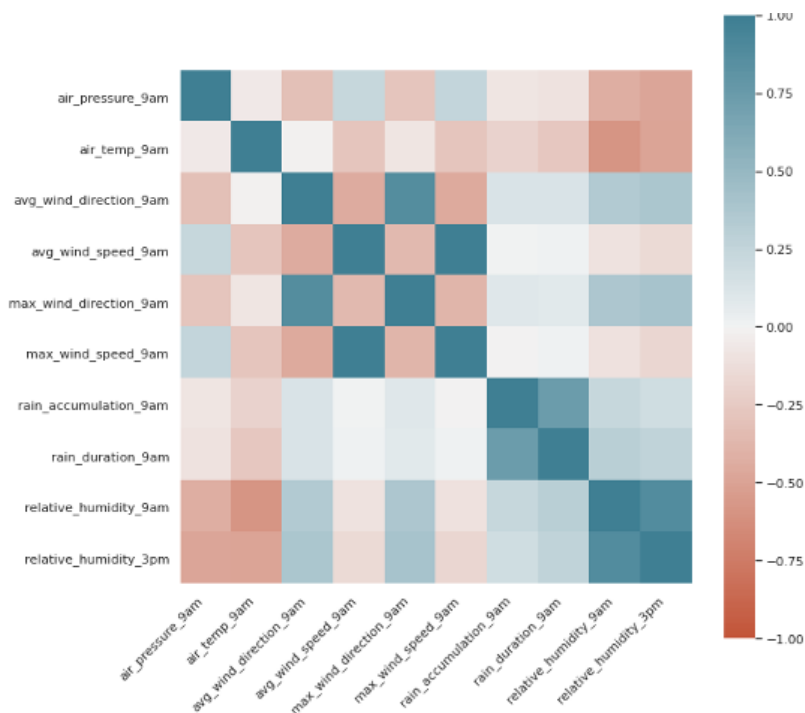
import seaborn as sns
sns.set(color_codes=True, font_scale=1.0)
from matplotlib import pyplot as plt

#Establecer el tamaño de la figura
plt.figure(figsize=(10,10))

ax = sns.heatmap(
    pearson_corr_df,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)

#La barra de anotación
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);

```



Y ahora usando el paquete corrplot de heatmapz:

```
!pip install heatmapz
from heatmap import corrplot
```

```
plt.figure(figsize=(8, 8))
corrplot(pearson_corr_df ,size_scale=300);
```

