Análisis gráfico de información

La representación gráfica es muy importante porque permite establecer relaciones entre las variables de una forma que muchas personas pueden entender y para algunas personas, de una forma más rápida.

En esta sección exploraremos las características de los datos y las relaciones entre las variables de forma gráfica. **¿Crees que es importante el análisis gráfico de los datos en tu empresa?**

Todas las gráficas las produciremos con paquetes conocidos que funcionan en un solo nodo por lo que antes de crear las gráficas debemos hacer dos cosas.

1.- ***Primero, es posible que los datos sean más grandes que la capacidad del nodo que producirá las visualizaciones, por lo que tal vez sea necesario obtener una muestra. No se preocupe por perder datos, las gráficas sólo se pueden hacer con una cantidad limitada de datos de todas maneras porque de otra forma pierden significado.***

2.- Lo segundo es que hay que recolectar los datos y pasarlos a arreglos para alimentarlos a las funciones de graficación.

Muestreando los datos de RDDs y Spark Dataframes

*Hay dos funciones para muestrear. Una es "**takeSample**" que se hace sobre el **RDD** y "**sample**" que se hace sobre el **dataframe**. "takeSample" requiere que se le indique cuántas muestras debe tomar de forma aleatoria, mientras que "sample" requiere de una fracción del total. Ambos toman como argumento la semilla.*

Si el valor de la semilla siempre es el mismo, se podrán realizar los experimentos con exactamente los mismo valores. Si el valor de la semilla cambia un poco, los valores escogidos serán diferentes. Y esto es muy importante, porque también hay un parámetro que indica si se puede hacer la selección de valores usando valores repetidos, es decir, que varios valores se seleccionan varias veces. El hacer múltiples muestreos con reemplazo de los valores de entrada de modelos de machine learning es llamado *"**Boot Strapping**"*. Cada muestra generará un modelo diferente y es posible hacer una votación entre los diversos modelos encontrados para establecer la predicción final tanto en algoritmos de regresión como en algoritmos de clasificación.

Ejemplo de muestreo en el RDD usando takeSample. El arreglo original tiene 10 elementos. Se muestrean 20 con reemplazo:

```
#Crear RDD
rdd = sc.parallelize(range(0, 10))
rdd.take(10)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
rdd.takeSample(True, 20, 1)
```

```
[0, 6, 3, 4, 3, 1, 3, 7, 3, 5, 3, 0, 0, 9, 6, 5, 7, 9, 4, 7]
```

Ejemplo de muestreo en el RDD usando takeSample. El arreglo original tiene 10 elementos. Se muestrean 5 sin reemplazo:

```
rdd.takeSample(False, 5, 2)
```

```
[5, 9, 3, 4, 6]
```

Ejemplo de muestreo en el RDD usando takeSample. El arreglo original tiene 10 elementos. Se intentan muestrear 15 sin reemplazo. Sólo se obtienen 10:

```
rdd.takeSample(False, 15, 3)
```

```
[1, 5, 6, 0, 9, 4, 7, 2, 8, 3]
```

Ejemplo de muestreo en el RDD usando sample. El arreglo original tiene 10 elementos. Se muestrea el 50% sin establecer la semilla. Todas las muestras son diferentes:

```
rdd.sample(False, 0.5).take(5)
```

```
[0, 1, 3, 4, 8]
```

```
rdd.sample(False, 0.5).take(5)
```

```
[1, 2, 4, 5, 6]
```

```
rdd.sample(False, 0.5).take(5)
```

```
[0, 4, 6, 7]
```

Ejemplo de muestreo en Spark dataframes usando sample. El dataframe original tiene 10 elementos. Se muestrea el 50% con semilla 1. Todas las muestras son diferentes:

```
#Crear un Spark dataframe
df = spark.range(10)
df.show()
```

```
+---+
| id|
+---+
|  0|
|  1|
|  2|
|  3|
|  4|
|  5|
|  6|
|  7|
|  8|
|  9|
+---+
```

```
df.sample(withReplacement=False, fraction=0.5, seed=1).show()
```

```
+---+
| id|
+---+
|  0|
|  2|
|  3|
|  5|
|  6|
|  7|
```

```
|  8|
|  9|
+---+
```

Deberían ser 5, pero se entregan 8.

Ejemplo de muestreo en Spark dataframes usando ”sample”. El dataframe original tiene 10 elementos. Se muestrea el 50% con semilla 1. El reemplazo es verdadero, puede haber repeticiones:

```
df.sample(withReplacement=True, fraction=0.5, seed=1).show()
```

```
+---+
| id|
+---+
|  0|
|  2|
|  3|
|  4|
+---+
```

Ejemplo de muestreo en Spark dataframes usando sample. El dataframe original tiene 10 elementos. Se muestrea el 50% con semillas 2 y 3. Todas las muestras son diferentes:

```
df.sample(withReplacement=False, fraction=0.5, seed=2).show()
```

```
+---+
| id|
+---+
|  0|
|  1|
|  2|
|  3|
|  4|
|  5|
|  7|
+---+
```

```
df.sample(withReplacement=False, fraction=0.5, seed=3).show()
```

```
+---+
| id|
+---+
|  0|
|  2|
|  9|
+---+
```

NOTA: La selección de valores es probabilística, por la que no siempre se cumple con la fracción, sobre todo en muestras pequeñas.

Análisis de Datos Gráfico

Hemos visto como determinar las características de nuestras variables encontrando la media, la varianza, los cuartiles, etc. También hemos comenzado a establecer posibles relaciones entre las variables por medio de la correlación.

Ahora visualizamos las principales características de las variables y estableceremos las relaciones más importantes por medio de diversas gráficas.

Estas gráficas ya las conoces. En la mayoría de los casos utilizaremos Matplotly, por lo que no explicaremos las características de las gráficas en sí. Sólo como incorporar Spark con Matplotly.

Lo primero que hay que hacer es seleccionar la información de interés y pasar los Spark RDD y Dataframes a arreglos Numpy. Estos arreglos serán alimentados a las funciones gráficas.

Cargamos dataset con ***`daily_weather.csv`***

```
path_csv="/content/daily_weather.csv"
df = sqlContext.read.csv(path_csv,header=True,inferSchema=True)
df.show(10)
```

En el siguiente código seleccionamos la variable "air_temp_9am" y eliminamos los valores inválidos:

```
df_air_temp_9am=df.select("air_temp_9am")
df_air_temp_9am=df_air_temp_9am.na.drop()
```

Ahora accedemos al RDD para obtener el arreglo de valores y recolectarlo en el nodo central:

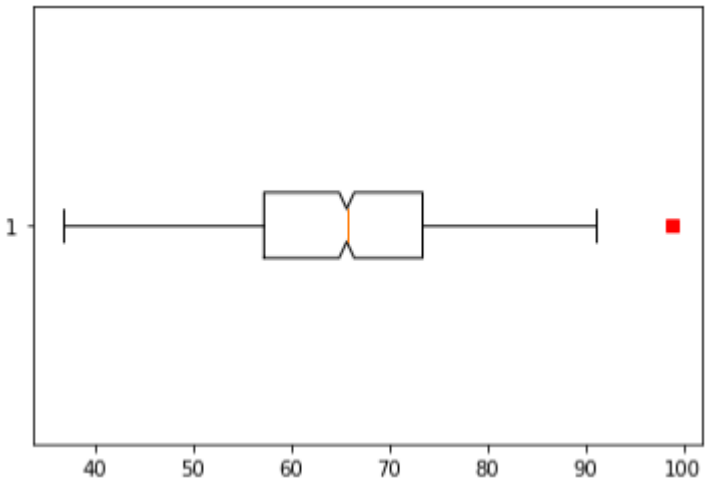
```
df_air_temp_9am=df_air_temp_9am.rdd.map(lambda row : row.air_temp_9am)
arr_air_temp_9am=df_air_temp_9am.collect()
df_air_temp_9am.take(10)
```

```
[74.822000,
 71.403843,
 60.638000,
 70.138895,
 44.294000,
 78.404000,
 70.043304,
 51.710000,
 80.582000,
 47.498000]
```

Nuestra primer gráfica es un Boxplot o gráfico de caja, que indica la media, los cuartiles y los valores extremos (cuadrados rojos o "rs"). El valor 0 indica que se realice con "cintura". Los comentarios indican diversas formas de graficar el Boxplot:

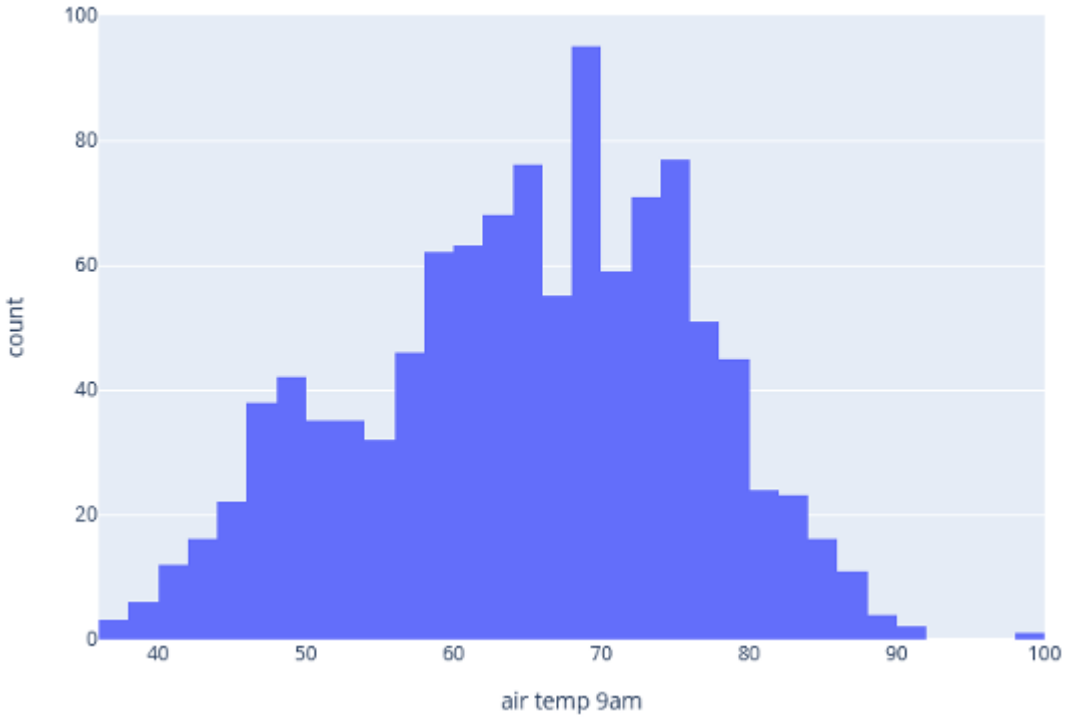
```
import matplotlib.pyplot as plt
plt.boxplot(arr_air_temp_9am,1,"rs",0)

#normal
#plt.boxplot(arr_air_temp_9am)
#con cintura=1, sin cintura=0
#plt.boxplot(arr_air_temp_9am,1)
#cambiar los puntos extremos a "green diamonds"
#plt.boxplot(arr_air_temp_9am,1,"gD")
#horizontal=0, vertical=1
plt.show()
```



En el siguiente ejemplo, creamos un histograma de los valores con Plotly Express:

```
import plotly.express as px
fig = px.histogram(x=arr_air_temp_9am,labels={'x':'air temp 9am', 'y':'Frecuencia'})
#fig = px.histogram(arr_air_temp_9am,nbins = 10)
fig.show()
```



Es posible también mostrar varias gráficas de cajas donde cada gráfica representa valores separados usando algún criterio. Por ejemplo, si queremos graficas la humedad relativa a las 9am

cuando la temperatura a las 9am es mayor a 65 Fahrenheit (18°), lo podríamos hacer con SQL. ***Por ejemplo:***

```
df_nona=df.na.drop()
df_nona.createOrReplaceTempView("Weather")
query='SELECT relative_humidity_9am from Weather where air_temp_9am>65'
df_people_names = spark.sql(query)
df_people_names.show(10)
```

```
+--------------------+
|relative_humidity_9am|
+--------------------+
|    42.42000000000046|
|   24.328697291802207|
|   12.189101868764444|
|   35.130000000000265|
|   10.657421657822635|
|    29.58000000000045|
|   22.070000000000228|
|   15.189999999999962|
|   12.110889335294598|
|   21.031461768790646|
+--------------------+
only showing top 10 rows
```

También podríamos hacerlos usando "withColumn". En el siguiente código se agrega una columna en la que se tiene True si la temperatura es mayor a 65 Fahrenheit y False si no. Después se cuenta las incidencias de cada una. Separar los datos es sólo cuestión de ver cuales renglones son falsos y cuales verdaderos separando la columns "relative_humidity_9am":

```
from pyspark.sql.functions import col
df_nona=df.na.drop()
# 2 Apply the transformation and add it to the DataFrame
df_nona = df_nona.withColumn("air_temp_by_rel_hum", col("air_temp_9am")>65)
#Separate True and False in air_temp_by_rel_hum
df_grouped_rel_hum = df_nona.groupby("air_temp_by_rel_hum")
df_grouped_rel_hum.count().show()
```

En el siguiente ejemplo vamos a separar los valores de humedad relativa a las 9am cuando la tempera a las 9am es superior o inferior a 65 grados y vamos a crear arreglos para graficarlos cada uno en su gráfico de caja:

```
#Divide data
df_gt_65=df_nona.filter(df_nona["air_temp_9am"]>65)
df_lt_65=df_nona.filter(df_nona["air_temp_9am"]<=65)

#Extract variable of interest
df_gt65_humidity=df_gt_65.select("relative_humidity_9am")
df_lt65_humidity=df_lt_65.select("relative_humidity_9am")

#Convert to array
df_gt65_humidity=df_gt65_humidity.rdd.map(lambda row : row.relative_humidity_9am)
arr_gt65_humidity=df_gt65_humidity.collect()
df_lt65_humidity=df_lt65_humidity.rdd.map(lambda row : row.relative_humidity_9am)
arr_lt65_humidity=df_lt65_humidity.collect()

#Boxplots
fig, ax = plt.subplots()
ax.boxplot(arr_gt65_humidity,0,"gD",positions = [1], widths = 0.6)
ax.boxplot(arr_lt65_humidity,0,"bx",positions = [2], widths = 0.6)

ax.set_xticklabels(['air temp 9am > 65', 'air temp 9am < 65'])
ax.set_ylabel('% Humidity')
ax.set_title('air humidity 9am')
```

En el siguiente ejemplo crearemos un diagrama de dispersión en el que graficaremos la temperatura a las 9am contra la humedad a las 9am. Primero verificamos que tengan la misma cantidad de valores:

```
#Now we develop a scatter plot
#Select data
df_nona=df.na.drop()
```

```
df_temp_9am=df_nona.select("air_temp_9am")
df_humidity_9am=df_nona.select("relative_humidity_9am")
print(df_temp_9am.count(),df_humidity_9am.count())
```

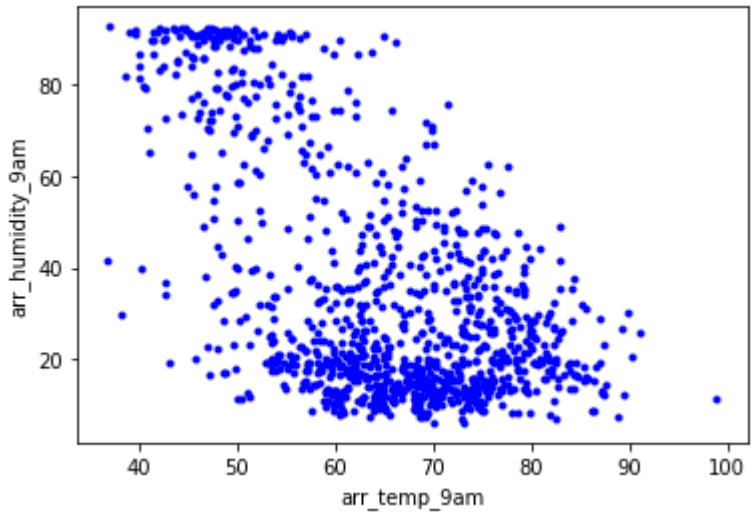
```
1064 1064
```

Ahora creamos los arreglos:

```
df_temp_9am=df_temp_9am.rdd.map(lambda row : row.air_temp_9am)
df_humidity_9am=df_humidity_9am.rdd.map(lambda row : row.relative_humidity_9am)
arr_temp_9am=df_temp_9am.collect()
arr_humidity_9am=df_humidity_9am.collect()
```

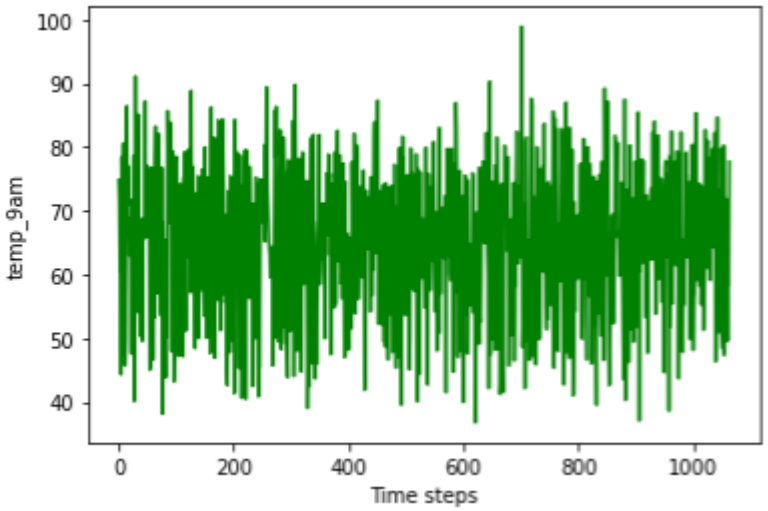
Y ahora graficamos:

```
#Plot
plt.plot(arr_temp_9am,arr_humidity_9am,'b.')
plt.xlabel("arr_temp_9am")
plt.ylabel("arr_humidity_9am")
plt.show()
```



En el siguiente ejemplo, aprovechando que ya tenemos el arreglo con los datos, graficamos la temperatura del aire a las 9am como una serie de tiempo:

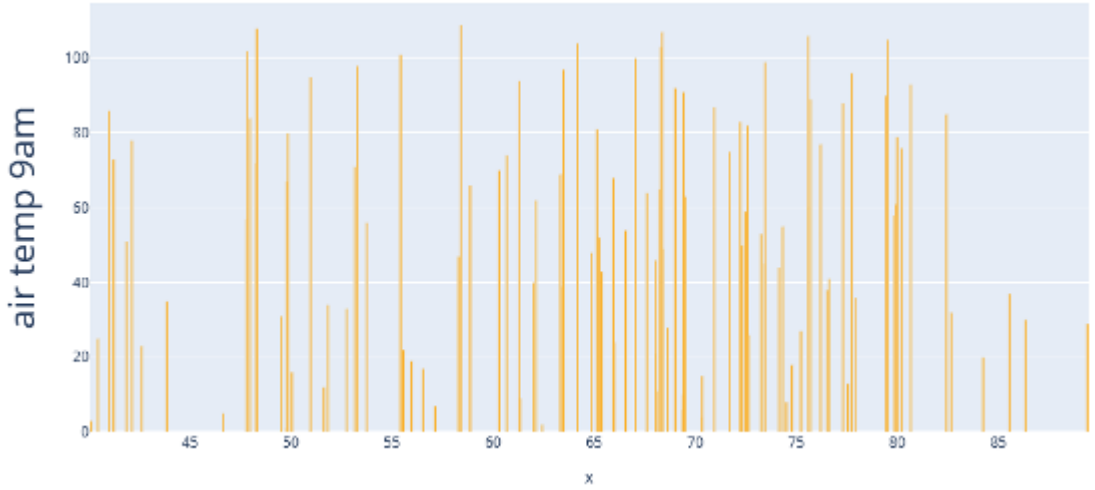
```
#Line plot
plt.plot(arr_temp_9am,'g')
plt.xlabel("Time steps")
plt.ylabel("temp_9am")
plt.show()
```



En el siguiente ejemplo graficamos la misma variable también como serie de tiempo pero usando una gráfica de barras:

```
#Bar plot
import plotly.express as px
fig=px.bar(x=arr_temp_9am)
fig.update_traces(marker_color='red',width=1.0)
fig.update_layout(
   autosize=False,
   width=1000,
   height=500,
   yaxis=dict(
      title_text="air temp 9am",
      ticktext=["Time"],
      tickmode="array",
      titlefont=dict(size=30),
   )
)
```

```
fig.show()
```



En este último ejemplo haremos una gráfica en 3D usando las variables "air_temp_9am, "relative_humidity_9am" y "air_pressure_9am":

```
#Create 3D Scatter plot
#Extract info
df_nona=df.na.drop()
df_nona=df_nona.sample(False,0.1)
df_temp_9am=df_nona.select("air_temp_9am")
df_humidity_9am=df_nona.select("relative_humidity_9am")
df_pressure_9am=df_nona.select("air_pressure_9am")

#Create arrays
df_temp_9am=df_temp_9am.rdd.map(lambda row : row.air_temp_9am)
df_humidity_9am=df_humidity_9am.rdd.map(lambda row : row.relative_humidity_9am)
df_pressure_9am=df_pressure_9am.rdd.map(lambda row : row.air_pressure_9am)
arr_temp_9am=df_temp_9am.collect()
arr_humidity_9am=df_humidity_9am.collect()
arr_pressure_9am=df_pressure_9am.collect()

#Plot 3D
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

```
ax.scatter(arr_temp_9am, arr_humidity_9am, arr_pressure_9am, marker='o')

ax.set_xlabel("air_temp_9am")
ax.set_ylabel("relative_humidity_9am")
ax.set_zlabel("air_pressure_9am")

plt.show()
```