Conceptos para entender el funcionamiento:

Jupyter Notebook

Jupyter Notebook es una aplicación web de código abierto. Cada desarrollador puede dividir el código en partes y trabajar en ellas sin importar el orden que se suele utilizar para visualizar datos en big data y data science

Dataframe

Un DataFrame es una estructura de datos tabular bidimensional en la que los datos se organizan en filas y columnas. Es una forma de representar y manipular datos en la programación, especialmente en entornos como Python con bibliotecas como Pandas.

Apache Spark:

Es un motor de analíticas para procesar datos con módulos SQL, streaming y aprendizaje automático. Spark se compone de 5 componentes claves, para este proyecto se utilizaron principalmente 2: Spark Core, el motor para procesar datos y Spark SQL, el componente para hacer consultas con bases de datos.

Descripción del sistema:

El sistema está conformado por 3 partes un Jupyter Notebook, una base de datos postgresSQL y el software Spark**:**

El Jupyter Notebook está realizado en python con uso de las librerías pandas y utilizando una sesion de spark este Jupyter Notebook lo que realiza es leer los datos de 2 documentos que manejan estadísticas de crímenes en costa rica, una proveniente del OIJ y otra proveniente del INEC.

Estos datos para que sean más fáciles de manipular se insertan en dataframes, en los que reciben manipulaciones para facilitar el análisis, estas modificaciones son: eliminar espacios en blanco, eliminar mayúsculas, eliminar las tildes y modificar algunos datos para que tenian diferencias entre los archivos(ejemplo ‘cañas’ del archivo INEC se modificó a ‘canas’ porque asi salia en el archivo del OIJ)

El Jupyter Notebook también tiene la funcionalidad de devolver una lista o la cantidad de distritos del conjunto de datos del OIJ que no coinciden con ningún distrito del conjunto de datos del INEC o viceversa(si así se desea)
Por último el Jupyter Notebook se conecta a la base de datos postgresSQL y escribe los dataframes del archivo del OIJ y del archivo INEC en la base de datos

(Es importante recalcar que el Jupyter Notebook no es el que realiza los procedimientos si no que es python solo que esta es una herramienta que facilita el proceso)

Gráficas con spark:

Lo primero que debemos hacer es cargar los datos de nuestros dataframes a una base de datos, en nuestro caso usamos PGAdmin para gestionar la base de datos SQL en donde almacenaremos la información, para esto usamos la función **write_spark_df_to_db** con la diferencia de que enviaremos directamente el dataframe a las tablas designadas:

```
write_spark_df_to_db(df_inec_spark, "INEC")
write_spark_df_to_db(df_oij_spark, "OIJ")
```

Descripción de las funciones:

Obtener distritos sin coincidencia:

```
from pyspark.sql.functions import col


def obtener_distritos_sin_coincidencia(df_1, df_2):
    distritos_df_1 = df_1.select('Distrito')
    distritos_df_2 = df_2.select('Distrito')


    distritos_no_coincidentes = distritos_df_1.subtract(distritos_df_2)


    distritos_no_coincidentes_list = [row.Distrito for row in
distritos_no_coincidentes.collect()]
    return distritos_no_coincidentes_list
```

Esta función se encarga de comparar dos conjuntos de datos representados por los DataFrames df_1 y df_2, con lo que se busca encontrar los distritos presentes en df_1 que no tienen correspondencia con los distritos de df_2.

Al inicio, la función selecciona la columna Distrito de cada DataFrame, creando dos nuevos conjuntos de datos que contienen únicamente esa información, distritos_df_1 para df_1 y distritos_df_2 para df_2.

Seguidamente, se emplea el método subtract para encontrar la diferencia entre los conjuntos de distritos distritos_df_1 y distritos_df_2. Esta operación genera un nuevo DataFrame que contiene los distritos presentes en distritos_df_1 pero ausentes en distritos_df_2.

Después, se transforma el DataFrame de distritos no coincidentes en una lista llamada distritos_no_coincidentes_list. Para finalmente, devolver dicha lista como resultado de la función.

Contar distritos sin coincidencia:

```
def contar_distritos_sin_coincidencia(df_1, df_2):
    distritos_df_1 = df_1.select('Distrito')
    distritos_df_2 = df_2.select('Distrito')


    distritos_no_coincidentes = distritos_df_1.subtract(distritos_df_2)


    cantidad_distritos_no_coincidentes =
distritos_no_coincidentes.count()
    return cantidad_distritos_no_coincidentes
```

Esta función se encarga de calcular la cantidad de distritos presentes en `df_1` que no coinciden con los distritos de `df_2`.
En primer lugar, al igual que en la función anterior, se selecciona la columna 'Distrito' de cada DataFrame, generando dos nuevos conjuntos de datos, `distritos_df_1` y `distritos_df_2`, respectivamente. Esto simplifica la comparación al enfocarse únicamente en la información relevante para este análisis.
Después, se utiliza el método `subtract` para encontrar la diferencia entre los conjuntos de distritos de `df_1` y `df_2`, lo que resulta en un nuevo DataFrame llamado `distritos_no_coincidentes`. Este DataFrame contiene los distritos presentes en `df_1` que no están en `df_2`.
Luego, se utiliza el método `count` en el DataFrame `distritos_no_coincidentes` para obtener el número total de registros (distritos) que cumplen esta condición de falta de coincidencia. La función devuelve este conteo, que representa la cantidad de distritos en `df_1` que no coinciden con los presentes en `df_2`. Este enfoque es útil para obtener una medida cuantitativa precisa de la discrepancia entre dos conjuntos de datos específicos en relación con una columna particular.

Quitar tildes:

```
from pyspark.sql.functions import col, regexp_replace


def quitar_tildes(dataframe, nombreColumna):
    caracteres_con_tilde = ["á", "é", "í", "ó", "ú", "ü"]
    caracteres_sin_tilde = ["a", "e", "i", "o", "u", "u"]
    expr = col(nombreColumna)
    for c_tilde, c_sin_tilde in zip(caracteres_con_tilde,
caracteres_sin_tilde):
    expr = regexp_replace(expr, c_tilde, c_sin_tilde)
```

```
  dataframe_sin_tildes = dataframe.withColumn(nombreColumna, expr)

return dataframe_sin_tildes
```

Esta función se encarga de eliminar las tildes de una columna específica en un DataFrame de Spark.
Para ello, primeramente, se definen dos listas, una con los caracteres que tienen tilde (`caracteres_con_tilde`) y otra con sus equivalentes sin tilde (`caracteres_sin_tilde`). Estas listas contienen las correspondencias de caracteres a reemplazar.
Se utiliza la función `regexp_replace` de Spark para reemplazar cada caracter con tilde por su equivalente sin tilde en la columna especificada del DataFrame. El bucle `for` se encarga de construir dinámicamente una expresión para reemplazar cada caracter, iterando sobre las listas de caracteres con y sin tilde.
La función crea una nueva columna en el DataFrame original, donde se han eliminado las tildes de la columna especificada. Esto se hace mediante `withColumn`, asignando el resultado de los reemplazos a la misma columna que se procesa.
Finalmente, la función devuelve el DataFrame modificado, eliminando las tildes de la columna indicada. Este tipo de función es útil para normalizar los datos y asegurarse de que los caracteres acentuados no generen problemas al realizar búsquedas o comparaciones en el DataFrame.

Estandarizar nombres de los distritos:

```
from pyspark.sql.functions import col
df_inec_corregido =

df_inec_spark.withColumn("Distrito2",when(col("Distrito")=='lapalmera',
'palmera')\
    .when(col("Distrito")=='laasuncion','asuncion')\
       .when(col("Distrito")=='latigra','tigra')\
           .when(col("Distrito")=='cañas','canas')\
               .when(col("Distrito")=='lagranja','granja')\
                   .when(col("Distrito")=='elgeneral','general')\
                       .when(col("Distrito")=='tapesco','tapezco')\


.when(col("Distrito")=='juanviñas','juanvinas')\


.when(col("Distrito")=='lafortuna','fortuna')\


.when(col("Distrito")=='cañonegro','canonegro')\
```

```
.when(col("Distrito")=='cañasdulces','canasdulces')
                                                \
                                                    .otherwise(col("Distrito")))


df_inec_corregido.show(5)
```

Este fragmento de código utiliza la función `withColumn` de Spark para crear una nueva columna llamada "Distrito2" en un DataFrame llamado `df_inec_spark`. Esta nueva columna es una corrección de la columna "Distrito", donde se han modificado los nombres de los distritos del INEC, para que coincidan con los del OIJ.

La función `when` y `otherwise` se utilizan para aplicar condiciones. En este caso, se verifica el valor de la columna "Distrito" y se reemplaza por un nuevo valor si coincide con los valores especificados en las condiciones (`'lapalmera'`, `'laasuncion'`, `'latigra'`, etc.). Si el valor no coincide con ninguno de estos, se conserva el mismo valor original de la columna
"Distrito".

Por ejemplo, si el valor de "Distrito" es `'lapalmera'`, se crea una nueva fila en la columna "Distrito2" con el valor `'palmera'`. Lo cual se repite para cada uno de los valores especificados en las condiciones.

Finalmente, se muestra una vista previa de los primeros cinco registros del DataFrame `df_inec_corregido` para verificar los cambios realizados en la columna "Distrito2".

Eliminar espacios:

```
def columna_sin_espacios(dataframe, nombreColumna):
    dataframe = dataframe.withColumn(nombreColumna,
regexp_replace(dataframe[nombreColumna], " ", ""))
    return dataframe
```

Esta función utiliza regex para reemplazar una expresión literal en una columna, la cual es un espacio por un carácter nulo, es decir lo elimina.

- Entradas:
- dataframe: El data frame donde se va a hacer la modificación.
- nombre Columna:La columna donde se hará la modificación.
- Return: El data frame modificado.

Convertir a minusculas:

```
def columna_a_minusculas(dataframe, nombreColumna):


dataframe=dataframe.withColumn(nombreColumna,lower(dataframe[nombreColu
mna]))
  return dataframe
```

Esta función utiliza una función proporcionada por spark para reemplazar convertir las palabras de una columna a minúsculas. l ❖ Entradas:

- ❖ dataframe: El data frame donde se va a hacer la modificación. ❖ nombre Columna:La columna donde se hará la modificación. ❖ Return: El data frame modificado.

Documentación de las visualizaciones:

Para la graficación de los datos utilizamos la librería MatPlotLib, la cual permite tomar los datos dentro de un dataframe Spark o Pandas y luego hacer la representación visual utilizando las funciones pertenecientes a la librería. Para utilizar la librería debemos asegurarnos de utilizar el siguiente código:

```
pip install matplotlib
```

```
import matplotlib.pyplot as plt
```

- **Gráfica #1:**

  Para realizar esta gráfica se tuvo que crear dos nuevas columnas llamadas TotalDelitos y CantonYDistrito

```
from pyspark.sql.functions import concat_ws
df_oij_spark_corrected =columna_a_minusculas(df_oij_spark,'Distrito')
df_oij_spark_corrected= columna_sin_espacios(df_oij_spark_corrected,'Distrito')
df_oij_spark_corrected= columna_sin_espacios(df_oij_spark_corrected,'Canton')
df_oij_spark_corrected =columna_a_minusculas(df_oij_spark_corrected,'Canton')
df_oij_spark_corrected= df_oij_spark_corrected.withColumn("CantonYDistrito", concat_ws("-","Distrito","Canton"))
df_oij_spark_corrected.show(5)
```

```
df_inec_spark_corrected =columna_a_minusculas(df_inec_spark,'Distrito')
df_inec_spark_corrected= columna_sin_espacios(df_inec_spark_corrected,'Distrito')
df_inec_spark_corrected= columna_sin_espacios(df_inec_spark_corrected,'Canton')
df_inec_spark_corrected =columna_a_minusculas(df_inec_spark_corrected,'Canton')
df_inec_spark_corrected= df_inec_spark_corrected.withColumn("CantonYDistrito", concat_ws("-","Distrito","Canton"))
df_inec_spark_corrected.show(5)
```

```
totalDelitos =spark.sql("SELECT CantonYDistrito as CantonYDistritoOIJ, COUNT(*) as TotalDelitos FROM OIJ GROUP BY CantonYDistrito")
```

Una vez creadas esas tablas, se hizo un Join para unir los datos de los Dataframes de INEC y OIJ, esto con el fin de poder relacionar los datos de ambas en relación al distrito

```
df__ejercicio_1 = df_inec_spark_corrected.join(totalDelitos,df_inec_spark_corrected['CantonYDistrito']==totalDelitos["CantonYDistritoOIJ"],'inner')
```

También se creó una tabla temporal a la cual se pueden hacer consultas

```
df__ejercicio_1.createOrReplaceTempView("Cantidad_Delitos")
```

Como para la gráfica se pidió únicamente analizar los datos de los 10 distritos con más delitos reportados entonces se realizó una función que devolviera el DF que contenga estos 10 distritos

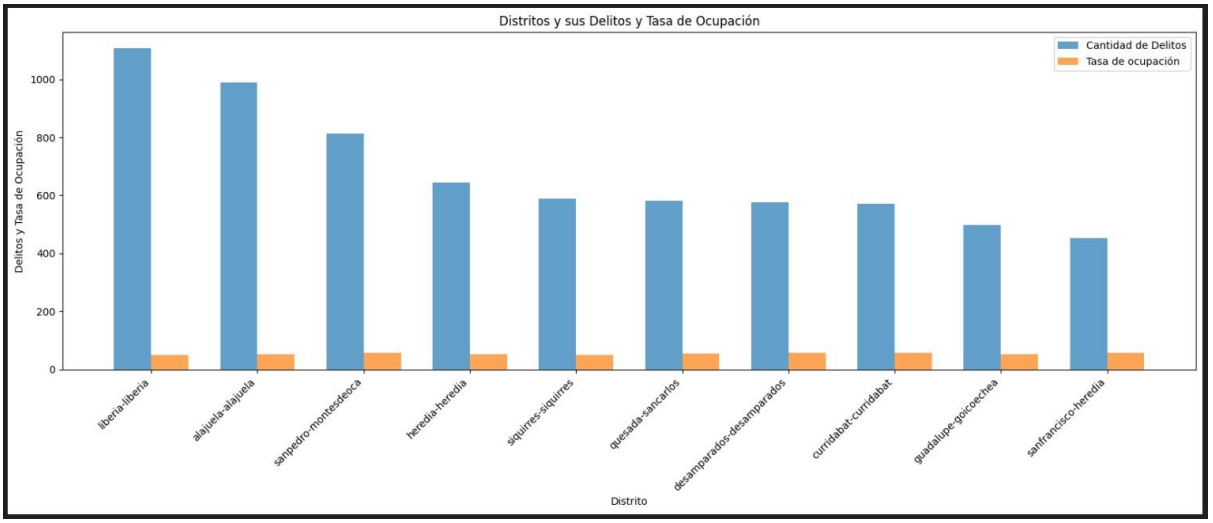
```
df__ejercicio_1_top10= spark.sql("Select * from Cantidad_Delitos ORDER BY TotalDelitos DESC LIMIT 10")
```

Seguido, pasamos el DF a pandas para poder obtener mejores resultados en la graficación.

```
df__ejercicio_1_pandas= df__ejercicio_1_top10.toPandas()
```

Con este DF en pandas, creamos los índices para el eje X, especificamos el grosor de las barras y el tamaño de la figura para luego crear las dos barras que representarán los datos a graficar. En estas barras estará la Cantidad de delitos cometidos por distrito y la tasa de ocupación relacionada a ese mismo distrito.

```
import numpy as np

# Assuming you have a Pandas DataFrame named df__ejercicio_1_pandas

# Create an array of indices for the x-axis
x_indices = np.arange(len(df__ejercicio_1_pandas['CantonYDistrito']))

# Set the width of the bars
bar_width = 0.35

# Plotting the bars side by side
plt.figure(figsize=(20, 6))
bar1 = plt.bar(x_indices, df__ejercicio_1_pandas["TotalDelitos"], width=bar_width, label="Cantidad de Delitos", align='center', alpha=0.7)
bar2 = plt.bar(x_indices + bar_width, df__ejercicio_1_pandas["Tasa de ocupación"], width=bar_width, label="Tasa de ocupación", align='center', alpha=0.7)

# Set the x-axis ticks and labels
plt.xticks(x_indices + bar_width / 2, df__ejercicio_1_pandas['CantonYDistrito'], rotation=45, ha="right")

# Set labels and

# Set labels and title
plt.xlabel("Distrito")
plt.ylabel("Delitos y Tasa de Ocupación")
plt.title("Distritos y sus Delitos y Tasa de Ocupación")
plt.legend()

# Show the plot
plt.show()
```



- **Gráfica #2:**

Para crear esta gráfica primero se transformaron las fechas del DF original en un objeto Datetime.

```
#Convertimos la fecha en un Datedatime
df_ejercicio_2["Fecha"]= pd.to_datetime(df_ejercicio_2["Fecha"],format='%m/%d/%Y')
```

✓ 0.0s

Posterior a eso con la función day_name, obtuvimos en qué día de la semana era esa fecha, y añadimos una columna con esa información.

```
#Con pandas obtenemos el dia de la semana
df_ejercicio_2['DiaSemana']=df_ejercicio_2['Fecha'].dt.day_name()
df_ejercicio_2.head()
```

✓ 0.0s

	Delito	SubDelito	Fecha	Hora	Victima	SubVictima	Edad	Genero	Nacionalidad	Provincia	Canton	Distrito	CantonYDistrito	DiaSemana
0	ASALTO	ARMA BLANCA	2011-01-10	18:00:00 - 20:59:59	PERSONA	PEATON [PERSONA]	Mayor de edad	HOMBRE	NICARAGUA	SAN JOSE	alajuelita	sanfelipe	sanfelipe-alajuelita	Monday
1	ASALTO	ARMA BLANCA	2011-02-02	03:00:00 - 05:59:59	PERSONA	PEATON [PERSONA]	Mayor de edad	HOMBRE	COSTA RICA	CARTAGO	oreamuno	sanrafael	sanrafael-oreamuno	Wednesday
2	ASALTO	ARMA BLANCA	2011-10-23	21:00:00 - 23:59:59	PERSONA	PEATON [PERSONA]	Mayor de edad	HOMBRE	COSTA RICA	HEREDIA	heredia	sanfrancisco	sanfrancisco-heredia	Sunday
3	ASALTO	ARMA BLANCA	2011-05-19	18:00:00 - 20:59:59	PERSONA	PEATON [PERSONA]	Mayor de edad	MUJER	COSTA RICA	HEREDIA	heredia	heredia	heredia-heredia	Thursday
4	ASALTO	ARMA BLANCA	2011-02-02	03:00:00 - 05:59:59	PERSONA	PEATON [PERSONA]	Mayor de edad	HOMBRE	COSTA RICA	GUANACASTE	liberia	liberia	liberia-liberia	Wednesday

Despues se sumaron los delitos que ocurrieron en ese distritoen ese día de la semana.

```
#Hacemos un agrupamiento con pandas
df_ejercicio_2_sumado = df_ejercicio_2.groupby(["CantonYDistrito","DiaSemana"]).size().reset_index(name="CantidadDelitos")
df_ejercicio_2_sumado.head(10)
```

✓ 0.0s

	CantonYDistrito	DiaSemana	CantidadDelitos
0	acapulco-puntarenas	Friday	2
1	acapulco-puntarenas	Tuesday	1
2	aguabuena-cotobrus	Friday	1
3	aguabuena-cotobrus	Monday	2
4	aguabuena-cotobrus	Saturday	3
5	aguabuena-cotobrus	Sunday	3
6	aguabuena-cotobrus	Thursday	3
7	aguabuena-cotobrus	Tuesday	3
8	aguabuena-cotobrus	Wednesday	4
9	aguasclaras-upala	Friday	5

Luego utilizando el DF creado en el ejercicio 1, filtramos el DF nuevo por distrito, de manera que coincidiera el cantón y distrito con el del primer elemento del DF del ejercicio 1.

```
#Distrito con mas delitos
distrito = df__ejercicio_1_pandas.iloc[0]
#Filtrar el df
df_ejercicio_2_por_distrito= df_ejercicio_2_sumado[df_ejercicio_2_sumado["CantonYDistrito"]==distrito["CantonYDistrito"]]
print(df_ejercicio_2_por_distrito)
```

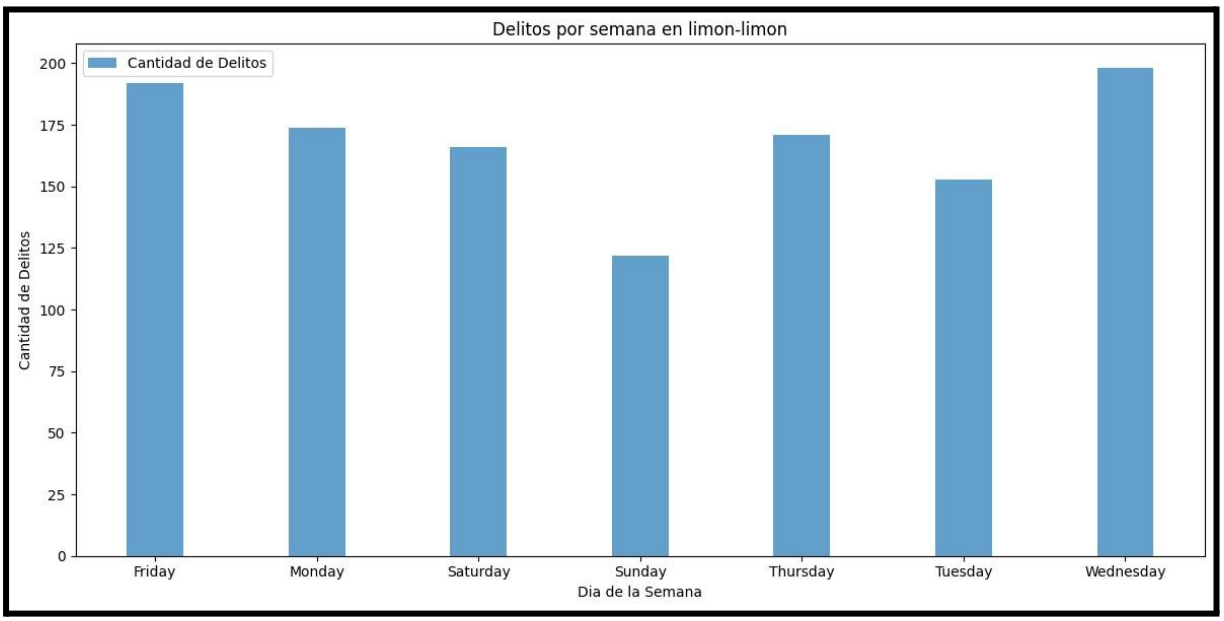
✓ 0.0s

```
     CantonYDistrito  DiaSemana  CantidadDelitos
1146     limon-limon     Friday              192
1147     limon-limon     Monday              174
1148     limon-limon   Saturday              166
1149     limon-limon     Sunday              122
1150     limon-limon   Thursday              171
1151     limon-limon    Tuesday              153
1152     limon-limon  Wednesday              198
```

Finalmente la graficación fue idéntica a la del ejercicio 1, con 1 barra menos.

```python
import matplotlib.pyplot as plt
import numpy as np


x_indices = np.arange(len(df_ejercicio_2_por_distrito['DiaSemana']))

bar_width = 0.35


plt.figure(figsize=(12, 6))
bar1 = plt.bar(x_indices, df_ejercicio_2_por_distrito["CantidadDelitos"], width=bar_width, label="Cantidad de Delitos", align='center', alpha=0.7)

plt.xticks(x_indices, df_ejercicio_2_por_distrito['DiaSemana'])




plt.xlabel("Dia de la Semana")
plt.ylabel("Cantidad de Delitos")
plt.title("Delitos por semana en " +df_ejercicio_2_por_distrito.iloc[0]["CantonYDistrito"] )
plt.legend()


plt.tight_layout()

plt.show()
```

✓ 0.1s



- **Gráfica #3 :**

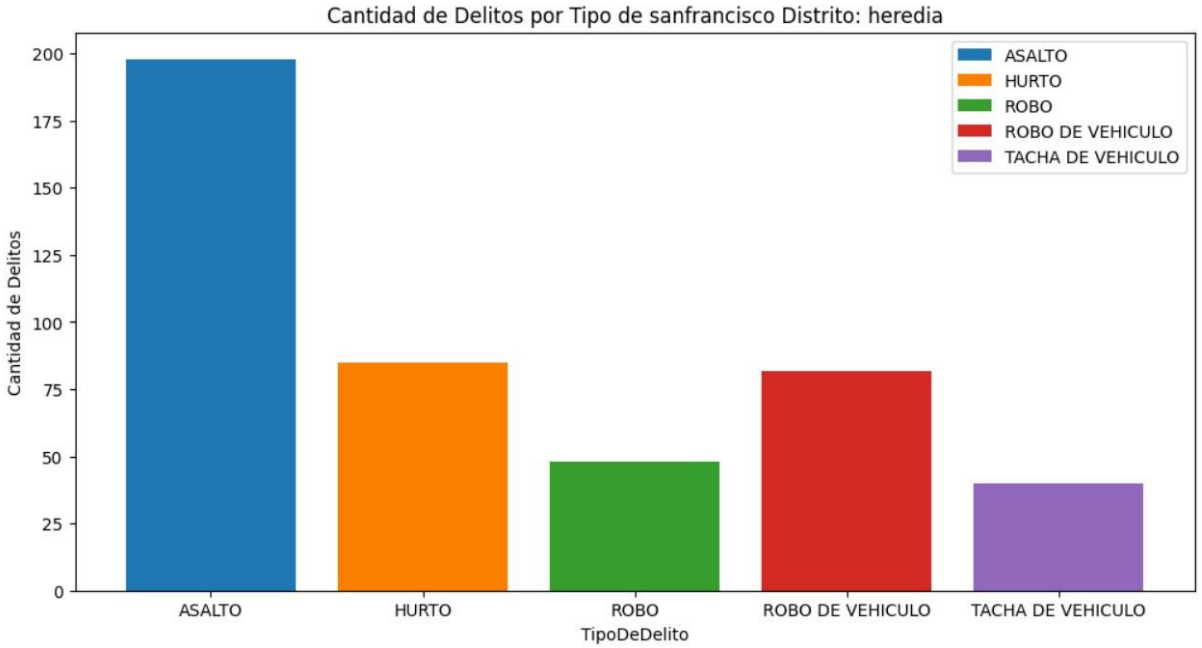
  Para la realización de la gráfica 3 como el objetivo era obtener la cantidad de tipos de crímenes de un distrito en específico, se crearon 2 variables, una para seleccionar el distrito del que se desea obtener la información y otra para el cantón al que pertenece

```python
#Aquí se puede seleccionar el distrito y el canton que se desea graficar
CantonConsulta3 = "sanfrancisco"
DistritoConsulta3 = "heredia"

TipoDelitosDistrito =spark.sql("SELECT CantonYDistrito, Delito AS TipoDeDelito, COUNT(*) AS CantidadDeDelitos FROM OIJ where CantonYDistrito = \
                               '"+CantonConsulta3+"-"+DistritoConsulta3+"'\
                                   GROUP BY CantonYDistrito, Delito ORDER BY CantonYDistrito,Delito")
```

  Esas variables se introducen en la una consulta SQL para poder obtener los datos solicitados

```
TipoDelitosDistrito_Pandas = TipoDelitosDistrito.toPandas()

plt.figure(figsize=(12, 6))
for tipo_delito, group in TipoDelitosDistrito_Pandas.groupby('TipoDeDelito'):
    plt.bar(group['TipoDeDelito'], group['CantidadDeDelitos'], label=tipo_delito)

plt.xlabel('TipoDeDelito')
plt.ylabel('Cantidad de Delitos')
plt.title("Cantidad de Delitos por Tipo de " + CantonConsulta3 +" Distrito: "+ DistritoConsulta3)
plt.legend()
plt.show()
```

Y para finalizar se pasa a Pandas para poder crear la visualización de datos.



- **Gráfica #4 :**

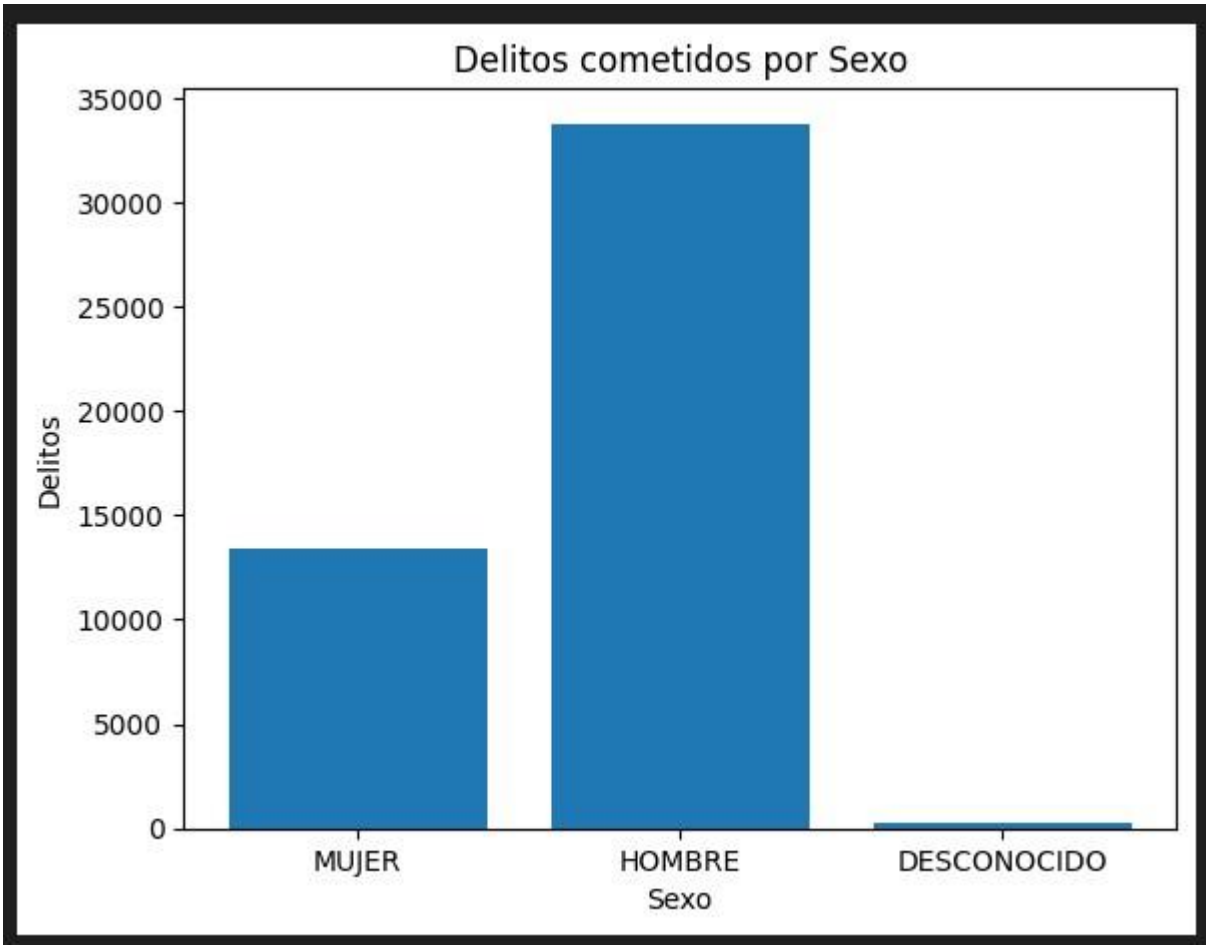
  Esta es posiblemente la más sencilla, ya que no involucra datos de más de un dataframe y únicamente se necesita realizar un conteo. Para esto se utilizará el dataframe existente y se agrupará por género, usando el siguiente código:

```
df_oij_graph5 = df_oij_graph
df_oij_graph5 = df_oij_graph.groupBy("Genero").count()
```

A partir de ahí, únicamente se pasa a Pandas y see genera el gráfico con la librería MatPlotLib similar a como se hizo anteriormente.

```
graph6 = df_oij_graph5.toPandas()
```

```
plt.title("Delitos cometidos por Sexo")
plt.xlabel("Sexo")
plt.ylabel("Delitos")
plt.bar(graph6["Genero"],graph6["count"])
```

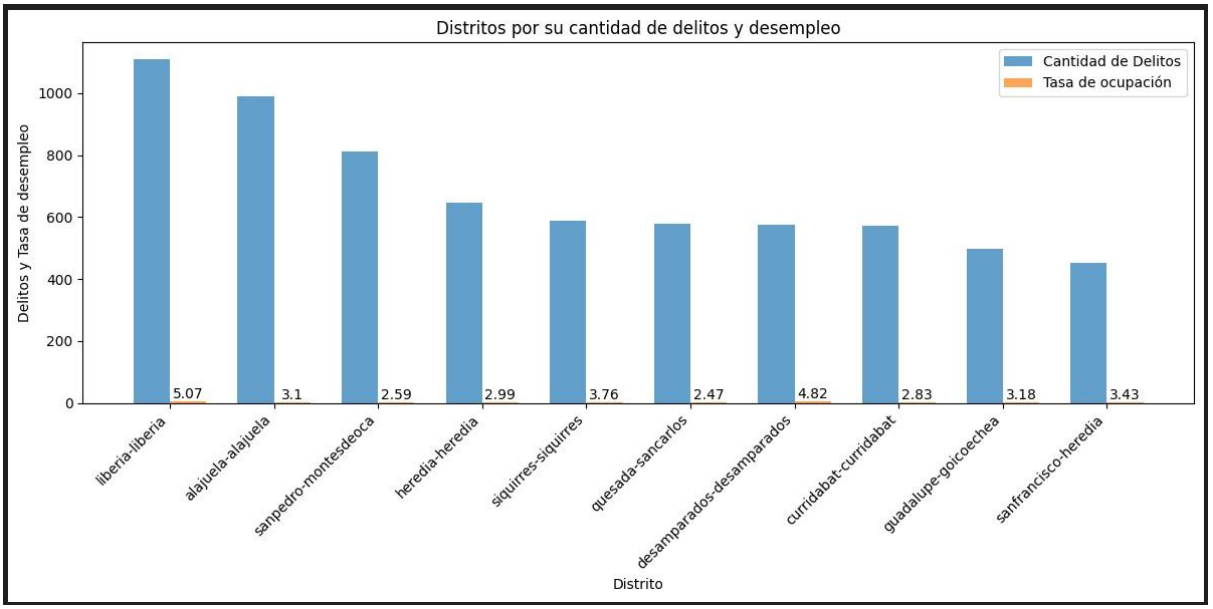


- **Gráfica #5 :**

  La última gráfica realmente realiza el mismo procedimiento que la primera, con la diferencia de que nos enfocamos en el índice de desempleo en vez del índice de ocupación.

```
plt.figure(figsize=(12, 6))
bar1 = plt.bar(x_indices, df__ejercicio_1_pandas["TotalDelitos"], width=bar_width, label="Cantidad de Delitos", align='center', alpha=0.7)
bar2 = plt.bar(x_indices + bar_width, df__ejercicio_1_pandas["Tasa de desempleo abierto"], width=bar_width, label="Tasa de ocupación", align='center', alpha=0.7)
```

  La razón por la que se realizó de esta manera, es porque llegó a nuestra mente una creencia común que sugiere que entre mayor sea el desempleo en un lugar, mayor será su criminalidad. Lo cual en este caso se evidencia gráficamente.



Conclusiones:

- Es muy importante generar los datos de negocio bien organizados de manera que el análisis sea sencillo. De esta forma se ahorran muchos recursos a la hora de la carga y transformación de datos, además de que no necesariamente la misma persona que haga el análisis va a ser la misma que genere el informe o reporte de datos, por lo cual lo mejor es hacerlo lo más organizado posible.
- Aún con datos organizados, el proceso de estandarización es esencial para el análisis, esto debido a que distintas fuentes tienen distintas reglas, como lo pueden ser la codificación de caracteres. Ser consciente de limpiar los datos antes de su uso es importante, limpiar caracteres especiales y en algunos casos, aunque no haya sido en el proyecto, transformar las palabras a su raíz etimológica.
- Cuando se quiere demostrar o verificar algo, si no hay 1 sola fuente que contenga la información necesaria, con una buena manipulación de datos se puede lograr sacar hipótesis de distintas fuentes confiables. Sabiendo realizar transformaciones y limpiezas y teniendo en cuenta inconsistencias entre las fuentes, es posible analizar muchos datos que son públicos pero no analizados.
- Manipular la información antes de procesarla no necesariamente debería limitarse a código, hay transformaciones que son más sencillas realizarlas en herramientas como excel, que pueden ahorrarnos muchas molestias a la hora de cargar los DF en pandas.
- A la hora de cargar los archivos, es importante entender las estructuras de los mismos, de manera que podamos realizar las transformaciones u omisiones necesarias. Por ejemplo, algunos datos pueden decir que son excel pero estar construidos en html.

Referencias:

https://medium.com/grabngoinfo/install-pyspark-3-on-google-colab-the-easy-way-577ec4a2bcd8

CÓMO INSTALAR APACHE SPARK (PYSPARK) en Windows 10 | Big Data en Python…

Install Apache PySpark on Windows PC | Apache Spark Installation Guide

Instituto Nacional de Estadísticas y Censos (2011). Censo 2011: Indicadores económicos, según provincia, cantón y distrito. Recuperado de https://admin.inec.cr/sites/default/files/media/reempleocenso2011-22.xls_2.xls Organismo de Investigación Judicial (2018). Estadísticas policiales. Recuperado de https://sitiooij.poder-judicial.go.cr/index.php/apertura/transparencia/estadisticas-policiales Suhong, K., Param, J., Parminder, K. y Pooya, T. (2018). Crime Analysis Through Machine Learning. IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). Seleccionar del 1 de enero 2011 al 31 de diciembre 2011.

https://spark.apache.org/docs/latest