PROYECTO FINAL CURSO

Clase Product Development

Sección V

Integrantes: Javier Navarro

Yalmar Cardenas

Marvin Donis

Victor Borrayo

Margerys Salgado

Objetivo: evaluar todos los conocimientos adquiridos por los estudiantes a lo largo del semestre, por lo que el proyecto consistirá en lo siguiente: Docker, Streamlit, y Dashboard.

Data: los datos provistos contienen información sobre los contagios, las recuperaciones y las muertes, dados por país y región.

Desarrollo del Proyecto

Servicios del Docker Container

A continuación, se enlistan los servicios y su configuración el el archivo docker-compose.yml:

- **Base de Datos MySQL:** Se configura una base de datos en MySQL que almacenará el registro de casos, la configuración es la siguiente:

```yaml
db:
    image: mysql:5.7
    volumes:
        #- ./db_data:/var/lib/mysql
        - ./script/test_covid.sql:/docker-entrypoint-initdb.d/1.sql
    restart: always
    ports:
        - 3306:3306
    environment:
        MYSQL_ROOT_PASSWORD: test123
        MYSQL_DATABASE: test
        MYSQL_USER: test
        MYSQL_PASSWORD: test123
```

- **Webserver:** El web server será el encargado de ejecutar todos los procesos de Airflow y se deberá configurar de la siguiente manera, el mismo, requiere configuración de una carpeta que será monitoreada para actualización de la base de datos, la carpeta /home/airflow/monitor:

```yaml
webserver:
    build: .
    restart: always
    depends_on:
        - postgres
    environment:
        - LOAD_EX=n
        - EXECUTOR=Local
    logging:
        options:
            max-size: 10m
            max-file: "3"
    volumes:
        - ./dags:/usr/local/airflow/dags
        - ./monitor:/home/airflow/monitor
        # - ./plugins:/usr/local/airflow/plugins
    ports:
        - "8080:8080"
    command: webserver
    #command: sh -c "/loadconfig.sh && exec webserver"
    healthcheck:
        test: ["CMD-SHELL", "[ -f /usr/local/airflow/airflow-webserver.pid ]
        interval: 30s
        timeout: 30s
        retries: 3
```

- **Streamlit:** El servicio de streamlit se conecta al puerto 8501 de la máquina local para poder acceder a la aplicación de streamlit desde un navegador web y se configura el volumen que contendrá el código de la app:

```yaml
streamlit:
    build: .
    restart: always
    volumes:
        - ./app:/usr/local/airflow/app
    ports:
        - "8501:8501"
```

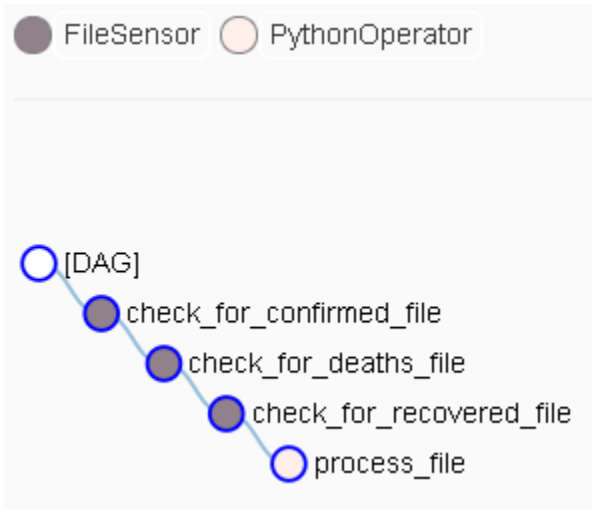
Construcción del Docker Container

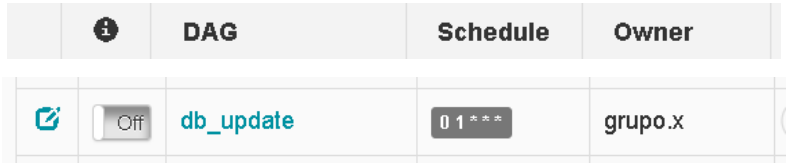
Para construir el docker container, una vez que se disponga del directorio principal, ejecutar el archivo Init.bat si la máquina local es Windows y Init.sh si la máquina es Linux.

DAG para Actualización de Base de Datos

El DAG utilizado para la actualización de la base de datos consiste en tres procesos conformados por un sensorFile cada uno, destinados a los archivos de casos confirmados, muertes y recuperados,una vez detectados los tres archivos, se ejecuta el último proceso del DAG, el cuál utiliza una función para realizar un pivote en las tablas y aplicar el formato requerido para la base de datos de MySQL, una vez están en el formato correcto, se realizan tres cargas a la base de datos y se procede a eliminar los archivos de la carpeta monitor,

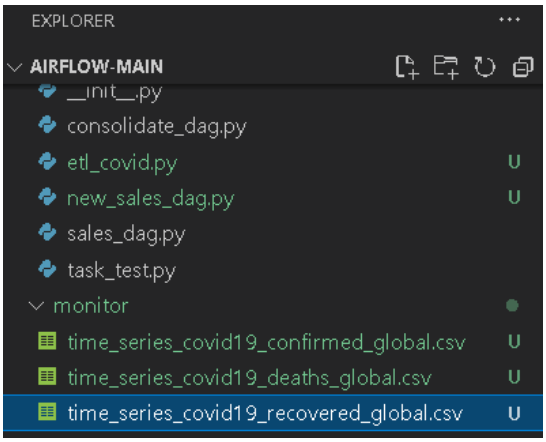
```
dag = DAG('db_update', description='Actualización base de datos COVID',
          default_args={
              'owner': 'grupo.x',
              'depends_on_past': False,
              'max_active_runs': 1,
              'start_date': days_ago(1)
          },
          schedule_interval='0 1 * * *',
          catchup=False)
```





- **Carga de Archivos**

Los archivos deben depositarse en la carpeta: /monitor, para que los 3 sensorFile los detecten y permitan al DAG pasar al siguiente proceso.

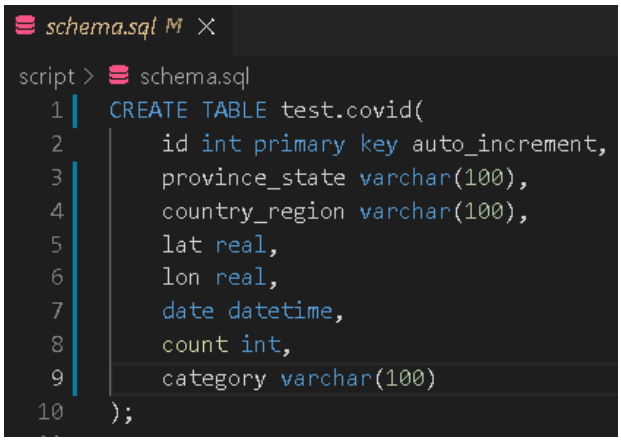


- **Procesamiento de los Archivos**

El procesamiento de archivos consiste en el pivote con la función stack de pandas para crear una columna de Fecha y la adición de una columna de Categoría para definir si son casos confirmados, muertes o recuperados, adicional a otras transformaciones menores como el formato de fecha.

- **Carga a la Base de Datos**

La carga a la base de datos utiliza la conexión de MySQL para ingresar cada uno de los dataframes resultantes a la base de datos en modalidad append, esto luego de haber ejecutado un delete que vacía la base de datos para evitar duplicados, finalmente los archivos son removidos del directorio para la próxima actualización.



Desarrollo del App Web con Streamlit

Streamlit es un Open-Source de Python que hace fácil la creación de aplicaciones web para mostrar resultados de datos estadísticos. Trabajar en Streamlit es relativamente fácil para cualquier usuario, bastara un comando de pocas líneas para ejecutarlo.

```
streamlit run your_script.py [-- script args]
```

Con la ejecución del código “streamlit run [nombre del archivo]” se activará un servidor Streamlit local y su aplicación se abrirá en una nueva pestaña en un navegador web predeterminado.
Para el desarrollo de la aplicación es necesario contar con múltiples archivos, de acuerdo con la necesidad de cada proyecto, por lo anterior, el proyecto presentado a través de este documento cuenta con diferentes tipos de páginas, dentro de las cuales se puede detallar:

- **Graficas.py:** contiene los códigos relacionados a la presentación de los datos a través de imágenes, esto con la finalidad de facilitar al lector de la data su análisis, a través de las distintas variables que la componen; dentro de la misma se puede observar graficas de barra, histogramas y pastel.

```
# ---- Functions ----
def app(df):
    st.markdown("<h1 style='text-align: center;'> Gráficas </h1>", unsafe_allow_html=True)
    col1, col2 = st.columns(2)

    df1 = df.groupby(['date','category']).sum()
    df1.reset_index(inplace=True)
    #st.dataframe(df1)

    fig1 = px.line(df1, x='date', y="count",
                   labels={'count':'Casos','date':'Fecha','category':'Categoria'},
                   line_group='category',
                   color='category',
                   color_discrete_map={
                  'Confirmed': '#F1C40F',
                  'Deaths': '#E74C3C',
                  'Recovered': '#2ECC71'
              })
```

```
    df2 = df.groupby(['country_region','category']).sum()
    df2.reset_index(inplace=True)


    fig2 = px.bar(df2, x='count', y="country_region",
                   labels={'count':'Casos','country_region':'Región','category':'Categoria'},
                   color='category',
                   orientation='h',
                   color_discrete_map={
                  'Confirmed': '#F1C40F',
                  'Deaths': '#E74C3C',
                  'Recovered': '#2ECC71'
              })
```

```
fig3 = px.pie(df2, values='count', names='category',
              labels={'count':'Casos','country_region':'Región','category':'Categoría'},
              color='category',
              color_discrete_map={
             'Confirmed': '#F1C40F',
             'Deaths': '#E74C3C',
             'Recovered': '#2ECC71'
         })

fig4 = px.histogram(df1, x="count",
                    labels={'count':'Casos','country_region':'Región','category':'Categoría'},
                    color='category',
                    color_discrete_map={
                   'Confirmed': '#F1C40F',
                   'Deaths': '#E74C3C',
                   'Recovered': '#2ECC71'
               })

with col1:
    st.markdown("<h1 style='text-align: center;'> Casos por dia </h1>", unsafe_allow_html=True)
    st.plotly_chart(fig1)
    st.markdown("<h1 style='text-align: center;'> Casos por categoria </h1>", unsafe_allow_html=True)
    st.plotly_chart(fig3)
with col2:
    st.markdown("<h1 style='text-align: center;'> Casos por región </h1>", unsafe_allow_html=True)
    st.plotly_chart(fig2)
    st.markdown("<h1 style='text-align: center;'> Histograma de Casos por Dia </h1>", unsafe_allow_html=True)
    st.plotly_chart(fig4)
```

- **Mapa.py:** contiene la programación necesaria para mostrar al lector una mayor visualización de acuerdo con la ubicación geográfica.

```
# ---- Funciones ----
def app(df2):
    df2.drop("province_state", axis=1, inplace=True)
    st.title("Mapa de COVID-19")
    try:
        df_grouped = df2.groupby(['lon','lat',"category","country_region"]).sum().reset_index().drop(0)
        #df_grouped = df_grouped[df_grouped.count >= 0]
        df_grouped_catetory = df_grouped.category.unique()
        # --- Use pandas to calculate additional data for each point ---
        df_grouped["radius"] = df_grouped["count"]/df_grouped["count"].max()* 80000
        #df_grouped =  df_grouped[df_grouped.radius > 0]
    except:
        pass
```

```
    # --- Define a layer to display on a map ---
    lista = []
    try:
        Confirmed = pdk.Layer(
            "ScatterplotLayer",
            df_grouped[df_grouped["category"] == "Confirmed"],
            pickable=True,
            opacity=0.8,
            stroked=True,
            filled=True,
            radius_scale=6,
            radius_min_pixels=1,
            radius_max_pixels=100,
            line_width_min_pixels=1,
            get_position="[lon, lat]",
            get_radius="radius",
            get_fill_color=[255, 140, 0],
            get_line_color=[0, 0, 0],
        )
```

```
    )
    lista.append(Confirmed)
except:
    pass
try:
    Deaths = pdk.Layer(
        "ScatterplotLayer",
        df_grouped[df_grouped["category"] == "Deaths"],
        pickable=True,
        opacity=0.8,
        stroked=True,
        filled=True,
        radius_scale=6,
        radius_min_pixels=1,
        radius_max_pixels=100,
        line_width_min_pixels=1,
        get_position="[lon, lat]",
        get_radius="radius",
        get_fill_color=[255, 0, 0],
        get_line_color=[0, 0, 0],
    )
```

```
    lista.append(Deaths)
except:
    pass
try:
    Recovered = pdk.Layer(
        "ScatterplotLayer",
        df_grouped[df_grouped["category"] == "Recovered"],
        pickable=True,
        opacity=0.8,
        stroked=True,
        filled=True,
        radius_scale=6,
        radius_min_pixels=1,
        radius_max_pixels=100,
        line_width_min_pixels=1,
        get_position="[lon, lat]",
        get_radius="radius",
        get_fill_color=[0, 255, 0],
        get_line_color=[0, 0, 0],
    )
```

```
    lista.append(Recovered)
except:
    pass

    # --- Set the viewport location and zoom level ---
try:
    view_state = pdk.ViewState(
        latitude=df_grouped.lat.mean(),
        longitude=df_grouped.lon.mean(),
        zoom=2,
        bearing=0,
        pitch=0
    )
except:
    pass

    # --- Render the map to the page ---
try:
    r = pdk.Deck(
        layers=lista,
        initial_view_state=view_state,
        tooltip={"text": "{count} cases\n{category}\n {country_region}"},
    )
```

```
    except:
        pass
    try:
        st.pydeck_chart(r)
    except:
        view_state = pdk.ViewState(
            latitude=14.61,
            longitude=-90.51,
            zoom=5,
            bearing=0,
            pitch=0
        )
        r2 = pdk.Deck(

            initial_view_state=view_state,
        )
        st.pydeck_chart(r2)
```

- **Tabla.py:** es importante en todo análisis poder desplegar la data, por tal razón, se incluye dentro del proyecto los datos en formato tabla, que permiten poder desplegar todas las variables.

```
# ---- Import libraries ----
import streamlit as st

def app(df):
    st.markdown("<h1 style='text-align: center;'> Tablas </h1>", unsafe_allow_html=True)
    st.dataframe(df)
    st.download_button(
       "Descargar tabla",
       df.to_csv(),
       "DATA.csv",
       "text/csv",
       key='download-csv'
       )
```

Adicionalmente, para la funcionalidad de este Open-Source es necesaria la incorporación de archivos importante que contribuyan al proceso, como:

- **App.py:** este archivo contiene todas las especificaciones necesarias para creación de la instancia del App, que permita la presentación de los datos de una manera eficiente y manejable para el usuario final.

```
# ---- formato de la pagina ----
st.set_page_config(layout="wide")

# ---- Custom imports -----
from multipage import MultiPage
from pages import tablas, graficas, mapa

# ---- Connect to data base ----
@st.cache(allow_output_mutation=True)
def getData():
    df = database.connect_to_database("select * from  covid")
    return df
df = getData()
df["count"][df["count"]<0] = 0
#df =df[df["count"]>=0]
```

```python
# ---- Create an instance of the app ----
app = MultiPage.filter(df)

# ---- Add all your application ----

app.add_page("Mapa", mapa.app)
app.add_page("Tablas", tablas.app)
app.add_page("Gráficas", graficas.app)


# ---- The main app ----
app.run()
```

- **Database.py:** para lograr la conexión entre la base de datos y el App, se creo dentro del Open-Source un archivo que contiene detalles y especificaciones de conexión a los datos.

```python
# ---- Import libraries ----
import ...

def connect_to_database(query):
    """
    Connect to a database.
    """
    server = '34.125.174.19'
    database = 'test'
    port = '3306'
    username = 'admin'
    passwd = 'password'
    #driver='{ODBC Driver 17 for SQL Server}'

    # Connect to the database
    cnx = mysql.connector.connect(user=username, password=passwd,
                                  host=server,
                                  database=database, port=port)
    #cnx.close()
    # Consulta a la base de datos
    frase = query
    df= pd.read_sql_query(frase, cnx)
    return df
```

- **Multipage.py:** como se detalló anteriormente, el desarrollo de proyecto requiere de la funcionalidad de todos los archivos que se crean para tal fin, por tal razón se crea el archivo Multipa.py, el cual contiene todas las funciones que hacen necesario la filtración de los datos por cada una de las variables que se desean.

```python
# ---- Import libraries ----
import ...

# ---- Define the multipage class to manage the multiple apps in our program ----
class MultiPage:

    # ---- Contrucctor de la clase ----
    def __init__(self,filter_country,filter_category,df) -> None:
        self.pages = []
        self.filter_country = filter_country
        self.filter_category = filter_category
        self.df = df

    # ---- Metodo para agregar paginas al proyecto ----
    def add_page(self, title, func) -> None:
        self.pages.append(
            {
                "title": title,
                "function": func
            }
        )
```

```python
    # ---- Funcion principal run
    def run(self):
        # ---- Drodown to select the page to run ----
        page = st.sidebar.selectbox(
            'App Navigation',
            self.pages,
            format_func=lambda page: page['title']
        )
        # ---- run the app function ----
        page['function'](self.df)
```

```python
    # ---- Decorador ----
    @classmethod
    def filter(cls,df):
        # ---- Filtro por pais ----
        country = st.sidebar.multiselect(
            'Country filter',
            df["country_region"].unique()
        )
        # ---- Filtro por categoria -----
        category = st.sidebar.multiselect(
            "Category filter",
            df["category"].unique(),
            default=df["category"].unique()
        )
        # ---- Filtro por fecha ----
        start_date = st.sidebar.date_input(
            "Start date",
            min_value=min(df["date"]),
            value=datetime.date(2020,1,23),
        )
        end_date = st.sidebar.date_input(
            "End date",
            min_value=min(df["date"])
        )
        st.sidebar.text(start_date)
```

```python
        df_filtered = df[df["category"].isin(category) &
         df["country_region"].isin(country)
         ]
        df_filtered = df_filtered[(df_filtered["date"].dt.date > start_date) &
                                  (df_filtered["date"].dt.date < end_date)]
        print(df_filtered.dtypes)
        return cls(country,category,df_filtered)
```

Mapa Iterativo

Como parte principal de este Proyecto esta la creación de un Mapa, que permitan la visualización de la totalidad de casos, recuperaciones y muertes, para una fecha y país determinado; proceso detallado a continuación:

- Los mapas están basados en la librería pydeck, la cual permite la generación de mapas en formato html. A su vez, la librería streamlit permite la integración de pydeck con st.pydeck_chart().
- La estructura básica del pydeck_chart consiste en la creación de distintas capas. Para propósito del mapa de covid 19 existían 3 categorías, por lo que las tres capas creadas son: confirmed, deaths y recovered.
- Para establecer el tamaño del punto en el mapa se utiliza una función normalizadora, de modo que se dividan todos los valores dentro del valor máximo para el conjunto de datos filtrado (df_filtered) y se multiplica por una constante para que sea visible en el mapa.

- Se utiliza la función try catch para validar que los filtros excluyan la creación de alguna tapa y se agregan todas las capas creadas a un array de capas.
- Se crea un objeto de tipo ViewState que permite establecer las configuraciones iniciales del mapa, para esto se centra la visualización en el promedio de las coordenadas.
- Para finalizar se utiliza la función pydeck.Deck(), que permite renderizar los elementos creados previamente en un solo mapa, haciendo uso del arreglo de capas y el ViewState.
- Se utiliza la función de streamlit streamlit.pydeck_chart(), la que recibe como argumento el objeto renderizado en el paso anterior.

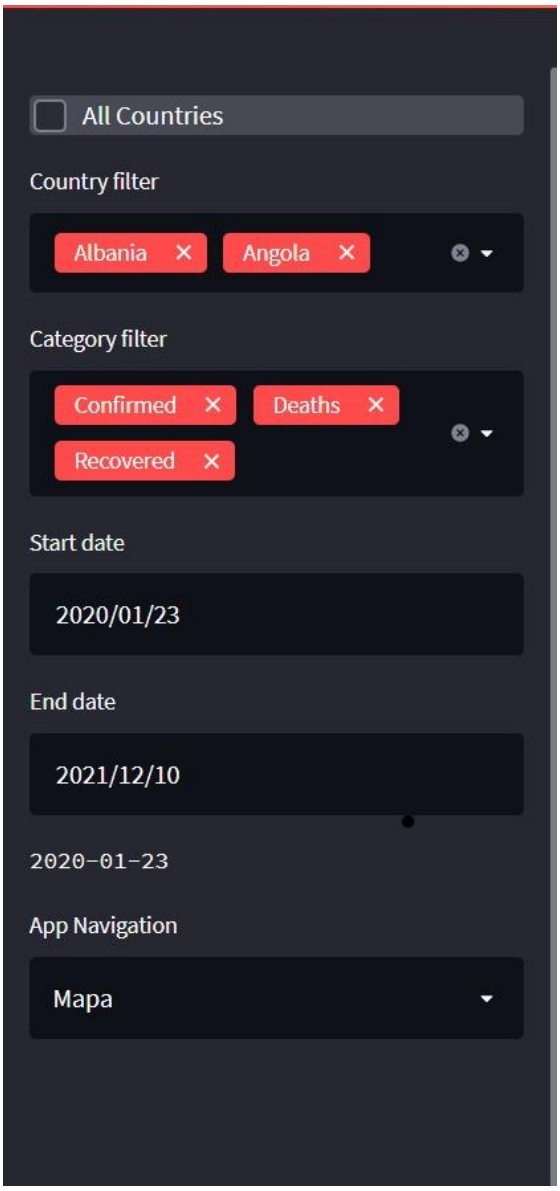
Dashboard

- Finalidades del Dashboard

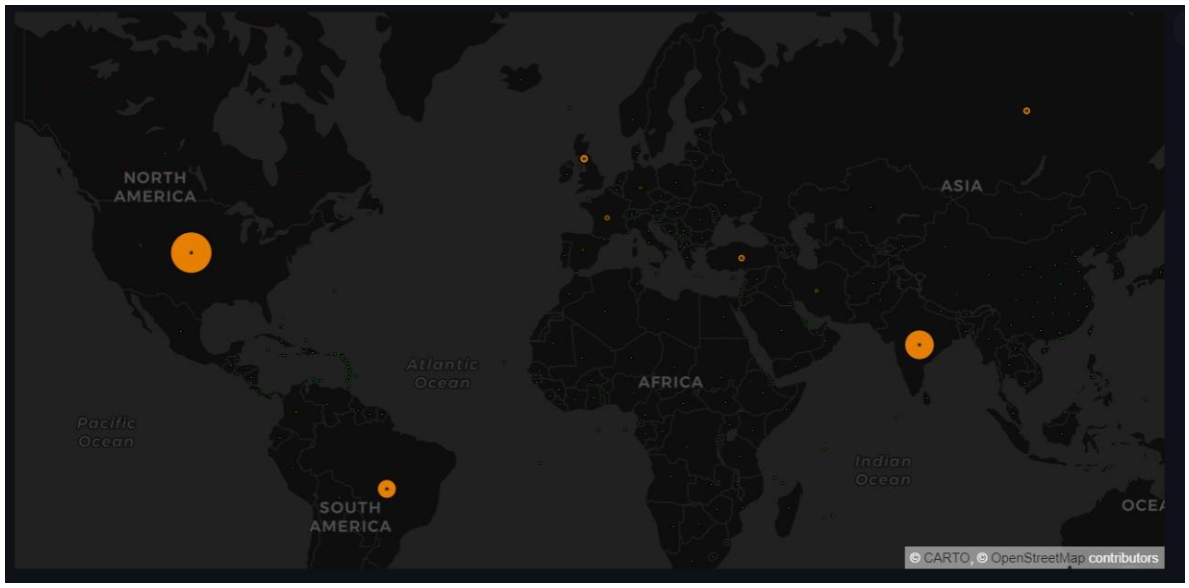
La visualización de datos es la representación gráfica de la información; mediante el uso de elementos visuales, como gráficos, mapas y tablas, que permitan la comunicación de una manera accesible al usuario final de los datos. En la era del big data y el acceso a volúmenes grandes de información, se hace más difícil el análisis e interpretación de los datos, siendo los mismos el elemento más importante dentro del análisis, ya que cuentan historias y los softwares de visualización nos ayudan a graficarlas de forma simple e interactiva.
Por lo anterior, Dashboard o paneles de datos consisten en la consolidación de la representación gráfica de todos aquellos indicadores que permiten la toma de decisiones en una organización.

- Instrucciones de ingreso al Dashboard
  Primeramente, se debe ingresar al siguiente link:
  https://localhost:8080
  Luego, se activa el DAG db_update, finalmente, para ingresar al dashboard primero debe esperar que el primer dag finalice de actualizar la base de datos, una vez ejecutado este proceso deberá ingresar al siguiente link:
  https://localhost:8501
- Especificaciones del Dashboard
  Filtros Generales

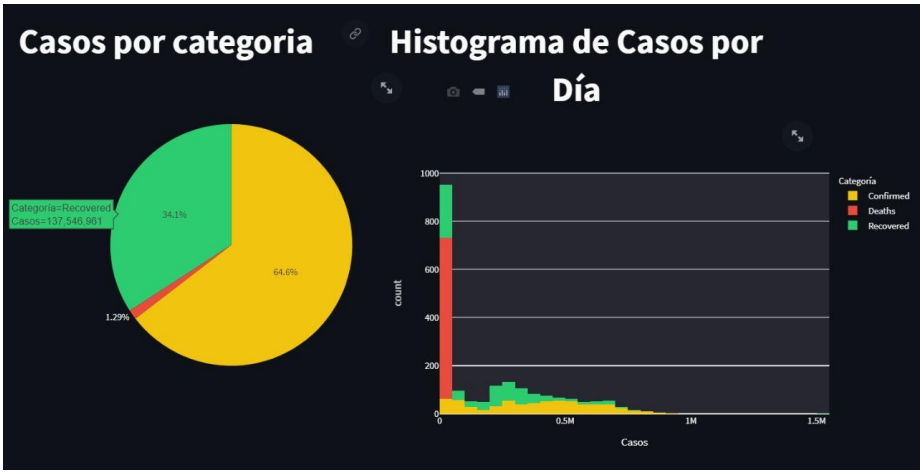
La finalidad del Dashboard es proveer al usuario una herramienta de análisis simple e interactiva, por lo anterior el proyecto que se presenta contiene diferentes barras de selecciones que permiten filtrar los datos, agrupar los datos, de acuerdo con la preferencia de cada usuario para su análisis, dentro de las mismas están:



Adicionalmente, el Dashboard le permite poder visualizar la data de acuerdo con sus preferencias, contempladas en cada una de páginas que se incluyen en el Proyecto presentado, dentro de las mismas están estadística básica, mapas y tablas.


NORTH AMERICA
ASIA
Atlantic Ocean
AFRICA
Pacific Ocean
Indian Ocean
SOUTH AMERICA
CARTO, OpenStreetMap contributors


Gráficas
Casos por dia
Casos por región
Casos
1.4M
1.2M
1M
0.8M
0.6M
0.4M
0.2M
0
Jul 2020
Jan 2021
Fecha
Región
Venezuela
Tunisia
Switzerland
Solomon Islands
San Marino
Poland
North Macedonia
Montenegro
Mali
Liberia
Kenya
Indonesia
Guatemala
Fiji
Dominica
Congo (Kinshasa)
Cambodia
Bolivia
Azerbaijan
Afghanistan
0
20M
40M
60M
Casos


Casos por categoria
Histograma de Casos por Día
Categoría=Recovered
Casos=137 546 961
34.1%
64.6%
1.29%
1000
800
600
400
200
0
count
0
0.5M
1M
1.5M
Casos
Categoría
Confirmed
Deaths
Recovered

Tablas

	id	province_state	country_region	lat	lon	date	count	category
1	554402	<NA>	Afghanistan	33.9391	67.7100	2020-01-24T00:00:00	0	Confirmed
2	554403	<NA>	Afghanistan	33.9391	67.7100	2020-01-25T00:00:00	0	Confirmed
3	554404	<NA>	Afghanistan	33.9391	67.7100	2020-01-26T00:00:00	0	Confirmed
4	554405	<NA>	Afghanistan	33.9391	67.7100	2020-01-27T00:00:00	0	Confirmed
5	554406	<NA>	Afghanistan	33.9391	67.7100	2020-01-28T00:00:00	0	Confirmed
6	554407	<NA>	Afghanistan	33.9391	67.7100	2020-01-29T00:00:00	0	Confirmed
7	554408	<NA>	Afghanistan	33.9391	67.7100	2020-01-30T00:00:00	0	Confirmed
8	554409	<NA>	Afghanistan	33.9391	67.7100	2020-01-31T00:00:00	0	Confirmed
9	554410	<NA>	Afghanistan	33.9391	67.7100	2020-02-01T00:00:00	0	Confirmed
10	554411	<NA>	Afghanistan	33.9391	67.7100	2020-02-02T00:00:00	0	Confirmed

Descargar tabla