Actividad Integradora M6

Keyla Patricia Islas Garrido **A01730349**

Instituto Tecnológico y de Estudios Superiores de Monterrey

Analítica de datos y herramientas de inteligencia artificial II

Gpo 502

Profesor: Fabiola Díaz Nieto

Índice

Resumen **1**

Introducción **1**

Librerías **1**

Plotly 1

Streamlit 2

Método **2**

Resultados **2**

Anexos **2**

Resumen

El Ayuntamiento de San Francisco ha desarrollado una iniciativa llamada DataSF con la misión de potenciar el uso de los datos. Creen que el uso de datos puede mejorar las operaciones y los servicios del Ayuntamiento de San Francisco. Esto, en última instancia, conduce a una mayor calidad de vida y de trabajo para los residentes, empresarios, empleados y visitantes de San Francisco.

El Departamento de Policía de San Francisco ha estado recopilando informes de incidentes desde 2018 hasta 2020. Este conjunto de datos incluye los informes de incidentes policiales presentados por los agentes y por los individuos a través del autoservicio de informes en línea para los casos que no son de emergencia.
Los informes incluidos son los correspondientes a incidentes ocurridos a partir del 1 de enero de 2018 y que han sido aprobados por un oficial supervisor.

El Ayuntamiento de San Francisco junto con el Departamento de Policía quieren desarrollar un mapa interactivo donde los ciudadanos puedan detectar zonas de riesgo u obtener información sobre los delitos que se producen para que puedan tomar precauciones. Además quieren que los ciudadanos ayuden a recoger más información sobre los delitos para alertar al Departamento de Policía e incluso como se hace en Los Ángeles para predecir un delito.

Introducción

En la sección de Anexos podemos encontrar la liga que redirecciona al sitio de descarga de los datos.

Este set de Datos nos presenta información sobre los reportes de incidentes en San Francisco, tenemos a nuestra disposición 36 columnas o atributos y 3,379,513 filas o registros de los cuales hay completos 13,662,468 registros o entradas por lo que hay 108,000,000 valores nulos que deben ser alterados o en dado caso, descartados.

El proceso de ETL (Extracción, Transformación y Limpieza de Datos) se explicará en detalle en la sección Método mediante código.

Librerías

Pandas

```
import pandas as pd
```

Pandas es una herramienta de manipulación y análisis de datos de código abierto rápida, potente, flexible y fácil de usar, construida sobre el lenguaje de programación Python.

Esta librería será usada para el proceso de ETL y para la construcción de sub sets de datos que servirán de base a alguna figura[fig2].

Numpy

```
import numpy as np
```

NumPy ofrece completas funciones matemáticas, generadores de números aleatorios, rutinas de álgebra lineal, transformadas de Fourier y mucho más.

Datetime

```
import datetime
from datetime import date, time, datetime, timedelta
```

El módulo datetime proporciona clases para manipular fechas y horas.
Aunque se admite la aritmética de la fecha y la hora, la implementación se centra en la extracción eficiente de atributos para el formato de salida y la manipulación.
Esta librería se usa para el proceso de Transformación de Datos.

Plotly

```
import plotly.express as px
import plotly.figure_factory as ff
import plotly.graph_objects as go
```

La biblioteca de gráficos Plotly de Python crea gráficos interactivos con calidad de publicación. Entre ellos diagramas de líneas, diagramas de dispersión, gráficos de área, gráficos de barras, barras de error, diagramas de caja, histogramas, mapas de calor, subgráficos, gráficos de ejes múltiples, gráficos polares y gráficos de burbujas.

En este caso usaremos Plotly para ejecutar 1 mapa geográfico [fig] , 1 gráfico de barras [fig2], 2 diagramas de líneas ([fig1], [fig3]), 1 gráfico jerárquico de sunburst [fig4] y un gráfico jerárquico de treemap [fig5].

Streamlit

```
import streamlit as st
```

Streamlit es una biblioteca de código abierto de Python que facilita crear y compartir aplicaciones web personalizadas para el aprendizaje automático y la ciencia de datos. En tan solo unos minutos puede construir y desplegar potentes aplicaciones de datos.

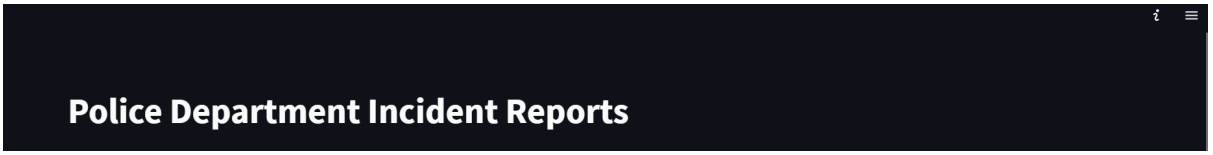
Para la creación de un tablero dinámico se estableció el uso de esta biblioteca en la plataforma de Jupyter en el software de Anaconda.

Método

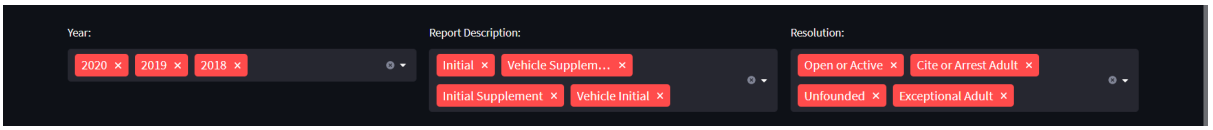
```
#Extracción de Datos.
#El csv utilizado se encuentra en la sección de Anexos.
#Se usa la librería de Pandas para leer el archivo y cargar los datos al entorno de ejecución.
```

```
df = pd.read_csv('Police_Department_Incident_Reports__2018_to_Present.csv')

#Limpieza de Datos
#Borramos aquellos atributos que tengan más del 75% de sus registros con valores nulos
df.dropna(axis = 1, thresh = int(len(df)*.75), inplace = True)
#Atributos eliminados:
#Filed Online
#HSOC Zones as of 2018-06-05
#OWED Public Spaces
#Central Market/Tenderloin Boundary Polygon - Updated
#Parks Alliance CPSI (27+TL sites)
#ESNCAG - Boundary File
#A partir de este punto hay dos opciones para seguir con la limpieza de datos sin manipular la información.
#Borrar los valores nulos sin tomar en cuenta antes la información dada sobre la base de datos por lo que queda una dataframe de 285237 rows × 30 columns
#O Eliminar los atributos que no están definidos en el diccionario de variables que son: 'SF Find Neighborhoods','Current Police Districts','Current Supervisor Districts','Analysis Neighborhoods','Areas of Vulnerability, 2016'
#Se opta por la segunda opción.
df.drop(['SF Find Neighborhoods','Current Police Districts','Current Supervisor Districts','Analysis Neighborhoods','Areas of Vulnerability, 2016'], axis = 1, inplace = True)
df.dropna(inplace =True)

#Transformación de Datos.
#A continuación se usa las librerías Datetime y Pandas para la transformación de algunas variables que se leyeron como tipo texto pero son referentes al tiempo
#Pasar variables correspondientes tipo texto a tipo fecha
df['Incident Datetime'] = pd.to_datetime(df['Incident Datetime'], format = '%Y/%m/%d %I:%M:%S %p')
df['Incident Date'] = pd.to_datetime(df['Incident Date'], format = '%Y/%m/%d')
h = []
for i in df.index:
  h.append(datetime.time(datetime.strptime(df.loc[i,'Incident Time'], '%H:%M')))
df['Incident Time'] = h
df['Report Datetime'] = pd.to_datetime(df['Report Datetime'], format = '%Y/%m/%d %I:%M:%S %p')

#Cambios que se requerirán para gráficos posteriores
df['diff'] =df['Report Datetime'] - df['Incident Datetime']
df['contador'] = 1

#Cambio requisito para establecer los filtros
df['Incident Datetime'] = df['Incident Datetime'].astype('M8[ms]').astype('O')
```

```
df['Incident Date'] = df['Incident Date'].astype('M8[D]').astype('O')
df['Report Datetime'] = df['Report Datetime'].astype('M8[ms]').astype('O')

#Empezamos con la creación de la app con la librería de Streamlit
#Se especifica que se usará la anchura de la pantalla sin importar el tamaño por lo que los objetos en la app podrán ajustarse
st.set_page_config(layout = 'wide')
#Se establece el título principal del Dashboard
```




```
st.title ("Police Department Incident Reports")

#Creación de los Filtros
#Se crean listas con los valores únicos correspondientes a las columnas categóricas elegidas como filtros. Se establecerán 5 filtros.
Year = df['Incident Year'].unique().tolist()
Descripcion = df['Report Type Description'].unique().tolist()
Solucion = df['Resolution'].unique().tolist()
Vecindario = df['Analysis Neighborhood'].unique().tolist()
Fecha = df['Incident Date'].unique()

#Integración de los filtros al Dashboard. Estos se encontrarán en la parte superior, debajo del título.
#Se crea la primera fila con tres objetos correspondientes a los filtros de Año, Descripción del Incidente y Estado del Reporte
```



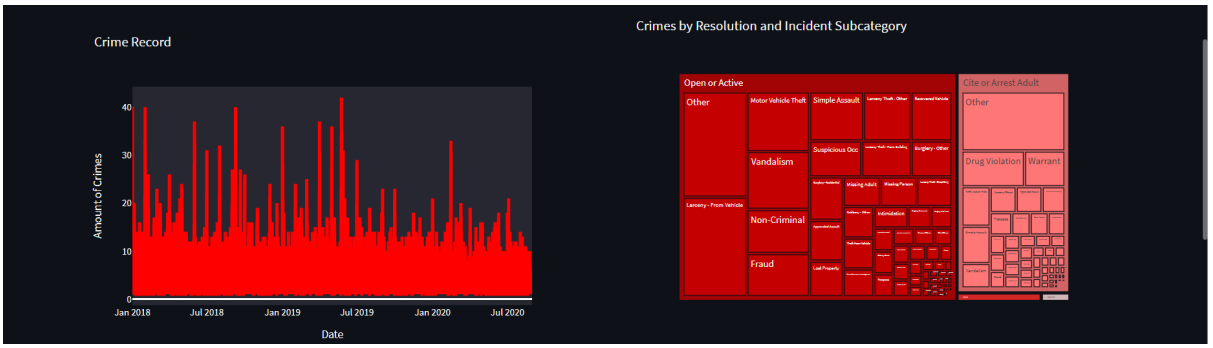

```
#Definición de los contenedores con st.columns()
a1,a2,a3 = st.columns(3)
#Asignación de los objetos a los contenedores
with a1:
    Year_s = st.multiselect('Year: ',
                    Year,
                    default = Year)
with a2:
    Descripcion_s = st.multiselect('Report Description: ',
                    Descripcion,
                    default = Descripcion)
with a3:
    Solucion_s = st.multiselect('Resolution: ',
                    Solucion,
```

```
                    default = Solucion)

#Se crea la segunda fila con dos objetos correspondientes a los filtros por Fecha y Vecindario
```


```
b1,b2 = st.columns((1,3))
with b2:
    Vecindario_s = st.multiselect('Neighborhood: ',
                    Vecindario,
                    default = Vecindario)
with b1:
    Fecha_s = st.date_input('Date: ',value=(Fecha.min(),Fecha.max()))

#Se crea una máscara que contiene la manipulación de los filtros por parte del cliente para que todos los gráficos respondan correctamente
mask = (df['Incident Year'].isin(Year_s)) & (df['Report Type Description'].isin(Descripcion_s)) & (df['Resolution'].isin(Solucion_s))& (df['Analysis Neighborhood'].isin(Vecindario_s))& (df['Incident Date'].between(*Fecha_s))


#Las 6 figuras presentes fueron hechas con la librería Plotly.

#GRAF 1 (fig)
#Es un mapa que muestra dónde ocurrieron los incidentes de acuerdo a la Latitud y Longitud.
#Cuando el señalador pasa sobre los puntos se muestra el Id del reporte, el Vecindario, las calles que intersectan el lugar del incidente, el distrito policiaco y la descripción del Incidente a manera de tooltip/label.
fig = px.scatter_mapbox(df[mask],
                    lat="Latitude",
                    lon="Longitude",
                    hover_name="Row ID",
                        hover_data=['Analysis Neighborhood','Intersection','Police District','Report Type Description'],
                    color_discrete_sequence=["red"],
                    zoom=11,
                    height=500)
fig.update_layout(mapbox_style="stamen-toner")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.update_layout(title='San Francisco Crime Map')
```

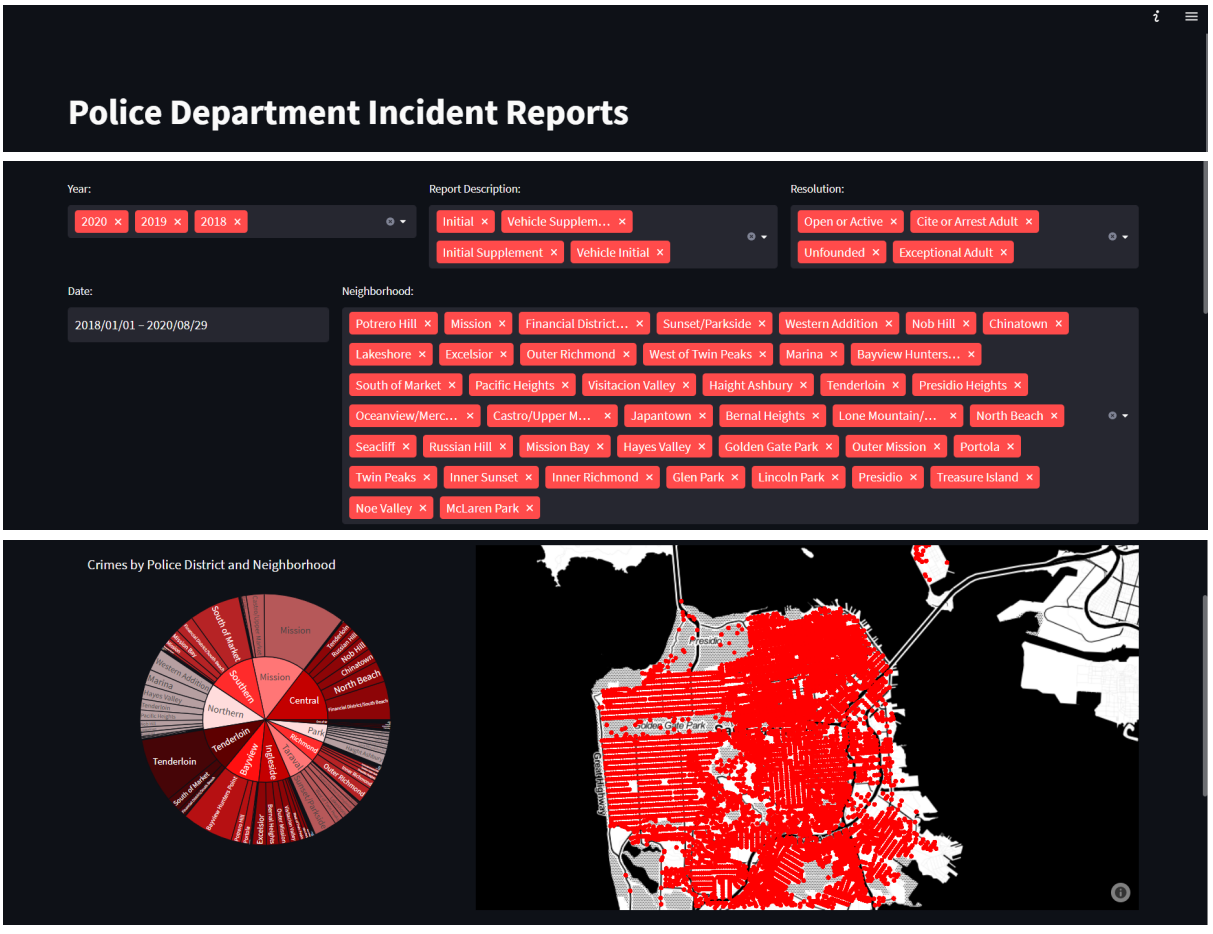
```
#GRAF 2 (fig1)
#Es un diagrama de líneas que muestra la cantidad promedio de reportes de incidentes por Hora del día (Hora-Minuto-Segundo)
#Como tooltip/label se muestra (HORA DEL DÍA, INCIDENTES PROMEDIO)
fig1 = go.Figure()
fig1.add_trace(go.Scatter(x=df[mask]['Incident Time'].sort_values().unique(), y=df[mask].groupby(['Incident Time']).mean()['Row ID'], line=dict(color='red', width=4)))
fig1.update_layout(title='Cases throughout the day', xaxis_title='Time', yaxis_title='Average Cases')
fig1.update_xaxes(showgrid=False)
fig1.update_yaxes(showgrid=False)

#GRAF 3 (fig2)
#Con la librería pandas se crea un subconjunto que agrupa las variables de Día de la Semana con el Estado del Reporte y regresa la media en días que se tardo en levantar el reporte desde que sucedió el incidente.
df1 = pd.DataFrame({'count' : df[mask].groupby( ['Incident Day of Week','Resolution'] )['diff'].mean()}).reset_index()
df1['Incident Day of Week'] = pd.Categorical(df1['Incident Day of Week'], ["Monday", "Tuesday", 'Wednesday', 'Thursday','Friday','Saturday', "Sunday"])
df1.sort_values("Incident Day of Week", inplace = True)
#Es una gráfica de barras que muestra el promedio de días de diferencia entre el levantamiento del reporte y la fecha del incidente por día categorizando al mismo tiempo con el Estado del Reporte.
fig2 = px.bar(df1,
             x="Incident Day of Week",
             y=((df1['count']/60)/60)/24,
             color="Resolution",
             color_discrete_sequence = ['#c40000','#e00110','#ff3030','#ff7676'],
               title = "Days of Difference: Report and Incident by Day of the Week and Incident Resolution")
fig2.update_layout(yaxis_title='Days of Difference between Report and Incident', xaxis_title='Day of the Week')


#GRAF 4 (fig3)
#Es un diagrama de líneas que muestra un conteo de Incidentes a través del tiempo: Historial de Incidentes de Crimen
fig3 = go.Figure()
fig3.add_trace(go.Scatter(x=df[mask].sort_values(by=['Incident Datetime'])['Incident Datetime'].unique(), y=df[mask].groupby(['Incident Datetime']).count()['Row ID'], line=dict(color='red', width=4)))
```

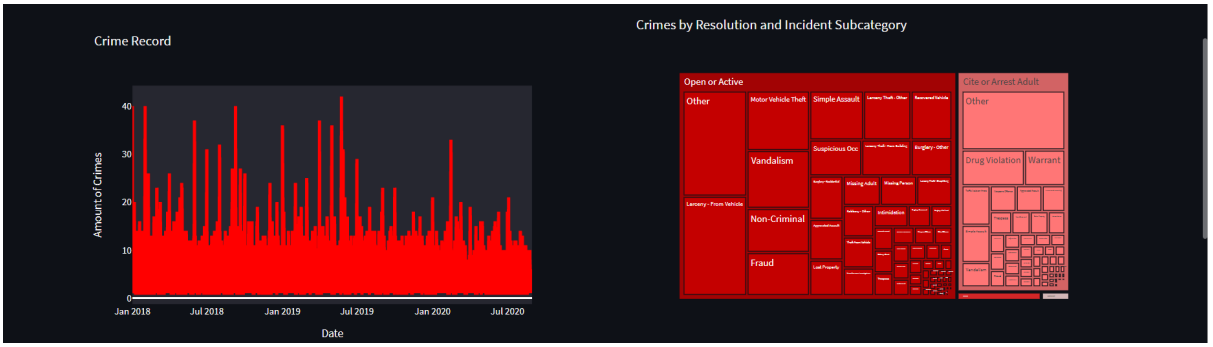
```
fig3.update_layout(title = 'Crime Record', xaxis_title = 'Date', yaxis_title='Amount of Crimes')
fig3.update_xaxes(showgrid=False)
fig3.update_yaxes(showgrid=False)

#GRAF 5 (fig4)
#Es un gráfico jerárquico que muestra en un círculo interior el Distrito Policial que atendió el reporte y en un círculo exterior el Vecindario donde el Incidente ocurrió. El área de estos círculos dependerá de la cantidad de Reportes registrados.
fig4 = px.sunburst(df[mask], color_discrete_sequence = ['#c40000','#ff7676'], values='contador', path=['Police District','Analysis Neighborhood'], hover_name='Analysis Neighborhood')
fig4.update_layout(plot_bgcolor="white", title = 'Crimes by Police District and Neighborhood')

#GRAF 6 (fig5)
#Es un gráfico jerárquico que muestra en un área exterior el Estado del reporte y en un área interior la Subcategoría del Incidente. El área general de estos cuadrados dependerá de la cantidad de Reportes registrados.
fig5 = px.treemap(df[mask], color_discrete_sequence = ['#c40000','#ff7676'], values='contador', path=['Resolution','Incident Subcategory'], hover_name='Incident Subcategory')
fig5.update_layout(plot_bgcolor="white", title = 'Crimes by Resolution and Incident Subcategory')

#Agregamos los gráficos hechos a la aplicación con la librería de Streamlit
```




```
#En la tercera fila, después de los filtros, estarán las figuras fig4 y fig
c1,c2 = st.columns((3,5))
with c1:
    st.plotly_chart(fig4)
with c2:
    st.plotly_chart(fig)
#En la cuarta fila estarán las figuras fig3 y fig5
```


```
d1,d2 = st.columns((1,1))
with d1:
    st.plotly_chart(fig3)
with d2:
    st.plotly_chart(fig5)
#En la quinta fila estarán las figuras fig2 y fig1
e1,e2 = st.columns((1,1))
with e1:
    st.plotly_chart(fig2)
with e2:
    st.plotly_chart(fig1)
```





Anexos

Datos : https://drive.google.com/file/d/1ioUp8979nNLh9h-CYVyJcRgAhxBhCLIm/view

Github: https://github.com/AlyekI/Integradora-UF6.git