Universidad del Rosario

Escuela de Ingeniería, Ciencia y Tecnología, Universidad del Rosario

Proyecto de Ingenieria: Tercera Entrega

Isabela Ruiz Bustos isabela.ruiz@urosario.edu.co
Miguel Angel Caicedo Carrasquilla miguela.caicedo@urosario.edu.co
MACC

26 de Mayo 2023

Primera Entrega

BASE DE DATOS

- Vehiculos.
- Validación de tarjetas.
- Salidas estación.

INTRODUCCIÓN

La movilización de los ciudadanos es esencial para que puedan realizar sus actividades cotidianas, algunos se pueden trasportar de manera privada, carros y/o motos; o publica, buses, metro y/o taxis. En el caso de Bogotá, contamos con una red de trasporte público llamada Transmilenio, esta también se conecta con el trasporte de la misma empresa, SITP. Enfocándonos exclusivamente en Transmilenio, consiste en buses biarticulados con estaciones; Los mismos cubren una gran parte de la ciudad con numerosas estaciones y portales.
Al analizar este servicio de trasporte publico queremos poder saber mas de que tan efectivo es este servicio. Al poder conocer de manera efectiva el tipo de vehículos, y la interacción que tienen las personas con el servicio podemos generar de una manera mas efectiva soluciones para mejorar la experiencia de los mismos.

DESCRIPCION DEL PROBLEMA

Este proyecto se centra en analizar si la calidad de los vehículos del transporte masivo de Transmilenio y la hora del día influyen en el pago del pasaje por parte de los usuarios. Se tiene como objetivo proporcionar información útil para la gestión del transporte masivo en la ciudad de Bogotá, y analizar la calidad del servicio ofrecido a los usuarios de Transmilenio.

REGLAS DE NEGOCIO

- Información del vehículo como: el ID, etiqueta, matrícula, nivel emision, marca y modelo.
- Cada vehículo realiza un solo servicio.
- Del servicio se conoce el tipo, el tipo de vehículo y accesibilidad.
- Cada servicio está registrado en una zona.
- De la zona se conoce el ID y el nombre,
- A cada zona pertenece una línea.
- De la línea se conoce el código y el nombre.
- Cada línea tiene varias estaciones.
- De las estaciones se conoce el código y el nombre.
- Las estaciones solo pertenecen a una linea.
- En las estaciones se realizan varias transacciones.
- Una zona puede ser operada por muchas concesiones.
- En cada concesión trabajan muchos operadores.
- De la concesión conocemos Id, zona y año otorgado.

- De los operadores conocemos Id y nombre.
- En las estaciones hay muchas salidas.
- En las salidas por intervalos se registra el total de las personas que pasan por mes.
- En las estaciones se pagan muchas transacciones.
- De las transacciones se conoce el dispositivo en las que se hacen y un código como discriminador.
- Las tarjetas dependen del saldo.
- Para realizar una transacción se debe conocer el saldo antes y el saldo después.
- De las tarjetas conocemos el ID, el tipo perfil, número de tarjeta y el tipo.
- Cada tarjeta es emitida por un único emisor, de este se sabe el nombre, tipo y el ID.

DIAGRAMA ENTIDAD RELACIONAL

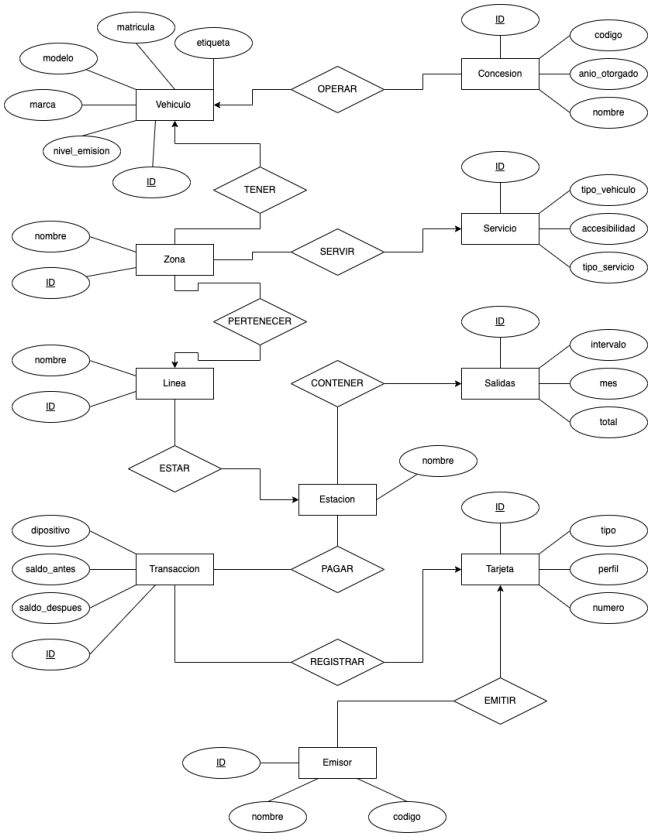

Figura 1: Diagrama entidad relacional.

Segunda Entrega

DIAGRAMA RELACIONAL NORMALIZADO EN TERCERA FORMA NORMAL

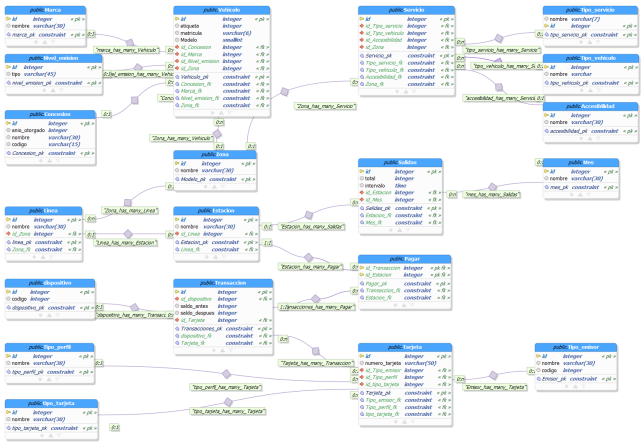

Figura 2: Diagrama entidad relacional normalizado.

IMPLENTACION EN POSTGRESQL

Usando las bases de datos que fueron anteriormente mencionadas, realizamos la unión de esta información dándonos las siguientes tablas:

- Zona.
- Accesibilidad.
- Concesión.
- Dispositivo.
- Tipovehículo.
- Marca.
- Nivel_emisión.
- Tipo_perfil.
- Tipo_tarjeta.
- Tipo_servicio.
- Mes.
- Tipo_emisor.
- Línea.
- Estación.
- Salidas.

- Tarjeta.
- Transacción.
- Servicio.
- Vehículo.
- Pagar.

Algunas de las tablas no tenían nombres adecuados para ser cargados de manera correcta en Postgres SQL, entonces realizamos modificaciones para que puedan ser usadas de manera adecuada. Algunos errores que se presentaron para tener que renombrar, es que presentaban el nombre de la tabla con dígitos o caracteres especiales.

CARGA DE DATOS

Se descargaron los cvs subidos por la alcandía de la Bogotá a la pagina de datos abiertos de Bogotá; y de los datos abiertos Transmilenio S.A. Estos archivos cvs sufrieron modificaciones como eliminación de comillas, tildes y espacios vacíos en las tablas.

POSIBLES ESCENARIOS DE ANÁLISIS

- Uso de trasporte público, con los datos almacenados en las tablas “Estación”, “Mes”, “Salidas”, “Servicio”, “Tipo_serivcio” y “Vehículo”, se podría realizar un análisis de uso del transporte publico en una determinada zona. Se podrían hacer preguntas como: ¿Cuántas personas utilizan el transporte público en una zona en particular en un mes determinado? ¿Cuál es el tipo de vehículo más utilizado? ¿Cuál es la estación más utilizada? ¿Cómo se comparan los niveles de emisión de los diferentes vehículos en esa zona?

- Comportamiento de los usuarios: Con los datos almacenados en las tablas "Tarjetaz "Transaccion", se podría realizar un análisis de comportamiento de los usuarios del transporte público. Se podrían hacer preguntas como: ¿Qué tipo de perfil de usuario utiliza más el transporte público? ¿En qué momento del día se realizan más transacciones? ¿Cuál es el promedio de saldo antes y después de una transacción? ¿Qué dispositivos se utilizan con más frecuencia para realizar transacciones?

- Accesibilidad: Con los datos almacenados en las tablas .Accesibilidadz "Servicio", se podría realizar un análisis de accesibilidad en el transporte público en una determinada zona. Se podrían hacer preguntas como: ¿Qué servicios de transporte público son accesibles para personas con discapacidades? ¿Cuál es el porcentaje de vehículos que cumplen con los requisitos de accesibilidad? ¿Cuál es el porcentaje de estaciones accesibles?

- Concesiones: Con los datos almacenados en las tablas Çoncesionz "Vehiculo", se podría realizar un análisis de las concesiones de transporte público en una determinada zona. Se podrían hacer preguntas como: ¿Cuántas concesiones se han otorgado en una zona en particular en un año determinado? ¿Cuáles son las marcas y modelos de vehículos más utilizados por los concesionarios? ¿Cuál es el porcentaje de vehículos que cumplen con los requisitos de emisión?

DESARROLLO EN PYTHON

Se creó el archivo de python y se hizo un import de la librería “psycopg2”, luego de realizar la conexión se usó la función “cursor.execute” para hacer las consultas de las tablas. En el repositorio de github está adjunto el archivo visualización.py donde se puede apreciar el código.

Tercera Entrega

EXPLORACIÓN REPOSITORIO GIT

I. Creación SQL

Este código consiste en la creación de varias tablas en una base de datos. Cada tabla representa una entidad en un sistema y define su estructura y relaciones con otras entidades. A grandes rasgos, las tablas creadas son las siguientes:

1. Zona: Representa una zona en el sistema, con un identificador y un nombre.
2. Accesibilidad: Representa el nivel de accesibilidad de algún elemento, con un identificador y un nombre.

3. Concesión: Representa una concesión con un identificador, nombre, código y año de otorgamiento.

4. Dispositivo: Representa un dispositivo con un identificador y un código.

5. Tipo_vehiculo: Representa un tipo de vehículo con un identificador y un nombre.

6. Marca: Representa una marca con un identificador y un nombre.

7. Nivel_emision: Representa el nivel de emisión de algún elemento, con un identificador y un tipo.

8. Tipo_perfil: Representa un tipo de perfil con un identificador y un nombre.

9. Tipo_tarjeta: Representa un tipo de tarjeta con un identificador y un nombre.

10. Tipo_servicio: Representa un tipo de servicio con un identificador y un nombre.

11. Mes: Representa un mes con un identificador y un nombre.

12. Tipo_emisor: Representa un tipo de emisor con un identificador, nombre y código.

13. Linea: Representa una línea en el sistema, con un identificador, nombre y el id de la zona a la que pertenece.

14. Estación: Representa una estación en el sistema, con un identificador, nombre y el id de la línea a la que pertenece.

15. Salidas: Representa las salidas de alguna estación en un mes específico, con un identificador, el total de salidas, un intervalo de tiempo, y los ids de la estación y el mes correspondientes.

16. Tarjeta: Representa una tarjeta en el sistema, con un identificador, número de tarjeta, y los ids del tipo de emisor, tipo de perfil y tipo de tarjeta correspondientes.

17. Transacción: Representa una transacción en el sistema, con un identificador, saldo antes de la transacción, saldo después de la transacción, y los ids de la tarjeta y el dispositivo correspondientes.

18. Servicio: Representa un servicio en el sistema, con un identificador y los ids de la zona, tipo de servicio, tipo de vehículo y accesibilidad correspondientes.

19. Vehículo: Representa un vehículo en el sistema, con un identificador, etiqueta, matrícula, y los ids de la concesión, modelo, marca, nivel de emisión y zona correspondientes.

20. Pagar: Establece una relación entre una estación y una transacción, utilizando los ids correspondientes.

Para ejecutar este código, se debe utilizar un gestor de bases de datos compatible (por ejemplo, PostgreSQL) y ejecutar cada una de las sentencias CREATE TABLE en secuencia en el gestor. Esto creará las tablas en la base de datos según la estructura definida.

II. Carga_Datos.sql

Este código realiza una serie de operaciones de carga de datos en las tablas creadas previamente. A continuación se describe cada operación:

1. COPY Zona (nombre) FROM '/Users/isabela/Zona.csv' WITH DELIMITER ';' CSV HEADER;
   - Esta instrucción carga datos en la tabla "Zona"desde un archivo CSV ubicado en la ruta '/Users/isabela/Zona.csv', utilizando ';' como delimitador y considerando la primera fila como encabezado
2. INSERT INTO Zona(nombre) VALUES ('NORTE');
   - Esta sentencia inserta un registro en la tabla "Zonaçon el valor 'NORTE' en el campo "nombre".

3. SELECT * FROM Zona;
    - Esta consulta selecciona todos los registros de la tabla "Zonaz los muestra en la salida.
4. Las siguientes operaciones (COPY, SELECT) realizan acciones similares a las descritas en los puntos anteriores, pero para las tablas restantes: Accesibilidad, Concesion, Dispositivo, Tipo_vehiculo, Marca, Nivel_emision, Tipo_perfil, Tipo_tarjeta, Tipo_servicio, Mes, Tipo_emisor, Linea, Estacion, Salidas, Tarjeta, Transaccion, Pagar, Servicio y Vehiculo.

Cada operación COPY carga datos en la tabla correspondiente desde un archivo CSV específico, utilizando el delimitador ';' y considerando la primera fila como encabezado. Las operaciones SELECT seleccionan todos los registros de cada tabla y los muestran en la salida. El propósito de estas operaciones es cargar datos preexistentes en las tablas para poblar la base de datos y realizar consultas posteriormente. Es importante asegurarse de que los archivos CSV mencionados en las operaciones COPY existan en las rutas proporcionadas y tengan el formato correcto para que la carga de datos sea exitosa.

III. Visualizacion.py

Este código utiliza la biblioteca psycopg2 para conectarse a una base de datos PostgreSQL y ejecutar varias consultas SQL para mostrar los elementos de diferentes tablas de la base de datos. El script realiza el mismo proceso para cada tabla que desea consultar. Las consultas se ejecutan mediante el método execute() del objeto cursor, y los resultados se obtienen mediante el método fetchall(). Luego, los resultados se recorren mediante un bucle for y se imprimen por pantalla los valores de cada campo. Después de ejecutar todas las consultas, el script cierra el objeto cursor y la conexión a la base de datos mediante los métodos close(). En caso de que ocurra un error durante la conexión o ejecución de las consultas, se captura la excepción psycopg2.Error y se muestra un mensaje de error. Aquí está el resumen de lo que hace el código:

1. Establece una conexión a la base de datos utilizando los parámetros de conexión proporcionados (usuario, contraseña, host, base de datos y puerto).
2. Imprime Çonexión correcta"si la conexión se establece correctamente.
3. Ejecuta una serie de consultas SQL para mostrar los elementos de diferentes tablas de la base de datos. Cada consulta selecciona columnas específicas de una tabla y recupera todos los registros de esa tabla.
4. Imprime los resultados de cada consulta en la consola, mostrando los valores de las columnas para cada registro.
5. Maneja cualquier excepción de psycopg2 que pueda ocurrir durante la conexión o ejecución de las consultas y muestra un mensaje de error.
6. Cierra el cursor y la conexión a la base de datos al finalizar.

Aquí está el resumen de lo que hace el código:

1. Tener instalada la biblioteca psycopg2.
2. Tener una base de datos PostgreSQL en ejecución con las tablas mencionadas en el código.
3. Proporcionar los valores correctos para los parámetros de conexión, como el nombre de usuario, contraseña, host, nombre de la base de datos y puerto.

Una vez que se cumplan estos requisitos, puedes ejecutar el código en un entorno de desarrollo Python o en la línea de comandos para obtener los resultados de las consultas en la consola. Asegúrate de tener los permisos adecuados para acceder a la base de datos y sus tablas.

IV. Connection1.py

Este código define una clase llamada Connection1 que se utiliza para abrir y cerrar una conexión a una base de datos PostgreSQL utilizando la biblioteca psycopg2. Aquí está el resumen de lo que hace el código:

1. La clase Connection1 tiene un constructor __init__ que inicializa la variable connection como None. Esta variable se utilizará para almacenar la conexión a la base de datos.

2. El método openConnection1 se encarga de abrir la conexión a la base de datos. Utiliza los parámetros de conexión proporcionados (nombre de usuario, contraseña, nombre de la base de datos, host y puerto) para establecer la conexión mediante psycopg2.connect. Si ocurre algún error durante la conexión, se imprime el mensaje de error.

3. El método closeConnection1 se encarga de cerrar la conexión a la base de datos. Llama al método close de la variable connection para cerrar la conexión.

V. ProyectoSQL.py

Aquí tienes las funciones definidas en el código proporcionado:

1. usuarios_por_zona(): Esta función devuelve una consulta SQL que calcula la cantidad total de personas que utilizan el transporte en cada zona. Agrupa los resultados por zona y los ordena en orden descendente según el número de personas. La consulta solo considera los datos del mes con ID 1.

2. usuarios_por_estacion(): Esta función devuelve una consulta SQL que calcula la cantidad total de personas que utilizan el transporte en cada estación. Agrupa los resultados por estación y los ordena en orden descendente según el número de personas. La consulta solo considera los datos del mes con ID 1.

3. tipo_perfil_mas_utilizado(): Esta función devuelve una consulta SQL que determina el tipo de perfil más utilizado. Cuenta el número de tarjetas para cada tipo de perfil y los ordena en orden descendente según el total utilizado.

4. salidas_por_estacion(): Esta función devuelve una consulta SQL que calcula el total de salidas registradas en cada estación. Agrupa los resultados por estación y los ordena en orden descendente según el número total de salidas. Solo considera los datos del mes con ID 1 y limita los resultados a las 5 estaciones con más salidas.

5. salidas_hora_pico_3_8(): Esta función devuelve una consulta SQL que calcula el total de salidas registradas en cada estación durante la hora pico entre las 15:00 y las 20:00 horas. Agrupa los resultados por estación y los ordena en orden descendente según el número total de salidas. Limita los resultados a las 10 estaciones con más salidas.

6. generar_grafico_pie_zonas(): Esta función devuelve una consulta SQL que cuenta la cantidad de estaciones de cada zona y la utiliza para generar un gráfico de pastel. Agrupa los resultados por zona.

7. registros_por_servicio(): Esta función devuelve una consulta SQL que cuenta la cantidad de registros por tipo de servicio. Agrupa los resultados por tipo de servicio.

8. salidas_americas(): Esta función devuelve una consulta SQL que cuenta la cantidad de salidas registradas en cada estación de la zona .ªméricas"(zona con ID 1).

9. salidas_NQSUR(): Esta función devuelve una consulta SQL que cuenta la cantidad de salidas registradas en cada estación de la zona "Norte-Quito Sur"(zona con ID 2).

10. salidas_CALLE80(): Esta función devuelve una consulta SQL que cuenta la cantidad de salidas registradas en cada estación de la zona Çalle 80"(zona con ID 4).

11. salidas_CALLE26(): Esta función devuelve una consulta SQL que cuenta la cantidad de salidas registradas en cada estación de la zona Çalle 26"(zona con ID 6).

12. salidas_CARACAS(): Esta función devuelve una consulta SQL que cuenta la cantidad de salidas registradas en cada estación de la zona Çaracas"(zona con ID 8).

13. salidas_CARRERA10(): Esta función devuelve una consulta SQL que cuenta la cantidad de salidas registradas en cada estación de la zona Çarrera 10"(zona con ID 9).

14. salidas_SUBA(): Esta función devuelve una consulta SQL que cuenta la cantidad de salidas registradas en cada estación de la zona "Suba"(zona con ID 10).

15. salidas_CARACASSUR(): Esta función devuelve una consulta SQL que cuenta la cantidad de salidas registradas en cada estación de la zona Çaracas Sur"(zona con ID 17).

16. salidas_NORTE(): Esta función devuelve una consulta SQL que cuenta la cantidad de salidas registradas en cada estación de la zona "Norte"(zona con ID 18).

Estas funciones pueden ser utilizadas para ejecutar las consultas SQL correspondientes en una base de datos PostgreSQL utilizando la biblioteca psycopg2. Para obtener los resultados de las consultas, necesitarás ejecutar las consultas utilizando una conexión a la base de datos y luego procesar los resultados según sea necesario.

VI. Analisis.py

Este código es una aplicación de tablero de control utilizando el framework Dash de Python. Dash es una biblioteca que permite crear aplicaciones web interactivas con componentes de visualización de datos. A grandes rasgos, el código importa los módulos necesarios, incluyendo Dash y componentes relacionados, como dcc (componentes de Dash Core), html (componentes HTML), dbc (componentes Bootstrap) y plotly.express (librería de visualización de datos). También importa otros módulos personalizados como Connection1.py y ProyectoSQL.py. Luego, se establecen los estilos externos utilizando la lista de hojas de estilo externas de Bootstrap. Se crea una instancia de la aplicación Dash y se asigna a la variable .ªpp". A continuación, se realizan consultas a una base de datos utilizando la clase Çonnection1"para obtener datos. Los datos se almacenan en DataFrames de pandas y se utilizan para crear gráficos de barras y gráficos circulares utilizando la biblioteca plotly.express. Los gráficos se configuran con opciones de estilo y se agregan al diseño de la aplicación utilizando componentes html.Div y dcc.Graph. Finalmente, se crea una estructura de pestañas utilizando el componente dbc.Tabs para mostrar los gráficos de barras y gráficos circulares en pestañas separadas. Para ejecutar este código, es necesario tener instaladas las bibliotecas Dash, Pandas y Plotly. Además, se deben proporcionar los módulos personalizados Çonnection1ƶ "ProyectoSQLçon sus implementaciones correspondientes. Una vez que todo está configurado, se puede ejecutar la aplicación Dash llamando al método run_server().ᵉⁿ la instancia de la aplicación .ªpp".

DESCRIPCIÓN ANÁLISIS REALIZADOS

Ventajas de cada visualización:

1. Gráficos de barras:
   - Permite comparar fácilmente la cantidad de usuarios o salidas entre diferentes zonas o estaciones.
   - Proporciona una representación visual clara de los datos.
   - Facilita la identificación de tendencias o patrones.
2. Gráficos circulares:
   a) Muestra la distribución relativa de registros por zona.
   b) Permite identificar rápidamente las zonas con mayor o menor número de registros.
   c) Proporciona una visualización concisa y fácil de entender.

Desventajas de cada visualización:

1. Gráficos de barras:
   - Puede resultar difícil comparar visualmente las diferencias exactas entre las barras si hay muchas categorías o valores cercanos.

- Requiere suficiente espacio en la pantalla para mostrar barras largas si los datos son numerosos.

2. Gráficos circulares:

   a) No permite comparar fácilmente las diferencias exactas entre las categorías.

   b) Puede ser difícil interpretar visualmente los tamaños relativos de las secciones en el gráfico circular.

RESULTADOS GRÁFICOS Y ANÁLISIS

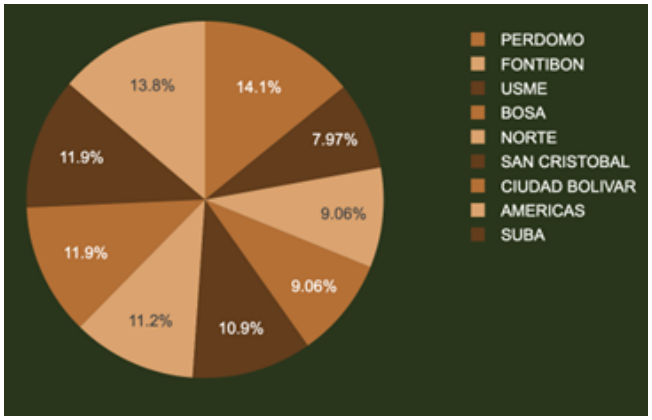

Figura 3: Distribución de registros por zonas.

Acá podemos apreciar que la distribución de registros por zonas es mayor en Fontibón (14.1 %) y menor en Usme (7.97 %).

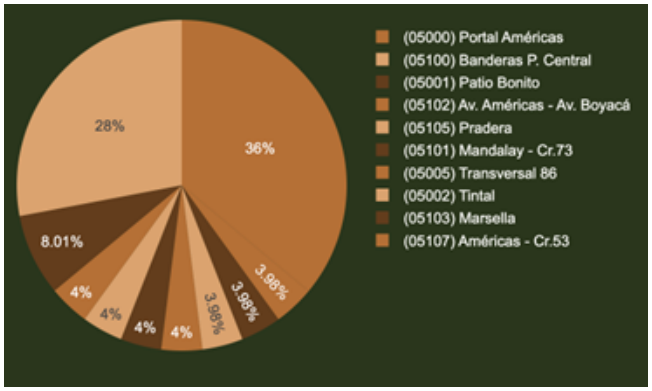

Figura 4: Distribución de registros zona Americas.

En esta grafica podemos apreciar la distribución por registro en la zona de las Américas, donde las estaciones Portal Américas (36 %) y Banderas P. Central (28 %), son las más concurridas. Agregado a esto Marsella (3.98 %) y Américas-Cr 53 (3.98 %) son las menos concurridas.

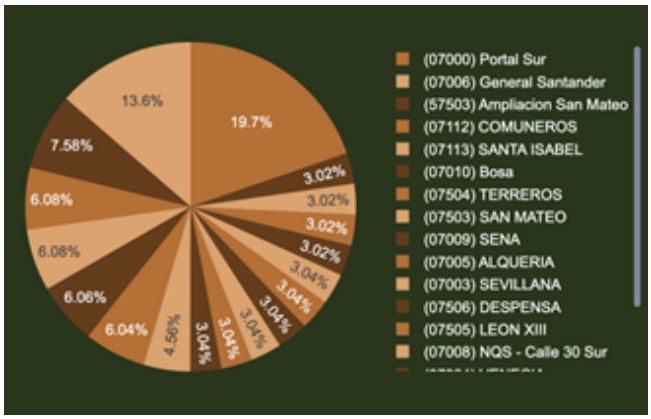

Figura 5: Distribución de registros zona NQS Sur.

En esta grafica podemos apreciar la distribución por registro en la zona NQS sur, donde las estaciones Portal Sur (19.7 %) y General Santander (13.6 %), son las más concurridas. Agregado a esto XXX (3.02 %) y XXX (3.02 %) son las menos concurridas.

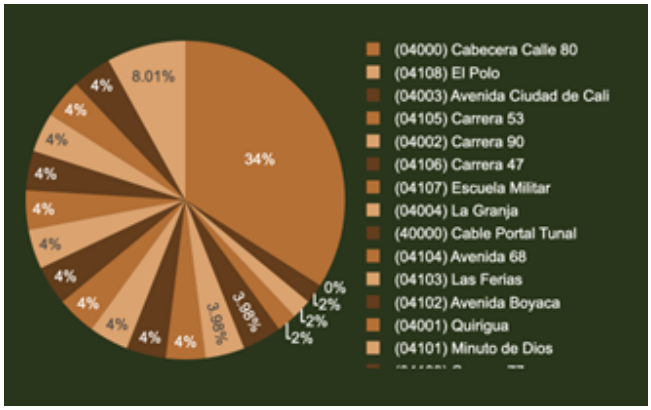

Figura 6: Distribución de registros zona calle 80.

En esta grafica podemos apreciar la distribución por registro en la zona calle 80, donde las estaciones Cabecera Calle 80 (34 %) y El Polo (8.01 %), son las más concurridas. Agregado a esto XXX (2 %) y XXX (0 %) son las menos concurridas.

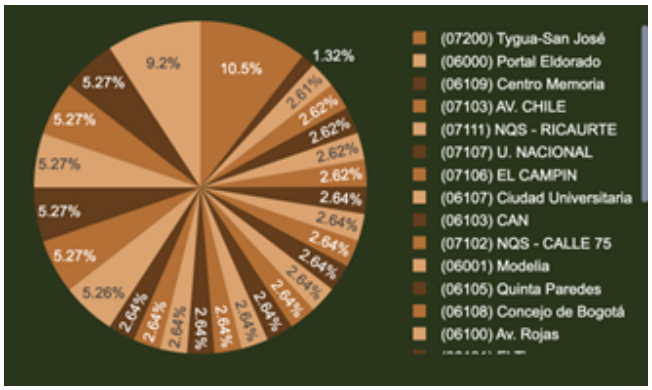

Figura 7: Distribución de registros zona calle 26.

En esta grafica podemos apreciar la distribución por registro en la zona calle 26, donde las estaciones Tygua-San José (10.5 %) y Portal Eldorado (9.2 %), son las más concurridas. Agregado a esto XXX (2.61 %) y XXX (1.32 %) son las menos concurridas.

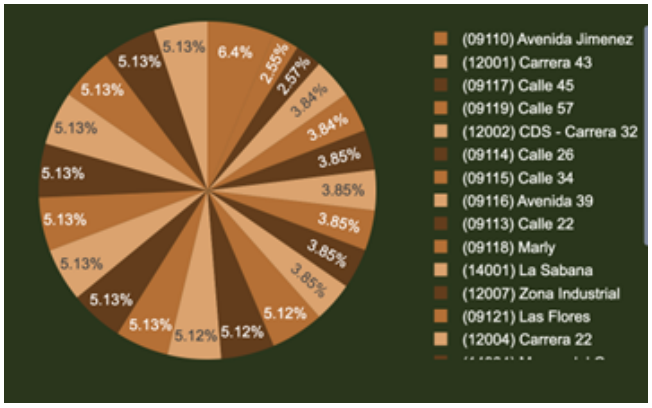

Figura 8: Distribución de registros zona Caracas.

En esta grafica podemos apreciar la distribución por registro en la zona Caracas, donde las estaciones Avenida Jimenez (6.4 %) y Carrera 43 (5.13 %), son las más concurridas. Agregado a esto XXX (2.57 %) y XXX (2.55 %) son las menos concurridas.

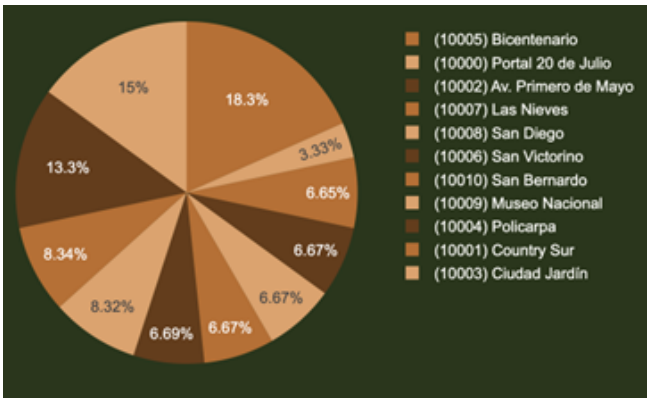

Figura 9: Distribución de registros zona Carrera 10.

En esta grafica podemos apreciar la distribución por registro en la zona Carrera 10, donde las estaciones Bicentenario (18.3 %) y Portal 20 de Julio (15 %), son las más concurridas. Agregado a esto Country Sur (6.65 %) y Ciudad Jardin (3.33 %) son las menos concurridas.

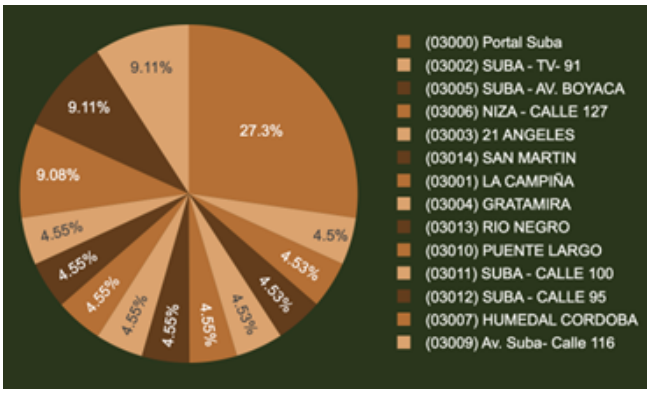

Figura 10: Distribución de registros zona Suba.

En esta grafica podemos apreciar la distribución por registro en la zona Suba, donde las estaciones Portal Suba (27.3 %) y Suba-TV-91 (9.11 %), son las más concurridas. Agregado a esto Humedal Córdoba (4.53 %) y Av. Suba-Calle 116 (4.5 %) son las menos concurridas.

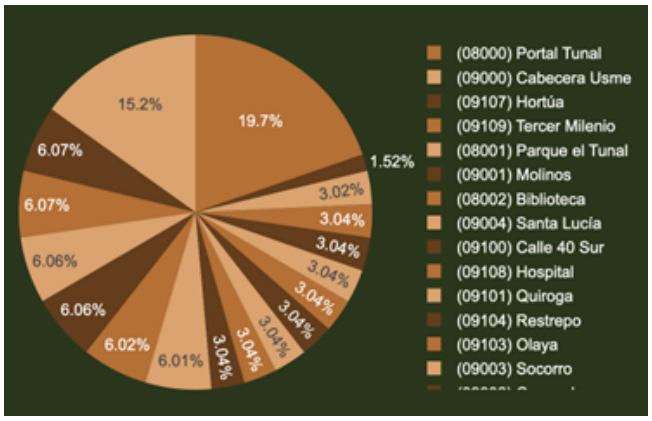

Figura 11: Distribución de registros zona Caracas Sur.

En esta grafica podemos apreciar la distribución por registro en la zona Caracas Sur, donde las estaciones Portal Tunal (19.7 %) y Cabecera Usme (15.2 %), son las más concurridas. Agregado a esto XXX (3.02 %) y XXX (1.52 %) son las menos concurridas.

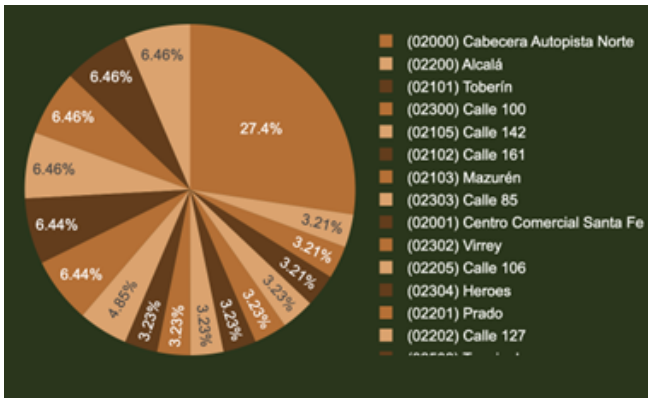

Figura 12: Distribución de registros zona Norte.

En esta grafica podemos apreciar la distribución por registro en la Norte, donde las estaciones Cabecera Autopista Norte (27.4 %) y Alcalá (6.46 %), son las más concurridas. Agregado a esto XXX (3.21 %) y XXX (3.21 %) son las menos concurridas.

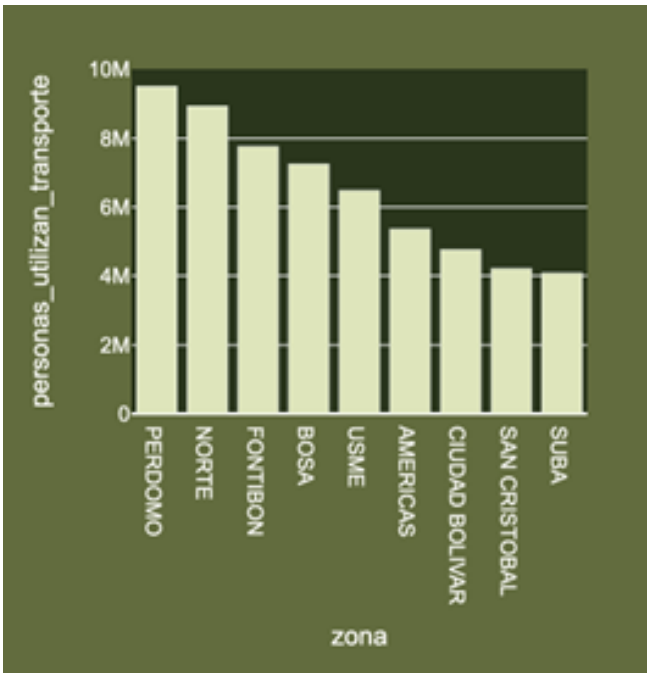

Figura 13: Usuarios por zona.

En esta grafica podemos apreciar los usuarios por zonas, donde la zona con mayor flujo de personas que utilizan este trasporte seria Perdomo con un promedio de 8M a 10 M, seguido de zona Norte; y la zona con menor flujo de personas que utilizan este trasporte seria Suba con un promedio de 4M.

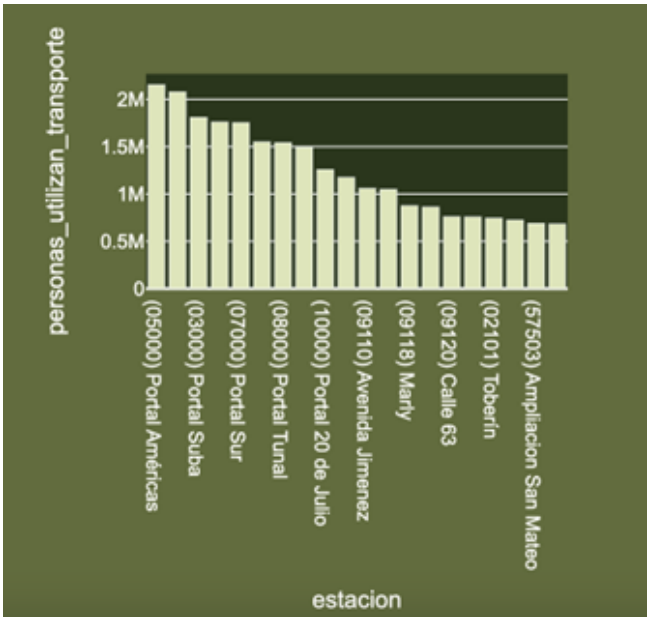

Figura 14: Usuarios por estación.

En esta grafica podemos apreciar los usuarios por estación, donde la estación con mayor flujo de

personas que utilizan este trasporte seria Portal Américas con un promedio de 2M, seguido de Portal Suba; y la estación con menor flujo de personas que utilizan este trasporte seria Ampliación San Mateo con un promedio de entre 0.5M a 1M.

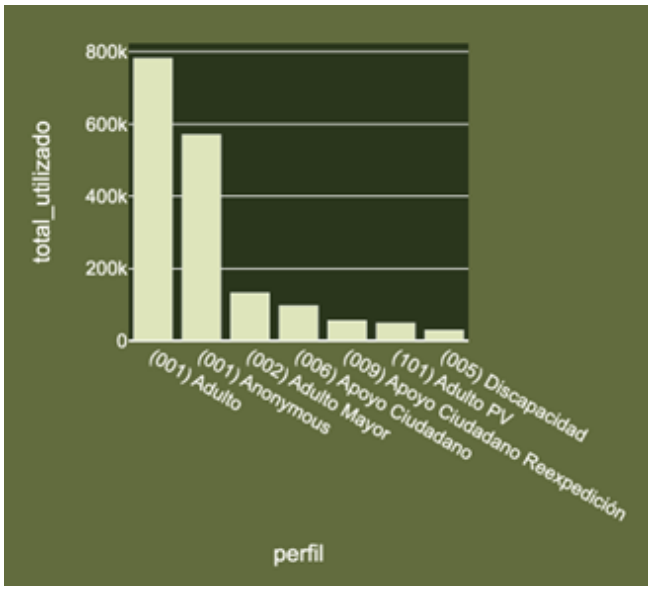

Figura 15: Usuarios por tipo perfil.

En esta grafica podemos apreciar el tipo de perfil comparado con el total de uso, donde el perfil que más utiliza este trasporte seria Adulto con un promedio de entre 600K a 800K veces de uso, seguido de Anonymous; y el perfil que menos utiliza este trasporte serían las personas con discapacidad con un promedio de entre 0 a 200K.

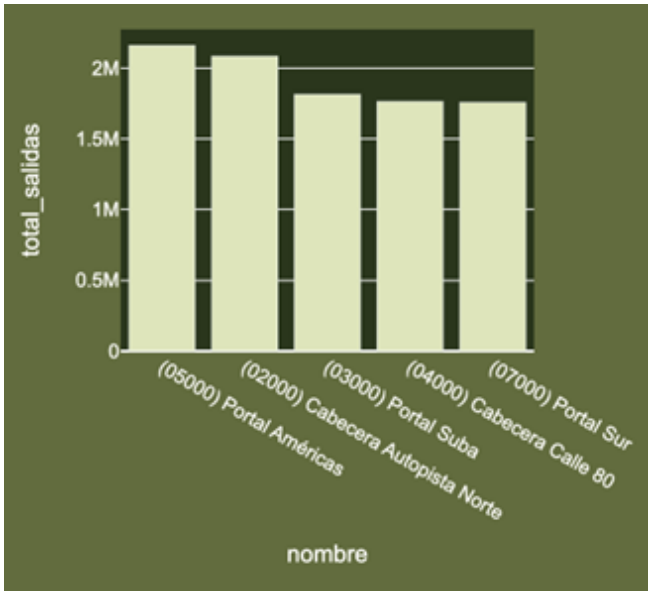

Figura 16: Salidas por estación.

En esta grafica podemos apreciar el total de salidas comparado con la estación, donde la estación que más salidas registradas tiene seria Portal Américas con un promedio mayor a 2M, seguido de Cabecera Autopista Norte; y la estación que menos salidas registradas tiene seria Portal Sur con un promedio entre 1.5M y 2M.

REPOSITORIO GITHUB

Repositorio.

A continuacion agregaremos los perfiles de Github de cada uno de los integrantes.

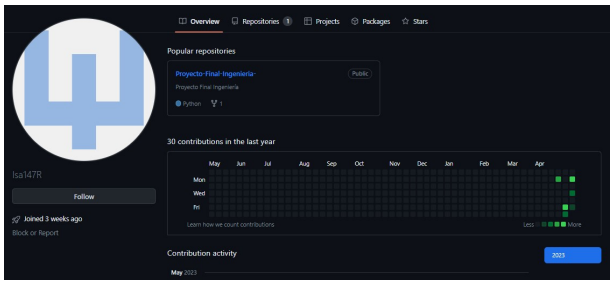

Figura 17: Perfil de github de Isabela Ruiz Bustos.

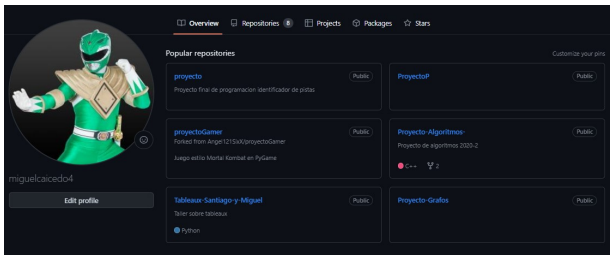

Figura 18: Perfil de github de Miguel Caicedo Carrasquilla