OBTENCIÓN DE INFORMACIÓN DEL TRÁNSITO VEHICULAR EN ZONAS DE ATENCIÓN PRIORITARIA PARA LA VIGILANCIA DE LA CALIDAD DEL AIRE

Instructivo: IN-DMA-008

Versión: 01

SUBDIRECCIÓN DE EVALUACIÓN DEL AMBIENTE ATMOSFÉRICO – DIRECCIÓN DE METEOROLOGÍA Y EVALUACIÓN AMBIENTAL ATMOSFÉRICA

Elaborado por:	Elvis Anthony Medina Dionicio Subdirector Subdirección de Evaluación del Ambiente Atmosférico José Hitoshi Inoue Velarde Analista en Vigilancia Atmosférica Global Subdirección de Evaluación del Ambiente Atmosférico	**Firma:** Firma Digital Senamhi Firmado digitalmente por MEDINA DIONICIO Elvis Anthony FAU 20131366028 soft Motivo: Soy el autor del documento Fecha: 15.12.2025 13:40:10 -05:00 Firma Digital Senamhi Firmado digitalmente por INOUE VELARDE Jose Hitoshi FAU 20131366028 soft Motivo: Soy el autor del documento Fecha: 15.12.2025 13:33:04 -05:00
Revisado por:	Sonia del Carmen Huamán Lozano Directora Unidad de Modernización y Gestión de la Calidad	**Firma:** Firma Digital Senamhi Firmado digitalmente por HUAMAN LOZANO Sonia Del Carmen FAU 20131366028 soft Motivo: Soy el autor del documento Fecha: 19.12.2025 15:28:31 -05:00
Aprobado por:	Grinia Jesús Avalos Roldán Director Dirección de Meteorología y Evaluación Ambiental Atmosférica	**Firma:** Firma Digital Senamhi Firmado digitalmente por AVALOS ROLDAN Grinia Jesus FAU 20131366028 soft Motivo: Soy el autor del documento Fecha: 23.12.2025 11:19:45 -05:00

1. OBJETIVO

Establecer las acciones para la obtención de información del tránsito vehicular (tiempo de demora) que contribuya a analizar su relación con el comportamiento de los contaminantes atmosféricos en Zonas de Atención Prioritaria a nivel nacional, en el marco de la vigilancia de calidad del aire.

2. DESARROLLO

2.1 Acrónimos y Siglas

2.1.1. DMA: Dirección de Meteorología y Evaluación Ambiental Atmosférica

2.1.2. DZ: Dirección Zonal

2.1.3. GESTA: Grupo de Estudio Técnico Ambiental de la Calidad del Aire

2.1.4. SEA: Subdirección de Evaluación del Ambiente Atmosférico

2.1.5. SENAMHI: Servicio Nacional de Meteorología e Hidrología del Perú

2.1.6. ZAP: Zona de Atención Prioritaria

2.2. Definiciones

2.2.1. Briefing (reunión informativa) ambiental atmosférico: Consiste en una reunión diaria en la cual se pone en discusión las condiciones de nubosidad, condiciones oceanográficas, condiciones meteorológicas, condiciones del tránsito vehicular y de monitoreo de calidad de aire de los últimos días en el área de estudio, con la finalidad de llegar a un consenso para el pronóstico de la calidad de aire del día siguiente.

2.2.2. Grupo de Estudio Técnico Ambiental de la Calidad del Aire - Gesta[1]: Están compuestos por representantes de instituciones de los sectores público y privado y por las personas naturales designadas por sus cualidades profesionales; y son los encargados de formular los Planes de Acción para el Mejoramiento de la Calidad del Aire en una Zona de Atención Prioritaria. Dentro de este grupo se considera al Servicio Nacional de Meteorología e Hidrología.

[1] Artículo 3 del Decreto Supremo N° 074-2001-PCM. Obtenido de: https://cdn.www.gob.pe/uploads/document/file/5224238/D.S%20074-2001-PCM.pdf?v=1696376300

2.2.3. Zona de Atención Prioritaria[2]: Son aquellos centros poblados que cuenten con actividades económicas que planteen real o potencial afectación en la calidad del aire, que posean actividad vehicular ambientalmente relevante, o que cuenten con una dinámica urbana que implique un potencial incremento de emisiones atmosféricas.

2.3. Recursos para la obtención de información de tránsito vehicular

2.3.1. Plataformas web de tránsito vehicular: Son sitios web que ofrecen información histórica y/o en tiempo real sobre el tránsito vehicular. Se basan en datos anónimos recopilados de dispositivos móviles, sensores y reportes de usuarios, permitiendo estimar el tiempo de demora, la velocidad promedio de los vehículos, visualizar los niveles de congestión vial mediante un sistema de colores, entre otros. Las condiciones a tomar en cuenta por parte de los usuarios para el acceso a la información varían dependiendo del tipo de información y del sitio web, entre los cuales podemos mencionar a Google Maps, Google Traffic, Tomtom, etc.

2.3.2. Python[3]: Es un lenguaje de programación interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, tipado dinámico, tipos de datos de muy alto nivel y clases. Python combina un poder destacado con una sintaxis muy clara y es ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos, el machine learning, entre otros.

2.3.3. Rutas: Nos referimos tanto a la vía en su totalidad como a un tramo específico de esta, seleccionado por su relevancia debido a la mayor congestión vehicular, con el fin de analizar la influencia del tiempo del tránsito en dicha vía sobre la calidad del aire.

2.3.4. Web scraping[4] (obtención de datos de sitios web): Es la técnica de extracción automática de datos a través de un programa que interactúa con una API. Esta práctica consiste en escribir un programa automatizado que consulta a un servidor web (simulando ser un navegador), descarga los datos (generalmente en formato HTML y otros archivos que componen las páginas web), y los analiza para extraer la información necesaria.

[2] Cuarta Disposición Complementaria Final del Decreto Supremo N° 003-2017-MINAM. Obtenido de: https://www.minam.gob.pe/wp-content/uploads/2017/06/DS-003-2017-MINAM.pdf
[3] Obtenido de: https://docs.python.org/es/3.10/faq/general.html#what-is-python https://aws.amazon.com/es/what-is/python/
[4] Obtenido de: https://edu.anarcho-copy.org/Programming%20Languages/Python/Web%20Scraping%20with%20Python,%202nd%20Edition.pdf

2.4. Acciones para la obtención de información de tránsito vehicular

Considerando que las ZAP se caracterizan por su alta actividad vehicular, la cual incide directamente sobre la calidad del aire debido a las emisiones generadas, la SEA de la DMA, en el marco de sus competencias en vigilancia y pronóstico de la calidad del aire, ha identificado la necesidad de obtener información del tránsito vehicular en estas zonas. Dicha información permite analizar su relación con el comportamiento espacial y temporal de los contaminantes atmosféricos.

Actualmente, la SEA viene obteniendo información para el Área Metropolitana de Lima y Callao, una de las 31 ZAP definidas a nivel nacional[5] y cuya principal fuente de contaminación del aire proviene del parque automotor, tal como se señala en el documento denominado Diagnóstico de la Gestión de la Calidad Ambiental del Aire de Lima y Callao[6].

Por tal motivo, en el presente instructivo, se detalla el conjunto de acciones[7], que automatizan la obtención de información del tránsito vehicular (tiempo de demora) mediante la ejecución de scripts, para su réplica en otras ZAP por parte de las DZ que participan en los GESTA a nivel Nacional. Siendo las actividades a seguir los siguientes:

- Captura de la Información del tiempo de demora respecto al tránsito vehicular.
- Elaboración de gráficos del tiempo de demora respecto al tránsito vehicular.

Los productos finales de los scripts elaborados se conforman por los registros horarios del tiempo de demora del tránsito vehicular para las rutas seleccionadas y el correspondiente gráfico de evolución temporal. La utilidad de la información obtenida se evidencia en los análisis de factores que influyen sobre el comportamiento de los contaminantes atmosféricos como parte de los briefings ambientales atmosféricos. En ese sentido, también contribuyen a la elaboración de reportes, boletines, estudios e investigaciones sobre calidad del aire en la mencionada ZAP.

Por otro lado, dado que el SENAMHI a través de sus DZ participa de los GESTA a nivel nacional, constituye información valiosa que contribuye a la elaboración de los Planes de Acción para Mejora de la Calidad del Aire en la ZAP, así como al monitoreo continuo de las condiciones del tránsito vehicular que permitan a los tomadores de decisión en las Comisiones Ambientales Municipales, promover

[5] Para mayor detalle revisar el siguiente enlace: https://www.minam.gob.pe/calidadambiental/zonas-priorizadas-de-calidad-del-aire

[6] Para mayor detalle revisar el siguiente enlace: https://sinia.minam.gob.pe/sites/default/files/sinia/archivos/public/docs/diagnostico_calidad_aire_0.pdf

[7] Basadas en metodologías de publicaciones científicas, p.ej, GOOGLE MAP TRAFFIC DATA SCRAPING AND MINING. Obtenido de: https://www.irjmets.com/uploadedfiles/paper/volume3/issue_4_april_2021/8350/1628083343.pdf

acciones que tengan la finalidad de reducir las emisiones del parque automotor en favor de la salud pública y del ambiente. Asimismo, genera oportunidad de complementar dicha información con monitoreos de calidad del aire en el ámbito de las DZ para la elaboración de reportes, boletines, estudios e investigaciones.

2.4.1. Captura de la información del tiempo de demora respecto al tránsito vehicular

a) Selección de vías de tránsito vehicular en la ZAP

Para elegir las vías del tránsito vehicular a evaluar dentro de una ZAP, es importante considerar el conocimiento previo de los servidores/as de las DZ del SENAMHI, así como la información proporcionada por los habitantes de la ZAP sobre las vías con mayor congestión vehicular, la cual se puede recopilar a través de encuestas, formularios u otros medios pertinentes. Asimismo, existen plataformas web de tránsito vehicular que te permiten obtener información histórica a nivel global.

Por otro lado, se precisa que la selección de vías no está limitada a una cantidad fija, ya que esta depende del comportamiento del tránsito vehicular en cada lugar y del criterio del evaluador para obtener información relevante que contribuya al análisis de sus efectos sobre la calidad del aire en la ZAP.

b) Obtención de las rutas relacionada a cada vía seleccionada

Seleccionadas las vías de tránsito vehicular, se obtienen las rutas asociadas mediante la plataforma web de Google Maps (www.google.com/maps), desde la cual se extrae la información sobre el tiempo de demora del tránsito vehicular.

La obtención de rutas, tomando como ejemplo la Av. Sánchez Cerro en la ciudad de Piura, comprende los siguientes pasos:

b.1) Se presiona la opción “Cómo llegar”, ubicada en el buscador de Google Maps, tal como se muestra en la Figura N° 1.

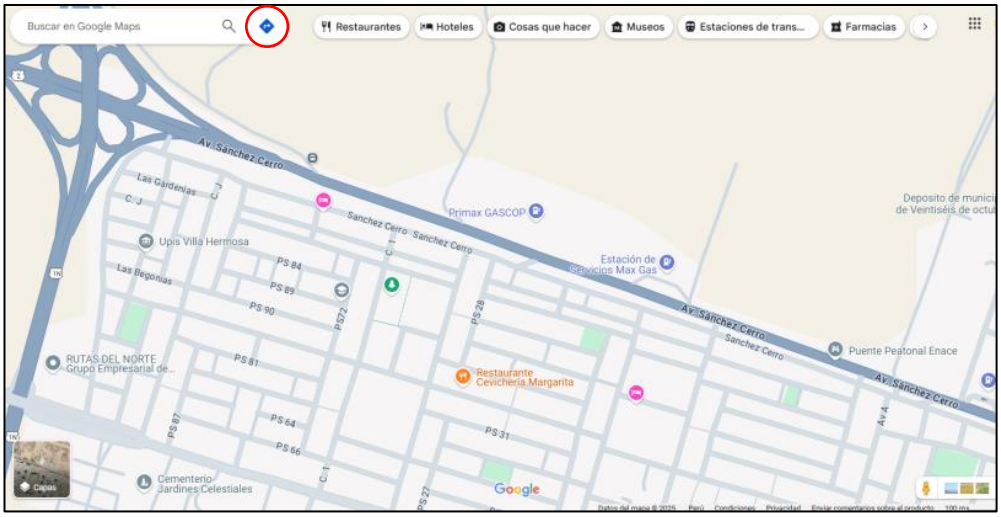

Figura N° 1. Vista de Google Maps que resalta (en círculo) la opción “Como llegar”.

b.2) Se coloca el punto de partida (1) y de llegada (2) de la ruta a definir. Estos pueden establecerse ingresando los nombres de los lugares o haciendo clic directamente en el mapa. Es importante considerar que ambos puntos deben encontrarse en la misma dirección de la vía, tal como se muestra en las Figura N° 2 y N° 3.

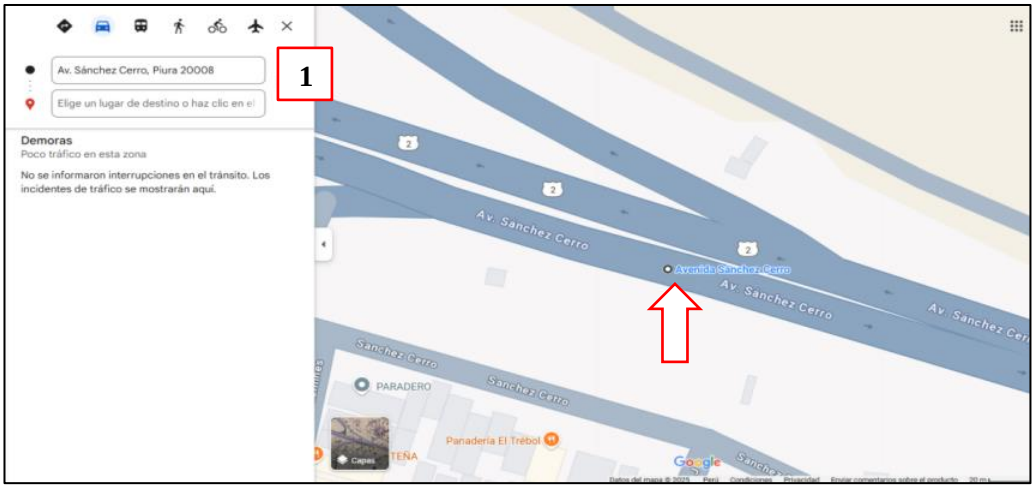

Figura N° 2. Vista de Google Maps que resalta la creación del punto de partida de la ruta.

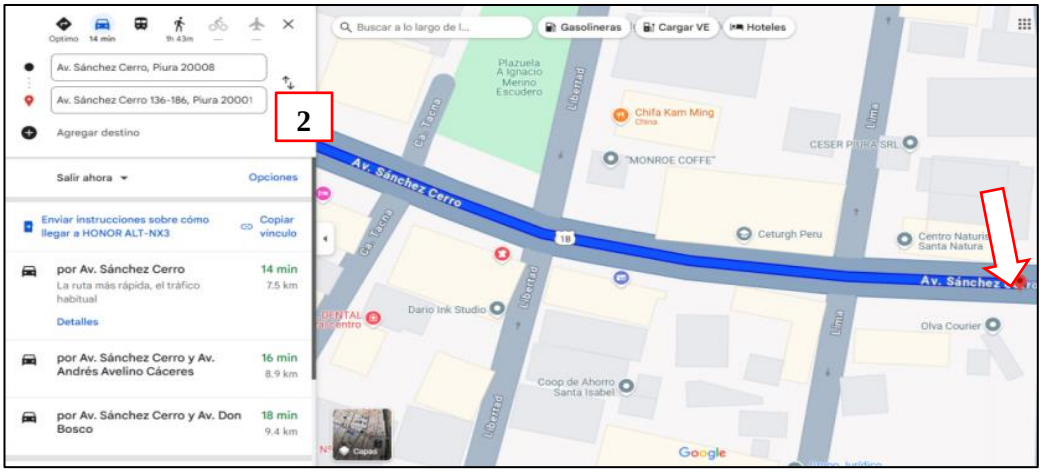

Figura N° 3. Vista de Google Maps que resalta la creación del punto de llegada de la ruta.

b.3) Definida la ruta, se visualiza el tiempo de demora entre ambos puntos. Posterior a ello, se copia su respectiva URL y se guarda de manera provisional en un bloc de notas o archivo similar, tal como se muestra en la Figura N° 4 y 5.

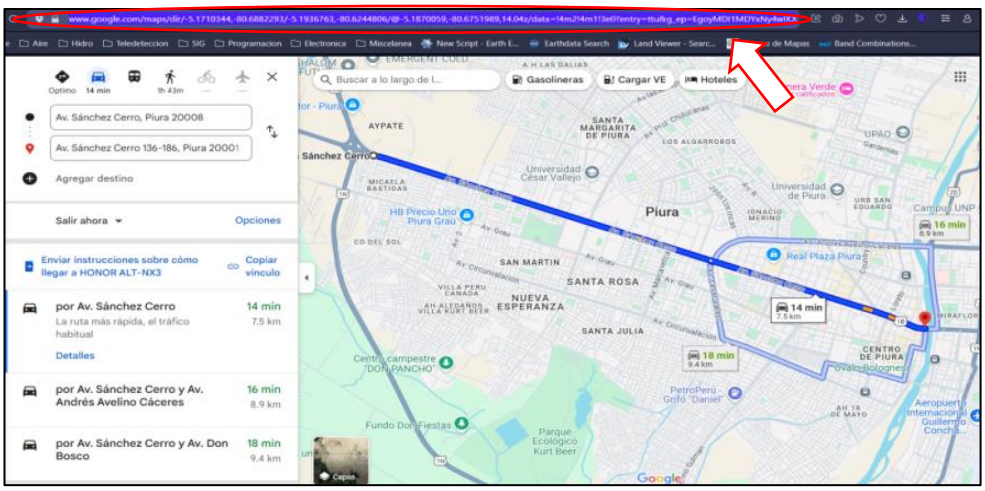

Figura N° 4. Vista de Google Maps que resalta el URL correspondiente a la ruta creada.

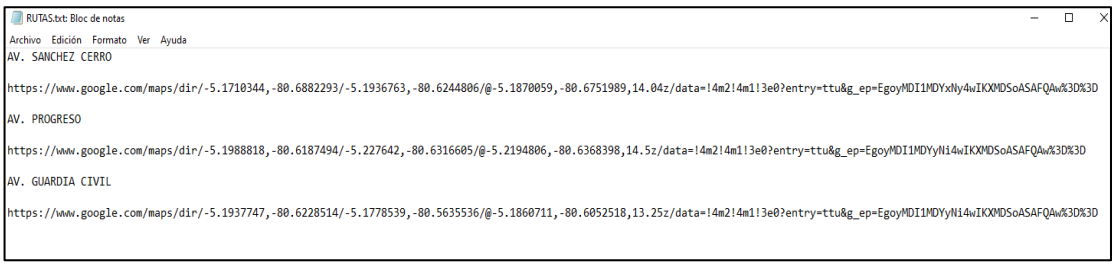

Figura N° 5. Vista de Block de notas que contiene las URL de las rutas creadas.

c) Obtención del identificador (*XPath*) que guarda la información del tiempo de demora en Google Maps

Obtenidas las rutas seleccionadas, se identifica el elemento HTML que contiene la información a capturar de la página web de Google Maps, siguiendo los siguientes pasos:

c.1) Se inspecciona el elemento que contiene el tiempo de demora, haciendo clic derecho y seleccionando la opción “Inspeccionar elemento”, tal como se muestra en la Figura N° 6.

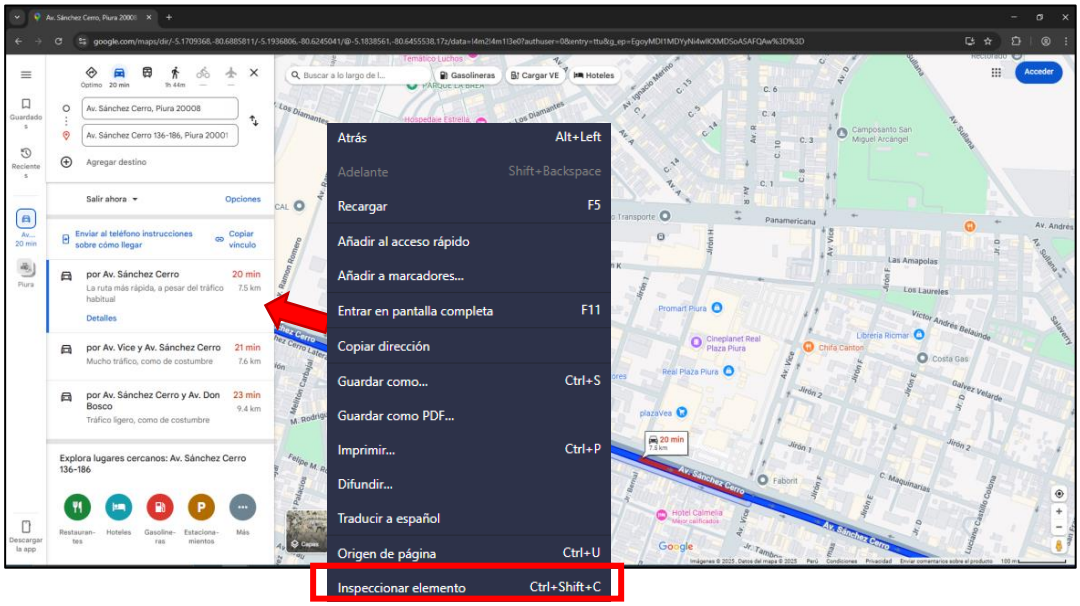

Figura N° 6. Vista de la opción “*Inspeccionar elemento*” en el navegador que muestra el Google Maps.

c.2) Al dar clic en la opción “Inspeccionar elemento”, se abrirá un panel al lado donde se resalta el elemento inspeccionado. Con el fin de copiar el identificador (XPath) del elemento HTML, se realiza un clic derecho sobre este y se selecciona la opción “Copy”, seguida de la opción “Copy XPath”, tal como se muestra en la Figura N° 7. Este XPath se guarda posteriormente para su uso en el script de captura de la información del tiempo de demora.

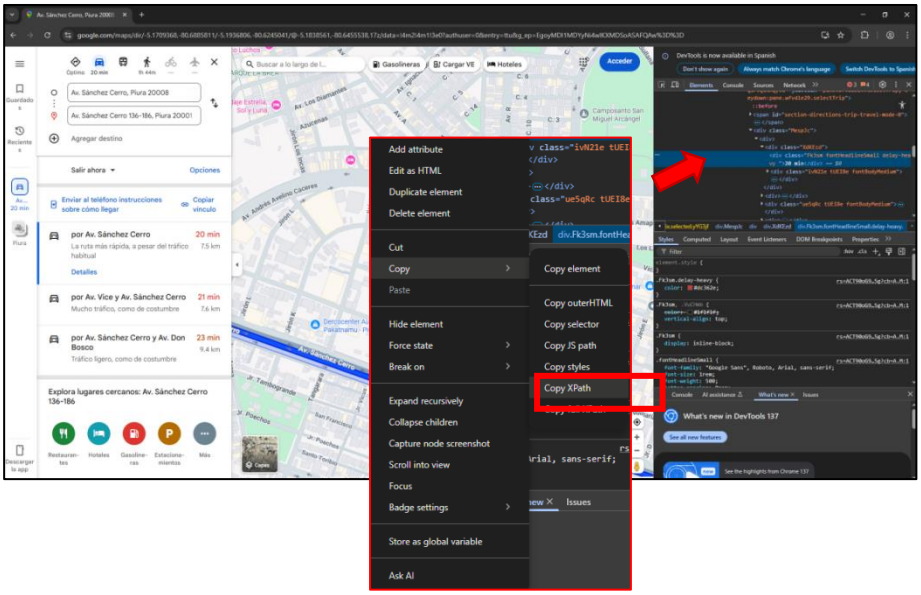

Figura N° 7. Vista de la opción "*Copy Xpath*" en el navegador que muestra el Google Maps.

d) Extracción del valor del tiempo de demora para cada ruta requerida:

Se elabora un script o secuencia de líneas de códigos en el lenguaje de programación Python, empleando la técnica de Web Scraping mediante librerías especializadas. Para tal fin, se siguen los siguientes pasos:

d.1) Se instalan las librerías que contienen los módulos o herramientas necesarias para realizar la captura de la información, en caso no se cuente con ellas. Dichas librerías son: *selenium, pandas, openpyxl, datetime, time y os*. Para ello, se utiliza el siguiente comando en la consola de Python: "*pip install nombre_del_módulo*". Por ejemplo: *pip install selenium*.

d.2) Se realiza la importación o activación de los módulos de las librerías, tal como se muestra en la Figura N° 8:

```
# Modulos importados

from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
import pandas as pd
from datetime import datetime
import os
import time

```

Figura N° 8. Vista de las líneas de código de Python que permiten activar los módulos de las librerías.

Donde:

- Línea 3: Importa el módulo *webdriver* de Selenium, el cual permite controlar un navegador web (como Chrome, Firefox, etc.) de forma automática mediante programación. En este caso, se utiliza para abrir una ventana de Google Chrome, navegar hacia una URL y extraer información.
- Línea 4: Importa la clase *Options* que permite configurar la forma en que se ejecuta el navegador Chrome. En este caso, se emplea para ejecutarlo en modo invisible (*headless*), es decir, en segundo plano.
- Línea 5: Importa la clase *By*, que se usa para especificar el método mediante el cual se localizará un elemento HTML.
- Línea 6: Importa la librería *pandas*, una de las más potentes para el análisis y la manipulación de datos en Python.
- Línea 7: Importa la clase *datetime* del módulo estándar *datetime*, la cual permite obtener la fecha y hora actual del sistema, y formatearlas según sea necesario.
- Línea 8: Importa el módulo *os*, el cual provee funciones para interactuar con el sistema operativo.
- Línea 9: Importa el módulo *time*, el cual proporciona funciones para gestionar tiempos de espera, fechas y relojes.

d.3) Se define la carpeta donde se guarda los resultados. Para ello, se copia la ruta de la carpeta desde el explorador de archivos, y se ejecuta el siguiente comando, tal como se muestra en la Figura N° 9:

```
11    # Definir la carpeta de trabajo
12    os.chdir(r'C:\Users\usuario1\Documents\SEA\trafico')
```

Figura N° 9. Vista de las líneas de código de Python que permiten definir la carpeta de trabajo.

d.4) Se define una función para la búsqueda de la información del tiempo de demora, una vez se cargue la URL de la ruta en el navegador Google Chrome, tal como se presenta en la Figura N° 10:

```
14  def find_element_by_xpath(url, xpath, zona, data_list):
15      chrome_options = Options()
16      chrome_options.add_argument("--headless")
17      # Configurar el navegador
18      driver = webdriver.Chrome(options=chrome_options)
19
20      try:
21          # Cargar la página
22          driver.get(url)
23          driver.implicitly_wait(10)
24
25          # Obtener el elemento y sus propiedades
26          element = driver.find_element(By.XPATH, xpath)
27          text = element.text
28          color = element.value_of_css_property("color")
29
30          print(f"Zona: {zona} | Tiempo: {text} | Color: {color}")
31
32          # Guardar resultado
33          new_row = {'Zona': zona, 'Tiempo de demora': text,
34                     'Color de demora': color, 'date': datetime.now()}
35          data_list.append(new_row)
36
37      except Exception as e:
38          print(f"Error en zona '{zona}': {e}")
39      finally:
40          driver.quit()
```

Figura N° 10. Vista de las líneas de código de Python que corresponde a una función que permite abrir la URL de la ruta.

Donde:

- Línea 14: Define la función que abre una URL de Google Maps, obtiene el tiempo de demora del tránsito vehicular en la ruta, y lo guarda en una lista. Para ello, requiere como argumentos a la URL de la ruta, el *XPath* del elemento HTML que contiene el tiempo de demora, la zona o nombre de la vía del tránsito, y la lista de Python donde se almacenará la información.
- Línea 15 a la 18: Configuran y abre el navegador Google Chrome en modo invisible (*headless*), es decir, ejecutándose en segundo plano sin necesidad de visualizar la ventana del navegador.
- Dentro de la función se incluyen líneas de código que permiten detectar y comunicar errores durante la ejecución. Para tales fines, se emplea una estructura de control como la que se muestra en las líneas 20 y de la 37 a la 40. En caso de fallo, el programa imprime el error y, al finalizar, cierra el navegador para liberar memoria.
- Líneas 22 y 23: Contienen instrucciones que permiten abrir la URL de la ruta y esperar hasta 10 segundos para que los elementos de la página se carguen completamente.
- Líneas 26 a la 28: Buscan el elemento HTML que contiene el tiempo estimado de tránsito vehicular, extraen el texto (por ejemplo, “12 min”) y el color asociado en formato RGBA (por ejemplo, rgba (255, 0, 0, 1)), el cual caracteriza cualitativamente el tiempo de demora.
- Línea 30: Muestra en la consola de Python los datos capturados para cada ruta definida, incluyendo la zona o nombre de la vía, el tiempo de demora y el color en codificación RGBA.

- Líneas 33 a la 35: Crean un objeto tipo diccionario con los datos capturados y lo añaden a una lista denominada **"data_list"**, la cual se define previamente como vacía para ir incorporando los registros obtenidos en cada ejecución de la función.

d.5) Una vez que se tenga definida la función de captura, se elabora la función principal del script, el cual contendrá los argumentos necesarios para ejecutar la función de captura, así como las acciones complementarias para automatizar el proceso, tal como se muestra en la Figura N° 11:

```
42  def main():
43      data = [
44          {"url": "https://www.google.com/maps/dir/-5.1710344,-80.6882293/-5.1936763,-80
45           "zona": "Avenida Sanchez Cerro"},
46          {"url": "https://www.google.com/maps/dir/-5.1988818,-80.6187494/-5.227642,-80.
47           "zona": "Avenida Progreso"},
48          {"url": "https://www.google.com/maps/dir/-5.1937747,-80.6228514/-5.1778539,-80
49           "zona": "Avenida Guardia Civil"},
50      ]
51
52      xpath = '//*[@id="section-directions-trip-0"]/div[1]/div/div[1]/div[1]'
53      data_list = []
54
55      for item in data:
56          find_element_by_xpath(item["url"], xpath, item["zona"], data_list)
57
58      df = pd.DataFrame(data_list)
59
60      try:
61          existing_df = pd.read_excel('tiempo_demora.xlsx')
62          df = pd.concat([existing_df, df], ignore_index=True)
63      except FileNotFoundError:
64          pass
65
66      df.to_excel('tiempo_demora.xlsx', index=False)
67      print(f"Datos guardados a las {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
```

Figura N° 11. Vista de las líneas de código de Python que corresponde a una función que permite capturar información de la URL de la ruta.

Donde:

- Línea 43 a la 50: Se definen, mediante una lista de diccionarios denominada **"data"**, las zonas o nombres de las rutas (según corresponda) junto con sus respectivas URL, las cuales serán utilizadas para la obtención de la información del tránsito vehicular desde Google Maps.
- Línea 52: Define el *XPath* del elemento HTML, que contiene la información del tiempo de demora para cada ruta en el portal web de Google Maps, el cual fue previamente identificado y guardado según lo descrito en el ítem **5.2.3**.
- Línea 53: Define un objeto tipo lista de Python denominado **"data_list"** como vacío para que pueda guardar la información capturada durante la ejecución del script.
- Línea 55 y 56: Realizan una iteración que ejecuta la función de captura para cada uno de los diccionarios definidos en la lista de

diccionarios **"data"**, utilizando el *XPath* y el objeto **"data_list"** previamente declarados.

- Línea 58: Convierte el objeto **"data_list"** en un objeto DataFrame de Python, permitiendo estructurar la información capturada en formato tabular para su posterior análisis o almacenamiento.
- Línea 60 a la 64: Lee un archivo Excel existente llamado "tiempo_demora.xlsx" con el fin de agregar los nuevos registros obtenidos en la ejecución actual. El contenido previo del archivo denominado como **"existing_df"** se combina con los nuevos datos **"df"**. Si el archivo no existe, se captura la excepción correspondiente, omitiendo el paso de concatenación y empleando únicamente el DataFrame recién generado **"df"**.
- Línea 66: Guarda el contenido del DataFrame **"df"** en un archivo Excel con el nombre **"tiempo_demora.xlsx"**, actualizando o creando el archivo según corresponda.
- Línea 67: Imprime un mensaje de confirmación indicando que el archivo fue actualizado correctamente e incluye el registro de fecha y hora de la operación.

d.6) Para finalizar esta parte, se programa la ejecución automática del script en intervalos de tiempo definidos, para lo cual se implementa las líneas de código que se muestran en la Figura N° 12:

```
69  if __name__ == "__main__":
70      while True:
71          main()
72          print("Esperando 1 hora...\n")
73          time.sleep(3600)
```

Figura N° 12. Vista de las líneas de código de Python que corresponde a una función que permite capturar información de la URL de la ruta cada cierto intervalo de tiempo.

Donde:

- Línea 69: Permite que el bloque de código se ejecute únicamente cuando el archivo es ejecutado directamente, y no cuando es importado como módulo en otro script.
- Línea 70: Establece un ciclo que mantiene la ejecución del código de manera continua, repitiéndose indefinidamente hasta que se detenga manualmente. Esta detención puede realizarse presionando el atajo Ctrl + C en la terminal o utilizando la opción de detener el kernel de Python en el entorno de desarrollo integrado empleado (por ejemplo, *Jupyter Notebook*, *Spyder* o *Visual Studio Code*).
- Línea 71: Realiza el llamado a la función principal *main()*, la cual ejecuta todo el proceso.

- Línea 72: Imprime en la consola un mensaje indicando que el script entrará en pausa durante una hora antes de volver a ejecutarse.
- Línea 73: Detiene temporalmente la ejecución durante 3600 segundos, equivalentes a una hora completa, antes de reiniciar el ciclo de ejecución.

d.7) Una vez ejecutado el script descrito[8], se obtienen registros con la información capturada del tiempo de demora del tránsito vehicular, los cuales se presentan de manera tabular en un archivo Excel denominado **"tiempo_demora.xlsx".** Un ejemplo de dichos registros se muestra en la Figura N° 13:

	A	B	C	D
1	Zona	Tiempo de demora	Color de demora	date
2	Avenida Sanchez Cerro	12 min	rgba(217, 48, 37, 1)	07:58.3
3	Avenida Sanchez Cerro	15 min	rgba(176, 91, 0, 1)	08:08.9
4	Avenida Sanchez Cerro	6 min	rgba(32, 33, 36, 1)	08:19.2
5	Avenida Sanchez Cerro	5 min	rgba(176, 91, 0, 1)	08:30.5
6	Avenida Sanchez Cerro	3 min	rgba(24, 128, 56, 1)	08:40.8
7	Avenida Sanchez Cerro	2 min	rgba(217, 48, 37, 1)	08:53.4
8	Avenida Sanchez Cerro	3 min	rgba(32, 33, 36, 1)	09:07.3
9	Avenida Progreso	10 min	rgba(24, 128, 56, 1)	09:18.9
10	Avenida Progreso	12 min	rgba(32, 33, 36, 1)	09:29.7
11	Avenida Progreso	4 min	rgba(32, 33, 36, 1)	09:41.2
12	Avenida Progreso	13 min	rgba(32, 33, 36, 1)	09:53.7
13	Avenida Progreso	17 min	rgba(32, 33, 36, 1)	10:05.0
14	Avenida Guardia Civil	4 min	rgba(176, 91, 0, 1)	10:17.1
15	Avenida Guardia Civil	3 min	rgba(176, 91, 0, 1)	10:29.5
16	Avenida Guardia Civil	2 min	rgba(24, 128, 56, 1)	10:40.5
17	Avenida Guardia Civil	2 min	rgba(24, 128, 56, 1)	10:52.1
18	Avenida Guardia Civil	5 min	rgba(217, 48, 37, 1)	11:03.2
19	Avenida Guardia Civil	4 min	rgba(176, 91, 0, 1)	11:14.7
20	Avenida Guardia Civil	1 min	rgba(24, 128, 56, 1)	11:26.1
21	Avenida Guardia Civil	1 min	rgba(24, 128, 56, 1)	11:37.6
22	Avenida Guardia Civil	1 min	rgba(217, 48, 37, 1)	11:56.4

Figura N° 13. Vista de los registros resultantes de la ejecución del script de captura de *información del tiempo de demora en las rutas seleccionadas.*

2.4.2. Elaboración de gráficos del tiempo de demora respecto al tránsito vehicular

a) Elaboración de gráfico del tiempo de demora para todas las rutas consideradas:

a.1) De manera similar al proceso de captura de información, el primer paso consiste en activar las librerías necesarias, las cuales son: *pandas, matplotlib, seaborn, date time y os*. En caso de no contar con alguna de estas librerías, se instalan con el siguiente comando en la consola de python: "*pip install nombre_del_módulo*". Por ejemplo: *pip install matplotlib*.

[8] Para poder descargar el script de captura de información del tiempo de demora del tránsito vehicular, puede ingresar al siguiente enlace: https://github.com/SENAMHI-SEA/Trafico_vehicular/blob/main/Get_Time_traffic.py

a.2) Las librerías y sus respectivos módulos, se activan mediante la importación en el entorno de trabajo, tal como se muestra en la Figura N° 14:

```
4   import pandas as pd
5   import matplotlib.pyplot as plt
6   from matplotlib.dates import DateFormatter
7   import seaborn as sns
8   from datetime import timedelta
9   import os
```

Figura N° 14. Vista de la activación de las librerías y módulos para generar el gráfico de tiempos de demora de las rutas seleccionadas.

Donde:

- Línea 4: Importa la librería *pandas* que permite manipular y analizar datos mediante estructuras tipo *DataFrame*.
- Línea 5: Importa el módulo *pyplot* de *matplotlib* (referido como *plt*), utilizado para la generación de gráficos.
- Línea 6: Importa la clase *DateFormatter* de *matplotlib*, empleada para el formateo de fechas en los ejes de los gráficos.
- Línea 7: Importa la librería *seaborn*, que permite mejorar la presentación visual de los gráficos basados en *matplotlib*.
- Línea 8: Importa la clase *timedelta* del módulo *datetime*, que permite realizar operaciones con diferencias de tiempo.
- Línea 9: Importa el módulo estándar *os*, el cual proporciona funciones para interactuar con el sistema operativo.

a.3) De manera opcional se pueden establecer configuraciones globales de estilo para los gráficos, a fin de mantener una presentación visual uniforme en todas las figuras, tal como se muestra en la Figura N° 15:

```
13  # Configuración global
14  sns.set(style='whitegrid')
15  plt.rcParams["figure.autolayout"] = True
```

Figura N° 15. Vista de las líneas de código para configuración global del estilo de los gráficos de tiempos de demora de las rutas seleccionadas.

Donde:

- Línea 14: Aplica un fondo con cuadrícula blanca a todos los gráficos mediante *seaborn*, mejorando la legibilidad visual.
- Línea 15: Ajusta automáticamente los márgenes de los gráficos para evitar el solapamiento de los elementos.

a.4) Asimismo, como parte de los insumos del script, es necesario definir parámetros que pueden ser ajustados para su réplica, tal como se muestran en la Figura N° 16:

```
17  # --- Parámetros ajustables ---
18  EXCEL_PATH = r'C:\Users\usuario1\Documents\SEA\trafico\tiempo_demora.xlsx'
19  TIME_COLUMN = 'Tiempo de demora'
20  DATE_COLUMN = 'date'
21  RUTAS = ["Avenida Sanchez Cerro", "Avenida Progreso", "Avenida Guardia Civil"]
```

Figura N° 16. Vista de las líneas de código para configuración global del estilo de los gráficos de tiempos de demora de las rutas seleccionadas.

Donde:

- Línea 18: Declara la ruta completa del archivo Excel que contiene los datos del tiempo de demora.
- Línea 19 y 20: Especifican los nombres de las columnas del archivo Excel, las cuales serán utilizadas dentro del script.
- Línea 21: Define las rutas seleccionadas correspondientes a la ZAP, sobre las cuales se pueden generar los gráficos individuales.

a.5) Posterior a ello, se define una función para lectura de la información de tiempo de demora a fin de elaborar el gráfico, tal como se muestra en la Figura N° 17:

```
23  # --- Funciones ---
24  def load_and_clean_data(path):
25      df = pd.read_excel(path)
26
27      df[TIME_COLUMN] = df[TIME_COLUMN].str.replace('min', '', regex=False).astype(float)
28
29      df[DATE_COLUMN] = pd.to_datetime(df[DATE_COLUMN], errors='coerce')
30
31      latest_date = df[DATE_COLUMN].max().date()
32      recent_days = [latest_date - timedelta(days=i) for i in range(3)]
33      df['only_date'] = df[DATE_COLUMN].dt.date
34      df = df[df['only_date'].isin(recent_days)].copy()
35      df.drop(columns='only_date', inplace=True)
36
37      df['hour'] = df[DATE_COLUMN].dt.hour
38      df['day'] = df[DATE_COLUMN].dt.date
39
40
41      return df
```

Figura N° 17. Vista de las líneas de código que conforman la función de lectura de la información del tiempo de demora.

Donde:

- Línea 25: Lee el archivo Excel y carga sus datos en un DataFrame **"df".**
- Línea 27: Elimina la palabra "min" de la columna de tiempo de demora y convierte sus valores a tipo numérico (*float*).
- Línea 29: Convierte la columna de fecha a tipo *datetime* para facilitar su manipulación temporal.
- Línea 31 y 32: Detectan el día más reciente en los datos y genera una lista con los últimos tres días (esto es modificable dependiendo de las necesidades del usuario).

- Línea 33 a la 35: Crean una columna auxiliar “only_date” para filtrar el DataFrame a fin de que contenga solo esos 3 días, y luego se elimina la columna auxiliar.
- Línea 37 y 38: Crean columnas auxiliares que contienen la hora (*hour*) y la fecha (*day*).
- Línea 41: Devuelve el DataFrame **“df”** con todas las modificaciones aplicadas.

a.6) Asimismo, se define una función que se encarga de agrupar los datos y organizar la tabla para graficar, la cual se muestra en la Figura N° 18:

```
44  def aggregate_and_pivot(df):
45      df_avg = (
46          df.groupby(['Zona', 'day', 'hour'])[TIME_COLUMN]
47          .mean()
48          .reset_index(name='avg_delay')
49      )
50      df_pivot = df_avg.pivot_table(index=['day', 'hour'], columns='Zona',
51                                    values='avg_delay').reset_index()
52      df_pivot = df_pivot.sort_values(by=['day', 'hour'])
53      df_pivot['date'] = pd.to_datetime(df_pivot['day'].astype(str) + ' ' +
54                                        df_pivot['hour'].astype(str) + ':00:00')
55      return df_pivot
```

Figura N° 18. Vista de las líneas de código que conforman la función de agregar y organizar la información del tiempo de demora.

Donde:

- Línea 45 a la 49: Agrupan los datos por zona o ruta, día y hora, calculando el promedio del tiempo de demora, ello a fin de tener un valor representativo por hora.
- Línea 50 y 51: Convierten el DataFrame a formato ancho, con una columna por cada Zona.
- Línea 52: Ordena los datos de manera cronológica.
- Línea 53 y 54: Crean una columna con la combinación de fecha y hora fecha completa y la hora, la cual se utiliza como eje x en el gráfico.
- Línea 55: Devuelve el DataFrame **“df_pivot”** en formato ancho con todas las modificaciones aplicadas.

a.7) Otra función que se define es la que calcula los límites de medianoche, con el propósito de diferenciar de manera más clara los días en los gráficos, tal como se muestra en la Figura N° 19:

```
58  def get_midnights(df):
59      start_date = df['date'].min().floor('D')
60      end_date = df['date'].max().ceil('D')
61      return pd.date_range(start=start_date, end=end_date, freq='D')
```

Figura N° 19. Vista de las líneas de código que conforman la función de calcular los límites de medianoche.

Donde:

- Línea 59 y 60: Identifican el primer y último día completo del conjunto de datos.
- Línea 61: Devuelve una serie de fechas a medianoche (00:00) que se utilizan para marcar los límites diarios en el gráfico.

a.8) De igual manera, para elaborar el gráfico del tiempo de demora de las rutas seleccionadas, se define la función que se muestra en la Figura N° 20:

```
64  def plot_multiple_stations(df_pivot, midnights, routes, save_path=None, ymin=None, ymax=None):
65      n = len(routes)
66      cols = 1
67      rows = (n + cols - 1) // cols
68      fig, axes = plt.subplots(rows, cols, figsize=(12, 4 * rows), sharex=True, sharey=True)
69      axes = axes.flatten()
70
71      for i, route in enumerate(routes):
72          ax = axes[i]
73          if route not in df_pivot.columns:
74              print(f"[Advertencia] {route} no está en los datos.")
75              ax.set_visible(False)
76              continue
77
78          sns.lineplot(x=df_pivot['date'], y=df_pivot[route], ax=ax, marker='o', color='blue')
79          ax.set_title(route, fontsize=16, weight='bold')
80          ax.set_ylabel("Tiempo de demora (min)", fontsize=16, weight='bold')
81          ax.set_xlabel("Fecha y hora", fontsize=16, weight='bold')
82          ax.xaxis.set_major_formatter(DateFormatter('%d-%b %H:%M'))
83          ax.tick_params(axis='x', labelrotation=45)
84          ax.tick_params(axis='both', labelsize=14)
85          ax.grid(True)
86          ax.figure.tight_layout()
87
88          if ymin is not None and ymax is not None:
89              plt.ylim(ymin, ymax)
90
91          for m in midnights:
92              ax.axvline(x=m, color='red', linestyle='dashed', linewidth=0.8)
93
94      for j in range(i + 1, len(axes)):
95          fig.delaxes(axes[j])
96
97      if save_path:
98          plt.savefig(save_path, format='jpg', dpi=300, bbox_inches='tight')
99          print(f"[Info] Panel guardado en: {save_path}")
100     plt.tight_layout()
101     plt.show()
```

Figura N° 20. Vista de las líneas de código que conforman la función de elaborar la gráfica del tiempo de demora.

Donde:

- Línea 64: Define la función y parámetros como el *DataFrame* en formato ancho con los tiempos promedio por fecha, hora y ruta (**df_pivot**), la lista de fechas a medianoche (**midnights**), la lista de nombres de rutas a graficar (**routes**), la ruta opcional donde se guarda el panel de gráficos en formato JPG (**save_path**), y los límites opcionales del eje *Y* (**ymin y ymax**).
- Línea 65 a la 67: Determinan cuántas rutas se deben graficar, establecen el número de columnas del panel (en este caso, 1, aunque puede modificarse según necesidad del usuario) y calculan el número de filas requeridas para acomodar todos los gráficos.

- Línea 68: Crea el panel con subgráficos, configurando que todos compartan los mismos ejes.
- Línea 69: Convierte la matriz *axes* a un arreglo unidimensional (1D) para facilitar la iteración.
- Línea 71 y 72: Iteran las rutas y asignan un subgráfico (*ax*) para cada una.
- Línea 73 a la 76: Verifican si la ruta se encuentra presente en el DataFrame; caso contrario, muestran una advertencia, ocultan el subgráfico correspondiente y continúan con el siguiente.
- Línea 78: Traza la serie temporal con la fecha en el eje *X* y el tiempo de demora en el eje *Y*, utilizando un marcador tipo punto por hora y color azul.
- Línea 79: Asigna un título al gráfico con el nombre de la ruta.
- Línea 80 y 81: Coloca títulos a los ejes Y y X.
- Línea 82: Aplica un formato personalizado a las fechas del eje *X* (por ejemplo, “06-Jul 12:00”).
- Línea 83: Rota a 45° las etiquetas del eje X para mejor legibilidad.
- Línea 84: Ajusta el tamaño de las etiquetas de ambos ejes.
- Línea 85: Activa la cuadrícula de grillas en el gráfico.
- Línea 86: Ajusta la disposición del gráfico dentro del panel.
- Línea 88 y 89: Configuran los límites del eje Y para todos los subgráficos.
- Línea 91 y 92: Dibujan las líneas verticales en cada medianoche para marcar la separación entre días.
- Línea 94 y 95: Eliminan los subgráficos vacíos en caso de que haya menos rutas que espacios disponibles.
- Línea 97 y 98: Guardan el gráfico en un archivo .jpg en la ruta de la carpeta de trabajo.
- Línea 100 y 101: Ajustan automáticamente los márgenes del panel y muestran en pantalla el panel completo de gráficos.

a.9) Una vez definidas las funciones específicas, se crea la función principal que hace referencia a las demás para su ejecución, tal como se muestra en la Figura N° 21:

```
102  def main_panel():
103      nombre_archivo = "panel_trafico.jpg"
104      base_dir = os.path.dirname(EXCEL_PATH)
105      save_path = os.path.join(base_dir, nombre_archivo)
106
107      if not os.path.exists(EXCEL_PATH):
108          print(f"[Error] Archivo no encontrado: {EXCEL_PATH}")
109          return
110
111      df = load_and_clean_data(EXCEL_PATH)
112      df_pivot = aggregate_and_pivot(df)
113      midnights = get_midnights(df_pivot)
114
115      plot_multiple_stations(df_pivot, midnights, routes=RUTAS, save_path=save_path,
116                             ymin=0, ymax=14)
117
118
119  if __name__ == "__main__":
120      main_panel()
```

Figura N° 21. Vista de las líneas de código que conforman la función principal que hace el llamado a las funciones específicas para su ejecución

Donde:

- Línea 103: Define el nombre del archivo de salida para el panel de gráficos.
- Línea 104 y 105: Crean la ruta completa del panel de gráficos a partir de la carpeta que contiene el archivo Excel.
- Línea 107 a la 109: Verifican si el archivo Excel existe, de no ser el caso, imprime un mensaje de error y detiene la ejecución de la función.
- Línea 111 a la 116: Hacen el llamado a las funciones específicas con sus respectivos argumentos para su ejecución.
- Línea 119 y 120: Ejecutan directamente la función principal directamente, separando así el código que se debe ejecutar automáticamente del que únicamente define funciones, clases u otros elementos del script.

a.10) Al ejecutarse todas las funciones definidas previamente, se genera el respectivo gráfico del tiempo de demora, el cual se presenta en la Figura N° 22. Este gráfico permite visualizar de manera clara la evolución del tiempo de tránsito por cada ruta seleccionada, facilitando el análisis y la comparación entre diferentes vías.

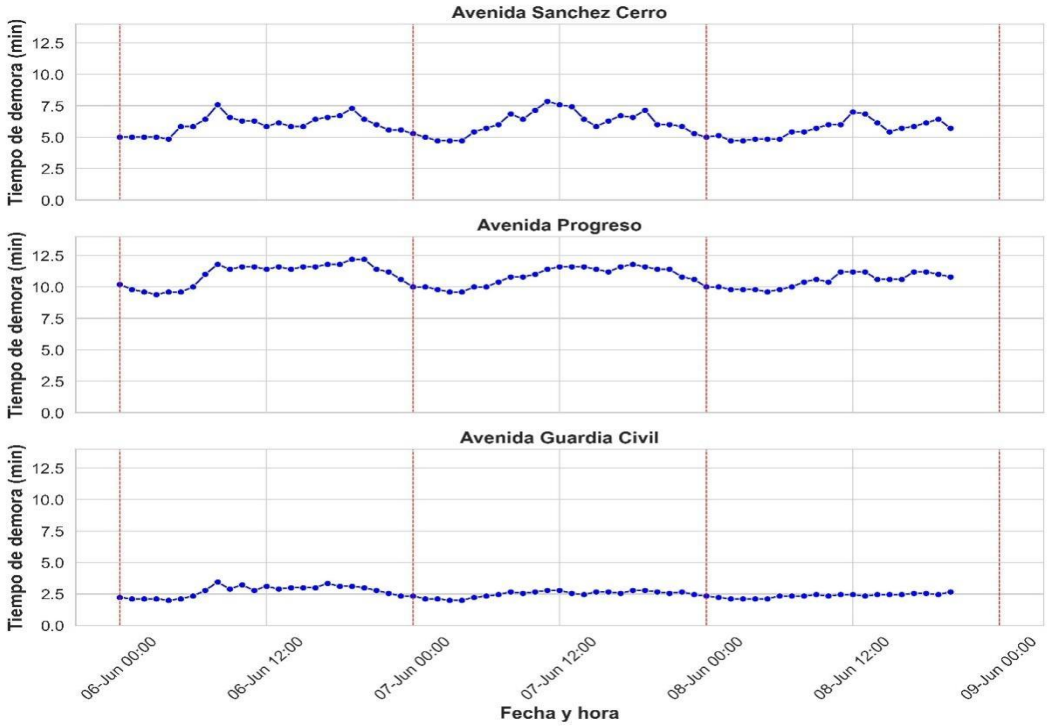

Figura N° 22. Gráfico del tiempo de demora a nivel horario de las rutas seleccionadas para la ZAP de análisis.

b) Elaboración del gráfico del tiempo de demora para una ruta en específico

b.1) En caso que se requiera elaborar el gráfico para una única ruta en específico, una vez ejecutados los pasos descritos en los literales a) hasta la g) del ítem 5.3.1, se debe modificar la función específica del graficado, sin considerar iteraciones, tal como se muestra en la Figura N° 23:

```
126  def plot_single_station(df_pivot, midnights, route_name, save_path=None):
127      plt.figure(figsize=(14, 6))
128      sns.lineplot(data=df_pivot, x='date', y=route_name, color='blue', marker='o')
129
130      plt.title(route_name, fontsize=14, weight='bold')
131      plt.xlabel("Fecha y Hora", fontsize=14, weight='bold')
132      plt.ylabel("Tiempo de demora (minutos)", fontsize=14, weight='bold')
133      plt.xticks(rotation=0)
134      plt.grid(True)
135      plt.gca().xaxis.set_major_formatter(DateFormatter('%d-%b %H:%M'))
136
137      for m in midnights:
138          plt.axvline(x=m, color='red', linestyle='dashed', linewidth=0.8)
139
140      if save_path:
141          plt.savefig(save_path, format='jpg', dpi=300)
142          print(f"[Info] Gráfico guardado en: {save_path}")
143
144      plt.tight_layout()
145      plt.show()
```

Figura N° 23. Vista de las líneas de código modificadas que conforman la función que elabora el gráfico para una única ruta de la ZAP de análisis.

b.2) De igual modo, la función principal que llama a otras funciones específicas también tendría que ser modificada tal como se muestra en la Figura N° 24:

```
148  def main_individual():
149
150      ruta = "Avenida Progreso"
151      nombre_archivo = f"{ruta.replace(' ', '_').lower()}.jpg"
152      base_dir = os.path.dirname(EXCEL_PATH)
153      save_path = os.path.join(base_dir, nombre_archivo)
154
155      if not os.path.exists(EXCEL_PATH):
156          print(f"[Error] Archivo no encontrado: {EXCEL_PATH}")
157          return
158
159      df = load_and_clean_data(EXCEL_PATH)
160      df_pivot = aggregate_and_pivot(df)
161      midnights = get_midnights(df_pivot)
162
163      if ruta not in df_pivot.columns:
164          print(f"[Error] La ruta '{ruta}' no se encuentra en los datos.")
165          return
166
167      plot_single_station(df_pivot, midnights, route_name=ruta, save_path=save_path)
168
169
170  if __name__ == "__main__":
171      main_individual()
```

Figura N° 24. Vista de las líneas de código modificadas que conforman la función principal que hace el llamado a funciones específicas para elaborar el gráfico de una única ruta de la ZAP de análisis.

Donde:

- Línea 150: Define el nombre de la ruta que se va a graficar.
- Línea 163 a la 165: Verifican si el nombre de la ruta se encuentra en el DataFrame con los datos capturados, de no ser el caso, imprime un mensaje de error.

b.3) Luego de ejecutar la función principal, se obtiene el gráfico individual de la ruta seleccionada tal como se visualiza en la Figura N° 25. Este gráfico permite visualizar de manera específica la evolución del tiempo de demora en la ruta elegida, facilitando su análisis detallado.

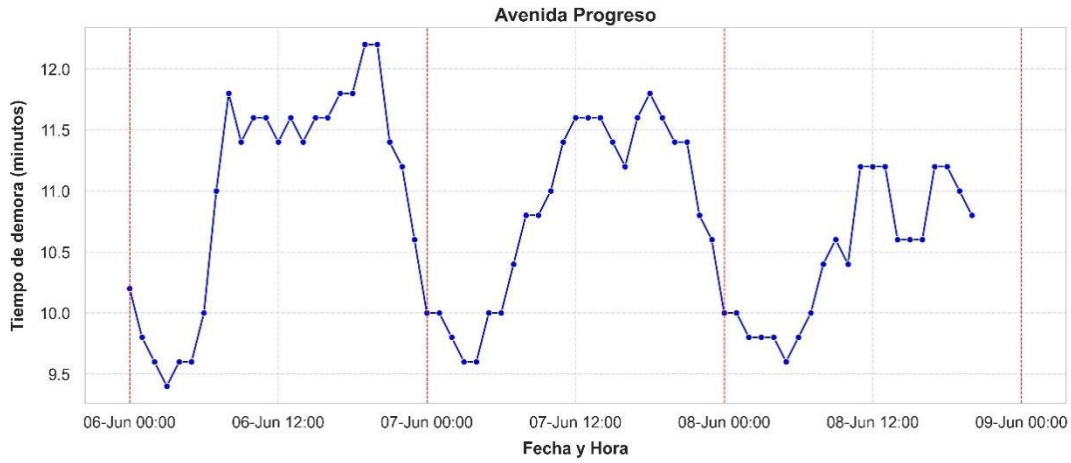

Figura N° 25. Gráfico del tiempo de demora a nivel horario de una ruta seleccionada para la ZAP de análisis.

En cuanto al diseño del gráfico se debe precisar que, el tamaño de los títulos, etiquetas de los ejes, colores de líneas y puntos, entre otros detalles, puede modificarse a requerimiento del usuario. En el presente documento solo se presenta un ejemplo de script[9] para generación de ambos gráficos descritos sobre el tiempo de demora en rutas de tránsito vehicular, el cual puede ser utilizado para la ZAP de análisis.

3. TABLA DE CONTROL DE CAMBIOS

Versión	Sección	Detalle de cambios
01	---	Versión inicial

[9] Para poder visualizar y descargar el script de generación de gráficos del tiempo de demora del tránsito vehicular, puede ingresar al siguiente enlace: https://github.com/SENAMHI-SEA/Trafico_vehicular/blob/main/Graphic_Time_traffic.py