UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

WEB SCRAPING, VISUALIZACIÓN Y ANÁLISIS BASES DE DATOS DE LA OPERACIÓN DEL SISTEMA ELÉCTRICO CHILENO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL ELÉCTRICA

LAURA BELÉN FIGUEROA GALLARDO

PROFESOR GUÍA:
LUIS VARGAS DÍAZ

MIEMBROS DE LA COMISIÓN:
GONZALO PAREDES MARTÍNEZ
PABLO GONZÁLEZ INOSTROZA

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA
POR: LAURA BELÉN FIGUEROA GALLARDO
FECHA: 2021
PROF. GUÍA: LUIS VARGAS DÍAZ

WEB SCRAPING, VISUALIZACIÓN Y ANÁLISIS BASES DE DATOS DE LA OPERACIÓN DEL SISTEMA ELÉCTRICO CHILENO

En el marco de la Operación del Sistema Eléctrico Chileno, existe una vasta cantidad de información disponible y, a la vez, gran cantidad de agentes energéticos e investigativos, que la utilizan para realizar estudios de proyecciones o balances. En este contexto no existe en Chile a la fecha una herramienta o plataforma que agrupe todos los datos disponibles, los procese y visualice de distintas maneras en un mismo sitio.

Considerando lo anterior, el objetivo del presente trabajo consiste en la creación de dos aplicaciones que aprovechen algunos de los recursos informativos que se encuentran publicados y de acceso abierto al público por parte de la Comisión Nacional de Energía (CNE) y el Coordinador Eléctrico Nacional (CEN), de tal manera que sean analizados y visualizados de manera atractiva en una única herramienta.

El trabajo completo está programado en lenguaje python y su principal ventaja es que estarán disponibles todos los códigos para cualquier persona que los necesite, teniendo la posibilidad de modificarlo y extraer aún más bases de datos para realizar análisis y visualización de ellas.

La aplicación basada en los recursos de la CNE visualiza la red de transmisión como una red de nodos y contiene los valores de Costos Marginales y Demanda proyectados. Mientras que, la aplicación que se realiza a partir de las bases de datos del CEN, que por su parte requiere de la realización de Web Scraping para la obtención de datos, muestra gráficos con los resultados del análisis de las bases de datos de los archivos RIO, Costos Marginales, archivos de Medida y Generación Real.

Durante el proceso de elaboración del trabajo se buscaron las librerías de trabajo más útiles, y eficientes para la creación de algoritmos, además de intentar evitar el uso de iteradores que ralentizaran el trabajo. El resultado final requiere de manera obligatoria que todas las redes y gráficos con información estén previamente emitidos.

A mis padres que con esfuerzo y mucho amor me criaron.

Agradecimientos

En primer lugar agradecer enormemente a mis padres, que me han apoyado y amado desde que nací, y que me han demostrado todo su amor y cariño en mis momentos más difíciles. A mi mamita por todos sus abrazos y a mi papi por mostrarme esta carrera tan linda. De no ser por él, de seguro hubiera estudiado otra cosa.

En segundo lugar, los despachadores de Enel Generación y Enel Green Power, a Don Julio y Don Armando, que lograron que me reencantara de esta carrera con todo el conocimiento que me transmitieron.

En tercer lugar a Martín, mi amor y compañero desde hace más de tres años, quien me ha acompañado, ayudado, enseñado y querido durante todo este proceso.

En cuarto lugar a todos mis amigos y amigas que conocí desde mi entrada a la u, a mi grupo de amigas: Angi, Pao, Lore, Ale, Lissy, Caro, Gigi, Belén, Tamara, Coté y Chio por haberme apañado a todas, desde largas jornadas de estudio, hasta carretes de los que ni nos acordamos. A todos mis demás amiguis que me hice en eléctrica y plan común: Tom, Dani Montaner, Pedro, Mauri, Fesita, Yerko, Oscarin, Bruno, Carita, Lucho Escares, Diego Gallardo, Andrés, Pollito, Albert, Chaguito, Pancho Barrera, Juanjo, Freddy, Isi, y Jeimy. Por apañar a todas, estar siempre presente para una ayudita, trasnochada, consejo o carrete. Y a todos mis amiguis que día a día me mandan buena onda: Carito y Hugo, Pauli, Héctor, Emi, Pancho, Felipe V. y Alejandro V.

Y, en quinto lugar a los miembros de esta comisión, al profesor por aceptarme como su alumna y darme la posibilidad de desarrollar este tema tan lindo. A Gonzalo por su paciencia, enseñanza y buena disposición. Finalmente, a Pablo por su espíritu de buen docente, quien reunión a reunión me motivó a seguir avanzando y me ayudó en gran manera con la organización de todas mis labores.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Índice de Códigos

Capítulo 1

Introducción

1.1. Motivación

El Coordinador Eléctrico Nacional (CEN) en conjunto con la Comisión Nacional de Energía (CNE) son los agentes encargados de la operación y regulación, respectivamente, del Mercado Eléctrico Chileno. Dichos organismos tienen a disposición pública diversa información relevante acerca del Sistema Eléctrico Chileno (SEN) como, por ejemplo, costos marginales, indicadores de desempeño de servicios auxiliares, precios por nudo, datos técnicos de centrales, cotas de embalses, demanda programada, política diaria, etc.

Actualmente no existe un visualizador o plataforma que agrupe varias bases de datos del CEN y de la CNE en Chile. En este contexto el trabajo de memoria consiste en elaborar una aplicación para cada fuente de información que realice la función anterior. La principal motivación de este trabajo radica en la utilidad que tiene para futuras modificaciones o para ser la base de herramientas útiles de visualización de información por parte de investigadores o entes del Mercado Eléctrico Chileno.

La forma en que se obtendrán las bases de datos del CEN será mediante la técnica de Web Scraping, la cual consiste en la extracción automatizada de contenido (información, documentos, imágenes, etc.) a nivel masivo desde la red, este método puede usarse en casos que se requiera extraer poca información, pero la mejor forma de aprovecharlo es para grandes cantidades de información. Si se necesita extraer una mínima cantidad de datos debe evaluarse el tiempo que toma la descarga manual versus la creación del algoritmo. Al usar esta técnica se asegura que la descarga de archivos sea eficiente, es decir, la automatización del proceso sea al menos cuatro veces más rápida que si fuese manual[1].

El trabajo consiste entonces, en la selección adecuada de bases de datos, la elección de un método eficaz para la descarga de archivos desde sitios web y la elaboración de una propuesta de visualización de la misma, tanto en el diseño de la apariencia de la herramienta como en la presentación de la información para ambas fuentes de datos.

[1]Dato calculado empíricamente en el transcurso de esta memoria.

1.2. Objetivo

El objetivo de esta memoria corresponde a desarrollar dos herramientas o aplicaciones programadas en lenguaje python y de acceso abierto (disponible para cualquier persona). La primera debe visualizar la red de transmisión en 220kV y 500kV proyectada por la CNE en el informe de Precio Nudo Corto Plazo y desplegar información relevante y actualizada referente a la operación del SEN y el Mercado Eléctrico Chileno. La segunda aplicación debe utilizar las bases extraídas mediante Web Scraping del CEN y visualizar el resultado de estas de manera amigable al usuario.

Se desprende como alcance del trabajo que las fuentes de información son únicamente la CNE y el CEN y se debe evitar hacer cruce de datos entre ellas, debido a que las bases a utilizar del CEN corresponden a archivos históricos[2] de la red de transmisión real, mientras que las de la CNE corresponden a bases con proyecciones de una red de transmisión simulada.

1.2.1. Objetivos Específicos

A partir del objetivo del trabajo, se desprenden los siguientes objetivos específicos:

- Para cada base de la CNE descargada,y para cada mes a partir de la fecha de emisión hasta 10 años después[3], emitir la red de transmisión con las lineas que estén operativas en el mes de estudio, asimismo la información mostrada debe ser la correspondiente para dicho mes.
- Descargar múltiples bases de datos del CEN y de la CNE. Los documentos del CEN, en su mayoría, deben ser descargados mediante Web Scraping.
- La información debe ser seleccionada y tratada, es decir, los datos mostrados no serán necesariamente los que vienen contenidos en los archivos descargados, si no que se calculará información relevante, como por ejemplo, promedio de datos, dato mínimo/-máximo, etc.
- Componer estrategia de visualización para la información, ya sea en gráficos, ventanas informativas, etc.

1.3. Estructura del Documento

El presente documento está conformado por los siguientes capítulos:

- **Capítulo 2: Marco Teórico.** En este capítulo se indaga acerca de las funciones del CEN y de la CNE, los tipos y metodos para realizar Web Scraping y los entornos de escritura disponibles para montar la aplicación.
- **Capítulo 3: Revisión bibliográfica.** Se busca en la literatura usos del Web Scraping con fines ingenieriles y para objetivos similares al de este trabajo.
- **Capítulo 4: Propuesta de trabajo.** Se define completamente la estructura y las bases del trabajo a realizar.

[2]De fechas anteriores hasta la fecha actual.

[3]Cada base del Precio Nudo a Corto Plazo contiene información hasta un horizonte de 10 años desde la emisión del archivo.

- **Capítulo 5: Metodología.** Se describen los procesos que realizan los códigos de lenguaje python que permiten ejecutar el Web Scraping, el Análisis de las Bases de Datos, y la Visualización del trabajo.
- **Capítulo 6: Resultados.** Se mostrará de manera visual el resultado final de la plataforma y los gráficos que contienen la información de las Bases de Datos.
- **Capítulo 7: Conclusiones.** Discusión acerca de los resultados, dificultades del trabajo y propuestas para el trabajo a futuro.

Capítulo 2

Marco Teórico

2.1. Coordinador Eléctrico Nacional

También llamado CEN (por sus iniciales), es una corporación autónoma de derecho público y sin fines de lucro que no forma parte de la Administración del Estado. Su organización, composición, atribuciones y funciones se rigen por lo establecido en la Ley N^o 20.936, la cual fue publicada en 2016 y creada exclusivamente ante la creación de esta entidad.

Hasta antes del 2017 el Sistema de Transmisión Chileno era coordinado por los CDEC (Centro de Despacho Económico de Carga) SING y SIC, quienes realizaban las tareas que hoy realiza el CEN. La razón de tener dos entidades radica en que el sistema no estaba conectado completamente desde Arica a Canutillar y tan solo luego de la interconexión SING-SIC ocurrida el 21 de noviembre de 2017 se logra un único Sistema de Transmisión (exceptuando las regiones más extremas del sur y la Isla de Pascua).

La principal función del CEN corresponde a la coordinación diaria de unidades generadoras (y programación de las mismas) y de las instalaciones que conforman el SEN y que operan de manera interconectada. Además de lo anterior, el CEN cumple con los siguientes objetivos:

1. Preservar la seguridad del servicio en el sistema eléctrico

2. Garantizar la operación más económica para el conjunto de las instalaciones del SEN.

3. Asegurar el acceso abierto a todos los sistemas de transmisión, en conformidad de la ley.

Además de la función anterior, el coordinador realiza más de 25 funciones [1], todas enmarcadas en el contexto del mercado eléctrico y principalmente en las áreas de Generación, Transmisión y Distribución. Finalmente, el compromiso que tiene el Coordinador tanto con el consumidor residencial como el industrial hace referencia a coordinar de manera segura y económica el Sistema Eléctrico Nacional, con el motivo principal de hacer posible que la electricidad esté disponible en los hogares, centros productivos, servicios públicos y en los diversos lugares del territorio nacional.

2.1.1. Informes emitidos por el CEN

Actualmente el coordinador[1] emite informes y reportes de distintos tópicos. Al navegar en la página del coordinador esta información se encuentra separada en cuatro categorías: Mercado, Planificación y Desarrollo, Operación y Reportes y estadísticas; algunos de los informes son:

- Antecedentes del cálculo para transferencias económicas: archivo emitido de manera mensual, el cual contiene informes relevantes tanto en balance económico y balance energético, como por ejemplo, contratos con empresas generadoras, precio nudo por barra, retiros de grandes clientes (mineras, edificios, empresas industriales) en sus respectivas barras de distribución, retiros e inyecciones por empresas generadoras, entre otros.
- Generación Real: archivo emitido de manera mensual que incluye el detalle de generación de cada central. Los datos se visualizan a lo largo del mes y de manera horaria, con lo que si una máquina o central no operó en un tramo horario se puede saber con certeza cual fue.
- Generación Sistémica Real: archivo emitido de manera diaria que incluye la potencia neta generada, el aporte de las generadoras ERNC y las restantes. Los datos están segregados de manera horaria.
- Demanda Real: archivo emitido de manera diaria que contiene la demanda horaria a nivel nacional.
- Registro de Instrucciones de Operación (archivos RIO): archivo emitido de manera diaria, incluye los movimientos más importantes durante el día relativos a la generación y transmisión eléctrica, como por ejemplo, agotamiento de un embalse, control de transferencia una línea, realización de control secundario o terciario de una unidad generadora, máquina operando al mínimo técnico, etc. Incluye también la configuración (de unidad generadora correspondiente) que está marginando para algunas barras en 220kV del sistema.
- Potencia transitada por el sistema de transmisión: archivo emitido de manera diaria que contiene de forma horaria las inyecciones y/o retiros de potencias que percibe una barra para una línea.
- Costo Marginal Real: archivo emitido de manera diaria que contiene el costo marginal horario para la mayoría de las barras en 220kV.

2.2. Comisión Nacional de Energía

La Comisión Nacional de Energía (CNE) es un organismo público y descentralizado, que se encarga de analizar precios, tarifas y normas técnicas que rigen el funcionamiento de las empresas de generación, transporte y distribución de energía, de tal modo que dicho servicio sea seguro, de calidad y lo más económico posible.

Desde el año 1978 (creación de la comisión) hasta el 2010 se consideraba esta entidad como un Ministerio, cuyo directorio estaba compuesto por un representante del Presidente

[1] www.coordinador.cl

de la República, el ministro de minería, el ministro de economía, entre otros. Actualmente y desde 2010 la Comisión pasó a ser un Organismo Público que depende del Ministerio de Energía.

La CNE realiza cuatro funciones principales:

1. Analizar de manera técnica la estructura y las tarifas de bienes y servicios energéticos.
2. Fijar las normas técnicas y de calidad indispensables para la operación y buen funcionamiento de las instalaciones energéticas.
3. Monitorear y proyectar el funcionamiento actual y esperado del sector energético, y proponer al Ministerio de Energía las normas legales y reglamentarias que se requieran, en las materias de su competencia.
4. Asesorar al Gobierno, en materias vinculadas al sector energético para su mejor desarrollo.

Finalmente, la CNE debe generar condiciones para un desarrollo seguro, sostenible, diversificado y de precios eficientes de los mercados energéticos chilenos a través de la generación de propuestas al Ministerio de Energía. Lo anterior permite cumplir los objetivos de la política pública, el monitoreo, análisis, tarificación y promulgación de normativas técnicas, económicas y de seguridad, además de asesorar a las autoridades en las materias del sector energético. La CNE calcula el precio de la energía para cada subestación existente de manera semestral en la Base del Precio Nudo Corto Plazo, la cual es utilizada en esta memoria.

2.3. Web Scraping

La técnica de Web Scraping consiste en la extracción de datos, información, imágenes, documentos (entre otros) directamente desde un navegador web. Su realización puede ejecutarla un usuario de manera manual o un bot, aunque lo más frecuente es que sea de manera automatizada. Tiene variadas aplicaciones y es posible implementarlo en distintas plataformas y con distintos procesos, es decir, no hay un método único de realizar Web Scraping.

Las extracciones de información más utilizadas son las siguientes:

1. Web Scraping estático de una sola página: todos los datos requeridos están disponibles en una única página.
2. Web Scraping estático de varias páginas del mismo dominio:
   (a) Crawling Vertical: consiste en la extracción de información, datos o imágenes que, desde la página inicial, redirigen hacia otra.
   (b) Crawling Horizontal: consiste en la extracción de información en la cual se hace necesario cambiar de página para poder recabar toda la información necesaria.
3. Web Scraping en páginas dinámicas: consiste en la extracción de información, datos, imágenes y/o documentos en páginas en las cuales se debe automatizar las acciones del navegador, por ejemplo, haciendo click sobre un botón para cargar toda la información disponible. Esto es diferente a cambiar de página debido a que al cambiar de página lo más probable es que solamente cambie el número de página de la URL, mientras que en este caso la URL (identificador de la página web) no se ve alterada.

4. Web Scraping de API: para realizar este tipo de extracción es necesario conocer la API (interfaz de programacion) que define la aplicación o página web y, en caso de no conocerla, se debe indagar en la estructura de esta. Este tipo de extracción permite obtener información y además retroalimentación de lo ocurrido con el requerimiento del bot o el usuario. En este caso se utiliza el lenguaje JSON (más adelante se explicará este lenguaje).

La implementación de esta técnica se divide en cuatro partes:

1. Buscar los enlaces deseados o la URL semilla.
2. El usuario o el bot debe realizar el requerimiento.
3. Se obtiene una respuesta en código html.
4. Se obtiene una respuesta en html y se extraen los datos del html. En caso de continuar el raspado se debe volver al segundo punto y repetir.

Podría agregarse un quinto punto si se automatiza la recurrencia de descarga para que esta acción se mantenga en el tiempo.

De lo anterior se podría concluir que es sencilla la implementación de Web Scraping. Sin embargo al momento de programar el proceso de extracción de datos deben considerarse una serie de imprevistos que existen al momento de navegar en la web, como por ejemplo, el uso de cookies, los anuncios en pantalla, permitir descargar elementos de un sitio, ventanas emergentes, entre otros. Cada una de estas acciones aumentan la complejidad en la técnica y requieren anteponerse a ellas para poder automatizar de manera eficaz el raspado de la web.

Cada vez que se realiza Web Scraping deben considerarse las implicancias legales y morales de su uso, con el fin de utilizar de manera responsable y leal la información obtenida del navegador web. Actualmente en Chile no hay leyes que penalicen esta técnica para los sitios web públicos, pero si es penalizado en otros países como Estado Unidos para los casos de: infracción de derechos de autor, infracción de la Ley de abuso y fraude informático e infracciones vinculadas al traspaso de bienes (Trespass to chattels)[2]. De la misma manera, debe tenerse precaución de no hacer requerimientos de manera desmesurada en una página, para así no colapsarla, ya que esta es una práctica habitual entre hackers. En cuanto a los temas morales, lo correcto sería que los datos extraídos fueran de uso personal o para fines no comerciales, de manera que al utilizar la información obtenida esta no afecte o realice algún daño sobre el sitio web que inicialmente exponía dicha información.

2.4. Lenguaje y escritura de una página web

Actualmente los lenguajes más utilizados para escribir una página web son JSON (JavaScript Object Notation) y XML (Extensible Markup Language), dentro del último se encuentra HTML (HyperText Markup Languaje) que corresponde a un tipo de XML.

Otra opción de escritura corresponde a PHP (Hypertext Preprocessor), el cual es un lenguaje de programación que se utiliza para la generación de páginas web de forma dinámica, el cual se adapta al código HTML. Puede considerarse como una opción para simplificar el lenguaje HTML, debido a que pueden introducirse pequeños fragmentos de PHP en HTML

para realizar diversas funciones. Estas incrustaciones pueden realizarse en cualquier parte del código y en más de una ocasión, alternando entre HTML y PHP la escritura de la misma.

2.4.1. Lenguaje HTML

Este lenguaje usado recurrentemente para desarrollar las páginas webs se basa en una estructura de árbol, donde los elementos están ordenados de manera estructurada y jerárquica. Para entender un poco el árbol, el tronco principal es designado como *body* y cada rama corresponde a un hijo de este body llamado *etiqueta html* o *tag html*. Cada etiqueta puede o no tener contenidas más de un tag. La manera en que se puede diferenciar que etiqueta es padre o hijo de otra es mediante la indentación (espacios entre el margen izquierdo y el comienzo de las letras).

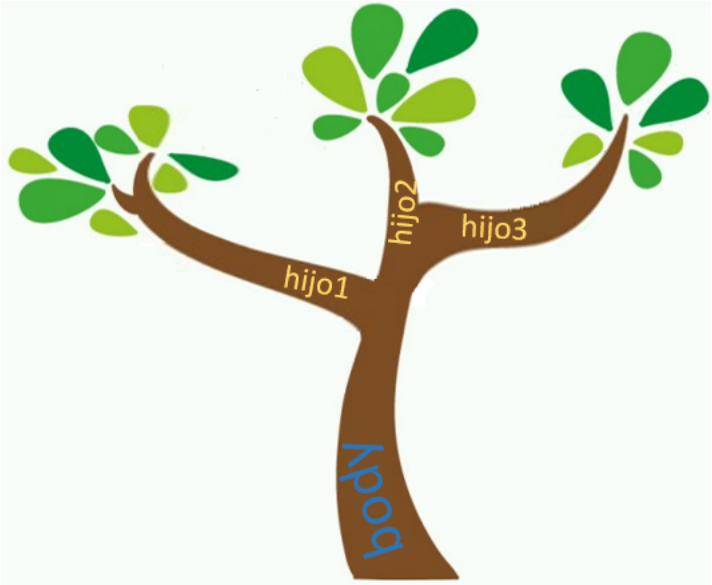


Figura 2.1: Estructura de árbol en un html. Elaboración Propia.

Para poder escribir en este lenguaje se utilizan los elementos " <> " para encerrar el body o un tag, cada vez que se define un elemento se realiza una apertura y un cierre, tal como se muestra en el código 2.1.

```
<body>
    contenido
</body>
```

Código 2.1: Ejemplo estructura apertura y cierre.

Existen distintos tipos de tags, y la mayoría tiene asociada una funcionalidad específica. Los más usados se muestran en la tabla 2.1.

Cada tag puede contener en su interior una serie de elementos que lo caracterizan aún más, los cuales son llamados atributos, los más comunes corresponden a class y id, siendo este último un valor que no se repetirá para ninguna otra etiqueta. Adicionalmente cada tag puede contener texto en su interior. Con todo lo anterior es posible armar un árbol más robusto, tal como se muestra fragmento de inicio de sesión contenido en el código 2.2.

Tag	Función
p	contiene largos textos
a	contiene links
button	botones
form	formularios para llenar
input	casillas para rellenar
div	contenedores de imagenes, texto, etc
h1, h2, h3, h4 y h5	titulos y subtitulos
img	imagenes
table	tablas

Tabla 2.1: Tags y funcionalidades.

```
        <body>
            <div class="contenedor" id="1">
                <input class="nombre de usuario" id="1.1">↩
                   Ingrese
                su nombre de usuario</input>
                <input class="clave" id="1.2">Ingrese su ↩
                   clave
                </input>
                <button id="1.3">Ok</button>
            </div>
        </body>
```

Código 2.2: Inicio de sesión.

2.4.2. Lenguaje JSON

Este tipo de lenguaje es un formato de texto sencillo para el intercambio de datos. Corresponde a un subconjunto de la notación de objetos en Javascript. Es un formato de texto completamente independiente de lenguaje, pero utiliza convenciones que son ampliamente conocidos por otros lenguajes de programación, entre ellos: C, $C++$, $C\#$, Java, JavaScript, Perl, Python. JSON está construido por dos estructuras principales:

1. Una colección de pares de nombre/valor, conocido también como objeto, registro, estructura, diccionario, tabla hash, lista de claves o arreglo asociativo.
2. Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Los objetos se presentan como conjuntos desordenados de pares nombre/valor y con llaves de apertura y cierre, en donde cada nombre es seguido por dos puntos (:) y los pares nombre/valor se separan por comas. Un arreglo es una colección de valores que lleva apertura y cierre de corchetes y se separan por comas. Un valor puede ser una cadena de caracteres con comillas dobles, un número, true, false, null,un objeto o un arreglo. Estas estructuras pueden anidarse. En el código 2.3 se muestra la gerencia de Ventas de una empresa en lenguaje JSON.

```
    {
```

```
    "departamento":"8",
      "nombredepto":"Ventas",
    "empleados":[
      {
        "nombre":"Pedro",
        "apellido":"Fernandez",
      }.
      {
        "nombre":"Martin",
        "apellido":"Rubio"
      }
    ]
  }
```

Código 2.3: Gerencia de Ventas.

Comúnmente es utilizado para transmitir datos en aplicaciones web, pero se requiere que la API de la página web o aplicación de la cual se desea hacer el raspado sea pública. En caso contrario deben realizarse indagaciones acerca de su estructura para poder realizar la extracción de datos.

2.5. Opciones de extracción

Hoy en día y debido a que el Web Scraping es una herramienta bastante utilizada, existen distintas formas de realizar raspado web. En particular se conocen tres métodos:

1. Usar un servicio de web scraping: hoy en día existen páginas web que se dedican a realizar raspados de web, algunas son gratuitas y otras pagadas. Estos servicios web permiten extraer datos estructurados de la página que se requiera. Algunos de estos son: 80legs.com, Datahut.co, Import.io, iWebscraping.com, Datahen.com, Scrapinghub.com, entre otros.
2. Usar un plugin en el navegador para web scraping local: para utilizar esta alternativa se necesitan dos pasos. El primero corresponde a la instalación de una extensión para screen scraping en el computador o usarse un programa para realizar scraping y segundo, es necesario definir las reglas para el rastreador web. Este tipo de solución es más adecuada para operaciones puntuales ya que requieren intervención manual y no están orientadas a ser ejecutadas automáticamente. Existen opciones gratuitas y otras pagadas, algunas de estas extensiones son: FMiner, OutWit, WebScraper.io, Screen-Scraper, DataMiner, TableTools2, artoo.js, entre otros.
3. Programar un algoritmo de web scraping: utilizando un lenguaje de programación y sus clases nativas o con alguna librería o framework específico es posible crear un algoritmo de extracción para web scraping. Al estar hecho a medida, es la solución que aporta mayor flexibilidad e integración. Algunas de las herramientas útiles son: Guzzle, CasperJS o pjscrape (PhantomJS, SlimerJS), Scrapy, Noodle, Beautiful Soup, Selenium WebDriver, Mechanize, Simple HTML DOM , Web-Harvest, Zombie, SpookyJS, entre

otros.

Para los fines de esta memoria se considerará solamente la alternativa de un algoritmo de elaboración propia. A continuación, se revisarán las plataformas en las que es posible la creación de este algoritmo.

2.5.1. Java

Java es un lenguaje de programación versátil orientado a objetos, es decir, los objetos se encargan de encapsular información, clases y funciones, las cuales se pueden manipular o se adicionan a distintos programas. Java es uno de los lenguajes más utilizados en proyectos de gran tamaño, que a simple vista pueden parecer sumamente complejos. Permite a los desarrolladores escribir sus aplicaciones una única vez y estas podrán ser ejecutadas en cualquier equipo o dispositivo sin importar el sistema operativo con el que funcionan.

Una de las ventajas de su uso es su compilación, pues es tan buena que se llega a asimilar al lenguaje ensamblador, es decir, desde la base puede ser interpretado. Esto ayuda a la ejecución de aplicaciones compiladas en Java, pues se puede ejecutar básicamente en cualquier lugar sin mayor problema, además de que es un lenguaje a código abierto.

Los usos más habituales del lenguaje de programación Java son realmente variados. Java es muy utilizado en todo tipo de aplicaciones de productividad como procesadores de texto u hojas de cálculo. En líneas generales, Java se encuentra en multitud de navegadores y programas pues cualquier aplicación desarrollada con este lenguaje puede ser ejecutada en un navegador.

Algunas librerías útiles para realizar Web Scraping en Java son:

- Noodle: utiliza un servidor y un módulo para consultar y extraer datos de documentos web compatible con JSON, XML y HTML.
- SpookyJS: permite abrir páginas web, procesarlas y evaluarlas.
- Zombie: entorno para probar código JavaScript del lado del cliente en un entorno simulado.
- Web-Harvest: herramienta de extracción de datos web de código abierto escrita en Java que permite recopilar las páginas web deseadas y extraer datos útiles de ellas, es compatible con HTML y XML.
- Pjscrape: permite raspar páginas en un contexto habilitado para Javascript completamente renderizado desde la línea de comandos, sin necesidad de navegador.
- Jsoup: proporciona una API para extraer la información.

2.5.2. Python

Python es un lenguaje de programación independiente de plataforma, orientado a objetos y preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo. Contiene gran cantidad de librerías que incorporan funcionalidad en el lenguaje, facilitando la realización de tareas.

Hay versiones disponibles de Python en muchos sistemas informáticos distintos. Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él.

Python tiene una sintaxis muy visual, gracias a una notación con indentación[2]. En muchos lenguajes para separar porciones de código se utilizan elementos como las llaves o las palabras clave begin y end. Esto ayuda a que todos los programadores adopten la misma notación y que los todos los tengan un aspecto muy similar.

Las librerías más útiles para realizar Web Scraping desde Python son:

- Selenium: esta herramienta tiene múltiples funciones en cuanto a realización de funciones en una página web, permitiendo abrir una página desde el compilador, acceder a datos, imágenes o archivos específicos, llenar formularios como loggin y password, hacer click, cambiar de pestaña, etc. Es compatible con XML y lenguaje XPATH. Muy útil para Web Scraping dinámico.
- Requests: esta librería permite acceder de manera sencilla a cualquier URL, independiente si necesita o no de autenticación. Es muy utilizada para extraer contenido desde la web y principalmente para la descarga de archivos que en el lenguaje madre de la página web contengan un enlace específico de descarga.
- Scrapy: esta librería es por lejos la librería más común para realizar web Scraping. Permite hacer requerimientos al servidor y analizar la información, haciendo posible la recolección de datos. Sus principales usos corresponde al crawling vertical y horizontal, teniendo incorporada la funcionalidad de recorrer múltiples páginas que estén relacionadas por paginación.
- Beautiful Soup: esta librería se utiliza para recorrer y analizar el código completo HTML de una página, pudiendo extraer del mismo información, valores o imágenes deseadas.
- Mechanize: librería utilizada principalmente para rellenar y enviar de manera automática formularios, activar el uso de cookies, y acciones que podrían interrumpir la automatización de un proceso de extracción de datos. Permite navegar por la web ya que tiene incorporadas funciones que brindan todos los enlaces en una página y todos los formularios.

[2]Margen al lado izquierdo de un código que denota el orden de ejecución de una linea.

Capítulo 3

Revisión Bibliográfica

Al indagar en la literatura, puede notarse que muchos estudios, investigaciones de mercado, entre otros, requieren de gran cantidad de información que es actualizada o informada de manera diaria o inclusive horaria. Para suplir esta búsqueda tediosa de descarga, se utiliza la técnica de Web Scraping, la cual no es útil para recabar poca cantidad de información (por ejemplo, un único dato puntual ocurrido en una específica fecha), debido al tiempo o costo que debe asignársele a la creación del algoritmo; más bien es utilizada para extraer información en masa.

Primeramente debe tenerse claridad del formato en que está contenida la información previo a la descarga, por ejemplo, si se necesita extraer datos que vienen contenidos en un archivo CSV que es emitido de manera diaria, es lógico preguntar ¿Cómo obtengo el archivo para una fecha específica?, ¿Se debe introducir la fecha y automáticamente se descarga? o ¿Es necesario hacer click sobre un botón?. La metodología para realizar una descarga varía según el tipo de información y depende también de cada sitio web.

Algunos estudios recurren a la extracción de información mediante la realización de un algoritmo que identifique patrones en la estructura del código fuente (en HTML) del sitio web, por ejemplo, estudios acerca del precio de productos similares en páginas de comercio como Ebay o Mercadolibre. Dichas páginas tienen su información representada en forma de registros webs o, mejor dicho, ordenan sus productos a partir de una base de datos con plantillas[1] o diseños fijos mediante script de servidor[3]. En el caso de los sitios anteriormente mencionados, existe un diseño estándar fijo para la imagen, el nombre y el precio del producto. De esta forma es posible encontrar patrones de registros en páginas web estructuradas y así extraer la información deseada.

En algunas ocasiones las plantillas están disponibles para los usuarios, aunque también es posible que no lo estén y en dicho caso es posible extraerlas o deducirlas mediante Web Scraping[4]. La utilidad de la obtención de estas plantillas radica en que se obtiene la información de una forma más limpia. La información irrelevante, como por ejemplo, paneles de navegación, anuncios, políticas de privacidad, entre otros, son descartados y así es posible

[1]Formato de presentación de un elemento, en el caso de un producto comercial, la plantilla corresponde a la estructura con sus caracteristicas: nombre, precio, tamaño, color, etc.

optimizar el espacio de almacenamiento.

Si bien hasta el momento solo se ha hecho incapié en que el Web Scraping se utiliza para la extracción de información y/o documentos, archivos, etc., se puede obtener información acerca de la navegación que un usuario realiza en un sitio web. Por ejemplo, si en un gráfico dinámico el movimiento del cursor realiza una acción (como un cambio de color, aparición de un texto, etc.), es posible registrar dicha acción. En ese caso, es necesario realizar un escaneo o registro constante del código fuente y guardar las modificaciones entre el escaneo anterior y el actual, de esta forma es posible obtener patrones secuenciales y también la frecuencia en que ocurren, los cuales son útiles en el reconocimiento de ataques de red coordinados y en la identificación de los modos de falla en cascada de un servidor web[5].

Otro uso de las librerías utilizadas para Web Scraping (Selenium, BeautifulSoup, Scrapy) consiste en la generación de datos de comportamiento del usuario en la web. A diferencia del caso anterior no se requiere de mapeo frecuente o de identificación de patrones, ya que el fin es facilitar la búsqueda de información útil para un usuario, lo que se traduce en una recomendación del sitio de un producto/información similar a la deseada. Para lo anterior debe registrarse el contenido de la barra de búsqueda en el sitio web para cada vez que el usuario la utilice y haga click en buscar .

A pesar de lo anteriormente escrito, deben tenerse en cuenta las limitantes para poder realizar Web Scraping. No basta solamente con la creación de un algoritmo que realice el requerimiento de descarga de archivos y/o lineas de texto, pues el propietario de una página puede prevenir que alguien más realice el raspado de manera automática, lo cual podría ser negativo si la cantidad de requerimientos es tal que colapse el sitio. Con el fin de evitar la extracción existen cinco [6] acciones que detendrán el raspado:

1. Bloquear solicitudes de requerimientos que se ejecutan con mucha rapidez y desde la misma computadora. Esto es sencillo ya que pueden identificar la IP del requerimiento y luego bloquear la acción.

2. Uso de Captcha (verificación mediante imágenes o cadenas de texto), para el caso de la descarga automática esto sólo puede remediarse (para el usuario del algoritmo) por medio de la creación de un reconocedor de imágenes muy avanzado.

3. Cambiar el código fuente (HTML) de manera regular. Ante esta acción cabe aclarar lo siguiente: cambiar el código fuente no es algo que se realice de manera diaria, ya que podría involucrar cambios visuales. Es más, cuando un sitio web cambia su aspecto visual el código fuente es modificado y el algoritmo de descarga para realizar Web Scraping debe modificarse.

4. Contener la información en otros formatos, como por ejemplo, PDF, videos, imagen, gift, u otro. El algoritmo de descarga podrá descargar el nuevo contenedor de información, pero para analizarlo deberá requerirse un humano o un reconocedor visual.

5. Crear páginas Honeypot: un honeypot sencillo simula una aplicación de servidor que facilita uno o varios servicios en red, es una página a la cual un usuario humano jamás visitaría, pero un bot sí al momento de extraer información. Esta página trampa permite recopilar datos del bot y bloquear o prevenir acciones dañinas.

3.1. Ejemplos Web Scraping

Para la realización de un algoritmo de Web Scraping (particularmente en lenguaje Python) es necesaria la elección de una libreía que se adapte a las necesidades el usuario. En el marco de la Ingeniería Eléctrica existe la librería PYIso (disponible en Python), la cual permite recopilar mayoritariamente datos ISO (información energética emitida por Operadores de Servicios Independientes y Organizaciones Regionales de Transmisión en Estados Unidos). Dichos datos informativos son emitidos en tiempo real cada 15 minutos. Gracias a esta librería se ven beneficiadas múltiples autoridades en materias energéticas, ya que las Organizaciones ISO cubren al rededor de 2/3 de los consumidores de electricidad de EE.UU. Esta librería permite que cualquier persona acceda a datos de carga y generación históricos y en tiempo real[7].

Continuando en el contexto eléctrico, también existe el proyecto Public Utility Data Liberation (Proyecto de Liberación de Datos de Servicio Públicos) en Estado Unidos, el cual toma información que está disponible de manera pública en la red y la trabaja de manera que lo recolectado se limpie, estandarice y se obtenga un cruce de datos de distintas fuentes. El enfoque actual a nivel de información ha sido para generación, uso de combustible, costos operativos y el historial de operaciones, de tal manera que los usuarios puedan explorar costos operativos de las plantas de energía (de manera individual), y puedan ver la influencia de los costos de combustible en la viabilidad de una tecnología de generación[8]. El estudio presentado en [8] realiza Web Scraping en lenguaje Python y el código está disponible abiertamente.

Actualmente en California y con el fin de realizar pronósticos de demanda de electricidad,se utiliza la libreria Selenium [9] (disponible en Python), la cual permite extraer contenido de manera automatica para cualquier fecha anterior en que haya información, o bien, para cualquier rango de fecha. Esta librería requiere de la revisión del código fuente del sitio para proceder con los requerimientos antes mencionados (disponible en lenguaje HTML).

También es utilizada la técnica de Web Scraping en situaciones que se necesiten extraer datos explícitos de la web, como por ejemplo, cuando el dato no está contenido en un archivo, si no que está disponible en una linea (o varias) de texto. En tal caso se requiere de dos técnicas: la primera corresponde a un rastreador, la cual se encarga de extraer contenido de la web a partir de una regla de coincidencia, como por ejemplo, una barra de búsqueda; y la segunda requiere de Web Scraping, para este tipo de extracción se recomienda la librería Scrapy. Un ejemplo de lo anteriormente descrito ocurre en Estados Unidos, donde se utiliza lo anterior para la recuperación de información petrolera con fines de ayuda a los investigadores de exploración y el desarrollo petrolero. Dicho sistema fue implementado en lenguaje Python y contiene un "calendario" para la descarga de información. Las conclusiones generales de aquel trabajo muestran que el sistema de recuperación personalizado es eficiente y ágil, mejora la eficiencia, precisión y nivel de automatización del trabajo [10].

Los usos de esta técnica son bastante variados e incursionan en todos los campos investigativos. En la literatura se encuentran ejemplos de extracciones de información en las redes sociales[11], en páginas de comercio[12], portales del clima[13], y otros. Se concluye que dada la cantidad de datos extraídos, el Web Scraping va de la mano con el Análisis de Bases de Datos y la Inteligencia Artificial.

Capítulo 4

Propuesta de trabajo

4.1. Elección del entorno de trabajo

Considerando todo lo investigado anteriormente, y el hecho de que tanto los lenguajes Python y Java ofrecen múltiples herramientas para desarrollar Web Scraping, se ha decidido trabajar en Python. Las razones son las siguientes:

- Python tiene bastantes librerías de código abierto que están disponibles para cualquier persona que quiera usarlas. Estos módulos no solamente son útiles para Web Scraping, si no también para el desarrollo completo del trabajo, principalmente visualización y análisis de bases de datos.
- El programa Qt Designer realiza creación de aplicaciones a medida, y las transcribe en lenguaje Python. Este programa se utiliza para la creación de ambas plataformas.
- Por comodidad y experiencia, ya que el lenguaje Python es familiar.

4.2. Descripción del trabajo

El trabajo realizado consistió en elaborar dos aplicaciones off-line, la primera denominada **Aplicación CNE** visualiza las simulaciones de la red de transmisión chilena obtenidas desde las bases del Precio Nudo a Corto Plazo de la CNE, únicamente se consideran las tensiones de 220kV y 500kV. Esta red muestra información relevante de la operación del SEN. La segunda plataforma llamada **Aplicación CEN** expone únicamente gráficos o cuadros de texto con información extraída desde de las bases de datos históricas del CEN. Toda la información está asociada a una barra, central o mes.

Ambas aplicaciones son capaces de mostrar, para cada mes y por los años que correspondan, la información a nivel mensual de los parámetros (seleccionados en este proyecto) de la operación del SEN, tales como, costos marginales, generación, información operacional, retiros e inyecciones, entre otros.

A partir de todo lo mencionado anteriormente, las secciones de descripción del trabajo, metodología y resultados, se separarán en dos subsecciones: Bases Precio Nudo Corto Plazo

(Aplicación CNE) y Bases Históricas (Aplicación CEN).

4.2.1. Base Precio Nudo Corto Plazo

La Base Precio Nudo Corto Plazo de la CNE contiene los resultados de simulaciones del SEN desde la fecha de emisión de la base hasta un horizonte de 10 años y con valores mensuales. Dichas proyecciones abarcan información de demanda y costos marginales para cada barra, y todas las lineas de transmisión que se utilizaron en dicha simulación con la fecha de creación y termino de operación de la línea. A partir de lo anterior, se elabora una aplicación off line, que abarque dos conceptos importantes:

1. Visualización de la red.
2. Contenido informativo referente a las barras del sistema.

Visualización

Para armar la red de transmisión es necesario descargar la Base Precio Nudo Corto Plazo disponible en la CNE, específicamente el archivo (zip) *Bases de Cálculo y Archivos de Salida Informe Técnico - Dat*, este archivo contiene todas las lineas del SEN de la simulación que realiza de manera semestral la CNE. El documento incluye los nombres de cada barra en los extremos de una linea y la temporalidad[1] de las líneas.

A partir del archivo anterior se extraen todas las barras, las líneas y la temporalidad de ellas. Una vez extraídas se emiten mapas mensuales con las lineas que están vigentes de acuerdo al mes correspondiente y con las barras claramente denominadas por su nombre. Cabe destacar que la base de la CNE considera *barras auxiliares*[2] que se incluyen en la red y que también tienen información asociada.

Con respecto al aspecto de la red de transmisión, las barras están posicionadas de tal manera que asemejen el sistema de transmisión real y con un código de colores para cada nivel de tensión tanto para líneas como para barras.

Contenido informativo

Sobre cada barra se despliega una ventana informativa con la información extraída de las salidas de las bases de la CNE, la cual contiene el resultado del despacho hidro-térmico para un horizonte de 10 años a partir de la fecha de emisión. La información tiene una extensión de no más de 5 lineas, con su respectiva variante para cada barra, mes y base en cuestión. El contenido informativo corresponde únicamente al Costo Marginal y la Demanda calculada para cada barra por mes.

4.2.2. Bases Históricas

Las Bases históricas del CEN se suben de manera diaria o mensual a la página del coordinador y también están disponibles para fechas anteriores. Para estas se elaboró una aplicación

[1]Fecha inicial y final en que una barra está operativa en la simulación del SEN.

[2]Barras que son denominadas *Aux* que físicamente no existen y cuya finalidad está atribuida a los cálculos de flujos de potencia que elabora la CNE.

off-line que muestra de manera visualmente agradable información referente a la operación del SEN. La información se calcula para cada mes, y si está disponible la base, a partir del 2010 hasta el 2020. Esta aplicación se compone de tres procesos fundamentales:

1. Web Scraping
2. Análisis Bases de Datos
3. Visualización de la aplicación y de la información

Web Scraping

Las bases de datos que se utilizan para esta aplicación son las siguientes:

1. Archivos de Costos Marginales: el CEN tiene disponible esta información en dos formatos: por barra y mensual, o general(incluye la mayoría de las barras para cualquier nivel de tensión) y por mes.
2. Archivos Generación Real: estos documentos contienen información de todas las unidades generadoras del país y generación de cada una de ellas a nivel horario durante el mes de interés.
3. Archivos de Medida: corresponde a archivos emitidos de manera mensual que contienen información de retiro e inyecciones de potencia activa y reactiva. Estos documentos ordenan los datos por cada medidor de subestación y a nivel horario. Están disponibles desde 2018.
4. Archivos RIO: estos archivos corresponden a información a nivel técnica-operacional de la operación, valga la redundancia, del CEN. El documento muestra acontecimientos relevantes de una unidad generadora o linea, como por ejemplo, máximo técnico, realización de control de frecuencia, salida de una unidad, entre otras. Estos archivos están disponibles desde Noviembre de 2017 y son emitidos de manera diaria.

El trabajo completo realiza la automatización de la descarga de archivos sobre 3 de las 4 bases anteriores (no se incluyen los archivos de medida). Para esto se elaboran algoritmos que navegan por el sitio del CEN y descargan automáticamente todos los archivos de las bases, a partir de Enero 2010, o de la primera existencia del archivo (en caso de que sea posterior al 2010) hasta el 2020. El algoritmo no solamente es capaz de descargar los archivos, si no que asegura la continuidad de descarga, es decir, para cada nuevo día se continúan descargando los nuevos archivos emitidos (en caso de que sean diarios, o su correspondiente mensual).

A pesar de realizar Web Scraping sobre 3 bases dispuestas en el coordinador, esta tecnica puede utilizarse para muchos otros documentos que emite el coordinador, ya sea las cartas que emite de manera diaria, archivos de flujos de transferencia, archivos de demanda, entre otros. En realidad cualquier documento, información o archivo que tenga disponible un enlace de descarga, o que necesite que el usuario rellene casillas (con fechas, formato del documento, entre otros) y presione botones, es viable para descargar mediante Web Scraping.

Análisis Bases de Datos

Considerando todas las bases de datos anteriormente descritas, se realizó la extracción y tratamiento de la información, de tal forma que se llevaran a cabo las siguientes 8 propuestas:

1. El promedio del costo marginal de cada barra.
2. El máximo y mínimo costo marginal promedio para una barra en un mes.
3. Cantidad de veces que el costo marginal en una barra es 0.
4. Indicar el porcentaje de tiempo (con respecto a las horas totales) de cada tipo de tecnología que marginó en un mes para las barras disponibles en los Archivos RIO.
5. Indicar cantidad de tiempo en el mes en que ocurrió una acción o consigna (Control de Frecuencia CF, Control de Trnasferencia CTx, Mínima/Plena Carga, etc.) para cada unidad generadora operativa en dicho mes.
6. Retiros e inyecciones de potencia por barra. Se propone mostrar el dato total y además el dato por medidores de subestación.
7. Indicar el aporte en términos de generación de las unidades hidráulicas, térmicas y renovables en el mes en función de la generación total.
8. Mostrar generación total en un mes y a nivel nacional.

Visualización

La aplicación consiste en cuatro pestañas de información, por cada base de información existe una pestaña. Dependiendo de la información existen dos formas viables de presentar los datos: mediante gráficos de torta o de barra, o por líneas de texto o tablas con la información contenida.

4.3. Formato de entrega

El trabajo anteriormente descrito está compuesto por los siguientes entregables:

1. Programa funcional en el computador utilizado para la realización de este trabajo.
2. La totalidad de los scripts que conforman el trabajo publicados en Google Colab, de forma que cualquier persona pueda acceder a los códigos mediante los enlaces entregados.
3. Todos los archivos que no sean script en Python que son necesarios para el trabajo.
4. El computador y disco duro que fueron asignados para la realización del trabajo. La importancia de estos elementos está en que el disco duro contiene descargadas todas las bases de datos y, para poder ejecutar el programa, es necesario de múltiples archivos que están disponibles en el, como por ejemplo archivos de coordenadas para las barras, drivers para el uso de entorno virtual, entre otros archivos que son adicionales a los datos extraídos de las bases de datos del CEN.
5. Los ejecutables de cada aplicación.

Cabe mencionar que el hecho de entregar todos los script con códigos, no asegura que cualquier persona puede hacer uso de la aplicación en toda su plenitud. En primer lugar, si se desea abrir la aplicación desde la terminal de Python, se recomienda tener conocimiento previo acerca del uso de la terminal en un nivel básico. Lo anterior se debe a que la realización del trabajo necesita que múltiples librerías de Python ya estén cargadas, y la descarga de estas se debe hacer librería por librería. En segundo lugar y recalcando lo anterior, algunas bases

de datos se obtienen mediante descarga manual, con lo que se deben seguir las instrucciones de descarga para ver toda la información que la aplicación tendrá a disposición.

Por otro lado, quienes deseen utilizar los ejecutables para abrir la aplicación, deben de tener guardados de manera obligatoria y en el mismo directorio que este, una serie de carpetas que contienen los archivos de resultados del análisis de las bases de datos.

Capítulo 5

Metodología

El siguiente capítulo describe el procedimiento para la creación de ambas aplicaciones.

5.1. Aplicación CNE

5.1.1. Obtención y especificación de las bases de datos

Antes de comenzar con la creación del algoritmo, es necesario descargar las Bases Precio Nudo Corto Plazo de la CNE. La descarga de estas bases se realiza de manera manual debido a que solamente se emiten dos al semestre. Para efectos prácticos de esta memoria solo serán consideradas las bases emitidas desde el 2018 al 2020.

La descarga de las bases se realiza desde el sitio web de la CNE sección Tarificación (Electricidad), sección Precio Nudo Corto Plazo[1] y elegir un proceso de fijación de precios (año). Posteriormente se descargan los archivos de la Subsección *Bases de Cálculo y Archivos de Salida Informe Técnico Preliminar* y descargar las bases bajo los textos Dat y CMg. En la figura 5.1 se muestran los botones de descarga.

Figura 5.1: Botones de Descarga de Bases Precio Nudo Corto Plazo de la CNE.

Una vez descargadas todas las bases, se guardan de manera ordenada en una ubicación específica, ya que la ejecución del algoritmo utiliza direcciones exactas para ubicar los documentos. Para guardar todas las bases se utiliza una carpeta llamada *Bases Precio Nudo y CMg* la cual se encuentra ubicada en el mismo directorio que los script de algoritmo que serán descritos mas adelante y que el ejecutable. El siguiente paso corresponde a modificar el nombre los archivos, incorporando al final de ellos la siguiente cadena de texto: "_año_semestre", como por ejemplo, "Dat_2020_1" corresponde a la base para el primer semestre del año 2020.

[1] https://www.cne.cl/tarificacion/electrica/precio-nudo-corto-plazo/

Luego se descomprimen los archivos, para lo cual se utilizó un pequeño algoritmo que identifica todos lo archivos de la carpeta antes mencionada, retorna una nueva carpeta con el contenido del archivo y elimina el archivo anterior (que estaba en formato .rar o .zip). Para lo anterior se utilizaron las siguientes funciones:

1. listdir y mkdir de la librería os: la primera entrega los archivos de un directorio y la segunda crea una carpeta para almacenar el contenido del zip.
2. Zipfile y extracall de la librería zipfile: la primera lee el archivo en formato zip y la segunda extrae el contenido a un directorio especificado
3. remove de la librería os: elimina archivos especificados.

Cabe mencionar que las funciones anteriores no deben tener ningún cambio en las mayúsculas o minúsculas especificadas, de haber alguno la compilación del script no reconocerá la función modificada.

5.1.2. Análisis y Tratamiento Bases de Datos

Para los fines de esta plataforma se utilizan solamente dos tipos de archivos de la CNE, los cuales corresponden a las entradas de esta aplicación:

1. El archivo con la operación de las líneas (obtenido en Dat, véase la sección 5.1.1).
2. El archivo de los costos marginales (obtenido en CMg, véase la sección 5.1.1)

Se habla de tipo de archivo debido a que en realidad se compilan tres archivos (SING, SIC y SIX[2]) en uno, para cada uno de los anteriormente mencionados.

Considerando lo anterior, el primer paso es concatenar los tres archivos en uno, esto se realiza para los archivos de líneas y para el de costos marginales. Posteriormente y con motivo de no ralentizar el análisis de las bases de datos, se procede a hacer una función que retorne un Dataframe[3] que solamente contenga las barras de interés, es decir, se procede a realizar un filtrado de tensión, el cual tendrá como salida únicamente las barras de 220 kV y 500 kV.

Para lo anterior se crearon dos funciones. La primera función se denomina *filtrar_tension_df(archivo)*, cuya entrada es un archivo de operación de líneas en el cual cada fila del documento corresponde a un circuito. Los procesos en orden cronológico que ejecuta esta función son los siguientes:

- Lee el archivo de entrada con la función read_csv de la librería pandas.
- Crea un dataframe a partir de las columnas de interés (barras A y B de una línea, línea, fecha de inicio de operación de la línea y fecha final de la operación de la línea) del archivo y adiciona dos columnas en blanco, "BarA" y "BarB", destinadas a guardar la tensión de cada barra extrema a la linea correspondiente. Lo anterior se realiza con la función DataFrame de pandas.
- Recorre el dataframe fila por fila y guarda los nombres de las barras A y B (extremos de la línea).

[2]Correspondiente a segmentos de la Interonexión del año 2017.
[3]Arreglo ordenado de datos con filas y columnas.

- Descompone el nombre por espacios, es decir, entrega una pequeña lista con todos los elementos que estaban separados por un espacio. Lo anterior se realiza con la función split().
- Compara el último elemento del nombre de la barra A y B (correspondiente por notación de la CNE a la tensión)y , en caso de ser 220 o 500, lo guarda en el dataframe en las columnas en blanco. En el caso de ser una tensión menor, el valor guardado es cero.
- Selecciona todas las filas que en las columnas BarA y BarB sean 220 0 500, esto se realiza con la función query().
- Retorna el arreglo de datos para las barras de interés.

La segunda función se denomina cargar_base(anho_base,semestre_base), la cual tiene como entradas el año y el semestre de la emisión de la base. El objetivo de esta función corresponde a concatenar los archivos SIC, SING y SIX en uno, y realizar el filtrado de líneas. Los procesos que realiza la función son los siguientes:

- Dependiendo del año y semestre de emisión de la base, el documento de Operación de Líneas tiene un nombre diferente, con lo que la función establece condiciones temporales para identificar el archivo. Para esto se detectó de manera manual la denominación de los documentos dependiendo del semestre.
- Una vez identificado el año y semestre de emisión, se procede a concatenar los tres archivos.
- Se crea una ubicación y nombre para el nuevo archivo, y le imprime el arreglo concatenado, guardando el contenido en un archivo.
- Invocando la función de filtrado de tensión, el nuevo arreglo es reducido a las líneas con las tensiones de interés, y nuevamente se guarda este archivo (en uno distinto al primero).
- Para evitar repetir procesos, la función asegura que el archivo no haya sido concatenado con anterioridad. De no existir en el directorio, se procede a guardar y, de forma análoga, se realiza el mismo procedimiento con el archivo con la tensión filtrada.

Finalmente sigue realizar el análisis de las bases de datos CMg, para lo cual se tiene la función info_precio_nudo(), la cual recibe como entradas: el año y semestre de emisión de la base, el año, el mes y la barra de la cual se desea obtener información. Cabe destacar que una base contiene información desde su emisión hasta un horizonte de 10 años hacia el futuro. El objetivo de esta función corresponde a entregar una linea de texto que contenga el Costo Marginal y la Demanda proyectada por la CNE para una barra en un mes específico. La función es bastante sencilla: lee el archivo de la base especificada, arma un arreglo de datos con las columnas de Costos Marginales, Demanda, año, mes y barra y con la función query obtiene un arreglo de una fila con la barra de interés. Este pequeño arreglo es trabajado y se extraen de él el Costo Marginal y la Demanda, los cuales son incorporados a una línea de texto que contiene el nombre de la barra y los datos extraídos. Esta línea de texto es la salida de la función.

Con el fin de que el lector tenga una noción de la red de transmisión que elabora el programa, se incluye en la figura 5.2 la Red de Enero 2020 de la base de datos del Primer Semestre de 2020.

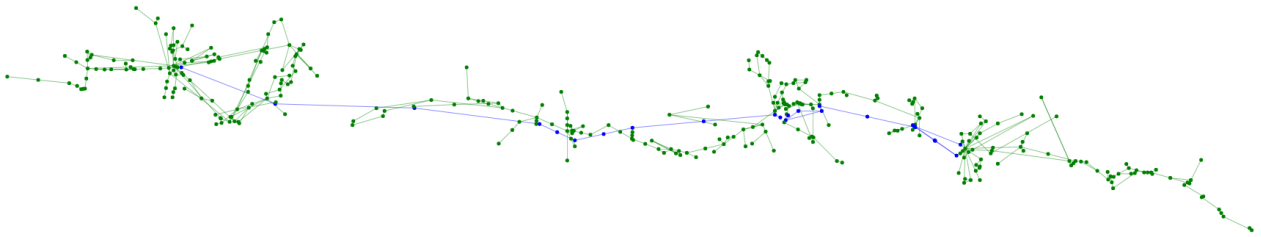


Figura 5.2: Red de Enero 2020 de la base de datos del Primer Semestre de 2020.

5.1.3. Creación de la Red de Transmisión

Antes de describir la metodología para la creación de la red de transmisión, es necesario comprender de qué forma visual será esta. Se tienen nodos que representan a cada barra, los cuales se marcan con color azul para 500 kV y color verde para 220 kV. Los nodos están unidos según el archivo de operación de líneas del Dat. Cabe destacar que por limitaciones de la librería no es posible trazar un recorrido entre dos nodos que no sea recto y tampoco es posible marcar múltiples circuitos, también el hecho de que existen bastantes barras auxiliares, las cuales no tienen posición real, a estas barras se les asignará una posición adecuada calculada a partir de sus nodos vecinos.

Considerando lo anterior, el primer paso (previo a la construcción de la red) es tener definidas las posiciones de todas las barras. Se elaboró de manera manual, y basándose en los mapas nacionales de transmisión emitidos por el CEN (años 2017 y 2019,) un archivo que contiene coordenadas referenciales para las 200 barras mas relevantes del sistema. El método de cálculo de posición consiste en tomar a sus vecinos y calcular un número aleatorio entre la posición de estos. La función tiene como salida el archivo con todas las barras del sistema para una base de datos.

El siguiente paso consiste en la creación de directorios para guardar los mapas. En la carpeta en que se encuentran los scripts se crea otra carpeta llamada *Mapas Generados*, luego una carpeta dentro de esta para cada base, denominada *Bases_precio_nudo*, las cuales en su interior contienen 10 carpetas, una para cada año a partir de la emisión de la base. La forma de crear estas carpetas consiste en crear una iteración para cada base, que contenga otra iteración para cada año, y posteriormente crear el directorio con la función mkdir de la librería os. Todas las iteraciones antes mencionadas se realizan con *for*.

En este punto ya está todo listo para comenzar a armar la red, pero por simplicidad del código se decide hacer dos funciones que serán invocadas en la función que emita los mapas mensuales. La primera función se denomina color_nodo(), y su única entrada corresponde al string[4] de nombre de barra. Esta función pregunta si el numero 220 o 500 está en la cadena de texto, de estar 220 la salida de la función es la palabra "green", en caso de ser 500, la salida será "blue". La segunda función es la encargada de realizar el filtrado de temporalidad y tiene por nombre filtrado_temporalidad(), esta función recibe de entrada las siguientes variables:

1. desdeSING: fecha de fin de operación de la línea extraída directamente desde el docu-

[4]Cadena o palabra de texto.

mento. La forma de escritura es la misma que para *hastaSING*

2. hastaSING: fecha de inicio de operación de la línea extraída directamente desde el documento. Esta escrita de la siguiente manera: si la línea está en operación el valor es $*$, en caso contrario el nombre es "Mes + (primeras tres letras de un mes con la primera letra en mayúscula) + - + año", como por ejemplo, "MesEne-2015" o "MesDic-2020".
3. mesd: corresponde a la fecha (mes y año) de inicio de entrada en operación de la línea, pero con el formato mejorado, para esto el programa previo a invocar la función toma *desdeSING* y le cambia el prefijo "MesEne" por "1".
4. mesh: corresponde a la fecha (mes y año) de inicio de de salida de operación de la línea, el formato se modifica análogamente a *mesd* pero con la variable *hastaSING*
5. fecha: parámetro correspondiente a la fecha en la cual se desea estudiar si la linea está operativa, está escrito de la siguiente forma: "año-mes-dia", como por ejemplo, "2020-01-10".

Esta función considera todas las posibilidades según la notación de la CNE en que una línea puede estar operativa para un mes, y retorna "True" en caso de que lo esté, por ejemplo, si *desdeSING* y *hastaSING* valen "*", retorna True, si solamente *desdeSING* vale "*", utiliza *mesh* para verificar si la fecha de interés es menor a ella, si lo fuera retorna True, y así para todas las posibilidades. En caso de no estar operativa para el mes de estudio, la función retorna False.

El paso que viene a continuación marca el comienzo de la emisión de los mapas. A pesar de que los mapas se crean con una función, previo a esta se importan las coordenadas de las barras y el archivo de operación de lineas con sus columnas. Para obtener el archivo de coordenadas se ejecuta el siguiente código 5.1.

```
import pandas as pd
coordenadas = pd.read_csv('example_coor_nn.csv',encoding='↩
   1252')
Barra=coordenadas['Barra'].tolist()
df_coord= pd.DataFrame({'Barra':coordenadas['Barra'],'X':↩
   coordenadas['X'],'Y':coordenadas['Y']})
```

Código 5.1: Importación de archivo coordenadas.

El archivo que contiene las coordenadas se llama "example_coor_nn.csv", el cual es leido con la función read_csv, luego se guarda en una lista la columna con el nombre de las barras, la cual será utilizada más adelante. La última línea corresponde a generar el dataframe de coordenadas, el cual está formado de tres columnas, la primera columna *Barra* contiene el nombre de las barras, la segunda columna X contiene la posición X en coordenadas cartesianas de la barra, y la tercera columna Y contiene la coordenada Y de la barra.

Con respecto al archivo de operación de lineas, al momento previo de emitir un mapa, este se importa y se extraen de el las columnas con el inicio y fin de la operación de una línea, y las que contienen las barras extremas a la linea, denominadas en el script como desdeSING, hastaSING, sources y targets, respectivamente.

Para poder emitir un mapa se creó la función *crear_mapa_mensual*, la cual recibe de en-

tradas: hastaSING, desdeSING, sources, targets, mes (de emisión del mapa), anho (de emisión del mapa), fechahoy (en formato "año-mes-dia" y para la emisión del mapa), df_coord (archivo de coordenadas), anho_base (año de emisión de la base) y semestre_base (semestre de emisión de la base, 1 o 2). En sí la función es bastante extensa, y será analizada por extractos, los cuales está ordenados de manera cronológica. El código 5.2 muestra el primer extracto de la función, el cual define los nodos y las uniones entre estos.

```
def crear_mapa_mensual(hastaSING,desdeSING,sources,targets,↩
   mes,anho,fecha,df_coord,anho_base,semestre_base):
    dic={1:'Enero',2:'Febrero',...,12:'Diciembre'}
    titulo_got_net="Red␣"+dic.get(mes)+'␣'+str(anho)
    got_net = Network(layout=None,height="100%", width="100%"↩
       ,notebook=True,heading=titulo_got_net)
    edge_data = zip(sources, targets,desdeSING,hastaSING)
    for e in edge_data:
        src = e[0]
        dst = e[1]
        desdeSING=e[2]
        hastaSING=e[3]
        mesd=str(desdeSING).split('-')
        mesh=str(hastaSING).split('-')
        meses_num = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
        meses_nom = ['MesEne', 'MesFeb',..., 'MesDic']
        dict_meses = dict(zip(meses_nom, meses_num))
        if not mesd[0]=='*':
            mesd[0] = dict_meses.get(mesd[0])
        if not mesh[0]=='*':
            mesh[0] = dict_meses.get(mesh[0])
        if filtrado_temporalidad(desdeSING,hastaSING,fecha,↩
           mesd,mesh)==True:
            got_net.add_node(src,src,color=color_nodo(src))
            got_net.add_node(dst,dst,color=color_nodo(dst))
            got_net.add_edge(src,dst)
    neighbor_map = got_net.get_adj_list()
    vector_auxiliar_coord=[]
    Barra=df_coord['Barra'].tolist()
```

Código 5.2: Extracto función generadora de mapas.

La primera acción del algoritmo consiste en definir la función y se dejar expresadas las entradas de la misma, posteriormente se toma el mes de emisión del mapa y se define el título del mismo, el cual irá posicionado sobre la red. Para armar la red se requiere de la función *Network* de la librería *pyvis.network*, la cual fue escogida debido que al momento de definir nodos en ella, estos adquieren atributos útiles, entre ellos, "label" el cual define un mensaje informativo sobre el nodo, "color": color para del nodo o línea, "x" e "y": coordenadas cartesianas del nodo, entre otros. Al comienzo de la función se le asigna un título a la red, el cual se añade con el atributo "title" en la función *Network*. La red es denominada *got_net*.

A partir de lo anterior se arma un elemento del tipo *zip* con las listas sources, targets, desdeSING y hastaSING. Para definir cada nodo se hace una iteración sobre cada fila del zip y se modifica el string hastaSING y desdeSING con ayuda de un diccionario con los nombres y número de los meses del año. Dentro de la iteración de cada fila se llama a la función filtrado_temporalidad y si esta retorna True se añaden a la red los nodos extremos de la barra y el trazo de línea, esto con las funciones *add_node* y *add_edge*, respectivamente. Una vez recorridas todas las filas del zip se define la función *neighbor_map*, un vector auxiliar y una lista con los nombres de todas las barras, los que serán de utilidad mas adelante.

Por limitaciones de la librería Network no es posible posicionar todos los nodos a gusto si se tiene al menos un nodo sin posición, con lo cual el siguiente paso corresponde a agregarle a los nodos atributos. Se asegura que todos los nodos tengan sus atributos definidos, para lo cual se utiliza el *vector_auxiliar_cord*. Se realiza una iteración cuya condición declara: si la extensión del *vector_auxiliar_cord* es distinta a la cantidad de nodos en la red (los nodos se definen como got_net.nodes) ejecute lo siguiente:

1. Para cada nodo de la red identificado por su atributo "id" emita la información de las bases de datos CMg de la CNE y asígnesela como atributo de título del nodo.
2. Si el nodo está incluido en el archivo de coordenadas *df_coord*, se extraen de ese dataframe sus coordenadas cartesianas x e y, y se asignan como sus atributos "x" e "y" respectivamente. Además se asigna el atributo "physics" en False, para que el nodo esté estático y no se pueda arrastrar con el cursor. Ya que el nodo está en el archivo de coordenadas, se añade el nombre del nodo al *vector_auxiliar_cord*.
3. Si el nodo no está incluido en el archivo de coordenadas *df_coord* se identifican los vecinos o vecino del mismo con la función neighbor_map, luego existen dos casos:
   (a) El nodo sin posición tiene solamente un vecino: en este caso si el nodo vecino tiene posición, a partir de esta se calcula una posición aleatoria y cercana al nodo original. Una vez que se definió la posición del nodo se agrega esta al arreglo de posiciones y el nombre de la barra al vector auxiliar.
   (b) El nodo sin posición tiene más de un vecino: se guardan las posiciones para los primeros 2 vecinos que ya tengan posición y, a partir de esas ubicaciones, se le define una al nodo original. Una vez que se definió la posición del nodo se agrega esta al arreglo de posiciones y el nombre de la barra al vector auxiliar.

   Para cualquiera de estos dos casos, de no haber las posiciones mínimas requeridas para definir una posición al nodo original, se pasa al siguiente nodo sin posición, ya que el algoritmo está arreglado para que itere hasta que en el vector auxiliar estén inscritos todos los nodos de la red.

Lo anterior se muestra en el código 5.3. Una vez que todos los nodos estén inscritos en la red la función ejecuta tres lineas (visibles en el código 5.4) que dan fin a la función. La primera linea decreta el atributo *toggle_drag_node* como False, lo que indica que la red no es arrastrable desde sus nodos. La segunda línea define el nombre con que se guardará el archivo con el mapa, cabe destacar que la librería solo admite mapas en formato html. La ultima línea guarda el mapa en el directorio especificado.

```
    while len(vector_auxiliar_coord)!=len(got_net.nodes):
        for node in got_net.nodes:
```

```
            nombre_nodo=str(node['id'])
            titulo=info_precio_nudo(anho_base,semestre_base,↩
               anho,mes,nombre_nodo)
            node['title']=titulo
            if (nombre_nodo in Barra)==True:
                linea_nodo=df_coord.query('Barra␣==␣↩
                   @nombre_nodo')
                x=(linea_nodo['X'].sum())#solo para q sea X
                y=(linea_nodo['Y'].sum())
                vector_auxiliar_coord.append(nombre_nodo)
                node['x']=x
                node['y']=y
                node['physics']=False
            else: #Asegurare que todos los nodos tengan ↩
               posicion
                pos_x=[]
                pos_y=[]
                xx=0
                yy=0
                vecinos=neighbor_map[node["id"]]
                for vecino_nodo in vecinos:
                    linea_nodo_vecino=df_coord.query('Barra␣↩
                       ==␣@vecino_nodo')
                    x1=(linea_nodo_vecino['X'].sum())#solo ↩
                       para q sea X
                    y1=(linea_nodo_vecino['Y'].sum())
                    pos_x.append(int(x1))
                    pos_y.append(int(y1))
                    if pos_x[0]>0 and len(pos_x)>0:
                        xx=randint(pos_x[0]-105,pos_x[0]+105)
                        yy=randint(pos_y[0]-105,pos_y[0]+105)
                        while (pos_x[0]-20)<xx<(pos_x[0]+20):
                            xx=randint(pos_x[0]-105,pos_x↩
                               [0]+105)
                        while (pos_y[0]-20)<yy<(pos_y[0]+20):
                            yy=randint(pos_y[0]-105,pos_y↩
                               [0]+105)
                    else:
                        if len(pos_x)==2 and pos_x[1]>0:
                            xx=randint(pos_x[1]-105,pos_x↩
                               [1]+105)
                            yy=randint(pos_y[1]-105,pos_y↩
                               [1]+105)
                        if len(pos_x)==3 and pos_x[2]>0:
                            xx=randint(pos_x[2]-105,pos_x↩
                               [2]+105)
                            yy=randint(pos_y[2]-105,pos_y↩
```

```
                                    [2]+105)
                node['x']=int(xx)
                node['y']=int(yy)
                df_coord.append({'Barra':nombre_nodo,'X':xx,'↩
                    Y':yy},ignore_index=True)
                vector_auxiliar_coord.append(nombre_nodo)
```

Código 5.3: Armado de red.

```
        got_net.toggle_drag_nodes(False) #nodos no ↩
            arrastrables
    nombre='Mapas Generados/Bases_precio_nudo_'+str(anho_base↩
        )+'_'+str(semestre_base)+'/'+str(anho)+'/↩
        Sistema_Electrico_Chileno_'+str(mes)+'_'+str(anho)+'.↩
        html'
    return got_net.save_graph(nombre) #guardar grafico
```

Código 5.4: Guardado de mapas.

Con motivo de no ralentizar la aplicación, se emiten todos los mapas previo a la creación de esta, tal como se muestra en el código 5.5. Para lo cual se realizan iteraciones en todas las bases (”anho_base” y ”semestre_base”), en todos los años (”anhos_proyectados”) y en todos los meses (”mes”) de interés. En cada iteración de selección de base se definen los parámetros *desdeSING*, *hastaSING*, *sources*, *targets* y *fechahoy*.Este paso es bastante lento, en la realidad emitir todos los mapas de una base puede tardar entre 12 a 24 horas, y para verificar el avance se imprimen mensajes en la consola cuando los mapas de un año y de una base están emitidos.

```
for anho_base in [2018,2019,2020]:
    for semestre_base in [1,2]:
        archivo_tensiones_filtrado='Bases Precio Nudo y CMg/↩
            Dat_'+str(anho_base)+'_'+str(semestre_base)+'/↩
            SENdef_tension_filtrada_'+str(anho_base)+'_'+str(↩
            semestre_base)+'.csv'
        got_data = pd.read_csv(archivo_tensiones_filtrado,↩
            encoding='1252')
        hastaSING=got_data['LinFecOpeFin']
        desdeSING=got_data['LinFecOpeIni']
        sources = got_data['LinBarA']
        targets = got_data['LinBarB']
        for anhos_proyectados in list(range(anho_base,↩
            anho_base+10)):
             for mes in list(range(1,13)):
                anho=anhos_proyectados
                fechahoy=str(anho)+'-'+str(mes)+'-2'
                crear_mapa_mensual(hastaSING,desdeSING,↩
                    sources,targets,mes,anho,fechahoy,df_coord,↩
                    anho_base,semestre_base)
```

```
            print('Mapas completo para  ' +str(anho)+' base '+↩
                str(anho_base))
        print('base '+str(anho_base)+'_'+str(semestre_base)+'↩
            completa')
```

Código 5.5: Emisión de mapas.

Con el fin de que el lector comprenda los elementos incorporados en la plataforma, la figura 5.3 muestra la Aplicación CNE. Por aspectos visuales la aplicación fue cortada, originalmente el recuadro blanco corresponde a un rectángulo en el que se incorpora la red.

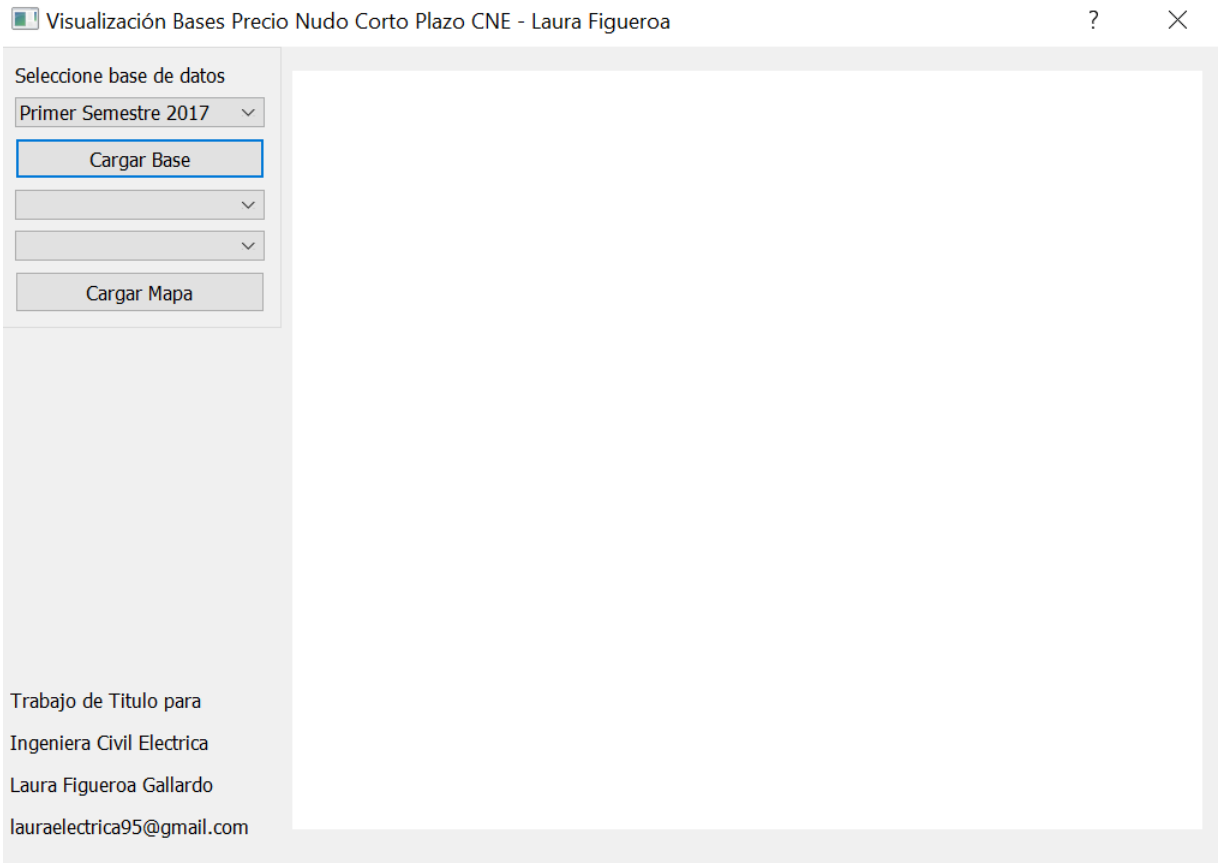


Figura 5.3: Estructura Aplicación CNE.

5.1.4. Creación de la Aplicación

Para la creación de la aplicación se utilizó el programa Qt Designer, el cual permite desarrollar interfaces gráficas de usuario[5]. La particularidad de usar este programa radica en que es fácil montar una aplicación con las cualidades que el desarrollador requiera (botones, pestañas, entrada de usuario con contraseña, etc.) y mediante una línea de texto en la consola de Python es posible generar el código de la plataforma en extensión py[6].

[5]Ventanas de interfaz que permiten, entre otras cosas, a un usuario realizar requerimientos.
[6]Extensión de los script de python.

El código que genera está basado en atributos de clase y caracteriza cada elemento y su configuración. A continuación se definen algunos elementos que se utilizan de aquí en adelante:

- Pushbutton: botón que el usuario puede clickear.
- Combobox: lista desplegable de la cual el usuario podrá leer y elegir solamente una opción.
- Label: elemento de texto.
- Event: modificación que realiza un usuario en la aplicación sobre un elemento, por ejemplo, apretar un Pushbutton es un event, o cambiar el elemento en un Combobox es un event.
- MainWindow: ventana principal de la aplicación sobre la cual se disponen los elementos.

Se decidió hacer una aplicación lo mas sencilla posible, por lo que la cantidad de elementos que contiene están reducidos al mínimo, así se tienen:

- Labels: uno en la parte superior para pedirle al usuario que seleccione una base de datos y otro en la parte inferior con la firma de la alumna.
- cbx_base: lista desplegable con las opciones de selección de bases de datos de la CNE.
- btn_cargar_base: pushbutton carga una base de datos.
- combobox_anho para escoger el año de la red que se desea visualizar.
- combobox_mes para escoger el mes de la red a visualizar.
- pushButton_cargarmapa: pushbutton que carga el mapa en el visualizador widget.
- widget: elemento del tipo *QWebEngineView* que es capaz de cargar elementos de extensión html.

Cabe destacar que Qt Designer diseña el formato de la ventana, es decir, los elementos que contiene, la ubicación y tamaño de estos, el texto que se visualiza, etc. Pero la ejecución de la aplicación requiere de la creación de un algoritmo, para esto se conectan todas las opciones de selección (combobox) con una acción y luego se compila todo en una clase.

La aplicación está compuesta de dos scripts independientes, el primero es el que se obtiene del diseño de Qt Designer, el cuales es convertido a lenguaje python mediante la siguiente linea de texto:

```
pyuic5 programa4_CNE.ui -o programa4_CNE.py
```

Siendo programa4_CNE.ui el archivo obtenido por Qt designer y programa4_CNE.py el código en lenguaje python, cabe destacar que no es necesario que se llamen iguales, esto se hace para evitar confusiones entre scripts. Los creadores de Qt Designer no recomiendan manipular el archivo a menos de no tener conocimiento al hacerlo[7], porque su escritura es

[7]La recomendación viene impresa como comentario en el código en .ui y se puede leer al convertir el archivo a .py: *WARNING: Any manual changes made to this file will be lost when pyuic5 is run again. Do not edit this file unless you know what you are doing.*

engorrosa y poco entendible, sin embargo el archivo si fue modificado. Por esta razón no se añaden extractos de este archivo en este informe.

Para explicar el segundo script se añaden extractos de código al documento. El primer fragmento se encuentra en el código 5.6, y consiste en la importación de las librerías que se utilizan. Cabe destacar que en este caso se importa la clase definida en el script anterior, la cual se denomina *programa4*.

```
import sys
from programa4_CNE import programa4
from PyQt5.QtWidgets import QDialog, QApplication
from PyQt5.QtCore import QUrl
```

Código 5.6: Importación de módulos.

El siguiente paso corresponde a crear la clase que describe el funcionamiento de la aplicación, lo cual es presentado en el código 5.7. La primera linea de este fragmento define la función ___*init*___, la cual establece la conexión entre los elementos:

- self.ui.btn_cargar_base.clicked.connect(self.cargar_combobox), esta línea se traduce en: "si el botón cargar base es apretado por el usuario, ejecute la orden contenida en la función cargar_combobox".
- self.ui.pushButton_cargarmapa.clicked.connect(self.btnIrClicked), esta línea se traduce en: "si el botón cargar mapa es apretado por el usuario, ejecute la orden contenida en la función btnIrClicked.

```
class programaapp4(QDialog):

    def __init__(self):
        super().__init__()
        self.ui = programa4()
        self.ui.setupUi(self)
        self.ui.btn_cargar_base.clicked.connect(self.↩
           cargar_combobox)
        self.ui.pushButton_cargarmapa.clicked.connect(self.↩
           btnIrClicked)  # app
        self.show()
```

Código 5.7: Conexiones elementos con interacción del usuario.

La función *cargar_combobox*() solamente recibe de argumentos la instancia self, y el evento generado al apretar el botón cargar base. El objetivo de esta función es rellenar las combobox de año y mes dependiendo de la base, por ejemplo, si se carga la base del 2018, las opciones de esta lista desplegable serían: 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026 y 2027, es decir, desde el año de emisión de la base hasta 9 años después. La combobox de mes siempre tendrá todos los meses del año, aunque para algunos casos[8] debería tener menos.

Con motivo de no acumular años repetidos en la combobox, la función comienza contando

[8]Para bases emitidas el 2^{do} semestre la combobox debería tener 6 meses para el primer y último año.

los elementos en ella, si hay más de un elemento, el script da la orden de eliminar todas las opciones en la lista desplegable. Luego la variable *base* recoge el valor que el usuario seleccionó en *cbx_base*, y guarda el año y semestre de la base, a partir del año agrega en la *comboBox_anho* cada año desde la emisión de la base hasta 9 más adelante con la función add.Item(). Lo anterior ocurre en forma análoga para la función *comboBox_mes*. El código 5.8 muestra la función cargar_comboBox.

```
    def cargar_combobox(self,event):
        option_anho=self.ui.comboBox_anho.count()
        if option_anho>1:
            self.ui.comboBox_anho.clear()
        option_mes = self.ui.comboBox_mes.count()
        if option_mes > 1:
            self.ui.comboBox_mes.clear()
        base=self.ui.cbx_base.itemText(self.ui.cbx_base.↩
           currentIndex())
        input_base=base.split('␣')
        dic_semestre={'Primer':1,'Segundo':2}
        semestre_input=dic_semestre.get(input_base[0])
        anho_input=int(input_base[2])
        anho_base=anho_input
        for anho in list(range(int(anho_base),int(anho_base)↩
           +10)):
            self.ui.comboBox_anho.addItem(str(anho))
        meses=['Enero','Febrero','Marzo','Abril','Mayo','↩
           Junio','Julio','Agosto','Septiembre','Octubre','↩
           Noviembre','Diciembre']
        for i in meses:
            self.ui.comboBox_mes.addItem(i)
```

Código 5.8: Función para llenado de listas desplegables.

Por su parte, la función btnIrClicked() (visible en el código 5.9) se activa cuando el usuario presiona el botón cargar mapa, cuyo funcionamiento es muy parecido al de la función anterior. Mediante itemText() la función recoge nuevamente el año y semestre de emisión de la base, como también el año y mes de visualización para el mapa. En función de estos cuatro parámetros la función busca el mapa en el directorio y lo carga con QUrl(), para luego mostrarlo en el widget utiliza load().

```
    def btnIrClicked(self, event):
        base = self.ui.cbx_base.itemText(self.ui.cbx_base.↩
           currentIndex())
        input_base = base.split('␣')
        dic_semestre = {'Primer': 1, 'Segundo': 2}
        semestre_base = dic_semestre.get(input_base[0])
        anho_base = int(input_base[2])
        anho_mapa = self.ui.comboBox_anho.itemText(self.ui.↩
           comboBox_anho.currentIndex())
```

```
        mes_mapa_input = self.ui.comboBox_mes.itemText(self.↩
           ui.comboBox_mes.currentIndex())
        dic={'Enero':1,'Febrero':2,'Marzo':3,'Abril':4,'Junio↩
           ':5,'Julio':7,'Agosto':8,'Septiembre':9,'Octubre'↩
           :10,'Noviembre':11,'Diciembre':12}
        mes_mapa=dic.get(mes_mapa_input)
        mapa = "file:///D:/Trabajo/Mapas␣Generados/↩
           Bases_precio_nudo_" + str(anho_base) + "_" + str(↩
           semestre_base) + "/" + str(anho_mapa) + "/↩
           Sistema_Electrico_Chileno_" + str(mes_mapa) + "_" +↩
            str(anho_mapa) + ".html"
        url = QUrl(mapa)
        self.ui.widget.load(url)
```

Código 5.9: Función cargar mapas.

Finalmente para poder ejecutar la aplicación se tienen cinco lineas de compilado, las cuales están disponibles en el código 5.10. Estas lineas están fuera del objeto de clase y al ejecutar el script completo la ventana con la aplicación se abre.

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    ventana = programaapp4()
    ventana.show()
    sys.exit(app.exec_())
```

Código 5.10: Inicialización Aplicación CNE.

5.1.5. Actualización Mapas

Considerando que las bases de datos de esta aplicación se descargan de manera manual, el usuario tendrá que realizar una acción para emitir los mapas correspondientes a cada base. Para ello se deben declarar el semestre y el año, y luego ejecutar el código 5.11 una vez que ya esté descargada la base.

```
anho_base=2021
semestre_base=1
archivo_tensiones_filtrado='Bases␣Precio␣Nudo␣y␣CMg/Dat_'+str↩
   (anho_base)+'_'+str(semestre_base)+'/↩
   SENdef_tension_filtrada_'+str(anho_base)+'_'+str(↩
   semestre_base)+'.csv'
got_data = pd.read_csv(archivo_tensiones_filtrado,encoding='↩
   1252')
hastaSING=got_data['LinFecOpeFin'] #en realiad es del SEN no ↩
   solo SING
desdeSING=got_data['LinFecOpeIni']
sources = got_data['LinBarA']
targets = got_data['LinBarB']
for anhos_proyectados in list(range(anho_base,anho_base+10)):
```

```
    for mes in list(range(1,13)):
        anho=anhos_proyectados
        fechahoy=str(anho)+'-'+str(mes)+'-2'
        crear_mapa_mensual(hastaSING,desdeSING,sources,↩
           targets,mes,anho,fechahoy,df_coord,anho_base,↩
           semestre_base)
        print('Mapas␣completo␣para␣␣␣'+str(anho)+'␣base␣'+str(↩
           anho_base))
print('base␣'+str(anho_base)+'_'+str(semestre_base)+'␣↩
   completa')
```

Código 5.11: Emisión mapas para otros años.

5.2. Aplicación CEN

La aplicación del CEN tendrá bastantes diferencias con la aplicación de la CNE, las principales son:

1. Utiliza cuatro bases de datos: Costos Marginales, Archivos de medida, Archivos de Generación Real y Archivos RIO (Registro de Instrucciones de Operación).
2. La distribución de la aplicación consiste en cuatro ventanas, no una.
3. Realiza Web Scraping para obtener la mayoría de las bases de datos.
4. No presenta una visualización de la red de transmisión nacional.

Sumado a las diferencias anteriores, es posible separar esta aplicación en tres apéndices: Web Scraping, Análisis Bases de Datos (que a la vez contiene la etapa de visualización de la información) y Creación de la Aplicación. Cada apéndice será categorizado dependiendo de la base de datos que se esté trabajando.

5.2.1. Web Scraping

Al comienzo del desarrollo de esta memoria fueron descargadas varias bases de datos con Web Scraping, las cuales pasaron por un filtro y finalmente se decidió utilizar solamente 4 bases de datos, las cuales fueron mencionadas anteriormente.

Bases Costos Marginales

Previo a descargar un archivo de esta base deben tenerse en cuenta las siguientes consideraciones:

1. El CEN podría subir más de una vez la versión definitiva de esta base mensual, producto de múltiples revisiones posteriores a la subida del archivo, con lo que podría haber más de un archivo.
2. Existen múltiples enlaces de descarga para este tipo de archivo. Según lo anterior, se seleccionaron en esta función solamente los enlaces que descargan archivos para el año 2020, ya que hacia atrás se entregarán las bases previamente descargadas.

3. En términos prácticos, los enlaces se descarga están compuestos por la fecha de emisión de la versión del archivo, específicamente mes y año. El algoritmo creado solamente considerará la versión emitida un mes después o menos del archivo.

Teniendo en cuenta lo anterior, se creó una función que tiene como entradas el mes y año correspondientes a la emisión del archivo. A partir de las variables anteriores se identifica la fecha propia de la información contenida en la base (un mes anterior a la emisión). Posteriormente se define el directorio final del archivo ya extraído en el formato de lectura (recordar que la base viene en formato .zip y el archivo de interés en .xlsm) y el directorio al cual se descarga el archivo (correspondiente a la misma ubicación en que se encuentra el script con la función). El algoritmo verifica que no exista el archivo mediante os.path.isfile() y, en caso de no existir, se le da la indicación de "intentar" descargarlo. Se le llama "intentar" debido a que utiliza el método *try/except*, si al tratar de descargar el archivo ocurre un error, este método asegura que el programa siga funcionando y ejecute la acción que sigue.

La librería *request* carga el enlace y su contenido en un directorio especificado, luego la función *Zipfile* de la librería zipfile descarga incluido en el zip en el directorio. Si el archivo no se descarga en el directorio final especificado, se traslada mediante la función move() de la librería shutill. En caso de que el algoritmo arroje error, o no corresponda el enlace de descarga, se pasa al siguiente. Actualmente esta función repite este proceso tres veces, una para cada enlace. El extracto de la función para solamente un enlace de descarga se muestra en el código 5.12.

```
def descarga_CMg(mes,anho):
    mes=mes-1
    if mes==-1:
        mes=12
        anho=anho-1
    mesmas1=mes+1
    if mesmas1<10:
        mesmas1='0'+str(mesmas1)
    if mes<10:
        mes='0'+str(mes)
    anho_emision=anho
    if int(mes)==12:
        mesmas1='01'
        anho_emision=anho+1
    archivogen="Archivos␣CMg/"+str(anho)+"/cmg"+str(anho)↩
       [-2:]+str(mes)+"_def.xlsm"
    archivo_descargado='cmg'+str(anho)[-2:]+str(mes)+'_def.↩
       xlsm'
    if os.path.isfile(archivogen)==False:
        try:
            enlaceCMg='https://www.coordinador.cl/wp-content/↩
               uploads/'+str(anho_emision)+'/'+str(mesmas1)+'/↩
               cmg'+str(anho)[-2:]+str(mes)+'_def.zip'
            r = requests.get(enlaceCMg, stream=True)
            z = zipfile.ZipFile(io.BytesIO(r.content))
```

```
            z.extractall()
            if os.path.isfile(archivogen)==False and os.path.↩
               isfile(archivo_descargado)==True:
                shutil.move(archivo_descargado,archivogen)
            if os.path.isfile('cmg'+str(anho)[-2:]+str(mes)+'↩
               _def.zip')==True:
                shutil.move('cmg'+str(anho)[-2:]+str(mes)+'↩
                   _def.zip',"Archivos␣CMg/"+str(anho)+"/cmg"+↩
                   str(anho)[-2:]+str(mes)+"_def.zip")
        except:
            'File␣is␣not␣a␣zip␣file'
```

Código 5.12: Descarga archivos Costos Marginales.

Archivos RIO

Para realizar la descarga de los archivos RIO debe tenerse en cuenta que estos comenzaron su emisión por parte del coordinador a mediados de noviembre de 2017, pero para fines prácticos se descargarán desde Enero 2018.

La emisión de estos archivos es de manera diaria, y es realizada mediante la librería Request ya que posee enlaces de descarga en el código fuente del sitio web[9]. El algoritmo que descarga los archivos consiste en una función que realiza lo siguiente:

- Recibe una fecha de argumento y arma el nombre del directorio del archivo.
- A partir de esa fecha busca en el directorio la existencia del archivo.
- Si el archivo existe la función termina, si el archivo no existe la función continúa.
- A partir del año, mes y día se arma el enlace de descarga. Se captaron cuatro enlaces distintos en el código fuente, esto se detecta al momento en que los archivos comienzan a descargarse y el programa detiene la ejecución del mismo debido a un error.
- Se utiliza la función *re = requests.get(enlace_archivo)*, la cual carga el enlace.
- Se abre la dirección del archivo guardada al principio, y se carga el contenido del archivo cargado con request.
- El programa utiliza las funciones os.chmod y os.access para permitir el acceso al archivo en caso de que estuviera bloqueado.

Este proceso está condicionado para que en caso de error en la descarga de un archivo, se pase al enlace siguiente y se pueda ejecutar con éxito la extracción del archivo. El error se detecta midiendo el peso en kb del archivo, el archivo aunque no exista con el primer enlace de descarga será extraído, pero tendrá un peso mínimo comparado con el archivo normal. El código 5.13 presenta un extracto de la función.

```
def descarga_RIO(fechahoy):
    #formato 'aaaa-mm-dd
    fecha=str(fechahoy).split('-')
    anho=str(fecha[0])
```

[9]https://www.coordinador.cl/operacion/documentos/registro-de-instrucciones-de-operacion-rio/

```
    mes=fecha[1]
    dia=fecha[2]
    archivoRIO="Archivos RIO/"+anho+"/RIO"+anho+mes+dia+".xls"
    if os.path.isfile(archivoRIO)==False:
        enlaceRIO='https://www.coordinador.cl/wp-content/uploads/'+anho+'/'+mes+'/RIO'+anho[2]+anho[3]+mes+dia+'.xls'
        re = requests.get(enlaceRIO)
        with open(archivoRIO, "wb") as code:
            code.write(re.content)
        if  os.access(archivoRIO, os.X_OK)==False or os.access(archivoRIO, os.R_OK)==False:
            os.chmod(archivoRIO, stat.S_IRWXO)
```

Código 5.13: Descarga Archivos RIO.

Archivos de Generación

La descarga de esta base de datos resultó ser aún mas sencilla que las anteriores, esto por dos razones: la primera es que el archivo solamente necesita de entradas un año y un mes, y la segunda radica en que desde el año 2010 a la fecha, se detectó solamente un enlace de descarga en el código fuente. El proceso de descarga sigue la idea anterior: a partir de las entradas (mes y año) se arma la dirección del archivo para el mes anterior (considerando que las entradas corresponden a la fecha actual), si el archivo no existe se carga mediante la librería request, y luego se verifica que estén habilitados los permisos de interacción con el archivo. La función se muestra en el código 5.14.

```
def descarga_gen(mes,anho):
    mes=mes-1
    if mes==-1:
        mes=12
        anho=anho-1
    archivoDR="Archivos Generacion Real/"+str(anho)+"/Generacion Real_"+str(anho)+"_"+str(mes)+".xlsx"
    if os.path.isfile(archivoDR)==False:
        enlaceDR='https://sipub.coordinador.cl/api/v1/static/gen_files/keys/'+str(anho)+'-'+str(mes)+'.xlsx'
        re = requests.get(enlaceDR)
        with open(archivoDR, "wb") as code:
            code.write(re.content)
            if  os.access(archivoDR, os.X_OK)==False or os.access(archivoDR, os.R_OK)==False:
                os.chmod(archivoDR, stat.S_IRWXO)
```

Código 5.14: Descarga archivos Generación Real.

Bases Archivos de Medida

Lamentablemente y debido al código fuente de la página web del Coordinador, específicamente del Sistema de Medidas[10],no fue posible implementar la automatización de descarga de estos archivos, los cuales son la base para el contenido de la sección *Retiros e Inyecciones* de la aplicación. Las razones específicas fueron las siguientes:

1. Los archivos no tienen en su código fuente un enlace directo de descarga, por lo tanto utilizar la librería *Requests* quedó descartado.
2. La página del sistema de medidas tiene más de 12 categorías, en las cuales el usuario puede hacer click y abrir la información disponible. Lamentablemente en el código fuente hay tres dificultades:
   (a) Las categorías en las cuales se pueden descargar archivos, están escritas con la misma clase, tienen pocos atributos y no tienen identificador, lo que hace imposible la identificación de la categoría de interés.
   (b) La dirección Xpath para descargar un archivo de interés solamente está disponible al momento en que se abre la categoría *REPORTE MEDIDAS HISTORIAS PRMTE*, y por la razón anterior no es posible abrirla.
   (c) El algoritmo no fue capaz de reconocer el trayecto desde la categoría de interés hasta el archivo final, se asume por la cantidad de símbolos que tienen las id. Sumado a lo anterior, y contrario a lo que hasta el momento se tenía claro, los archivos disponibles, a pesar de ser de años distintos, tienen el mismo identificador y ningún atributo que los diferencie. Esto se visualiza en la imagen 5.4.

Atributos descarga de archivo SSEE Q 1H 2018

```
▼<li role="treeitem" aria-selected="false" aria-level="6" aria-labelledby="SUBESTACION_Q_1H_
xlsx.zip_anchor" id="SUBESTACION_Q_1H_xlsx.zip" class="jstree-node  jstree-leaf">
    <i class="jstree-icon jstree-ocl" role="presentation"></i>
  ▶<a class="jstree-anchor" href="#" tabindex="-1" id="SUBESTACION_Q_1H_xlsx.zip_anchor">…
  </a> == $0
  </li>
```

Atributos descarga de archivo SSEE Q 1H 2020

```
▼<li role="treeitem" aria-selected="false" aria-level="6" aria-labelledby="SUBESTACION_Q_1H_
xlsx.zip_anchor" id="SUBESTACION_Q_1H_xlsx.zip" class="jstree-node  jstree-leaf">
    <i class="jstree-icon jstree-ocl" role="presentation"></i>
  ▶<a class="jstree-anchor" href="#" tabindex="-1" id="SUBESTACION_Q_1H_xlsx.zip_anchor">…
  </a>
  </li>
```

Figura 5.4: Atributos de descarga para un archivo de medidas en años distintos.

Iterar sobre elementos del tipo webdriver no es trivial, ya que los elementos tienen nombres inentendibles para un humano. Se intentó en reiteradas ocasiones generar el código que tomara los elementos y los activara o bien los descargara, pero los intentos no resultaron exitosos, por lo que finalmente se decide que la descarga de esta base de datos debe realizarse para cada mes de manera manual.

[10]https://medidas.coordinador.cl/

5.2.2. Análisis Bases de datos

Para el análisis de las cuatro bases anteriormente nombradas, se realizó el mismo procedimiento general: primero crear una función cuyas entradas fueran la fecha actual (en formato int) y si corresponde una barra o subestación. El resultado de esta función arroja un dataframe con las columnas de interés como por ejemplo fecha, hora y variable (costo marginal, generación, etc). El arreglo se utiliza de entrada para una nueva función que emite los gráficos y la información de interés. Todas las funciones dejan guardados los resultados en archivos del tipo txt, png o csv (si se guarda el arreglo). De la misma forma, cada script crea un directorio para guardar todos los archivos generados.

Costos Marginales

Desde el 2010 a la fecha, se encontraron 4 formatos distintos para este tipo de archivo:

1. Los archivos con información del SIC emitidos los años 2010, 2011 y 2012, incluyen dos pestañas de interés:
   (a) Una pestaña que incluye como columnas: nombre de barra, subsistema al que pertenece, y factores de penalización por bloque horario para cada día del mes.
   (b) Una pestaña que incluye tablas con los valores de costos marginales para todas las horas del mes. La cantidad de tablas disponible corresponde a la cantidad de subsistemas, es decir, por cada subsistema hay una tabla de costos marginales asociada.

   Para calcular el costo marginal en cierta barra, se extrae el subsistema al que pertenece, luego se guardan los valores de los factores de penalización y se multiplican por la tabla de costos marginales del subsistema al que pertenezca la barra.
2. Los archivos con información del SIC emitidos desde el 2013 hasta el 2017, contienen:
   (a) Una pestaña que incluye una columna con los costos marginales por barra para todas las horas del mes.
   (b) Una pestaña que incluye filas con los factores de penalización para una barra, por bloque horario, para todos los días del mes.
3. Los archivos con información del SING tienen el mismo formato desde el 2010 hasta el 2017, el cual consiste en una única columna que contiene el costo marginal real para todas las horas del mes.
4. Los archivos con información del SEN, emitidos desde el 2018 a la fecha conservan el formato de los archivos SIC 2013-2017.

Para poder calcular el valor real de costo marginal en los formatos 1, 2 y 4 se multiplicó el valor de los factores de penalización por el valor de la tabla o columna que contenía los costos marginales (ya sea por subsistema o no).

Se implementó una función cuyas entradas corresponden al mes, año y barra de interés. Esta función retorna un arreglo con todos los días del mes como columnas, y como filas las horas del día. Se consideran un total de 25 horas diarias, y en caso de no haber hora 25 o 24 se rellena tal fila con ceros. Los valores de la tabla se llenan con el cálculo del costo marginal.

La función considera los 4 formatos anteriormente descritos en el algoritmo. En el código

5.15 se inserta el extracto de la función que considera la ejecución del algoritmo para el formato 1.

```
def datos_CMg(mes,anho,barra):
    if anho==2010 or anho==2011 or anho==2012:
        archivo1='Archivos␣CMg/'+str(anho)+'/cmg'+str(anho)[-2:]+str(mesf)+'.xls'
        tablas_CM=pd.read_excel(archivo1,encoding='1252',sheet_name='matrices␣cmg',skiprows=5)
        tablas_FP=pd.read_excel(archivo1,encoding='1252',sheet_name='Factores␣de␣Penalizaci n',skiprows=4)
        if (barra in tablas_FP['BARRA'].tolist())==True:
            a=2
            df_subsistemaFP=tablas_FP.query('BARRA==@barra').iloc[0:1]
            subs1=df_subsistemaFP['SUBSISTEMA'][0:2].tolist()
            subsistema=subs1[0]
            if subsistema!='S1':
                index_subs=tablas_CMg.query('S1==@subsistema').index.tolist()
                indice_subs=index_subs[0]
            if subsistema=='S1':
                indice_subsistema=0
            df_subsistema=tablas_CMg.iloc[indice_subs+2:indice_subs+27,:]
            df_subsistema.set_index(list(df_subsistema)[0],inplace=True)
            df_subsistema.columns=list(range(1,31))
            df_CMg=pd.DataFrame({'Hora':list(range(1,26)),'1':None,...,'31':None})
            dias_mes=calendar.monthrange(anho,mes)[1]
            str_FP=''
            for dia in list(range(1,dias_mes+1)):
                for hora in list(range(1,26)):
                    str_FP=str_FP_columna(hora,dia)
                    if str_FP=='1' or str_FP=='2' or str_FP=='3':
                        str_FP=int(str_FP)
                    FP=df_subsistemaFP[str_FP].sum()
                    CMg_previo=df_subsistema.iloc[hora-1,df_subsistema.columns.get_loc(dia)]
                    if CMg_previo=='Revisar␣Hrs.':
                        CMg_previo=0
                    CMg_calculado=FP*(CMg_previo)
                    df_CMg.iloc[hora-1,df_CMg.columns.get_loc(str(dia))]=CMg_calculado
```

Código 5.15: Análisis Costos Marginales.

Para que la función acceda a un formato, se utilizan dos variables: *anho* y *a*. La primera se utiliza debido a que los formatos de cada archivo están diferenciados por año. La segunda es una variable auxiliar que indica que el algoritmo ya escogió un formato, si a=0, aún no se ha procesado un formato. Una vez que el algoritmo escoge un formato se arman dos arreglos:

1. El primero contendrá solamente las tablas con los costos marginales para los subsistemas ("subs") y cuyo header[11] corresponde a los días del mes para la tabla del primer subsistema.
2. El segundo arreglo corresponde a la tabla que incluye todos los factores de penalización para cada barra.

Se incluyen saltos de linea ("skiprows") para omitir las primeras filas vacías. A partir del segundo arreglo se extrae la columna que contiene todas las barras y se verifica si la barra de entrada de la función está o no en la columna. Si la barra está, el valor "a" cambia y se guarda en un pequeño arreglo (de una fila) la fila con los factores de penalización de la barra. Lo anterior se implementa con la función query(). A partir del arreglo se extrae el subsistema.

Se busca el índice de la tabla que contiene los valores previos de costos marginales para el subsistema de la barra. Considerando que el header del arreglo anterior corresponde a la primera fila de la tabla de el subsistema 1 ("S1"), se hace una excepción con dicho índice, imponiendo en cero el valor. En caso de que el subsistema no sea S1, el índice se busca en la columna de nombre S1 y se obtiene con el método query() e index(). A partir del arreglo y del índice, es posible extraer la tabla completa con los valores de costos marginales, debido a que su extensión corresponde a los días del mes (largo) y horas del día (alto). Por default todas las tablas incluyen la hora 25 con el valor "Revisar Hrs", el cual es cambiado a cero.

Finalmente, para obtener los valores reales del costo marginal, se arma un arreglo con 31 columnas vacías ("None") y 25 horas por columna. Para cada día (columna) y para cada hora (casilla) se multiplica el valor del factor de penalización por el valor del costo marginal. Para poder seleccionar el valor del factor de penalización correspondiente por bloque horario se incorporó la función *str_FP*, la cual retorna un 1, 2 o 3 dependiendo de que bloque horario sea. Una vez seleccionado el valor del FP se multiplica por el costo marginal y se incorpora el valor al arreglo vacío.

El segundo formato utiliza la misma metodología pero más simplificada debido a que no tiene la característica de subsistemas. En este caso es aún más directo, ya que a partir de la pestaña de costos marginales, se crea un arreglo con tres columnas: fecha, hora y valores de costo marginal para la barra de interés. A partir de este arreglo y el de factores de penalización, se calcula uno por uno el valor del costo marginal real y se incorpora en el arreglo vacío que al final retornará la función. La única diferencia que adiciona esta función en comparación con la anterior, es que analiza los nombres de barra, si la barra tiene el número 220 o 500 en su nombre, le permite armar el arreglo, en caso contrario no.

Para procesar el tercer formato de archivo, se implementó un código con bastantes diferencias a las anteriores. En primer lugar, el archivo descargado a pesar de tener extensión

[11] Primera fila del arreglo que define las columnas.

.xls no es compatible con excel, y se procesa con la función read_html(), la cual entrega mini arreglos que se concatenan para formar un único arreglo de datos. En segundo lugar solamente se carga una vez el archivo, ya que no hay factores de penalización. En tercer lugar, se arma el header del arreglo que contendrá los costos marginales a partir de la primera fila del arreglo concatenado y se fija como tal con la función .columns(), el header tiene por nombre *col_tablas2*. Posteriormente el proceso es análogo a los anteriores. El código 5.16 presenta el extracto de código con la primera parte del procesamiento del tercer formato.

```
    if (anho in [2010,2011,2012,2013,2014,2015,2016,2017]) ==↩
       True and a==0:
        archivo1='Archivos␣CMg/'+str(anho)+'/sing/↩
           rpt_cmg_en_barra_t_'+str(mesf)+str(anho)+'.xls'
        tabla_archivo = pd.read_html(archivo1)
        tablas=pd.concat(tabla_archivo)#,encoding='1252')
        tablas2=tablas.iloc[2:]
        header_ksi=tablas.iloc[1:2].values.tolist()
        header_def=[]
        for i in header_ksi:
            for j in i:
                header_def.append(j)
        tablas2.columns=header_def
        col_tablas2=list(tablas2)
        for i in col_tablas2:
            if (barra in i) ==True:
                barra=i
```

Código 5.16: Procesamiento tercer formato para Costos Marginales.

La función anterior retorna el arreglo denominado como df_CMg, el cual contiene una tabla de dimensión 31 columnas por 25 filas. Para poder entregar una visualización clara de los datos, se decidió calcular el promedio diario del costo marginal para cada barra, y el conjunto de todos los promedios del mes, presentando dichos datos en un gráfico de barras, en el cual también estará marcado el promedio mensual del costo marginal. Análogamente, se detallará a un costado del gráfico el mínimo y máximo costo marginal en el mes, y la cantidad de veces que fue cero.

La función que entrega la visualización y la información de esta base se denomina *promedio_CMg*(), y sus entradas corresponden al arreglo df_CMg (emitido por la función anterior), la barra, el mes y el año de interés. La función comienza definiendo cuantos días hay en ese mes, esto es automático gracias a la función *monthrange* de la librera *calendar*. Posteriormente se crean varias listas vacías:

1. promedio_dias: corresponde al promedio diario del costo marginal para los días que no sean domingo.
2. dias: corresponde al número de día que no es domingo en el mes de interés.
3. promedio_domingos: corresponde a la lista con el promedio del costo marginal para todos los días domingo
4. domingos: lista con la numeración de todos los días domingo.

5. dias_raros: corresponde a una lista de tuplas en la que se guarda en primera posición la cantidad de horas del día, y en segunda posición el número del día. Esta lista solo almacena datos si el día tiene 23 o 25 horas, es decir, ocurre cambio de hora.

De la misma manera, en comienzo se definen dos variables con valor igual a 0. La primera corresponde a *suma_dias*, la cual suma todos los promedios diarios, mientras que la segunda es *ceros*, parámetro que acumula las veces en que el costo marginal vale cero. Posteriormente se recorren todos los días del mes y cada día se extrae a una lista la columna con los costos marginales para dicho día. Luego se cuentan las veces en que el valor es cero mediante la función *count(0)*, cuyo valor se guarda en "*veces_cero*. En el contador suma_dias se almacena la suma de la columna con los costos marginales.

Para poder obtener el promedio del costo marginal diario, el programa por default dividirá la suma de valores en 24, pero es necesario identificar cuantas horas hay en el día. Para ello se suman los valores de costos marginales desde la hora 23 a la 25 y de la hora 25. Si estos valores equivalen a cero, significa que el día tenía 23 o 25 horas, respectivamente. En ese caso se guarda la tupla y se calcula el promedio diario en función de las horas detectadas.

De manera independiente, se calcula qué día de la semana es cada día del mes mediante la función *isoweekday*, la cual arroja 1 si el día es lunes, 2 si es martes, y así sucesivamente. Si el día es domingo, se almacena aparte la numeración del día y el valor del promedio del costo marginal. Para calcular el promedio mensual se tomará la suma de estos valores obtenida en *suma_dias*, y se dividirá por las horas que correspondan. Para identificar las horas se utiliza la tupla *dias_raros*. Finalmente se obtienen los valores máximo y mínimo con las funciones *max*() y *min*(), respectivamente.

Para elaborar el gráfico de barras se utiliza la libreria matplotlib y se consideran dos agrupaciones x e y. El primero formado por *dias* y *promedio_dias* y el segundo formado por *domingos* y *promedio_domingos*. El extracto de la función se presenta en el código 5.17.

```
def promedio_CMg(df_CMg,barra,anho,mes):
    dias_mes=calendar.monthrange(anho,mes)[1]
    suma_dias,ceros=0 #este es para el promedio mensual
    promedio_dias,dias,promedio_domingos,domingos,dias_raros↩
       =[]
    for i in list(range(1,dias_mes+1)):
        columna=df_CMg[str(i)].tolist()
        veces_cero=columna.count(0)
        suma_dia=df_CMg[str(i)].sum()
        suma_dias+=suma_dia
        #verificar si hay dias con cambio de hora
        suma23=df_CMg[str(i)].iloc[22:].sum()
        suma25=df_CMg[str(i)].iloc[24:].sum()
        promedio_dia=suma_dia/24
        #analogo para el dia 25, diferencia en f. ceros
        if suma23==0:
            dias_raros=[23,i]
            promedio_dia=suma_dia/23
```

```
            ceros+=veces_cero-2
        tipo_dia=datetime.isoweekday(datetime(anho,mes,i))
        #analogo para los dias domingo
        if tipo_dia!=7:
            promedio_dias.append(promedio_dia)
            dias.append(i)
        if len(dias_raros)==0:
            ceros+=veces_cero-1
    if len(dias_raros)==0:
        promedio_mes=(suma_dias)/(24*dias_mes)
    elif len(dias_raros)==2:
        promedio_mes=(suma_dias)/(dias_raros[0]+24*(dias_mes↩
           -1))
    promedios_todos=promedio_dias+(promedio_domingos)
    max_CMg=max(promedios_todos)
    min_CMg=min(promedios_todos)
```

Código 5.17: Elaboración de gráficos para Costos Marginales.

Archivos RIO

La visualización de la información de esta base contempla tres elementos:

- Un gráfico de torta en que se indica, para una barra, que porcentaje del tiempo estuvo marginando una unidad de generación térmica e hidráulica. La cantidad de barras que pueden acceder a este gráfico es limitada por la información de los Archivos RIO.
- Un gráfico de barras que indica la cantidad de veces en que el total de centrales del sistema incurrieron en una de las siguientes consignas: vertimiento (V), fuera de servicio (FS), mínimo técnico (MT), en prueba (EP), proceso a mínimo técnico (PMT), mínimo técnico provisorio (MTP) y seguridad del sistema(SS).
- Una tabla que contiene la cantidad de veces y el tiempo en que una unidad generadora incurrió en las consignas anteriormente mencionadas.

Para la función que emite el arreglo de datos debe considerarse que, a medida que los archivos se van emitiendo, estos cambian con el tiempo:

- Se modifica la fila inicial del archivo.
- Se modifican los nombres de cada columna.
- Algunos meses incluyen las barras con las unidades que marginan en ella y otros no.
- Algunos meses incluyen las barras con las unidades que marginan, pero no incluyen todas.

Lo anterior tiene como consecuencia que la función que trabaja los datos del archivo, sea monótona y repita procesos varias veces, ya que debe estar preparada para cualquiera de estos cambios, por ejemplo, lee el archivo saltándose la primea fila (así la segunda queda como header) y luego lee saltándose la primera y segunda fila (para que la tercera quede como header). De la misma manera, se identifican varias columnas de interés con distintos

nombres. Razón por la cual la función es tediosa de leer y no se incorpora a este apéndice, mas si será explicada en detalle e incluída en el Código 7.2 en los Anexos.

La función tiene como entrada el mes y año de interés. Los recibe y a partir de ellos arma un listado con todos los días del mes. Cabe mencionar que los archivos RIO se emiten uno al día. A partir de la fecha se arma el directorio con la ubicación del archivo y se carga con la función ExcelFile(). Posteriormente se identifican las pestañas de interés mediante sheet_names(). Este archivo contiene tres pestañas: la primera corresponde aun glosario con las siglas, la segunda a la pestaña con la información operacional del día, y la tercera con la información de los servicios complementarios. La pestaña de interés corresponde a la segunda, la cual puede llamarse: *Mov Centrales CMG*, *MOV-CMG* o *Movimiento de Centrales*. Es obligatorio identificarla por el nombre debido a que no todos los archivos incorporan las tres pestañas, con lo que un índice fijo podría elegir otra pestaña. Una vez encontrada la pestaña se lee únicamente aquella pestaña con la función read_excel().

El arreglo que emite la función tiene incorporadas las siguientes columnas:

- 'indice': columna adicional que indica el número de fila.
- 'fecha': fecha en formato aaaa-mm-dd
- 'Tipo de dia': día de la semana (lunes, martes, etc.). Esta salida tenía un uso al comienzo del trabajo pero luego se descarto su utilidad.
- 'Hora Movi.': Corresponde a la hora exacta de una instrucción o cambio de consigna para una línea o unidad generadora.
- 'Central-Unidad': central o unidad generadora que incurrió en una consigna.
- 'Configuracion': configuración de una unidad generadora que incurrió en una consigna, algunas veces esta casilla puede ser idéntica a la anterior.
- 'Estado': estado funcional de una unidad generadora.
- 'Consigna': consigna que marca una unidad generadora.
- 'CRUCERO___220', 'DALMAGRO___220', 'CARDONES_220', 'PAZUCAR___220', 'QUILLOTA___220', 'AJAHUEL___220', 'CHARRUA___220', 'PMONTT____220': cada una de estas barras tiene asignada una columna en la cual se indica que configuración está marginando en ella.

Una vez que el algoritmo es capaz de encontrar estas columnas en el archivo, las guarda e incorpora en un dataframe llamado *df_rio*. Existe la posibilidad de que no estén en el archivo todo el listado de barras que indican qué unidad está marginando, en ese caso las columnas quedan en vacío (None). Probablemente el algoritmo lea hasta tres veces el archivo con distinto salto en las primeras filas, esto se hará hasta que existan la mayoría de las columnas (todas excepto las barras). Una vez armado el primer arreglo (para el primer día del mes) con las columnas mencionadas anteriormente, se guarda y se procede a recorrer el listado de fechas desde el día 2 hasta el último día del mes, repitiendo todo el proceso. Los archivos se van concatenando con el método concat() y se les define un índice adecuado a partir del largo actual del arreglo (concatenado para cada día) y del largo de una columna del segundo arreglo a concatenar. Finalmente la función retorna df_rio.

Para generar el gráfico de torta se creó la función *func_tec_marginando*, la cual recibe

como entradas df_rio, el año, mes y la barra de interés (de las columnas del archivo RIO). Al igual que la función anterior, y debido a variaciones en el archivo, esta función es repetitiva pero asegura que todos los datos serán tratados, se adjunta en el Código 7.3 en Anexos.

La ejecución consiste en crear contadores de tiempo con la función timedelta[12] para las tecnologías hidráulicas, térmicas y además un tercer contador para aquellos días en que no vengan incorporadas las barras en el archivo. Se creó un vector para cada tecnología que incluye todas las configuraciones posibles que podrían marginar en una barra. La función recorre todas las filas del arreglo de entrada con la función iterrows() (la cual guarda el índice de la fila) y almacena la configuración en formato string (*valor_columna*) y sin procesar (*configuracion1*), la hora (*hora1*) y fecha (*fecha1*) indicados en la fila. Si los formatos son legibles la función pasa a calcular la fecha y hora de la siguiente fila.

Si la configuración previamente guardada, está en el vector de tecnologías hidráulicas, se procede a hacer la resta entre fechas y a acumular dicho valor en el contador, de manera análoga para la tecnología térmica. Si la configuración no estuviera en ninguno, o simplemente no hubiera configuración, se suma al tercer contador. Una vez el algoritmo haya recorrido todo el arreglo de datos, se calculan las horas del mes y el porcentaje en que cada tecnología marginó sobre la barra para armar el gráfico de torta con la función pie() de matplotlib. La función completa será anexada al final del documento.

Para generar el gráfico de barras se crearon dos funciones, la primera denominada *contador_accion*, la cual tiene como entradas df_rio y una consigna de interés y de salida un arreglo de datos que contiene todas las centrales que incurrieron en una consigna específica y la cantidad de veces y tiempo de recursión de la misma. La segunda tiene por nombre *unidades_marcando* cuyas entradas son nuevamente df_rio, el mes y el año.

La función *contador_accion* realiza las siguientes acciones en orden cronológico:

- Guarda el header de df_rio y le modifica el nombre a la columna que almacena las configuraciones, esto por problemas de lectura ya que el nombre original contiene un guión.
- A partir de df_rio se arma un arreglo denominado *accion1* que contiene solamente aquellas filas en que la consigna marca el valor indicado en la entrada de la función (query()) y almacena la fila que contiene la unidad generadora (accion). Esta fila es filtrada con el método unique() y entrega como resultado dos listas: la primera contiene el nombre de centrales sin valores repetidos (*centrales*) y la segunda contiene la cantidad de veces que estaban repetidas en la fila (*veces_accion*).
- Se arma un arreglo con las siguientes columnas:
    1. Central: cadena de texto con la unidad generadora.
    2. Accion: cadena de texto con la consigna que realiza la unidad generadora.
    3. Veces: número de veces en el mes en que la unidad generadora marca dicha consigna.
    4. Contador: tiempo acumulado que la unidad generadora incurrió en la consigna en un mes, del tipo timedelta().

[12]Esta función recibe de entradas días, horas y segundos y permite la suma y resta horaria.

- Se realiza una iteración sobre la lista de centrales, detectada anteriormente, que incurren en la consigna de entrada de la función. La iteración incluye los siguientes pasos:
    1. Crea un contador en cero para cada central con la función timedelta(0,0,0).
    2. Si la consigna de entrada es "FS" el contador de salida estará en cero y este valor se agrega al arreglo.
    3. Si la consigna de entrada es distinta a "FS", se selecciona con el método query() y el arreglo *accion1*, todas las filas en que la central de interés incurrió en la consigna de entrada. Este conjunto de filas se guarda en la variable *df_accion_central*. Luego se realiza una segunda iteración que va desde 0 hasta el último elemento de la lista *veces_accion*. La iteración ejecuta las siguientes instrucciones:
        (a) A partir del índice de la iteración y la función iloc(), retorna la fila del arreglo *df_accion_central* que contiene la unidad de interés y la consigna de interés. Desde esa fila se extrae la hora y fecha en que la central incurrió en la consigna.
        (b) Debido al formato de archivo del CEN, se encontraron al rededor de 3 fechas que no fueron posibles de analizar debido a que el archivo esta incompleto. El algoritmo verifica que la fecha tomada anteriormente no sea alguna de estas y prosigue con la ejecución del código.
        (c) Se guarda la hora en que ocurrió la consigna y la configuración, la cual se obtiene a partir del arreglo de una línea. El formato de hora es del tipo timedelta.
        (d) Se crea otro arreglo llamado *posible_fin_accion1*, el cual selecciona con el método query todas las filas del df_rio que tengan en común: la fecha, la central y la configuración de interés.
        (e) Si el arreglo está vacío la función no hace nada.
        (f) En caso contrario extrae la columna con las horas del arreglo anterior, la cual contiene todas las horas desde que la central incurrió en la acción hasta el ultimo día del mes.
        (g) Busca el índice en que esta contenida la hora en que la central incurrió en la acción y además guarda la cantidad de horas que hay registradas.
        (h) Dada la cantidad de coincidencias que se han incorporado al arreglo, se asume que la hora siguiente en la lista corresponde a la hora que marca el fin de la consigna, la cual es guardada en formato timedelta. Lo anterior es independiente de la consigna asociada a la segunda hora.
        (i) El contador suma la diferencia de horas.
        (j) En caso de que la hora corresponda al día siguiente, se verifica aquello y se suma al contador.
    4. Una vez terminado el recorrido por todas las veces que la central incurrió en la consigna, se guarda el valor del contador en el dataframe creado al comienzo.
    5. El valor del índice cambia al siguiente.
- Se termina la función.

Debido a la complejidad de la función, esta no será incluida en este apéndice pero si incorporada en la sección de Anexos Código 7.4.

Para poder emitir el gráfico de barras se ejecuta la función *unidades_marcando*, la cual recibe como entradas: df_rio, mes y año de interés. Las salidas de esta función corresponden al gráfico de barras y un arreglo que contiene todos los contadores por central con consigna asociados.

La función comienza generando para cada consigna un arreglo que contiene las centrales con la consigna asociada y el contador de tiempo, esto mediante la función *contador_accion*. Posteriormente define que si existe una sigla que no tenga datos asociados, el arreglo estará vacío. Para generar el gráfico de barras se tiene como eje X los nombres de todos los estados[13] posibles de interés y como eje Y el largo de la extensión del arreglo de cada consigna. El gráfico se arma con la función matplotlib. Con respecto al dataframe de salida de la función se concatenan todos los arreglos del principio en uno solo. A su vez también se emite la lista de todas las centrales que tienen asociada una condición de interés, esta lista se utiliza más adelante. El extracto de la función se muestra en el código 5.18.

```
def unidades_marcando(df_rio,mes,anho):
    MT=contador_accion(df_rio,'MT')
        .
        .
        .
    V=contador_accion(df_rio,'V')
    if MT['Central'].sum()==0:
        MT=''
        .
        .
        .
    if V['Central'].sum()==0:
        V=''
    x=['FS','MT','PMT','SS','MTP','EP','V']
    y=[len(FS),len(MT),len(PMT),len(SS),len(MTP),len(EP),len(↩
       V)]
    consignas=[MT,PMT,SS,MTP,EP,V]
    df_consignas=FS
    for consig in consignas:
        if len(consig)!=0:
            df_consignas=pd.concat([df_consignas,consig])
    centrales_def=pd.unique((df_consignas['Central'].tolist()↩
       ))
    return df_consignas,centrales_def
```

Código 5.18: Función unidades_marcando().

A pesar de que la función tiene dos salidas, los elementos obtenidos se guardan en png (gráfico de barras), txt (listado centrales) y csv (arreglo de datos). Finalmente se creó la función *datos_x_central*, la cual recibe como entradas una unidad generadora del listado emitido por la función *unidades_marcando*, y el dataframe obtenido de la misma función. La ejecución de instrucciones es la siguiente:

[13]Estados operacionales, instrucciones y consignas.

- La función reduce el arreglo hasta las filas que únicamente contengan la central mediante la función query() y convierte a lista las columnas del arreglo (Veces, Contador y Acción) con la función tolist().
- Guarda la extensión de las listas (recordar que como es un arreglo de datos, las tres listas tienen la misma extensión).
- Se itera en torno al largo de lista: para cada iteración se arma una lista más pequeña con los tres valores de las columnas ya guardadas.
- Una vez que terminan las iteraciones se elabora una tabla de datos con la función format(). Esta tabla es la salida de la función.

La función completa se muestra en el código 5.19.

```
def datos_x_central(df_consignas,central):
    veces=df_consignas.query('Central␣==␣@central')['Veces'].↩
       tolist()
    tiempo=df_consignas.query('Central␣==␣@central')['↩
       Contador'].tolist()
    accion=df_consignas.query('Central␣==␣@central')['Accion'↩
       ].tolist()
    cantidad_consignas=len(veces)#con tiempo o veces daba el ↩
       mismo resultado
    tuplas=[['Consigna','Ocurrencia','Tiempo␣acumulado']]
    for i in list(range(0,cantidad_consignas)):
        if accion[i]!='FS':
            tuplas.append([accion[i] ,str(veces[i])+'␣[veces/↩
               mes]',(str(tiempo[i])[:-7]).replace('days','↩
               d as')+'␣horas'])
        elif accion[i]=='FS':
            tuplas.append([accion[i] ,str(veces[i])+'␣[veces/↩
               mes]',(str(tiempo[i])[:-7]).replace('days','↩
               d as')])
    a=('\n'.join(['''.join(['{:16}'.format(x) for x in r]) for↩
        r in tuplas]))
    return a
```

Código 5.19: datos_x_central().

Generación Real

El análisis de esta base comienza con la emisión de un arreglo con los datos de interés. La función responsable de aquello se llama *datos_gen*, la cual recibe como entradas el mes y año de interés. La función entrega un arreglo con las siguientes columnas:

- Fecha: fecha diaria en formato aaaa-mm-dd.
- Dia: únicamente un string con el numero de día
- Tipo_dia: columna indicando las dos primeras letras del día de la semana de la fecha indicada, por ejemplo "Lu" para día lunes.

- Tipo: tipo de fuente de generación: térmica, eólica, etc.
- Subtipo: subtipo de fuente de generación, por ejemplo de embalse o pasada.
- Generación: columna con la cantidad de generación nacional para cierta fecha.

La función resulta ser bastante sencilla: en primer lugar se importa el archivo y se guarda como dataframe, posteriormente se guardan las columnas *Fecha* y *Total* extraídas del archivo. La segunda columna corresponde a la suma diaria de la generación de una central. Cabe mencionar, que el archivo incluye una fila para cada día que la central estuvo generando. En segundo lugar y para cada fecha, se arma una columna con el tipo de día (función isoweek) y con el día (en número). Finalmente se arma el archivo. Esta función está disponible en el código 5.20.

```
def datos_gen(mes,anho):
    archivo='Archivos␣Generacion␣Real/'+str(anho)+'/↩
       Generacion␣Real_'+str(anho)+'_'+str(mes)+'.xlsx'
    tabla_gen = pd.read_excel(archivo,encoding='1252',↩
       sheet_name='Sheet',skiprows=3)#index_col=4)
    gen_diaria=tabla_gen['Total']
    gen_fecha=tabla_gen['Fecha']
    gen_dia,gen_tdia=[]
    n=0
    for i in gen_fecha.tolist():
        if n<(len(gen_fecha)-1):
            fecha=str(i).split('-')
            tipo_dia=datetime.isoweekday(datetime(int(fecha↩
               [0]),int(fecha[1]),int(fecha[2])))
            if tipo_dia==1:
                gen_tdia.append('Lu')
            if tipo_dia==2:
                gen_tdia.append('Ma')
            ...
                gen_tdia.append('Do')
            gen_dia.append(int(fecha[2]))
            n+=1
        if n==(len(gen_fecha)-1):
            gen_dia.append(None)
            gen_tdia.append(None)
            n+=1
    gen_tipo=tabla_gen['Tipo']
    gen_subtipo=tabla_gen['Subtipo']
    df_gen = pd.DataFrame({'Fecha':gen_fecha, 'Dia':gen_dia,'↩
       Tipo_dia':gen_tdia,'Tipo':gen_tipo,'Subtipo':↩
       gen_subtipo,'Generacion':gen_diaria})
    df_gen=df_gen.query('Generacion␣>␣0') #no importa si no ↩
       se genero
    return df_gen
```

Código 5.20: Función datos_gen().

Se creó una función llamada gen_pref_user() cuyas entradas corresponden al arreglo emitido por la función anterior (df_gen) y la variable segregación. Si segregación vale 0, la función emite un gráfico de torta que indica el porcentaje de aporte en energía que tuvo cada tipo de tecnología en la generación del mes de interés. Si segregación vale 4, la función emite un gráfico de barras con la generación diaria y la distribución del aporte por tecnología, además de un texto en que se indica el promedio diario de generación.

La función comienza armando una lista con todas las fechas del mes, y listas vacías para incorporar la suma diaria del aporte de generación para cada tecnología (en el extracto 5.19 se aprecia de ejemplo para el tipo hidráulica de pasada). Por cada día del mes la función arma un arreglo con la generación de ese día, a partir de esa tabla genera arreglos más pequeños con la generación diferenciada por tecnología de interés. Lo anterior se realiza con la función query y los parámetros *Fecha* y *Tipo* (y *Subtipo* si corresponde). Para armar el gráfico, se tiene que el eje X corresponde a una lista con todos los días del mes, mientras que el eje Y corresponde a la lista de generación que se fue acumulando día a día. En el caso del gráfico de torta, se calcula la suma total de generación en el mes y el aporte en porcentaje de cada tecnología. El valor de la generación mensual total se calcula sumando la columna denominada ”Generación”, esto con el método .sum(). El código 5.21 muestra lo anterior.

```
def gen_pref_user(df_gen,segregacion):
    dic={1:'Enero',2:'Febrero'...,12:'Diciembre'}
    dia_uno=df_gen['Fecha'][0]
    inicio=dia_uno.split('-')
    anho=inicio[0]
    mes=inicio[1]
    dias_mes=calendar.monthrange(int(anho),int(mes))[1]
    inicio = datetime(int(anho),int(mes),1)
    fin    = datetime(int(anho),int(mes),dias_mes)
    lista_fechas = [(inicio + timedelta(days=d)).strftime("%Y↩
       -%m-%d") for d in range((fin - inicio).days + 1)]
    dia_uno=datetime(int(anho),int(mes),1)
    datos_final=''
    gen_pasada=[]
    for dia in lista_fechas:
        df_dia=df_gen.query('Fecha␣==␣@dia')
        df_hidro_pasada=df_dia.query('Tipo␣==␣"Hidr ulica"␣↩
           and␣Subtipo␣==␣"Pasada"')['Generacion'].sum()
        gen_pasada.append(df_hidro_pasada)
    x=list(range(1,dias_mes+1))
    y3=gen_pasada
```

Código 5.21: Función gen_pref_user().

Archivos de Medida

La visualización de esta base tiene como resultados la emisión de un gráfico de barras con los retiros e inyecciones de potencia activa por medidor para cada barra. Además se presenta un pequeño cuadro resumen en que se incluyen el valor total de retiros e inyecciones

asociados.

El formato de esta base de datos es el siguiente: Para cada mes del año, existen tantas carpetas como letras del abecedario, cada carpeta tiene como identificador una letra, la cual corresponde a la inicial de una subestación. Dentro de la carpeta existen múltiples archivos que contienen ordenadas alfabéticamente las subestaciones. La cantidad de archivos solamente depende de la cantidad de subestaciones que comiencen con la inicial, un archivo no corresponde a una subestación, si no que agrupa varias. Cada archivo contiene en su interior 4 columnas por cada medidor de cada subestación: Retiros de Potencia Activa, Retiros de Potencia Reactiva, Inyecciones de Potencia Activa e Inyecciones de Potencia Reactiva. Las columnas contienen valores para todas las horas del mes.

El análisis de esta base solamente contemplará las columnas Retiros e Inyección de Potencia Activa y requiere de únicamente una función que realiza el análisis y la visualización de la información. La función encargada de lo anterior se denomina *iny_retiros* y tiene como entradas el mes, año y la subestación de interés. El algoritmo realiza las siguientes instrucciones:

- Identifica la inicial de la subestación, y en torno a ella se detecta la carpeta que contiene todos los archivos con dicha inicial con la función listdir().
- Se comienza una iteración que recorre todos los archivos del directorio especificado.
- Se lee el archivo dos veces: la primera vez para extraer en la cuarta fila los nombres de todos los medidores mediante la función list(), estos incluyen el nivel de tensión de la subestación. La segunda lectura se realiza para extraer el nombre de la subestación en la séptima fila y dejar aquella fila como header del dataframe. De la misma manera se obtiene la lista de subestaciones con el método list().
- Se define un contador llamado *indice*, el cual inicializa en 0.
- Si la subestación de entrada no está incluida en el listado de barras, se termina la iteración y pasa al siguiente archivo.
- En caso contrario (esté incluida), se crea otro contador llamado *veces_barra*, el cual contará todas las veces en que una columna se llame igual a la subestación de interés. Cabe mencionar que cuando un arreglo tiene nombres de columnas repetidos, la segunda y demás columnas que repitan el nombre, tendrán de sufijo *.numero*, siendo número un entero entre 1 y el valor de repeticiones menos 1.
- Entonces, si la subestación está repetida, se sumara la cantidad de repeticiones a *veces_barra* y se guardan los índices de la columna. Los cuales van de 4 en 4 considerando las cuatro columnas asociadas a un medidor.
- Si la subestación no se repite, se extrae del arreglo original solamente cuatro columnas. En caso contrario y considerando que las medidas de los medidores de una subestación son consecutivas en el archivo, se toman las columnas del arreglo desde el índice de la subestación original, hasta el índice más 4 de la última encontrada.
- Este proceso se repite para todos los archivos, ya que la información de los medidores de una subestación puede estar repartida entre archivos.
- El algoritmo obtendrá un arreglo final dedicado a todos los medidores de una subestación. El paso siguiente consiste en separar a dos arreglos: el primero para considerar únicamente las inyecciones de potencia activa, y el segundo para los retiros de potencia

activa, lo anterior se implementa con la función filter(), la cual selecciona todas las columnas que cumplan con una condición de semejanza, seleccionando todas las columnas cuyo encabezado sea "Inyección Potencia Activa" y "Retiro Potencia Activa.

- Posteriormente se guardará una lista con todos los identificadores de medidores, los cuales corresponden a una parte del nombre de la casilla.
- Finalmente se itera desde 0 hasta la cantidad de repeticiones, también se agrega a una lista vacía la suma de inyecciones y retiros (en listas separadas), lo anterior mediante las funciones sum()[14] y append()[15].
- Con los vectores de inyección, retiros e identificador del medidor, es posible realizar el gráfico con matplotlib.

5.2.3. Automatización del proceso

El trabajo completo necesita la automatización de dos procesos: el primero consiste en mantener la continuidad en la descarga de las bases de datos, y el segundo consta de emitir los gráficos e información que se visualizará en la aplicación. Para esto se creó una función llamada *actualizar_bases_graficos()* que incluye ambos procesos y que además no tiene entradas.

La función puede separarse a su vez en dos partes: la primera parte dedicada a descargar archivos para los costos marginales, generación real y archivos RIO. Esta función repite el mismo procedimiento tres veces: a partir del año y mes introducidos se crea el directorio con el archivo de la base de datos, se revisa si existe en el directorio o no, en caso de existir la función continúa con la siguiente base, en caso contrario lo descarga y se asegura que esté correctamente descargado. Para verificar lo anterior mide su peso en kilobytes con la función sizefile(), que a partir de un archivo entrega su peso en kb. Si pesa menos que 14746 kilobytes lo elimina y espera a que se vuelva a invocar la función para volver a descargarlo.

La segunda parte de la función repite el proceso para emitir los gráficos de las cuatro bases. En primer lugar verifica que el archivo esté descargado, en caso contrario el algoritmo pasa a la siguiente base. Luego comprueba, si corresponde, la existencia del listado de barras o subestaciones (que se menciona más adelante), si este existe verifica si los gráficos asociados a cada elemento de la lista estén generados, si no lo existen los genera invocando las funciones correspondientes. En caso de que la base no tenga un listado asociado, el algoritmo analiza si existe o no el gráfico que emite la función de análisis de bases de datos de la base correspondiente, si no existen los gráficos la función los emite. Esta función se incluye en la sección de Anexos Código 7.5.

Para ejecutar la función anterior, se requieren dos condiciones: la primera consiste en agendar la función, lo que se realiza gracias a la función *schedule*. Esta función ejecutará cada 10 horas la función a partir del momento en que el script corra. La función se muestra en el código 5.22.

```
schedule.every(10).hours.do(actualizar_bases_graficos)
while True:
```

[14]Suma todos los valores de una columna.
[15]Agrega valores al final de una lista.

```
schedule.run_pending()
time.sleep(1)
```

Código 5.22: Automatización del proceso.

Con el fin de que el lector tenga un acercamiento a la Aplicación CNE, la figura 5.5 muestra la parte superior de la aplicación. La parte inferior fue recortada debido a que solamente es color de fondo sobre el cual se incorporan los gráficos y cuadros de texto.

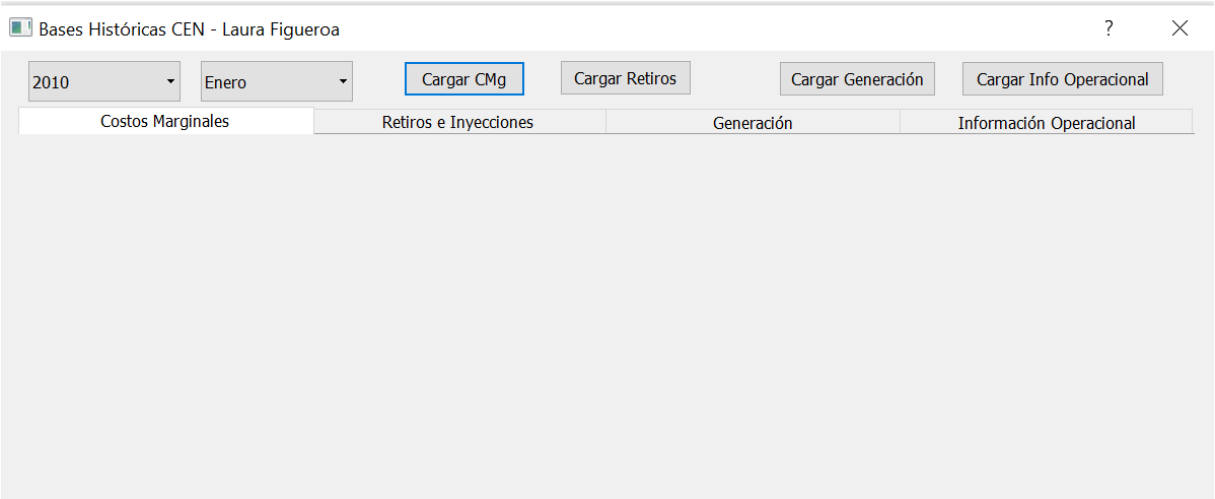


Figura 5.5: Aplicación CEN.

5.2.4. Creación de la aplicación

De manera análoga a la aplicación de las bases de datos de la CNE, se utilizó la plataforma QT Designer para la creación de esta. El código generado por el programa anteriormente mencionado no será mostrado mas si la compilación de los elementos y funciones que hacen posible que el usuario pueda tener interacciones con la aplicación. Las únicas modificaciones que se le hicieron a dicho código tienen relación únicamente con la posición y tamaño de los elementos en la ventana principal.

Previo a comenzar a construir la aplicación, es fundamental importar todas las funciones que son necesarias en el proceso. Posteriormente se definen las partes y acciones que conforman el programa.

En la ventana principal o MainWindow, se incorporan 4 pestañas: pest_CMg, pest_flujos (para los retiros e inyecciones se usa la notación de *flujos* durante toda la aplicación), pest_gen y pest_RIO, cada una de estas tiene como objetivo ser el espacio en la pantalla en el cual la información de las cada base se muestra. En la parte superior del MainWindow se tienen 2 combobox y 4 botones (pushButton_CMg, pushButton_RIO, pushButton_FLUJOS y pushButton_GEN). Los combobox (llamados comboBox_anho y comboBox_mes) dan opción al usuario de cargar un mes específico para visualizar información, una vez seleccionada la fecha el usuario debe apretar uno de los botones para cargar la información que desea ver. Cada botón carga la información de una base diferente, y estos son los únicos que existen en toda la aplicación.

Debido a que existen distintas formas de presentar la información, cada botón ejecuta una acción diferente:

- Al presionar el botón de *Costos marginales*, se activa la función *cargar_cbbx_CMg*, la cual a partir del mes y año que el usuario escogió, rellena el combobox que tiene en su ventana. Este combobox muestra las barras de las cuales se tiene información referente a Costos Marginales.
- El botón de *Información Operacional* carga una lista desplegable que contiene las configuraciones que marcaron cierto tipo de consignas para un mes mediante la función *cargar_cbbx_centrales_consigna*. También carga un gráfico con la función *cargar_graficoRIO2*.
- El botón *Retiros e Inyecciones* carga la lista de subestaciones en 220 kV y 500 kV que tienen disponible información de retiros e inyección de potencia activa. Utiliza la función *cargar_combobox_flujos2*.
- El botón de *Generación Real* carga dos gráficos y un texto con información. Lo anterior lo realiza gracias a la función *cargar_graficogen*.

En la definición inicial de la ventana también vienen incluidas las conexiones entre eventos de elementos dentro de las pestañas y la acción que ejecutan, los cuales se describirán mas adelante. El código 5.23 muestra el extracto que compila los elementos anteriormente descritos.

```
class MainWindowapp(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = MainWindow()
        self.ui.setupUi(self)
        #cargando pestanhas
        self.ui.mdiArea.addSubWindow(self.ui.pest_CMg)
        self.ui.mdiArea.addSubWindow(self.ui.pest_flujos)
        self.ui.mdiArea.addSubWindow(self.ui.pest_gen)
        self.ui.mdiArea.addSubWindow(self.ui.pest_RIO)
        #cargando combobox personalizados
        self.ui.pushButton_CMg.clicked.connect(self.↩
           cargar_cbbx_CMg)
        self.ui.pushButton_RIO.clicked.connect(self.↩
           cargar_cbbx_centrales_consigna)
        self.ui.pushButton_FLUJOS.clicked.connect(self.↩
           cargar_combobox_flujos2)
        #cargando graficos y combobox
        self.ui.pushButton_RIO.clicked.connect(self.↩
           cargar_graficoRIO2)
        self.ui.pushButton_GEN.clicked.connect(self.↩
           cargar_graficogen)
        self.ui.comboBox_barras_CMg.activated.connect(self.↩
           cargar_graficoCMg)
        self.ui.comboBox_centrales_marginando.activated.↩
```

```
            connect(self.cargar_graficoRIO1)
        self.ui.comboBox_centrales_consigna.activated.connect↩
            (self.cargar_tablaRIO)
        self.ui.comboBox_flujos.activated.connect(self.↩
            cargar_graficoflujos)
        self.show()
```

Código 5.23: Definición elementos Aplicación CEN.

A continuación se caracterizan todas las funciones que definen las acciones del usuario, y algunas que entregan la información de las combobox. El análisis se realizara por pestaña.

Costos Marginales

La pestaña de Costos Marginales consta de tres elementos visuales:

- comboBox_barras_CMg: lista desplegable que se carga de manera automática cuando el usuario hace click en el botón *Cargar CMg*.
- grafico_CMg: espacio en el que se inserta un gráfico de barras cuando el usuario selecciona una barra del combobox.
- texto_info_CMg: texto informativo con datos como promedio, mínimo y máximo del costo marginal en una barra.

Para cargar el combobox se necesitan dos funciones: una que emita la lista de barras que estarán disponibles para el usuario, y otra que active la lista en el combobox cuando el usuario desee cargarla. A continuación se explicará la primera función, la cual se llama *lista_barras(mes,anho)*.

Previo a realizar la función cabe destacar que los archivos de costos marginales son diferentes dependiendo del año de emisión:

- Archivos del SING emitidos hasta el 2017 entregan columnas con los valores explícitos del costo marginal.
- Archivos del SIC emitidos hasta el 2017 entregan tablas de valores con el cálculo del costo marginal a partir de los factores de penalización.
- Archivos emitidos desde el año 2018 agrupan el SEN y mantienen el formato de los archivos del SING.

Por la misma razón la función repite su procedimiento en tres ocasiones. A partir del año y mes que se seleccionan como entradas, la función busca el archivo de costos marginales correspondiente para ese mes, lo guarda en un arreglo del tipo dataframe, y crea una lista vacía para depositar las barras correspondientes. Dependiendo del formato de archivo realiza lo siguiente:

- Si el archivo tiene el formato de archivo SIC, la función lee el archivo en dicha pestaña y extrae la columna que incluye todos los nombres de barra, posteriormente aplica la función unique(), la cual devuelve una lista sin valores repetidos, y filtra según voltaje: "si en el nombre de la barra están contenidos 220 o 500, se agrega el nombre de la barra

a la lista vacía".

- Si mantiene el formato del SING: el archivo podría tener dificultades al abrirse, por lo que no se leerá de la forma habitual, en este caso se utiliza la funcion read_html(). Para obtener el arreglo de datos se utiliza la función concat() y entregarle de argumento de lectura de la función anterior. Luego se repite el proceso anterior: se extrae el nombre de las columnas, se agregan los elementos a la lista vacía, se eliminan repetidos y se filtra por nivel de tensión.

Finalmente, la lista con los nombres de las barras para un mes en específico queda guardada en el directorio local en formato .txt.

Como se mencionó anteriormente, la función para cargar la lista desplegable se activa con el evento de apretar el botón "Cargar Costos Marginales". Las primeras instrucciones que realiza la función corresponden a leer el año y mes seleccionados por el usuario, lo cual se realiza con las funciones itemText() y currenIndex(), posteriormente se crea una lista vacía y se define el nombre de la ubicación del archivo con las barras. Luego hay dos caminos para la ejecución del algoritmo, el primer camino corresponde a que efectivamente exista el archivo con las listas, de ser ese el caso, el archivo es leído, y modificado para luego incorporar todos los elementos de él en la lista vacía. De no existir el archivo, se invoca la función *lista_barras()* y con esta llenar la lista vacía. El paso final corresponde a verificar que no hayan elementos en el combobox, para que no se acumulen los que hay en un mes y otro. De haber elementos en el combobox estos son borrados con la función clear(). Finalmente se agregan todos los elementos de la lista con la función addItems().

El codigo 5.24 incluye un extracto con las principales partes de la función, algunas líneas como, por ejemplo, transformar el mes desde "Enero" hasta "1" fueron eliminadas, para facilitar la comprensión del lector.

```
def cargar_cbbx_CMg(self,event):
    anho = int(self.ui.combobox_anho.itemText(self.ui.↩
       combobox_anho.currentIndex()))
    mes_input = self.ui.combobox_mes.itemText(self.ui.↩
       combobox_mes.currentIndex())
    barras=[]
    nombre_txt='Graficos␣CMg/lista_cbbx_'+str(anho)+'_'+str(↩
       mes)+'.txt'
    if os.path.isfile(nombre_txt)==False:
        barras=lista_barras(mes,anho)
    if os.path.isfile(nombre_txt)==True:
        archivo = open(nombre_txt, 'r')
        c=archivo.read()
        lista=c.split(',')
        for i in lista:
            barras.append(i)
        archivo.close()
    option_barras= self.ui.comboBox_barras_CMg.count()
    if option_barras > 1:
        self.ui.comboBox_barras_CMg.clear()
```

```
    self.ui.comboBox_barras_CMg.addItems(barras)
```

Código 5.24: Función cargar_cbbx_CMg().

Una vez seleccionada una opción del combobox, se cargan automáticamente el gráfico y el texto informativo emitidos por la función *promedio_CMg* (vista en la sección anterior). La función es similar a la anterior e inclusive más sencilla. La primera instrucción que recibe el compilador corresponde a borrar los elementos contenidos en *grafico_CMg* (el cual contiene al gráfico) esto con motivo de no superponer imágenes. Posteriormente se leen y guardan el año y mes ya insertados previamente por el usuario (en el próximo extracto aparecen con puntos debido a que se repite el método que en la función anterior). La diferencia con la función anterior está en que ahora se incorpora el parámetro recibido del combobox, el cual se lee de la misma manera que antes, con la funciones itemText() y currenIndex(). En caso de existir el archivo con el gráfico de barras, se carga de manera inmediata, al igual que la información contenida en cadena de texto. De no estar disponible el archivo, ambos elementos se calculan con las funciones datos_CMg() y promedio_CMg(). A continuación se inserta la función reducida en el código 5.25.

```
def cargar_graficoCMg(self,event):
    self.ui.grafico_CMg.clear()
    anho = .......
    mes_input = .......
    barra_input=(self.ui.comboBox_barras_CMg.itemText(self.ui↩
       .comboBox_barras_CMg.currentIndex()))
    archivo_CMg='Graficos␣CMg/'+str(anho)+'_'+str(mes)+'_'+↩
       barra_input+'.png'
    archivo_info='Graficos␣CMg/texto_CMg_'+str(anho)+'_'+str(↩
       mes)+barra_input+'.txt'
    if os.path.isfile(archivo_info)==True:
        c = open(archivo_info, 'r')
        texto_CMg=c.read()
        c.close()
    if os.path.isfile(archivo_CMg)==False:
        df_CMg=datos_CMg(mes,anho,barra_input)
        texto_CMg=promedio_CMg(df_CMg,barra_input,anho,mes)
    self.ui.grafico_CMg.setPixmap(QPixmap(archivo_CMg))
    self.ui.texto_info_CMg.setText(texto_CMg)
```

Código 5.25: Función cargar_graficoCMg().

Archivos RIO

La pestaña correspondiente a la información extraída de los Archivos RIO contiene 5 elementos:

- Una lista desplegable que contiene el listado de barras de las cuales se tiene información de la unidad que está marginando en ellas durante un día. Esta lista siempre contiene las mismas barras y se activa al momento en que la aplicación se abre.

- El gráfico de torta que emite la función *func_tec_marginando()*, este se activa cuando el usuario selecciona una barra del combobox anterior.
- Un gráfico de barras que se emite de manera automática cuando el usuario aprieta el botón *Cargar Info Operacional*. Este gráfico está disponible una vez que la función unidades_marcando() se ejecuta.
- Una tabla de datos que se genera para una unidad generadora y que contiene el tiempo y la cantidad de veces que dicha central estuvo marcando las consignas anteriores.
- Una segunda lista desplegable que permite escoger una unidad generadora que tuvo como consigna una de las siguientes opciones: FS, V, MT, PMT, SS, MPT y EP. Esta lista se genera al ejecutar la función unidades_marcando().

Desde este punto en adelante, todas las funciones generadoras de combobox, información (ya sea en texto o tablas) y gráficos, se basan en la misma estrategia:

1. Recoger año, mes y de ser el caso una barra o subestación desde una combobox.
2. Revisar si el archivo que contiene el gráfico/texto/tabla existe.
3. Si no existe generarlo y si existe cargarlo.
4. Vaciar el combobox o qlabel
5. Añadir elementos.

Cabe destacar que el combobox_centrales_consigna no requiere de una función ya que está definido y fijo para cualquier mes de entrada. A pesar de que todas las funciones sigan el mismo proceso, que los eventos que las activan varían según la función, los cuales son:

Evento	Función
Selección de combobox_centrales_marginando	cargar_graficoRIO1()
Botón "Cargar Info Operacional"	cargar_graficoRIO2()
Selección de combobox_centrales_consigna	cargar_tablaRIO()
Botón "Cargar Info Operacional"	cargar_cbbx_centrales_consigna()

Tabla 5.1: Eventos y funciones para pestaña Información Operacional.

A continuación se adjuntan los cuatro diagramas correspondientes a las funciones que emiten el gráfico de torta, el gráfico de barras, la tabla informativa y la lista desplegable, respectivamente.

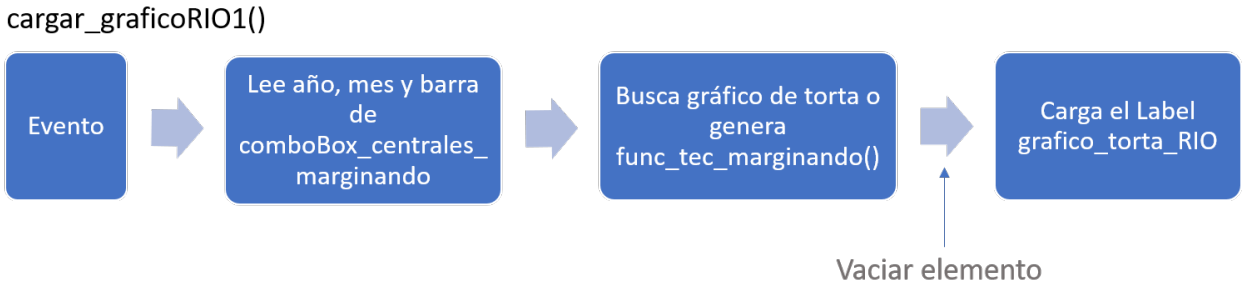


Figura 5.6: Diagrama de flujo para función cargar_graficoRIO1()

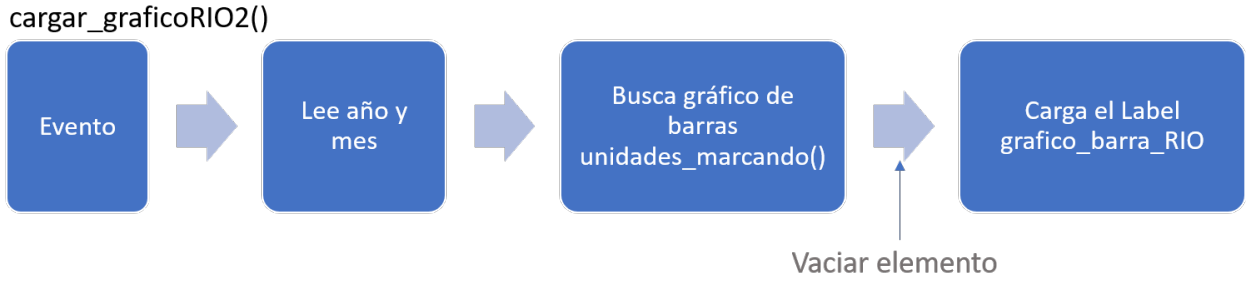


Figura 5.7: Diagrama de flujo para función cargar_graficoRIO2().

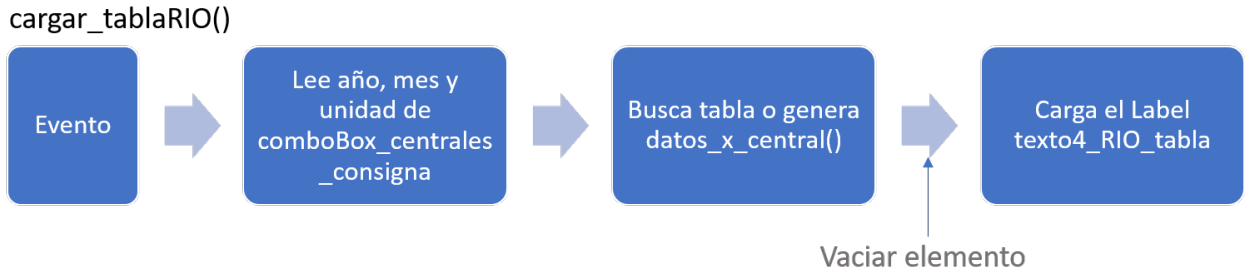


Figura 5.8: Diagrama de flujo para función cargar_tablaRIO().

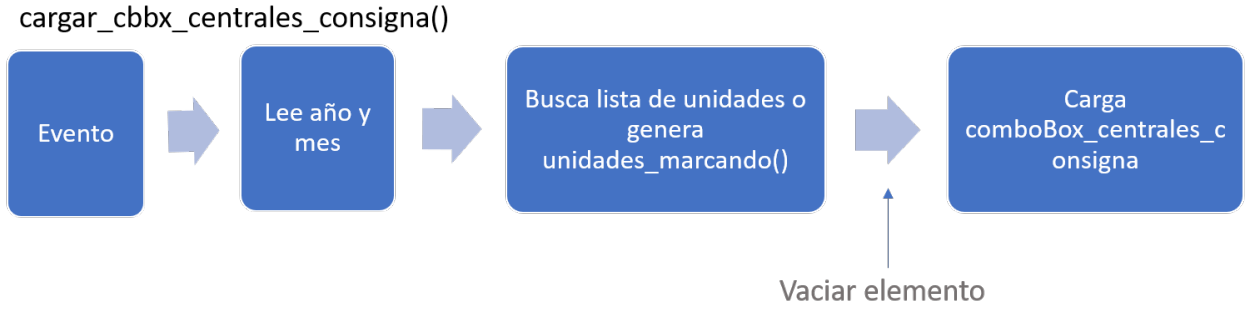


Figura 5.9: Diagrama de flujo para función cargar_cbbx_cenrales_consigna().

Generación Real

La pestaña de Generación Real corresponde a la pestaña más sencilla de todas, la razón radica en que la información (2 gráficos y un label con texto) que será visualizada solo requiere de entrada el año y mes que el usuario selecciona en la parte superior de la ventana. Así, el evento de esta función consiste en presionar el botón *Cargar Generación*. La información a desplegar en esta pestaña está descrita en la sección anterior.

Esta visualización se activa cuando el usuario presiona el botón superior *Cargar Generación*. En caso de no estar previamente generados los gráficos, se invocan las funciones *datos_gen()* y *gen_pref_user()*, en donde la primera sirve de argumento para la segunda.

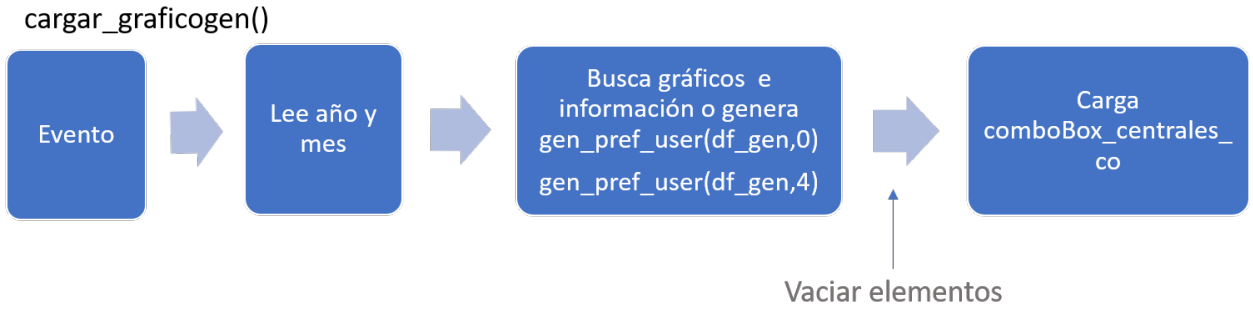


Figura 5.10: Diagrama de flujo para función cargar_graficogen().

Archivos de Medida

La pestaña correspondiente a los archivos de medida se denomina *Retiros e Inyecciones*, la cual incluye tres elementos en ella: el primero corresponde a una lista desplegable que contiene todas las subestaciones en 220 kV y 500 kV existentes para un mes de interés. El segundo y tercer elemento corresponden al gráfico emitido por el análisis de las bases y el texto informativo.

Los archivos de medida por su parte, solamente están disponibles desde el 2018 en la página del coordinador, por lo que esta pestaña no ejecutará ninguna acción si la fecha de entrada es anterior a esa.

La pestaña funciona de la siguiente manera: el usuario selecciona un año y mes de entrada y aprieta el botón *Cargar Retiros*, al momento en que se presiona, se carga la lista desplegable, la cual es emitida con la función *lista_subestaciones*, cuyas entradas son año y mes. A partir de las entradas, la función busca el directorio y lee todas las carpetas de archivos ordenadas desde la A a la Z, abre cada carpeta y acumula los nombres de todas las subestaciones. Finalmente la lista con los nombres se filtra por tensión y se eliminan los elementos repetidos. Este listado es guardado en un archivo de extensión txt. La función completa se muestra en el código 5.26.

```
def lista_subestaciones(mes,anho):
    lista_SSEE=[]
    for carpeta_SSEE in os.listdir('Archivos de medida/'+str(↩
       anho)+'/'+str(mes)):
```

```
        for archivo_letra in os.listdir('Archivos de medida/'+str(anho)+'/'+str(mes)+'/'+carpeta_SSEE):
            ubicacion='Archivos de medida/'+str(anho)+'/'+str(mes)+'/'+carpeta_SSEE+'/'+archivo_letra
            archivo_medidas = pd.read_excel(ubicacion,encoding='1252',skiprows=6)
            archivo_tensiones = pd.read_excel(ubicacion,encoding='1252',skiprows=3)
            columnas=list(archivo_medidas)
            tensiones=list(archivo_tensiones)
            for i in list(range(6,len(columnas),4)):
                if ('500' in tensiones[i])==True or ('220' in tensiones[i])==True:
                    nombre_barra=columnas[i].split('.')
                    lista_SSEE.append(nombre_barra[0])
        lista_def=pd.unique(lista_SSEE)
    lista_sube=''
    for i in lista_def:
        lista_sube+=i+str(',')
    nombre_txt='Graficos Flujos/lista_cbbx_'+str(anho)+'_'+str(mes)+'.txt'
    archivo = open(nombre_txt, 'w')
    archivo.write(lista_sube)
    archivo.close()
    return lista_def
```

Código 5.26: Función lista_subestacioes().

Una vez que se selecciona una subestación del combobox, se crea el evento y se posiciona el gráfico en la ventana. De esta manera se tienen de eventos: presionar el botón *Cargar Retiros* y seleccionar una subestación. Los diagramas de flujo de esta pestaña son los siguientes:

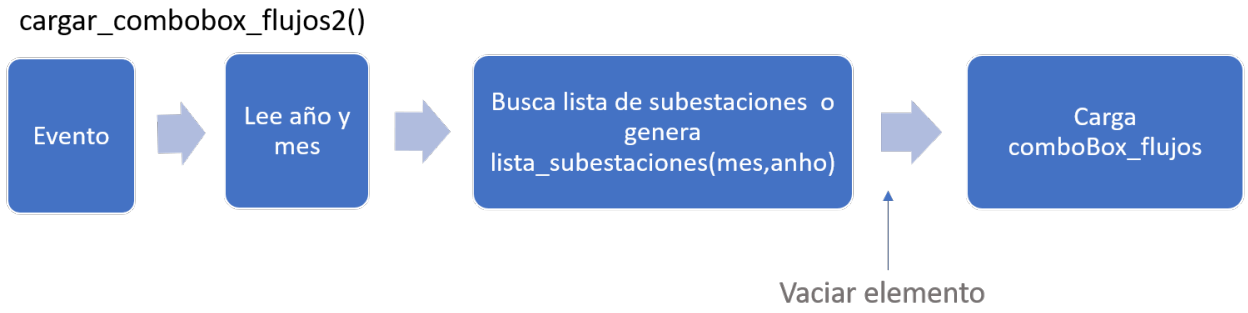


Figura 5.11: Diagrama de flujo para función cargar_combobox_flujos2().

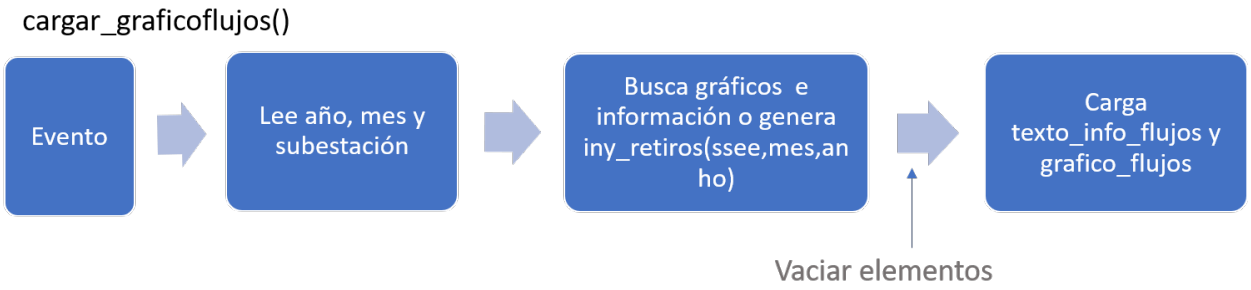


Figura 5.12: Diagrama de flujo para función cargar_graficoflujos().

Capítulo 6

Resultados

El siguiente capítulo se separa en tres apéndices: el primero corresponde a detallar los resultados de la aplicación CNE, el segundo describe la aplicación del CEN, y el tercero describe el uso de los ejecutables y la forma en cómo es posible continuar con la emisión de información en el tiempo.

6.1. Aplicación CNE

La aplicación elaborada a partir de las bases de la CNE resultó completamente exitosa. Al correr el script se visualizan dos botones, tres combobox, y un visor en blanco, tal como muestra la figuera 6.1.

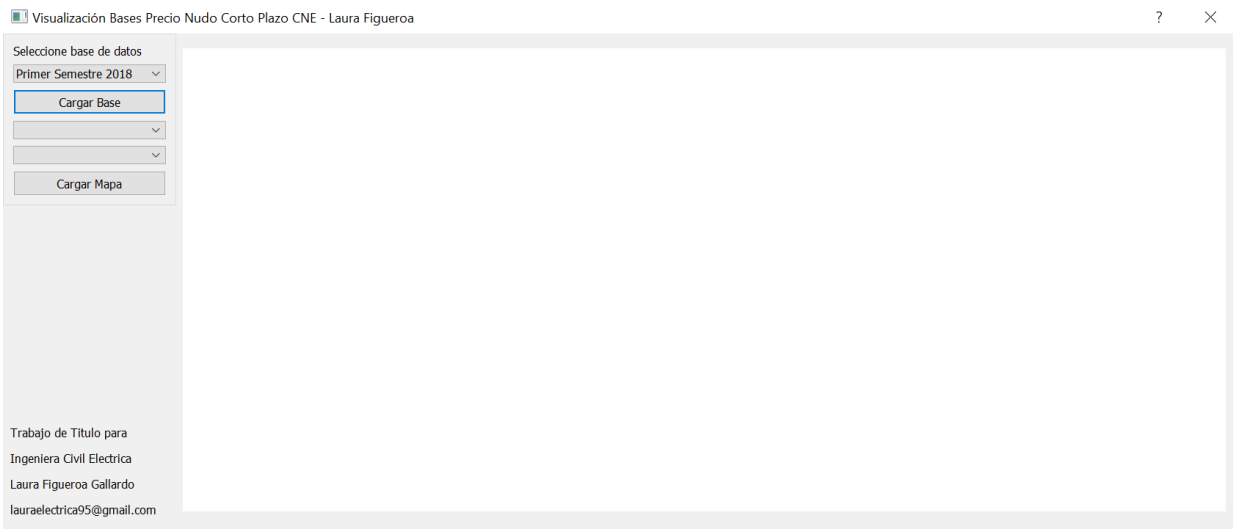


Figura 6.1: Aplicación sin interacción con el usuario.

El usuario puede elegir una base en la primera lista desplegable. Un aspecto negativo de esta lista desplegable, es que en la medida que transcurra el tiempo, y se emitan nuevas bases, el usuario deberá incorporarla a la combobox de manera manual mediante la incorporación

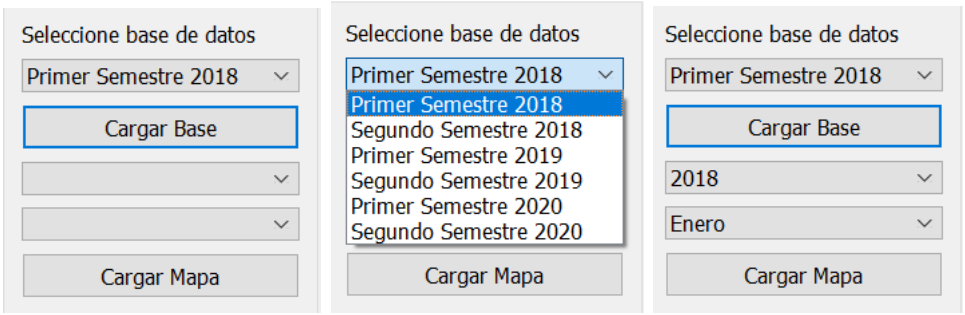
de dos lineas en el script que define los elementos del programa. Las líneas a insertar se muestran en el código 6.1.

```
self.cbx_base.addItem("")
self.cbx_base.setItemText(numero, _translate("programa4", "↩
   Semestre␣y␣a  o "))
```

Código 6.1: Adición de año.

La primera linea debe insertarse en la clase que define la aplicación, llamada *programa4*, y la segunda línea debe insertarse en la función que define los textos de los elementos que están fijos, denominada *retranslateUi*. Cabe destacar que no es posible aumentar la ventana a tamaño completo, esto se debe a que se definieron coordenadas exactas para cada elemento, y al expandir la ventana, las coordenadas se descuadran y el formato original se pierde. Este aspecto no aporta al resultado final, es solamente estético.

Actualmente el usuario puede elegir la base sin problema, y en el momento en que presiona el botón cargar base, se cargan las listas desplegables de año y mes, tal como se muestra a continuación en la figura 6.2.



(a) Combobox iniciales. (b) Opciones combobox bases. (c) Combobox cargadas.

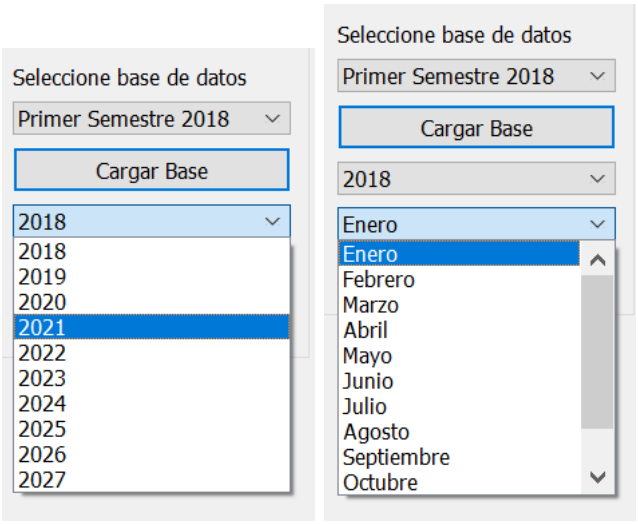
Figura 6.2: Cargando combobox año y mes.

Una vez cargada la base es posible seleccionar un año y fecha de interés para visualizar una red simulada. La lista desplegable año, muestra 10 años consecutivos a partir del año de emisión de la base. Y la combobox mes, mostrará todos los meses del año, tal como se visualiza en la figura 6.3.

Luego de que el usuario selecciona una fecha y posteriormente presiona el botón *Cargar Mapa*, se visualiza el mapa de la red correspondiente en el lado derecho de la aplicación, tal como se muestra en la figura 6.4.

Pese a no poder expandir la ventana, el mapa tiene la cualidad de poder acercarse o alejarse de la pantalla, pudiendo hacer acercamientos sobre cualquier barra, porque además es factible arrastrar la imagen. Una mejor visualización del mapa anterior se aprecia en la figura 6.5.

Como se aprecia en la figura 6.5, existen puntos verdes y puntos azules. Los puntos verdes representan las barras del sistema de transmisión cuya tensión es 220 kV, mientras que las



(a) Combobox año. (b) Combobox mes.

Figura 6.3: Listas desplegables año y mes cargadas.

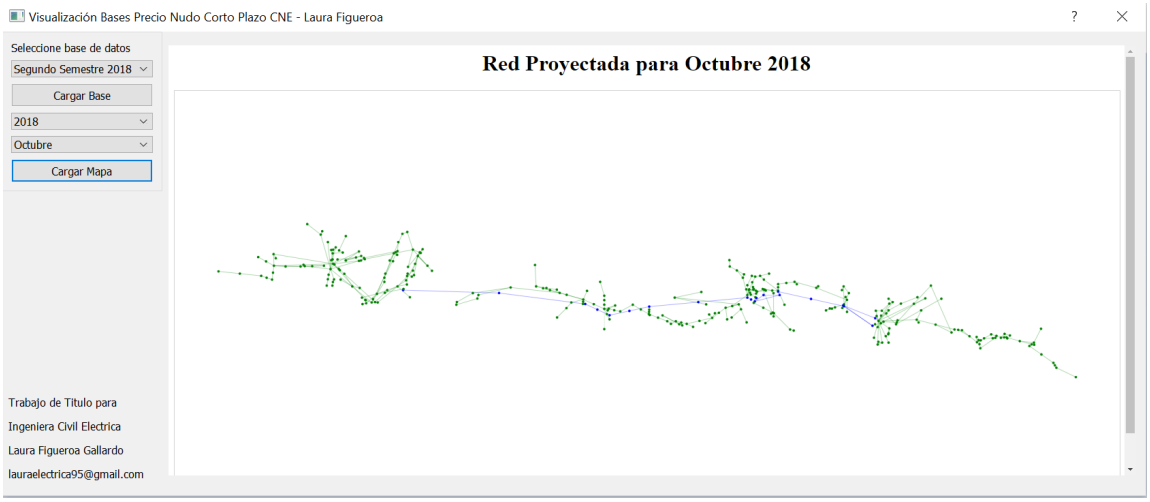


Figura 6.4: Mapa cargado.

azules corresponden a 500 kV. El extremo izquierdo de la imagen posiciona el norte y el extremo derecho, el sur.

Si el usuario posiciona el cursor sobre una barra se visualiza una ventana con información del resultado del análisis de las bases de datos, tal como se muestra en la figura 6.6. Un acercamiento de lo anterior se presenta en la imagen 6.7.

Debe considerarse que existen algunos nodos que quedan sobrepuestos unos con otros, esto se debe al hecho de que no se conocen sus posiciones reales, y se calculan con un algoritmo que arroja como su posición un valor aleatorio calculado a partir de la posición de sus vecinos.

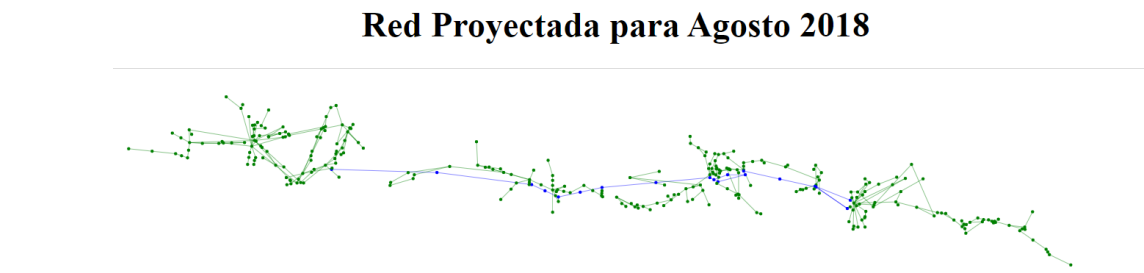


Figura 6.5: Acercamiento a la red.

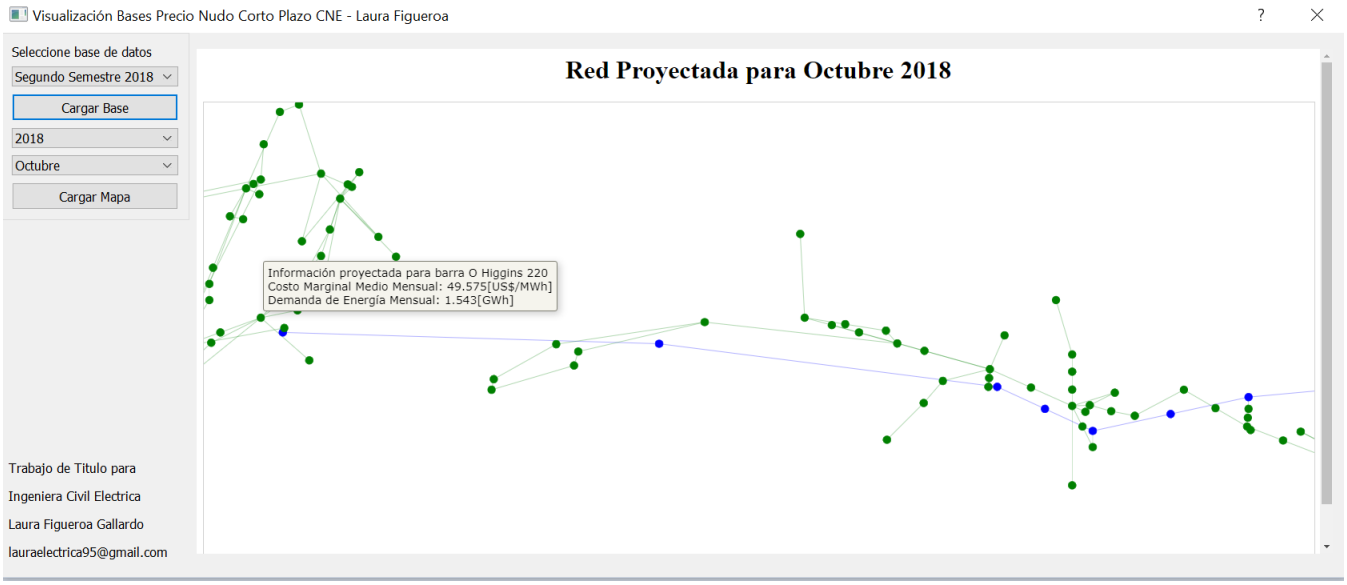


Figura 6.6: Contenido ventana informativa.

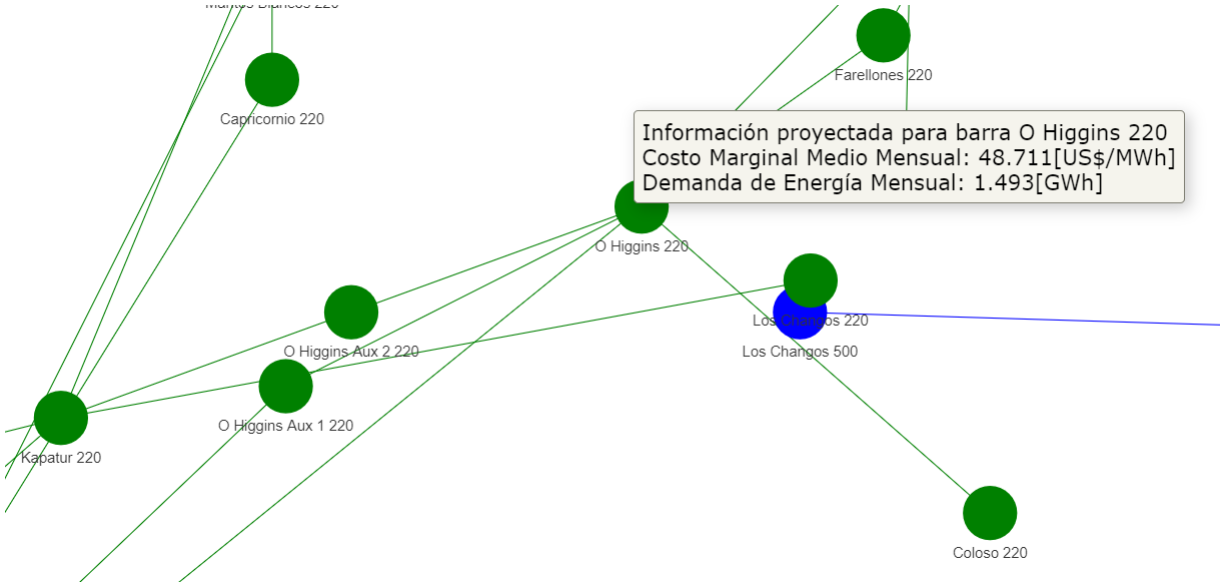


Figura 6.7: Acercamiento al contenido informativo.

6.2. Aplicación CEN

La siguiente sección se dividirá en dos partes: la primera parte consiste en mostrar la información resultante tras el análisis de las bases de datos, ya sea en gráficos o en su formato correspondiente. La segunda parte consiste en mostrar la visualización completa de la aplicación.

6.2.1. Visualización información de bases de datos

Costos Marginales

El resultado tras el análisis de la base de datos de los costos marginales, genera dos archivos: el primero un gráfico de barras y el segundo una nota de texto de extensión txt.

El gráfico de barras contiene para cada día del mes el valor promedio del costo marginal. Se asignó un código de colores para las barras: las rojas representan los días domingo y las azules a los demás días de la semana. También se añadió una linea horizontal paralela al eje X con el valor promedio mensual del costo marginal.

Los gráficos de esta base están disponibles a partir de Enero 2010 y para efectos de esta memoria hasta diciembre 2020, aún así es posible emitirlos por los meses siguientes a la fecha anterior. La visualización del gráfico se aprecia en la figura 6.8.

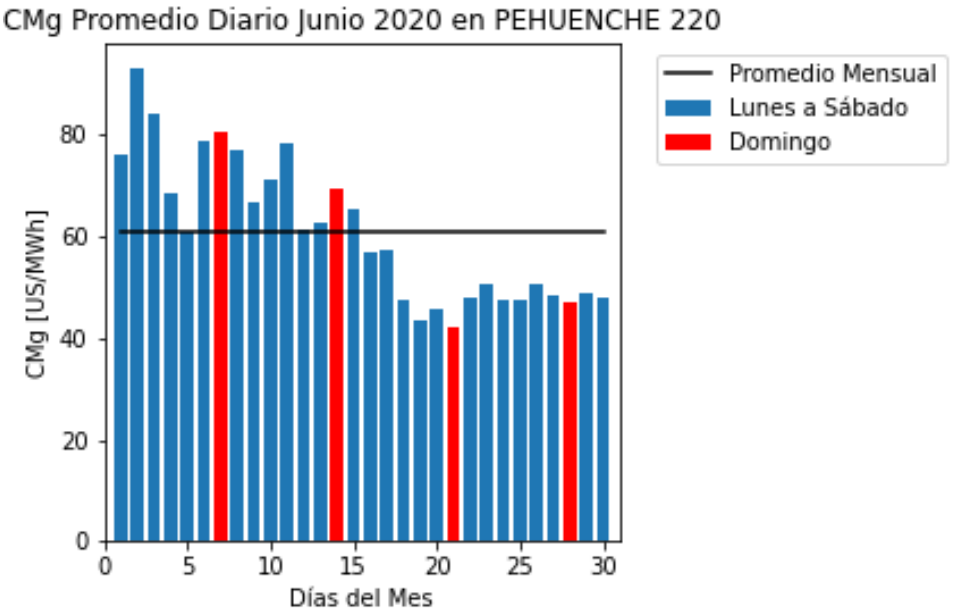


Figura 6.8: Gráfico costos marginales para la barra Pehuenche 220 en Junio 2020.

En el momento que el script de análisis de bases de datos genera el gráfico anterior, se emite un texto que contiene: promedio mensual del costo marginal, máximo y mínimo de este, y cantidad de veces en que el costo marginal es 0 en el mes. El texto generado para la barra del gráfico anterior es el siguiente:

Promedio Mensual:
60.880[US/MWh]
CMg Min:
42.323[US/MWh]
CMg Max:
93.216[US/MWh]
CMg = 0 en el mes:
0

Archivos RIO

A partir del análisis de las bases de datos de los archivos RIO se han obtenido dos gráficos y una tabla con información. El primer gráfico corresponde a un gráfico de torta que contiene el porcentaje en función del tiempo, de las veces que marginaron las tecnologías hidráulica y térmica, incluyendo también la categoría no hay información para los días del mes en que efectivamente no hay información al respecto. El segundo gráfico corresponde a las veces en el mes que el total de unidades de generación (operativas en el mes) incurrió en una de las siguientes consignas: vertimiento (V), fuera de servicio (FS), mínimo técnico (MT), en prueba (EP), proceso a mínimo técnico (PMT), mínimo técnico provisorio (MTP) y seguridad del sistema (SS). La tabla por su parte, muestra la cantidad de veces y tiempo en que una unidad en específico incurrió durante el mes en las consignas anteriormente descritas, en caso de no haber incurrido en alguna de esas consignas, no se muestran datos.

El gráfico de torta anteriormente descrito está disponible únicamente para las barras: Crucero 220, Diego de Almagro 220, Cardones 220, Pan de Azúcar 220, Quillota 220, Alto Jahuel 220, Charrúa 220 y Puerto Montt 220. El código de colores del gráfico es el siguiente: amarillo representa que no hay información, azul para las unidades hidráulicas y gris para las unidades térmicas. En la imagen 6.9 se muestra el gráfico de Puerto Montt en Enero 2020.

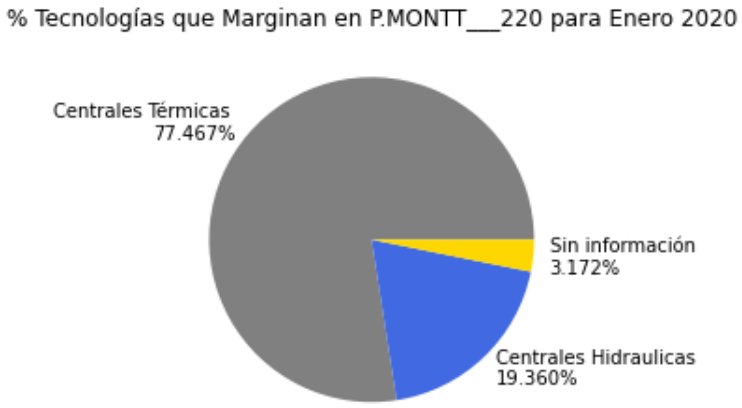


Figura 6.9: Gráfico % tecnologías marginando en Puerto Montt 220 para Enero 2020.

Como se mencionó anteriormente, el gráfico de barras contiene la ocurrencia mensual del conjunto de todas las unidades generadoras que incurrieron en una de las consignas

anteriormente descritas, es decir, para un mes se emite únicamente uno de estos gráficos. En este caso el código de colores es únicamente de barras amarillas para la ocurrencia mensual de cada consigna. La figura 6.10 muestra el gráfico para Enero 2020.

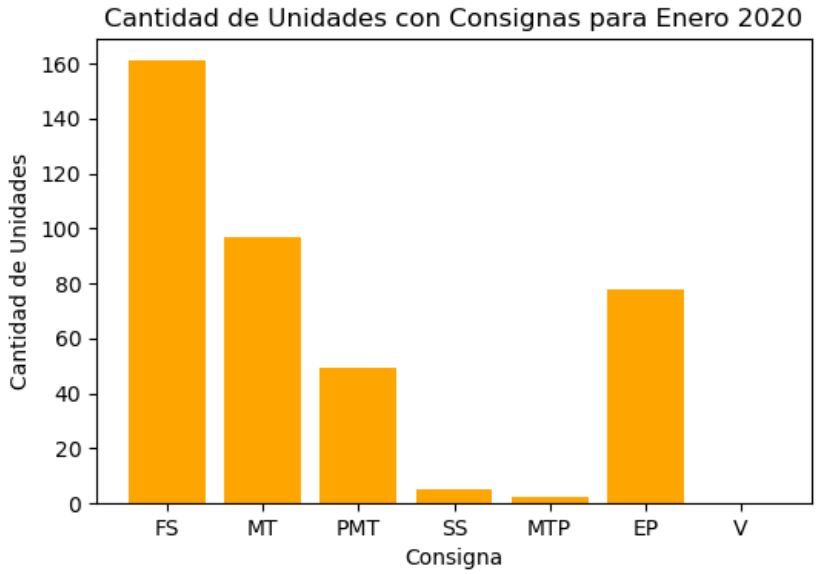


Figura 6.10: Gráfico tecnologías marginando para Puerto Montt 220 en Enero 2020.

Finalmente la tabla con las recurrencias se muestra en la figura 6.11 para la unidad Angostura 2 en Enero 2020.

Consigna	Ocurrencia	Tiempo acumulado
FS	15 [veces/mes]	
MT	11 [veces/mes]	4 días, 6:00:00 horas
PMT	1 [veces/mes]	0:59:59 horas

Figura 6.11: Tabla de consignas y ocurrencias en Angostura-2 para Enero 2020.

Generación Real

El resultado del análisis de esta base arrojó como resultado dos gráficos y un cuadro de texto. El primer gráfico es uno de barras en que cada barra representa un día del mes y la generación total del día. Cada barra contiene segmentos de colores, los cuales representan el aporte de generación eléctrica asociada al tipo de fuente energética. El código de colores de dichos segmentos es: marrón para fuentes térmicas, azul para hidráulica de embalse, verde para hidráulica de pasada, morado para fuentes eólicas y finalmente amarillo para fuentes solares. El segundo gráfico es de torta y contiene el aporte en porcentaje de cada fuente de energía a la generación mensual a nivel nacional. El código de colores es el mismo que para el gráfico anterior. Las figuras 6.12 y 6.13 muestran los gráficos de barras y torta, respectivamente.

El texto generado tras el análisis de esta base contiene la generación total mensual a nivel nacional, y el promedio diario de esta. el texto para Enero 2020 es el siguiente:

Generación Total Mes:
6800.946[GWh]
Generación Promedio Diaria:
219.385[GWh]

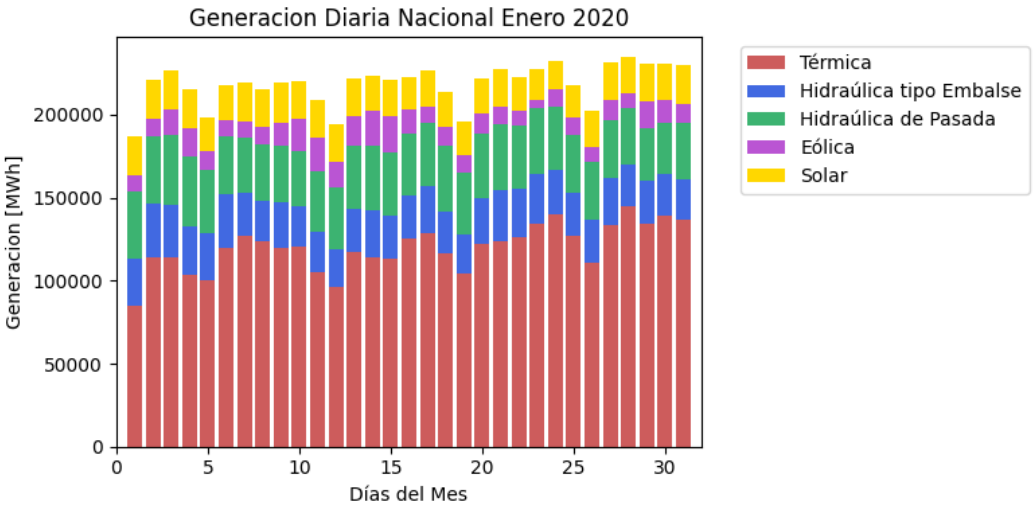


Figura 6.12: Gráfico de barras Generación Real para Enero 2020.

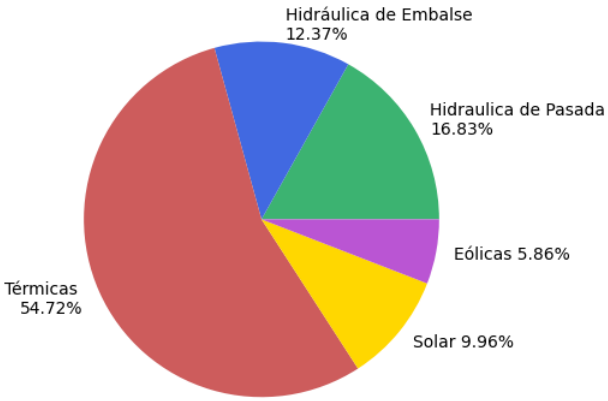


Figura 6.13: Gráfico de torta Generación Real para Enero 2020.

Archivos de Medida

Para esta base el resultado del análisis genera un gráfico de barras mensual y un cuadro de texto. El gráfico muestra los retiros (en azul) e inyecciones (en amarillo) en potencia activa de cada medidor de una subestación. Mientras que el cuadro de texto muestra el total mensual de retiros e inyecciones de la subestación escogida por el usuario. A continuación se muestra el cuadro de texto para la Subestación Atacama en Enero 2020 y luego el grafico en la figura 6.14.

Retiros Total Mes: 5136.663 (MWh)
Inyeccion Total Mes:3365.530 (MWh)

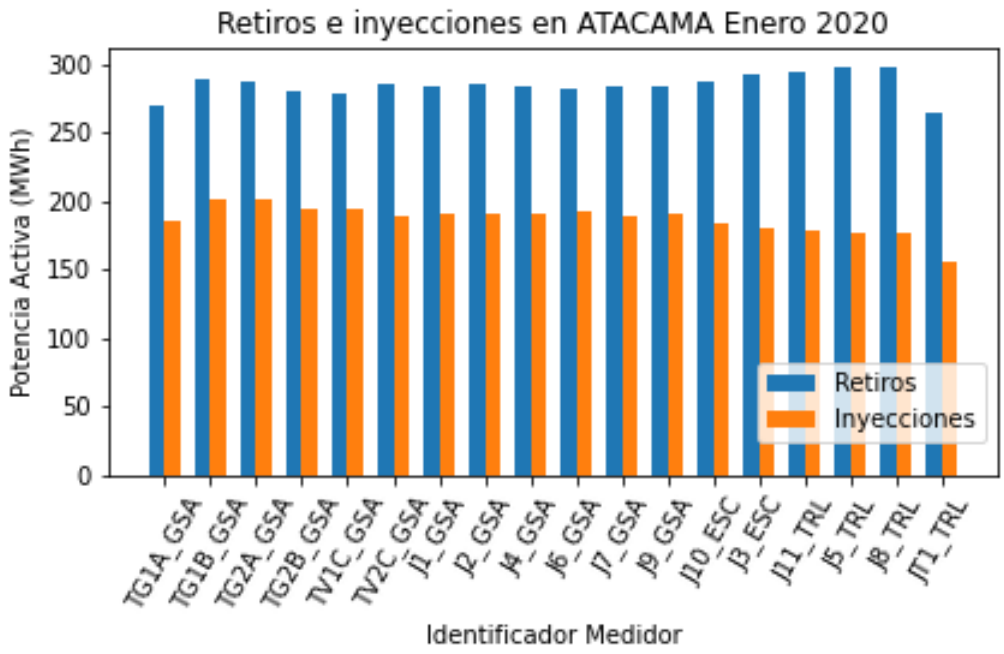


Figura 6.14: Gráfico Retiros e Inyecciones de Subestación Atacama para Enero 2020.

6.2.2. Aplicación CEN

El resultado de la aplicación consiste en un panel superior y varias pestañas debajo de este. El panel cuenta con dos listas desplegables, la primera para seleccionar un mes y la segunda para elegir un año, estos valores representan la fecha de entrada de la cual se espera obtener una visualización de información por parte del usuario. El panel superior incluye además cuatro botones: *Cargar CMg*, *Cargar Retiros*, *Cargar Generación* y *Cargar Info Operacional*. Cada botón realiza una acción distinta y asociada a una pestaña de la aplicación. Cabe mencionar que debido a que algunas bases de datos no estaban disponibles desde el 2010, algunos botones no provocarán modificaciones en las pestañas informativas si la fecha ingresada es anterior a la fecha en que se emiten las bases de datos.

La visualización inicial de la aplicación completa se muestra en la figura 6.15. Por default al abrir la aplicación se mostrará una de las pestañas, y además, siempre y cuando no se esté generando contenido para una pestaña, el primer botón quedará enmarcado en contorno celeste.

No es posible ampliar la ventana de la aplicación a pantalla completa, ya que se perderían las posiciones de los elementos que hay en cada pestaña. En la esquina superior derecha de la aplicación aparece un signo de interrogación, el cual no tiene ninguna utilidad. El programa de diseño QtDesigner la añade al diseño de la ventana, pero no es posible eliminarlo.

Las imágenes de la figura 6.16 muestran que las listas desplegables de año y mes del panel superior están completamente funcionales, sin embargo y como se mencionó anteriormente, algunas fechas no tienen información asociada, las fechas son:

- Enero 2010 hasta Diciembre 2017: no existen gráficos o datos asociados a esta fecha

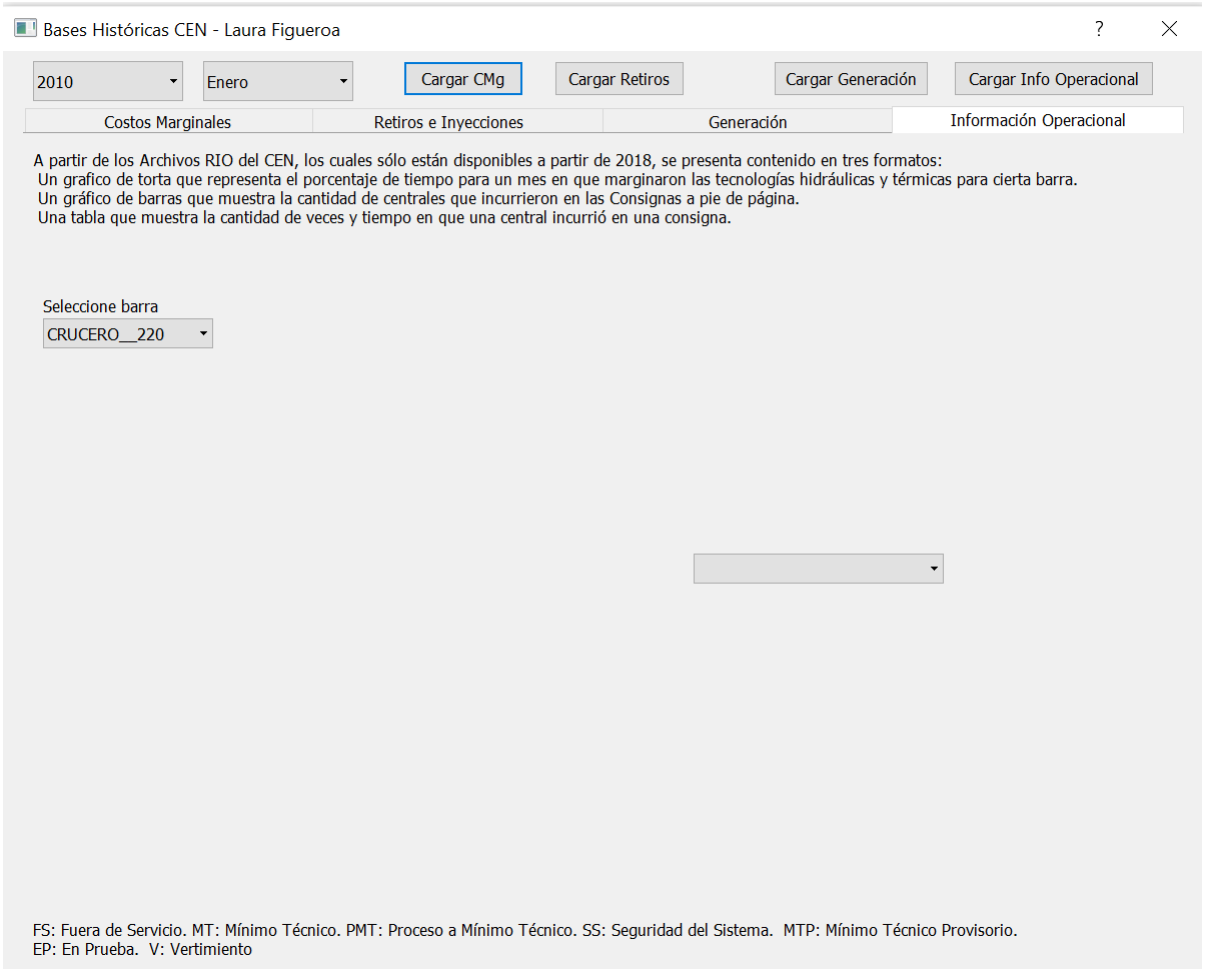
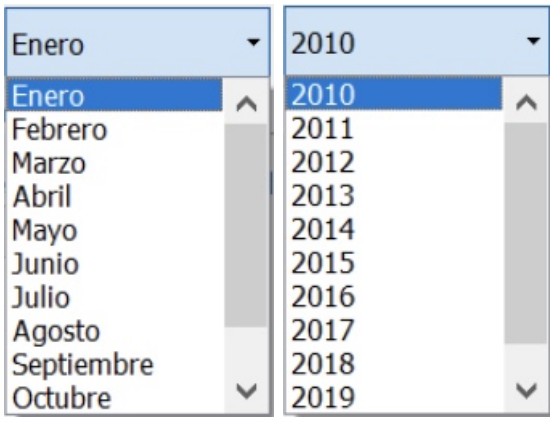


Figura 6.15: Visualización inicial aplicación CEN.

para la base de los Archivos RIO ni para la base de los Archivos de Medida.

Debido a lo anterior, al presionar un botón de estas bases habiendo ingresado una fecha del tramo anterior, no realizará ninguna acción ni error en el programa.



(a) Combobox año. (b) Combobox mes.

Figura 6.16: Listas desplegables año y mes aplicación CEN.

Esta aplicación tendrá disponibles en su visualización todos los gráficos emitidos hasta diciembre 2020 y es completamente funcional. Para una mejor presentación de resultados, a continuación se mostrarán las pestañas por separado.

Pestaña Costos Marginales

Previo a presionar un botón, la pestaña de Costos Marginales está compuesta por dos elementos ubicados en la parte superior: el primero corresponde a un encabezado de texto que informa el contenido de la pestaña, y el segundo a una lista desplegable que inicialmente esta vacía. Estos elementos se aprecian en la figura 6.17.

En el momento en que el usuario aprieta el botón *Cargar CMg*, se actualiza la lista desplegable de la pestaña con todas las barras de las cuales se tiene información de los costos marginales para ese mes. La lista desplegable se ve como en la figura 6.18. La notación en que se encuentran las barras en la lista desplegable corresponde a la misma que utiliza el CEN en los archivos de Costos Marginales.

Una vez que el usuario escoge una barra, se carga de manera automática el gráfico mostrado en la sección anterior, y un pequeño párrafo informativo. Lo anterior se muestra en la figura 6.19.

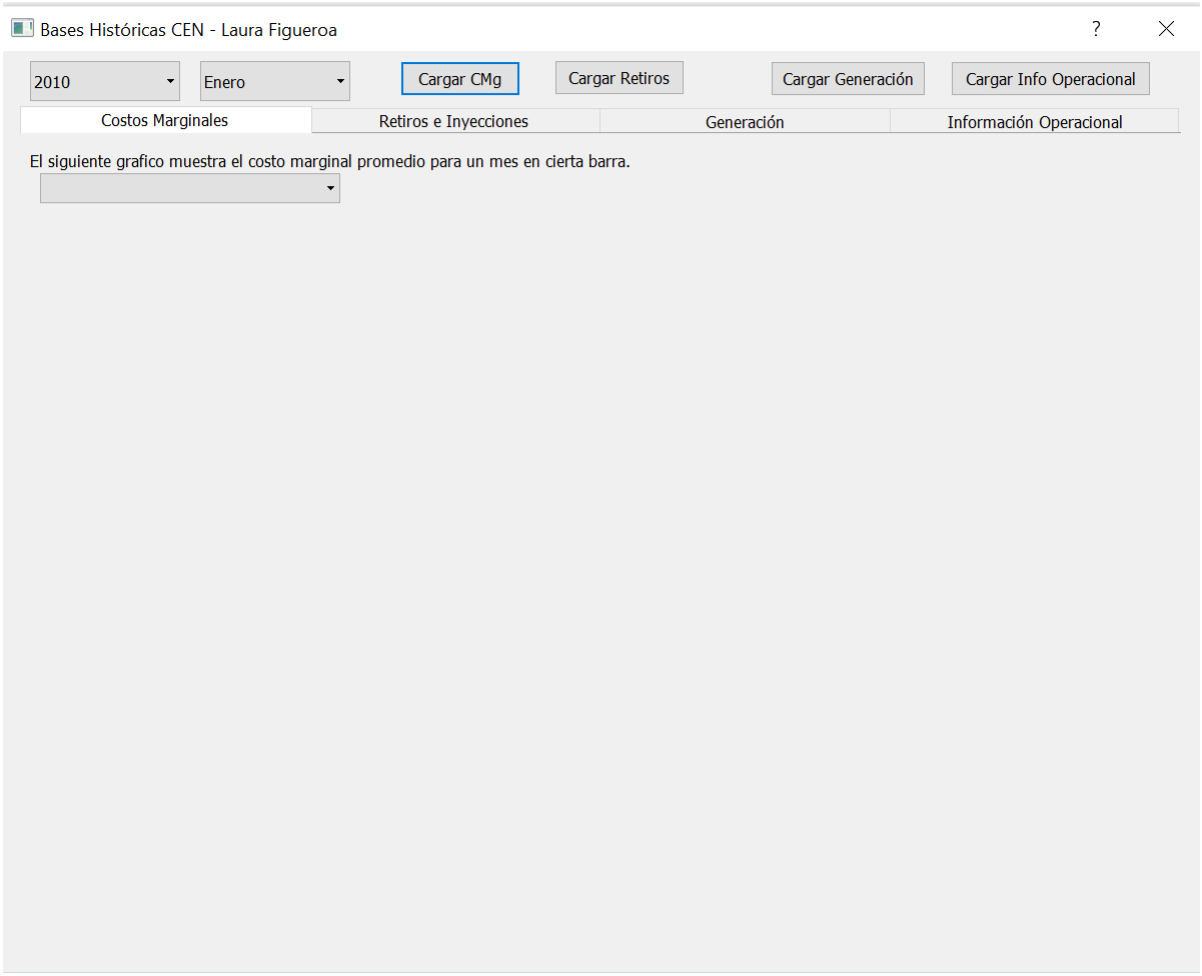


Figura 6.17: Visualización inicial pestaña Costos Marginales.

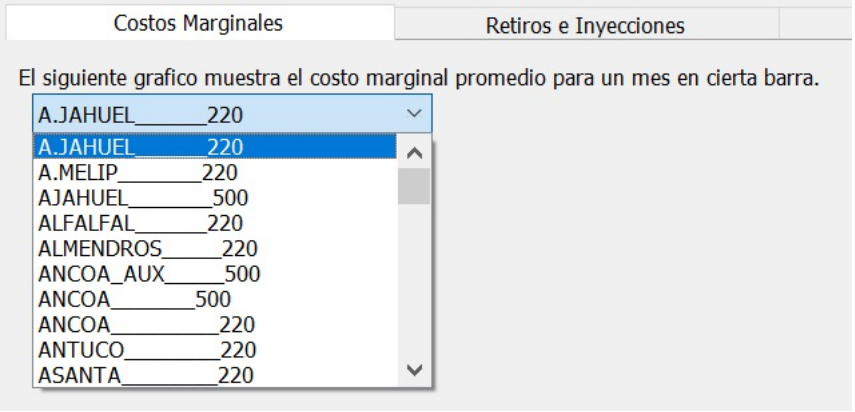


Figura 6.18: Visualización combobox pestaña Costos Marginales.

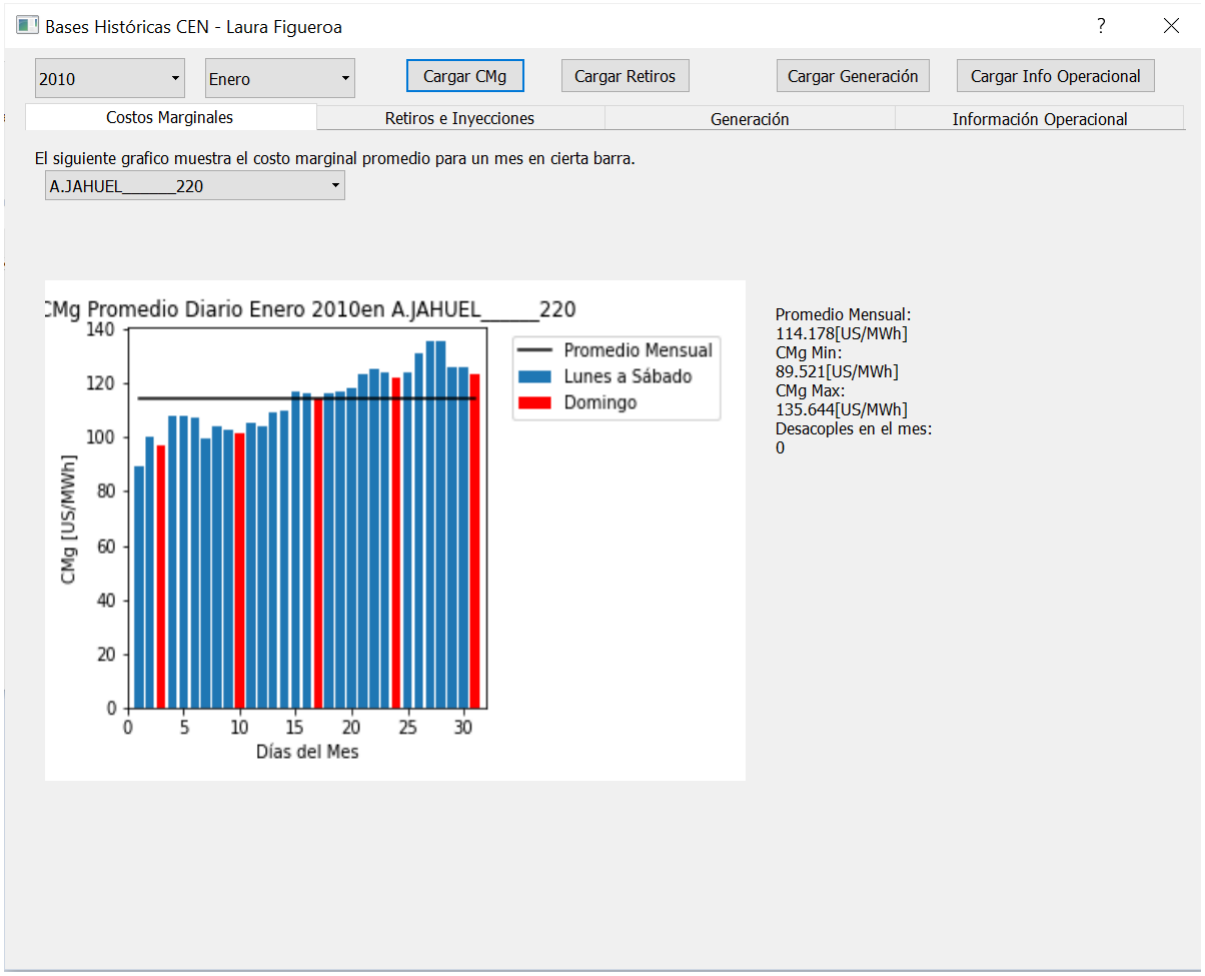


Figura 6.19: Pestaña CMg.

Pestaña Información Operacional

Al abrir esta pestaña sin haber presionado el botón *Cargar Info Operacional*, se visualizan cuatro elementos en la pestaña: un encabezado de texto que describe todos los componentes que estarán disponibles en la ventana, una lista desplegable que contiene una cantidad fija y limitada de barras (a partir de ellas se generan gráficos de torta), una segunda lista desplegable que carga sus opciones al apretar el botón del panel superior (muestra las unidades generadoras que incurrieron en una consigna específica para el mes de selección) y, un encabezado de texto en la parte inferior de la pestaña que contiene un pequeño glosario de acrónimos que será útil cuando se carguen los gráficos. Todos los elementos anteriores se visualizan en la figura 6.20.

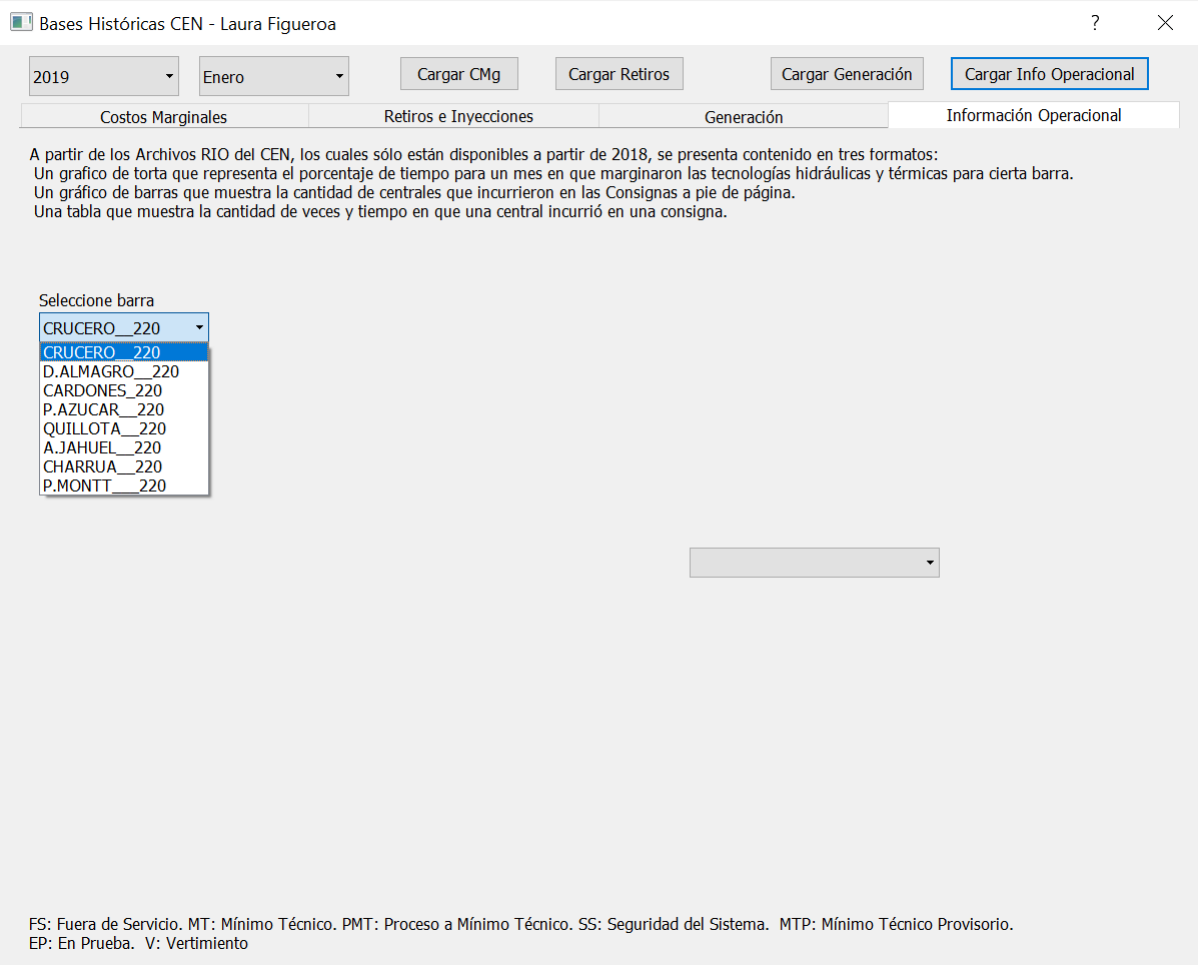
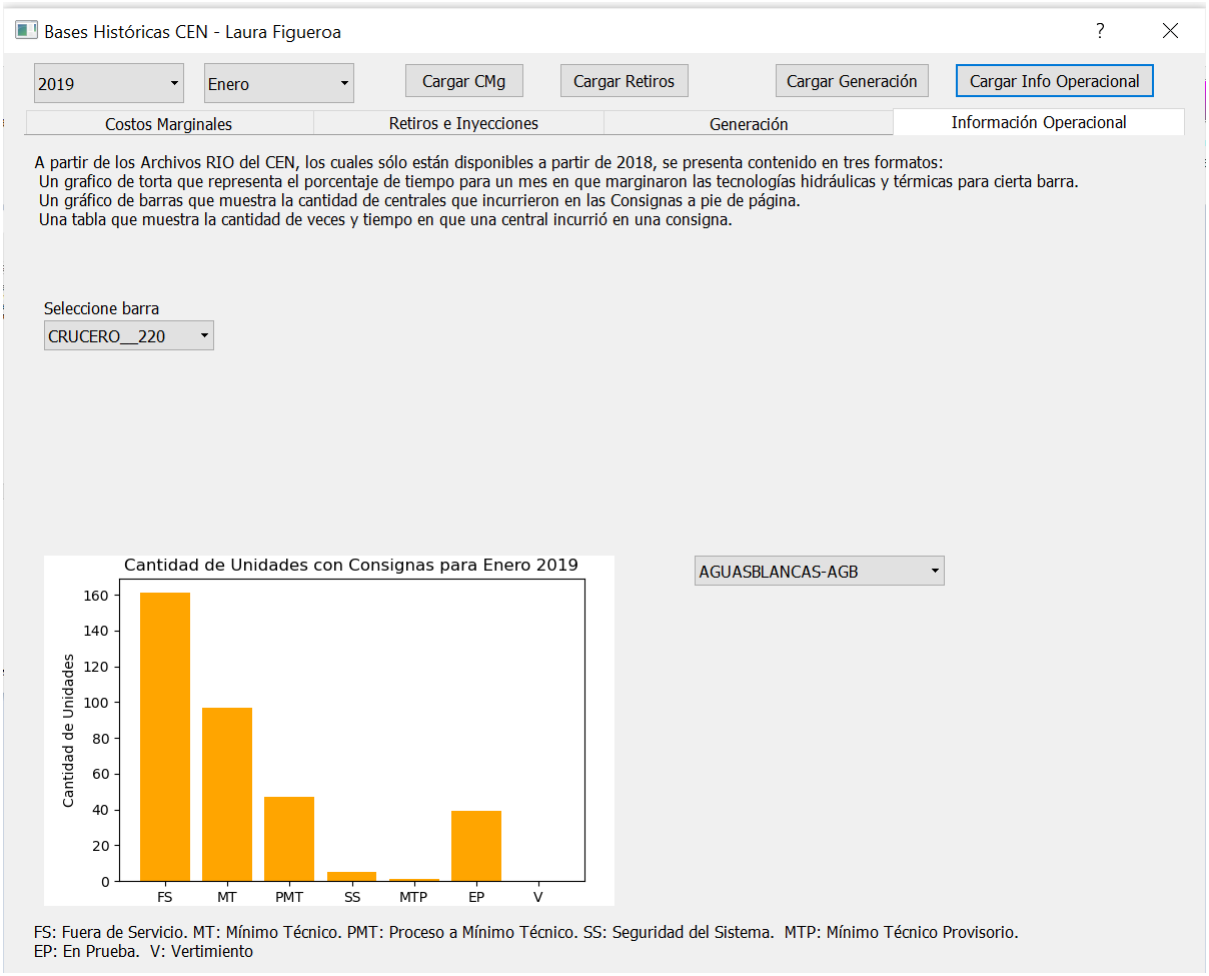
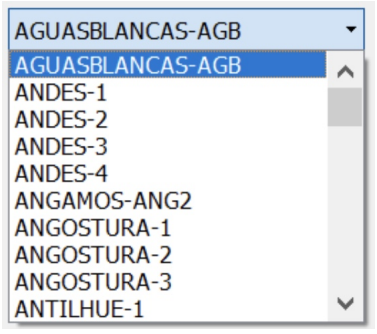


Figura 6.20: Pestaña inicial Información Operacional.

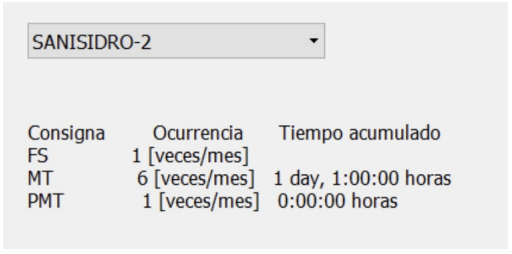
Esta pestaña está completamente funcional a partir de Enero 2018 y, cuando el usuario selecciona una fecha correcta y luego presiona el botón *Cargar Info Operacional* se carga un gráfico de barras en la esquina inferior izquierda y la lista desplegable que contiene las unidades generadoras que marcaron consigna en el mes. Al presionar una opción de lista desplegable, se carga de manera automática la tabla informativa de ocurrencias de consignas para una unidad generadora. La tabla se posiciona debajo de la lista desplegable. Todo lo anterior se muestra en la figura 6.21.



(a) Grafico de barras.



(b) Combobox unidades generadoras.



(c) Tabla informativa.

Figura 6.21: Elementos pestaña Info Operacional.

Para visualizar el gráfico de torta que muestra el porcentaje de tecnologías de generación que marginan sobre una barra, en donde solamente se debe elegir una opción de la lista desplegable superior (previo a la elección de una fecha en el rango admitido). La posición del gráfico es al costado de la lista desplegable y se muestra en la figura 6.22 junto con el conjunto completo de todos los elementos seleccionados para esta pestaña.

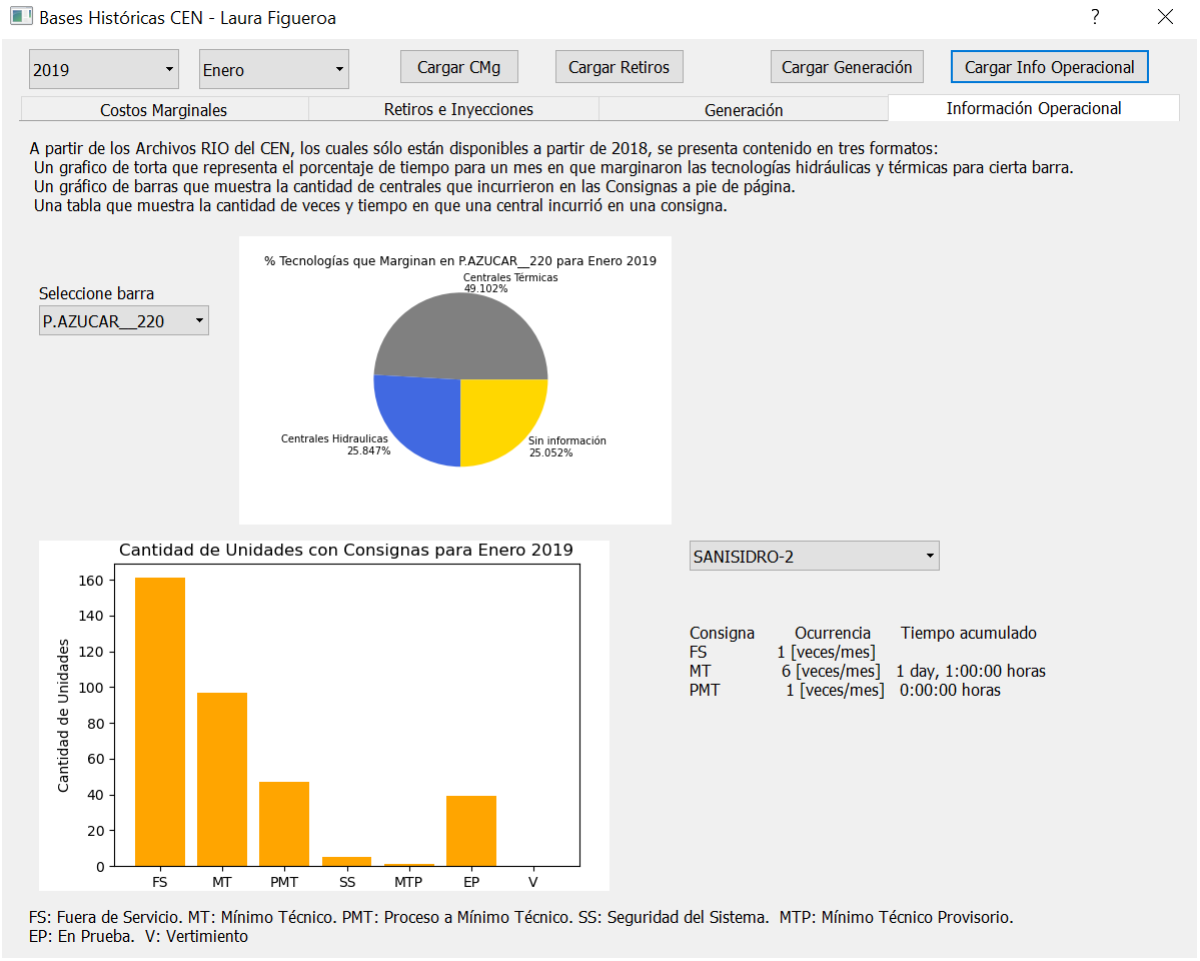


Figura 6.22: Pestaña Información Operacional.

Pestaña Generación Real

Esta pestaña resulta ser la más sencilla de la aplicación debido a que visualiza información a nivel nacional y no asociada a una barra, subestación o unidad generadora. Al abrir esta pestaña sin haber presionado el botón *Cargar Generación* aparece únicamente un encabezado de texto. Una vez se aprieta el botón se cargan dos gráficos y un pequeño recuadro con información. Lo anterior se aprecia en la figura 6.23.

Se aprecia que el cuadro de texto está desplazado hacia la derecha, lo anterior ocurre debido a que Python emite los gráficos de torta con distintas dimensiones, con lo que al posicionarlo hacia la derecha se asegura que no quede tapado por la imagen.

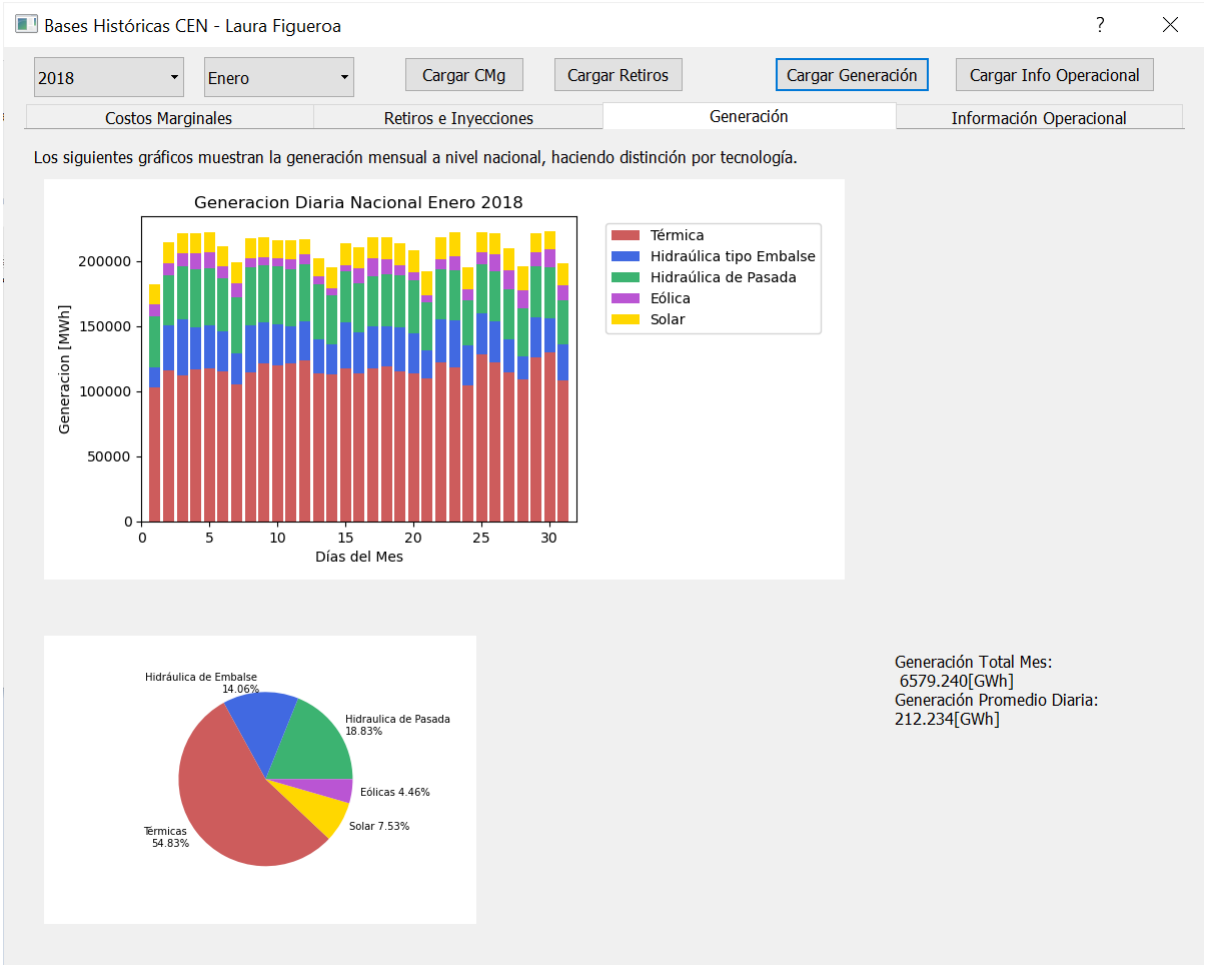


Figura 6.23: Pestaña Información Operacional.

Pestaña Retiros e Inyecciones

Esta pestaña está compuesta inicialmente por dos elementos: un encabezado de texto explicativo en la parte superior, y una lista desplegable bajo el texto anterior. Inicialmente la lista no está cargada, por lo que es necesario que el usuario seleccione una fecha correcta y luego presione el botón *Cargar Retiros*. La lista se aprecia en la figura 6.24.

En el momento en que el usuario selecciona una opción de la lista desplegable de subestaciones, se carga en la ventana un gráfico de barras y un cuadro de texto informativo. Lo anterior se muestra en la figura 6.25.

6.3. Ejecutables e Instrucciones para la continuidad de descarga

Para cada aplicación fue posible realizar un ejecutable, el cual corresponde a un acceso directo desde cualquier computador para el uso de cada una de las plataformas (CEN y CNE) sin la necesidad de tener que abrir un script en Python. Pese a existir el ejecutable, este tiene ciertas limitaciones y condiciones de uso, las cuales se describen a continuación:

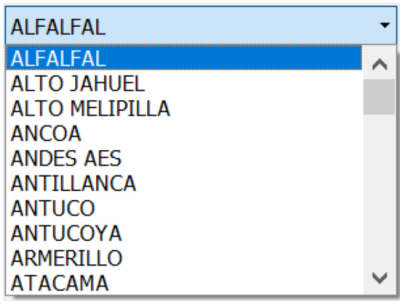


Figura 6.24: Lista desplegable pestaña Retiros e Inyecciones.

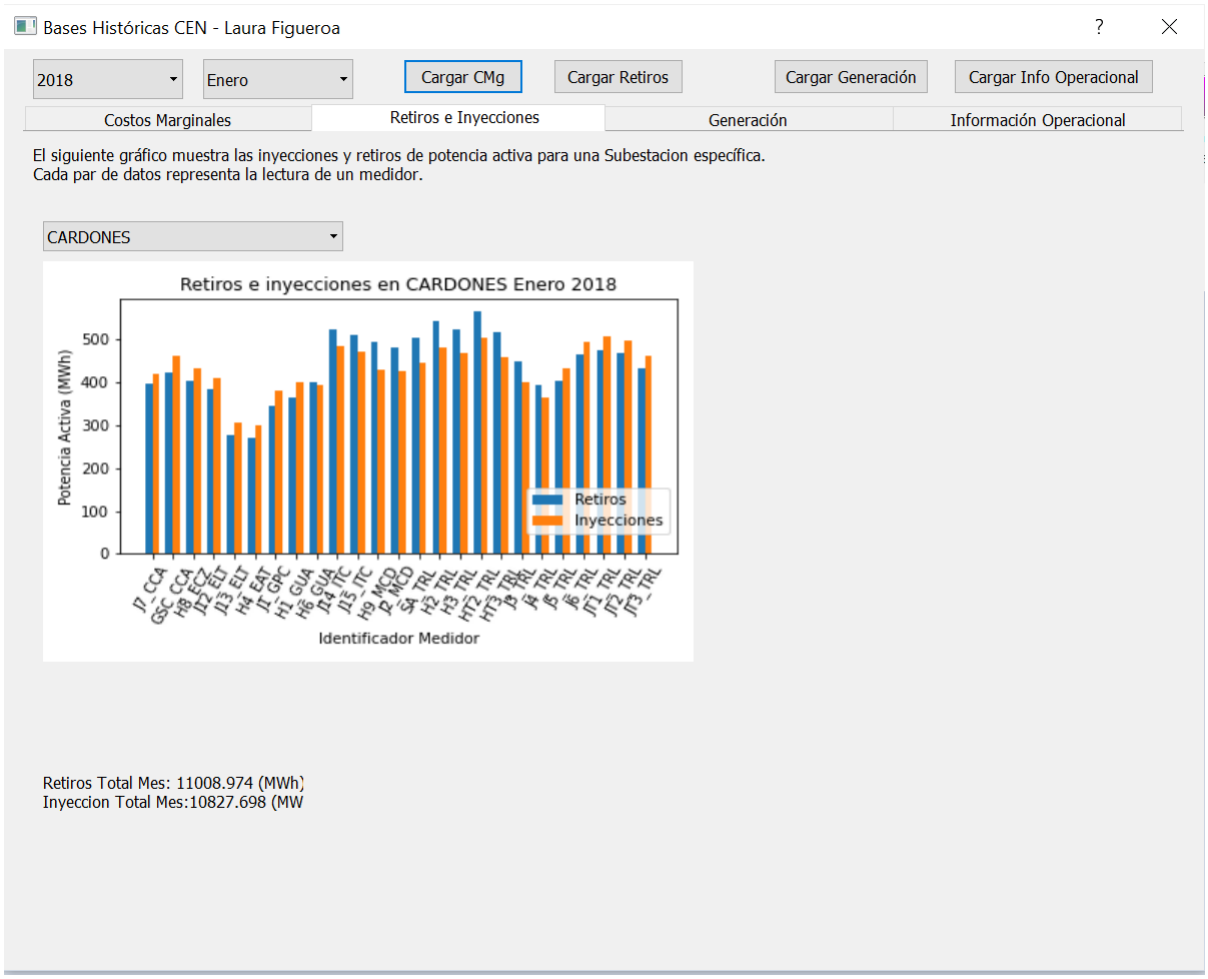


Figura 6.25: Pestaña Retiros e Inyecciones.

- El ejecutable no permitirá el correcto uso de cada aplicación si el computador que lo ejecute no tiene descargadas las carpetas con los archivos que la aplicación le muestre al usuario. La lista de carpetas con archivos corresponde a los mapas generados (CNE), gráficos de costos marginales (CEN), gráficos de generación real (CEN), gráficos de retiros e inyecciones (CEN) y gráficos de costos marginales. En caso de abrir el programa y no tener las carpetas descargadas, al dar instrucciones de mostrar un gráfico que no esté descargado la aplicación se cerrará de manera automática.
- Las carpetas deben estar guardadas en el mismo directorio que el ejecutable y con los siguientes nombres: Mapas Generados, Graficos Gen, Graficos CMg, Graficos RIO y Graficos Flujos (para los retiros e inyecciones). De haber una mínima variación en los nombres o ubicación de estas carpetas, el programa no funcionará según lo esperado.
- El ejecutable no realiza la descarga automática de los archivos de las bases de datos.

Considerando que el ejecutable no es capaz de realizar Web Scraping, si el usuario desea tener la posibilidad de emitir y ver gráficos informativos para fechas futuras es requisito mantener en ejecución en un entorno de desarrollo (Spyder, Pycharm u otro) el script que realiza la descarga de datos y la emisión de los gráficos de información. Este script debe estar ubicado en el mismo directorio que el ejecutable, y por esta razón deberán estar en aquel directorio todos los script que son necesarios para emitir los gráficos y todas las carpetas con las bases de datos. En caso contrario la aplicación quedará estancada hasta la fecha anteriormente informada.

Se ha decidido esta modalidad debido a que la emisión de gráficos para un mes puede tardar más de 12 horas e inclusive más de un día si se emitieran todos juntos. Añadir un botón de actualizar gráficos en la aplicación (con el motivo de que el ejecutable descargue los datos y emita los gráficos) es algo bastante lento y dejaría la aplicación sin la posibilidad de uso durante ese periodo y probablemente la aplicación terminaría colapsando. Lo anterior se ve potenciado con la forma en que fue creada la aplicación, el script ejecuta la instrucción de abrir la aplicación y luego la orden de interactuar con el usuario, es decir, mientras la aplicación está abierta no es posible ejecutar una instrucción diferente a las que el usuario puede acceder, con lo que se descarta implementar la automatización automática (sin un botón en la plataforma).

Considerando lo anterior, los ejecutables entregados tendrán una fecha de visualización tope hasta Diciembre 2020. Si el usuario desea ver gráficos del año 2021 deberá incorporar este año de manera manual, como fue explicado en secciones anteriores, y luego volver a emitir un ejecutable. Para ello se recomienda instalar el modulo PyInstaller y luego utilizar en la terminal correr el comando: auto-py-to-exe, el cual realiza la tarea de emitir un ejecutable, para lo cual el archivo a transformar se llama untitled2.py. En este caso podría haber problemas y se recomienda previamente instalar los módulo PyQtWebEngine, calendar, pandas, pyvis y matplotlib.

Es muy importante recalcar que ante un cambio en el formato de archivos (de cualquiera de las bases) tiene como repercusión que el programa no analice dicho archivo. Esto incluye variaciones como cambio en las mayúsculas en una pestaña de excel, saltos de fila diferentes previo a una tabla de excel, entre otros. Este problema también ocurre en caso de modificaciones a los enlaces de descarga para los archivos durante el web scraping. Si el coordinador

modifica dichos enlaces, el archivo no se podrá descargar y el usuario deberá cambiar el enlace de forma manual. En cualquiera de estos casos no se requiere de un nuevo ejecutable.

6.3.1. Web Scraping y Automatización de procesos

El resultado del Web Scraping fue completamente exitoso. Pese a que esta aplicación abarca hasta el año 2020, se espera que los usuarios puedan hacer uso de esta hasta el tiempo que estimen conveniente. Con este fin se creó un script que mantiene la continuidad de descarga de las bases de datos y de la emisión de gráficos. Deben considerarse los tiempos de emisión de archivos si se desea ver un gráfico con fecha menor a dos meses de la información. Los gráficos y opciones de combobox se emiten una vez se descargue el o los archivos necesarios. Los tiempos de emisión de gráfico/lista son los siguientes:

- Combobox pestaña retiros e inyecciones: demora al rededor de 10 a 20 minutos en emitir la lista para un mes.
- Gráfico e información pestaña retiros e inyecciones: tarda al rededor de 3 a 12 minutos en emitir los archivos correspondientes a una subestación en un mes. Mientras más medidores tenga la subestación de interés, mas demora en ejecutar la función generadora.
- Gráficos pestaña generación: demoran al rededor de 5 min por emisión mensual de información.
- Combobox barras de pestaña costos marginales: demora entre 10 a 15 min en emitir la lista mensual.
- Gráfico e información costos marginales: tarda al rededor de 5 min para cada barra. Lo que e traduce en al rededor de 12 a 18 horas por mes.
- Gráfico de torta pestaña información operacional: demora al rededor de 1 a 2 min por gráfico.
- Gráfico de barras y lista desplegable en pestaña archivos rio: el gráfico y la lista son generados por la misma función, la cual demora al rededor de 3 min en ejecutarse. A partir de las opciones de la lista anterior se produce una tabla informativa, que para su visualización requiere un archivo en extensión csv previamente emitido con lo anterior. A partir de dicho archivo se genera la tabla, la cual no está previamente generada debido a que no demora más de 5 segundos en estar lista.

Con lo que no basta con que el programa descargue el o los archivos correspondientes por mes, si no que se debe esperar pacientemente que el algoritmo emita los gráficos para poder visualizarlos.

Lo que debe realizar el usuario para poder mantener la continuidad de descarga es bastante sencillo. Debe, en primer lugar, tener todos los archivos (ya sea carpetas de bases de datos, carpetas de gráficos y el script de automatización y todos los script con las funciones que realizan análisis de bases de datos) en la misma ubicación. En caso contrario el programa no podrá correr. A continuación se muestra la lista con las carpetas y scripts necesarios para que se ejecute todo lo anterior:

1. Archivos CMg, Archivos de medida, Archivos Generacion Real y Archivos RIO: estas cuatro carpetas contienen los archivos descargados y los que se descargarán. A cada base le corresponde una de estas carpetas.

2. Graficos CMg, Grafico Flujos, Graficos Gen y Graficos RIO: estas cuatro carpetas guardan todos los gráficos emitidos, la informacion de los cuadros de texto y todas las opciones de las listas desplegables. Por cada base es una carpeta.
3. extraccion_CMg_SIC.py, extraccion_flujos_Transf.py, extraccion_tablas_archivosRIO.py, extraccion_tablas_gen_Real.py: estos cuatro script realizan el análisis completo de cada base de datos.
4. descarga bases de datos.py: este es el script que realiza la automatización del proceso.

La forma en que el usuario realiza la descarga de archivos y emisión de archivos se realiza abriendo el último archivo anterior y ejecutándolo. La acción anterior no requiere ni realiza cambios sobre el ejecutable del CEN.

6.4. Repositorio

Finalmente, se tienen tres enlaces importantes que guardan relación con el trabajo realizado. El primer enlace contiene todos los códigos necesarios para la elaboración de la Aplicación CNE, estos se encuentran en la plataforma Google Colab. El segundo enlace, y de manera análoga al primero, contiene los códigos de la Aplicación CEN. El tercer enlace por su parte redirige hasta una dirección de Google Drive. Este último repositorio contiene todos los script utilizados en este trabajo, ambos ejecutables (denominados Aplicacion CEN 2021 y Aplicación CNE 2021) y todas las carpetas que contienen graficos o bien mapas de la red. Se espera hacia el futuro poder almacenar las bases de datos en tal dirección. Los enlaces son los siguientes:

1. Aplicación CNE:
   https://colab.research.google.com/drive/
   1E0TvBtpQnLHWFMLOnM9bB62xNDJCbhhq?usp=sharing
2. Aplicacion CEN:
   https://colab.research.google.com/drive/
   1L4RSqdD8nns8E51LSnlqsU1rrEoQycqc?usp=sharing
3. Drive:
   https://drive.google.com/drive/folders/
   1RkakwADXRgxBy93SXiYMo3tKpI8C6X-U?usp=sharing

Debido a la extensión del enlace, este se separó en reglones para no confundir al lector en caso de usar guiones.

Capítulo 7

Conclusión

Si bien, tanto el CEN como la CNE disponen de mucha información pública referente a la operación del SEN y el mercado eléctrico chileno, reunir dicha información en un sólo lugar suena impensado y complicado de hacer, en este trabajo se plantearon las bases para establecer aquello.

Se logró de manera exitosa realizar Web Scraping en la página del CEN y, además, se incluyeron ejemplos con breves recomendaciones para ampliar la descarga automatizada hacia otros documentos que son de uso recurrente entre los agentes del mercado eléctrico chileno, tales como las cartas que el coordinador emite de manera diaria. En cuanto a la CNE, fue posible descargar de manera manual las bases del Precio Nudo a Corto Plazo, y utilizarlas como los cimientos de la aplicación de la CNE.

En cuanto a la aplicación de la CNE, se crearon mapas mensuales con las simulaciones obtenidas de las salidas de las Bases Precio Nudo Corto Plazo. Si bien las posiciones de las barras no corresponden a las posiciones reales, fue posible localizar en forma manual alrededor de 180 barras con posiciones referenciales que sigan la geometría actual de su posición. Las barras que no tenían esta referencia, fueron posicionadas con un algoritmo que establece una posición cercana a sus barras vecinas, con lo que en muchas ocasiones es normal ver barras sobrepuestas. Lo importante es estar consciente de que la posición es solamente referencial.

El resultado del trabajo para la aplicación anterior, produjo todas las visualizaciones de la red para el horizonte esperado (10 años a partir de la emisión de la base de datos), y la estrategia de visualización de la red culminó con una ventana informativa con los resultados del análisis de las bases del Precio Nudo a Corto Plazo que incluyen datos de Demanda y Costo Marginal sobre todas las barras. Esta estrategia de visualización resulta amigable con el usuario y además es posible extenderla, pudiendo añadirle más información en el futuro. Su única limitante consiste en que la información solamente puede presentarse como texto y no en gráficos.

Por su parte, la aplicación de la CNE utilizó como estrategia de visualización, en primer lugar, la agrupación de todos los resultados de una base en una pestaña de la aplicación. En segundo lugar, con la mayoría de la información contenida en gráficos y cuadros de texto, los

cuales representan los resultados del análisis de las bases de datos.

Finalmente, se concluye entonces que ambas aplicaciones resultaron exitosas en cuanto a los objetivos planteados al principio de este documento. Este trabajo sin duda resulta ser la base para aplicaciones más complejas y con mayor contenido informativo, ya que la incorporación de nueva información tiene a favor los siguientes puntos:

1. El método de descarga utilizado para la realización de Web Scraping resultó exitoso y puede extrapolarse a más archivos o bases de datos.
2. La forma en que está realizado el análisis de las bases de datos de la CNE resulta sencillo si se tienen archivos muy similares, con lo que también es posible aplicarlo en otras bases de datos
3. Para ampliar las bases del CEN y realizar análisis sobre estas, debe buscarse entre los archivos ya trabajados alguno que se asemeje y así copiar el algoritmo y adaptarlo a una base nueva.
4. Debido a que la aplicación fue creada con el programa Qt Designer, y al hecho de que aún se conservan los archivos de creación original de la aplicación, es posible incorporar nuevas pestañas en la aplicación del CEN. Lo anterior se debe realizar con mucho cuidado y añadiendo elementos sobre la aplicación ya creada.

7.1. Recomendaciones y Trabajo futuro

Este proyecto fue elaborado como la base para futuras aplicaciones más complejas y con mayor contenido informativo referente al mercado eléctrico chileno. Siendo esta una primera versión, tiene bastantes características que debiesen mejorarse.

Los códigos de análisis de bases de datos no son robustos, es decir, ante mínimas variaciones en el formato de los archivos el programa se cae. Se espera que la siguiente persona que continúe este proyecto sea capaz de anticipar algunas de estas modificaciones y condicionar al programa para que no genere errores. El método try-except es una buena solución para evitar estos problemas. Las variaciones o errores más comunes en un archivo son:

1. Saltos en las primeras filas del archivo excel.
2. Cambios o variaciones de nombre en las pestañas del archivo.
3. Cambios o variaciones en el nombre de las columnas de interés.
4. Para archivos que se emiten de manera diaria, suele ocurrir que algunos archivos (no más de 5 al año) contengan columnas vacías de datos. En este caso el código actual podría fallar, ya que algunos scripts utilizan la primera fila de datos para establecer el cambio de día.
5. Algunos archivos no contienen todas las columnas de interés. Particularmente esto ocurrió con los Archivos RIO, en los que de manera arbitraria faltaba cierta columna de información con las unidades que marginaban sobre cierta barra en particular (barra Las Palmas). En este caso y, debido a la cantidad de veces que este problema ocurrió, se optó por no considerar dicha barra en el análisis.

De manera análoga los script de descarga de archivos (Web Scraping), están programados

únicamente para una cantidad limitada de enlaces de descarga. Los enlaces de descarga son escritos únicamente por el CEN, y esta entidad puede modificarlos cuando lo considere necesario, con lo que predecir los nuevos enlaces no es una opción viable. Actualmente el método de descarga utiliza la librería Requests, la cual únicamente necesita la url para extraer archivos desde la web. Una solución tentativa para prevenir este problema corresponde a modificar la forma de descarga, para lo cual se propone la creación del algoritmo con la librería Selenium, en tal caso el algoritmo debe realizar todos los pasos que un usuario humano realizaría para efectuar una descarga, ya sea rellenar casillas con fechas o formatos, presionar botones, entre otros. Esta librería es más avanzada que Requests pero es independiente del enlace de descarga. El nuevo algoritmo se caería únicamente cuando el coordinador modifique su sitio web, ya que los atributos de cada elementos cambiarían.

En cuanto al aspecto visual de ambas aplicaciones, se propone modificar el orden de los elementos. Esto debido a que actualmente los elementos están ubicados según coordenadas cartesianas que especifican la posición en la ventana, la limitación de usar coordenadas corresponde a no poder ampliar la aplicación a ventana completa, debido a que se descuadran los elementos. La solución que se propone corresponde a elaborar grupos y sub grupos que contengan elementos (botones, cadenas de texto, gráficos, etc.), y posicionar espacialmente los grupos en la ventana, ya sea indicando lado izquierdo, derecho o centro, y parte superior, central e inferior de la misma. De esta manera será posible expandir la plataforma. También se recomienda posicionar los botones de carga de información para pestañas dentro de estas.

Ambas aplicaciones serían mucho más útiles si se les añadiera más contenido, por ejemplo, información de demanda, de flujos de transferencia, entre otros. Para la aplicación CEN se proponen las siguientes contribuciones al trabajo:

- Cada vez que el usuario posicione el cursor sobre la barra de un gráfico de barras, valga la redundancia, visualice sobre esta el valor exacto de la columna, del cual actualmente se puede obtener una aproximación mediante la visualización de este.
- Mostrar flujos de transferencia por las líneas.
- Para una selección acotada de consumidores eléctricos (como Sodimac, Parque Arauco y clientes libres en general), mostrar a cuanto equivale su consumo comparado con una vivienda. En este caso la información estaría asociada a una comuna, región o sector.
- Para aquellos consumidores anteriores, identificar cuales son clientes solares[1].
- A partir de los pliegos tarifarios de cada distribuidora, se propone comparar la tarifa actual del mecanismo regulado (para quienes no tienen medidor inteligente) versus una tarifa a costo reflexiva con distintos valores para un numero adecuado de bloques al día y asociado al costo marginal de esa zona. Los bloques (3 a 6 idealmente) tendrán precios distintos que reflejen el costo marginal de la energía.
- Dar la posibilidad al usuario de que seleccione la forma en que desea visualizar los gráficos, es decir, la segregación de información podría ser mensual, diaria, semanal, por tipo de día de la semana, percentiles, etc.

En cuanto a la aplicación CNE se recomienda asignar posiciones a las barras basadas en el sistema real de transmisión chileno, demarcar las líneas que contengan múltiples circuitos,

[1]Su consumo coincide o se asemeja a las horas de luz del día.

ya sea con un comentario o visualmente con más trazos paralelos de línea. También sería agradable para el usuario poder arrastrar los nodos, de manera que si algunos se sobreponen, el pueda moverlos cerca de su posición inicial.

Finalmente, lo ideal sería que la aplicación pudiera realizar todo el trabajo y el sistema completo ser independiente a un usuario que deba ejecutar el script de Web Scraping y emisión de gráficos informativos. Al incorporar este proceso en los ejecutables un usuario que desconoce de programación tendrá mejor acceso al trabajo.

Bibliografía

[1] *Objetivos y funciones*. Sitio Web Coordinador Electrico Nacional. Chile.

[2] *A dive into Web Scraper world*, D. Kumar and L. Singh. Published by IEE. India. 2016.

[3] *Structured web information extraction using repetitive subject pattern*, W. Thamviset and S. Wongthanavasu. 2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology. Thailand. 2012.

[4] *Extracting templates from Web pages*, R. Manjula and A. Chilambuchelvan. 2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE). India. 2013.

[5] *Method for Extracting Patterns of Coordinated Network Attacks on Electric Power CPS Based on Temporal–Topological Correlation*, L. Wang. In IEEE Access, Vol. 8. 2020.

[6] *A Study on Web Scraping*, R. Singh, International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering Vol. 8. India. 2019.

[7] *What's an ISO?*, Pyiso Documentation. Pyiso Web Page. 2016.

[8] *The Public Utility Data Liberation (PUDL) Project*, Catalist Cooperative. Catalyst Web Page. United States. 2019.

[9] *Web-Scraping Electricity Data with Selenium*, K. Fries. Medium Online Blog. 2019.

[10] *A Framework of Petroleum Information Retrieval System Based on Web Scraping with Python*, Y. Ren and Y. Ren. .ª Framework of Petroleum Information Retrieval System Based on Web Scraping with Python,"2018 15th International Conference on Service Systems and Service Management (ICSSSM). China. 2018.

[11] *A Collaborative and Content Based Event Recommendation System Integrated with Data Collection Scrapers and Services at a Social Networking Site*, M. Kayaalp, T. Ozyer and S. T. Ozyer. ,"2009 International Conference on Advances in Social Network Analysis and Mining. Greece. 2009.

[12] *The use of web scraping in computer parts and assembly price comparison*,L. R. Julian and F. Natalia. 2015 3rd International Conference on New Media (CONMEDIA). Indonesia. 2015.

[13] *Web Scraping Techniques to Collect Weather Data in South Sumatera*, Fatmasari, Y. N. Kunang and S. D. Purnamasari. 2018 International Conference on Electrical Engineering and Computer Science (ICECOS). Indonesia. 2018.

Anexos

A continuación se adjuntan la funciones que emitió el programa QtDesginer para la creación de la aplicación. Algunos parámetros fueron modificados.

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import QUrl
from PyQt5.QtWebEngineWidgets import QWebEngineView

class programa4(object):
    def setupUi(self, programa4):
        programa4.setObjectName("programa4")
        programa4.resize(1680, 680)
        self.verticalGroupBox.setObjectName("verticalGroupBox↩
           ")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.↩
           verticalGroupBox)
        self.verticalLayout.setObjectName("verticalLayout")
        self.txt_selec_base = QtWidgets.QLabel(self.↩
           verticalGroupBox)
        self.txt_selec_base.setObjectName("txt_selec_base")
        self.verticalLayout.addWidget(self.txt_selec_base)
        self.cbx_base = QtWidgets.QComboBox(self.↩
           verticalGroupBox)
        self.cbx_base.setObjectName("cbx_base")
        self.cbx_base.addItem("")
        self.cbx_base.addItem("")
        self.cbx_base.addItem("")
        self.cbx_base.addItem("")
        self.cbx_base.addItem("")
        self.cbx_base.addItem("")
        self.cbx_base.addItem("")
        self.cbx_base.addItem("")
        self.verticalLayout.addWidget(self.cbx_base)
        self.btn_cargar_base = QtWidgets.QPushButton(self.↩
           verticalGroupBox)
        self.btn_cargar_base.setMinimumSize(QtCore.QSize(92, ↩
           0))
        self.btn_cargar_base.setObjectName("btn_cargar_base")
        self.verticalLayout.addWidget(self.btn_cargar_base)
        self.comboBox_anho = QtWidgets.QComboBox(self.↩
           verticalGroupBox)
        self.comboBox_anho.setObjectName("comboBox_annho")
        self.verticalLayout.addWidget(self.comboBox_anho)
        self.comboBox_mes = QtWidgets.QComboBox(self.↩
           verticalGroupBox)
        self.comboBox_mes.setObjectName("comboBox_mes")
```

```
        self.verticalLayout.addWidget(self.comboBox_mes)
        self.pushButton_cargarmapa = QtWidgets.QPushButton(↩
            self.verticalGroupBox)
        self.pushButton_cargarmapa.setObjectName("↩
            pushButton_cargarmapa")
        self.verticalLayout.addWidget(self.↩
            pushButton_cargarmapa)
        self.verticalLayoutWidget = QtWidgets.QWidget(↩
            programa4)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect↩
            (245, 20, 1421, 631))
        self.verticalLayoutWidget.setObjectName("↩
            verticalLayoutWidget")
        self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.↩
            verticalLayoutWidget)
        self.verticalLayout_2.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout_2.setObjectName("verticalLayout_2↩
            ")
        self.widget =QWebEngineView(self.verticalLayoutWidget↩
            )
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.↩
            QSizePolicy.Expanding, QtWidgets.QSizePolicy.↩
            Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.widget.sizePolicy()↩
            .hasHeightForWidth())
        self.widget.setSizePolicy(sizePolicy)
        self.widget.setObjectName("widget")
        self.verticalLayout_2.addWidget(self.widget)
        self.verticalLayoutWidget_2 = QtWidgets.QWidget(↩
            programa4)
        self.verticalLayoutWidget_2.setGeometry(QtCore.QRect↩
            (10, 530, 200, 131))
        self.verticalLayoutWidget_2.setObjectName("↩
            verticalLayoutWidget_2")
        self.creditos = QtWidgets.QVBoxLayout(self.↩
            verticalLayoutWidget_2)
        self.creditos.setContentsMargins(0, 0, 0, 0)
        self.creditos.setObjectName("creditos")
        self.label = QtWidgets.QLabel(self.↩
            verticalLayoutWidget_2)
        self.label.setObjectName("label")
        self.creditos.addWidget(self.label)
        self.label_2 = QtWidgets.QLabel(self.↩
            verticalLayoutWidget_2)
        self.label_2.setObjectName("label_2")
```

```
        self.creditos.addWidget(self.label_2)
        self.label_3 = QtWidgets.QLabel(self.↩
            verticalLayoutWidget_2)
        self.label_3.setObjectName("label_3")
        self.creditos.addWidget(self.label_3)
        self.label_4 = QtWidgets.QLabel(self.↩
            verticalLayoutWidget_2)
        self.label_4.setObjectName("label_4")
        self.creditos.addWidget(self.label_4)
        self.retranslateUi(programa4)
        QtCore.QMetaObject.connectSlotsByName(programa4)
    def retranslateUi(self, programa4):
        _translate = QtCore.QCoreApplication.translate
        programa4.setWindowTitle(_translate("programa4", "↩
            Visualizaci n␣Bases␣Precio␣Nudo␣Corto␣Plazo␣CNE"))
        self.txt_selec_base.setText(_translate("programa4", "↩
            Seleccione␣base␣de␣datos"))
        self.cbx_base.setItemText(0, _translate("programa4", ↩
            "Primer␣Semestre␣2017"))
        self.cbx_base.setItemText(1, _translate("programa4", ↩
            "Segundo␣Semestre␣2017"))
        self.cbx_base.setItemText(2, _translate("programa4", ↩
            "Primer␣Semestre␣2018"))
        self.cbx_base.setItemText(3, _translate("programa4", ↩
            "Segundo␣Semestre␣2018"))
        self.cbx_base.setItemText(4, _translate("programa4", ↩
            "Primer␣Semestre␣2019"))
        self.cbx_base.setItemText(5, _translate("programa4", ↩
            "Segundo␣Semestre␣2019"))
        self.cbx_base.setItemText(6, _translate("programa4", ↩
            "Primer␣Semestre␣2020"))
        self.cbx_base.setItemText(7, _translate("programa4", ↩
            "Segundo␣Semestre␣2020"))
        self.label.setText(_translate("programa4", "Trabajo␣↩
            de␣Titulo␣para"))
        self.label_2.setText(_translate("programa4", "↩
            Ingeniera␣Civil␣Electrica"))
        self.label_3.setText(_translate("programa4", "Laura␣↩
            Figueroa␣Gallardo"))
        self.label_4.setText(_translate("programa4", "↩
            lauraelectrica95@gmail.com"))
        self.btn_cargar_base.setText(_translate("programa4", ↩
            "Cargar␣Base"))
        self.pushButton_cargarmapa.setText(_translate("↩
            programa4", "Cargar␣Mapa"))
```

Código 7.1: Creación aplicación CNE.

A continuación se adjunta la función datos_archivosRIO(), la cual retorna el arreglo con las columnas de interés de los archivos RIO para un mes.

```
def datos_archivosRIO(mes,anho):
    inicio = datetime(anho,mes,1)
    dias_mes=calendar.monthrange(int(anho),int(mes))[1]
    fin    = datetime(anho,mes,dias_mes)
    lista_fechas = [(inicio + timedelta(days=d)).strftime("%Y↩
       -%m-%d") for d in range((fin - inicio).days + 1)]
    if mes<10:
        mes='0'+str(mes)
#crearemos el df con el dia uno para luego concatenar los ↩
   restos del mes
    archivo1='Archivos␣RIO/'+str(anho)+'/RIO'+str(anho)+str(↩
       mes)+'01.xls'
    xls = pd.ExcelFile(archivo1)
    hojas = xls.sheet_names
    if 'Mov␣Centrales␣CMG' in hojas:
        hoja='Mov␣Centrales␣CMG'
    if 'MOV-CMG' in hojas:
        hoja='MOV-CMG'
    if 'Movimiento␣de␣Centrales' in hojas:
        hoja='Movimiento␣de␣Centrales'
    tabla_RIO = pd.read_excel(archivo1,encoding='1252',↩
       sheet_name=hoja,skiprows=1)#index_col=4)
    columnas=tabla_RIO.columns
    fecha1=[str(anho)+'-'+str(mes)+'-'+str(1)]*(len(tabla_RIO↩
       [columnas[0]]))
    if ('Central-Unidad' in columnas)==True:
        col_central=tabla_RIO['Central-Unidad']
    if ('Central/Equipo' in columnas)==True:
        col_central=tabla_RIO['Central/Equipo']
    if ('Equipo␣/' in columnas)==True:
        col_central=tabla_RIO['Equipo␣/']
    tabla_RIO20 = pd.read_excel(archivo1,encoding='1252',↩
       sheet_name=hoja)#index_col=4)
    tabla_RIO2021=tabla_RIO[1:]
    date='01'+'-'+str(mes)+'-'+str(anho)
    columnas_2021=list(tabla_RIO2021)
    if (date in columnas_2021)==True:
        col_central=tabla_RIO2021[date]
    if (columnas[3] in columnas)==False:
        col_central=tabla_RIO[columnas[3]]
    tdia=datetime.isoweekday(datetime(int(anho),int(mes),1))
    tdia=[tdia]*(len(tabla_RIO[columnas[0]]))
    indice1=list(range(0,len(fecha1)))
    a=0
```

```
    if a==0:
        tabla_RIO_2 = pd.read_excel(archivo1,encoding='1252',↩
           sheet_name=hoja,skiprows=2)#index_col=4)
        if ('D.ALMAGRO__220' in columnas)==True:
            if ('CRUCERO__220' in columnas)==True:
                    crucero=tabla_RIO['CRUCERO__220']
            if ('CRUCERO__220' in columnas)==False:
                    crucero=tabla_RIO['Barras␣CMG']
            col_DALMAGRO=tabla_RIO['D.ALMAGRO__220']
            col_CARDONES=tabla_RIO['CARDONES_220']
            col_PAZUCAR=tabla_RIO['P.AZUCAR__220']
            col_QUILLOTA=tabla_RIO['QUILLOTA__220']
            col_AJAHUEL=tabla_RIO['A.JAHUEL__220']
            col_CHARRUA=tabla_RIO['CHARRUA__220']
            col_PMONTT=tabla_RIO['P.MONTT___220']
        if ('D.ALMAGRO__220' in columnas)==False and ('D.↩
           ALMAGRO__220' in list(tabla_RIO_2))==True:
            crucero=tabla_RIO_2['CRUCERO__220']
            col_DALMAGRO=tabla_RIO_2['D.ALMAGRO__220']
            col_CARDONES=tabla_RIO_2['CARDONES_220']
            col_PAZUCAR=tabla_RIO_2['P.AZUCAR__220']
            col_QUILLOTA=tabla_RIO_2['QUILLOTA__220']
            col_AJAHUEL=tabla_RIO_2['A.JAHUEL__220']
            col_CHARRUA=tabla_RIO_2['CHARRUA__220']
            col_PMONTT=tabla_RIO_2['P.MONTT___220']
        if ('D.ALMAGRO__220' in columnas)==False and ('D.↩
           ALMAGRO__220' in list(tabla_RIO_2))==False:
            crucero=None
            col_DALMAGRO=None
            col_CARDONES=None
            col_PAZUCAR=None
            col_QUILLOTA=None
            col_AJAHUEL=None
            col_CHARRUA=None
            col_PMONTT=None
        df_rio= pd.DataFrame({'indice':indice1,'fecha':fecha1↩
           ,'Tipo␣de␣dia':tdia,'Hora␣Movi.':tabla_RIO['Hora␣↩
           Movi.'],'Central-Unidad':col_central,'Configuracion↩
           ':tabla_RIO['Configuraci n'],'Estado':tabla_RIO['↩
           Estado'],'Consigna':tabla_RIO['Consigna'],'↩
           CRUCERO__220':crucero,'DALMAGRO__220':col_DALMAGRO,↩
           'CARDONES_220':col_CARDONES,'PAZUCAR__220':↩
           col_PAZUCAR,'QUILLOTA__220':col_QUILLOTA,'↩
           AJAHUEL__220':col_AJAHUEL,'CHARRUA__220':↩
           col_CHARRUA,'PMONTT___220':col_PMONTT})
    for fecha in lista_fechas[1:]:
        partes_fecha=fecha.split('-')
```

```
        dia=partes_fecha[2]
        archivo_RIO='Archivos RIO/'+str(anho)+'/RIO'+str(anho)+str(mes)+str(dia)+'.xls'
        xls = pd.ExcelFile(archivo_RIO)
        hojas = xls.sheet_names
        if 'Mov Centrales CMG' in hojas:
            hoja='Mov Centrales CMG'
        if 'MOV-CMG' in hojas:
            hoja='MOV-CMG'
        if 'Movimiento de Centrales' in hojas:
            hoja='Movimiento de Centrales'
        tabla_RIO = pd.read_excel(archivo_RIO,encoding='1252',sheet_name=hoja,skiprows=1)#index_col=4)
        tabla_RIO202 = pd.read_excel(archivo_RIO,encoding='1252',sheet_name=hoja)#index_col=4)
        tabla_RIO2021=tabla_RIO202[1:]
        date=dia+'-'+partes_fecha[1]+'-'+partes_fecha[0]
        columnas_2021=list(tabla_RIO2021)
        columnas=tabla_RIO.columns
        fecha2=[str(anho)+'-'+str(mes)+'-'+str(dia)]*(len(tabla_RIO[columnas[0]]))
        indice=list(range(len(df_rio['indice']),(len(fecha2)+len(df_rio['indice']))))
        tdia=datetime.isoweekday(datetime(int(anho),int(mes),int(dia)))
        tdia=[tdia]*(len(tabla_RIO[columnas[0]]))
        if anho>2018 or (anho==2018 and int(mes)>6):
            lista_barras=['D.ALMAGRO__220','CARDONES_220','PAZUCAR__220','QUILLOTA__220','AJAHUEL__220','CHARRUA__220','PMONTT___220']
            tabla_RIO_2 = pd.read_excel(archivo_RIO,encoding='1252',sheet_name=hoja,skiprows=2)#index_col=4)
            if ('D.ALMAGRO__220' in columnas)==True:
                if ('CRUCERO__220' in columnas)==True:
                    crucero=tabla_RIO['CRUCERO__220']
                if ('CRUCERO__220' in columnas)==False:
                    crucero=tabla_RIO['Barras CMG']
                col_DALMAGRO=tabla_RIO['D.ALMAGRO__220']
                col_CARDONES=tabla_RIO['CARDONES_220']
                col_PAZUCAR=tabla_RIO['P.AZUCAR__220']
                col_QUILLOTA=tabla_RIO['QUILLOTA__220']
                col_AJAHUEL=tabla_RIO['A.JAHUEL__220']
                col_CHARRUA=tabla_RIO['CHARRUA__220']
                col_PMONTT=tabla_RIO['P.MONTT___220']
            if ('D.ALMAGRO__220' in columnas)==False and ('D.ALMAGRO__220' in list(tabla_RIO_2))==True:
                crucero=tabla_RIO_2['CRUCERO__220']
```

```
                col_DALMAGRO=tabla_RIO_2['D.ALMAGRO__220']
                col_CARDONES=tabla_RIO_2['CARDONES_220']
                col_PAZUCAR=tabla_RIO_2['P.AZUCAR__220']
                col_QUILLOTA=tabla_RIO_2['QUILLOTA__220']
                col_AJAHUEL=tabla_RIO_2['A.JAHUEL__220']
                col_CHARRUA=tabla_RIO_2['CHARRUA__220']
                col_PMONTT=tabla_RIO_2['P.MONTT___220']
            if ('D.ALMAGRO__220' in columnas)==False and ('D.↩
               ALMAGRO__220' in list(tabla_RIO_2))==False:
                crucero=None
                col_DALMAGRO=None
                col_CARDONES=None
                col_PAZUCAR=None
                col_QUILLOTA=None
                col_AJAHUEL=None
                col_CHARRUA=None
                col_PMONTT=None
        #         col_LPALMAS=None
            if ('Central-Unidad' in columnas)==True:
                col_central=tabla_RIO['Central-Unidad']
            if ('Central/Unidad' in columnas)==True:
                col_central=tabla_RIO['Central/Equipo']
            if ('Equipo␣/' in columnas)==True:
                col_central=tabla_RIO['Equipo␣/']
            if (date in columnas_2021)==True:
                col_central=tabla_RIO2021[date].tolist()
            df_rio2= pd.DataFrame({'indice':indice,'fecha':↩
               fecha2,'Tipo␣de␣dia':tdia,'Hora␣Movi.':↩
               tabla_RIO['Hora␣Movi.'],'Central-Unidad':↩
               col_central,'Configuracion':tabla_RIO['↩
               Configuraci n'],'Estado':tabla_RIO['Estado'],'↩
               Consigna':tabla_RIO['Consigna'],'CRUCERO__220':↩
               crucero,'DALMAGRO__220':col_DALMAGRO,'↩
               CARDONES_220':col_CARDONES,'PAZUCAR__220':↩
               col_PAZUCAR,'QUILLOTA__220':col_QUILLOTA,'↩
               AJAHUEL__220':col_AJAHUEL,'CHARRUA__220':↩
               col_CHARRUA,'PMONTT___220':col_PMONTT})

        df_rio = pd.concat([df_rio,df_rio2])
    df_rio.set_index('indice',inplace=True)
    return df_rio
```

Código 7.2: Función generadora datos Archivos RIO.

A continuación se adjunta la función func_tec_marginando(), la cual retorna el gráfico de torta para una barra del Archivo RIO. Los vectores hidro y termo fueron reducidos.

```
def func_tec_marginando(df_rio,barra,mes,anho):
    archivo=[]
    contador_hidro=timedelta(0,0,0)
    contador_termo=timedelta(0,0,0)
    contador_nn=timedelta(0,0,0)
    dic={'CRUCERO__220':'CRUCERO__220',...,'P.MONTT___220':'↩
       PMONTT___220'}
    columna=dic.get(barra)
    #configuraciones posibles
    termo=['ATACAMA-1TG1A_TG1A_GNL_INFLEX',...,'_INFLEX']
    hidro=['CIPRESES_sinv',...,'CIPRESES_vtodo']
    for index, row in df_rio.iterrows():
        valor_columna=str(row[columna])
        configuracion1=row[columna]
        hora1=row['Hora␣Movi.']
        fecha1=row['fecha']
        if valor_columna=='nan':
            if index<(len(df_rio['fecha'])-2) and index>0:
                hora2=df_rio.iloc[index+1, df_rio.columns.↩
                   get_loc('Hora␣Movi.')]
                fecha2=df_rio.iloc[index+1, df_rio.columns.↩
                   get_loc('fecha')]
                if type(hora2)==float:
                    hora2=df_rio.iloc[index+2, df_rio.columns↩
                       .get_loc('Hora␣Movi.')]
                    fecha2=df_rio.iloc[index+2, df_rio.↩
                       columns.get_loc('fecha')]
                if type(hora2)!=float and type(hora1)!=float:
                    fecha1_s=fecha1.split('-')
                    fecha2_s=fecha2.split('-')
                    t_nn=timedelta(int(fecha1_s[2]),hora1.↩
                       hour*3600+hora1.minute*60)
                    t_sig_nn=timedelta(int(fecha2_s[2]),hora2↩
                       .hour*3600+hora2.minute*60)
                    contador_nn+=(t_sig_nn-t_nn)
        elif valor_columna!='nan' and fecha1!='2019-04-06':
            if type(hora1)==str:
                if hora1[0]=='*':
                    hora1=hora1[1:]
                    hora1_split=hora1.split(':')
                    hora1=datetime(1,1,1,int(hora1_split[0]),↩
                       int(hora1_split[1]),int(hora1_split[2])↩
                       )
            if index<(len(df_rio['fecha'])-2) and index>0:
```

```
                hora2=df_rio.iloc[index+1, df_rio.columns.↩
                   get_loc('Hora Movi.')]
                fecha2=df_rio.iloc[index+1, df_rio.columns.↩
                   get_loc('fecha')]
                if type(hora2)==float:
                    hora2=df_rio.iloc[index+2, df_rio.columns↩
                       .get_loc('Hora Movi.')]
                    fecha2=df_rio.iloc[index+2, df_rio.↩
                       columns.get_loc('fecha')]
                if type(hora2)!=float and type(hora1)!=float:
                    fecha1_s=fecha1.split('-')
                    fecha2_s=fecha2.split('-')
                    if (configuracion1 in hidro)==True:
                        if type(hora2)==str:
                            if hora2[0]=='*':
                                hora2=hora2[1:]
                                hora2_split=hora2.split(':')
                                hora2=datetime(1,1,1,int(↩
                                   hora2_split[0]),int(↩
                                   hora2_split[1]),int(↩
                                   hora2_split[2]))
                        t_hidro=timedelta(int(fecha1_s[2]),↩
                           hora1.hour*3600+hora1.minute*60)
                        t_sig=timedelta(int(fecha2_s[2]),↩
                           hora2.hour*3600+hora2.minute*60)
                        contador_hidro+=(t_sig-t_hidro)
                    elif (configuracion1 in termo)==True:
                        if type(hora2)==str:
                            if hora2[0]=='*':
                                hora2=hora2[1:]
                                hora2_split=hora2.split(':')
                                hora2=datetime(0,0,0,int(↩
                                   hora2_split[0]),int(↩
                                   hora2_split[1]),int(↩
                                   hora2_split[2]))
                        t_termo=timedelta(int(fecha1_s[2]),↩
                           hora1.hour*3600+hora1.minute*60)
                        t_sig=timedelta(int(fecha2_s[2]),↩
                           hora2.hour*3600+hora2.minute*60)
                        contador_termo+=(t_sig-t_termo)
    dias_mes=calendar.monthrange(int(anho),int(mes))[1]
    t_mes=dias_mes*24*60*3600 #mes en segundos
    dia_ter=contador_termo.days
    segundos_ter=contador_termo.seconds
    dia_hidro=contador_hidro.days
    segundos_hidro=contador_hidro.seconds
    dia_nn=contador_nn.days
```

```
    segundos_nn=contador_nn.seconds
    porcentaje_nn= (dia_nn*24*60*3600+segundos_nn)*100/t_mes
    porcentaje_ter=(dia_ter*24*60*3600+segundos_ter)*100/↩
       t_mes
    porcentaje_hidro=(dia_hidro*24*60*3600+segundos_hidro)↩
       *100/t_mes
    if (porcentaje_nn+porcentaje_hidro+porcentaje_nn)<98:
        porcentaje_nn=100-porcentaje_hidro-porcentaje_ter
    if porcentaje_ter==0 and porcentaje_hidro==0:
        porcentaje_nn=100
    if porcentaje_nn!=100:
        plt.pie([porcentaje_ter,porcentaje_hidro,↩
           porcentaje_nn], labels=['Centrales␣T rmicas␣\n'+↩
           str("{0:.3f}".format(porcentaje_ter))+'%','↩
           Centrales␣Hidraulicas␣\n'+str("{0:.3f}".format(↩
           porcentaje_hidro))+'%',"Sin␣informaci n␣\n"+str("↩
           {0:.3f}".format(porcentaje_nn))+'%'],colors=['grey'↩
           ,'royalblue','gold'])
    if porcentaje_nn==100:
        plt.pie([porcentaje_nn], labels=["Sin␣informaci n␣\n↩
           "+str("{0:.3f}".format(porcentaje_nn))+'%'],colors↩
           =['gold'])
    dic_meses={1:'Enero',2:'Febrero',3:'Marzo',4:'Abril',5:'↩
       Mayo',6:'Junio',7:'Julio',8:'Agosto',9:'Septiembre',10:↩
       'Octubre',11:'Noviembre',12:'Diciembre'}
    plt.title('%␣Tecnolog as␣que␣Marginan␣en␣'+barra+'␣para␣↩
       '+dic_meses.get(int(mes))+'␣'+str(anho))
    plt.savefig('Graficos␣RIO/'+str(anho)+'_'+str(mes)+'_'+↩
       barra+'_torta.png')
    plt.show()
    plt.clf()
    plt.cla()
    plt.close()
```

Código 7.3: Función func_tec_marginando.

A continuación se adjunta la función contador_accion(), la cual el arreglo de datos con todas las centrales que incurrieron en una consigna y el tiempo que acumularon en dicho estado.

```
def contador_accion(df_rio,accion_unidad):
    columnas=list(df_rio)
    if ('Central-Unidad' in columnas)==True:
        ind_cen=columnas.index('Central-Unidad')
        columnas[ind_cen]='Central'
        df_rio.columns=columnas
    accion1=df_rio.query('Consigna == @accion_unidad') #todos los que hacen la accion
    accion=accion1['Central']
    repeticiones=np.unique(accion.tolist(),return_counts=True)
    centrales=repeticiones[0]
    veces_accion=repeticiones[1]
    indice=0
    df_datos=pd.DataFrame({'Central':centrales,'Accion':[accion_unidad]*len(centrales),'Veces':veces_accion,'Contador':None})
    for central in centrales:
        contador_central=timedelta(0,0,0)
        if accion_unidad=='FS':
            contador_central==timedelta(0,0,0)
            df_datos.iloc[indice, df_datos.columns.get_loc("Contador")]=contador_central
        elif veces_accion[indice]>0 and accion_unidad!='FS':
            df_accion_central=accion1.query('Central == @central')#todas las veces que estuvo en MT
            for vez in list(range(0,veces_accion[indice])):
                linea_df_1accion=df_accion_central.iloc[vez:vez+1]
                fecha1=linea_df_1accion['fecha'].sum()
                hora_sum=linea_df_1accion['Hora Movi.'].sum()
                if fecha1!='2018-07-31' and fecha1!='2018-03-31' and fecha1!='2019-04-06' and type(hora_sum)!=float and str(fecha1)!='0':
                    dia2=int(fecha1[-2:])+1
                    fecha2=fecha1[0:-2]+str(dia2)
                    hora=linea_df_1accion['Hora Movi.'].tolist()
                    hora_timedelta=timedelta(dia2-1,hora[0].hour,hora[0].minute*60)#podria ser hour*3600
                    configuracion=linea_df_1accion['Configuracion'].sum()
```

```
                posible_fin_accion1=df_rio.query('Central
                    == @central and Configuracion == 
                    @configuracion and fecha == @fecha1 ')
                if posible_fin_accion1['fecha'].sum()!=0:
                    #si no hay valores no sume nada
                    horas_posible1=posible_fin_accion1['
                        Hora Movi.'].tolist()
                    index_hora=horas_posible1.index(hora
                        [0])
                    cant_horas_posibles=len(
                        horas_posible1)
                    if horas_posible1[cant_horas_posibles
                        -1]!=hora[0]:
                        hora_posible_timedelta=timedelta(
                            dia2-1,horas_posible1[
                            index_hora+1].hour*3600,
                            horas_posible1[index_hora+1].
                            minute*60)
                        contador_central+=(
                            hora_posible_timedelta-
                            hora_timedelta)
                    elif horas_posible1[
                        cant_horas_posibles-1]==hora[0]:
                        posible_fin_accion2=df_rio.query(
                            'Central == @central and 
                            Configuracion == @configuracion
                            and fecha == @fecha2 ')
                        horas_posible2=
                            posible_fin_accion1['Hora Movi.
                            '].tolist()
                        hora_posible_timedelta2=timedelta
                            (dia2,horas_posible2[0].hour,
                            horas_posible2[0].minute*60)
                        contador_central+=(
                            hora_posible_timedelta2-
                            hora_timedelta)
            df_datos.iloc[indice, df_datos.columns.get_loc("
                Contador")]=contador_central
            df_datos.sort_values('Veces',ascending=False)
        indice+=1
    return (df_datos)
```

Código 7.4: Función contador_accion().

A continuación se adjunta la función actualizar_bases_graficos(), la cual extrae las bases de datos y emite los gráficos correspondientes.

```
def actualizar_bases_graficos():
    fecha_actual=time.strftime("%x")
 #   fecha_actual='08/01/21'
    f=fecha_actual.split('/')
    anho='20'+str(f[2])
    mes=f[1]
    dia=int(f[0])-2
    if int(dia)<10:
        dia='0'+str(dia)
    fecha=anho+'-'+mes+'-'+dia
    #descarga diaria, se considera para dos dias anteriores
    archivoRIO="Archivos RIO/"+anho+"/RIO"+anho+mes+dia+".xls↩
       "
    if os.path.isfile(archivoRIO)==False:
        descarga_RIO(fecha)
    if os.path.isfile(archivoRIO)==True:
        sizefile = int(os.path.getsize(archivoRIO))
        if sizefile<=14764:
            os.remove(archivoRIO)
    #agendaremos la descarga para gen
    archivogen="Archivos Generacion Real/"+str(anho)+"/↩
       Generacion Real_"+str(anho)+"_"+str(int(mes))+".xlsx"
    if os.path.isfile(archivogen)==False:
        descarga_gen(mes,anho)
    if os.path.isfile(archivogen)==True:
        sizefile2 = int(os.path.getsize(archivogen))
        if sizefile2<=14764:
            os.remove(archivogen)
    #agendaremos la descarga para CMg
    archivoCMg="Archivos CMg/"+str(anho)+"/cmg"+str(anho)↩
       [-2:]+str(mes)+"_def.xlsm"
    if os.path.isfile(archivoCMg)==False:
        descarga_CMg(mes,anho)
    if os.path.isfile(archivoCMg)==True:
        sizefile2 = int(os.path.getsize(archivoCMg))
        if sizefile2<=14764:
            os.remove(archivoCMg)
    #graficos archivos rio
    if int(mes)==1:
        mes=12
        anho=int(anho)-1
    else:
```

```
        mes=int(mes)-1
    dias_mes=calendar.monthrange(int(anho),int(mes))[1]
    inicio = datetime(int(anho),int(mes),1)
    fin    = datetime(int(anho),int(mes),dias_mes)
    lista_fechasRIO = [(inicio + timedelta(days=d)).strftime(↩
       "%Y-%m-%d") for d in range((fin - inicio).days + 1)]
    #aeguraremos que el mes est  completo
    for fecha_dia in lista_fechasRIO:
        fecha=fecha_dia.split('-')
        anho=fecha[0]
        mes=fecha[1]
        dia=fecha[2]
        archivoRIO="Archivos␣RIO/"+anho+"/RIO"+anho+mes+dia+"↩
           .xls"
        if os.path.isfile(archivoRIO)==False:
            descarga_RIO(fecha_dia)
            if os.path.isfile(archivoRIO)==True:
                sizefile = int(os.path.getsize(archivoRIO))
                if sizefile<=14764:
                    os.remove(archivoRIO)
        if dia==dias_mes and os.path.isfile(archivoRIO)==True↩
           :
            marcando='Graficos␣RIO/lista_centrales_RIO_'+str(↩
               int(anho))+'_'+str(int(mes))+'.txt'
            if os.path.isfile(marcando)==False:
                df_rio=datos_archivosRIO(int(mes),int(anho))
                unidades_marcando(df_rio,int(mes),int(anho))
                lista_barras_m=['CRUCERO__220','D.↩
                   ALMAGRO__220','CARDONES_220','P.AZUCAR__220↩
                   ','QUILLOTA__220','A.JAHUEL__220','↩
                   CHARRUA__220','P.MONTT___220']
                for i in lista_barras_m:
                    graf_torta_RIO='Graficos␣RIO/'+str(anho)+↩
                       '_'+str(mes)+'_'+i+'_torta.png'
                    if os.path.isfile(graf_torta_RIO)==False:
                        func_tec_marginando(df_rio,i,int(mes)↩
                           ,int(anho))
    #graficos GEN
    archivo_barras='Graficos␣Gen/'+str(anho)+'_'+str(mes)+'↩
       _barras.png'
    archivo_torta='Graficos␣Gen/'+str(anho)+'_'+str(mes)+'↩
       _torta.png'
    if os.path.isfile(archivogen)==True and archivo_barras==↩
       False and archivo_torta==False:
        if os.path.isfile(archivo_barras)==False:
            df_gen=datos_gen(int(mes),int(anho))
            gen_pref_user(df_gen,4)
```

```
        if os.path.isfile(archivo_torta)==False:
            df_gen=datos_gen(int(mes),int(anho))
            gen_pref_user(df_gen,0)
    #graficos CMg
    if os.path.isfile(archivoCMg)==True:
        nombre_txt_CMg='Graficos CMg/lista_cbbx_'+str(anho)+'_'+str(mes)+'.txt'
        if os.path.isfile(nombre_txt_CMg)==False:
            lista_barras(mes,anho)
        barras=[]
        archivo = open(nombre_txt_CMg, 'r')
        c=archivo.read()
        lista=c.split(',')
        for i in lista:
            barras.append(i)
        archivo.close()
        for barrita in barras:
            grafico_CMg='Graficos CMg/'+str(anho)+'_'+str(mes)+'_'+barrita+'.png'
            if os.path.isfile(grafico_CMg)==False:
                df_CMg=datos_CMg(mes,anho,barrita)
                promedio_CMg(df_CMg,barrita,anho,mes)
    #graficos flujos
    if int(mes)==1:
        mes=12
        anho=int(anho)-1
    else:
        mes=int(mes)-1
    contenido = os.listdir('Archivos de medida/'+str(anho)+'/'+'+int(mes))
    if len(contenido)>2:
        nombre_txt_flujos='Graficos Flujos/lista_cbbx_'+str(anho)+'_'+str(mes)+'.txt'
        if os.path.isfile(nombre_txt_flujos)==False:
            lista_subestaciones(mes,anho)
            ssee=[]
            archivo = open(nombre_txt_flujos, 'r')
            c=archivo.read()
            lista=c.split(',')
            for i in lista:
                ssee.append(i)
            archivo.close()
            for subestacion in ssee:
                nombre_txt2='Graficos Flujos/'+subestacion+'_'+str(anho)+'_'+str(mes)+'.png'
                if os.path.isfile(nombre_txt2)==False:
                    iny_retiros(subestacion,mes,anho)
```

```
        if os.path.isfile(nombre_txt_flujos)==True:
            ssee=[]
            archivo = open(nombre_txt_flujos, 'r')
            c=archivo.read()
            lista=c.split(',')
            for i in lista:
                ssee.append(i)
            archivo.close()
            for subestacion in ssee:
                nombre_txt2='Graficos␣Flujos/'+subestacion+'_↩
                   '+str(anho)+'_'+str(mes)+'.png'
                if os.path.isfile(nombre_txt2)==False:
                    iny_retiros(subestacion,mes,anho)
```

Código 7.5: Función actualizar_bases_graficos().