UNIVERSIDAD DEL BIO-BIO

Facultad de Ciencias Empresariales

Departamento Sistemas de la Información

PROYECTO DE TÍTULO PARA OPTAR A TITULO DE INGENIERO DE EJECUCIÓN EN COMPUTACIÓN E INFORMÁTICA

DESARROLLO DE PROTOTIPO DE PLATAFORMA WEB PARA LA VISUALIZACIÓN DE DATOS SOCIOECONÓMICOS ENFOCADOS EN LA REGIÓN DEL BIO-BÍO

Alumno: Rodrigo Hernández Martínez

Profesor Guía: Pedro Campos Soto

CONCEPCION, 2018

Tabla de Contenidos

Resumen

En este Proyecto de Título se presenta el desarrollo de un prototipo de plataforma web para datos estadísticos socioeconómicos de la región del Biobío, Se explican las motivaciones y los antecedentes que despertaron el interés para desarrollar el proyecto. Además presenta un marco teórico en donde se contextualiza el desarrollo, incluyendo conceptos de visual analytics, así como como información de las encuestas a usar. Posteriormente se presentan las herramientas que utilizadas para el desarrollo del prototipo, incluyendo lenguajes de programación, librerías, y framework, además del modelo de desarrollo a usar y se explican las ventajas del prototipo frente a otros software que actualmente existen. Se describe el diseño del prototipo, incluyendo el diseño arquitectónico, pasando por el diseño de los datos, las actividades y la interfaz gráfica de usuario. Se entregan detalles de implementación del prototipo, incluyendo la selección de la información a mostrar y la elección de métodos de visualización, enfatizando la carga de datos a la base de datos, las consultas y la programación de la interacción.. Finalmente se muestran los resultados en términos de visualizaciones predefinidas, las visualizaciones dinámicas y un módulo de administración de variables que permite controlar las variables utilizadas en las visualizaciones dinámicas.

Capítulo 1. Introducción

1.1 Motivación

Actualmente, se realizan estudios a nivel regional y nacional para determinar el estado de la economía del país a nivel social, como por ejemplo, la Encuesta de Caracterización Socioeconómica Nacional CASEN[1] , o la nueva Encuesta Nacional de Empleo NENE[2], que miden la situación económica de las familias chilenas y la situación de empleabilidad de los chilenos respectivamente. Los informes de los resultados son mostrados en el sitio web del Ministerio del Desarrollo Social en el caso de CASEN y el Instituto Nacional de Estadísticas en el caso de NENE, en formato pdf o xls, que se componen principalmente de texto y de información tabular, lo que es adecuado para un estudio en detalle por parte de expertos en el área pero no para su fácil comprensión por parte del público en general.

Debido a la importancia y el impacto particular de estas encuestas para la opinión pública y para la prensa, sería beneficioso facilitar la interpretación de manera integral de sus resultados. Sin embargo, el formato actual no facilita su interpretación y el análisis rápido. Además, los resultados no son tan accesibles para el público general, predominando el lenguaje técnico, lo que impide aún más su comprensión integral.

En este informe de Proyecto de Título se presenta el desarrollo de un prototipo de plataforma web de visualización de datos, que muestre representaciones gráficas de algunos datos relevantes de la región del Biobío, particularmente los datos que conciernen a las encuestas CASEN, NENE y NESI, basándose en conceptos de la literatura sobre visualización de datos.

1.2 Antecedentes

A principios de este año se estrenó un sitio web desarrollado por un investigador chileno del MIT (Instituto Tecnológico de Massachusetts), llamado Data Chile[3], el cual muestra

[1] Encuesta CASEN disponible en: http://observatorio.ministeriodesarrollosocial.gob.cl/casen/casen_obj.php
[2] Encuesta NENE disponible en: http://historico.ine.cl/canales/chile_estadistico/mercado_del_trabajo/nene/nene.php
[3] Página web de Data Chile disponible en: https://es.datachile.io/

información de datos públicos de los diferentes organismos de nuestro país. La información va desde economía, comercio, salud y educación, hasta información de demografía, índices de delincuencia y de resultados de elecciones presidenciales. Esta plataforma muestra datos de una manera en la que el usuario pueda interactuar con la página y los datos que muestra.

Sin embargo, desde antes, el MIT ha buscado mostrar datos de manera integral a través de visualizaciones interactivas, como es el caso del Observatory of Economic Complexity[4], que muestra datos del comercio internacional. En la actualidad, el observatorio muestra más de 20 millones de visualizaciones interactivas, conectando a cientos de países con sus destinos de exportación y con los productos que comercializan.

Estos casos muestran la relevancia que tiene hoy en día poder mostrar información de manera visual, facilitando su comprensión por parte de la población en general. Tomando como base estas experiencias, es que se propone este proyecto, para facilitar la visualización de datos económicos de la Región del Biobío.

1.3 Objetivos

Para la realización de este proyecto, se ha planteado el cumplimiento de los siguientes objetivos, separándolos en un objetivo general y objetivos específicos, estos son:

1.3.1 Objetivo General

Desarrollar un prototipo de plataforma web interactiva que permita visualizar datos socioeconómicos de la región del Biobío de manera dinámica e intuitiva, para facilitar la comprensión de los resultados de estudios de situación económica para un público general, utilizando para ello metodologías de visual analytics, que ayuden a mejorar la recepción cognitiva de la información a través de elementos visuales.

[4] Página web del Ovservatory of Economic Complexity disponible en: https://atlas.media.mit.edu/en/

1.3.2 Objetivos Específicos

1. Revisar el estado del arte en análisis visual de datos, con énfasis en visualización de datos socioeconómicos, para conocer la información y los avances más actualizados sobre el tema.
2. Estudiar la estructura de encuestas a utilizar y seleccionar las técnicas de visualización a implementar, para determinar los datos a utilizar de la encuesta, la información que se mostrará y la visualización que mejor se adapte a esta.
3. Implementar y adaptar las técnicas de visualización de datos seleccionadas, para conseguir el correcto funcionamiento de las visualizaciones, tanto a nivel visual como a nivel de la obtención de la información.
4. Desarrollar prototipo de plataforma web para la visualización de datos socioeconómicos de la Región del Biobío.

1.4 Alcances

- Las encuestas corresponden a las más actualizadas al momento de empezar el desarrollo del proyecto, estas son: encuesta CASEN 2015, NENE 2018 trimestre Enero-Febrero-Marzo, NESI 2016.
- El proyecto está enfocado para un usuario final que pueda visualizar información a través de la web.

1.5 Estructura del informe

Este informe se estructura de la siguiente forma. En este primer capítulo, se presentan la motivación, los antecedentes y los objetivos a cumplir. En el capítulo 2 se presenta el marco teórico de este trabajo. En el capítulo 3 se presentan las herramientas de desarrollo a utilizar y dónde interviene cada herramienta en el desarrollo, además de los requerimientos. En el capítulo 4 se presenta el diseño de la solución, tanto a nivel arquitectónico como detallado. En el capítulo 5 se detalla la implementación realizada, enfatizando el proceso desde la obtención de datos a la visualización. En el capítulo 6 se muestran ejemplos de las visualizaciones del prototipo donde se detalla la información que muestra cada visualización. Finalmente, en el capítulo 7 se presentan las conclusiones de este trabajo.

Capítulo 2. Marco Conceptual

2.1 Visual Analytics

En las últimas décadas se ha dado un aumento sustancial en la generación de datos que, en conjunto con las mejoras continuas en las formas de almacenamiento de datos, han influido en la forma de guardar los datos, haciéndolo sin filtrado ni refinamiento. De esta manera, cada vez más empresas, instituciones u organizaciones de diferentes rubro han encontrado posibilidades de almacenamiento al alcance de ellos, generando grandes volúmenes de datos. Sin embargo, los datos por sí solos no son relevantes, sino la información que se extrae de ellos, por ello han surgido problemas por la sobrecarga de datos para obtener información útil. Estos problemas son (Keim & Thomas, 2008):

- La información es irrelevante para la tarea que se necesita.
- La información no se procesa correctamente
- La información no se presenta correctamente

El correcto análisis de la información, que es desordenada e inconsistente, es crucial para la toma de decisiones de los analistas, o equipos de respuesta. Esto implica una extracción relevante de datos y un procesamiento adecuado de ellos. El objetivo del *visual analytics* es convertir la sobrecarga de información en una oportunidad. Los responsables de la toma de decisiones deberían poder examinar esta corriente de información masiva, multidimensional, de múltiples fuentes y variable en el tiempo para tomar decisiones efectivas en situaciones de tiempo crítico.

Visual analytics se puede definir como la ciencia del razonamiento analítico de la información facilitado por las interfaces visuales interactivas (Thomas & Cook, 2005). Estas interfaces deben garantizar el análisis de datos mediante el uso de métodos de visualización de datos que permitan facilitar la comprensión y la interacción del usuario.

El uso de representaciones visuales e interactivas de elementos abstractos, como por ejemplo el empleo o la calidad socioeconómica, permite ampliar el procesamiento cognitivo del ser

humano, ayudando a percibir la información de manera más eficiente, reduciendo el tiempo de búsqueda de información, permitiendo la proliferación de ideas y relaciones de manera rápida y finalmente, tomar acciones y decisiones.

2.1.1 Elementos

El proceso de *visual analytics* es iterativo y consta de tres elementos para producir el conocimiento: los datos, los modelos analíticos, y la visualización (Sacha et al., 2014).

Los datos son los que describen los hechos, debe ser representativo y deben estar relacionados con el proceso analítico, de lo contrario es difícil que revelen relaciones significativas en el problema. Además los procesos de creación, recolección y selección de los datos en el proceso de análisis visual, determinan la confiabilidad de los resultados del mismo

Los modelos analíticos deben ser una representación de la descripción de los datos y pueden ir desde estadísticas descriptivas hasta algoritmos de *data mining*. El proceso KDD (*Knowledge Discovery in Databases*) produce modelos a partir de datos. Estos modelos cumplen el propósito de entregar datos complementarios para generar conocimiento y ayudar a la toma de decisiones, por ejemplo, un resultado estadístico puede llevar a confirmar o no una hipótesis.

Las interfaces visuales deben cumplir ciertas condiciones para una visualización integral de datos, estas son (Rayón, 2015):

- Señalar relaciones, tendencias o patrones
- Explorar datos para inferir nuevo conocimiento
- Facilitar el entendimiento de un concepto, idea o hecho
- Permitir la observación de una realidad desde diferentes puntos de vista
- Recordar una idea principal.

Además deben incluir la intervención humana a través de la interacción dando un punto de conexión entre los datos y el usuario. Todo lo anterior, se muestra esquemáticamente en la Figura 1.

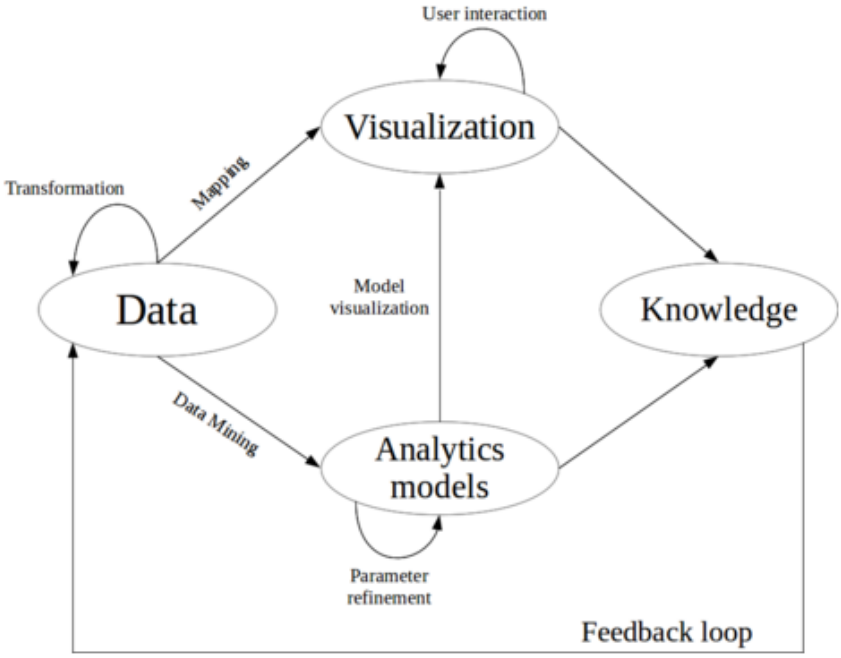

Figura 1. Elementos de visual analytics, tomado de (Rayón, 2015).

2.1.2 Alcances

Visual Analytics es mucho más que sólo visualización, que para el campo de las ciencias de la información es una representación gráfica de datos o conceptos, sino que es un enfoque integral que combina visualizaciones, factores humanos y el análisis de datos (Keim & Thomas, 2008).

La figura 2 muestra el alcance de estos factores. En el campo de visualizaciones, *visual analytics* integra el análisis de información, el análisis geoespacial y el análisis científico. Para los factores humanos se integra la interacción, la cognición, la percepción, la difusión, la producción y la presentación, que permiten la comunicación entre el ser humano y el computador. La producción, se define bajo el contexto de *visual analytics* como la creación de

materiales para resumir los resultados de un análisis, de la misma forma, la presentación se define como el agrupamiento de esos materiales para que sean comprensibles, y la difusión como el proceso para compartirlos. En el campo de análisis de datos se integra la gestión de datos y la representación del conocimiento, la generación del conocimiento y el análisis estadístico.

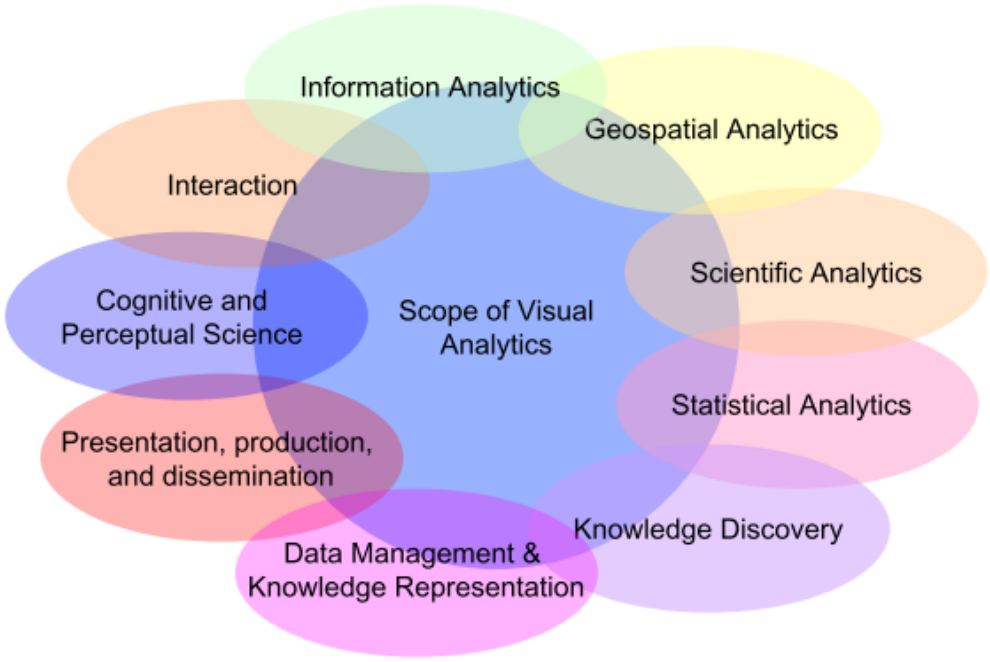

Figura 2. Alcance de Visual Analytics, tomado de (Keim & Thomas, 2008)

2.2 Tipos de Visualización de Datos

2.2.1 Treemap

Los Treemaps son métodos de visualización de espacio lleno capaces de representar jerárquicamente grandes volúmenes de datos cuantitativos. Un Treemap funciona dividiendo el área de visualización en una secuencia anidada de rectángulos, cuyas áreas corresponden a un atributo del conjunto de datos, combinando efectivamente los aspectos de un diagrama de Venn y un gráfico circular (Shneiderman & Wattenberg, 2001).

Figura 6. Ejemplo de un Treemap, tomado de (Ribecca, 2014)

El punto más importante al momento de implementar un Treemap es el método de ordenación de los rectángulos, y desde su introducción como elemento visual, se han desarrollado una gran variedad de algoritmos de ordenamiento de Treemaps, entre ellos se encuentran (Shneiderman & Wattenberg, 2001), (Bederson, Shneiderman, & Wattenberg, 2002):

- Binary Tree
- Ordered and quantum treemap
- Slice-and-dice treemap
- Squarified treemap
- Cluster treemap
- Strip treemap
- Pivot Treemap

2.2.1.1 Squarified Treemaps

El algoritmo Squarified Treemap nos entrega una serie de ventajas con respecto al resto. Según (Shneiderman & Wattenberg, 2001) en su estudio, utilizaron dos métricas para comparar los algoritmos: La relación de aspecto y la función de cambio de diseño. Una relación de aspecto y una función de cambio baja es importante para mantener un Treemap ordenado. En los resultados de sus estudios se determinó que un Squarified Treemap posee

una muy baja relación de aspecto, pero una función de cambio un poco grande. Sin embargo este cambio se puede reparar agregando información visual a los rectángulos del Treemap, de esta forma se puede saber dónde se ubica cada rectángulo en el Treemap. Esto se observa en la figura 7.

Algorithm	Aspect Ratio	Change
Slice-and-dice	26.10	0.46
Pivot-by-middle	3.97	1.08
Pivot-by-size	3.14	4.07
Cluster	1.79	7.67
Squarified	1.74	8.27

Figura 7. Comparación de algoritmos de Treemap para una distribución normal logarítmica 8x3, tomado de (Shneiderman & Wattenberg, 2001)

El propósito del algoritmo Squarified Treemap es permitir que la relación de aspecto entre los rectángulos sea lo más parecido posible entre ellos, asemejando lo más posible a un cuadrado. La figura muestra el proceso de llenado de un Squarified Treemap.

Figura 8. Proceso de generación de un Squarified Treemap, tomado de (Bruls, Huizing, & van Wijk, 2000)

Como se puede observar en la figura 8, el algoritmo va generando los rectángulos de acuerdo a la relación de aspecto entre ellos, por lo que mientras más rectángulos aparezcan, el cambio es más notorio, y la proporción de los rectángulos cambia. La figura 9 explica este proceso a través de un algoritmo en pseudocódigo.

```
procedure squarify(list of real children, list of real row, real w)
begin
    real c = head(children);
    if worst(row, w) ≤ worst(row++[c], w) then
        squarify(tail(children), row++[c], w)
    else
        layoutrow(row);
        squarify(children, [], width());
    fi
end
```

Figura 9. Algoritmo de Squirified Treemap, tomado de (Bruls et al., 2000)

El algoritmo es un procedimiento recursivo en el cual se toman dos decisiones: El rectángulo se agrega a la fila actual, o la fila actual se fija y empieza un nuevo procedimiento (*squarify*) con el espacio que queda. El algoritmo soporta una función *width*() que retorna la longitud más corta del subrectángulo restante en donde la fila es colocada y una función *layoutrow*() que agrega otra fila más al rectángulo (Bruls et al., 2000).

2.2.2 Heatmap

Un Heatmap muestra datos a través de variaciones del color sobre un área. Los colores se ordenan como una termografía, estableciendo como jerarquía dos polos, de manera que los colores cálidos (como el rojo, el naranjo y el amarillo) representan mayor magnitud y los colores fríos (como el azul y el verde) representan menor magnitud de esta manera entregan información en tres dimensiones, incluyendo al largo, el alto, y el color. Estos se pueden

combinar con un mapa geográfico para crear mapas de calor geográficos, y de esta manera eliminar parcialmente el problema de los mapas coropléticos en el uso del color (Mena, 2015).

Figura 10. Heatmaps. a) Forma común de uso b) Forma de uso con un mapa geográfico, tomado de (Wied, 2016)

2.2.3 Grafos

Los grafos son representaciones de relaciones binarias entre nodos o vértices conectados con arcos o aristas. Existen dos tipos de grafos, los grafos dirigidos y los grafos no dirigidos, la diferencia entre ellos es que en los grafos dirigidos los arcos tienen orientación.

Además los grafos tienen algunas propiedades, estas son (Wikipedia, 2003), (Bondy & Murty, 1982):

- **Adyacencia:** dos aristas son adyacentes si tienen un vértice en común, y dos vértices son adyacentes si una arista los une.
- **Incidencia:** una arista es incidente a un vértice si ésta lo une a otro.
- **Ponderación:** corresponde a una función que a cada arista le asocia un valor (costo, peso, longitud, etc.), para aumentar la expresividad del modelo.
- **Etiquetado:** distinción que se hace a los vértices y/o aristas mediante una marca que los hace unívocamente distinguibles del resto.

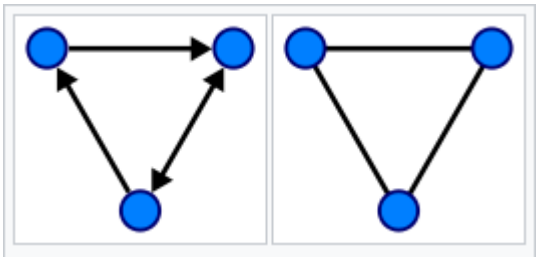

Figura 11. Ejemplos de grafo dirigido (derecha) y grafo no dirigido (izquierda)

En la figura 11 se observa que los grafos dirigidos tienen sus arcos con orientación, ya sea unidireccional o bidireccional, mientras que el grafo no dirigido no las tiene.

2.3 Información socioeconómica del país

2.3.1 Encuesta CASEN

La Encuesta Casen, o Encuesta de Caracterización Socioeconómica Nacional, es una encuesta a nivel nacional, regional y comunal, realizada en Chile desde el año 1985, cada dos o tres años(Wikipedia, 2013).

Esta encuesta es mandatada por el Ministerio de Desarrollo Social de Chile ex Ministerio de Planificación y Cooperación (Mideplan) y se caracteriza por medir las condiciones socioeconómicas de los hogares de Chile, en relación al acceso a la salud, la educación, el trabajo y a las condiciones actuales de la vivienda (Misnisterio de Desarrollo Social, 2011). Entrega datos relevantes para el país, como los índices de pobreza, indigencia, y las situaciones de ingresos por hogar. Además entrega información sobre aquellos con beneficios del Estado y permite el desarrollo de nuevos proyectos enfocados en las personas.

La encuesta Casen es ampliamente utilizada por otros Ministerios, entidades y servicios públicos para el diseño y evaluación de sus políticas y programas. Además diversas

universidades, centros académicos y otras entidades usan la información dada por la encuesta para desarrollo de proyectos de investigación e innovación.

La información que proporciona la encuesta, constituye un antecedente básico para focalizar el gasto social y sirve de gran manera al proceso de descentralización de la gestión del Estado, permite obtener resultados a nivel regional y para un número muy significativo de las comunas del país.

Los objetivos generales de la Encuesta Casen son (Wikipedia, 2013), (Ministerio de Desarrollo Social, 2016):

- Conocer periódicamente la situación de los hogares y de la población, especialmente de aquella en situación de pobreza y de aquellos grupos definidos como prioritarios por la política social, con relación a aspectos demográficos, de educación, salud, vivienda, ocupación e ingresos.
- Evaluar la cobertura y la distribución del gasto fiscal de los principales programas sociales de alcance nacional entre los hogares según su nivel de ingreso, así como el impacto de este gasto en el ingreso de los hogares y en la distribución del mismo.
- Medir la magnitud e incidencia de la pobreza en los hogares y en la población, permitiendo además la caracterización de los hogares y población en situación de pobreza utilizando otras dimensiones incluidas en la encuesta en sus distintos módulos.
- Evaluar los programas sociales en curso, determinar las líneas de acción a seguir y las correcciones y ajustes a implementar para lograr que el gasto social llegue a los segmentos poblacionales identificados como focos prioritarios de las políticas sociales y de cada uno de los programas.
- Caracterizar a la población por nivel de ingreso de los hogares, según sus condiciones habitacionales, educacionales, inserción al mercado del trabajo, composición de los ingresos familiares y otras variables relevantes.

La Encuesta CASEN consta de 7 módulos, estos son (Ministerio de Desarrollo Social, 2016):

- **Módulo Registro de Residentes:** Este módulo registra información para la identificación de las personas, los distintos núcleos familiares y hogares que habitan cada vivienda, como por ejemplo: sexo, edad, estado civil o conyugal, jefatura de hogar y de núcleo. Permite estimar indicadores tales como el tamaño, el tipo y la composición de los hogares.
- **Módulo Educación:** Este módulo incluye un conjunto de preguntas que permite estimar indicadores como la tasa de analfabetismo, los niveles de escolaridad de la población, la cobertura de los diferentes niveles de enseñanza por dependencia administrativa, la cobertura y focalización de diversos programas en este sector, tales como, el Programa de Alimentación Escolar (PAE), la entrega de útiles y textos escolares, atención dental y médica, entre otros.
- **Módulo Trabajo:** Este módulo consta de preguntas que permiten: estimar indicadores sobre la situación ocupacional de la población (tasa de participación, tasa de desocupación, tasa de ocupación); caracterizar la situación laboral y previsional de la fuerza de trabajo ocupada (rama de actividad, tamaño de la empresa, grupo ocupacional, categoría ocupacional, jornada de trabajo, contrato, seguro de cesantía, afiliación y cotización previsional, ocupación secundaria).
- **Módulo Ingresos:** Este módulo permite recoger información sobre las diferentes corrientes de ingreso que reciben las personas y los hogares, ya sea como fruto de su participación en el proceso productivo y en la propiedad de los activos, o como receptores de transferencias de privados o transferencias monetarias del Estado efectuadas como parte de sus programas sociales.
- **Módulo Salud:** Este módulo permite estimar indicadores tales como la cobertura de los sistemas previsionales de salud; el acceso a prestaciones de salud de carácter preventivo por parte de los niños y niñas durante la primera infancia, de los adultos mayores y de las mujeres durante el embarazo y lactancia; el acceso efectivo a servicios y atenciones de salud ante la demanda espontánea por enfermedad o accidente; la cobertura y focalización de los principales programas públicos de salud de alcance nacional (PNAC, PACAM, Papanicolau, mamografía, y Régimen de

Garantías Explícitas en Salud (AUGE-GES); y la percepción de la población sobre su estado de salud.

- **Módulo Residentes:** Las respuestas de este módulo permite estimar indicadores tales como nivel educacional de los padres del jefe de núcleo y su cónyuge, pertenencia a pueblos originarios o indígenas, utilización de las lenguas originarias por parte de la población indígena, participación en organizaciones, tenencia de bienes muebles y acceso a tecnologías de comunicación. Adicionalmente, se incorporan preguntas sobre: orientación sexual; identidad de género; apoyos de que dispone el hogar; trato injusto o discriminatorio a miembros del hogar.
- **Módulo Vivienda:** Este módulo se compone de preguntas que permiten estimar indicadores sobre las condiciones de habitabilidad en que residen los hogares del país, tales como materialidad y saneamiento de la vivienda, allegamiento, hacinamiento y cobertura de programas habitacionales, además de preguntas relacionadas con el entorno de la vivienda.

2.3.2 Nueva Encuesta Nacional de Empleo (NENE)

La Nueva Encuesta Nacional de Empleo (NENE) es un instrumento que mide la evolución de la fuerza y la relación laboral en Chile. Es aplicada desde el año 2010 reemplazando a la Encuesta Nacional de Empleo (ENE). De esta manera se puede clasificar a la población de Chile en términos laborales(INE, 2010b).

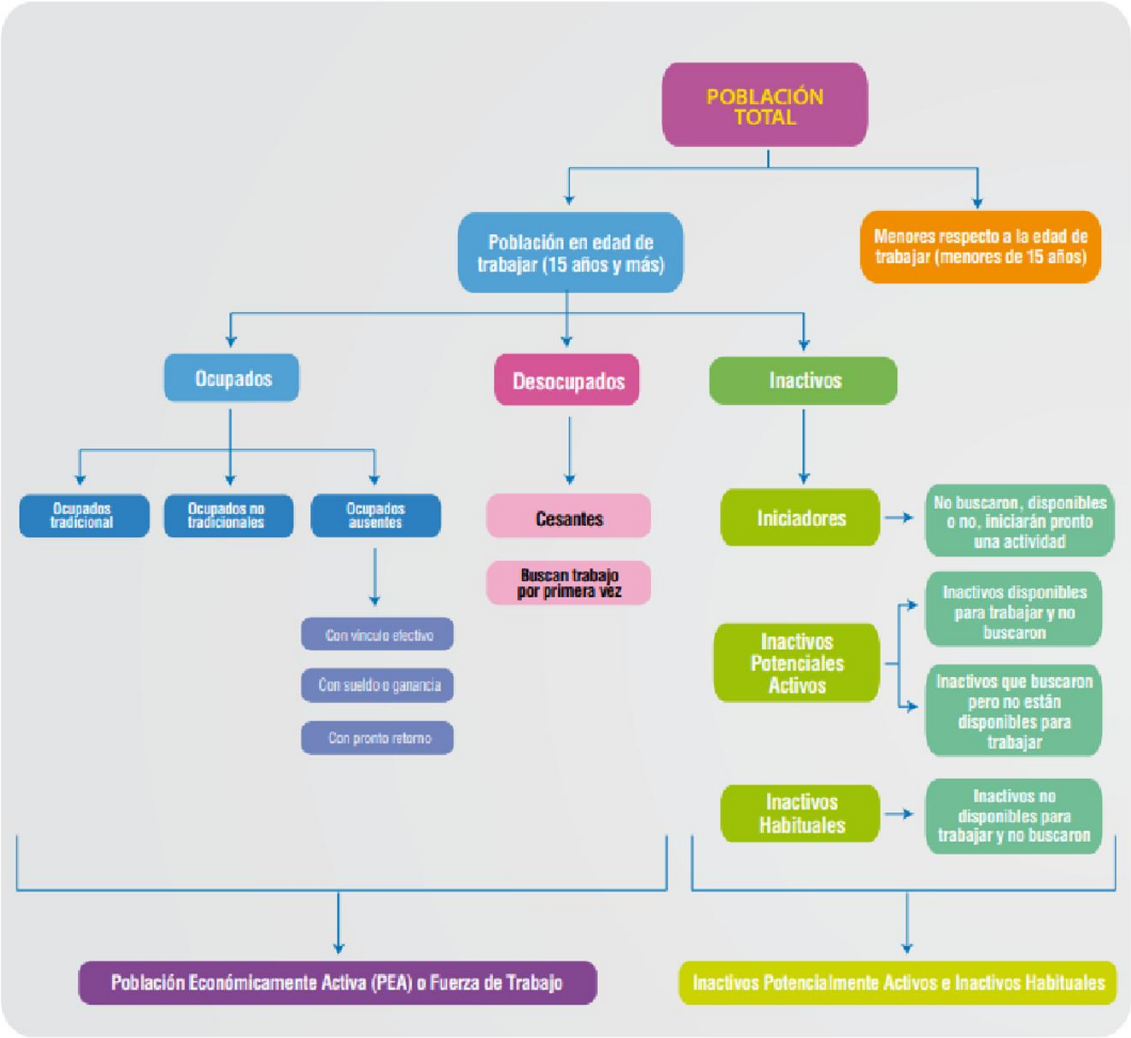

Figura 12. Clasificación de la Población según la NENE, tomado de (INE, 2010b)

Según la figura 12 la población económicamente activa (PEA) o fuerza de trabajo son aquellas personas que están en edad de trabajar (sobre los 15 años) y que están ocupados o desocupados; y las personas inactivas son aquellas en edad de trabajar pero que no lo hacen por distintos motivos.

Los ocupados se clasifican en tres categorías (INE, 2010a):

- Ocupados Tradicionales: Que perciben su trabajo como tal.

- Ocupados No Tradicionales: Que dicen haber trabajado en una labor pero no lo perciben como un trabajo tradicional.
- Ocupados Ausentes: Que perciben su trabajo como tal pero que mientras se realiza la encuesta no están trabajando, ya sea por vacaciones, permisos o licencia médica.

Los desocupados se clasifican en función de si tuvieron un trabajo anteriormente en: Cesantes, y buscan trabajo por primera vez.

Los inactivos son aquellas personas que no tienen empleo y tampoco lo buscaron y/o están disponibles para trabajar, aquí están, por ejemplo, los jubilados, las amas de casa, etc. Se clasifican según las razones por las cuales no trabajan, estas son:

- Iniciadores (que pretender iniciar un emprendimiento próximamente)
- Razones estaciónales
- Razones de desaliento
- Razones temporales
- Razones familiares permanentes
- Razones de estudio
- Razones de pensión o montepiado
- Razones de jubilación
- Razones de salud permanentes
- Sin deseos de trabajar

La aplicación de esta encuesta se desarrolla para cumplir con los siguientes objetivos:

- Permitir a instituciones públicas y privadas el desarrollo de políticas laborales
- Permitir a instituciones internacionales realizar estudios de comparabilidad
- Conocer y analizar el comportamiento de la fuerza de trabajo
- Analizar sectorialmente el empleo
- Estudiar el desempleo

La Nueva Encuesta Nacional de Empleo consta de 5 módulos (INE, 2018a):

- **Módulo de identificación de la persona encuestada:** Este módulo permite obtener datos referentes a la persona que responde la encuesta así como datos referentes a la aplicación de la encuesta, tales como: el estado conyugal, mes en el que se realizó la encuesta, sexo, nivel educacional, relación con el jefe de hogar entre otros.
- **Módulo A:** Este módulo contiene preguntas que permiten obtener datos sobre la situación laboral en la semana de referencia. Permite caracterizar el comportamiento laboral en una semana.
- **Módulo B:** Este módulo contiene preguntas que permiten caracterizar la actividad principal que desempeña el encuestado, tales como, la rama de actividad, cotizaciones, tipo de sector del trabajo, seguro de desempleo, entre otros.
- **Módulo C:** Este módulo permite obtener información sobre las horas trabajadas del encuestado, así como el tipo de jornada, las horas habituales de trabajo, días semanales de trabajo, entre otros.
- **Módulo E:** Este módulo contiene preguntas para las personas que están buscando empleo, tales como, el interés de la búsqueda de empleo, razones por las cuales busca empleo, gestión de la búsqueda de empleo, entre otros.

2.3.3 Nueva Encuesta Suplementaria de Ingresos (NESI)

La Nueva Encuesta Suplementaria de Ingresos (NESI) es un módulo complementario a la Nueva Encuesta Nacional de Empleo que se levanta una vez al año en el trimestre Octubre-Diciembre en todas las regiones de Chile. Su objetivo principal es caracterizar los ingresos laborales de las personas que son clasificadas como Ocupadas por la NENE y otras fuentes de ingresos de los hogares (INE, 2018b)

Los ingresos provenientes de otras fuentes, percibidas tanto por las personas como por los hogares, se circunscriben al concepto de ingreso corriente. Los ingresos provenientes de fuentes laborales o de la ocupación están referidos a los ingresos netos, es decir, excluyen

descuentos legales e impuestos, y distinguen aquellos cuya fuente de origen es el trabajo dependiente de aquellos cuyo origen es el trabajo independiente (INE, 2018b).

La Encuesta Suplementaria de Ingresos cuenta con 5 secciones (INE, 2017):

- **Sección de Identificación del Hogar y antecedentes generales del Jefe de Hogar:** Esta sección contiene preguntas que permiten identificar el tipo de estrato, región y comuna del hogar, edad del jefe de hogar, sexo, nivel educacional entre otros.
- **Sección B:** Esta sección permite caracterizar la actividad principal del jefe de hogar, y contiene preguntas tales como el grupo de ocupación, categoría internacional del empleo, tipo de contrato, rama de actividad económica entre otros.
- **Sección C:** Esta sección contiene solo dos preguntas y están destinadas a cuantificar las horas trabajadas por el jefe de hogar.
- **Sección E:** Esta sección muestra datos acerca de la búsqueda de empleo del jefe de hogar, caracterizando los datos del empleo anterior.
- **Sección D:** Esta sección permite caracterizar y cuantificar los ingresos del jefe de hogar, y tiene preguntas que permiten identificar la fuente de ingresos, el uso de subsidios, becas o algún beneficio estatal, entre otros.

2.4 Conceptos importantes presentes en encuestas socioeconómicas

2.4.1 Tasa de Desigualdad

La tasa de Desigualdad se refiere a la desigualdad económica y social en Chile y se obtiene por los datos de la Encuesta Casen. Esta tasa permite observar la relación con la desigualdad de otros países, además de mostrar cuál es la brecha socioeconómica del país, para tomar medidas que permitan disminuir esta brecha.

Para ello se utilizan ciertas herramientas que permitan cuantificar la desigualdad, estas son: El coeficiente de Gini, y el índice 10/10 y 20/20. El coeficiente de Gini es la medida de desigualdad que muestra cuánto se aleja la distribución de ingresos de un país respecto a una situación de perfecta igualdad de ingresos, mientras más cercano esté a 1 significa que hay

más desigualdad. El valor del indicador se deriva a partir de la "curva de Lorenz" y fluctúa entre 0 y 1, donde 0 significa que no existe desigualdad (todas las personas tienen el mismo nivel de ingresos) y 1 representa el mayor nivel de desigualdad posible (una persona tiene todos los ingresos del país y el resto, nada). El índice 10/10 corresponde al cociente entre el ingreso mensual promedio per cápita del 10% más rico de la población y el 10% más pobre, mientras que el índice 20/20 muestra esta misma relación, pero entre el 20% más rico y el 20% más pobre (Departamento Administrativo Nacional de Estadística (DANE), 2015).

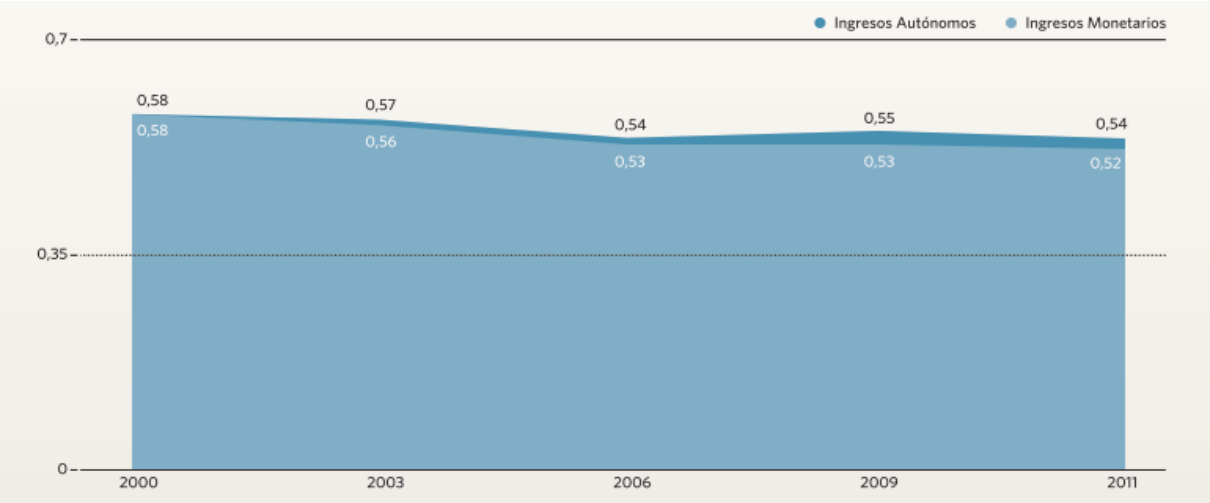

Figura 13. Ejemplo de coeficiente de Gini, tomado de (Departamento Administrativo Nacional de Estadística (DANE), 2015)

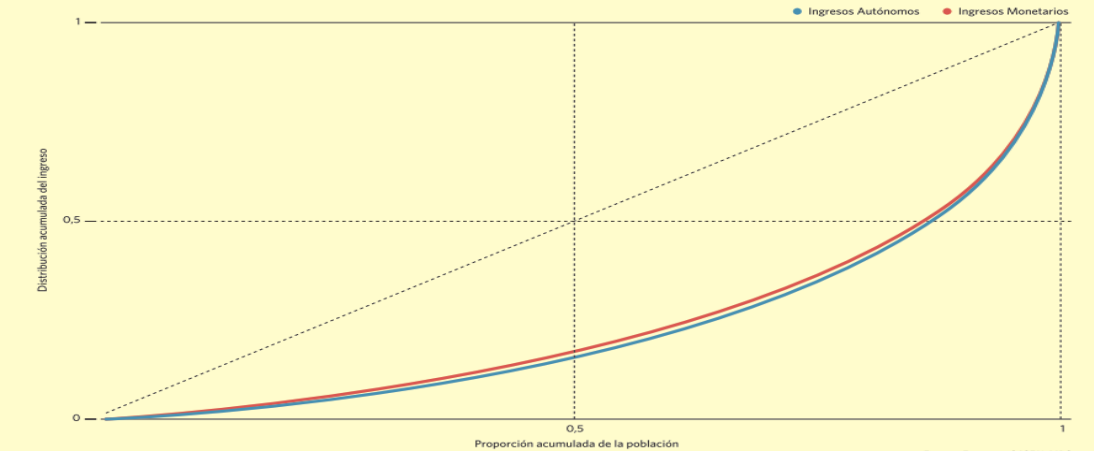

Figura 14. Ejemplo de curva de Lorenz, tomado de (Departamento Administrativo Nacional de Estadística (DANE), 2015)

En las figuras 13 y 14 se miden los ingresos autónomos y los ingresos monetarios según la encuesta CASEN. Se puede observar en la figura 14 que la distribución más igualitaria es de los ingresos monetarios (son más cercanos al valor 1), ya que en ellos se incluyen también los subsidios monetarios que el Estado entrega a las personas con ingresos más escasos. En la figura 13 se observa el comportamiento de la desigualdad con coeficientes de Gini obtenidos de la encuesta CASEN desde el año 2000 hasta el 2011. Se observa que el nivel de desigualdad no ha variado mucho, pero que el coeficiente de Gini es más bajo en los ingresos monetarios, indicando que los beneficios estatales ayudan a reducir la desigualdad.

2.4.2 Conmutación Laboral

La conmutación laboral se refiere a cuando las personas desempeñan su actividad laboral fuera de la región en la que residen. También es aplicable para sectores geográficos más pequeños, como provincias y comunas (Pino A. & Concha M., 2012).

La conmutación se ha convertido en un fenómeno global que además de su dimensión urbana, es también regional. Además, se reconoce que tanto sus formas como causas son cada vez más diversas, predominando causas como los costos, el tiempo y las distancias de transporte y nuevos sistemas de turno de trabajo.

Según la literatura (Pino A. & Concha M., 2012), en Chile se ha visto un claro ejemplo de conmutación regional en la región de Antofagasta. Las familias han optado por vivir en otra ciudad y dirigirse a trabajar a esta región, debido a la mejor calidad de vida que pueden obtener a partir de los ingresos provenientes de Antofagasta, ya que el costo de vida es más alto, por lo tanto los sueldos son mayores.

2.4.3 Brechas Socioeconómicas

En Chile, no todas las personas tienen la misma calidad de vida, por lo que hay diferencias entre diferentes grupos de personas, esta diferencia es denominada brecha socioeconómica. Las más importantes son (Fundación superación de la pobreza, 2005): brecha de ingresos, brecha educacional, brecha en salud, brecha en vivienda y en participación.

Las brechas de ingresos tienen como referencia la línea de pobreza; esto es el nivel de ingreso requerido para la adquisición de una canasta de bienes y servicios, para satisfacer las necesidades básicas. El indicador más tradicional de la brecha, es la medición del porcentaje de personas por debajo de la línea de pobreza.

La educación representa un aspecto básico de las políticas orientadas a reducir las desigualdades en la distribución de ingresos y superar la pobreza, mientras más equitativamente estén distribuidas las oportunidades de educación, más igualitaria será la distribución de los ingresos resultantes, por ello las brechas en educación generalmente contemplan temas de acceso a la educación y calidad de la educación.

Las brechas en salud van desde el acceso a los servicios de salud cuando se presentan urgencias, enfermedades crónicas o cuando es necesaria una consulta médica hasta el tipo de alimentación que consumen las personas, de la higiene ambiental donde comúnmente habitan y la seguridad de sus lugares de trabajo.

Las brechas en vivienda van desde la diferencia en el espacio de la vivienda por persona que la habita, hasta el grado de segregación en las ciudades.

Las brechas en participación se refiere a la frecuencia de participación en organizaciones, relacionado directamente con la confianza social.

Capítulo 3. Ambiente de Desarrollo

3.1 Herramientas de Desarrollo

3.1.1 MySQL

Es un sistema de gestión de bases de datos relacional desarrollado por Oracle Corporation y está considerada como la base datos de código abierto más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web. Es un sistema desarrollado bajo licencia dual: Licencia pública general/Licencia comercial, esto significa que MySql está desarrollado por una Empresa privada que posee el copyright de la mayor parte del código. Por esto existen varias versiones: Community, que es la que tiene licencia general, y versiones Enterprise, para empresas que quieran usarlo en iniciativas privadas.

Está desarrollado en su mayor parte en ANSI C y C++. Tradicionalmente se considera uno de los cuatro componentes de la pila de desarrollo LAMP y WAMP y es muy utilizado en aplicaciones web, como Wordpress, debido a que su motor no transaccional MyISAM es mucho más rápido en la lectura que en la concurrencia en la modificación (MySQL AB, 2005).

3.1.1.1 Características

- Amplio subconjunto del lenguaje SQL. Algunas extensiones son incluidas igualmente.
- Disponibilidad en gran cantidad de plataformas y sistemas.
- Posibilidad de selección de mecanismos de almacenamiento que ofrecen diferentes velocidades de operación, soporte físico, capacidad, distribución geográfica, transacciones.
- Transacciones y claves foráneas.
- Conectividad segura.
- Replicación.
- Búsqueda e indexación de campos de texto.

En este proyecto usaremos MySQL como motor de base de datos, principalmente por su gratuidad y por su amplio uso en desarrollo web, ya que el prototipo de datos estadísticos económicos de la región del Biobío será desarrollado con programación web.

3.1.2 Phyton

Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Una de sus principales características es que es un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico, es multiplataforma y tiene licencia de código abierto, por lo que existen muchas extensiones para diferentes funcionalidades (Molina, Loja, Zea, & Loaiza, 2016).

Por sus características se ha decidido usar este lenguaje de programación en el prototipo, usándolo principalmente para la carga de datos al servidor y para realizar las consultas a la base de datos.

3.1.3 Django

Es un framework de desarrollo web de código abierto, que está escrito en Python y que respeta el patrón de diseño modelo-vista-template. La figura 15 muestra el funcionamiento de este paradigma.

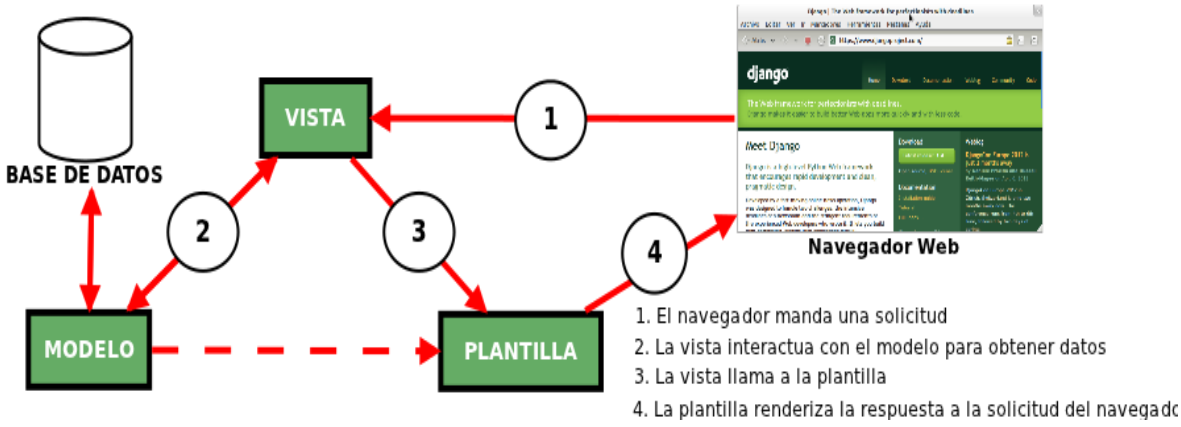

Figura 15. Patrón de diseño Modelo-Vista-Template de Django, tomado de (Infante Moreno, 2012)

Como lo muestra la figura 15, primero el navegador manda una petición, que es procesada por la vista, luego esta interactúa con el modelo para que este entregue los datos, finalmente, la vista llama a la plantilla que muestre la respuesta en el navegador.

La meta fundamental de Django es facilitar la creación de sitios web complejos (Molina et al., 2016). Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio "No te repitas" (DRY, del inglés Don't Repeat Yourself). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos. Otras características de Django son (Wikipedia, 2012):

- Un mapeador objeto-relacional.
- Aplicaciones "enchufables" que pueden instalarse en cualquier página gestionada con Django.
- Una API de base de datos robusta.
- Un sistema incorporado de "vistas genéricas" que ahorra tener que escribir la lógica de ciertas tareas comunes.
- Un sistema extensible de plantillas basado en etiquetas, con herencia de plantillas.
- Un despachador de URLs basado en expresiones regulares.
- Un sistema "middleware" para desarrollar características adicionales; por ejemplo, la distribución principal de Django incluye componentes middleware que proporcionan cacheo, compresión de la salida, normalización de URLs, protección CSRF y soporte de sesiones.
- Soporte de internacionalización, incluyendo traducciones incorporadas de la interfaz de administración.
- Documentación incorporada accesible a través de la aplicación administrativa (incluyendo documentación generada automáticamente de los modelos y las bibliotecas de plantillas añadidas por las aplicaciones)

3.1.4 JavaScript

Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente, permitiendo el mejoramiento de la interfaz de usuario y páginas web dinámicas. También es usado en aplicaciones externas a la web, por ejemplo en documentos PDF y aplicaciones de escritorio (como por ejemplo, los widgets). Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. En la actualidad es utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como AJAX (Flanagan, 2006)

3.1.5 JSON

Es un acrónimo de JavaScript Object Notation y es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente (Bray, 2017)

3.1.6 savReaderWriter

Es una librería de Python que permite leer y escribir archivos en el sistema SPSS, esto incluye archivos .SAV o .ZSAV. Funciona tanto en plataforma Windows como en Linux, Mac OS y Solaris (Roskam, 2013).

3.1.7 Heatmap.js

Es una librería de JavaScript que permite mostrar heatmaps. Desarrollada por Patrik Weid, permite mostrar información de tres dimensiones, usando el largo, el alto y el color. Es

posible juntarla con otras librerías de JavaScript y posee un plugin para poder usarlo con mapas geográficos. Posee verisón gratuita y de pago[5].

3.1.8 Leaflet.js

Es una librería de código abierto desarrollado por Vladimir Agafonkin en colaboración con la fundación OpenStreetMap y permite mostrar mapas tanto en sitios web de escritorio como en móviles, con características de interacción, visuales, de personalización y de rendimiento[6].

3.2 Modelo de proceso de desarrollo

Un modelo es una abstracción que muestra las principales características de un proceso, en este caso, se modela el proceso de desarrollo de software que consta de 4 actividades: Análisis, Diseño, Construcción y Pruebas.

El modelo de prototipado evolutivo, es un modelo que permite que todo un sistema o parte de él se construyan rápidamente para aclarar aspectos del mismo y cuando todas las aclaraciones estén completas se procede a desarrollar el producto final. Este modelo muestra una solución que se adapta a las características de este proyecto, ya que proporciona los siguientes beneficios para un equipo de desarrollo de una sola persona:

- Se pueden obtener resultados parciales funcionales
- Se acomoda a la exploración e incorporación de nuevas funcionalidades y requisitos
- Hay menor riesgo de fracaso total del proyecto

[5] Documentación y API de heatmap.js disponible en: https://www.patrick-wied.at/static/heatmapjs/docs.html
[6] Documentación y API de leaflet disponible en https://leafletjs.com/reference-1.3.4.html

Capítulo 4. Diseño del Prototipo

4.1 Requerimientos

Los requerimientos para el desarrollo del prototipo de plataforma web para datos estadísticos socioeconómicos de la región del Biobío no son muchos, ya que es un prototipo y se enfocan solamente bajo el contexto del uso de visual analytics. Estos son:

- El prototipo debe mostrar información utilizando tipos de visualización de datos estudiados.
- El prototipo debe ser una aplicación web.
- La información que muestre el prototipo será seleccionada con la consulta a expertos en el área de economía.
- El prototipo debe permitir la interacción del usuario con las visualizaciones.

4.2 Software existente para Visual Analytics

4.2.1 SAS

Es un Paquete de Software creado por SAS Institute. Fue desarrollado principalmente como un método para la organización y control de grandes bases de datos. SAS fue diseñado de manera tal que permite la recolección, transformación, análisis y reporte de datos (SAS Institute Inc., 2018a).

Además, permite soluciones de software a gran escala para áreas como administración, gestión de recursos humanos, gestión financiera, inteligencia de negocios y más.

SAS cuenta con muchos productos, entre los principales se encuentran: SAS Visual Analytics, SAS Statistics, SAS Model Manager y SAS Detection and Investigation

SAS Visual Analytics es el producto que utiliza visual analytics en el entorno SAS, sus principales características son (SAS Institute Inc., 2018b):

- Exploración de datos
- Generación de Reportes

- Técnicas de Visualización de Datos avanzadas
- Tableros de instrumentos interactivos
- Analítica de Ubicación
- Flexibilidad en la implementación de los datos, ya sea en la nube o en medios físicos
- Análisis de Texto

4.2.2 Microsoft Power BI

Power BI es un conjunto de herramientas de análisis empresarial. Permite la conexión a diferentes orígenes de datos, preparación de datos, generación de análisis ad hoc, informes que luego se publican para la organización en la Web y en dispositivos móviles. Fue lanzado el 24 de Julio del 2015 y cuenta con tres versiones de pago: Power BI Desktop, Power BI Pro y Power BI Premiun (Microsoft, 2018a).

Las características principales de Power BI son (Microsoft, 2018b):

- Conexión con diferentes orígenes de Datos.
- Interfaz para el modelamiento y preparación de datos
- Análisis avanzados con la sencillez de Excel
- Visualizaciones de datos interactivas, tanto integradas como de creación propia
- Inserción de informes de Power BI en sitios Web existentes

4.2.3 Tableau

Es un Paquete de productos software para la visualización de datos interactivos enfocados en el Business Intelligence. Fue desarrollado por Tableau Software desde el año 2003 y cuenta con dos principales productos: El Tableau Creator, que permite crear visualizaciones de manera interactiva a partir de bases de datos, el Tableau Server que cuenta con

funcionalidades para difundir datos en la empresa y el Tableau Online que tiene funcionalidades similares al Tableau Server pero todo es hecho en la nube.

El Tableau Creator consta de dos productos: El Tableau Desktop y el Tableau Prep. Las Características principales de Tableau Desktop son (Tableau Software, 2017a):

- Interfaz que permite el análisis visual en tiempo real utilizando dashboards interactivos
- Conexión a datos físicos o en la nube, tanto si se trata de Big data, bases de datos SQL, hojas de cálculo, o aplicaciones en la nube
- Creación de Mapas interactivos utilizando códigos postales para un mapeo de más de 50 países en el mundo.

Las características principales del Tableau Prep son (Tableau Software, 2017c):

- Posee tres vistas coordinadas para poder preparar los datos para la interacción
- Edición inmediata de datos o tipos de datos sin importar en dónde esté guardado el dato.
- Conexión a datos físicos o en la nube, tanto si se trata de Big data, bases de datos SQL, hojas de cálculo, o aplicaciones en la nube.

Las principales características de Tableau Server son (Tableau Software, 2017d):

- Otorga la posibilidad de compartir los datos con la organización en un entorno de confianza
- Permite compartir las visualizaciones y datos en tiempo real
- Conexión a diferentes tipos de fuentes de datos empresariales como Oracle, Microsoft SQL Server, Cloudera Hadoop entre otros
- Administración centralizada de metadatos y reglas de seguridad
- Integración de Protocolos de Seguridad actuales, como Active Directory, Kerberos, OAuth u otros.
- Flexibilidad en el almacenamiento de datos

Las principales características de Tableau Online son (Tableau Software, 2017b):

- Permite compartir y colaborar a través de la nube
- Interacción, edición y creación en la web
- Conexión con cualquier tipo de datos en cualquier lugar

4.2.4 SAP Lumira

SAP Lumira es la solución del paquete de software SAP que permite realizar visualizaciones de datos de forma automatizada y personalizada. Fue creada como una actualización y reemplazo del software Visual Intelligence de SAP. El objetivo principal de SAP con este software es crear una plataforma que de soporte a la metodología story-telling a los informes utilizando las visualizaciones (Garmendia, 2016).

Las principales características de SAP Lumira son (Axalpha Consulting, 2018):

- Puede adquirir datos desde distintas fuentes
- Potencia la inversión realizada en datos analíticos
- Combina datos de distintas fuentes sin usar código
- Depura, agrupa y añade cálculos
- Visualizaciones personalizadas
- Vistas disponibles en SAP Hana y cloud
- Puede compartir presentaciones con usuarios móviles y web

4.2.5 D3 plus

Es una librería que actúa como una extensión de la popular librería D3 y está hecha en JavaScript. La versión más reciente es la versión 2.0[7]. Con D3 plus se pueden lograr visualizaciones con pocas líneas de código, sólo es necesario agregar los parámetros específicos para el tratamiento de los datos y D3plus se hace cargo de las cosas genéricas, como la ubicación, las animaciones, los eventos del mouse y la asignación del color. Cuenta

[7] Documentación de D3plus v2.0 disponible en: https://d3plus.org/docs/

con una gran cantidad de ejemplos y una gran comunidad que hace seguimiento de errores y solicitudes de características (DataWheel, 2018).

4.3 Herramientas existentes y utilidad de la plataforma propuesta

La mayoría de las herramientas disponibles permiten construir visualizaciones a partir de datos de manera interactiva, por tanto podrían ser utilizadas para implementar la plataforma propuesta. En este sentido, D3plus surge como mejor opción, dado que 1) es de uso gratuito; y 2) proporciona un API y documentación sobre su uso desde código.

También podría surgir la interrogante de ¿por qué construir una plataforma si existen estas herramientas? Al comparar las características de las herramientas investigadas con las características del Prototipo de Plataforma Web para la visualización de datos Socioeconómicos de la región del Biobío, se pueden identificar dos aspectos claves que los diferencian, estos son:

4.3.1 El Propósito

Si bien es cierto, todas las herramientas mencionadas anteriormente utilizan Visual Analytics, al igual que el prototipo, éstos tienen propósitos diferentes. Mientras que el propósito de estas herramientas es probar diferentes análisis de datos en tiempo real, el propósito del prototipo es el de una plataforma web donde se puedan visualizar los datos de encuestas socioeconómicas importantes. Esto requiere de un pre procesamiento de los datos, que incluye un estudio de las encuestas, y una selección de visualizaciones de datos que se acomode a la información a mostrar.

4.3.2 Capacidades del Usuario Final

El usuario final destinado para las herramientas estudiadas es notoriamente distinto al usuario final destinado para el prototipo. El usuario final para estas herramientas debe tener necesariamente conocimientos, tanto del uso del entorno donde se ejecuta el software, como del uso del software mismo, además de los conocimientos de la lógica del negocio y del contexto de la información para realizar un análisis con visual Analytics, mientras que el

usuario final para el prototipo de plataforma web no requiere necesariamente de estos conocimientos, por el contrario, este prototipo está diseñado para que cualquier persona pueda obtener la información solo con verlo.

4.4 Diseño Arquitectónico

Para el diseño arquitectónico se muestra primero la interacción de las tecnologías a utilizar. Se utiliza como base el modelo de Cliente-servidor, ya que el prototipo será una aplicación web. En la siguiente figura se observan los módulos que actúan en el prototipo.

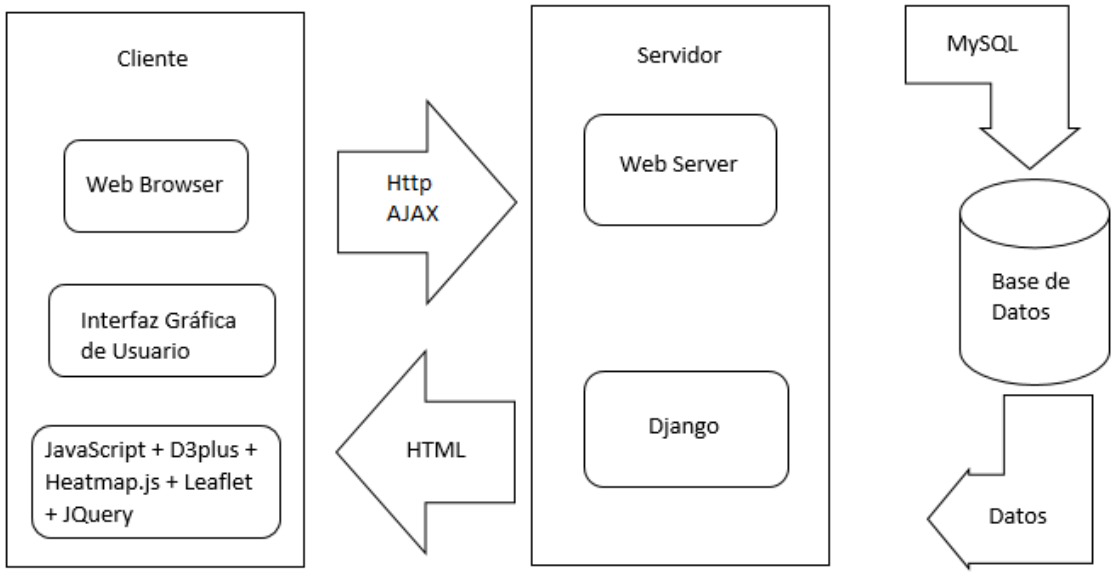

Figura 16. Diseño Arquitectónico del Prototipo de plataforma web para datos estadísticos socioeconómicos de la región del Biobío

En la figura 16 se puede observar los principales bloques que participan en el prototipo. Se aprecia que existe transferencia de datos entre el cliente y el servidor y entre el servidor y la base de datos y que el bloque del servidor tiene en su interior alojado el framework Django.

Además se observan las diferentes herramientas de desarrollo, tales como HTML, D3plus, Heatmap.js, Leaflet y JavaScript.

La siguiente figura presenta los principales módulos lógicos del prototipo y muestra cómo interactúan entre sí y con la base de datos.

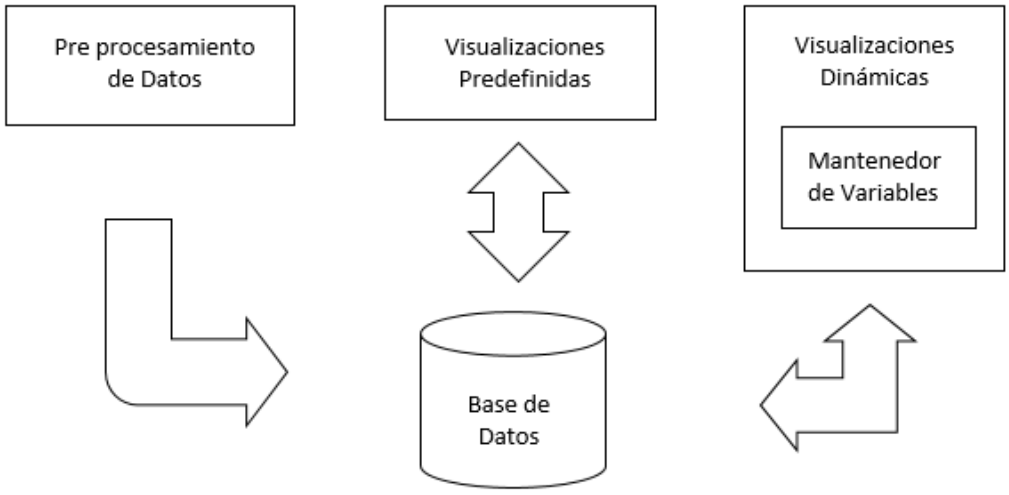

Figura 17. Diseño lógico de los módulos del Prototipo de plataforma web para datos estadísticos socioeconómicos de la región del Biobío

Como se puede observar en la figura 17, son tres los principales módulos del prototipo: el módulo de pre procesamiento de datos, cuyo flujo de datos con la base de datos es unidireccional, y los módulos de visualizaciones predefinidas y dinámicas que tienen cuya interacción es bidireccional. Se observa también que el módulo de visualizaciones dinámicas cuenta con un sub módulo de mantenedor de variables de las encuestas. En el capítulo 5 se entregan mayores detalles respecto de la implementación de cada uno de estos módulos.

4.5 Diseño detallado

4.5.1 Diagrama de Componentes

En este diagrama se observa de manera más detallada el funcionamiento del prototipo, y en dónde interviene cada componente.

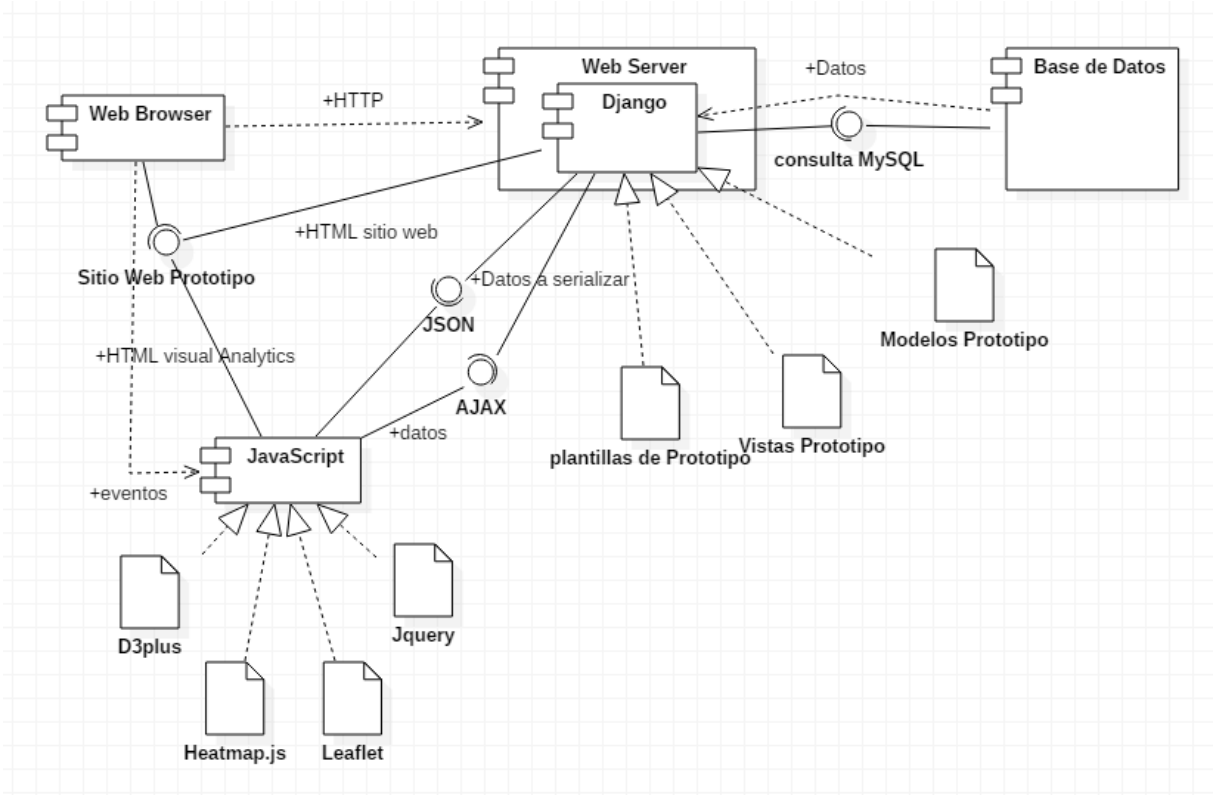

Figura 18. Diagrama de Componentes del Prototipo de plataforma web para datos estadísticos socioeconómicos de la región del Biobío

La figura 18 muestra con más detalle el funcionamiento del Prototipo de plataforma web para datos estadísticos socioeconómicos de la región del Biobío. El componente Web Browser es el que envía, mediante protocolo HTTP, la carga de la página en donde se encuentra el Prototipo. Esto lo recibe el componente Web Server, que tiene instalado el framework Django, quien finalmente realiza la consulta en el lenguaje MySQL a la base de datos. La

base de datos entrega los datos correspondientes y Django, bajo su estructura Modelo-Vista-Template representados en Modelos Prototipo, vistas Prototipo y plantillas Prototipo respectivamente, es quien envía el HTML de la página web y también los datos serializados a JSON para que JavaScript pueda entenderlos en el lado del Cliente. Finalmente en JavaScript y usando D3plus se pueden generar las visualizaciones en HTML para que se cargue completamente el sitio web del Prototipo y se envíen como respuesta al Web Browser.

4.6 Interfaz Gráfica de usuario

Para la interfaz gráfica de usuario se ha determinado un layout que tiene una barra lateral de botones y una barra para el header de la página. En la barra lateral están las opciones de Encuesta CASEN, NENE y NESI y deben de mostrar las opciones para la información que se quiere mostrar. Se usará el color blanco contrastado con el color gris como se puede observar en la figura 19.

Figura 19. Layout de interfaz gráfica de usuario del prototipo de plataforma web de datos estadísticos socioeconómicos de la región del Biobío

Capítulo 5. Detalles de Implementación

En este capítulo se describe la implementación de los módulos que componen el prototipo propuesto, explicando su propósito, lógica de procesamiento, modelo de datos asociado, detalles de código y visualización generada. Mayores detalles sobre el código implementado se pueden encontrar en el anexo 2 de este informe.

5.1 Módulo de Pre procesamiento de Datos

El propósito de este módulo es de transformar los datos de las encuestas para poder ser consumibles por el prototipo. Estos datos están disponibles en formato .SAV que es una extensión de archivos guardados por el sistema SPSS, que es un software estadístico de pago creado por IBM y que utilizan tanto el INE como el Ministerio de Desarrollo Social.

La siguiente figura muestra un ejemplo de la lectura de un archivo .SAV a través de un software especializado

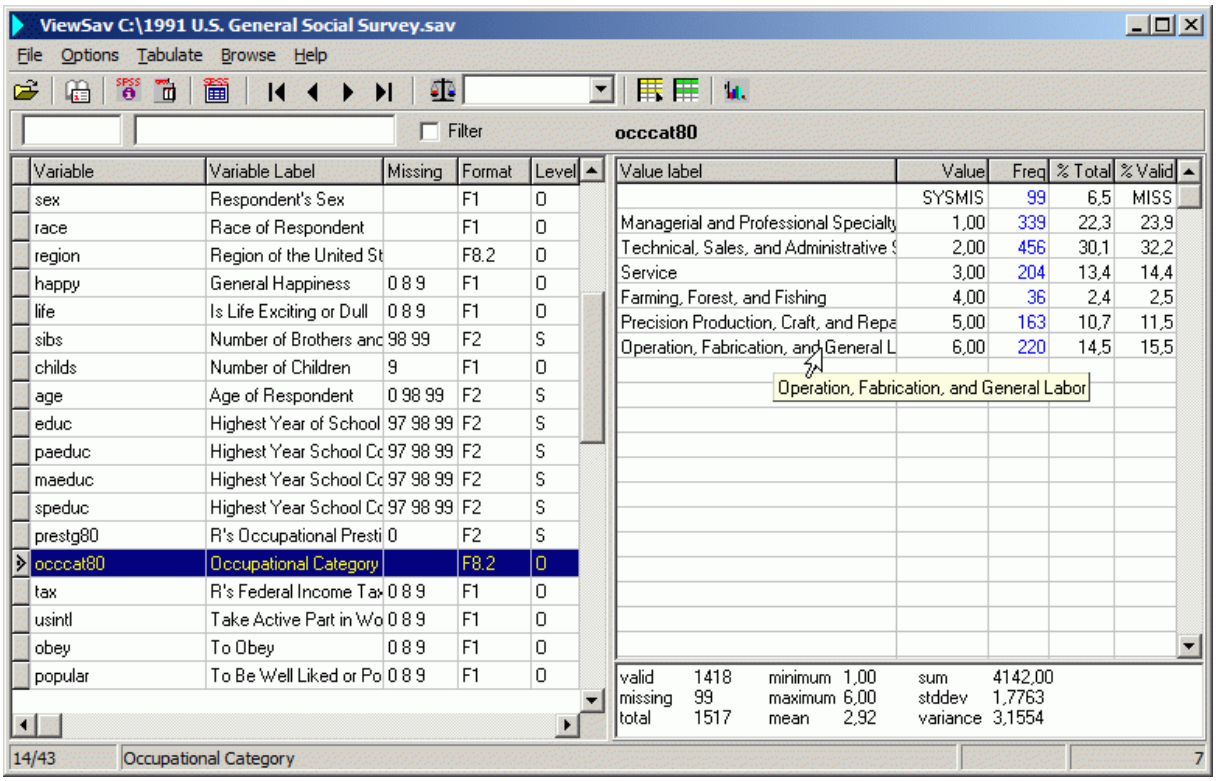

Figura 20. Ejemplo de lectura de un archivo .SAV por un software especializado

Se puede observar en la figura 19 que las variables se muestran a la izquierda de la pantalla, con el código de la variable, el nombre, y los valores se muestran a la derecha. Para obtener los datos se necesita leer un archivo .SAV, pero además cuando se obtengan los datos se deben guardan en la base de datos para que puedan ser consumidos por el prototipo. Por lo que se deben seguir etapas extras para obtener el resultado deseado.

5.1.1 Etapas

Para obtener los datos necesarios para ser consumidos por el prototipo se determinaron etapas que se muestran en la figura 21.

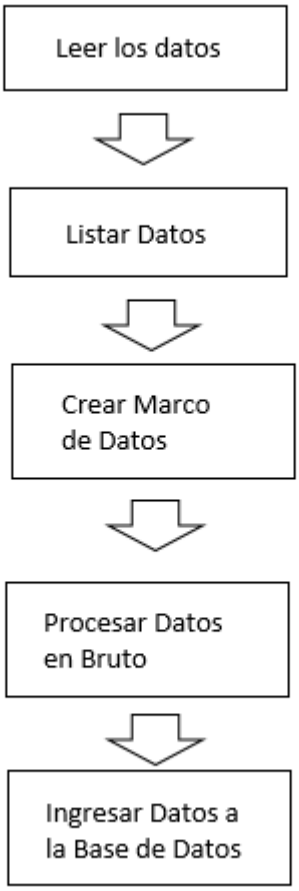

Figura 21. Esquema de los pasos a seguir para el pre procesamiento de datos

Como muestra la figura 21, primero se deben leer los datos, para ello se utiliza la librería de Python savReaderWriter que puede leer los archivos .SAV, sin embargo, el resultado de esta lectura es un objeto, y para poder leerlo es necesario listar estos datos, para después crear un marco de datos donde se ordenen los datos por la variable a la que pertenecen, esto se puede lograr con la librería de Python pandas, que permite analizar listas y crear tablas. Después hay que procesar los datos en bruto, ya que algunos datos, como los nulos vienen con un valor predeterminado que no es comprensible para ingresarlo a la base de datos. Finalmente se ingresan los datos a la base de datos a través de los modelos de Django.

La siguiente figura muestra un ejemplo sencillo de cómo son los datos en bruto y como son los datos cuando quedan listados y ordenados antes de ser procesados.

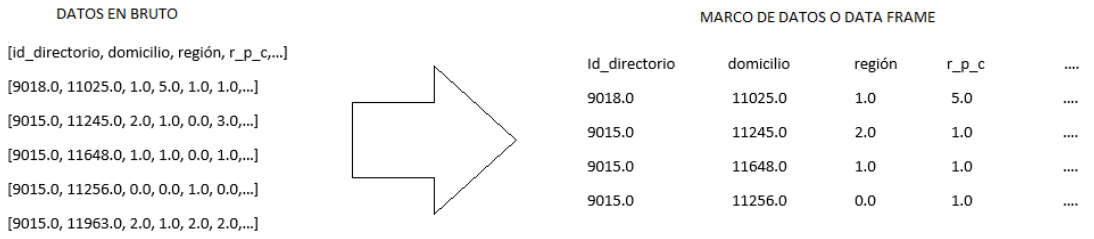

Figura 22. Ejemplo de la transformación de los datos antes de ser procesados

Como se puede observar los datos en bruto son un conjunto de arreglos, obtenidos después de leerlos del archivo .SAV. Para obtener el resultado mostrado en el marco de datos, se siguen las tres primeras etapas de la figura 21. La siguiente figura muestra las líneas de código necesarias para lograr esto.

```
raw_data = spss.SavReader(filename, returnHeader=True, rawMode=True)
raw_data_list = list(raw_data)
data = pd.DataFrame(raw_data_list)
```

Figura 23. Código en Python para leer datos desde un archivo .SAV

Como muestra la figura 23 el método SavReader devuelve los datos en bruto, para ello es necesario cambiar el parámetro *rawMode* a true, esto significa que para los campos nulos se devuelve un valor -numérico y los strings están en un formato de múltiplo de 8 bits. Esto se realiza para una lectura más rápida de los datos. Además se modifica el parámetro *returnHeader* a true, para que se pueda identificar a qué variable pertenece cada dato.

Luego, para analizar los datos y ordenarlos, de manera que queden listos para ser guardados en la base de datos, se crea una lista con los datos obtenidos y luego se usa el método *DataFrame* de pandas para obtener los datos de manera tabular.

Para ingresar los datos a la base de datos se usa Django, donde es posible crear registros de MySQL a través de clase *Model* de Django,para ello se importa la clase de Django que tiene

el modelo del encuestado que se necesita, por esta razón los modelos de Django se crean antes de ingresar los datos a la base de datos.

5.1.2 Modelo de Datos

El modelo de datos que se utiliza en este módulo lo muestra la figura

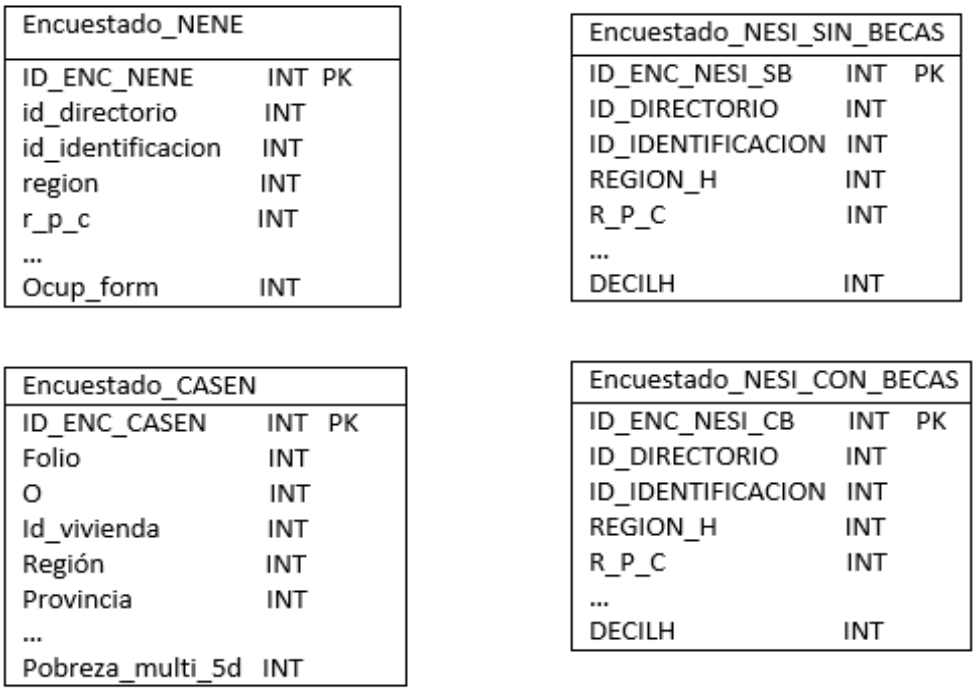

Figura 24. Modelo físico de datos que interactúa en el módulo de Pre procesamiento de datos

Como muestra la figura 24, no existen relaciones entre los resultados de las encuestas, por lo que se presentan sin ningún tipo de conexión. Se observa que se guardan en diferentes tablas a los encuestados de la NESI, pues están disponibles separados por los que tienen beneficios estatales y los que no, por ello las tablas tienen solo este atributo que las diferencia entre ellas.

5.2 Módulo de Visualizaciones Dinámicas

El módulo de visualizaciones dinámicas tiene el propósito de proporcionar una forma para que el usuario pueda generar sus propias visualizaciones, donde se pueda seleccionar las

variables que quiera visualizar dinámicamente. Este módulo tiene un sub módulo, que es un mantenedor de variables.

5.2.1 Mantenedor de variables de encuestas

Para poder visualizar un Treemap de manera dinámica es necesario que el usuario seleccione la variable que quiere visualizar, sin embargo, las variables no son conocidas por el usuario y mucho menos sabe qué significa cada una de ellas. Las variables están representadas con un código que solamente puede ser traducido utilizando los catálogos de variables proveídos por los organismos encuestadores. La siguiente figura muestra un pequeño extracto del catálogo de variables de la encuesta NENE.

Variable	Nombre variable	Cód.	Nombre del código
nivel	Identifica el nivel educacional más alto aprobado, agrupando a las personas que terminaron el nivel, así como las que no lo terminaron	0	Nunca Estudió
		1	Sala Cuna
		2	Kínder
		3	Básica o Primaria
		4	Media Común
		5	Media Técnico Profesional
		6	Humanidades
		7	Centro Formación Técnica
		8	Instituto Profesional
		9	Universitario
		10	Post títulos
		11	Magíster
		12	Doctorado
		14	Normalista
		999	Nivel Ignorado
termino_nivel	Identifica el cumplimiento de todos los requisitos para obtener certificación del nivel educacional declarado como el más alto	1	Sí
		2	No
est_conyugal	Señala el estado conyugal o civil de la persona encuestada	0	No corresponde responder
		1	Casado(a)
		2	Conviviente
		3	Soltero(a)
		4	Viudo(a)
		5	Separado(a) de hecho o Anulado
		6	Divorciado(a)

Figura 25. Extracto de catálogo de variables NENE

La tabla de la figura 25 está dividida por 4 columnas. Empezando por la de la izquierda tenemos a la variable, que corresponde al código con el que se identifica la variable en las respuestas de los encuestados, después está la columna que describe el nombre de la variable, es decir lo que realmente significa, luego se tiene la columna llamada código, este será llamado valor para no entrar en confusiones. Esta columna muestra los valores que puede

tomar cada variable y en la última columna se tiene al nombre del valor. Por ejemplo se tiene la variable "est_conyugal", cuyo nombre hace referencia al estado conyugal del encuestado, puede tomar 7 valores que son números del 0 al 6, y estos valores tienen un significado también, que son descritos en la última columna.

Es por ello que se hace necesario tener un lugar donde se puedan guardar tanto los códigos como el significado de las variables, para que el usuario pueda entender lo que se selecciona. Además del listado de valores asociados a cada variable.

El mantenedor de variables cumple el propósito de administrar tanto las variables que se pueden encontrar en las encuestas, como de los valores que puede tomar cada variable si es que los tiene, ya que existen variables que no poseen valores como por ejemplo, las variables que conciernen a ingresos, muchas veces son valores numéricos que representan cantidad, o las variables en donde la respuesta es escrita. Debido al propósito del mantenedor, está pensado para ser usado por un administrador, no por el usuario final.

El mantenedor tiene ocho tareas correspondientes al CRUD de las variables y los valores.

5.2.1.1 Agregación

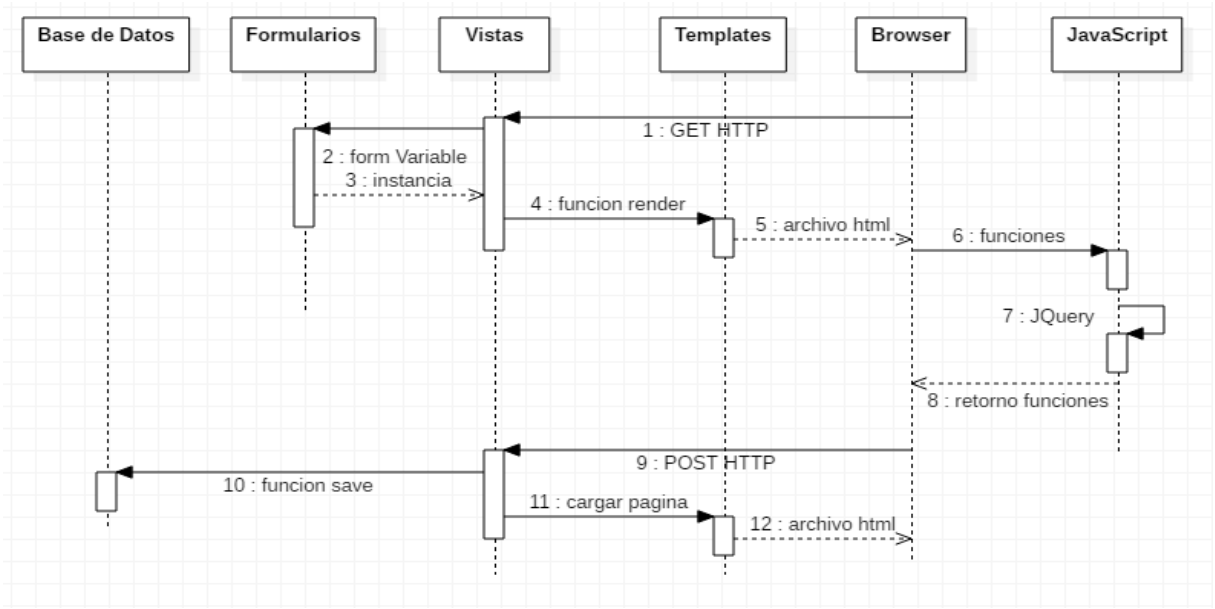

Figura 26. Diagrama de Actividad de agregación de variables y valores

El diagrama de la figura 26 muestra las actividades al momento de agregar una variable, pero para los valores sigue casi exactamente el mismo proceso. Comienza en el paso 1 con una solicitud GET, que llama a la función que está en la vista, esta pide una instancia del formulario en el paso 2 y se retorna a la vista la instancia en el paso 3. Luego, en el paso 4 la vista utiliza la función render que carga el template correspondiente, y este en el paso 5 lo muestra en el Browser del cliente. Sólo para el caso de los valores se sigue al paso 6 donde se cargan funciones JQuery para agregar dinámicamente valores adicionales y para validar los datos.

El paso 9 corresponde a cuando se realiza una solicitud POST a la vista, que usa la función save en el paso 10 para guardar los nuevos registros a la base de datos, luego en el paso 11 se carga una página de redireccionamiento utilizando el template que corresponda en el paso 12 para luego mostrarlo en el Browser.

5.2.1.2 Modificación

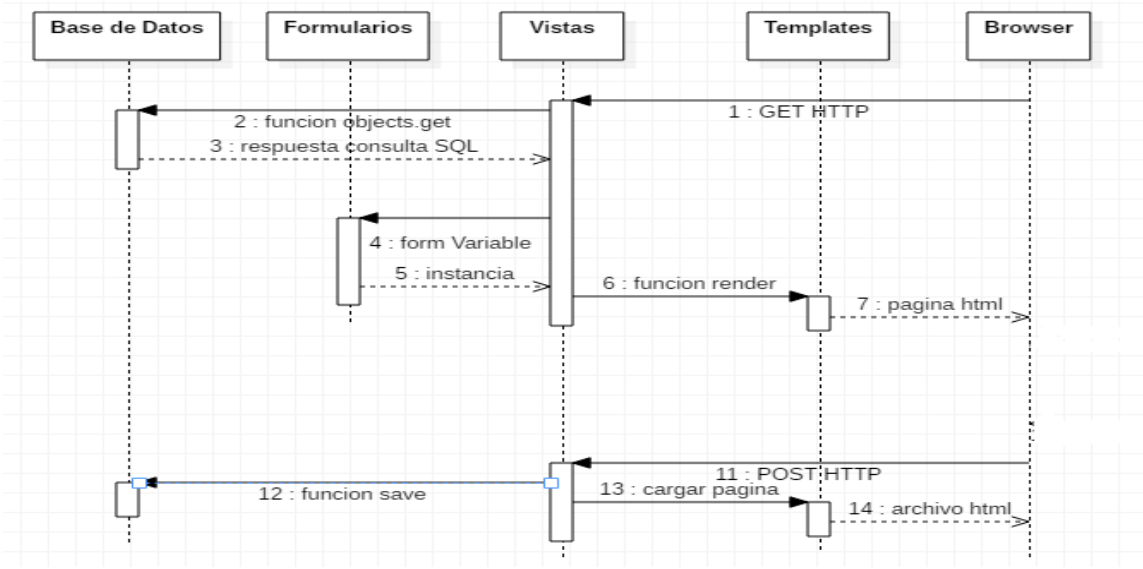

Figura 27. Diagrama de Actividad para la modificación de variables y valores

En el caso de la modificación de variables, como se muestra en la figura 26, el paso 1 comienza con una solicitud GET a la vista, que realiza una consulta en el paso 2 con la función *objects.get* que obtiene sólo el registro que se desea modificar. Después en el paso 4

se pide una instancia del formulario para mostrar los datos consultados, Luego la vista, en el paso 6 y 7 se realizan de la misma manera que en la agregación. También las actividades son las mismas en la solicitud POST, desde los pasos 11 al 14.

5.2.1.3 Eliminación

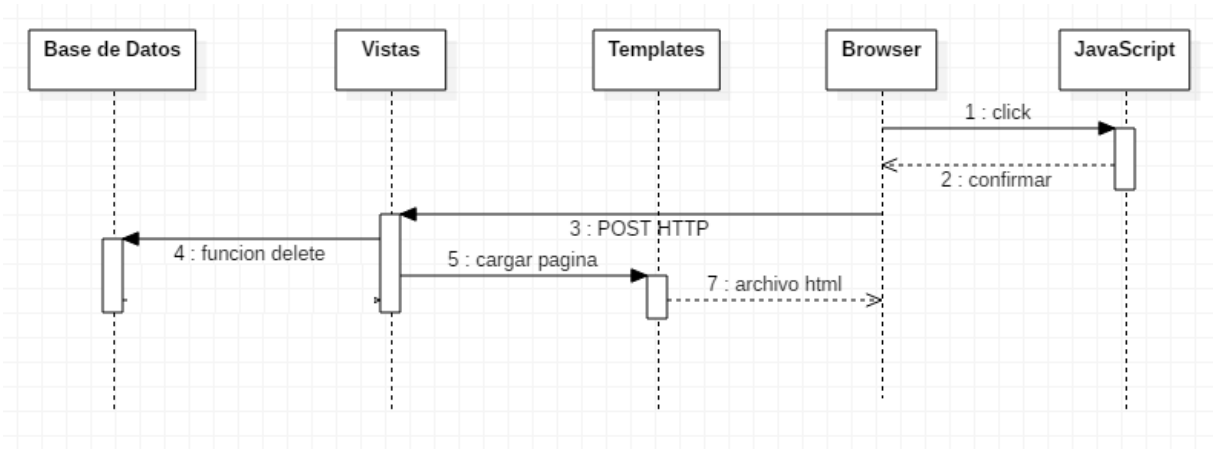

Figura 28. Diagrama de Actividad para la eliminación de variables y valores

Como muestra la figura 28, esta función está pensada para ser iniciada desde el usuario al presionar click en un botón para eliminar, se ejecuta en JavaScript una función de confirmación, en el paso 1, Luego de recibir la confirmación en el paso 2 se realiza una solicitud POST en el paso 3 a la vista, que ejecuta la función delete que elimina un registro de la base de datos en el paso 4. Luego el paso 5 y 7 es cargar la página de redireccionamiento que se muestra finalmente en el browser.

5.2.1.4 Listar

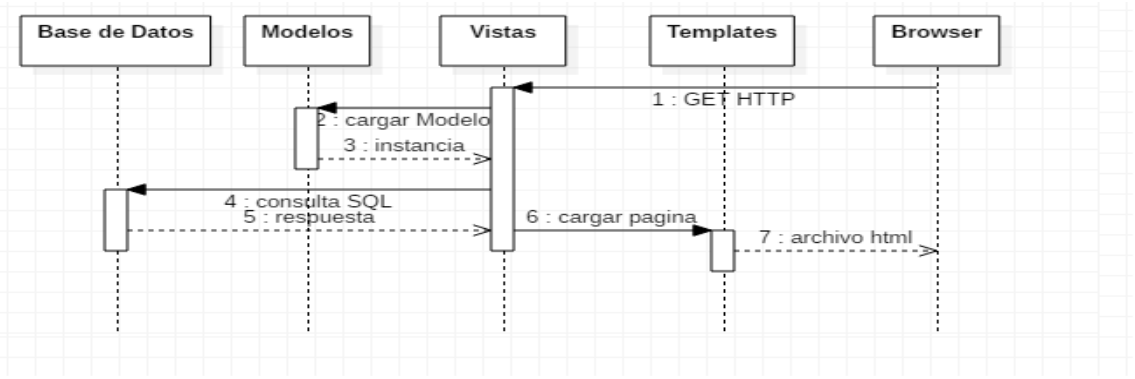

Figura 29. Diagrama de Actividad para listar variables y valores

Como se puede observar en la figura 29, esta actividad solo usa la solicitud GET, activada por el usuario en el paso 1, y la recibe la vista, que carga una instancia del modelo que corresponda, ya sea variable o valor, en el paso 2 y 3. Luego en el paso 4 y 5 se realiza una consulta para pedir todos los registros, y finalmente, en el paso 6 y 7 se carga el template y se muestra en el browser.

5.2.1.5 Resultados de visualizaciones

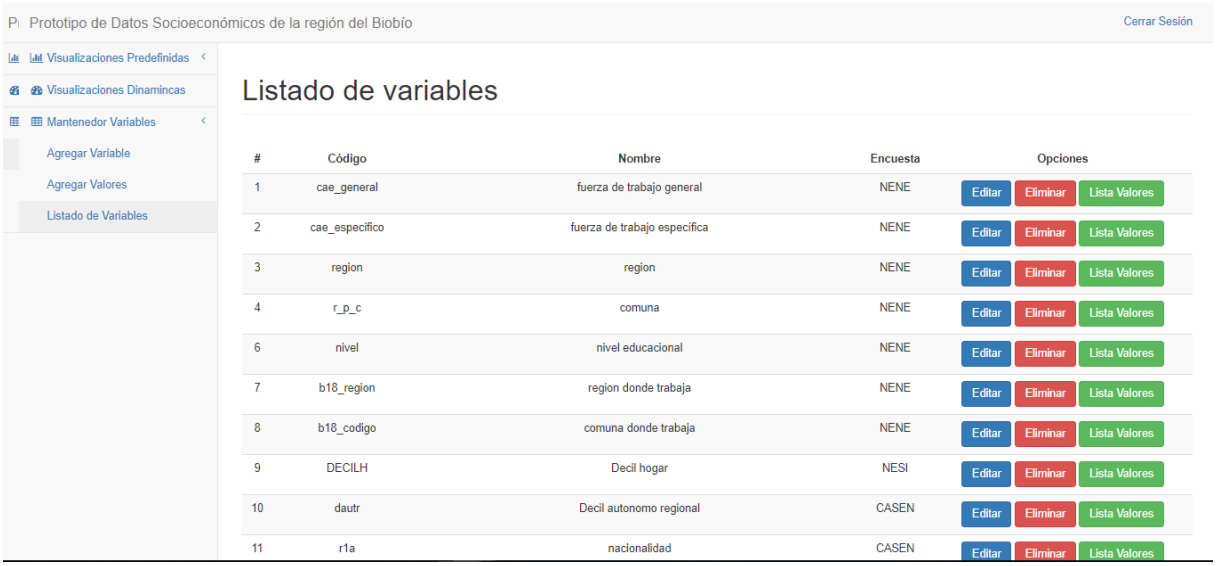

Figura 30. Resultado del mantenedor de variables

Como se puede observar en la figura 30 está el listado de variables, se pueden observar los botones para editar, eliminar, y lista de los valores de la variable. En la siguiente figura se muestra

5.2.2 Módulo de Visualizaciones de Treemap Dinámicas

Como se explicó en la sección 5.2, el módulo de visualizaciones tiene el propósito de permitir al usuario generar visualizaciones de manera dinámica. Para ello es necesario que el usuario pueda seleccionar las variables para generar la visualización que será un Treemap. Se llamará a esta parte del módulo como el "selector de variables" y constará de un formulario que contiene campos para generar dos niveles de visualización, es decir, se podrán seleccionar

dos variables para generar una visualización que vincula la primera variable seleccionada con la segunda. La siguiente figura muestra el funcionamiento del módulo de visualizaciones dinámicas.

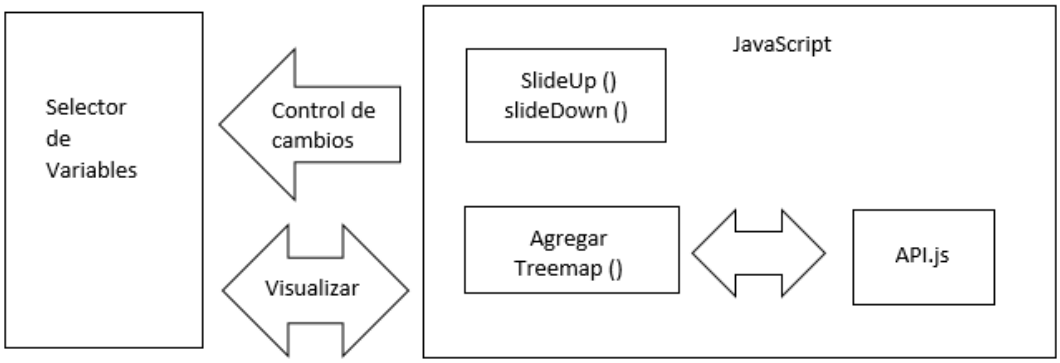

Figura 31. Esquema del funcionamiento del módulo de visualizaciones dinámicas

Como muestra la figura 31 la visualización se logra usando JavaScript. Con funciones de JQuery como *slideUp* y *slideDown* se realiza un control en los cambios que se generan al seleccionar una variable y también cuando se realiza el click para iniciar la visualización. Cuando esto ocurre se llama a la función *agregar Treemap* que retorna finalmente la visualización a través de un prototipo de API que se encarga de comunicarse con las librerías D3 plus, heatmap.js y Leaflet. Las principales actividades que involucran el proceso de visualizar dinámicamente y las interacciones entre ellos se pueden observar en la siguiente figura.

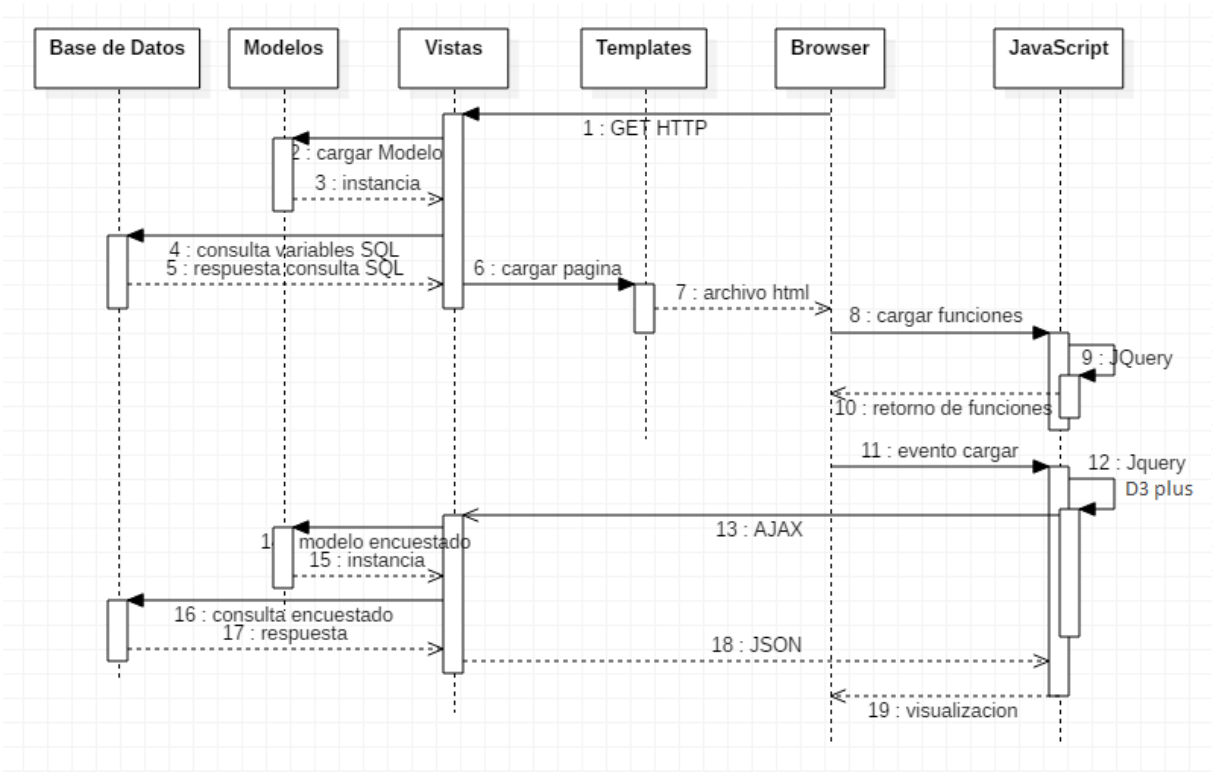

Figura 32. Diagrama de Actividad para las visualizaciones Dinámicas

Como se muestra en la figura 32, primero se realiza la carga de las variables para mostrarlas en el selector de relaciones, esto empieza en el paso 1 con la solicitud GET y sigue el mismo procedimiento ya visto en el mantenedor de variables hasta el paso 7, Luego en los pasos 8 al 10 se cargan funciones JavaScript para controlar los eventos de cambio en las variables a elegir. En el paso 11 es cuando se desea cargar el Treemap, por lo que se llama a la función agregar Treemap que se conecta al API haciendo una llamada para crear un Treemap. En la API se ejecuta una petición AJAX usando JQuery que se conecta directamente con la vista que realiza la carga del modelo que corresponda en los pasos 14 y 15 y también la consulta a la base de datos en el paso 16 y 17, luego se serializan los datos a JSON y se envían a JavaScript, en el paso 18, para que cree el Treemap usando D3 plus y se visualice finalmente en el paso 19.

5.2.3 Modelo de Datos

El modelo de datos que involucra crear Treemaps consta de dos partes: La primera es la que se observa en la Figura 24, y la segunda es la que se presenta en la siguiente figura y son los datos que usa el selector y el mantenedor de variables.

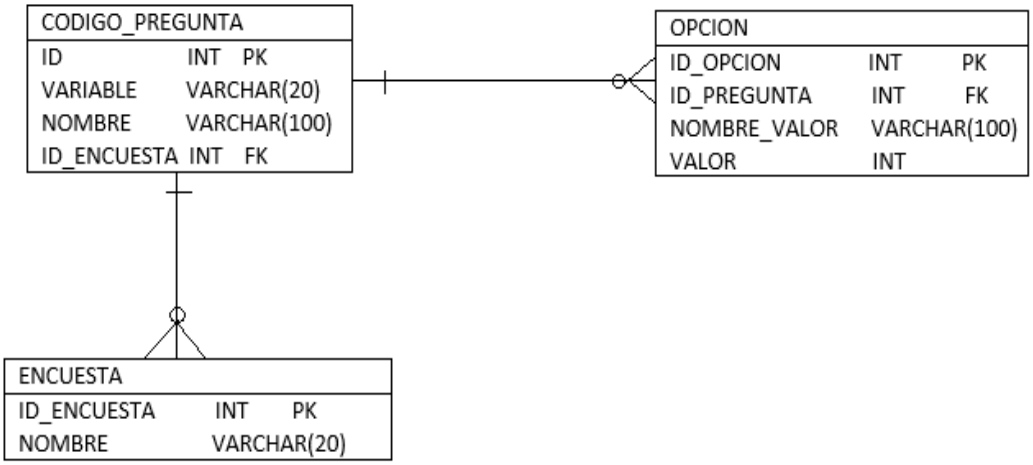

Figura 33. Modelo de datos para el selector de datos

Como se puede observar hay 3 tablas. La tabla Encuesta guarda los datos de las encuestas, La tabla Codigo_pregunta guarda los datos de la variable, como el nombre, y el código de la variable. Cada variable puede tener un número indeterminado de valores que puede ir desde 0 a n, estos se guardan en la tabla opción, que contiene el nombre del valor y el valor en sí. Para poder ingresar estos datos es necesario tener los catálogos de los códigos de variables y sus valores. Estos están disponibles en la página del INE y del Ministerio de Desarrollo Social respectivamente.

5.2.4 Modelo de Análisis

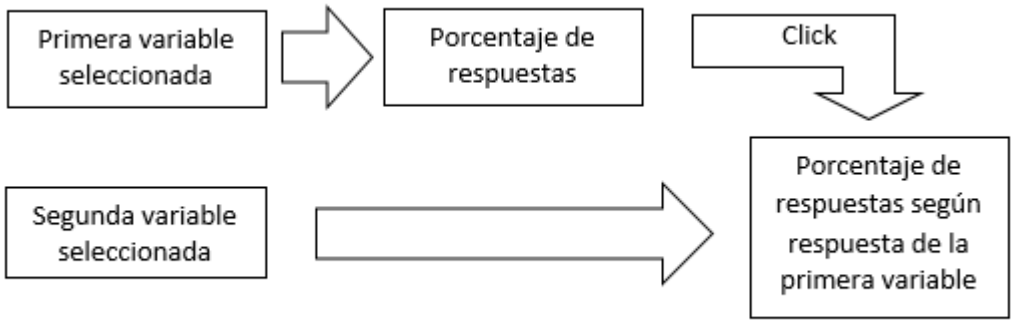

Figura 34. Modelo de análisis para las visualizaciones dinámicas

Como muestra la figura, para tratar los datos se necesitan las variables seleccionadas. Primero se obtienen los porcentajes de las respuestas de la primera variable, y con un click en cada respuesta, se obtienen los porcentajes de respuestas de la segunda variable en función de la respuesta clickeada de la primera variable.

5.2.5 Resultados de Visualizaciones

Figura 35. Resultado visualización dinámica con selector

Como se puede observar en la figura 35 en la parte derecha está el selector de relaciones. Tiene seleccionado la fuerza de trabajo general en la variable 1 y el nivel educacional en la variable 2. El resultado es el esperado, se muestra un Treemap con la fuerza de trabajo general y cuando se presiona un recuadro se carga un segundo Treemap en la parte inferior de la pantalla, enfocado en los datos de la caja seleccionada del Treemap superior. En el ejemplo de la figura, se ha seleccionado el grupo "Menor de 15 años", pidiendo el desglose de "nivel educacional" para el segundo Treemap. Se puede observar que en el Treemap inferior no aparecen individuos con nivel educacional superior a la enseñanza media.

5.3 Módulo de Visualizaciones Predefinidas

El propósito de este módulo es el de proporcionar un conjunto de visualizaciones para que sea consumida de forma inmediata por el usuario, dando la oportunidad de mostrar información previamente analizada, ya que muchas de estas visualizaciones muestran información que no puede ser obtenida solo de las variables, sino que existe un procesamiento de datos adicional.

5.3.1 Diseño de interacciones

A pesar que las visualizaciones predefinidas muestran cosas diferentes y tienen diferente procesamiento de los datos, estas siguen las mismas actividades para poder mostrarse en el prototipo. La siguiente figura muestra las actividades de las visualizaciones predefinidas.

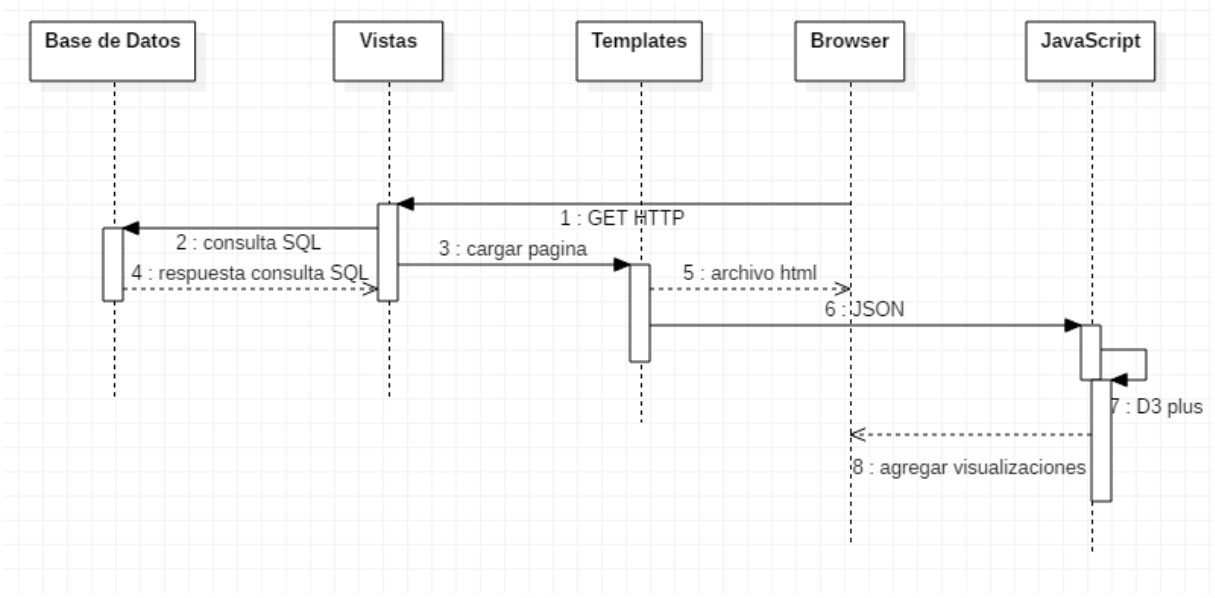

Figura 36. Diagrama de Actividad para las visualizaciones predefinidas

Como muestra la figura 36, las actividades comienzan en el paso 1 con una solicitud GET a la vista que corresponde a la visualización a mostrar, esta realiza una consulta a la base de datos, a través de los modelos de Django, para obtener los datos específicos para la visualización, luego se carga el template en el paso 3 y se envían también los resultados de la consulta en formato JSON en el paso 6, JavaScript convierte esos datos en objetos para luego procesarlos

según la visualización con D3 plus en el paso 7, y al final se muestran los resultados en el Browser en el paso 8.

5.3.2 Modelo de datos

Para obtener los datos de los encuestados se utiliza el mismo modelo de datos de la figura 24. Adicionalmente se creó una nueva tabla de datos, la que se muestra en la siguiente figura.

COMUNA		
CODIGO_COMUNA	INT	PK
NOMBRE_COMUNA	VARCHAR(50)	
LATITUD	DECIMAL(10,2)	
LONGITUD	DECIMAL(10,2)	

Figura 37. Tabla adicional del modelo de datos para la visualización predefinida

Como se observa en la figura se utilizará una tabla adicional que guarda las comunas de la octava región, el código corresponde al código región provincia comuna de las encuestas, y además cuenta con una latitud y una longitud, ya que esta tabla se usará en las visualizaciones con Heatmap.js y Leaflet, siendo este último una librería que proporciona un mapamundi donde se pueden ubicar los lugares geográficos utilizando los valores de latitud y longitud en decimal.

5.3.3 Información seleccionada

A partir de los resultados de las encuestas se puede obtener mucha información que podría ser útil para ser analizada y evaluada, sin embargo, para mostrar el concepto de *visual analytics* en el prototipo se ha escogido mostrar una serie de información relevante previamente consultada por profesores de Economía: Cesar Salazar del departamento de gestión empresarial y Andrés Acuña del departamento de economía y finanzas de la Universidad del Biobío. Esta información es:

- Brecha de ingresos que son las diferencias en los ingresos, entre los más ricos y más pobres.
- Tasa de Desigualdad, que mide la desigualdad que hay entre los más ricos y más pobres
- Conmutación Laboral, que es cuando se vive en una región y se trabaja en otra.

También se ha decidido mostrar los siguientes datos utilizando la visualización Heatmap, para poder comparar geográficamente su incidencia:

- Desempleo en todas las encuestas
- Pobreza multi-dimensional según CASEN
- Pobreza por ingresos según CASEN
- Extrema pobreza por ingresos según CASEN

5.3.4 Conmutación Laboral

El siguiente esquema muestra los pasos a seguir para mostrar la conmutación laboral.

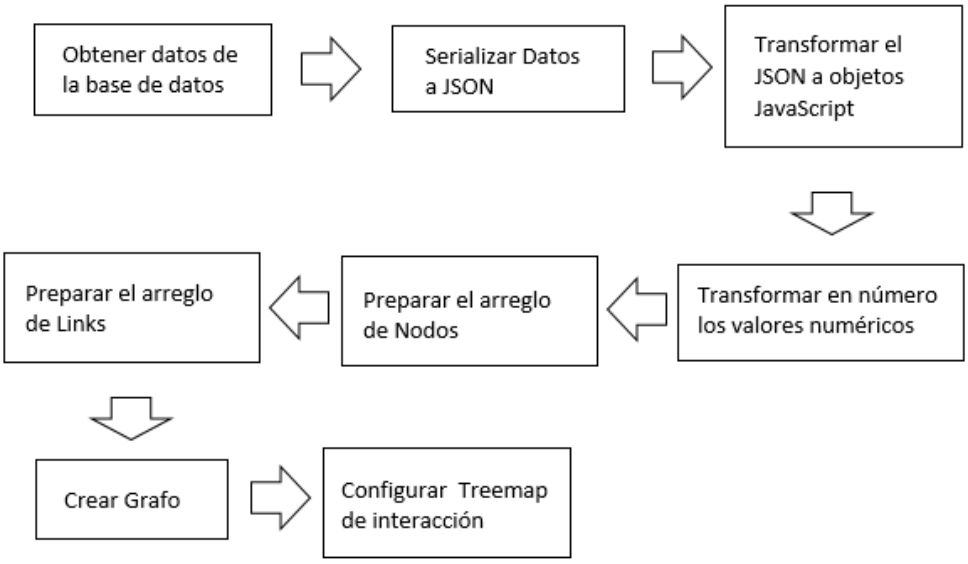

Figura 38. Esquema del procesamiento de datos para mostrar la conmutación laboral

Como muestra la figura 38, primero es necesario es realizar una consulta donde obtendremos los datos correspondientes a la comuna donde reside el encuestado y a la comuna donde desempeña sus funciones laborales. Esto se realiza mediante la función *objects.filter.only* de Django, luego se deben serializar los datos, para que pueda ser leído por JavaScript. Luego cuando se carga el template, se transforman los datos a objetos JavaScript para que los pueda usar la función del API, sin embargo, aunque se transforme el JSON a objeto, los valores son todos string, aunque sean numéricos, por ello es necesario transformar el tipo de variable a entero en los valores que corresponda. Luego de eso se prepara el arreglo de nodos y de links. El arreglo de nodos debe tener una posición x, una posición y un identificador obligatoriamente, mientras que los links deben tener un par de valores que hacen referencia a la posición de cada identificador en el arreglo de nodos, de manera que queda un valor como fuente y un valor como destino.

Luego se crea el grafo haciendo una llamada a la función de la API. La siguiente figura muestra el código para hacer la llamada al API para crear el grafo.

5.3.4.1 Modelo de Análisis

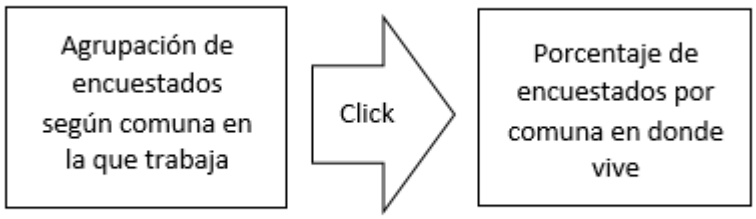

Figura 39. Modelo de Análisis de la conmutación laboral

Como muestra la figura 39, lo primero que se realiza para el tratamiento de datos es una agrupación de los encuestados según la comuna donde trabaja, esto se logra mediante el factor de expansión. Luego se calcula los porcentajes de encuestados en función a la comuna donde reside, utilizando la misma variable.

5.3.5 Brechas de Ingresos

Para mostrar las brechas de ingresos se utilizan los datos de la encuesta NESI, con y sin beneficios estatales. El siguiente esquema muestra los pasos a seguir para mostrar las brechas de ingresos

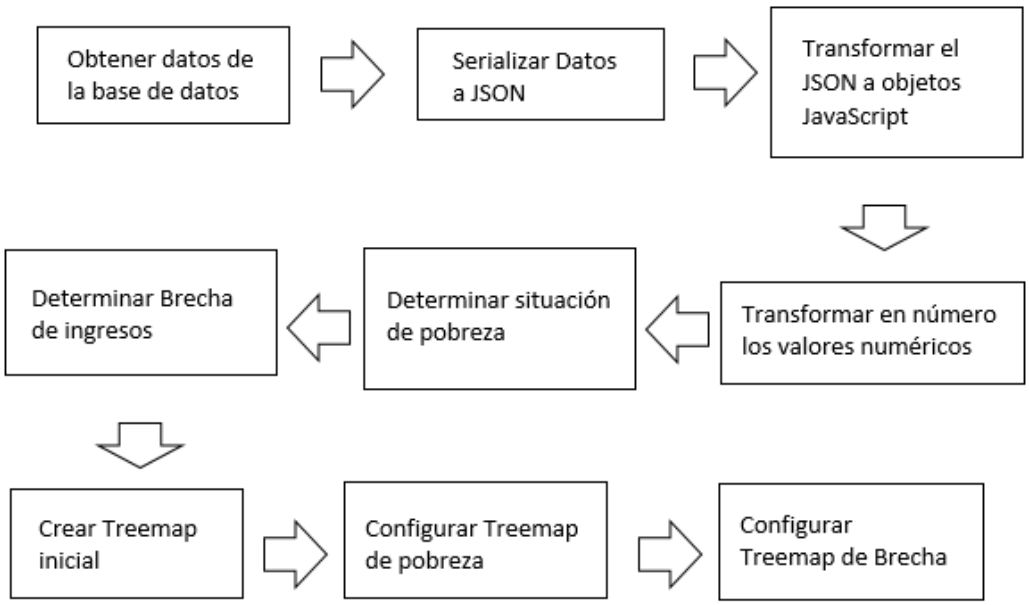

Figura 40. Esquema de procesamiento de datos para mostrar las brechas de ingresos

Para determinar las brechas de ingresos es necesario conocer las líneas de pobreza por ingresos y las líneas de extrema pobreza por ingresos[8]. Estas son extraídas a partir de la información de la encuesta CASEN y la encuesta de Presupuestos Familiares que da información sobre la canasta básica de una familia promedio. Un hogar es considerado pobre cuando no alcanza a suplir con la canasta básica para el hogar, por lo que mientras más integrantes tienen más cara es la canasta básica. Estos valores son ingresados en una función de JavaScript para que puedan usarse para clasificar a los encuestados según sus ingresos.

El proceso de obtención de datos es igual que en la conmutación laboral, primero se realiza las consultas, luego se serializan las respuestas, posteriormente se transforman a objetos JavaScript y se cambia el tipo de dato a los valores que son numéricos. Después viene la determinación de la pobreza, que consiste en separar a los encuestados en dos arreglos: uno

[8] Información sobre las líneas de pobreza y extrema pobreza usando la encuesta CASEN 2015 disponible en: http://observatorio.ministeriodesarrollosocial.gob.cl/layout/doc/ipc/Valor%20CBA%20y%20Lineas%20de%20Pobreza%20Agosto%202015.pdf

que tiene a los clasificados como pobres, y el otro a los clasificados como extremadamente pobres. Después se toman estos arreglos y se determina la brecha de ingresos que tienen para dejar de ser pobres según los habitantes del hogar. Cabe destacar que este proceso se realiza tanto para los encuestados con beneficios como para los encuestados sin beneficios estatales, ya que como se explica en la sección 5.1.2 las tablas están separadas, y en la visualización hay que juntarlas para que muestre la información de manera completa.

5.3.5.1 Modelo de Análisis

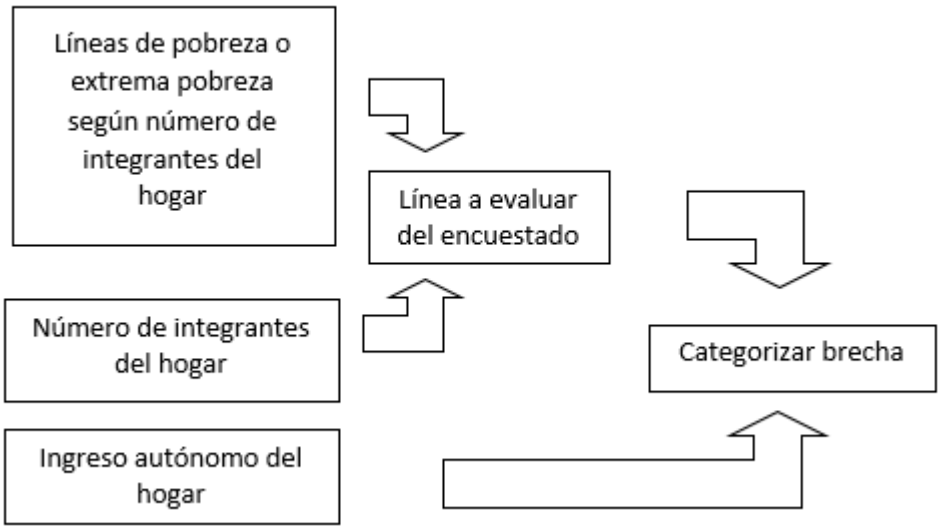

Figura 41. Modelo de Análisis para las brechas de ingresos

Para el tratamiento de datos primero es necesario obtener la línea de pobreza o extrema pobreza para el encuestado dependiendo de los habitantes del hogar del encuestado. Luego se realiza una resta entre el ingreso autónomo del hogar con la línea obtenida. Finalmente, dependiendo de los resultados se categoriza la brecha por rango. Esto se realiza para cada encuestado y al final se obtienen los porcentajes de cada categoría.

5.3.6 Desigualdad

La siguiente figura muestra los pasos a seguir para mostrar la desigualdad.

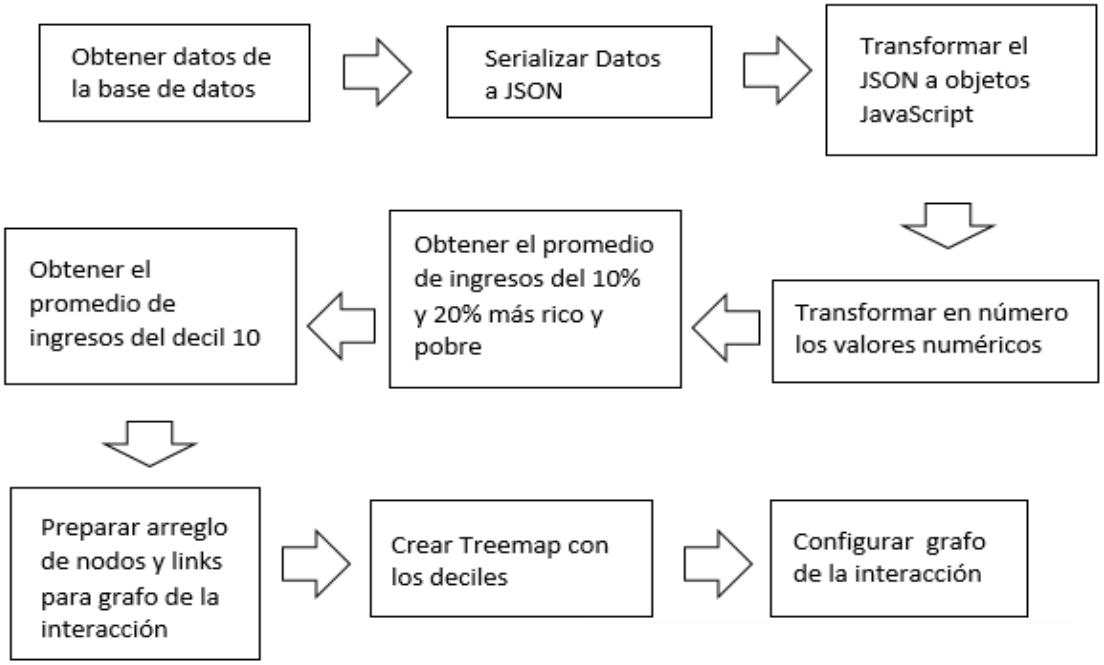

Figura 42. Esquema del procesamiento de datos para mostrar la desigualdad.

Para medir la desigualdad en el prototipo se utilizó el índice 10/10 y el índice 20/20 y se mostrarán en un grafo en donde se diferencian cuantas veces mayor es el ingreso autónomo del 10% más rico de la región que el 10% más pobre de la región y otro para el 20% más rico y el 20% más pobre. Además se mostrarán los deciles regionales en un Treemap y se mostrará cuantas veces es un decil respecto del decil más rico, es decir, el decil 10. Para ello primero se obtienen los datos a través de una consulta, se serializan las respuestas, luego en JavaScript se transforman en objetos y se cambia el tipo de dato numérico a los que corresponden. Después se debe obtener el promedio de ingresos de los más ricos y más pobres de la región, para después calcular los índices de desigualdad. Después hay que obtener el promedio de ingresos de Decil 10, ya que con los deciles se va a realizar las interacciones, y el decil 10 es el más rico. Luego hay que preparar los nodos del grafo, estos serán los correspondientes al índice 10/10 y 20/20, y también la diferencia entre los deciles. Luego de preparar los nodos hay que

hacer lo mismo con el arreglo de links, y finalmente se debe crear el Treemap con los deciles de la región, y se debe configurar la interacción con el grafo, ya que cada vez que se presione un decil se mostrarán los índices y la relación del decil seleccionado con el decil 10.

5.3.6.1 Modelo de Análisis

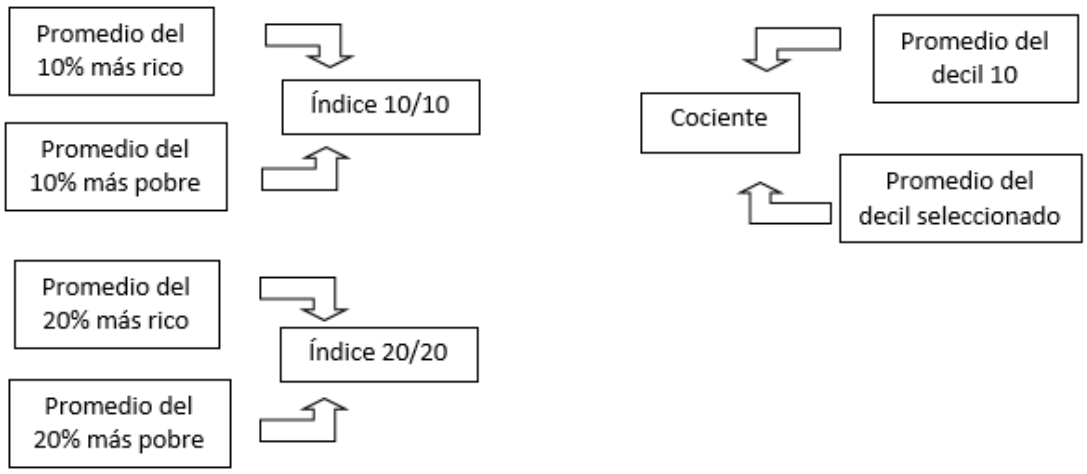

Figura 43. Modelo de Análisis para la desigualdad

El tratamiento de datos para mostrar la desigualdad consiste en obtener los índices 10/10 y 20/20. Además de obtener el cociente entre el promedio de ingresos del decil 10 con el promedio de ingresos del decil seleccionado. Este último proceso se realiza de manera dinámica dependiendo del decil seleccionado y los índices solo se realizan una vez, al cargar la visualización.

5.3.7 Visualizaciones usando Heatmap.js

Para visualizar todos los datos que usan heatmap.js se usa el procedimiento que muestra la figura 40.

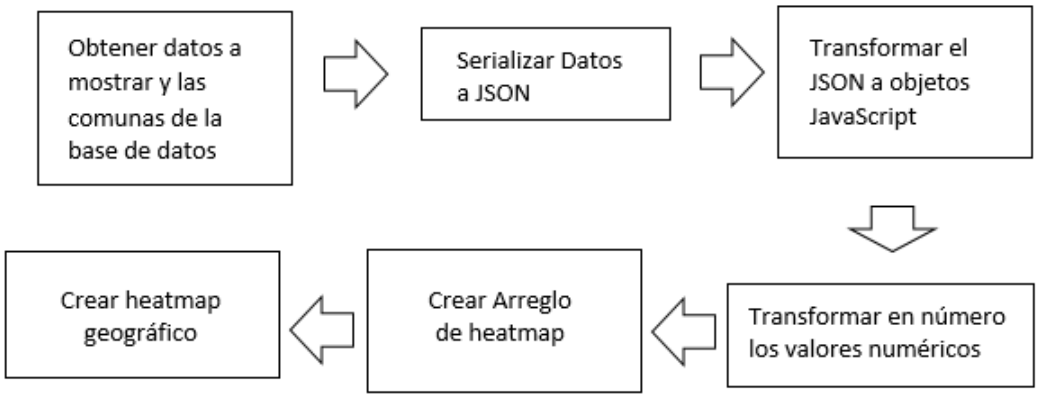

Figura 44. Esquema del procesamiento de datos para mostrar los datos con Heatmap.js

Para estas visualizaciones se usarán datos adicionales, como lo son las coordenadas geográficas de las comunas de la región. Primero hay que realizar una consulta para obtener las respuestas de los encuestados y otra para las comunas, luego se deben serializar a JSON y luego se envían a JavaScript para transformarlos en objetos y cambiar el tipo a datos numéricos. Luego se crea el arreglo para mostrarlo en el heatmap. El arreglo debe contener objetos que representan a cada comuna y debe tener obligatoriamente los siguientes atributos: latitud, longitud y un valor contable, para las visualizaciones se usará el factor de expansión. Sin embargo este factor de expansión debe procesarse para obtener los datos que se necesitan, dependiendo de la visualización, por ejemplo, para la visualización de desempleo, se saca una tasa dividiendo la suma de factores de expansión de los encuestados cesantes con la suma de factores de expansión de encuestados que son ocupados y cesantes. Finalmente se llama a la función que crea el heatmap en la API.

5.3.7.1 Modelo de Análisis

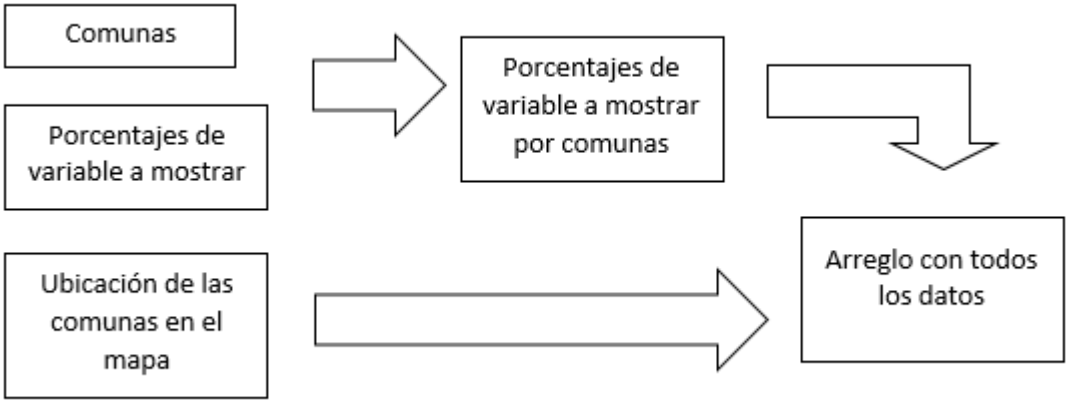

Figura 45. Modelo de análisis para las visualizaciones usando heatmap.js

Como muestra la figura 45, cada vez que se carga un Heatmap, se necesitan tres arreglos: las comunas, la ubicación de las comunas en el mapa, y los porcentajes de variable a mostrar. Este porcentaje se obtiene sumando los factores de expansión por encuestado. El objetivo final del tratamiento de datos es obtener un solo arreglo con los porcentajes de variable por comuna y con la ubicación de cada comuna.

5.3.8 Resultados de visualizaciones

Se mostrará un ejemplo de visualizaciones usando D3 plus y un ejemplo utilizando Heatmap.js. Para el ejemplo de D3 plus se muestra el resultado de la visualización de la conmutación Laboral.

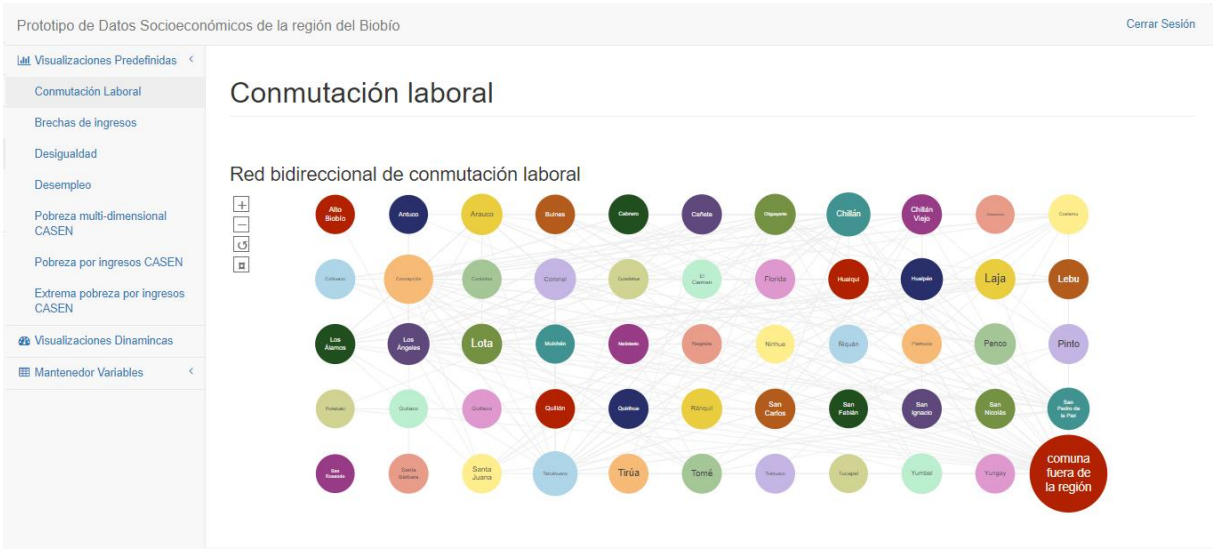

Figura 46. Resultado de la visualización de la conmutación Laboral

La figura 46 muestra el grafo de la conmutación laboral. Los nodos adquieren color de manera aleatoria, además tienen agregado el distintivo del tamaño, como se puede observar, existe un nodo particularmente grande en la esquina inferior derecha, que corresponde a un nodo que representa a cualquier otra región de Chile. Demostrando que de la conmutación existente la mayoría es conmutación regional, más que conmutación comunal.

Si se selecciona algún nodo automáticamente se abrirá un Treemap que muestra información más precisa sobre la conmutación laboral en cada comuna.

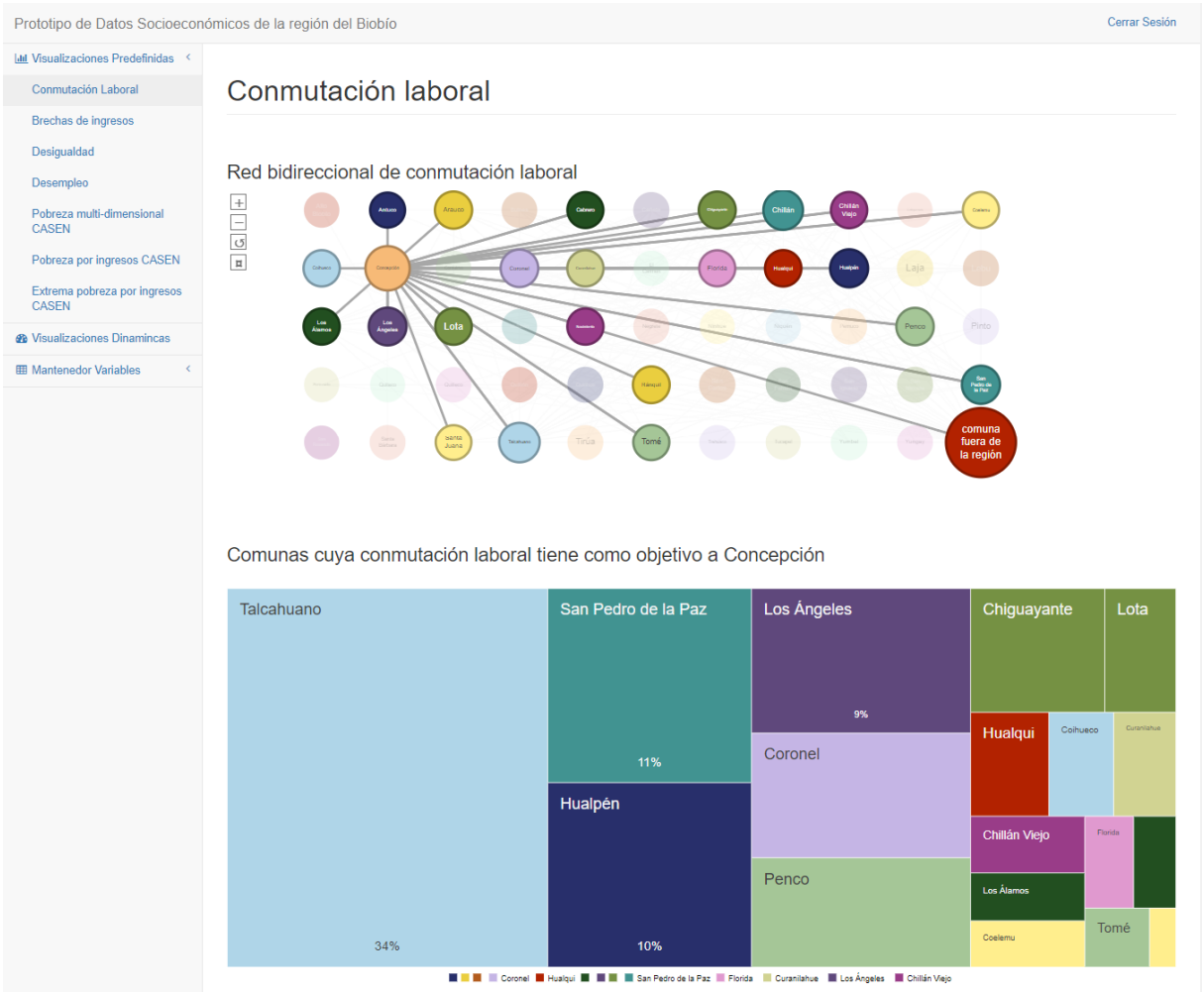

Figura 47. Resultado de la interacción en el grafo

Como muestra la figura 47, se puede observar a través del grafo las conexiones entre comunas y en el Treemap se muestra el porcentaje de conmutación hacía la comuna seleccionada. En el ejemplo, el grafo muestra la conmutación laboral hacia Concepción y el Treemap muestra el porcentaje de conmutación laboral de las comunas involucradas hacia Concepción. Se observa que la conmutación laboral más grande viene desde Talcahuano con un 34%, seguido por San Pedro de La Paz y Hualpén con un 11% y 10% respectivamente.

Para una visualización usando Heatmap.js se usa un ejemplo que muestra la pobreza multidimensional en la región.

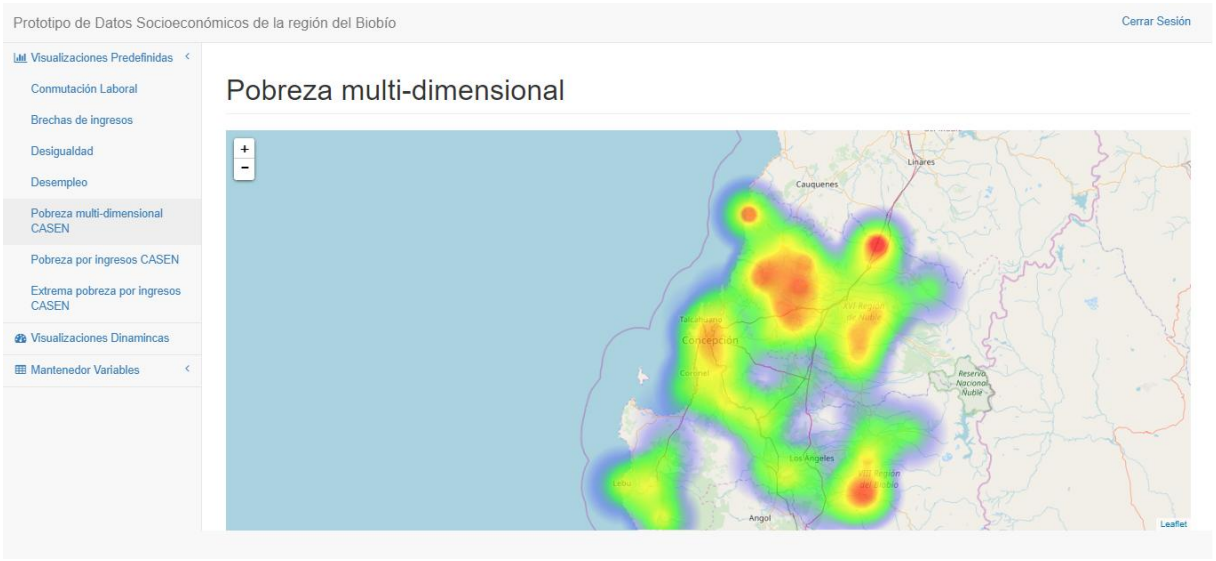

Figura 48. Resultado visualización pobreza multi dimensional usando hetmap.js con Leaflet

Como muestra la figura 48, se aprecia dónde hay más pobreza multi dimensional en la región, usando los colores cálidos donde existe mayor cantidad de personas calificadas como pobres y los colores fríos donde hay menor cantidad de personas pobres. Todo esto según la encuesta CASEN.

Capítulo 6. Conclusiones

6.1 Cumplimiento de objetivos

Para el desarrollo de este proyecto se revisó el estado del arte en cuanto a la visualización de datos, con énfasis en datos económicos y del análisis visual de datos. Además se estudió la estructura de las encuestas seleccionadas y los tipos de visualización de datos. Se seleccionaron los tipos de visualización a usar y también se seleccionaron e implementaron técnicas para poder demostrar el uso de visual analytics. Finalmente se desarrolló un prototipo de plataforma web para datos estadísticos socioeconómicos en la región del Biobío, pasando por las 4 grandes actividades en el proceso de desarrollo de software: análisis, diseño, construcción y pruebas.

6.2 Resultados del proyecto

Al finalizar el proyecto se demuestra que el uso de visual analytics es una alternativa factible para mejorar la manera en la que se presentan los datos, al permitir la interacción del usuario. Se demuestra que existen tecnologías para su implementación y que existe una amplia gama de tipos de visualización para poder combinar con visual analytics. Además existen tipos de visualización de datos a los que se les puede sacar más provecho, como es el caso del Treemap.

Se observa que a pesar de que los datos son simples, el pre procesamiento de los datos de las bases de datos de las encuestas, y su posterior almacenamiento en un formato eficiente para su uso por parte de las herramientas utilizadas supuso un proceso laborioso. El tratamiento de los datos y la integración de diferentes herramientas es una parte sustancial de este proyecto, ya que se requirió la incorporación de diferentes tecnologías, como d3plus, que es una tecnología relativamente reciente cuya última versión no tiene un gran soporte en la red, junto a bibliotecas de conversión de formatos como savReaderWriter, y frameworks para plataformas Web como Django, entre otros, además de lenguajes de programación como Python, JavaScript y SQL.

El conocimiento obtenido de este proyecto además puede ser usado como punto de partida para la implementación de una plataforma web definitiva, en donde se puedan agregar toda la información que puede ser extraída de las encuestas CASEN, NENE y NESI, e incluso puede servir para el desarrollo de plataformas web que utilicen información obtenida de encuestas de cualquier ámbito, o poder combinar información de encuestas de diferente índole.

Referencias

Axalpha Consulting. (2018). SAP Lumira. Retrieved from https://www.axalphaconsulting.com/es/content/51-sap-lumira

Bederson, B. B., Shneiderman, B., & Wattenberg, M. (2002). Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Transactions on Graphics*, *21*(4), 833–854. https://doi.org/10.1145/571647.571649

Bondy, J. A., & Murty, U. S. R. (1982). *Graph Theory with Applications*. *Operational Research Quarterly (1970-1977)* (Vol. 28). https://doi.org/10.2307/3008805

Bray, T. (2017). The JavaScript Object Notation (JSON) Data Interchange Format. Retrieved from https://www.rfc-editor.org/info/rfc8259.

Bruls, M., Huizing, K., & van Wijk, J. J. (2000). Squarified Treemaps, 33–42. https://doi.org/10.1007/978-3-7091-6783-0_4

DataWheel. (2018). D3plus. Retrieved from https://www.datawheel.us/portfolio/d3plus/

Departamento Administrativo Nacional de Estadística (DANE). (2015). Pobreza Y Desigualdad. *Pobreza Monetaria Y Multidimensional En Colombia 2014*. Retrieved from http://www.dane.gov.co/files/investigaciones/condiciones_vida/pobreza/bol_pobreza_14_.pdf

Flanagan, D. (2006). JavaScript:The Definitive Guide (5th ed., p. 1032). O'Reilly Media, Inc.

Fundación superación de la pobreza. (2005). Brechas socioeconómicas de la población chilena.

Garmendia, I. (2016). ¿Qué es SAP Lumira? Retrieved from https://orekait.com/blog/sap-lumira-que-es/

INE, I. N. de E. de C. (2010a). Nueva Encuesta Nacional de Empleo, Manual Conceptual -

Abril 2010, 1–35.

INE, I. N. de E. de C. (2010b). Qué es la Nueva Encuesta Nacional de Empleo, 4.

INE, I. N. de E. de C. (2017). *Encuesta Suplementaria de Ingresos: Manual y guía de variables base hogares ESI 2016 con becas*.

INE, I. N. de E. de C. (2018a). *Diccionario de Variables bases de datos*.

INE, I. N. de E. de C. (2018b). Síntesis de resultados ENCUESTA SUPLEMENTARIA DE INGRESOS 2017.

Infante Moreno, S. (2012). Curso Django: Entendiendo cómo trabaja Django. Retrieved from http://www.maestrosdelweb.com/curso-django-entendiendo-como-trabaja-django/

Keim, D., & Thomas, J. (2008). Scope and Challenges of Visual Analytics. *IEEE Visualization Conference 2007*, *4404*(4404), 1–58. https://doi.org/10.1007/978-3-540-71080-6_6

Mena, D. (2015). MAPAS DE CALOR (HEATMAPS): QUÉ SON, PARA QUÉ SIRVEN Y POR QUÉ USARLOS. Retrieved from https://wanaleads.com/mapas-de-calor-heatmaps-para-optimizar-web/

Microsoft. (2018a). Power BI. Retrieved from https://powerbi.microsoft.com/es-es/what-is-power-bi/

Microsoft. (2018b). Power BI Desktop. Retrieved from https://powerbi.microsoft.com/es-es/desktop/

Ministerio de Desarrollo Social. (2016). Libro de códigos CASEN.

Misnisterio de Desarrollo Social. (2011). Encuesta CASEN. Retrieved from http://observatorio.ministeriodesarrollosocial.gob.cl/index.php

Molina, J., Loja, N., Zea, M., & Loaiza, E. (2016). Evaluación de los Frameworks en el Desarrollo de Aplicaciones Web con Python. *Revista Latinoamericana de Ingenieria de Software*, *4*(4), 201–207. https://doi.org/10.18294/relais.2016.201-207

MySQL AB. (2005). *MySQL Administrator's Guide*.

Pino A., O., & Concha M., G. (2012). LA VIII REGIÓN DEL BIO BÍO (NENE año 2012) INTERREGIONAL COMMUTATION , A VIEW FROM THE BIO BIO REGION VIII (NENE 2012) Osvaldo Pino A . Departamento de Economía y Finanzas Facultad de Ciencias Empresariales , Universidad del Bío-Bío Gary Concha M . De, 44–58.

Rayón, A. (2015). VISUAL ANALYTICS: LA VISUALIZACIÓN ANALÍTICA, EFICIENTE E INTELIGENTE DE DATOS. Retrieved from https://blogs.deusto.es/bigdata/visual-analytics-la-visualizacion-analitica-eficiente-e-inteligente-de-datos/

Ribecca, S. (2014). The Data Visualization Catalogue. Retrieved May 2, 2018, from https://datavizcatalogue.com/index.html

Ropinski, T., & Preim, B. (2008). Taxonomy and Usage Guidelines for Glyph-based Medical Visualization. *SimVis*, (February), 121–138. Retrieved from http://www.researchgate.net/publication/221510582_Taxonomy_and_Usage_Guidelines_for_Glyph-based_Medical_Visualization/file/9fcfd50c74c6f18101.pdf

Roskam, A.-J. (2013). savReaderWriter documentation. Retrieved from https://pythonhosted.org/savReaderWriter/index.html

Sacha, D., Stoffel, A., Stoffel, F., Kwon, B., Ellis, G., & Keim, D. (2014). Knowledge Generation Model for Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics*, *20*(12), 1604–1613. https://doi.org/10.1109/TVCG.2014.2346481

SAS Institute Inc. (2018a). SAS. Retrieved from https://www.sas.com/en_us/software/stat.html#modal1-fcd9ad7e12dc46abadaa89fd14224231

SAS Institute Inc. (2018b). SAS Visual Analytics. Retrieved from https://www.sas.com/es_cl/software/visual-analytics.html

Shneiderman, B., & Wattenberg, M. (2001). Ordered treemap layouts. *IEEE Symposium on*

Information Visualization, 2001. INFOVIS 2001., *2001*, 2–7. https://doi.org/10.1109/INFVIS.2001.963283

Tableau Software. (2017a). Tableau Desktop. Retrieved from https://www.tableau.com/es-es/products/desktop

Tableau Software. (2017b). Tableau Online. Retrieved from https://www.tableau.com/es-es/products/cloud-bi

Tableau Software. (2017c). Tableau Prep. Retrieved from https://www.tableau.com/es-es/products/prep

Tableau Software. (2017d). Tableau Server. Retrieved from https://www.tableau.com/es-es/products/server

Thomas, J. J., & Cook, K. A. (2005). Illuminating the path: The research and development agenda for visual analytics. *IEEE Computer Society*, 184. https://doi.org/10.3389/fmicb.2011.00006

Ward, M. O. (2008). Multivariate Data Glyphs : Principles and Practice. *In Handbook of Data Visualization, Springer*, (January 2008), 179 – 198. https://doi.org/10.1007/978-3-540-33037-0

Wied, P. (2016). Heatmap.js. Retrieved from https://www.patrick-wied.at/static/heatmapjs/

Wikipedia. (2003). Grafo. Retrieved from https://es.wikipedia.org/wiki/Grafo

Wikipedia. (2012). Django(framework). Retrieved from https://es.wikipedia.org/wiki/Django_(framework)

Wikipedia. (2013). Encuesta CASEN. Retrieved from https://es.wikipedia.org/wiki/Encuesta_Casen

Anexo 1. Tipos de Visualización de Datos

A1.1 Glifos

En el contexto de la visualización de datos, un glifo es una representación visual de una porción de datos donde los atributos de una entidad gráfica están dictados por uno o más atributos de un registro de datos (Ward, 2008). Los elementos que proporcionan información en un glifo son (Ropinski & Preim, 2008):

- **La forma:** en relaciones espaciales, la forma de un glifo cobra una gran importancia, ya que puede percibirse fácilmente y sin ambigüedades. Según su forma se pueden distinguir dos grupos de glifos: los glifos básicos, que son objetos geométricos que se les pueden cambiar sus propiedades geométricas (alto, largo, ancho, radio, etc.), y los glifos compuestos, que están formados por glifos básicos, éstos requieren una forma de mapeo más especializada y son comúnmente usados para mostrar datos multivariados.

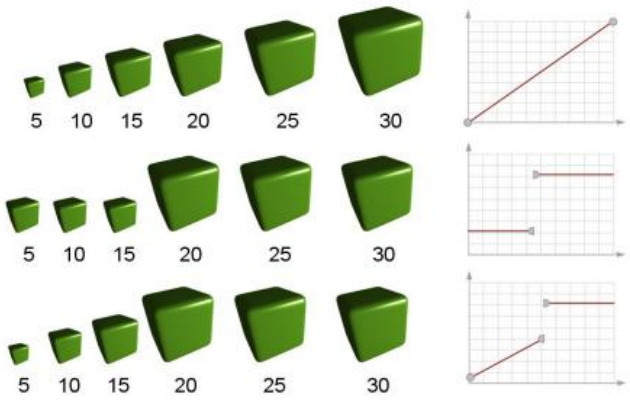

Figura 49. Representación en glifo (izquierda) de las funciones (derecha).
Tomado de (Ropinski & Preim, 2008)

- **El color:** para percibir cambios en la superficie del glifo, el uso del color es la mejor opción, por la facilidad de percepción, sin embargo el uso de cambios de color es para

variaciones más generales, ya que cuando hay muchas cantidades diferentes en las mediciones, la variación del color es más difícil de percibir. Sin embargo se pueden obtener otros resultados si se incorporan otros elementos como la transparencia del color para mostrar información direccional.

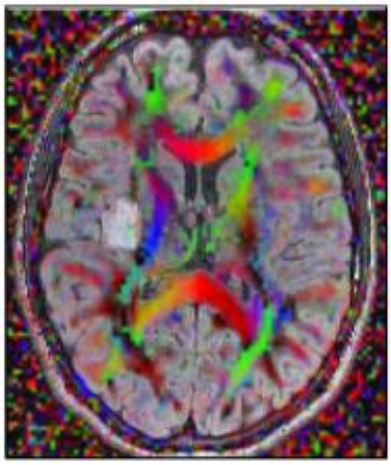

Figura 50. Uso del color y transparencia del color, Tomado de (Ropinski & Preim, 2008)

- **La colocación:** Dependiendo de dónde se ubique un glifo en relaciones espaciales puede entregar información relacionada al sector en donde está ubicado. En la figura 50 podemos ver que el uso de la colocación de glifo y el cambio de forma es crucial para determinar los resultados del análisis.

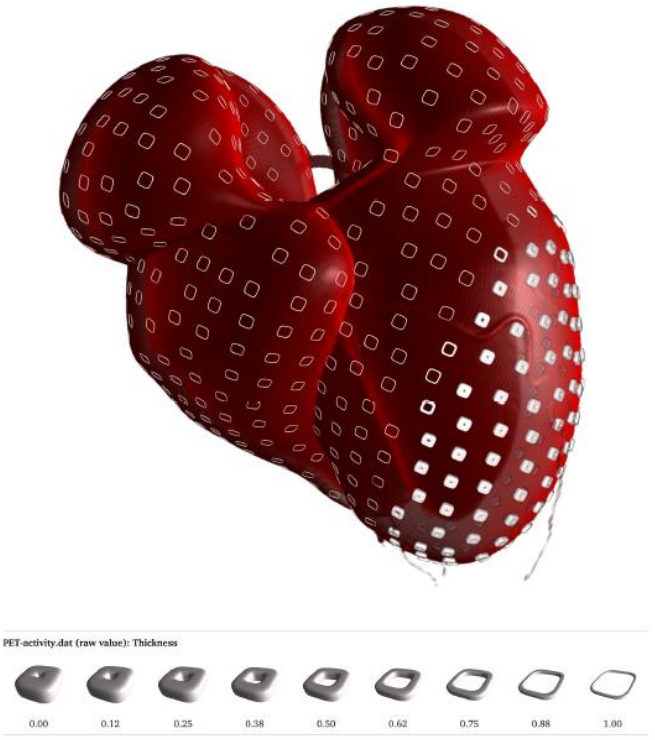

Figura 51. Representación con glifos de una tomografía por emisión de positrones (PET) en conjunto con una tomografía computarizada que muestra un miocardio, Tomado de (Ropinski & Preim, 2008)

A1.2 Gráficos

Los gráficos son los medios de representación de datos más comunes y utilizan recursos visuales como las líneas, los vectores, las superficies, o símbolos para demostrar relaciones numéricas o estadísticas, algunos utilizan coordenadas cartesianas para determinar la relación en función de los ejes. Existen muchos tipos de gráficos, los más comunes son los siguientes (Ribecca, 2014):

Gráfico de Barras

Es un gráfico que representa los datos en forma de barras separados en categorías, todas las barras comparten un elemento de su forma igual (puede ser ancho o largo), y el otro varía dependiendo de los resultados de las mediciones que se quieren mostrar.

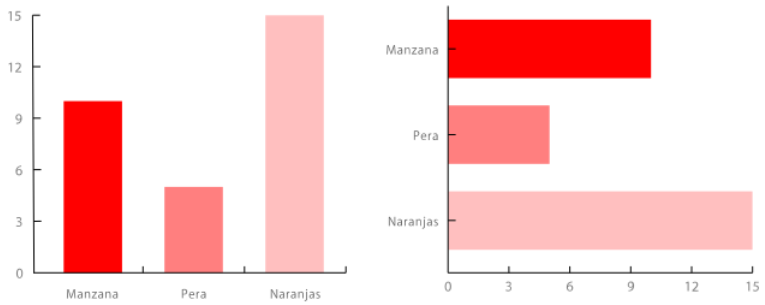

Figura 52: ejemplo de gráfico de barras, tomado de (Ribecca, 2014)

Histogramas

Es un gráfico que representa una variable, que generalmente es un intervalo, en forma de barras unidas, donde su superficie es proporcional la frecuencia de los valores representados. Es usada para relacionar variables cuantitativas continuas.

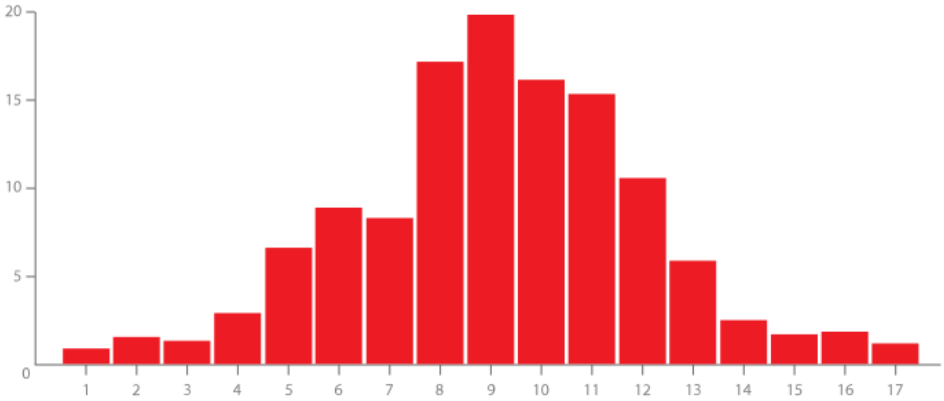

Figura 53: Ejemplo de histograma, tomado de (Ribecca, 2014)

Gráfico de dispersión

Es un gráfico que representa la relación de una variable independiente respecto de una variable dependiente. La variable independiente se controla para determinar el comportamiento de la variable dependiente obteniendo un indicador. Después de obtener los

indicadores suficientes se puede determinar la correlación de las dos variables. Mientras la unión de los indicadores (que comúnmente son puntos) se parezca más a una línea mayor es la correlación que existe.

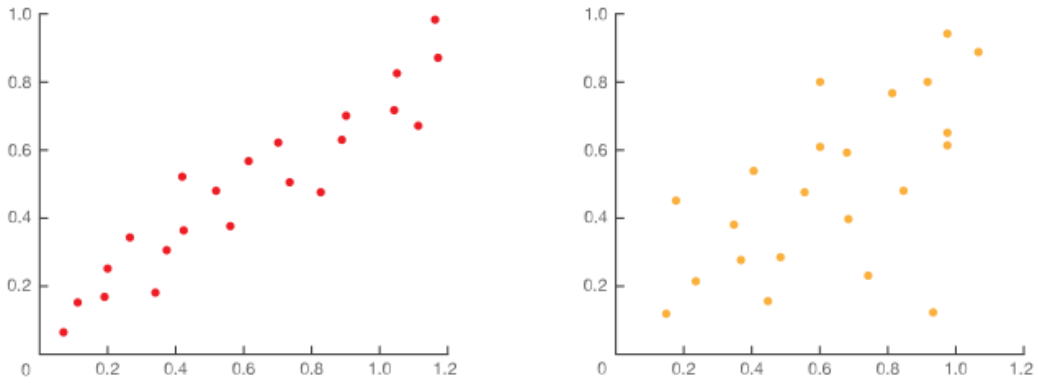

Figura 54: Ejemplo de gráfico de dispersión, tomado de (Ribecca, 2014)

Grafico circular

Es un gráfico que muestra la división por sectores de un todo, que es un elemento concreto representado en un círculo, y cada sector es la representación de una característica especial en ese elemento que la diferencia del resto. Es comúnmente usado para mostrar porcentajes y proporciones.

Figura 55: Ejemplo de gráfico circular, tomado de (Ribecca, 2014)

Gráfico de Líneas

Es un gráfico que utiliza una línea para mostrar el flujo de los resultados en función de dos ejes cartesianos que representan una variable dividida en una unidad de medida definida. Es comúnmente usado para mostrar patrones y tendencias en función del tiempo.

Figura 56: Ejemplo de gráfico de Líneas, tomado de (Ribecca, 2014)

Gráfico de Área

Este es un gráfico de líneas pero con el área debajo de la línea llenada con un color o textura, y cumple la misma función que los gráficos de línea pero con la diferencia de que el gráfico de línea puede contener más de una variable a medir, es decir pueden haber muchas líneas en un gráfico de línea pero solo una en un gráfico de área.

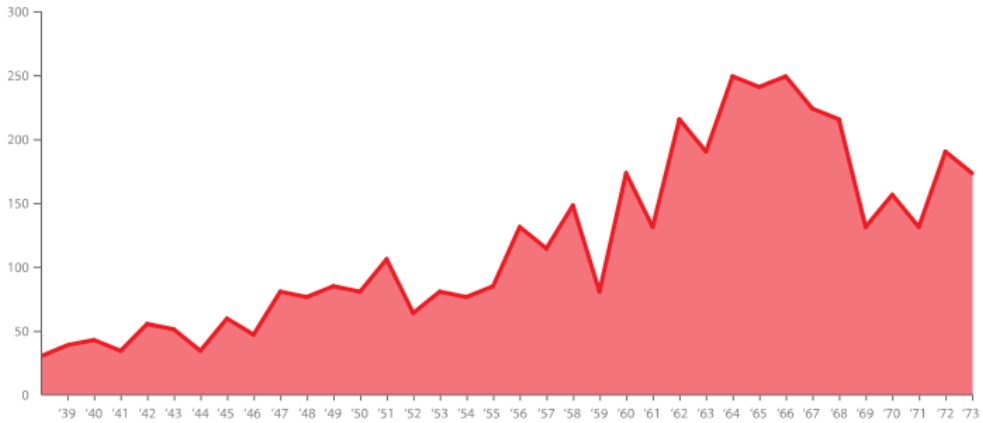

Figura 57: Ejemplo de gráfico de área, tomado de (Ribecca, 2014)

Gráficos para datos multivariables

Se utilizan para representar visualmente el comportamiento de tres o más variables al mismo tiempo, son comúnmente usados en la estadística multivariante. La desventaja más común de estos gráficos es que tienen un límite en las variables a mostrar antes de que tengan problemas de legibilidad, los más utilizados son:

Gráfico Radial

También conocido como gráfico Polar o gráfico de Estrella, este gráfico muestra múltiples variables cuantitativas, en función de una escala determinada. A cada variable se le asigna un eje que empieza en el centro de manera que todos se disponen radialmente y con igual distancia entre ellos. Cada valor se traza en el eje individual de la variable y todos los valores se unen con líneas formando un polígono.

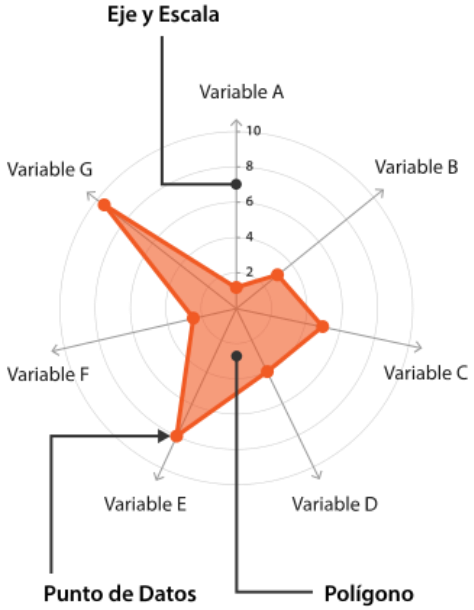

Figura 58: Ejemplo de gráfico radial, tomado de (Ribecca, 2014)

Trazado de coordenadas paralelas

Este gráfico muestra muchas variables y las relaciones entre ellas a través de un trazo de valores numéricos. En este gráfico cada variable recibe su eje y su escala, se colocan de forma paralela entre sí a igual distancia unas de las otras y cada variable se conecta a través de una línea.

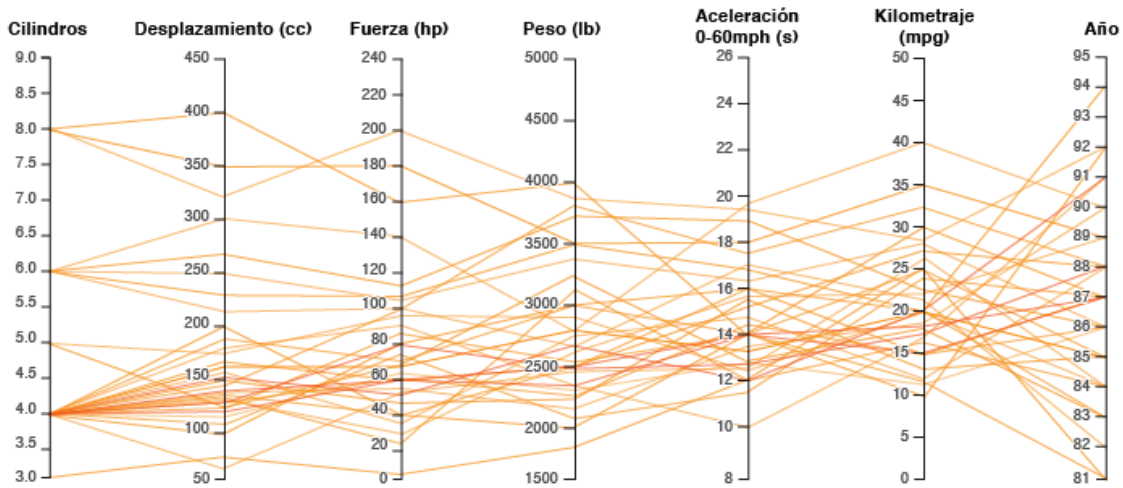

Figura 59: Ejemplo de trazado de coordenadas paralelas, tomado de (Ribecca, 2014)

Gráfico de área apilada

Estos funcionan de la misma manera que los gráficos de Área, con la diferencia de que usan series de datos múltiples que comienzan alejados del punto dejado por la serie anterior.

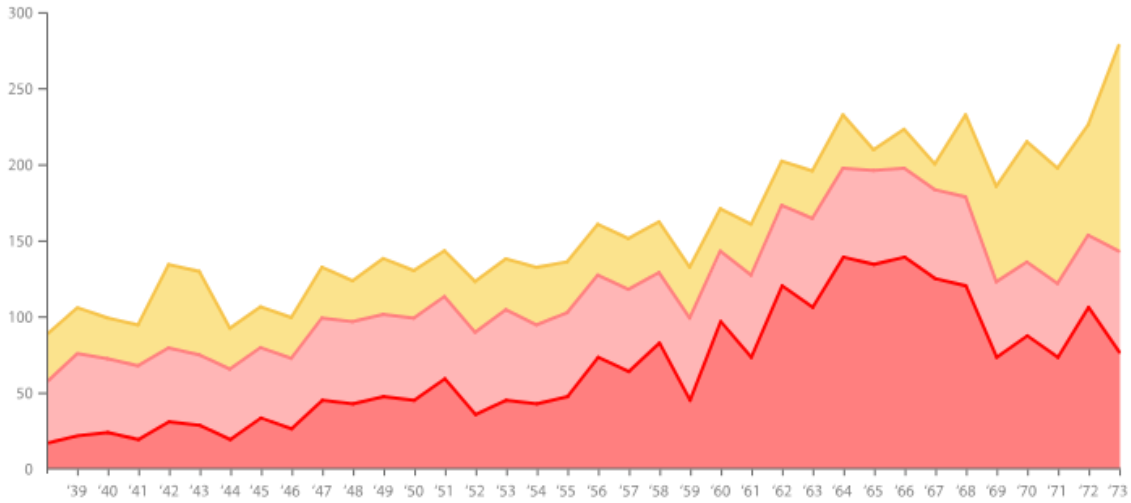

Figura 60: Ejemplo de gráfico de área apilada, tomado de (Ribecca, 2014)

Gráfico de Flujo

Este gráfico es una variación del gráfico de área apilada, pero con la diferencia de que cada valor utiliza como base una línea variable. El tamaño de cada flujo individual es proporcional al valor de la categoría y el eje que se traza paralelamente al flujo representa al tiempo. Es ideal para representar tendencias y patrones de grandes volúmenes de datos.

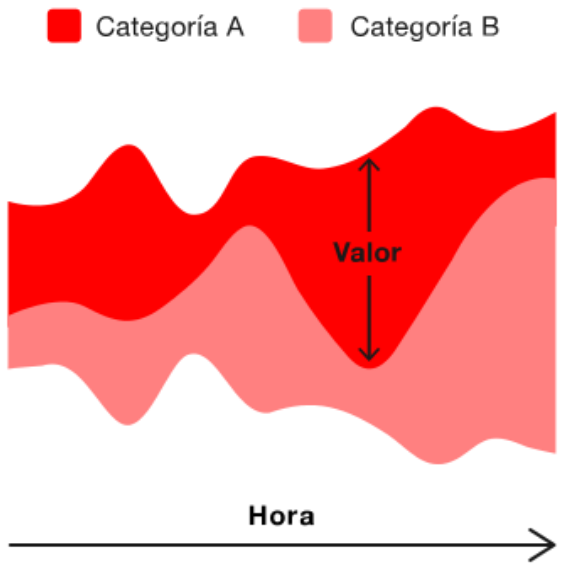

Figura 61: Ejemplo de gráfico de flujo, tomado de (Ribecca, 2014)

Gráfico de Burbujas

Estos gráficos, al igual que los gráficos de dispersión, utilizan coordenadas cartesianas para mostrar correlaciones entre variables, la diferencia es que en este gráfico además de las coordenadas cartesianas se utiliza el área de los puntos como una variable más, que debe ser coloreada ya sea para mostrar alguna categoría o para mostrar una variable adicional.

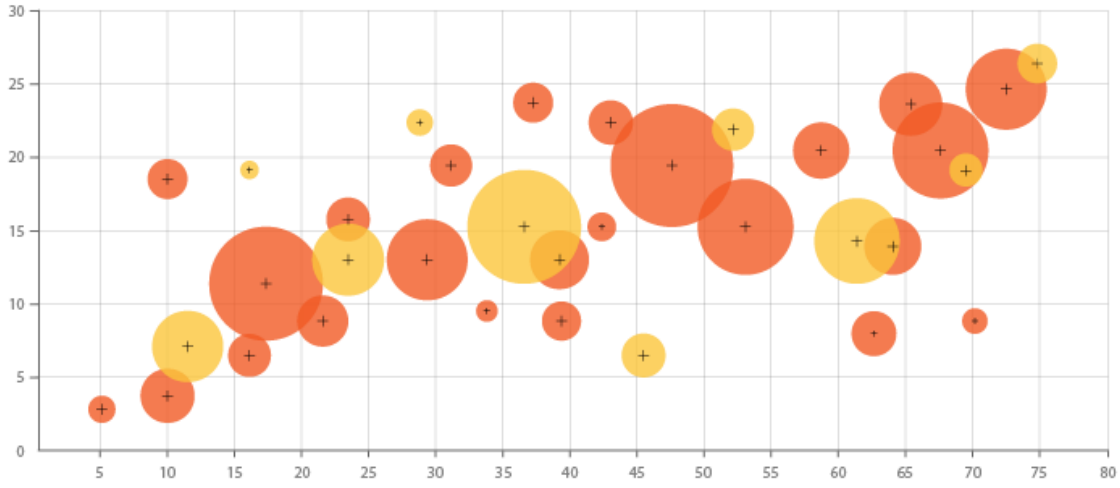

Figura 62: Ejemplo de gráfico de burbujas, tomado de (Ribecca, 2014)

A1.3 Diagrama de Árbol Circular

El diagrama de árbol circular es una variante del Treemap que muestra la jerarquía de datos usando círculos en lugar de rectángulos. La rama del árbol es un círculo y las subramas u hojas son representados como círculos dentro de él y se pueden utilizar los tamaños para mostrar variables del conjunto de datos. Este diagrama tiene la ventaja de mostrar la jerarquía mejor que un treemap, sin embargo utiliza mucho más espacio que un treemap para mostrar datos.

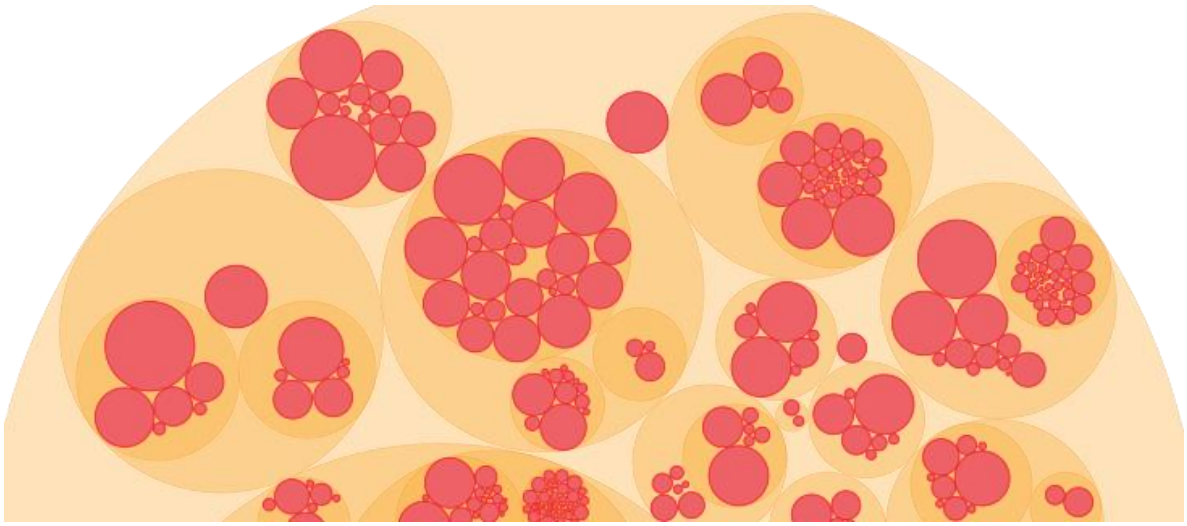

Figura 63: Ejemplo de diagrama de árbol circular, tomado de (Ribecca, 2014)

A1.4 Mapa Coroplético

Se utiliza para mostrar valores sobre un área geográfica específica dividiéndola por sectores o regiones. Se utiliza una progresión de colores para representar la variable en cada región. Sin embargo el uso del color no permite una comparación exacta de los datos, ya que las regiones más grandes quedan más enfatizadas sin importar el color que posean, afectando la percepción correcta de los datos.

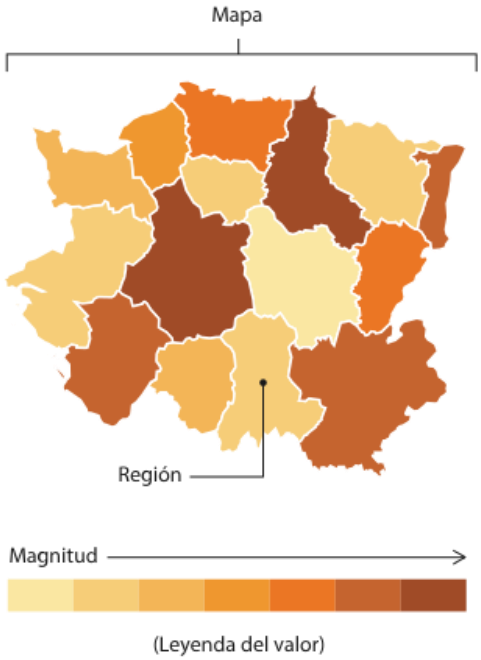

Figura 64: Ejemplo de mapa coroplético, tomado de (Ribecca, 2014)

A1.5 Mapa de Burbujas

Este mapa utiliza burbujas para representar los datos variando el área de estos sobre una ubicación geográfica, esto permite mostrar datos independientemente del tamaño de la región geográfica. Sin embargo, es necesario tener cuidado de que las burbujas no se superpongan unas de otras, ya que dificulta la representación de los datos.

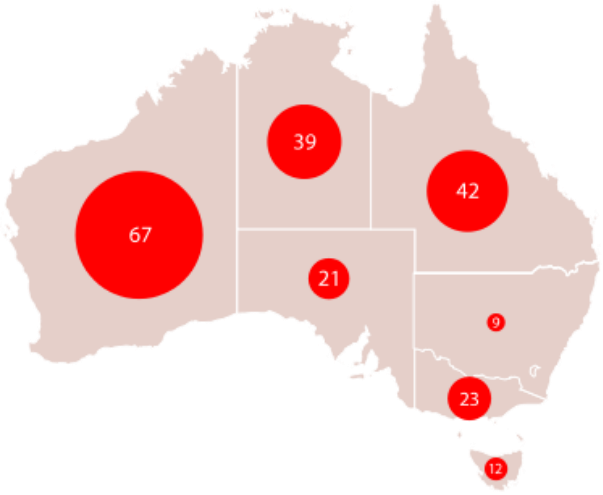

Figura 65: Ejemplo de mapa de burbujas, tomado de (Ribecca, 2014)

A1.6 Mapa de Puntos

Los mapas de puntos proporcionan una manera de representar densidad o distribución de datos en una región geográfica utilizando puntos de igual tamaño, cada punto representa un objeto o alguna unidad de medida. Se utilizan para la visión general de datos por su fácil comprensión visual, sin embargo no sirven para la entrega de datos exacta.

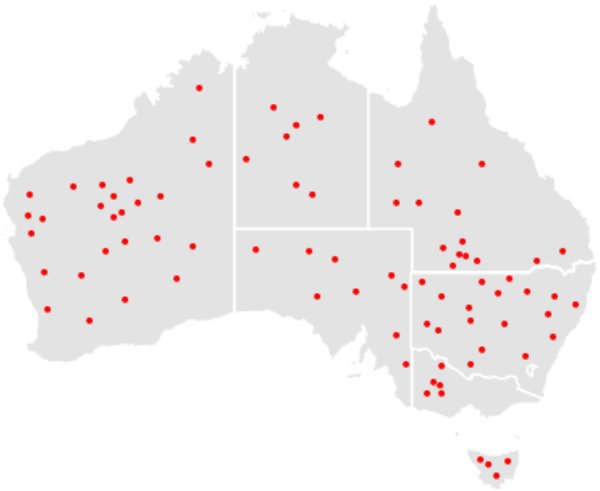

Figura 66: Ejemplo de mapa de puntos, tomado de (Ribecca, 2014)

Anexo 2. Rutinas de Programación

A2.1 Pre procesamiento de datos

```
1  import pandas as pd
2  from datetime import datetime
3  import savReaderWriter as spss
4
5
6  print ("Lectura de Archivo SAV SPSS")
7
8  filename = 'Casen 2015.sav'
9
10 instanteInicial = datetime.now()
11 raw_data = spss.SavReader(filename, returnHeader=True, rawMode=True)
12 raw_data_list = list(raw_data)
13 data = pd.DataFrame(raw_data_list)
14
```

Figura 67. Código en Python para leer datos desde un archivo .SAV

Como muestra la figura 67 el método SavReader en la línea 11 devuelve los datos en bruto, para ello es necesario cambiar el parámetro *rawMode* a true, esto significa que para los campos nulos se devuelve el valor -1.7976931348623157e+308 y los strings están en un formato de múltiplo de 8 bits. Esto se realiza para una lectura más rápida de los datos. Además se modifica el parámetro *returnHeader* a true, para que se pueda identificar a qué variable pertenece cada dato.

A continuación se debe analizar los datos y ordenarlos, de manera que queden listos para ser guardados en la base de datos, para ello se utilizará otra librería de Python llamada Pandas, que posee métodos que permiten ordenar los datos. Primero se crea una lista con los datos obtenidos y luego obtendremos los datos de manera tabular utilizando la clase *DataFrame* de Pandas en la línea 13.

Para ingresar los datos a la base de datos se usa Django, donde es posible crear registros de MySQL a través de clase *Model* de Django, se necesita primero importar las clases de Django, ya que la lectura de datos SPSS se realizaron en un archivo Python independiente. Esto es debido a que estos scripts no son parte del funcionamiento del servidor. En la figura

68 se muestra un ejemplo donde se importa la clase Django de los datos de la encuesta NESI sin becas, es decir, aquellos que reciben ingresos pero que no tienen ningún beneficio estatal dentro de sus ingresos.

```
7   import os,django,sys
8   sys.path.append("C:\primero")
9   os.environ.setdefault("DJANGO_SETTINGS_MODULE", "primero.settings")
10  from django.conf import settings
11  django.setup()
12  from apps.prueba.models import EncuestadoNesiSinBecas
13
```

Figura 68. Código en Python para importar una clase de Django

En la línea 8 se explicita la dirección del proyecto en Django, en la siguiente se definen los ajustes definidos en Django y luego en la línea 11 se usa el método *setup*() para cargar el framework. Finalmente se importa la clase del encuestado NESI sin becas.

Después de importar la clase de Django, se puede empezar a insertar los encuestados a la base de datos. La figura 69 muestra el procedimiento.

```
59  contador2 = 0
60  tupla = []
61
62  for x in range (0,data.shape[0]-1):
63
64      codigo = int(data.iloc[x+1,3])
65      if (codigo == 8):
66          contador2 = contador2 + 1
67          tupla.append(contador2)
68          for y in range (0,data.shape[1]):
69              valor = None if (str(data.iloc[x+1,y]) == '-1.7976931348623157e+308') else data.iloc[x+1,y]
70              tupla.append(valor)
71          encuestado = EncuestadoCASEN(*tupla)
72          tupla.clear()
73          encuestado.save()
74          print("ingresado 1 encuestado: "+ str(contador2))
75
```

Figura 69. Código para guardar un registro en la base de datos usando Django y los datos obtenidos del savReaderWriter

Como lo muestra la figura 69 primero se crea un arreglo en donde se irán guardando todas las respuestas del encuestado. En la línea 62 se recorren los datos con un bucle *for* y utilizando el método *shape* de pandas. Dentro del bucle *for*, en la línea 64 y 65 se verifica si el encuestado

pertenece a la octava región, que es la que va a guardarse en la base de datos. Luego se realiza otro bucle *for*, esta vez para recorrer las respuestas de cada encuestado. En la línea 70 se toma el valor de la respuesta, que puede ser un número entero, un número decimal o una cadena de caracteres. Si no encuentra ningún valor se agrega el nulo (None en Python). Después, en la línea 70 se guarda el dato en el arreglo, y así se va completando con las respuestas. Posteriormente se crea una instancia del modelo de Django que hace referencia a la tabla de base de datos, agregando el arreglo como argumento del constructor.

Antes de crear la instrucción en MySQL es necesario limpiar el arreglo para que pueda llenarse en la siguiente iteración. Finalmente, con el método *save*() se genera una instrucción que realiza el INSERT de SQL, almacenando un encuestado en la base de datos. Este proceso se repite hasta tener a todos los encuestados de la encuesta.

A2.2 Programando con Django

Para programar con Django se debe seguir con el patrón modelo-vista-template (vea sección 3.1.3). Para que esto sea posible, cuando se crea un proyecto de Django este crea un fichero donde se ordenan los elementos del proyecto. En la siguiente figura se ve un ejemplo.

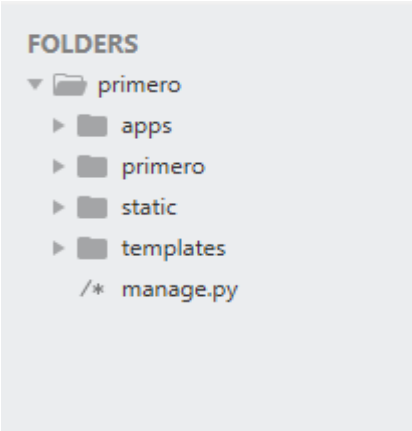

Figura 70. Estructura de proyecto de Django

Como se puede observar en la figura 70, existen 4 carpetas. En la primera se encuentran los archivos correspondientes a la aplicación, acá se encuentran los modelos, las vistas, los formularios y las urls. En la siguiente carpeta se encuentran los archivos correspondientes a la

configuración interna de Django para que funcione el proyecto siguiendo protocolos y de forma segura, por ejemplo, acá se realizan las configuraciones de la base de datos, agregando el usuario, la contraseña, y el tipo de base de datos. La penúltima carpeta contiene los archivos estáticos, que pueden ser, por ejemplo, los archivos de estilos css o los archivos de scripts de JavaScript. La última carpeta contiene los templates a utilizar en el proyecto.

Los modelos que representan los datos de las tablas de la base de datos. Por lo tanto se hacen 6 tablas, una que corresponde a los metadatos de las encuestas, una que corresponde a los valores que pueden tomar las respuestas por cada metadato, una donde se guardan los nombres de las encuestas, y por último las 3 tablas que corresponden a las respuestas de los encuestados, una para la NENE, una para la NESI y una para la CASEN. En la siguiente figura se muestra un ejemplo de un modelo en Django.

```
48  class CodigoPregunta(models.Model):
49      id = models.AutoField(primary_key=True,db_column='id')
50      variable = models.CharField(max_length=20)
51      nombre = models.CharField(max_length=100)
52      id_encuesta = models.ForeignKey('Encuesta',models.DO_NOTHING,db_column='id_encuesta')
53      class Meta:
54          managed = False
55          db_table = 'codigo_pregunta'
56
57      def __str__(self):
58          return self.nombre
```

Figura 71. Ejemplo del código de un modelo en Django

Como se puede observar en la figura 71, los modelos pueden tener una clase Meta en donde se pueden agregar funciones para los modelos.

Además de los modelos, en Django se deben definir los formularios a utilizar que hacen referencia a un modelo, esto para hacer más legible el código de html en los templates y también para poder usar el sistema de etiquetas de Django. En la siguiente figura se muestra un ejemplo de un formulario.

```python
class CodigoPreguntaForm(forms.ModelForm):

    class Meta:
        model = CodigoPregunta
        fields = [
            'variable','nombre','id_encuesta'
        ]
        labels = {
            'variable': 'Codigo variable',
            'nombre':'Nombre variable',
            'id_encuesta': 'Encuesta',
        }
        widgets = {
            'id_encuesta': forms.Select(attrs={'class':'form-control','id':'encuesta'}),
            'variable': forms.TextInput(attrs={'class':'form-control','id':'codigoVariable','name':'codigoVariable'}),
            'nombre':forms.TextInput(attrs={'class':'form-control','id':'nombreVariable','name':'nombreVariable'}),
        }

```

Figura 72. Ejemplo del código de un formulario en Django

Como se puede observar, el formulario debe tener una serie de campos para poder mostrarse en el template, el primero es al modelo al que hace referencia, el segundo son los atributos del modelo, luego se debe indicar como se llamarán estos atributos en el formulario, y al final se agregan opciones de cómo será el elemento html para cada atributo, incorporando opciones de clase css, identificadores, nombre, o cualquier campo de atributo de una etiqueta html.

Las vistas en Django corresponden al código que se encarga de realizar las consultas a la base de Datos para después cargar los templates con los datos pedidos. Se debe definir las funciones con la palabra clave *def* y en el prototipo se utiliza una función para cada acción que realiza el usuario que involucre consulta o inserción de datos, sin embargo dependiendo de la situación los datos son tratados de diferente manera. En algunos casos solo es necesario realizar la consulta y después mostrarlos en pantalla o guardarlos en la base de datos, en otros, sin embargo es necesario hacer operaciones adicionales para que los datos pasen desde la base de datos al cliente y viceversa. En la siguiente figura se muestra un ejemplo donde se explica esto.

```python
# Create your views here.
def fuerzaTrabajo(request):
    encuestados = EncuestadoNene.objects.only("fact","cae_general","cae_especifico")
    encuestadosJSON = serializers.serialize('json',encuestados, fields =["fact","cae_general","cae_especifico"])
    context = {'encuestados': encuestados, 'encuestadosJSON': encuestadosJSON }
    return render(request,'fuerza de trabajo.html',context)
```

Figura 73. Ejemplo de código de una vista en Django

Los templates de Django son, en términos generales, el código html que se mostrará al usuario, sin embargo, estos tienen una funcionalidad mucho más dinámica. Esto se logra

gracias a las etiquetas de Django que permiten, por ejemplo, hacer estructuras de control como una sentencia if o una sentencia for. Además se pueden heredar templates, permitiendo hacer una estructura html base y después modificar el contenido de acuerdo a la página en donde se diriga. Además se utilizan para hacer referencia a los datos que se entregan desde las vistas. En la siguiente figura se muestra un ejemplo de una template con etiquetas Django.

```
{% extends 'base4.html' %}
{% load staticfiles %}
{% block content %}



    <div class="row">
            <!--<h1 class="page-header">Cantidad de personas por ubicación geográfica</h1>-->
            <h1 class="page-header" id="titulo">Listado de valores</h1>
            <br>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <td class="text-center"><strong>#</strong></td>
                        <td class="text-center"><strong>Nombre valor</strong></td>
                        <td class="text-center"><strong>valor</strong></td>
                        <td class="text-center"><strong>Opciones</strong></td>
                    </tr>
                </thead>
                <tbody>
                    {% if opciones %}
                    {% for opcion in opciones %}
                    <tr>
                        <td class="text-center">{{opcion.id_opcion}}</td>
                        <td class="text-center">{{opcion.nombre_valor}}</td>
                        <td class="text-center">{{opcion.valor}}</td>
                        <td class="text-center">
                            <a href="{% url 'modificarOpcion' opcion.id_opcion %}" class="btn btn-primary">Editar</a>
                            <a href="{% url 'eliminarOpcion' opcion.id_opcion %}" class="btn btn-danger" onclick="return confirm('¿Estás seguro de borrar
                                este elemento?')">Eliminar</a>
                        </td>
                    </tr>
                    {% endfor %}
                    {% endif %}
                </tbody>

            </table>
    </div>
{% endblock %}

{% block otrosscripts %}
{% endblock %}
```

Figura 74. Ejemplo del código de un template en Django

Para agregar funcionalidades del lado del cliente a las vistas se utilizan la etiqueta *load staticfiles* para acceder a los archivos JavaScript o a los estilos css para agregar un diseño personalizado.

A2.3 Programación del Procesamiento de Datos del Prototipo

Mantenedor de variables de encuesta

Agregar Variable

```
131  def agregarVariable(request):
132
133      if(request.method == 'POST'):
134          form = CodigoPreguntaForm(request.POST)
135          if form.is_valid():
136              form.save()
137              return redirect('index')
138      else:
139          form = CodigoPreguntaForm()
140      return render(request,'agregarVariable.html',{'formularioPregunta':form})
141
142
143  def modificarVariable(request, id_variable):
144      variable = CodigoPregunta.objects.get(id=id_variable)
```

Figura 75. Código función agregar Variable en Django

Como muestra la figura 75 esta función realiza dos acciones dependiendo de la solicitud Http. Si recibe un POST, se carga un formulario con los datos recibidos (línea 134), luego Django se cerciora que el formulario cumpla con las condiciones del modelo (línea 135) para luego realizar un INSERT MySQL a través de la función *save()*. Si recibe un GET se crea un formulario vacío y se carga el template con el formulario.

Modificar Variable

```
143  def modificarVariable(request, id_variable):
144      variable = CodigoPregunta.objects.get(id=id_variable)
145
146      if request.method == 'GET':
147          form = CodigoPreguntaForm(instance=variable)
148      else:
149          form = CodigoPreguntaForm(request.POST,instance=variable)
150          if form.is_valid():
151              form.save()
152              return redirect('listarVariables')
153      return render(request,'agregarVariable.html',{'formularioPregunta':form})
```

Figura 76. Código función modificar variable en Django

La función de la figura 76 es similar a la anterior, sin embargo aparte de la solicitud Http, esta función recibe el id referente a la variable a modificar. Al principio, en la línea 144 se realiza una consulta SQL donde se pide la variable a modificar, y al momento de crear el formulario (línea 147 y línea 149) se ingresa como parámetro *Instance* a la variable. De esta forma se muestra el formulario con los datos de la variable. La función *save()* esta vez realiza un UPDATE en vez de un INSERT a la base de datos.

Eliminar Variable

```
160  def eliminarVariable(request,id_variable):
161      variable = CodigoPregunta.objects.get(id=id_variable)
162      variable.delete()
163      return redirect('listarVariables')
```

Figura 77. Código función eliminar Variable en Django

Como se puede observar en la figura 77, esta función recibe la solicitud Http y el id de la variable a eliminar, primero se realiza la consulta para obtener la variable y luego se realiza el DELETE en SQL a través de la función Django *delete().* Antes de realizar esta operación en el template se encuentra un script en JavaScript que realiza una confirmación. Esto se puede ver en la siguiente figura.

```
<a href="{% url 'eliminarVariable' variable.id %}" class="btn btn-danger" id="eliminar" onclick="return
    confirm('¿Estás seguro de borrar este elemento?')">Eliminar</a>
```

Figura 78. Etiqueta HTML con una función JavaScript en el parámetro onclick

Como se muestra en la figura 78, en el template la etiqueta <a> que hace referencia a la función eliminar tiene el parámetro onclick que ejecuta la función especificada en el parámetro al momento de clickear, esta función en JavaScript utiliza la función *confirm()* que despliega un mensaje de confirmación en la página antes de realizar la eliminación.

Listar Variables

```
155  def listarVariables(request):
156      variables = CodigoPregunta.objects.all()
157      context = {'variables':variables}
158      return render(request,'listarVariables.html',context)
```

Figura 79. Código función listar variables en Django

La función de la figura 79 recibe la solicitud Http y lo que hace es primero hacer una consulta SQL para pedir a todas las variables utilizando la función *objects.all()* de Django, luego se crea un diccionario con las variables y al final se carga el template y se envían las variables.

Agregar Valores

```
188  def agregarOpcion(request):
189      form = OpcionForm()
190      if request.method == 'POST':
191          i=1
192          variable = CodigoPregunta.objects.get(id=request.POST["id_pregunta"])
193          while "Valor"+str(i) in request.POST:
194              valor = int(request.POST["Valor"+str(i)])
195              nuevaOpcion = Opcion(id_pregunta=variable,nombre_valor=request.POST["nombreValor"+str(i)],valor=valor)
196              nuevaOpcion.save()
197              i = i+1
198          return redirect('listarVariables')
199      else:
200          return render(request,'agregarOpcion.html',{'formularioOpcion':form})
```

Figura 80. Código función agregar valores en Django

Como lo muestra la figura 80, esta función funciona de manera similar a la de agregación de variables, sin embargo, esta función admite agregar múltiples valores de una sola variable a la vez. Para lograr esto se creó un formulario para un solo valor y con la ayuda de JQuery se logra hacer una carga dinámica de valores. Cuando se reciben los valores, se realiza una estructura while in donde se crean los nuevos valores agregando directamente los atributos del modelo con los valores recibidos del arreglo POST, luego se usa la función *save()* para hacer un INSERT En JQuery se utilizan dos funciones: una que agrega el html de un valor y otra que refresca los parámetros id y name de las etiquetas input de html para que se diferencien al momento de crear las instancias de los modelos.

```
23  function agregaOpcion(){
24      subForm = '<div id="subForm">'+
25                      '<h4 style="padding-left: 50px;"></h4>'+
26                      '<div class="form-group">'+
27                          '<label class="col-sm-2 control-label">Nombre Valor</label>'+
28                                  '<div class="col-sm-10">'+
29                                    '<input class="form-control" id="" name="">'+
30                                    '<span class="help-block">Campo obligatorio</span>'+
31                                  '</div>'+
32                      '</div>'+
33                      '<div class="form-group">'+
34                          '<label class="col-sm-2 control-label">Valor</label>'+
35                          '<div class="col-sm-10">'+
36                            '<input class="form-control" id="" name="" type="number">'+
37                            '<span class="help-block">Campo obligatorio</span>'+
38                          '</div>'+
39                      '</div>'+
40                  '</div>';
41      $("#codigos").append(subForm);
42  }
```

Figura 81. Función JavaScript para agregar dinamismo al ingreso de valores en el template

Como se muestra en la figura 81, la función consta de un string que contiene las etiquetas html y al final en la línea 41 se realiza la función JQuery *append()* que agrega el contenido el string en la página. Después de esta función se ejecuta la función refresh.

```
1   function refresh(){
2       var i=1;
3       var contador = 1;
4       $('#codigos').children().each(function(){
5           $(this).find('h4').html('Valor '+i)
6           $(this).find('input').each(function(){
7               if(contador == 1){
8                   $(this).attr('id','nombreValor'+i);
9                   $(this).attr('name','nombreValor'+i);
10                  contador++;
11              }else{
12                  $(this).attr('id','Valor'+i);
13                  $(this).attr('name','Valor'+i);
14                  contador=1;
15              }
16          });
17          i++;
18      })
19  }
20
```

Figura 82. Función JavaScript para actualizar parámetros HTML de los input

Como se observa en la figura 82, lo que hace esta función es cambiar los parámetros id y name de las etiquetas input, para que cuando se envíen por medio de la solicitud POST la función de Django pueda agregar correctamente los valores.

Eliminar Valor

```
182  def eliminarOpcion(request,id_opcion):
183      opcion = Opcion.objects.get(id_opcion=id_opcion)
184      opcion.delete()
185      return redirect('listarVariables')
```

Figura 83. Código función eliminar valor en Django

La función de la figura 83 funciona de la misma manera que la función para eliminar variables, la única diferencia es que se usa el modelo de los valores. Se crea una instancia del modelo que hace referencia a el valor en la base de datos y luego se ejecuta la consulta DELETE a través de la función *delete()* de Django.

Modificar Valor

```
170  def modificarOpcion(request,id_opcion):
171      opcion = Opcion.objects.get(id_opcion=id_opcion)
172
173      if request.method == 'POST':
174          form = OpcionForm(request.POST,instance=opcion)
175          if form.is_valid():
176              form.save()
177              return redirect('listarVariables')
178      else:
179          form = OpcionForm(instance=opcion)
180      return render(request,'modificarOpcion.html',{'formularioOpcion':form})
```

Figura 84. Código función modificar valor en Django

Como se observa en la figura 84, esta función funciona de la misma manera que la función de modificar variable, solo que se utiliza la instancia de un valor en lugar de una variable. Se destaca que en esta forma de modificar no se usa un formulario dinámico por eso es diferente al código para agregar valores, a diferencia de las variables, cuyo código de agregar y modificar es muy similar entre ellos.

Listar Valores

```
165  def listarOpciones(request,id_variable):
166      opciones = Opcion.objects.filter(id_pregunta=id_variable)
167      context = {'opciones':opciones}
168      return render(request,'listarOpciones.html',context)
```

Figura 85. Código función listar valores en Django

La función de la figura 85 es similar al listado de variables, la diferencia radica en que esta función recibe el parámetro que hace referencia a la id de la variable aparte de la solicitud Http, y también en la consulta SQL. Aquí se utiliza la función *objects.filter()* que genera una consulta de todos los valores, pero como argumento se debe ingresar una condición que irá en el WHERE de la consulta SQL, para filtrar los datos.

A2.3 Visualizaciones Predefinidas

Conmutación Laboral

```
31  def empleo(request):
32      encuestados = EncuestadoNene.objects.only("r_p_c","b18_codigo","fact")
33      encuestadosJSON = serializers.serialize('json',encuestados, fields =["r_p_c","b18_codigo","fact"])
34      context = {'encuestados': encuestados, 'encuestadosJSON': encuestadosJSON }
35      return render(request,'empleo.html',context)
36
```

Figura 86. Código función para obtener datos de conmutación laboral en Django

Como muestra la figura 86 esto se logra mediante una vista Django que realiza una consulta SQL pidiendo las variables “r_c_p” para la comuna de residencia, la variable “b18_codigo” que hace referencia a la pregunta en la encuesta: ¿En qué comuna o localidad se ubica la empresa, negocio, institución o actividad por cuenta propia donde realizó su trabajo la semana de referencia? , y por último, la variable “fact” que hace referencia al factor de expansión del encuestado, para que los porcentajes mostrados en la visualización estén expandidos estadísticamente hablando. Después de esto se realiza una serialización en JSON que provee Django en la función *serializers.serialize(),* finalmente se carga el template y se envían los resultados.

En el template se realiza un cambio en el JSON para proteger al arreglo de errores de sintaxis cuando se utilice en el archivo JavaScript, esto se logra utilizando las etiquetas Django de los templates con la palabra reservada *escapejs*

```
 9        <script >
10
11            var encuestadosJSON = "{{encuestadosJSON|escapejs}}";
12            var encuestadosJS = JSON.parse(encuestadosJSON);
13
14        </script>
```

Figura 87. Código JavaScript para eliminar errores de sintaxis usando etiquetas Django

Después de ello, como muestra la figura 87, se cambia el formato JSON a objetos de JavaScript con la función *JSON.parse()* en la línea 12.

Lo siguiente es trabajar con el arreglo de objetos de JavaScript para transformarlo en una visualización, para ello se utiliza una función como muestra la siguiente figura

```
function Visualización(array){

    var total = 0;
    var encuestados = new Array();
    var nodos = new Array();
    var links = new Array();
    var comunas = ["Alto Biobío","Antuco","Arauco","Bulnes","Cabrero","Cañete",
                   "Chiguayante","Chillán","Chillán Viejo","Cobquecura","Coelemu","Coihueco","Concepción","Contulmo",
                   "Coronel","Curanilahue","El Carmen","Florida","Hualqui","Hualpén","Laja","Lebu","Los Álamos","Los Ángeles",
                   "Lota","Mulchén","Nacimiento","Negrete","Ninhue","Ñiquén","Pemuco","Penco","Pinto","Portezuelo","Quilaco","Quilleco",
                   "Quillón","Quirihue","Ránquil","San Carlos","San Fabián","San Ignacio","San Nicolás","San Pedro de la Paz",
                   "San Rosendo","Santa Bárbara","Santa Juana","Talcahuano","Tirúa","Tomé","Trehuaco","Tucapel","Yumbel","Yungay","comuna fuera de l

    for(var i in array){
        encuestados.push(array[i].fields);
    }

    for(var j in encuestados){
        //factor
        encuestados[j].fact = parseInt(encuestados[j].fact);
        total = total + encuestados[j].fact;
        //obteniendo los nombres de las provincias y las comunas
        comuna_provincia = getComunaProvincia(encuestados[j].r_p_c);
        encuestados[j].comuna = comuna_provincia[0];
        encuestados[j].provincia = comuna_provincia[1];
    }

    // creando nodos para grafo
    if(document.getElementById("red")!= null){
        for (var i =0, h = 0, v = 0; i < comunas.length; i++) {
            nodos[i] = {"id" : comunas[i],"x" : h,"y" : v,"fact" : 0};
            h++;
            if(h == 11){v++; h = 0;}
        }
    //creando links
        for(var j in encuestados){
            ubicacion_trabajo = getComunaProvincia(encuestados[j].b18_codigo);
            comuna_trabajo = ubicacion_trabajo[0];
            if(!existeLink(comunas.indexOf(encuestados[j].comuna),comunas.indexOf(comuna_trabajo),links) && comuna_trabajo != encuestados[j].comuna){
                links.push({"source": comunas.indexOf(encuestados[j].comuna), "target": comunas.indexOf(comuna_trabajo)});
                agregarFact(nodos,encuestados[j].fact,comuna_trabajo);
            }
            else if(comuna_trabajo != encuestados[j].comuna){
                agregarFact(nodos,encuestados[j].fact,comuna_trabajo);
            }

        }

        var tooltip = {
            body: function(d) {
              var porcentaje = (d.fact/total)*100;
              porcentaje = porcentaje.toFixed(2);
              var table = "<table class='tooltip-table'>";
              table += "<tr><td class='title'>trabajan aquí:</td><td class='data'>" + porcentaje + "%</td></tr>";
              table += "</table>";
              return table;
            },
            footer: function(d) {
              return "<sub class='tooltip-footer'>Enero-Febrero-Marzo 2018</sub>";
            },
            title: function(d) {
              var txt = d.id;
              return txt.charAt(0).toUpperCase() + txt.substr(1).toLowerCase();
            }
        };
        var onclick = function(d){
            arreglo = new Array();
            for(j in links){
                if(links[j].target == comunas.indexOf(d.id)){
                    var nombre_nodo = comunas[links[j].source];
                    for(x in nodos){
                        if (nodos[x].id == nombre_nodo){
                            arreglo.push({"name": nombre_nodo, "fact" : nodos[x].fact});
                            break;
                        }
                    }
                }
            }
            console.log(arreglo);
            if ($("#titulo").length > 0) {
                while(document.getElementById("titulo").hasChildNodes()){
                    document.getElementById("titulo").removeChild(document.getElementById("titulo").firstChild);
                }
                $("#titulo").append("<h3> Comunas cuya conmutación laboral tiene como objetivo a "+d.id+"</h3>");
            }else{
                $("#titulo").append("<h3> Comunas cuya conmutación laboral tiene como objetivo a "+d.id+"</h3>");
            }
            espacio = document.getElementById("treemap-red");
            while(espacio.hasChildNodes()){
                espacio.removeChild(espacio.firstChild);
            }
            crearTreemapSinAJAX('fact','name','treemap-red',arreglo);
        };

        crearGrafo('red',nodos,links,tooltip,onclick,'fact');

    }
}
```

Figura 88. Función JavaScript con D3 plus y JQuery para visualizar la conmutación Laboral

Como muestra la figura 88, la serialización que realiza Django genera un string que separa la clave primaria del encuestado del resto de los campos, por ello lo primero que se debe hacer es separar los campos por encuestado, esto se realiza en la estructura for de la línea 178, luego se realiza un parseo a número entero al factor de expansión, ya que la serialización a JSON transformó los números a caracteres. También se obtienen las comunas y provincias gracias a una función de JavaScript que transforma el número del código de comuna en el nombre de la comuna y el nombre de la provincia a la que pertenece cada encuestado. En la línea 192 se crea el arreglo de nodos y en la línea 200 se crea el arreglo de links. En la línea 230 se crea la función de interacción, esta función crea el Treemap después de hacer click en un nodo del grafo, esto se hace en la línea 256. Finalmente se crea el Grafo en la línea 257.

Brechas de Ingresos

La siguiente figura muestra cómo se obtiene la línea de pobreza usando el número de habitantes de un hogar.

```
1   function getLineaPobreza(numeroHabitantesHogar) {
2
3       if(numeroHabitantesHogar == 1){
4           return 148974;
5       }else if(numeroHabitantesHogar == 2){
6           return 242009;
7       }else if(numeroHabitantesHogar == 3){
8           return 321437;
9       }else if(numeroHabitantesHogar == 4){
10          return 393145;
11      }else if(numeroHabitantesHogar == 5){
12          return 459611;
13      }else if(numeroHabitantesHogar == 6){
14          return 522176;
15      }else if(numeroHabitantesHogar == 7){
16          return 581674;
17      }else if(numeroHabitantesHogar == 8){
18          return 638666;
19      }else if(numeroHabitantesHogar == 9){
20          return 693555;
21      }else{
22          return 746640;
23      }
24  }
```

Figura 89. Función para obtener línea de Pobreza en JavaScript

Como muestra la figura 89 se retorna un valor numérico dependiendo del número de habitantes del hogar del encuestado. La función para la pobreza extrema usa la misma lógica pero con diferentes valores de retorno.

Para determinar la brecha que existe es necesario saber para los encuestados pobres y extremadamente pobres cuanto ingreso necesitan para salir de la pobreza, para ello se creó una función que determina por rango cuanto se necesita para salir de ese estado.

```
50  function getBrecha(Linea, ingreso){
51      diferencia = linea - ingreso;
52      if( diferencia < 10000){
53          return "menor o igual a $10.000";
54      }else if(diferencia > 10000 && diferencia <= 25000){
55          return "entre $10.001 y $25.000";
56      }else if(diferencia >25000 && diferencia <= 50000){
57          return "entre $25.001 y $50.000";
58      }else if(diferencia > 50000 && diferencia <= 75000){
59          return "entre $50.001 y $75.000";
60      }else if(diferencia >75000 && diferencia <= 100000){
61          return "entre $75.001 y $100.000";
62      }else{
63          return "mayor a $100.000";
64      }
65  }
```

Figura 90. Función para determinar brecha de ingresos en JavaScript

Como muestra la figura 90, se determinaron 6 rangos para determinar las brechas de ingresos, dependiendo de la línea que se recibe como parámetro y del ingreso del hogar del encuestado.

Para poder asignar en que categoría de pobreza se encuentra un hogar se optó por separar a los encuestados en dos arreglos antes de crear el Treemap. Para ello se usa una función que separa el arreglo original en dos y agrega campos referentes a la pobreza por ingresos y a la brecha de ingresos.

```
function determinarPobreza(encuestados,encuestadosPobres,encuestadosMuyPobres){

    for(var j in encuestados){
        lineaPobreza = getLineaPobreza(encuestados[j].TOTPERH);
        lineaPobrezaExtrema = getLineaPobrezaExtrema(encuestados[j].TOTPERH);
        if(encuestados[j].ING_AUT < lineaPobrezaExtrema){
            encuestados[j].pobreza = "Pobreza Extrema";
            encuestadosPobres.push(encuestados[j]);
            encuestadosPobres[encuestadosPobres.length-1].linea = lineaPobrezaExtrema;

        }else if(encuestados[j].ING_AUT < lineaPobreza){
            encuestados[j].pobreza = "Pobreza";
            encuestadosMuyPobres.push(encuestados[j]);
            encuestadosMuyPobres[encuestadosMuyPobres.length -1].linea = lineaPobreza;

        }else{
            encuestados[j].pobreza = "No pobre";
        }


    }
    for(var h in encuestadosPobres){
        encuestadosPobres[h].brecha = getBrecha(encuestadosPobres[h].linea,encuestadosPobres[h].ING_AUT);
    }
    for(var k in encuestadosMuyPobres){
        encuestadosMuyPobres[k].brecha = getBrecha(encuestadosMuyPobres[k].linea,encuestadosMuyPobres[k].ING_AUT);
    }
}
```

Figura 91. Función para determinar la situación de pobreza en JavaScript

Como muestra la figura 91, primero se determina la línea de pobreza y de extrema pobreza para cada encuestado, luego se determina si existe pobreza extrema, pobreza o no pobre. En caso de las dos primeras se agrega como campo del encuestado la línea de pobreza o extrema pobreza según corresponda.

Después de ello, en la línea 88 para los arreglos de pobres y extremadamente pobres se determina la brecha para cada uno.

Como la encuesta NESI separa en tablas diferentes a los encuestados con o sin beneficios del estado, en la consulta a la base de datos se debe pedir por ambos, como lo muestra la figura 92.

```
def ingresos(request):
    encuestadosSB = EncuestadoNesiSinBecas.objects.only("ING_AUT","TOTPERH","FACT_HOG")
    encuestadosCB = EncuestadoNesiConBecas.objects.only("ING_AUT","TOTPERH","FACT_HOG")

    encuestadosSB_JSON = serializers.serialize('json',encuestadosSB, fields =["ING_AUT","DECILH","TOTPERH","FACT_HOG"])
    encuestadosCB_JSON = serializers.serialize('json',encuestadosCB, fields =["ING_AUT","DECILH","TOTPERH","FACT_HOG"])

    context = {'encuestadosSB_JSON':encuestadosSB_JSON,'encuestadosCB_JSON':encuestadosCB_JSON}
    return render(request,'ingresos.html',context)
```

Figura 92. Función para obtener los datos de ingresos en Django

Como se observa en la figura 92, se realiza la misma consulta, con los mismos parámetros, pero a diferentes tablas, luego se serializan por separado y al final en la línea 44 se juntan en un diccionario para enviarlos al template.

```
95
96  function visualizacion(array,array2) {
97      var encuestadosCB = new Array();
98      var encuestadosPobresCB = new Array();
99      var encuestadosMuyPobresCB = new Array();
100     var encuestadosSB = new Array();
101     var encuestadosPobresSB = new Array();
102     var encuestadosMuyPobresSB = new Array();
103
104     for(var i in array){
105         encuestadosCB.push(array[i].fields);
106         encuestadosCB[encuestadosCB.length-1].FACT_HOG = parseInt(encuestadosCB[encuestadosCB.length-1].FACT_HOG);
107         encuestadosCB[encuestadosCB.length-1].TOTPERH = parseInt(encuestadosCB[encuestadosCB.length-1].TOTPERH);
108
109     }
110     for(var i in array2){
111         encuestadosSB.push(array2[i].fields);
112         encuestadosSB[encuestadosSB.length-1].FACT_HOG = parseInt(encuestadosSB[encuestadosSB.length-1].FACT_HOG);
113         encuestadosSB[encuestadosSB.length-1].TOTPERH = parseInt(encuestadosSB[encuestadosSB.length-1].TOTPERH);
114     }
115
116     determinarPobreza(encuestadosCB,encuestadosPobresCB,encuestadosMuyPobresCB);
117     determinarPobreza(encuestadosSB,encuestadosPobresSB,encuestadosMuyPobresSB);
118
119
120     var dataInicial = [
121         {nombre : "Con Beneficios estatales",sum : encuestadosCB.length},
122         {nombre : "Sin Beneficios estatales",sum : encuestadosSB.length},
123     ];
124
125     var onclickCB = function(d) {
126                     $("#treemap-habitantes").append("<h3>Brechas de ingresos para superar la pobreza<h3><br>")
127                     if(d.pobreza == "Pobreza"){
128                         espacio = document.getElementById("treemap-habitantes");
129                         while(espacio.hasChildNodes()){
130                             espacio.removeChild(espacio.firstChild);
131                         }
132                         crearTreemapSinAJAX('FACT_HOG',["pobreza","brecha"],'treemap-habitantes',encuestadosPobresCB);
133
134                     }else if (d.pobreza == "Pobreza Extrema"){
135                         espacio = document.getElementById("treemap-habitantes");
136                         while(espacio.hasChildNodes()){
137                             espacio.removeChild(espacio.firstChild);
138                         }
139                         crearTreemapSinAJAX('FACT_HOG',["pobreza","brecha"],'treemap-habitantes',encuestadosMuyPobresCB);
140                     }else{
141                         espacio = document.getElementById("treemap-habitantes");
142                         while(espacio.hasChildNodes()){
143                             espacio.removeChild(espacio.firstChild);
144                         }
145                     }
146                 };
147     var onclickSB = function(d) {
148                     if(d.pobreza == "Pobreza"){
149                         espacio = document.getElementById("treemap-habitantes");
150                         while(espacio.hasChildNodes()){
151                             espacio.removeChild(espacio.firstChild);
152                         }
153                         crearTreemapSinAJAX('FACT_HOG',["pobreza","brecha"],'treemap-habitantes',encuestadosPobresSB);
154
155                     }else if (d.pobreza == "Pobreza Extrema"){
156                         espacio = document.getElementById("treemap-habitantes");
157                         while(espacio.hasChildNodes()){
158                             espacio.removeChild(espacio.firstChild);
159                         }
160                         crearTreemapSinAJAX('FACT_HOG',["pobreza","brecha"],'treemap-habitantes',encuestadosMuyPobresSB);
161                     }
162                     else{
163                         espacio = document.getElementById("treemap-habitantes");
164                         while(espacio.hasChildNodes()){
165                             espacio.removeChild(espacio.firstChild);
166                         }
167                     }
168                 }
169
170     crearTreemapSinAJAX('sum','nombre','treemap',dataInicial,function(d){
171
172         espacio = document.getElementById("treemap-pobreza");
173             while(espacio.hasChildNodes()){
174                 espacio.removeChild(espacio.firstChild);
175             }
176
177             if(d.nombre == "Con Beneficios estatales"){
178                 espacio = document.getElementById("treemap-habitantes");
179                 while(espacio.hasChildNodes()){
180                     espacio.removeChild(espacio.firstChild);
181                 }
182                 crearTreemapSinAJAX('FACT_HOG',"pobreza",'treemap-pobreza',encuestadosCB,onclickCB);
183
184
185             }else{
186                 espacio = document.getElementById("treemap-habitantes");
187                 while(espacio.hasChildNodes()){
188                     espacio.removeChild(espacio.firstChild);
189                 }
190                 crearTreemapSinAJAX('FACT_HOG',"pobreza",'treemap-pobreza',encuestadosSB,onclickSB);
191             }
192     });
193
```

Figura 93. Función JavaScript con D3 plus para la brecha de ingresos

Desigualdad

```python
def desigualdad(request):
    encuestadosCASEN = EncuestadoCASEN.objects.only("yautcorh","dautr","expr")

    encuestadosCASEN_JSON = serializers.serialize('json',encuestadosCASEN, fields =["yautcorh","dautr","expr"])

    context = {'encuestadosCASEN_JSON':encuestadosCASEN_JSON}
    return render(request,'desigualdad.html',context)

```

Figura 94. Función para obtener los datos para la desigualdad en Django

Al igual que en la visualización predefinida anterior, se realiza una serialización a JSON para enviar los datos a JavaScript usando también la template para ello.

Para el cálculo del índice 10/10 primero se realizó un cálculo de promedio del ingreso autónomo del hogar del decil 1 y el decil 10, para ello se crean dos funciones que recorren el arreglo de encuestados y realizan el cálculo con una estructura de control if. Para el cálculo del índice 20/20 se realiza lo mismo pero en la estructura de control if se pregunta si es que el encuestado pertenece a los dos primeros o dos últimos deciles, es decir el decil 1 y 2 para los pobres y el decil 9 y 10 para los ricos. La siguiente figura muestra el ejemplo del código para obtener el promedio de ingresos del 10% más rico en la región.

```javascript
function promedio10MasRico(array) {
    total = 0;
    promedio = 0;
    cantidad = 0;
    for(var j in array){
        if(array[j].dautr == 10){
            cantidad = cantidad +1;
            total = total + parseInt(array[j].yautcorh);
        }
    }

    promedio = total/cantidad;
    return promedio;
}
```

Figura 95. Función JavaScript para obtener el 10% más rico

Para el cálculo del promedio del ingreso autónomo del hogar por deciles se crea una función llamada *promedioDecilX()* que recibe el arreglo de encuestados y el decil que se desea sacar el promedio.

```
function promedioDecilX(array,decil){
    total = 0;
    promedio = 0;
    cantidad = 0;
    for(var j in array){
        if(array[j].dautr == decil){
            cantidad = cantidad +1;
            total = total + parseInt(array[j].yautcorh);
        }
    }
    promedio = total/cantidad;
    return promedio;
}
```

Figura 96. Función JavaScript para obtener el promedio de ingresos de un decil.

Como muestra la figura 96, esta función funciona muy parecido a los del ejemplo anterior con la diferencia de que esta es más dinámica y puede funcionar para cualquier decil. La única diferencia reside en la estructura de control if donde se pregunta por el decil recibido como parámetro.

Lo que queda ahora es codificar la visualización, para ello se usará un Treemap que contendrá todos los deciles y su proporción regional. Cuando se presione click sobre un decil se desplegará un Grafo que contendrá tres partes: la primera el índice 10/10, la segunda el índice 20/20 y la tercera la relación entre el decil seleccionado en el Treemap con el decil 10.

```
134
135 function visualizacion(array) {
136
137     var encuestados = new Array();
138     var masRicos10 = 0;
139     var masRicos20 = 0;
140     var masPobres10 = 0;
141     var masPobres20 = 0;
142
143
144     for(var i in array){
145         encuestados.push(array[i].fields);
146     }
147
148     for(var i in encuestados){
149         encuestados[i]. dautr = parseInt(encuestados[i].dautr);
150         encuestados[i]. expr = parseInt(encuestados[i].expr);
151         encuestados[i]. yautcorh = parseInt(encuestados[i].yautcorh);
152         encuestados[i]. decil = getDecil(encuestados[i].dautr);
153     }
154
155     masRicos10 = promedio10MasRico(encuestados);
156     masPobres10 = promedio10MasPobre(encuestados);
157     var indice10 = masRicos10/masPobres10;
158     indice10 = indice10.toFixed(2) + " veces";
159     masRicos20 = promedio20MasRico(encuestados);
160     masPobres20 = promedio20MasPobre(encuestados);
161     var indice20 = masRicos20/masPobres20;
162     indice20 = indice20.toFixed(2)+" veces";
163
164     var promDecil10 = promedioDecilX(encuestados,10);
165
166     var nodos = [
167         {"id":"10% más Pobre", "x":1,"y":1},
168         {"id":"10% más Rico", "x":5,"y":1},
169         {"id":indice10, "x":3,"y":1},
170         {"id":"20% más Pobre", "x":1,"y":3},
171         {"id":"20% más Rico", "x":5,"y":3},
172         {"id":indice20, "x":3,"y":3},
173         {"id":"Decil 10","x":1,"y":5},
174     ];
175
176     links = [
177         {"source":0,"target":2},
178         {"source":1,"target":2},
179         {"source":3,"target":5},
180         {"source":4,"target":5},
181         {"source":6,"target":8},
182         {"source":7,"target":8},
183     ];
184
185     var onclick = function(d) {
186             console.log(d);
187             var promDecilX = promedioDecilX(encuestados,getNumeroDecil(d.decil));
188             var indice = promDecil10/promDecilX;
189             if(indice == 1){
190                 indice = "es 1 vez";
191                 var nodoDecil = {"id":"decil 10","x":5,"y":5};
192             }
193             else{
194                 indice = "es "+indice.toFixed(2) + " veces";
195                 var nodoDecil = {"id":d.decil,"x":5,"y":5};
196             }
197
198             var nodoProm = {"id":indice,"x":3,"y":5};
199             nodos.splice(7,2,nodoDecil,nodoProm);
200
201             espacio = document.getElementById("grafo");
202             while(espacio.hasChildNodes()){
203                 espacio.removeChild(espacio.firstChild);
204             }
205
206             crearGrafo('grafo',nodos,links);
207
208             };
209
210     crearTreemapSinAJAX('expr','decil','treemap-desigualdad',encuestados,onclick);
211 }
```

Figura 97. Función JavaScript más D3plus para la visualización de la desigualdad

Como muestra la figura 97, lo que se hace desde la línea 137 a la 153 es preparar los datos y declarar las variables a utilizar, convirtiendo a entero los valores numéricos que se habían transformado a string con el JSON. Desde la línea 155 a la 162 se obtienen los índices 10/10 y 20/20. En la línea 165 se obtiene el promedio de ingreso del decil 10 y desde la línea 166 a la 183 se preparan los arreglos de nodos y links que se usarán para visualizar el grafo.

En la línea 185 se crea el objeto Treemap de D3plus que recibe los datos de los encuestados ordenados por decil. En la función on() en la línea 191 se escribe la función para crear los grafos. Cuando se presiona click sobre algún decil se calcula el promedio de ese decil con la función promedioDecilX() y después se crea el nuevo índice. El objeto grafo no permite que existan nodos con igual identificador, así que es necesario hacer una modificación cuando en el Treemap se presiona el decil 10, por lo que en la línea 198 se crea un nodo decil 10 que comienza con minúscula para diferenciar del original que comienza con mayúscula. Después de eso se crean los nodos que faltan y se agregan al arreglo de nodos con la función splice() que recibe 4 parámetros: los dos primeros es para eliminar los nodos que existan al final del arreglo, en la posición 7 y 8, ya que de lo contrario cada vez que se presione un decil el arreglo de nodos se incrementaría en tamaño y solo se quiere mostrar tres relaciones. Los otros dos parámetros son los nodos a agregar.Finalmente se limpia la página y se crea el objeto network de D3plus dando como parámetros a los nodos y los links.

Visualizaciones con Heatmap

```
122  function crearArregloCoordenadasComuna(datos){
123      var comunas = [];
124      for(var comuna in datos){
125          var nuevo = {"lat":datos[comuna].LATITUD,"lng":datos[comuna].LONGITUD,"count":0,"nombre":datos[comuna].NOMBRE_COMUNA};
126          comunas.push(nuevo);
127      }
128      return comunas;
129  }
130
```

Figura 98. Función para preparar arreglo para Heatmap

La función retorna un arreglo que tiene los datos de latitud, longitud, nombre y un campo que contendrá la suma de los factores de expansión, este se debe inicializar en 0. Con este arreglo

más los datos pedidos, se puede crear el heatmap geográfico. La siguiente figura muestra cómo se crea un heatmap geográfico.

```
function crearHeatMap(datos,variable,factor,comunas,ubicacion){
    //esta funcion utiliza la librería hetmap.js y el plugin leaflet heatmap
    for(var x in datos){
        for(var i in comunas){
            if(datos[x][variable] == comunas[i].nombre){
                comunas[i].count = comunas[i].count + datos[x][factor];
                break;
            }
        }
    }

    var testData = {
      max: 8,
      data: comunas,
    };

    var baseLayer = L.tileLayer(
      'http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',{
        attribution: 'Map data &copy; <a href="http://openstreetmap.org">OpenStreetMap</a> contributors,' +
        '<a href="http://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, Imagery © <a href="http://cloudmade.com">CloudMade</a>',
        maxZoom: 18
      }
    );

    var cfg = {
      "radius": 0.1,
      "maxOpacity": .7,
      "scaleRadius": true,
      "useLocalExtrema": true,
      latField: 'lat',
      lngField: 'lng',
      valueField: 'count'
    };


    var heatmapLayer = new HeatmapOverlay(cfg);

    var map = new L.Map(ubicacion, {
      center: new L.LatLng(-36.772778,-73.063056),
      zoom: 9,
      layers: [baseLayer, heatmapLayer]
    });

    heatmapLayer.setData(testData);

}
```

Figura 99. Función JavaScript para crear un Heatmap

Las primeras líneas de la función tienen como objetivo completar el campo de la suma de los factores de expansión del arreglo para el heatmap, luego se crea la capa de Leaflet que contiene el mapa y el zoom máximo. Luego se crea una variable donde se ingresa la configuración del heatmap, como el radio, la opacidad, el campo de latitud, longitud y el valor, entre otros. Luego en la línea 110 se crea la capa del heatmap, y al final se crea el mapa completo, se ingresa datos iniciales, como la coordenada del centro del mapa a mostrar, el zoom por defecto, y las capas que incluirá el mapa. Finalmente a la capa del heatmap se incorporan los datos del arreglo de comunas.

A2.4 Módulo de visualizaciones Dinámicas

Selector de Relaciones

Para poder enviar las variables separadas por encuestas se realizan 3 consultas. La siguiente figura muestra esto en las líneas 203 a 206. Adicionalmente se aprovecha el campo del formulario de variables para filtrar por encuesta en el template.

```
202  def administrarRelaciones(request):
203      varCASEN = CodigoPregunta.objects.filter(id_encuesta__nombre='CASEN')
204      varNENE = CodigoPregunta.objects.filter(id_encuesta__nombre='NENE')
205      varNESI = CodigoPregunta.objects.filter(id_encuesta__nombre='NESI')
206      encuesta = CodigoPreguntaForm()
207      context = {'varCASEN':varCASEN,'varNENE':varNENE,'varNESI':varNESI,'encuesta':encuesta}
208      return render(request,'relaciones.html',context)
```

Figura 100. Función para obtener los datos de las variables separadas por encuesta en Django

Como muestra la figura 100 se crea un diccionario en la línea 207 donde se ingresan las variables separadas y el formulario. Luego se carga el template y se envía el diccionario de datos.

En el template se carga el campo de id de encuesta del formulario y también se crean formularios html con dos campos por encuesta en donde se ingresan las variables a mostrar y se ocultan mediante la propiedad hidden de html.

```
23
24  $(document).ready(function () {
25
26      $("#formNENE").hide();
27      $("#formNESI").hide();
28      $("#formCASEN").hide();
29      var encuesta;
30      var titulo;
31      var subtitulo;
32      var codigo1;
33      var codigo2;
34
35
36      $("#encuesta").on("change",function(){
37          encuesta = $("#encuesta option:selected").text();
38          if(encuesta == 'NENE'){
39              $("#formNENE").slideDown();
40              $("#formNESI").slideUp();
41              $("#formCASEN").slideUp();
42
43              titulo = $("#varNENE-1 option:selected").text();
44              subtitulo = $("#varNENE-2 option:selected").text();
45              codigo1 = $("#varNENE-1 option:selected").val();
46              codigo2 = $("#varNENE-2 option:selected").val();
47
48              $("#varNENE-1").on("change",function(){
49                  titulo = $("#varNENE-1 option:selected").text();
50                  codigo1 = $("#varNENE-1 option:selected").val();
51              });
52
53              $("#varNENE-2").on("change", function(){
54                  subtitulo = $("#varNENE-2 option:selected").text();
55                  codigo2 = $("#varNENE-2 option:selected").val();
56              });
57
58          }
59          else if(encuesta == 'NESI'){
60              $("#formNENE").slideUp();
61              $("#formNESI").slideDown();
62              $("#formCASEN").slideUp();
63
64              titulo = $("#varNESI-1 option:selected").text();
65              subtitulo = $("#varNESI-2 option:selected").text();
66              codigo1 = $("#varNESI-1 option:selected").val();
67              codigo2 = $("#varNESI-2 option:selected").val();
68
69              $("#varNESI-1").on("change",function(){
70                  titulo = $("#varNESI-1 option:selected").text();
71                  codigo1 = $("#varNESI-1 option:selected").val();
72              });
73
74              $("#varNESI-2").on("change", function(){
75                  subtitulo = $("#varNESI-2 option:selected").text();
76                  codigo2 = $("#varNESI-2 option:selected").val();
77              });
78
79
80          }else if(encuesta == 'CASEN'){
81              $("#formNENE").slideUp();
82              $("#formNESI").slideUp();
83              $("#formCASEN").slideDown();
84
85              titulo = $("#varCASEN-1 option:selected").text();
86              subtitulo = $("#varCASEN-2 option:selected").text();
87              codigo1 = $("#varCASEN-1 option:selected").val();
88              codigo2 = $("#varCASEN-2 option:selected").val();
89
90              $("#varCASEN-1").on("change",function(){
91                  titulo = $("#varCASEN-1 option:selected").text();
92                  codigo1 = $("#varCASEN-1 option:selected").val();
93              });
94
95              $("#varCASEN-2").on("change",function(){
96                  subtitulo = $("#varCASEN-2 option:selected").text();
97                  codigo2 = $("#varCASEN-2 option:selected").val();
98              });
99
100
101         }else{
102
103             $("#formNENE").slideUp();
104             $("#formNESI").slideUp();
105             $("#formCASEN").slideUp();
106
107         }
108     });
109
110     $("#botonGuardar").on("click",function(t){
111         if($("#encuesta").val()!=''){
112             t.preventDefault();
113             agregarTreemap(codigo1,codigo2,titulo,encuesta,subtitulo);
114         }
115     });
116 });
```

Figura 101. Función JavaScript y JQuery para el comportamiento del formulario del selector

Como se puede observar en la figura 101, el comportamiento dinámico del formulario se consigue mediante las funciones de JQuery *slideUp()* para ocultar y *slideDown()* para mostrar. Esto se puede ver desde la línea 39 a la línea 105. Se llama primero al select de html con JQuery y dependiendo de lo que se selecciona se muestra un formulario html y se ocultan los otros. Además se cambian las variables declaradas en las líneas 29 a 33, ya que estas son las variables que serán los argumentos para crear los Treemaps. Si no se selecciona ninguna encuesta se ocultan todos los formularios.En la línea 110 se programa el funcionamiento del botón, si el campo de la encuesta tiene algún valor eso significa que se realizó la selección de variables y se invoca la función agregarTreemap, de lo contrario no hace nada.

Generando un Treemap dinámicamente

Para crear el Treemap es necesario primero limpiar la página de manera dinámica, sin cargar denuevo el html por lo que se utiliza una función para ello creada con JQuery.

```
15  function prepararPagina(array){
16      $("#visualizacion").children().each(function(){
17          $(this).remove();
18      });
19      for(var i in array){
20          $("#visualizacion").append("<div id="+array[i]+"></div>");
21      }
22  }
```

Figura 102. Función JavaScript para preparar página antes de crear el Treemap

Como muestra la figura 102, primero se eliminan las etiquetas html y después se crean nuevas. Se utiliza el arreglo que recibe de parámetro para crear etiquetas div con el id de los elementos del arreglo.

Después de ello se crea el Treemap, esto se logra con la función mostrada en la siguiente figura, desarrollada de tal manera que pueda ser usada como si fuera parte de un API.

```
function CrearTreeMap(elementoJQuery,encuesta,variable,ubicacion,onclick,filtrovariable,valorvariable){//encuesta = st

    var factor;
    ubicacion = "#"+ubicacion;
    if (encuesta == 'NENE') {factor = 'fact'}
    else if(encuesta == 'CASEN'){factor = 'expr'}
    else {factor = 'FACT_HOG'}
    var ruta;
    if(filtrovariable != undefined && valorvariable != undefined){
        ruta = "/prototipo/filtrar/"+encuesta+"/"+variable+"/"+factor+"/"+filtrovariable+"="+valorvariable;
    }
    else{
        ruta = "/prototipo/respuestas/"+encuesta+"/"+variable+"/"+factor
    }
    console.log(ruta);

    $.ajax({
    url : ruta,
    data : elementoJQuery.serialize(),
    dataType: 'json',
    success: function(data){
        parsearEnteros(data);
        if(onclick != undefined){
            new d3plus.Treemap()
                .data(data)
                .height(400)
                .select(ubicacion)
                .groupBy(variable)
                .sum(factor)
                .on("click",onclick)
                .render()
        }else {
            new d3plus.Treemap()
                .data(data)
                .height(400)
                .groupBy(variable)
                .select(ubicacion)
                .sum(factor)
                .render()
        }
    }
    });

}
```

Figura 103. Función JavaScript para crear un Treemap usando AJAX

Como muestra la figura 103, la función recibe 7 parámetros: el elemento que tiene la función, el nombre de la encuesta, la variable a consultar, el elemento html donde se mostrará la visualización, una función onclick para agregar dinamismo al Treemap, una variable de filtro opcional y el valor de la variable de filtro, que también es opcional.

Lo primero que realiza la función es determinar el nombre de la variable del factor de expansión que es diferente en cada encuesta, luego se crea la ruta de Django que realiza la consulta a la base de datos, esta ruta es diferente si se usa una consulta con filtro o sin filtro.

Luego se realiza la petición AJAX para poder consultar asíncronamente a la base de datos. Esto se logra con la función de JQuery $.ajax() que tiene tres parámetros: la ruta los datos a

enviar y una función para ejecutarse después de recibir la respuesta desde Django. En esta función en la línea 21 se crean los Treemaps dependiendo si tiene el argumento onclick o no.

Cuando se ejecuta el Ajax se envía la petición a la ruta y luego espera la respuesta de las funciones de Django para consultar a la base de datos. En la siguiente figura se observa la función de Django que realiza la consulta sin filtro

```
63  def obtenerRespuestaConFactor(request,encuesta,variable,factorExpansion):
64
65      if encuesta == 'CASEN':
66          respuestas = EncuestadoCASEN.objects.only(variable,factorExpansion)
67      elif encuesta == 'NENE':
68          respuestas = EncuestadoNene.objects.only(variable,factorExpansion)
69      elif encuesta == 'NESI':
70          respuestas = EncuestadoNesiConBecas.objects.only(variable,factorExpansion)
71      else :
72          respuestas = None
73
74      opciones = Opcion.objects.filter(id_pregunta__variable=variable).only("nombre_valor","valor")
75
76      if respuestas != None:
77
78          respuestas = [ serializar(respuesta,opciones,variable,factorExpansion) for respuesta in respuestas ]
79          return HttpResponse(json.dumps(respuestas),content_type='application/json')
80      else :
81          return render(request,'index.html')
```

Figura 104. Función para obtener las respuestas de cualquier encuesta.

Como se puede observar en la figura 104, esta función recibe tres parámetros: la encuesta, la variable y el factor de expansión. Lo primero es hacer una estructura *if else* para realizar la consulta a la base de datos con la función de Django *objects.only()* que obtiene todos los registros de la base de datos pero solo la variable a consultar y el factor de expansión. Luego se hace una consulta a la tabla de valores de la variable que se consulta y al final se realiza una serialización utilizando una estructura de control *for in*.

La serialización se realiza con una función de Django y recibe 4 parámetros: una respuesta de un ecuestado, los valores que puede tomar esa respuesta, la variable de la respuesta y el factor de expansión.

```python
def serializar(respuesta,opciones,variable,codigoFactExp):

    atributo = getattr(respuesta,variable)
    factorExpansion = str(getattr(respuesta,codigoFactExp))

    if opciones != None:
        for opcion in opciones:
            if atributo == opcion.valor :
                return {codigoFactExp : factorExpansion,variable : opcion.nombre_valor}
        return  {codigoFactExp : factorExpansion,variable : 'NULO'}
    else:
        return {codigoFactExp:factorExpansion,variable:atributo}
```

Figura 105. Función para serializar de manera personalizada en Django.

La respuesta es un objeto de Python definido en el modelo de encuestados, así que para obtener el valor de la respuesta se utiliza la función *getattr()* lo mismo para el factor de expansión luego se realizan estructuras de control para crear el formato JSON dependiendo de los resultados de las consultas SQL.

Volviendo a la figura 104 después de la serialización se utiliza el método *HttpResponse()* de Django con la función *json.dumps()* se envía la respuesta en formato JSON.

La función para realizar las consultas para treemaps con filtro es la siguiente y consta de dos parámetros más que la función para treemaps sin filtro: la variable del filtro y el valor de la variable.

```python
def obtenerRespuestaConFiltro(request,encuesta,variable,factorExpansion,filtro,valorFiltro):


    valorFtr = Opcion.objects.filter(id_pregunta__variable=filtro,nombre_valor=valorFiltro).only("valor")

    if encuesta == 'CASEN':
        respuestas = EncuestadoCASEN.objects.filter(**{filtro : valorFtr[0].valor}).only(variable,factorExpansion)
    elif encuesta == 'NENE':
        respuestas = EncuestadoNene.objects.filter(**{filtro : valorFtr[0].valor}).only(variable,factorExpansion)
    elif encuesta == 'NESI':
        respuestas = EncuestadoNesiConBecas.objects.filter(**{filtro : valorFtr[0].valor}).only(variable,factorExpansion)
    else :
        respuestas = None

    opciones = Opcion.objects.filter(id_pregunta__variable=variable).only("nombre_valor","valor")

    if respuestas != None:

        respuestas = [ serializar(respuesta,opciones,variable,factorExpansion) for respuesta in respuestas ]
        return HttpResponse(json.dumps(respuestas),content_type='application/json')
    else:
        return render(request,'index.html')
```

Figura 106. Función para obtener las respuestas de cualquier encuesta con un filtro

Como muestra la figura 106, el funcionamiento de esta función es prácticamente igual al otro, solo que la función que realiza la consulta SQL es objects.filter().only() que realiza una consulta con condición WHERE y solo pide la respuesta de la variable y el factor de expansión.

La siguiente figura muestra un ejemplo de cómo se llaman a las funciones para crear los Treemaps en la página de manera dinámica y con dos niveles de información.

```
function agregarTreemap(var1,var2,titulo,encuesta,subtitulo){

    prepararPagina(["titulo","primerTreemap","subtitulo","segundoTreemap","heat"]);
    $("#titulo").html("<h1 class=page-header>"+titulo+"</h1>");
    CrearTreeMap(encuesta,var1,'primerTreemap',function(d){
        var filtro = d[var1];
        $("#segundoTreemap").empty();
        $("#subtitulo").html("<h3 class=page-header>"+subtitulo+"</h3>");
        CrearTreeMap(encuesta,var2,'segundoTreemap',function(r){},var1,filtro);
    });//escribir las funciones treemap
    return false;
}
```

Figura 107. Función JavaScript para visualizar un Treemap Dinámicamente

Como se observa en la figura 107, solo se llaman dos funciones: la función para preparar la página y la función para crear el Treemap. Se puede apreciar que la función *crearTreeMap()* se puede usar recursivamente agregando en el parámetro onclick una función que crea otro Treemap, esta vez agregando un filtro que corresponde a la variable que creó el Treemap anterior.

Anexo 3. Pruebas

A3.1 Pruebas sobre Treemap

Para probar el funcionamiento de las visualizaciones elegidas se ha decidido utilizar unos datos de prueba en una tabla de base de datos llamada Persona, con atributos nombre, apellido, sexo y provincia. Se generará un Treemap ordenado por apellido, y cuando se presione click sobre un apellido se mostrará otro en donde se muestra cual porcentaje de los apellidos pertenece al sexo masculino y cual al sexo femenino. La figura 108 muestra la tabla que se usó para ello.

Id	Rut	Nombre	apellido	provincia	sexo
1	18.070.334-9	Rodrigo	Hernandez	Biobío	Masculino
2	11.497.476-5	Magdalena	Martinez	Biobío	Femenino
3	17.844.426-3	Angelo	Hernandez	Arauco	Masculino
4	9.889.503-5	Erika	Acosta	Concepción	Femenino
5	12.043.018-1	Washington	Hernandez	Ñuble	Masculino
6	12.356.897-9	Jimena	Hernandez	Biobío	Femenino
7	7.235.458-6	Santiago	Martinez	Ñuble	Masculino

Figura 108. Tabla de datos de Prueba

A simple vista podemos observar que el apellido que más se repite es Hernandez, y el que menos se repite es Acosta, de tal forma que si se usa un Treemap debería mostrar un recuadro más grande para Hernández. Esto se puede observar en la figura 109 que es el resultado del uso del Treemap de la librería d3plus.

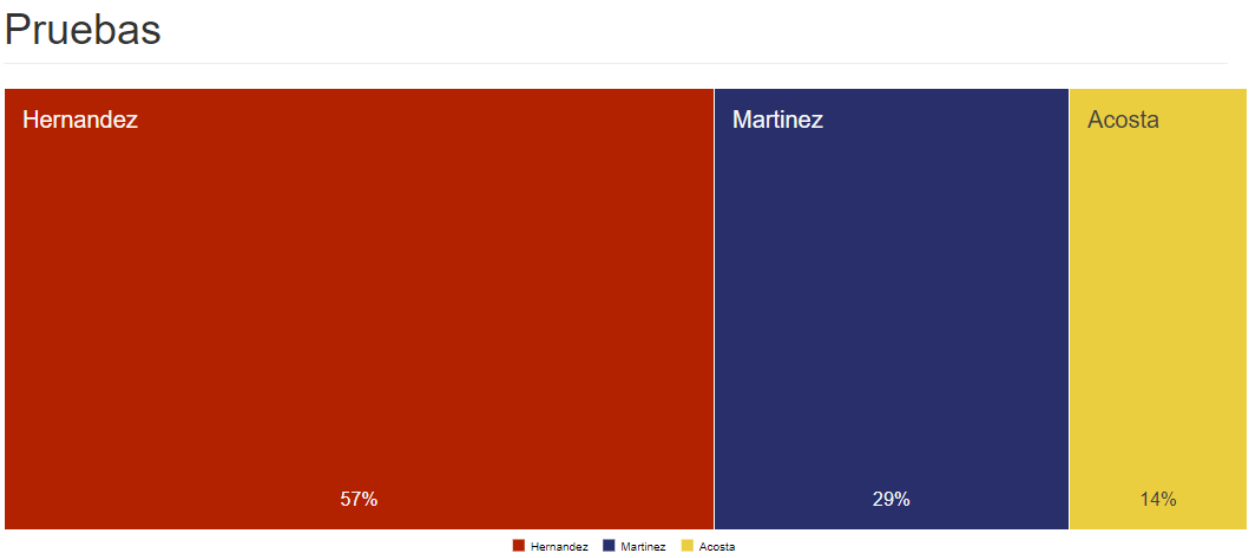

Figura 109. Treemap de la tabla de la figura 108 ordenado por apellidos

Se observa que el Treemap está ordenado por la categoría del apellido que va desde el que tiene mayor porcentaje al menor, además se diferencian los apellidos por colores para hacerlo más entendible a la vista. Si se presiona el click sobre alguno de los recuadros del Treemap este interactuará creando otro Treemap que muestre el porcentaje de personas del sexo masculino o femenino dependiendo del apellido. La figura 110 muestra un ejemplo.

Figura 110. Ejemplo de interacción del Treemap de la figura 109

Este ejemplo hace referencia al apellido Martinez. Como se puede observar en la tabla de la figura 103 el apellido Martinez lo tienen dos personas, una es masculina y la otra femenina, por lo tanto el Treemap debería mostrar un recuadro dividido a la mitad, mostrando un 50% masculino y un 50% femenino. La figura 110 muestra el resultado esperado, con un rectángulo dividido a la mitad y haciendo referencia al apellido a través del color.

A3.2 Pruebas sobre Grafo

Como el grafo se utiliza solamente para relaciones binarias es que usaremos un arreglo de nodos y un arreglo de arcos para la prueba. Los nodos son 6, uno para Masculino, uno para femenino, y cuatro personas: dos masculinas y dos femeninas. La figura 111 muestra el resultado de la visualización del grafo.

Figura 111. Prueba de visualización del grafo

Como se puede observar, d3plus genera un conjunto de nodos donde previamente se indicaron sus posiciones, cada uno con un color y mostrando los arcos tenuemente, casi imperceptible al ojo humano. Tiene en el costado un set de herramientas de animación por defecto, permitiendo agregar o quitar zoom, volver atrás y la opción de ver completo. Al colocar el cursor del mouse por encima de los nodos es posible que la visualización interactúe, mostrando información adicional, además al presionarlos debería mostrar la conexión que

corresponde al sexo de cada persona. En la figura 107 se mostrará un ejemplo de cómo interactúa el grafo.

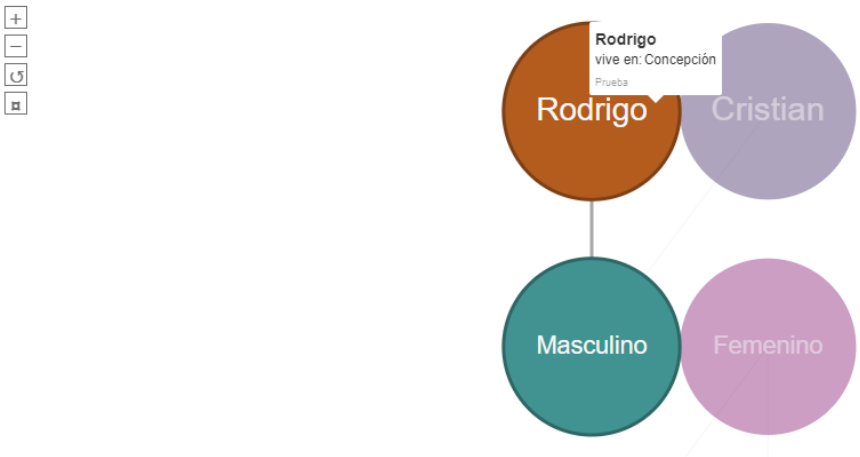

Figura 112. Ejemplo de interacción con el grafo de la figura 56

Se puede observar en la figura 112 que el grafo interactúa como se esperaba, muestra un cuadro con información adicional y además resalta la relación que existe entre el nodo Rodrigo con el nodo Masculino, haciendo que los otros nodos se vean más opacos, además muestra el arco que los une y se genera una animación que hace que los nodos tomen el protagonismo, incluso sacando a los nodos que estaban abajo. Con las herramientas de la esquina superior izquierda se puede volver atrás o quitar el zoom para volver a ver el grafo como en la figura 111.