

DOCUMENTACIÓN CUBOS DE DATOS

Descripción de limpieza y transformaciones
de datos

Versión 6

(Julio 2022)



TABLA DE CONTENIDO

Presentación	1
1. Censo Económico	2
1.1. inegi_economic_census	2
1.2. Inegi_economic_census_additional	5
1.3. Inegi_economic_census_sex	10
1.4. Inegi_economic_census_2014_ent	13
1.5 inegi_economic_census_2014_mun	16
1.6. Indicators_economic_census	19
2. Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH)	35
2.1. Inegi_enigh_expense_items	35
2.2. Inegi_enigh_income_source	38
2.3. inegi_enigh_income	41
3. Empleo	43
3.1. Inegi_enoe	43
3.2. inegi_etoe	48
4. Salud	53
4.1. Health_establishments	53
4.2. Health_resources	56
5. Propiedad Industrial	59
5.1. impi_companies_level	59
5.2. Impi_country_level	62
5.3. impi_state_level	65
6. Comercio Exterior	68
6.1. economy_foreign_trade_nat/state/mun	68
6.2. banxico_trade_flow	71
6.3. inegi_foreign_trade_country	74

6.4. inegi_foreign_trade_product	77
6.5. inegi_foreign_trade_state	80
6.6. trade_i_baci_a_12	83
7. Inversión Extranjera Directa (IED)	86
7.1. fdi_10_year_country/country_investment/investment	86
7.2. fdi_2_state_investment	91
7.3. Fdi_3_country_origin	94
7.4. fdi_4_investment_type	97
7.5. fdi_9_quarter/quarter_investment/_year/_year_investment	100
7.6. fdi_quarter_* y fdi_year_*	106
8. Asociación Nacional de Universidades e Instituciones de Educación Superior (ANUIES)	116
8.1. Anuies_enrollment	116
8.2. Anuies_origin	119
8.3. Anuies_status	122
9. Población y Vivienda	125
9.1. Conapo_metro_area_population	125
9.2. Inegi_population_total	127
9.3. Inegi_population	130
9.4. Population_basic_quest_by_age	134
9.5. Population_projection	136
9.6. Inegi_housing	138
9.7. Inegi_housing_basic	142
9.8. inegi_housing_ranges_basic	144
10. Consejo Nacional de Evaluación de Política de Desarrollo Social (CONEVAL)	147
10.1. Coneval_gini_nat	147
10.2. Coneval_gini_ent	150
10.3. Coneval_gini_mun	152

10.4. Coneval_poverty	154
11. Seguridad Pública	156
11.1. Inegi_envipe	156
11.2. Sensnsp_crimes	159
12. Economía	161
12.1. Inegi_denue	161
12.2. Inegi_gdp	164
12.3. Indicators_i_wdi_a	167
13. Infonavit	171
13.1. Infonavit_age_range_credits	171
13.2. Infonavit_credit_line	173
13.3. Entity_credits	175
13.4. house_credits	177
13.5. income_level_credits	179
13.6. payment_entity_credits	181
13.7. product_credits	183
14. Gasto Público	185
14.1. budget_transparency_annual_pipeline	185
15. Parques Industriales	188
15.1. industrial_parks	188
16. Remesas	191
16.1. banxico_countries_remittances	191
16.2. banxico_income_remittances	194
16.3. banxico_mun_income_remittances	197
17. Complejidad Económica	200
17.1. complexity_eci	200
17.2. complexity_pci	206

17.3. complexity_eci_a_hs12_hs6	211
18. COVID-19	213
18.1. Gobmx_covid	213
18.2. Gobmx_covid_stats_nation	216
18.3. Gobmx_covid_stats_state	219
18.4. Gobmx_covid_stats_metroarea	222
18.5. Gobmx_covid_stats_mun	226
19. Precios de productos	229
19.1. sniim	229

Presentación

DataMéxico (www.datamexico.org) es una plataforma digital a cargo de la Secretaría de Economía que permite la integración, visualización y análisis de datos para mejorar la toma de decisiones de políticas públicas enfocadas en la innovación, inclusión y diversificación de la economía mexicana.

En este contexto, la plataforma incluye una gran cantidad de datos, los cuales, antes de ser incluidos en el sitio, deben pasar por un proceso de transformación y limpieza para obtener conjuntos de datos ordenados y que permitan ser integrados en las diferentes visualizaciones presentadas en el sitio.

Con el objetivo de profundizar en el procesamiento de los datos contenidos en DataMéxico, este documento describe las transformaciones realizadas sobre los diferentes conjuntos de datos en cada pipeline de ETL, detallando el proceso desde el formato original del archivo de datos hasta su forma final.

Adicionalmente, al finalizar el documento se encuentra el anexo “Proceso de Backend” que entrega detalles del desarrollo y ejecución de pipelines y esquemas, correspondientes a los pasos previos a la ingestión de datos en la API Interactiva de DataMéxico (<https://api.datamexico.org/ui>).

1. Censo Económico

A continuación se detalla el proceso de ETL para datos del Censo Económico en diferentes años. Debido a temas de anonimización se trabajaron los datos en cubos diferentes, existiendo pipelines que procesan datos para todos los años disponibles, otros pipelines solo para datos del 2014 y otros solo para el 2019. Cabe mencionar que los cubos que tienen desagregación por industrias, utilizan el Sistema de Clasificación Industrial de América del Norte, SCIAN 2018.

1.1. `inegi_economic_census`

Descripción general y ejecución

El pipeline del censo económico (`pipeline_economic_census`) es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos del censo económico entregados por INEGI (Instituto Nacional de Estadística y Geografía). Estos datos incluyen cantidades asociadas a las actividades económicas, tales como; unidades económicas (UE), personal, inversión total, entre otros. Los datos son descargados desde [el repositorio de datos de INEGI](#) y almacenados en un compartimiento de Google Storage privado, pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos Clickhouse siguiendo un modelo relacional.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de dos formas:

```
(Python)          ~/data-etl/etl/inergi_economic_census/$ python
pipeline_economic_census.py
```

```
(bamboo-cli)  ~/data-etl/etl/inergi_economic_census/$ bamboo-cli --folder .
--entry python pipeline_economic_census
```

Descarga de datos

Una vez obtenidos los datos desde el sitio de INEGI y almacenados en un compartimiento privado de GCP storage, se establece una conexión privada validada con credenciales fuertemente encriptadas, para ejecutar una descarga segura hasta el servidor de procesamiento de los datos.

Lectura de datos

Una vez descargados los datos, estos se leen y almacenan en un *DataFrame* de la librería pandas. Al momento en el que se elaboró esta documentación, el *DataFrame* inicial contenía 437.990 filas y 192 columnas. La siguiente figura muestra las primeras filas y algunas columnas del *DataFrame* inicial obtenido.

Año Censal	Entidad	Municipio	Actividad Económica	UE Unidades económicas	...	P030C Variaci�n de inventarios de productos en proceso (millones de pesos) ¹	Q000B Depreciac�n total de activos fijos (millones de pesos) ¹	Q010A Acervo total de maquinaria y equipo de producci�n (millones de pesos)	Q400A Acervo total de equipo de c�mputo y perif��ricos (millones de pesos)
437985	2004.0	08 Chihuahua	045 Meoqui	Elaboraci�n de helados y paletas	311520	5.0 ...	NaN	NaN	NaN
437986	2004.0	08 Chihuahua	048 Namiquipa	3115CC Clases agrupadas por el principio de c...	3.0 ...	NaN	NaN	NaN	NaN
437987	2004.0	08 Chihuahua	050 Nuevo Casas Grandes	3115CC Clases agrupadas por el principio de c...	11.0 ...	NaN	NaN	NaN	NaN

Figura 1. Formato original datos del Censo Econ mico

Transformaci n y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformaci n y limpieza para poder obtenerlos en el formato deseado. Primero, se eliminan del conjunto de datos las filas y columnas que corresponden a valores totales y las que

se encuentran vacías. Cabe destacar que los valores totales no son necesarios porque pueden ser calculados agregando los valores individuales. Luego, se renombran las columnas que tienen código de medida, dejando sólo el código de medida como nombre, por ejemplo; la columna "Q000A Acervo total de activos fijos (millones de pesos)" pasa a ser "Q000A".

Posteriormente, se crean números identificadores (ID) para cada entidad federativa y municipio y se extrae el ID de cada actividad económica para luego incluirlos en el *DataFrame* como un identificador de cada uno de ellos, además, se guardan en tablas diferentes en la base de datos cada columna ID generada junto con su columna original, por ejemplo, *Municipio* con *mun_id* irán en una tabla. Finalmente, se eliminan del *DataFrame* las columnas duplicadas y las columnas que ya no son necesarias, por ejemplo; *Municipio* ya no es necesaria, porque ahora existe *mun_id* que corresponde al ID de cada municipio, el cual está adecuadamente indexado y guardado en otra tabla de dimensiones de la base de datos.

Al momento en el que se elaboró esta documentación, el *DataFrame* final contenía 426.563 filas y 101 columnas. La estructura final de la tabla se presenta a continuación:

	year	mun_id	national_industry_id	ue	h001a	h000a	h010a	...	p030a	p030b	q010a	q020a	q030a	q400a	q900a
426558	2004	8045	311520	5.0	13	13.0	1	...	0.000	0.000	0.126	1.080	0.060	0.000	0.157
426559	2004	8048	3115CC	3.0	89	83.0	80	...	0.006	0.003	7.231	2.503	2.654	0.397	0.356
426560	2004	8050	3115CC	11.0	28	28.0	6	...	0.006	0.000	0.847	1.200	0.285	0.000	0.101
426561	2004	8052	311520	5.0	29	28.0	20	...	0.000	0.000	0.515	2.750	0.200	0.020	0.042
426562	2004	8060	311520	3.0	9	9.0	0	...	0.000	0.000	0.110	0.080	0.000	0.000	0.026

Figura 2. Formato final tabla de datos del Censo Económico

1.2. Inegi_economic_census_additional

Descripción general y ejecución

El pipeline `economic_census_2019_additional.py` procesa los datos facilitados por el Instituto Nacional de Estadística y Geografía (INEGI) que toman relación con el Censo Económico de 2019. En específico, contiene información adicional sobre indicadores a nivel de entidad y municipio de distintas actividades económicas. Los datos son descargados desde el sitio del [Instituto Nacional de Estadística y Geografía](#), y son almacenados en Google Storage para ejecutar su posterior procesamiento.

El procesamiento consiste en un proceso de limpieza y transformación, donde luego son ingestados en una base de datos de Clickhouse siguiendo un modelo relacional.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de dos formas:

(Python)	<code>~/data-etl/etl/inegi_economic_census/\$</code>	<code>python</code>
	<code>economic_census_2019_additional.py</code>	
(bamboo-cli)	<code>~/data-etl/etl/inegi_economic_census/\$</code>	<code>bamboo-cli --folder .</code>
	<code>--entry economic_census_2019_additional</code>	

Descarga de datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv, y son 32 los archivos que se descargan para obtener la totalidad de los datos.

Una vez descargados los archivos a la carpeta temporal, mediante un *for loop*, se procede a leer con la función *read_csv* de Pandas cada uno de los archivos. Posterior a haber renombrado las columnas, y eliminada la variable *ENTIDAD*, se concatena cada archivo a un *DataFrame* llamado *t*.

Luego de haber concatenado los 32 archivos, se crea el *DataFrame* *df* como una copia de *t*.

Finalmente, se crea la columna *year*, la cual almacena el periodo en estudio. En este caso, corresponde a 2019.

Transformación - Entidades por Sector

Esta etapa procede como sigue:

- Se crea una copia del conjunto de datos total en un *DataFrame* llamado *base*.
- Selección de filas donde la variable *CODIGO* tenga largo igual a 2, sea igual a 31-33, o sea igual a 48-49.
- Selección de datos donde las variables *ID_ESTRATO*, *MUNICIPIO*, y *CODIGO*, no posean valores *Nan*.
- Creación de la variable *sector_id*, al eliminar espacios vacíos de la variable *CODIGO*.
- Creación de la variable *ent_id*, cuyos valores corresponden a los de la variable *ENTIDAD* con *dtype Integer*.
- Se eliminan variables que no serán utilizadas: *ID_ESTRATO*, *CODIGO*, *MUNICIPIO*, y *ENTIDAD*.

- Se crean variables que no existan en el conjunto de datos con el valor 0. Entre las variables que deben existir, se encuentran: *ent_id*, *mun_id*, *sector_id*, *subsector_id*, y *rama_id*.
- Creación de la variable *level*. Se asigna el valor único de 1.

Posterior a ello, se crea una copia del *DataFrame* procesado, llamándolo *df_ent_sec*. Dicho *DataFrame* junto al *DataFrame* base, continúan a la siguiente etapa de transformación.

Transformación - Entidades por Sub Sector

En esta etapa se ejecutan las siguientes acciones:

- Se crea un nuevo *DataFrame* llamado *df*. Dicho *df* es simplemente una copia de *base*.
- Selección de filas donde la variable *CODIGO* tenga longitud igual a 3.
- Selección de datos donde las variables *ID_ESTRATO*, *MUNICIPIO*, y *CODIGO*, no posean valores *Nan*.
- Creación de la variable *subsector_id*, cuyos valores corresponden a los de la variable *CODIGO* con *dtype Integer*.
- Creación de la variable *ent_id*, cuyos valores corresponden a los de la variable *ENTIDAD* con *dtype Integer*.
- Se eliminan variables que no serán utilizadas: *ID_ESTRATO*, *CODIGO*, *MUNICIPIO* y *ENTIDAD*.
- Se crean variables que no existan en el conjunto de datos con el valor 0. Entre las variables que deben existir, se encuentran: *ent_id*, *mun_id*, *sector_id*, *subsector_id*, y *rama_id*.
- Creación de la variable *level*. Se asigna el valor único de 2.

Posterior a ello, se crea una copia del *DataFrame* procesado, llamándolo *df_ent_sub*. Dicho *DataFrame* junto al *DataFrame* base y *df_ent_sec*, continúan a la siguiente etapa de transformación.

Transformación - Entidades por Rama

Esta etapa procede como sigue:

- Se crea un nuevo *DataFrame* llamado df. Dicho *DataFrame* es simplemente una copia de *base*.
- Selección de filas donde la variable *CODIGO* tenga longitud igual a 4.
- Selección de datos donde las variables *ID_ESTRATO*, *MUNICIPIO* y *CODIGO*, no posean valores *NaN*.
- Creación de la variable *rama_id*, cuyos valores corresponden a los de la variable *CODIGO* con *dtype Integer*.
- Creación de la variable *ent_id*, cuyos valores corresponden a los de la variable *ENTIDAD* con *dtype Integer*.
- Se eliminan variables que no serán utilizadas: *ID_ESTRATO*, *CODIGO*, *MUNICIPIO* y *ENTIDAD*.
- Se crean variables que no existan en el conjunto de datos con el valor 0. Entre las variables que deben existir, se encuentran: *ent_id*, *mun_id*, *sector_id*, *subsector_id* y *rama_id*.
- Creación de la variable *level*. Se asigna el valor único de 3.

Posterior a ello, se crea una copia del *DataFrame* procesado, llamándolo *df_ent_ram*. Dicho *DataFrame* junto al *DataFrame* base, *df_ent_sec*, y *df_ent_sub*, continúan a la siguiente etapa de transformación.

Transformación - Municipio por Sector

En esta etapa se ejecutan las siguientes acciones:

- Se crea un nuevo *DataFrame* llamado df. Dicho df es simplemente una copia de *base*.
- Selección de filas donde la variable *CODIGO* tenga largo igual a 2, sea igual a 31-33, o sea igual a 48-49.
- Selección de datos donde las variables *ID_ESTRATO*, *MUNICIPIO* y *CODIGO*, no posean valores *NaN*.

- Creación de la variable *sector_id*, al eliminar espacios vacíos de la variable *CODIGO*.
- Creación de la variable *mun_id*, cuyos valores corresponden a la agregación de las variables *ENTIDAD* y *MUNICIPIO*, transformados a *dtype Integer*.
- Se eliminan variables que no serán utilizadas: *ID_ESTRATO*, *CODIGO*, *MUNICIPIO* y *ENTIDAD*.
- Se crean variables que no existan en el conjunto de datos con el valor 0. Entre las variables que deben existir, se encuentran: *ent_id*, *mun_id*, *sector_id*, *subsector_id* y *rama_id*.
- Creación de la variable *level*. Se asigna el valor único de 4.

Posterior a ello, se crea una copia del *DataFrame* procesado, llamándolo *df_mun_sec*. Dicho *DataFrame* junto al *DataFrame* base, *df_ent_sec*, *df_ent_sub*, y *df_ent_ram*, continúan a la siguiente etapa de transformación.

Concatenación

En esta etapa, ingresan los conjuntos de datos: *base*, *df_ent_sec*, *df_ent_sub*, *df_ent_ram* y *df_mun_sec*. Dado su estructura similar, la idea es agrupar todos los conjuntos de datos en uno solo. Dado ello, se llevan a cabo las siguientes etapas:

- Mediante un *for loop* y la función *append* de Pandas, se concatenan todos los conjuntos de datos en uno llamado *df*.
- Transformación de *dtypes* a conveniencia.
- Creación de la variable *year*. Dicha variable almacena el periodo en estudio.
- Transformación de nombres de columnas a minúsculas. Se utiliza la función *str.lower()* de Pandas.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 42.507 filas y 189 columnas. A continuación puede ser visualizada su estructura de salida:

	ue	a111a	a121a	a131a	a211a	...	ent_id	mun_id	subsector_id	rama_id	level
5	37	188.191	150.755	37.436	-2.869	...	1.0	0.0	0.0	0.0	1
39	19	8651.804	3349.414	5302.390	322.953	...	1.0	0.0	0.0	0.0	1
78	13	NaN	NaN	NaN	NaN	...	1.0	0.0	0.0	0.0	1
96	408	7843.008	5432.334	2410.674	71.039	...	1.0	0.0	0.0	0.0	1
244	5588	326901.167	228336.310	98564.857	5802.321	...	1.0	0.0	0.0	0.0	1

Figura 3. Formato final tabla de datos del Censo Económico 2019

1.3. Inegi_economic_census_sex

Descripción general y ejecución

El pipeline del censo económico por sexo (`economic_census_sex_pipeline`) es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos del censo económico entregados por INEGI (Instituto Nacional de Estadística y Geografía) desagregando los datos según el sexo de los trabajadores. Incluye características asociadas al personal en las diferentes industrias.

Los datos son descargados desde [el repositorio de datos de INEGI](#) y almacenados en un compartimiento de Google Storage privado, pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos *Clickhouse* siguiendo un modelo relacional.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales la librería pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de dos formas:

```
(Python)           ~/data-etl/etl/inegi_economic_census/$ python
                  economic_census_sex_pipeline.py
```

```
(bamboo-cli)   ~/data-etl/etl/inegi_economic_census/$ bamboo-cli --folder .
--entry python economic_census_sex_pipeline
```

Descarga de datos

Una vez obtenidos los datos desde el sitio de INEGI y almacenados en un compartimiento privado de GCP storage, se establece una conexión privada validada

con credenciales fuertemente encriptadas, para ejecutar una descarga segura hasta el servidor de procesamiento de los datos.

Lectura de datos

Una vez descargados los datos, estos se leen y almacenan en un *DataFrame* de la librería pandas. Al momento en el que se elaboró esta documentación, el *DataFrame* inicial contenía 437.990 filas y 192 columnas. La siguiente figura muestra las primeras filas y algunas columnas del *DataFrame* inicial obtenido.

	Año Censal	Entidad	Municipio	Actividad Económica	UE Unidades económicas	...	P030C Variaci�n de inventarios de productos en proceso (millones de pesos) ¹	Q000B Depreciac�n total de activos fijos (millones de pesos) ¹	Q010A Acervo total de maquinaria y equipo de producci�n (millones de pesos)	Q400A Acervo total de equipo de c�mputo y perif��ricos (millones de pesos)
437985	2004.0	08 Chihuahua	045 Meoqui	311520 Elaboraci�n de helados y paletas	5.0	...	NaN	NaN	NaN	NaN
437986	2004.0	08 Chihuahua	048 Namiquipa	3115CC Clases agrupadas por el principio de c...	3.0	...	NaN	NaN	NaN	NaN
437987	2004.0	08 Chihuahua	050 Nuevo Casas Grandes	3115CC Clases agrupadas por el principio de c...	11.0	...	NaN	NaN	NaN	NaN

Figura 4. Formato original datos del Censo Econ mico por sexo

Transformaci n y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformaci n y limpieza para poder obtenerlos en el formato deseado. Primero, se eliminan del *DataFrame* las filas y columnas que no ser n utilizadas, estas corresponden a valores agregados, valores totales, filas vac as y filas y columnas repetidas. Cabe destacar que los valores totales no son necesarios porque pueden ser calculados agregando los valores individuales.

Posteriormente, se crean n meros identificadores (ID) para cada entidad y municipio y se extrae el ID de cada actividad econ mica para luego incluirlos en el *DataFrame* como un identificador de cada uno de ellos, adem s, se guardan en tablas diferentes en la base de datos cada columna ID generada junto con su columna original, por ejemplo, Municipio con *mun_id* ir n en una tabla.

Luego, se divide el *DataFrame* en dos, uno para cada sexo, a cada uno de ellos se les renombran sus columnas, pasando de códigos a nombres más específicos, por ejemplo; “*H101B: production_personnel_sales_and_service*”, pasa a ser “*production_personnel_sales_and_service*”. Además a cada *DataFrame* se le agrega una columna identificadora del sexo, donde hombres corresponde a 1 y mujeres a 2. Finalmente se concatenan nuevamente los *DataFrame*, esta vez ordenados por sexo.

Finalmente, se obtienen agregaciones de los datos agrupando el *DataFrame* por año, municipio, sexo e industria.

Al momento en el que se elaboró esta documentación, el *DataFrame* final contenía 853.126 filas y 13 columnas. La estructura final de la tabla se presenta a continuación:

year	mun_id	national_industry_id	sex	employed_workers	...	administrative_accounting_and_managerial_staff	owners_family_and_other_unpaid_workers
2004	1001	114119	1	18	...	0	7
2004	1001	114119	2	0	...	0	0
2004	1001	21CCCC	1	394	...	10	1
2004	1001	21CCCC	2	7	...	7	0
2004	1001	23611C	1	5332	...	301	32
...
2014	32058	8114CC	2	6	...	0	3
2014	32058	81CCCC	1	1	...	0	1
2014	32058	81CCCC	2	7	...	0	5
2014	32058	SCCCCC	1	6	...	0	1
2014	32058	SCCCCC	2	2	...	1	1

Figura 5. Formato original datos del Censo Económico por sexo

1.4. Inegi_economic_census_2014_ent

Descripción general y ejecución

El pipeline del censo económico 2014 por entidad federativa (*economic_census_2014_ent*) es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos del censo económico 2014 a nivel estatal entregados por INEGI (Instituto Nacional de Estadística y Geografía). Estos datos incluyen métricas asociadas a las actividades económicas por entidad federativa, tales como unidades económicas, producción bruta total, depreciación total de activos fijos, salarios de los empleados, entre otros.

Los datos son descargados desde [el repositorio de datos de INEGI](#) y almacenados en un compartimiento de Google Storage privado, pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos Clickhouse siguiendo un modelo relacional.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Al existir 32 entidades federativas, este pipeline realiza el proceso de ETL de manera iterativa sobre cada una de ellas. En otras palabras, los mismos cuatro pasos del pipeline se ejecutan para cada estado.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de dos formas:

(Python)	<code>~/data-etl/etl/inegi_economic_census/\$</code>	<code>python</code>
	<code>economic_census_2014_ent.py</code>	

```
(bamboo-cli) ~/data-etl/etl/inergi_economic_census/$ bamboo-cli --folder .
--entry economic_census_2014_ent
```

Descarga de datos

Una vez obtenidos los datos desde el sitio de INEGI y almacenados en un compartimiento privado de GCP storage, se establece una conexión privada validada con credenciales fuertemente encriptadas, para ejecutar una descarga segura hasta el servidor de procesamiento de los datos.

Cabe destacar que existe un archivo .csv por cada entidad federativa, por lo que al terminar de iterar sobre todas ellas, se habrán descargado un total de 32 archivos.

Lectura de datos

Una vez descargados los datos para la entidad federativa de turno, estos se leen y almacenan en un *DataFrame* de la librería pandas. A modo de ejemplo, para la entidad federativa 28 (Tamaulipas), al momento en el que se elaboró esta documentación, el *DataFrame* inicial tenía 36.667 filas y 105 columnas. La siguiente imagen muestra las primeras filas y algunas columnas del *DataFrame* inicial obtenido.

	clave_entidad	entidad_federativa	clave_municipio	municipio	clave_actividad_economica	...	Q010A	Q020A	Q030A	Q400A	Q900A
36662	28	Tamaulipas	43.0	Xicoténcatl	SCCC	...	99.279	2.8	0.938	0.168	0.37
36663	28	Tamaulipas	43.0	Xicoténcatl	SCCCC	...	99.279	2.8	0.938	0.168	0.37
36664	28	Tamaulipas	43.0	Xicoténcatl	SCCCC	...	99.279	2.8	0.938	0.168	0.37
36665	28	Tamaulipas	43.0	Xicoténcatl	SCCCCC	...	99.279	2.8	0.938	0.168	0.37
36666	28	Tamaulipas	43.0	Xicoténcatl	SCCCCC	...	99.279	2.8	0.938	0.168	0.37

Figura 6. Formato original datos del Censo Económico 2014

Transformación y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Primero, se filtran y reemplazan todos los valores nulos, dejando solo los valores no nulos dentro del *DataFrame*, además, se filtran las claves de actividad económica dejando solo las que contienen 6 dígitos.

Finalmente, se eliminan del *DataFrame* todas las columnas que ya no son necesarias, dado que mantenerlas solo genera redundancia en los datos. También se agrega una última columna correspondiente al año de realización del censo, que en este caso corresponde a 2014.

Al momento en el que se elaboró esta documentación, para la entidad federativa 28 (Tamaulipas), el *DataFrame* final contenía 1.960 filas y 101 columnas. La estructura final de la tabla se presenta a continuación:

	ent_id	national_industry_id	ue	a111a	a121a	a131a	a211a	...	q000d	q010a	q020a	q030a	q400a	q900a	year
4605	28	813230	25	6.978	4.372	2.606	0.633	...	0.0	0.666	2.649	1.581	0.312	0.672	2014
4606	28	813230	26	19.140	5.646	13.494	0.033	...	0.0	0.076	3.406	0.635	0.586	1.485	2014
4607	28	813230	105	39.247	17.349	21.898	1.245	...	0.0	9.581	83.068	1.994	0.503	1.389	2014
4616	28	SCCCCC	24	33615.827	2941.778	30674.049	3096.148	...	0.0	19049.194	3738.023	158.497	0.784	294.842	2014
4617	28	SCCCCC	24	33615.827	2941.778	30674.049	3096.148	...	0.0	19049.194	3738.023	158.497	0.784	294.842	2014

Figura 7. Formato final tabla de datos del Censo Económico 2014

1.5 inegi_economic_census_2014_mun

Descripción general y ejecución

El pipeline del censo económico 2014 por municipio (`economic_census_2014_mun`) es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos del censo económico 2014 a nivel municipal entregados por INEGI (Instituto Nacional de Estadística y Geografía). Estos datos incluyen métricas asociadas a las actividades económicas por municipio, tales como unidades económicas, producción bruta total, depreciación total de activos fijos, salarios de los empleados, entre otros.

Los datos son descargados desde [el repositorio de datos de INEGI](#) y almacenados en un compartimiento de Google Storage privado, pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos Clickhouse siguiendo un modelo relacional.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Este pipeline es casi idéntico al pipeline del censo económico 2014 por entidad federativa, la diferencia es que el presente pipeline procesa los datos a nivel municipal.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de dos formas:

(Python)

```
~/data-etl/etl/inegi_economic_census/$ python economic_census_2014_mun.py
```

(bamboo-cli)

```
~/data-etl/etl/inegi_economic_census/$ bamboo-cli --folder . --entry
economic_census_2014_mun
```

Al existir 32 entidades federativas, este pipeline realiza el proceso de ETL de manera iterativa sobre cada una de ellas. En otras palabras, los mismos cuatro pasos del pipeline se ejecutan para cada estado. En este pipeline en específico, se desagregan los datos por cada municipio.

La razón para tener un pipeline para nivel estatal y otro a nivel municipal radica en la anonimización de los datos, donde el acumulado de los municipios no entrega el valor correcto a nivel estatal.

Descarga de datos

Una vez obtenidos los datos desde el sitio de INEGI y almacenados en un compartimiento privado de GCP storage, se establece una conexión privada validada con credenciales fuertemente encriptadas, para ejecutar una descarga segura hasta el servidor de procesamiento de los datos.

Cabe destacar que existe un archivo .csv por cada entidad federativa, por lo que al terminar de iterar sobre todas ellas, se habrán descargado un total de 32 archivos.

Lectura de datos

Una vez descargados los datos para la entidad federativa de turno, estos se leen y almacenan en un *DataFrame* de la librería pandas. A modo de ejemplo, para la entidad 28 (Tamaulipas), al momento en el que se elaboró esta documentación, el *DataFrame* inicial tenía 36.667 filas y 105 columnas. La siguiente imagen muestra las primeras filas y algunas columnas del *DataFrame* inicial obtenido.

	clave_entidad	entidad_federativa	clave_municipio	municipio	clave_actividad_economica	...	Q010A	Q020A	Q030A	Q400A	Q900A	
36662	28	Tamaulipas	43.0	Xicoténcatl		SCCC	...	99.279	2.8	0.938	0.168	0.37
36663	28	Tamaulipas	43.0	Xicoténcatl		SCCCC	...	99.279	2.8	0.938	0.168	0.37
36664	28	Tamaulipas	43.0	Xicoténcatl		SCCCC	...	99.279	2.8	0.938	0.168	0.37
36665	28	Tamaulipas	43.0	Xicoténcatl		SCCCCC	...	99.279	2.8	0.938	0.168	0.37
36666	28	Tamaulipas	43.0	Xicoténcatl		SCCCCC	...	99.279	2.8	0.938	0.168	0.37

Figura 8. Formato original datos del Censo Económico 2014 a nivel municipal

Transformación y limpieza

Luego de la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Primero, se filtran y reemplazan todos los valores nulos, dejando solo los valores no nulos dentro del *DataFrame*, además, se filtran las claves de actividad económica dejando solo las que contienen 6 dígitos.

Posterior al filtrado, se genera un número identificador (ID) para cada municipio, el cual será el identificador del municipio dentro del cubo final. De forma paralela se guardará el ID del municipio junto al nombre del municipio en una tabla diferente en la base de datos.

Finalmente, se eliminan del *DataFrame* todas las columnas que ya no son necesarias, dado que mantenerlas solo genera redundancia en los datos. También se agrega una última columna correspondiente al año de realización del censo, que en este caso corresponde a 2014.

Al momento en el que se elaboró esta documentación, para la entidad federativa 28 (Tamaulipas), el *DataFrame* final contenía 7.124 filas y 101 columnas. La estructura final de la tabla se presenta a continuación:

	national_industry_id	ue	a111a	a121a	a131a	a211a	a221a	...	q010a	q020a	q030a	q400a	q900a	mun_id	year	
36636		812110	7	0.546	0.226	0.320	0.002	0.002	...	0.152	0.295	0.045	0.000	0.168	28043	2014
36642		812CCC	3	0.474	0.262	0.212	0.017	0.017	...	0.414	0.800	0.030	0.000	0.049	28043	2014
36650		8131CC	6	20.968	9.665	11.303	1.190	1.190	...	27.000	1.890	5.165	0.226	0.049	28043	2014
36656		813230	3	0.013	0.018	-0.005	0.000	0.000	...	0.000	0.070	0.000	0.000	0.011	28043	2014
36666		SCCCCC	7	128.471	63.474	64.997	20.971	20.569	...	99.279	2.800	0.938	0.168	0.370	28043	2014

Figura 9. Formato final tabla de datos del Censo Económico 2014 a nivel municipal

1.6. Indicators_economic_census

Descripción general y ejecución

El pipeline *indicators_economic_census_2019.py* es un compilado de diferentes indicadores de la industria mexicana provenientes del [Censo Económico 2019](#).

Dentro de los indicadores incluidos se encuentran aquellos asociados al financiamiento, tecnologías de información, características del personal ocupado y manejo del negocio, compras y ventas por internet, medio ambiente y ciencia y tecnología.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de la siguiente forma:

```
(bamboo-cli)  ~/data-etl/etl/inegi_economic_census/$ bamboo-cli --folder .
--entry indicators_economic_census_2019
```

Lectura de datos

El pipeline utiliza 29 conjuntos de datos extraídos de los tabulados del Censo Económico 2019. Cada archivo tiene su estructura propia dependiendo del indicador en cuestión y, por ende, también se realiza un procesamiento diferente a cada archivo de datos para finalizar concatenándolos en una única tabla de datos.

A modo de ejemplo, a continuación se visualiza un extracto de la estructura inicial del conjunto de datos que contiene el indicador asociado a fuentes de financiamiento por tamaño de la unidad económica. La estructura inicial se encuentra en formato .xlsx, posee varias filas vacías y otras combinadas, pero en términos generales cuenta con 14.386 filas y 24 columnas:

Entidad federal	Código de actividad económica		Denominación	Total de unidades económicas	Unidades económicas que si obtuvieron financiamiento		Bancos		Cajas de ahorro popular		Proveedores (incluye contado comercial)	
	Sector	Subsector			Absoluto	Porcentaje	Absoluto	Porcentaje	Absoluto	Porcentaje	Absoluto	Porcentaje
00 NAL			Total nacional	4.799.915	596.330	12.42	275.103	46.13	116.445	19.86	60.334	10.22
00 NAL		01) 0 a 10	Hasta 10 personas	4.555.234	526.766	11.56	218.302	41.44	117.043	22.22	48.826	9.27
00 NAL		02) 11 a 50	11 a 50 personas	193.376	50.116	25.92	40.490	80.79	1.286	2.57	8.604	17.17
00 NAL		03) 51 a 250	51 a 250 personas	40.531	15.834	38.85	13.869	85.95	101	0.46	2.377	17.01
00 NAL		04) 251 y más	251 y más personas	10.544	3.114	34.28	2.702	74.76	15	0.42	733	20.28
00 NAL	Sector 11		Sector 11 Agricultura, cría y explotación de animales, aprovechamiento forestal, pesca y caza (solo Pesc y Aprovech)	24.372	1.795	7.37	724	40.89	372	20.72	285	15.68
00 NAL	Sector 11	01) 0 a 10	Hasta 10 personas	19.729	1.142	5.79	467	40.89	303	26.53	92	8.08
00 NAL	Sector 11	02) 11 a 50	11 a 50 personas	3.768	488	12.95	176	36.07	60	12.30	151	30.94
00 NAL	Sector 11	03) 51 a 250	51 a 250 personas	822	143	17.40	72	50.35	9	8.28	39	27.27
00 NAL	Sector 11	04) 251 y más	251 y más personas	133	32	41.57	19	85.85	0	0.00	3	13.64
00 NAL	Sector 11	Subsector 112	Subsector 112 Cria y explotación de animales	3.658	523	14.27	162	30.98	147	28.11	118	22.56
00 NAL	Sector 11	Subsector 112 (01) 0 a 10	Hasta 10 personas	3.079	325	10.56	78	24.00	123	37.85	53	16.31
00 NAL	Sector 11	Subsector 112 (02) 11 a 50	11 a 50 personas	419	151	30.26	47	51.13	22	14.47	50	30.07
00 NAL	Sector 11	Subsector 112 (03) 51 a 250	51 a 250 personas	78	40	50.63	31	77.50	2	5.00	3	12.00
00 NAL	Sector 11	Subsector 112 (04) 251 y más	251 y más personas	9	7	77.78	6	85.71	0	0.00	1	14.29
00 NAL	Sector 11	Subsector 114	Subsector 114 Pesc, caza y captura	19.627	1.149	5.78	482	42.69	204	18.07	159	13.29
00 NAL	Sector 11	Subsector 114 (01) 0 a 10	Hasta 10 personas	15.543	759	4.79	356	46.50	172	22.66	34	4.64
00 NAL	Sector 11	Subsector 114 (02) 11 a 50	11 a 50 personas	3.072	282	9.19	96	34.04	27	9.57	83	29.43
00 NAL	Sector 11	Subsector 114 (03) 51 a 250	51 a 250 personas	683	80	11.71	24	30.00	5	6.25	31	38.75
00 NAL	Sector 11	Subsector 114 (04) 251 y más	251 y más personas	29	8	27.59	6	75.00	0	0.00	2	25.00

Figura 10. Formato original indicadores del Censo Económico 2019 - Financiamiento

Es importante notar que cada archivo posee diferentes cantidades de columnas y filas, por ello, el pipeline es extenso dado que debe hacer un pre-procesamiento de cada archivo en forma separada.

Limpieza y Transformación

El proceso de limpieza y transformación inicia con la lectura de los 29 archivos de datos y otros dos archivos de dimensiones que contienen el nombre de los indicadores y categorías en español e inglés junto a sus *id's*.

Las primeras líneas crean los diccionarios para categorías e indicadores usando los 2 archivos leídos previamente. Estos diccionarios serán utilizados al finalizar el proceso para estandarizar los datos.

Luego se procesan los archivos de datos agrupándolos por indicadores:

Indicadores de financiamiento

Financiamiento por tamaño y edad de las unidades económicas:

- Lectura de los archivos *crednce19_03*, *crednce19_04*, *crednce19_05*, *crednce19_06*.

- Definición de variables que permiten procesar cada archivo según características propias de su estructura.
- Eliminación de filas que contienen las palabras "Todos", "Sector", "Subsector". Esto con el objetivo de eliminar filas con totales que son redundantes cuando se trabaja en formato *tidy*.
- Aplicación de la función *general_format* alojada en la carpeta *util* y desarrollada para evitar pasos repetitivos en el pipeline.
- Creación de las variables de interés: *indicator*, *value_no_financing*, *pct_no_financing*
- Creación de dos *DataFrames*: *df_crednce19_03_B1* y *df_crednce19_03_B2* con las columnas de interés.

Fuentes de financiamiento:

- Aplicación de la función *categories_format* alojada en la carpeta *util* y desarrollada para evitar pasos repetitivos en el pipeline. En este caso, se aplica al archivo *crednce19_03* que detalla las fuentes de financiamiento utilizadas por las unidades económicas

Se concatenan los *DataFrames* que ya tienen la estructura deseada generando el *DataFrame df_financing1*.

Acceso a financiamiento, cuenta bancaria y crédito bancario a nivel nacional y estatal:

- Lectura de los archivos *crednce19_03*, *crednce19_07*, *crednce19_08*.
- Selección de columnas de interés.
- Aplicación de la función *geo_data* alojada en la carpeta *util* y desarrollada para evitar pasos repetitivos en el pipeline.
- Creación de la columna *indicator*.
- Unión de los archivos con la función *append* de pandas para obtener el *DataFrame df_financing2*.

Razones para no acceder a crédito o cuenta bancaria:

- Lectura de los archivos `crednce19_07` y `crednce19_08`.
- Definición del nombre de columnas de interés.
- Aplicación de las funciones `general_format`, `yes_no_format` y `categories_format` alojadas en la carpeta `util` y desarrolladas para evitar pasos repetitivos en el pipeline.
- Unión de los archivos con la función `append` de pandas para obtener el `DataFrame df_financing3`.

Finalmente se concatenan los `DataFrames df_financing1`, `df_financing2` y `df_financing3` para obtener una única tabla con los indicadores de financiamiento.

Indicadores de acceso a internet

Acceso a internet según cantidad de unidades económicas:

- Lectura de los archivos `ecomnce19_05` y `ecomnce19_06`.
- Definición de variables que permiten procesar cada archivo según características propias de su estructura.
- Aplicación de la función `general_format` alojada en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Selección de columnas de interés.
- Aplicación de la función `melt` de pandas para lograr una tabla en formato `tidy`.

Acceso a internet según porcentaje de unidades económicas:

- Lectura de los archivos `ecomnce19_03` y `ecomnce19_03`.
- Se aplican los mismos pasos anteriores para conseguir una tabla con los valores porcentuales

Se concatenan los `DataFrames` que contienen valores y porcentajes generando `df_internet1`.

Acceso a internet según nivel geográfico:

- Lectura de los archivos `ecomnce19_05` y `ecomnce19_06`.
- Definición de variables que permiten procesar cada archivo según características propias de su estructura.
- Selección de columnas de interés.
- Aplicación de la función `geo_data` alojada en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Creación de la columna indicator.
- Unión de los archivos con la función `append` de pandas para obtener el `DataFrame df_internet2`.

Finalmente se concatenan los `DataFrames df_internet1` y `df_internet2` para obtener una única tabla con los indicadores de internet.

Indicadores de manejo del negocio

Problemáticas que enfrentan las unidades económicas por sector económico:

- Lectura del archivo `probnce19_03`.
- Definición del nombre de las columnas.
- Aplicación de las funciones `general_format` y `categories_format` alojadas en la carpeta `util` y desarrolladas para evitar pasos repetitivos en el pipeline.

Problemáticas que enfrentan las unidades económicas por nivel geográfico:

- Eliminación de filas innecesarias del archivo `probnce19_03` y renombre de columnas.
- Selección de filas que contienen la palabra "personas" en la columna `category` y que contienen nulos en la columna `sector_id`.
- Reemplazo de valores nulos por ceros.
- Creación de las columnas `ent_id` y `nation_id`.

- Aplicación de la función `categories_format` alojada en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Creación de la columna `indicator`.

Se concatenan las tablas generadas para las problemáticas a nivel geográfico y sector económico en el `DataFrame df_business1`.

Cuentan con sistema contable:

- Lectura del archivo `contnce19_02`.
- Definición del nombre de las columnas.
- Aplicación de la función `general_format` alojada en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Creación de la columna `subsector_id`.
- Aplicación de las funciones `yes_no_format` y `categories_format` alojadas en la carpeta `util` y desarrolladas para evitar pasos repetitivos en el pipeline.
- Creación de un nuevo `DataFrame` a partir del archivo `contnce19_02` que contendrá los valores por nivel geográfico.
- Aplicación de la función `geo_data` alojada en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Creación de la columna `indicator`.
- Concatenación de archivos para conseguir la tabla `df_business2`.

Transacciones monetarias:

- Lectura del archivo **`pagonce19_02`**.
- Definición del nombre de las columnas.
- Aplicación de las funciones `general_format` y `categories_format` alojadas en la carpeta `util` y desarrolladas para evitar pasos repetitivos en el pipeline.
- Creación de un nuevo `DataFrame` a partir del archivo `pagonce19_02` que contendrá los valores por nivel geográfico.

- Aplicación de la función `geo_data` alojada en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Aplicación de la función `melt` de pandas para lograr una tabla en formato `tidy`.
- Creación de las columnas `percentage` e `indicator`.
- Concatenación de archivos para conseguir la tabla `df_business3`.

Finalmente se concatenan los `DataFrames` `df_business1`, `df_business2` y `df_business3` para obtener una única tabla con los indicadores de manejo del negocio.

Indicadores de personal

Rotación y permanencia del personal:

- Lectura del archivo `capance19_02`.
- Definición del nombre de las columnas.
- Aplicación de las funciones `general_format` y `yes_no_format` alojadas en la carpeta `util` y desarrolladas para evitar pasos repetitivos en el pipeline.
- Creación de la columna `indicator`.
- Con esto se logra el `DataFrame` `df_staff1`.

Capacitación del personal:

- Lectura del archivo `edadnace19_02`.
- Definición del nombre de las columnas.
- Aplicación de la función `general_format` alojada en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Selección de variables de interés.
- Creación de la columna `category` que utiliza un diccionario creado previamente.
- Creación de dos `DataFrames`, uno contiene los datos de capacitación según edad del personal y el otro según nivel de estudios del personal.

- En cada uno de los *DataFrames* se renombra la columna *category* por *indicator*, se aplica la función *melt* de pandas para conseguir una tabla en formato *tidy* y se crean las columnas *percentage* e *indicator*.
- Concatenación de los *DataFrames* para conseguir la tabla *df_staff2*.

Rotación y permanencia del según edad y nivel educacional:

- Lectura del archivo *edadnace19_03*.
- Definición del nombre de las columnas.
- Aplicación de las funciones *general_format* y *yes_no_format* alojadas en la carpeta *util* y desarrolladas para evitar pasos repetitivos en el pipeline.
- Creación de la columna *indicator*.
- Mediante un ciclo *for* se trabajan en forma separada los *DataFrames* con edad y educación y luego se unen con la función *append* de pandas. En el ciclo se seleccionan las columnas de interés, se aplica la función *melt* de pandas para conseguir una tabla en formato *tidy*, se crean las columnas *percentage* e *indicator*, se renombran las columnas *category* por *indicator* y *category_temp* por *category* y se elimina la columna *total_staff*. Al finalizar el ciclo se obtiene el *DataFrame* *df_staff3*.

Finalmente se concatenan los *DataFrames* *df_staff1*, *df_staff2* y *df_staff3* para obtener una única tabla con los indicadores de personal.

Indicadores de tecnologías de información

Servicios de cómputo e internet:

- Lectura de los archivos *ticsnace19_01* y *ticsnace19_02*.
- Definición de variables que permiten procesar cada archivo según características propias de su estructura y definición del nombre de columnas.
- Aplicación de la función *general_format* alojada en la carpeta *util* y desarrolladas para evitar pasos repetitivos en el pipeline.
- Aplicación de la función *yes_no_format* dos veces consecutivas para generar dos *DataFrames*, uno asociado a servicios de internet y otro asociado a servicios de cómputo.

- Creación de la columna *indicator*.
- Concatenación de los *DataFrames* para conseguir la tabla *df_tic1*.

Usos de internet:

- Lectura de los archivos *ticsnce19_03* y *ticsnce19_04*.
- Definición de variables que permiten procesar cada archivo según características propias de su estructura y definición del nombre de columnas.
- Aplicación de las funciones *general_format* y *categories_format* alojadas en la carpeta *util* y desarrolladas para evitar pasos repetitivos en el pipeline.
- Creación de la columna *indicator*.
- Concatenación de los *DataFrames* para conseguir la tabla *df_tic2*.

Usos de internet por nivel geográfico:

- Lectura del archivo *ticsnce19_03*.
- Selección de columnas de interés.
- Aplicación de la función *geo_data* alojada en la carpeta *util* y desarrollada para evitar pasos repetitivos en el pipeline.
- Creación de la columna *indicator*.
- Con esto se logra el *DataFrame* *df_tic3*.

Finalmente se concatenan los *DataFrames* *df_tic1*, *df_tic2* y *df_tic3* para obtener una única tabla con los indicadores de tecnologías de información.

Indicadores de innovación

Innovación por nivel geográfico:

- Lectura del archivo *innonce19_01*.
- Definición de columnas de interés.
- Aplicación de la función *geo_data* alojada en la carpeta *util* y desarrollada para evitar pasos repetitivos en el pipeline.

- Renombre de columnas.
- Creación de las columnas *percentage* e *indicator*.
- Eliminación de la columna *total_ue*.
- Con esto se logra el DataFrame *df_innovation1*.

Actividades de innovación en 2018:

- Lectura del archivo *innonce19_04*.
- Definición de columnas de interés.
- Aplicación de la función *general_format* alojada en la carpeta *util* y desarrollada para evitar pasos repetitivos en el pipeline.
- Creación de la columna *category*.
- Selección de las columnas de interés
- Creación de la columna *pct_**.
- Renombre de columnas.
- Aplicación de la función *yes_no_format* alojada en la carpeta *util* y desarrollada para evitar pasos repetitivos en el pipeline.
- Con esto se logra el DataFrame *df_innovation2*.

Personal en actividades de innovación en 2018:

- Se utiliza el archivo *innonce19_04* leído previamente.
- Selección de las columnas de interés
- Creación de la columna *pct_**. Se debe tener cuidado con las divisiones por cero. Por ello se condiciona la división sólo si el denominador es diferente de cero y luego llenan las columnas vacías con ceros.
- Renombre de columnas.
- Aplicación de la función *categories_format* alojada en la carpeta *util* y desarrollada para evitar pasos repetitivos en el pipeline.
- Con esto se logra el DataFrame *df_innovation3*.

Innovación por año:

- Lectura del archivo *innonce19_06*.
- Definición de columnas de interés.
- Aplicación de la función *general_format* alojada en la carpeta *util* y desarrollada para evitar pasos repetitivos en el pipeline.
- Creación de la columna *category*.
- Selección de las columnas de interés.
- Aplicación de la función *melt* de pandas para lograr una tabla en formato *tidy*.
- Creación de la columna *percentage*.
- Con esto se logra el *DataFrame df_innovation4*.

Finalmente se concatenan los *DataFrames df_innovation1*, *df_innovation2*, *df_innovation3* y *df_innovation4* para obtener una única tabla con los indicadores de innovación.

Indicadores de medio ambiente

Conocimiento de las normas ambientales:

- Lectura del archivo *medance19_01*.
- Eliminación de filas innecesarias del archivo
- Definición y renombre de columnas de interés.
- Selección de filas no vacías en la columna *sector_id*.
- Modificación de las columnas *ent_id*, *nation_id* y *sector_id* para dejarlas estandarizadas.
- Se crean dos *DataFrames*, uno para cantidad de empresas y otro para valores porcentuales. A ambos se le aplica la función *melt* de pandas para llevar las tablas a formato *tidy*.
- Aplicación de la función *merge* de pandas para unir ambas tablas.
- Creación de las columnas *subsector_id* e *indicator*.

- Con esto se logra el DataFrame `df_enviromental1`.

Personal involucrado en actividades de protección ambiental:

- Lectura del archivo `medance19_05`.
- Eliminación de filas innecesarias del archivo
- Definición y renombre de columnas de interés.
- Selección de filas no vacías en la columna `sector_id`.
- Modificación o creación de las columnas `ent_id`, `nation_id`, `sector_id` `subsector_id` y `category` para dejarlas estandarizadas.
- Aplicación de la función `yes_no_format` alojada en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Con esto se logra el DataFrame `df_enviromental2`.

Separación de residuos:

- Lectura del archivo `medance19_03`.
- Definición del nombre de columnas de interés.
- Aplicación de las funciones `environmental`, `yes_no_format` y `categories_format` alojadas en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Eliminación de la columna `category`.
- Renombre de la columna `indicator` por `category`.
- Creación de la columna `indicator`.
- Concatenación de tablas para generar el DataFrame `df_enviromental3`.

Gastos e inversión en actividades de protección del medio ambiente / Usos del agua:

- Lectura de los archivos `medance19_07`, `medance19_08` y `medance19_09`.
- Definición de variables y nombres de columnas que permiten procesar cada archivo según características propias de su estructura.

- Aplicación de la funciones `environmental` y `yes_no_format` alojadas en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Creación de un nuevo `DataFrame` donde se calculan los valores porcentuales de los indicadores.
- Modificación en el nombre de columnas que contienen el `string` “`value _`”.
- Aplicación de la función `categories_format` alojada en la carpeta `util` y desarrollada para evitar pasos repetitivos en el pipeline.
- Eliminación de la columna `category`.
- Renombre de la columna `indicator` por `category`.
- Creación de la columna `indicator`.
- Concatenación de tablas para generar el `DataFrame df_enviromental4`.

Finalmente se concatenan los `DataFrames df_enviromental1`, `df_enviromental2`, `df_enviromental3` y `df_enviromental4` para obtener una única tabla con los indicadores de medio ambiente.

Como último paso y ya habiendo procesado todos los archivos, se concatenan los `DataFrames` finales de cada etapa para generar una única tabla con todos los indicadores, se usan los diccionarios definidos al inicio para estandarizar las columnas `indicator` y `category`, las columnas correspondientes se pasan a formato entero o flotante, según corresponda, y se reemplaza el `id` de nación por `mex`.

El `DataFrame` que se obtiene tras el proceso de ETL agrupa todos los indicadores en una tabla con solo 8 columnas y varios miles de filas. Un extracto de la tabla final se muestra a continuación:

nation_id	ent_id	sector_id	subsector_id	value	percentage	indicator	category
1	0	11	0	1142	5.79	115	1
1	0	11	0	488	12.95	115	2
1	0	11	0	143	17.4	115	3
1	0	11	0	22	41.51	115	4
1	0	0	112	325	10.56	115	1
...
1	0	0	812	0	0	12	4
1	0	0	813	550	3.61	12	1
1	0	0	813	151	4.49	12	2
1	0	0	813	4	2.58	12	3
1	0	0	813	0	0	12	4

Figura 11. Formato final tabla de indicadores del Censo Económico 2019 - Financiamiento

Funciones adicionales

Durante la explicación del proceso de ETL se mencionaron algunas funciones desarrolladas para evitar código repetitivo al momento de procesar cada archivo. A continuación se entrega una breve descripción de ellas.

1) general_format

A la función se le pasan 6 parámetros que se utilizan en diferentes partes del código:
`df, a, b, column_name, fix_subsector, rows_drop.`

- El primer parámetro corresponde al *DataFrame*.
- El segundo y tercer parámetro son útiles para eliminar filas vacías de los archivos.
- El cuarto parámetro es una variable que contiene el nombre de las columnas del *DataFrame*.
- El quinto parámetro permite reemplazar una palabra en la columna *subsector_id*.

- El último parámetro permite eliminar de la columna `category` todas las filas que contengan los `string` indicados en el parámetro.
- Adicionalmente se aplican otras transformaciones para estandarizar el `DataFrame`, entre ellas, modificaciones en las filas `category`, `sector_id` y `subsector_id`; la columna `ent_id` contendrá sólo ceros y la columna `nation_id` contendrá sólo unos.

2) yes_no_format

A la función se le pasan 3 parámetros que se utilizan en diferentes partes del código: `df`, `a` y `b`.

- Mediante un ciclo `for` se crean dos `DataFrames` con un grupo de columnas en común y otras columnas dadas por los parámetros `a` y `b`.
- Creación de la columna `indicator` usando los parámetros `a` y `b`.
- Renombre de columnas
- Concatenación de ambos `DataFrames`.

3) categories_format

A la función se le pasan 2 parámetros que se utilizan en diferentes partes del código: `dfy` y `names`.

- Creación de un `DataFrame` que contiene un grupo de columnas fijas y un listado de columnas dadas por el parámetro `names` que contienen el `string` "value".
- Se aplica la función `melt` de pandas para llevar el `DataFrame` a un formato `tidy`.
- Se repite el proceso creando un segundo `DataFrame` que ahora contendrá las columnas dadas por el parámetro `names` que contienen el `string` "pct".
- Se utiliza la función `merge` de pandas para unir ambos `DataFrames` donde corresponda.

4) geo_data

A la función se le pasan 4 parámetros que se utilizan en diferentes partes del código: *df*, *selected_columns*, *cut1* y *cut2*.

- Los parámetros *cut1* y *cut2* se utilizan para eliminar filas innecesarias de los archivos.
- El parámetro *selected_columns* se utiliza para crear el *DataFrame* con las columnas de interés contenidas en el parámetro.
- Se eliminan filas que contengan los *strings* “personas”, “Todos” y “Rama” en la columna *category*.
- Se modifican las columnas *sector_id*, *subsector_id*, *ent_id* y *nation_id* para estandarizar los valores.

5) environmental

A la función se le pasan 4 parámetros que se utilizan en diferentes partes del código: *df*, *a*, *b* y *column_name*.

- Los parámetros *a* y *b* se utilizan para eliminar filas innecesarias de los archivos.
- El parámetro *column_name* se utiliza para renombrar las columnas del *DataFrame*.
- Se seleccionan las filas de interés: aquellas que presentan "00 NAL" en la columna *ent_id* y aquellas donde *sector_id* es nulo.
- Se modifican las columnas *sector_id*, *subsector_id*, *ent_id* y *nation_id* para estandarizar los valores.

2. Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH)

A continuación se detalla el proceso de ETL para los datos de la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH). Se desarrollaron cuatro pipelines que separan los datos de ingresos del hogar, gastos del hogar, población y deciles.

2.1. Inegi_enigh_expense_items

Descripción General y Ejecución

El pipeline `enigh_expense_items.py` procesa los datos facilitados por el Instituto Nacional de Estadística y Geografía (INEGI) que toman relación con la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH). En ello, el pipeline se refiere a las partidas de gastos. Los datos son descargados desde la plataforma del [Instituto Nacional de Estadística y Geografía](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas, numpy y string y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline de la siguiente forma:

(Python) `~/data_etl/etl/enigh/$ python enigh_expense_items.py`

(bamboo-cli) `~/data_etl/etl/enigh/$ bamboo-cli --folder . --entry enigh_expense_items --year=<year_value>`

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y existe un archivo por año. Para el año 2020 se identifican 1.462 filas y 8 columnas. A continuación, se visualiza la estructura inicial de los datos.

	0	1	2	3	4	5	6	7
	NACIONAL	NaN	NaN	1.069282e+09	NaN	29910.260144	NaN	NaN
ALIMENTOS, BEBIDAS Y TABACO	NaN	NaN	4.068325e+08	NaN	11380.039840	NaN	NaN	
ALIMENTOS Y BEBIDAS CONSUMIDAS DENTRO DEL HOGAR	NaN	NaN	3.504048e+08	NaN	9801.625402	NaN	NaN	
CEREALES	NaN	NaN	5.898776e+07	NaN	1650.023030	NaN	NaN	
CARNES	NaN	NaN	8.016922e+07	NaN	2242.516993	NaN	NaN	

Figura 12. Formato original datos gastos de los hogares (ENIGH)

Extracción y Transformación

En este etapa se procede de la siguiente forma:

- Selección de filas y columnas. Se eliminan las últimas 10 filas ya que poseen anotaciones y también, las últimas columnas ya que poseen valores nulos.
- Renombre de variables.
- Se aplica la función *capwords* de *pandas* para capitalizar el nombre de las partidas de gasto.
- Creación de la variable *ent_id*, la cual define la entidad en cuestión.
- Utilización de la función *ffill()* de *pandas* para llenar espacios vacíos en la variable *ent_id*.
- Eliminación de valores iguales a algún nombre de estado en la variable *expense_item*.
- Lectura desde la base de datos en *clickhouse* de la tabla *dim_shared_geography_ent*. Dichos datos son almacenados en un *dataframe* de *pandas* llamado *states_db*.
- Reemplazo de la entidad México por Estado De México.
- Reemplazo de valores en la variable *ent_id* por los *ids*, haciendo uso del *dataframe* *states_db*.
- Creación de la variable *year*, la cual define el periodo en estudio.

- Creación de la variable `nat_id`, la cual toma el valor de mex si el valor de la variable `ent_id` es igual a cero, y el valor de cero en algún otro caso.
- Reemplazo de valores en la variable `expense_item` por sus id. Para ello se llama a un conjunto de datos que facilita el reemplazo.
- Transformación de tipo de dato a entero para las variables: `year`, `expense_item`, `ent_id`, `expense`, y `mean`.
- Utilización de deflactores entregados por la Secretaría de Economía para transformar los valores de años 2016 y 2018 a montos del 2020 para que puedan ser comparados.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 1.188 filas y 5 columnas. La estructura final de la tabla se presenta a continuación:

<code>nat_id</code>	<code>ent_id</code>	<code>year</code>	<code>expense_item</code>	<code>expense</code>
mex	0	2020	1101	58987760
mex	0	2020	1102	80169217
mex	0	2020	1103	8770594
mex	0	2020	1104	31932749
mex	0	2020	1105	13341157

Figura 13. Formato final tabla de datos gastos de los hogares (ENIGH)

2.2. Inegi_enigh_income_source

Descripción General y Ejecución

El pipeline `enigh_enigh_source.py` procesa los datos facilitados por el Instituto Nacional de Estadística y Geografía (INEGI) que toman relación con la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH). En ello, el pipeline se refiere a las fuentes de ingreso. Los datos son descargados desde la plataforma del [Instituto Nacional de Estadística y Geografía](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas, numpy, string y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

(Python) `~/data_etl/etl/enigh/$ python enigh_income_source.py`

(bamboo-cli) `~/data_etl/etl/enigh/$ bamboo-cli --folder . --entry enigh_income_source --year=<year_value>`

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería `bamboo-lib` se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y existe un archivo por año. Para el año 2020 se identifican 209 filas y 15 columnas. A continuación, se visualiza la estructura inicial de los datos.

0	1	... 13 14
NACIONAL	NaN	... NaN NaN
INGRESO DEL TRABAJO	NaN	... NaN NaN
RENTA DE LA PROPIEDAD	NaN	... NaN NaN
TRANSFERENCIAS	NaN	... NaN NaN
ESTIMACIÓN DEL ALQUILER DE LA VIVIENDA	NaN	... NaN NaN

Figura 14. Formato original datos por tipos de ingreso ENIGH

Lectura y Transformación

Esta etapa sigue el siguiente proceso:

- Selección de filas y columnas. Se eliminan las últimas 11 filas ya que poseen anotaciones y también, las últimas 13 columnas ya que poseen valores nulos.
- Renombre de variables.
- Se aplica la función `melt` de pandas, para transformar la estructura del `dataframe` y obtener un formato `tidy`.
- Se aplica la función `capwords` de pandas para capitalizar el nombre de las fuentes de ingreso.
- Creación de la variable `ent_id`, la cual define la entidad en cuestión. Dicha columna se crea a partir de valores en la variable `income_source`.
- Utilización de la función `ffill()` de pandas para llenar espacios vacíos en la variable `ent_id`.
- Eliminación de valores iguales a algún nombre de estado en la variable `income_source`.
- Lectura desde la base de datos en `clickhouse` de la tabla `dim_shared_geography_ent`. Dichos datos son almacenados en un `dataframe` de pandas llamado `states_db`.
- Reemplazo de la entidad México por Estado De México.
- Reemplazo de valores en la variable `ent_id` por los `ids`, haciendo uso del `dataframe` `states_db`.
- Creación de la variable `year`, la cual define el periodo en estudio.

- Creación de la variable `nat_id`, la cual toma el valor de mex si el valor de la variable `ent_id` es igual a cero, y el valor de cero en algún otro caso.
- Reemplazo de valores en la variable `income_source` por sus id. Para ello se llama a un conjunto de datos que facilita el reemplazo.
- Transformación de tipo de dato a entero para las variables: `year`, `income_source`, `ent_id`, y `value`.
- Utilización de deflactores entregados por la Secretaría de Economía para transformar los valores de años 2016 y 2018 a montos del 2020 para que puedan ser comparados.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 1.650 filas y 6 columnas. La estructura final de la tabla se presenta a continuación:

<code>nat_id</code>	<code>ent_id</code>	<code>year</code>	<code>decile</code>	<code>income_source</code>	<code>value</code>
mex	0	2020	I	1	15608541
mex	0	2020	I	2	356286
mex	0	2020	I	3	11253576
mex	0	2020	I	4	8189438
mex	0	2020	I	5	119363

Figura 15. Formato final tabla de datos ingresos ENIGH

2.3. inegi_enigh_income

Descripción General y Ejecución

El pipeline *enigh_decile_income* procesa los datos facilitados por el Instituto Nacional de Estadística y Geografía (INEGI) que toman relación con la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH). En ello, el pipeline se refiere a los deciles de ingreso. Los datos son descargados desde la plataforma del [Instituto Nacional de Estadística y Geografía](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de Clickhouse, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

(bamboo-cli)

```
~/data_etl/etl/enigh/$ bamboo-cli --folder . --entry enigh_decile_income
--year=<year_value>
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx; y se identifican 12 columnas y 44 filas. A continuación, se visualiza la estructura inicial de los datos.

	Unnamed: 0	I	II	III	IV	V	VI	VII	VIII	IX	X
0	Estados Unidos Mexicanos	9113.0	16100.0	21428.0	26696.0	32318.0	38957.0	47264.0	58885.0	78591.0	166750.0
1	Aguascalientes	12902.0	21645.0	28059.0	34001.0	41215.0	48847.0	57987.0	70360.0	92628.0	185797.0
2	Baja California	14265.0	22711.0	28532.0	34732.0	41413.0	48917.0	58997.0	73533.0	98477.0	170193.0
3	Baja California Sur	15578.0	26001.0	33234.0	40973.0	48877.0	57578.0	69223.0	84042.0	104247.0	208003.0
4	Campeche	8686.0	14720.0	19538.0	24371.0	29640.0	35722.0	44455.0	57599.0	77733.0	164497.0
5	Coahuila de Zaragoza	12288.0	20587.0	26802.0	32449.0	38573.0	45343.0	54510.0	66729.0	89461.0	172504.0

Figura 16. Formato original datos deciles de ingresos ENIGH

Extracción y Transformación

Este paso del pipeline sigue el siguiente proceso:

- Transformar el archivo .xlsx a un *DataFrame* de Pandas.
- Renombre de la columna *Unnamed: 0* a *ent_id*.
- Lectura de un archivo que contiene los estados, para reemplazar el nombre del estado en la variable *ent_id* con el id del estado.
- Transformación de la estructura a un formato *tidy*. Para ello, se utiliza la función *melt* de Pandas. La idea de ello es tener una mejor visualización de los deciles de Ingreso.
- Creación de la columna *year* que toma relación al año de estudio.

Posterior al proceso de transformación, obtenemos un *DataFrame* con 4 columnas y 330 filas. La estructura final de la tabla se presenta a continuación:

ent_id	decile	value	year
0	33	I	9113 2018
1	1	I	12902 2018
2	2	I	14265 2018
3	3	I	15578 2018
4	4	I	8686 2018
...

Figura 17. Formato final tabla de datos con deciles de ingreso

3. Empleo

A continuación se detalla el proceso de ETL para los datos de la Encuesta Nacional de Ocupación y Empleo (ENOE) y para la Encuesta Telefónica de Ocupación y Empleo (ETOE) que surgió como una alternativa a ENOE durante los primeros meses de la pandemia generada por el COVID-19. Cabe mencionar la desagregación por industrias utiliza el Sistema de Clasificación Industrial de América del Norte, SCIAN 2018, mientras que las ocupaciones utilizan Sistema Nacional de Clasificación de Ocupaciones, SINCO.

3.1. Inegi_enoe

Descripción general y ejecución

El pipeline `enoe_pipeline.py` de ENOE (Encuesta Nacional de Ocupación y Empleo) es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos de dicha encuesta entregados por INEGI (Instituto Nacional de Estadística y Geografía). Estos datos ofrecen información mensual y trimestral de la fuerza de trabajo, la ocupación, la informalidad laboral, la subocupación y la desocupación.

Los datos son descargados desde [el repositorio de datos de INEGI](#) y almacenados en un compartimiento de Google Storage privado, pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos Clickhouse siguiendo un modelo relacional.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de la siguiente forma:

(bamboo-cli)

```
~/data-etl/etl/enoe/$ bamboo-cli --folder . --entry enoe_pipeline
```

Descarga de datos

Desde el sitio de INEGI se descargan cuatro archivos para ingestar, estos son "*coel1t*", "*coe2t*", "*sdemt*" y "*vivt*", todos en formato CSV, cada uno corresponde a una parte de la ENOE y a un año y trimestre específico. Una vez descargados estos archivos son almacenados en un compartimiento privado de GCP storage, luego, se establece una conexión privada validada con credenciales fuertemente encriptadas, para ejecutar una descarga segura hasta el servidor de procesamiento de los datos.

Lectura de datos

Una vez descargados los datos, estos se leen y almacenan en un *DataFrame* de la librería pandas, cada uno de los archivos son leídos por separados y guardados en *DataFrames* separados, a continuación se presentan las primeras filas y columnas de estos archivos.

	cd_a	ent	con	v_sel	n_hog	h_mud	n_ren	eda	p1b	p2_1	...	p2_9	p2a_anio	p2b	\
0	01	09	00501	01	1	0	01	35			...			NaN	
1	01	09	00501	01	1	0	02	33	2		...		2009	1	
2	01	09	00501	02	1	0	01	32	1		...			NaN	
3	01	09	00501	02	1	0	02	29	1		...			NaN	
4	01	09	00501	03	1	0	01	54	2		...			NaN	
	p3	p4a	p5b_thrs	p5b_tdia	fac				code						
0	6270	5510	032	4	1060	0900501011001									NaN
1	NaN	NaN	NaN		1060	0900501011002									NaN
2	6260	2210	NaN		1060	0900501021001									NaN
3	1330	6112	NaN		1060	0900501021002									NaN
4	NaN	NaN	NaN		1060	0900501031001									NaN

Figura 18. Formato original datos del archivo *coel1t*

Al momento de la elaboración de esta documentación, el *DataFrame* inicial proveniente del archivo "*coel1t*" contenía 312.167 filas por 23 columnas.

Figura 19. Formato original datos del archivo coe2t

El *DataFrame* proveniente del archivo "coe2t" contenía 312.167 filas por 14 columnas

Figura 20. Formato original datos del archivo sdmel

El *DataFrame* proveniente del archivo "sdmet" contenía 406.797 filas por 24 columnas.

	mun	ent	con	v_sel	code
0	001	01	00001	01	010000101
1	001	01	00001	02	010000102
2	001	01	00001	03	010000103
3	001	01	00001	04	010000104
4	001	01	00001	05	010000105

Figura 21. Formato original datos del archivo vivt

El *DataFrame* proveniente de "vivt" contiene 120.427 filas por 5 columnas.

Transformación y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Primero, se estandarizan los nombres de las columnas de los dos primeros *DataFrames*, quitando símbolos no deseados y dejando todo en letras minúsculas. Continuando con la estandarización, se reindexan ambos *DataFrames* basándose en valores únicos individuales de sus columnas. Finalmente, se juntan ambos *DataFrames* en uno.

Posteriormente, se cargan los siguientes dos *DataFrames*, los relacionados a datos sociodemográficos y a vivienda, los cuales son sometidos a estandarización de sus datos tal como los anteriores. Luego se juntan estos *DataFrames* con los dos anteriores, generando un solo *DataFrame* que contiene todos los datos.

Se generan los números identificadores geográficos, basados en los municipios y entidades federativas. Luego se reemplazan los valores nulos por valores numéricos para poder realizar agregaciones de los datos. Se generan números identificadores para cada columna y este reemplaza al valor original en el *DataFrame*, por ejemplo; "Posee un trabajo o negocio" tendrá el número 1 si posee trabajo o negocio y 2 si no posee. Esta referencia será guardada por separado en otra tabla, de la forma; "posee trabajo o negocio" = 1; "No" = 2.

Posteriormente creados todos los ID's, se procede a agrupar el *DataFrame* por cada fila única. Luego se integran en el *DataFrame* los valores de ingreso para cada una de las filas.

Finalmente, se estandariza el *DataFrame* por última vez, quitando los valores repetidos e innecesarios, también validando el tipo de dato que es cada columna, dejándolas en un tipo de dato estándar para que sea leído apropiadamente por la base de datos.

Al momento en el que se elaboró esta documentación, el *DataFrame* final contenía 288.614 filas y 23 columnas. La estructura final de la tabla se presenta a continuación.

	code	mun_id	population	population_monthly	mensual_wage	\
0	1T20101	9002	1060	NaN	0.0	
1	2T20101	9002	1060	NaN	0.0	
2	3T20101	9002	1060	NaN	0.0	
3	4T20101	9002	1060	NaN	0.0	
4	5T20101	9002	1060	NaN	0.0	
	has_job_or_business	second_activity	eap	occ_unocc_pop	eap_comp	...
0		NaN	7.0	1.0	1.0	1.0
1		2.0	NaN	1.0	2.0	6.0
2		1.0	7.0	1.0	1.0	3.0
3		1.0	7.0	1.0	1.0	3.0
4		2.0	NaN	2.0	3.0	0.0
	classification_formal_informal_jobs_first_activity	age	\			
0			2.0	35		
1			0.0	33		
2			2.0	32		
3			2.0	29		
4			0.0	54		
	actual_job_industry_group_id	sex	actual_job_position	\		
0		5510	1.0	6270.0		
1		0	2.0	NaN		
2		2210	1.0	6260.0		
3		6112	2.0	1330.0		
4		0	1.0	NaN		
	actual_job_hrs_worked_lastweek	actual_job_days_worked_lastweek	\			
0		32.0		4.0		
1		NaN		NaN		
2		NaN		NaN		
3		NaN		NaN		
4		NaN		NaN		
	workforce_is_wage	workforce_is_wage_monthly	quarter_id			
0		1060	NaN	20101		
1		1060	NaN	20101		
2		1060	NaN	20101		
3		1060	NaN	20101		
4		1060	NaN	20101		

Figura 22. Formato final tabla de datos ENOE

3.2. inegi_etoie

Descripción general y ejecución

El pipeline *etoie_pipeline.py* de ETOE (Encuesta Telefónica de Ocupación y Empleo) es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos de dicha encuesta entregados por INEGI (Instituto Nacional de Estadística y Geografía). Estos datos ofrecen información relevante para monitorear la situación de la ocupación y empleo en el periodo de contingencia del COVID-19.

Los datos son descargados desde [el repositorio de datos de INEGI](#) y almacenados en un compartimiento de Google Storage privado, pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos Clickhouse siguiendo un modelo relacional.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de la siguiente forma:

(bamboo-cli)

```
~/data-etl/etl/etoie/$ bamboo-cli --folder . --entry etoe_pipeline
```

Descarga de datos

Desde el sitio de INEGI se descargan cuatro archivos para ingestar, estos son "*coel1t*", "*coe2t*", "*sdemt*" y "*vivt*", todos en formato DBF, cada uno corresponde a una parte de la encuesta ETOE y a un año y trimestre específico. Una vez descargados estos archivos son almacenados en un compartimiento privado de GCP storage, luego, se

establece una conexión privada validada con credenciales fuertemente encriptadas, para ejecutar una descarga segura hasta el servidor de procesamiento de los datos.

Lectura de datos

Una vez descargados los datos, estos se leen y almacenan en un *DataFrame* de la librería pandas, cada uno de los archivos son leídos por separados y guardados en *DataFrames* separados, a continuación se presentan las primeras filas y columnas de estos archivos.

ent	cd_a	con	v_sel	n_hog	h_mud	n_ren	eda	p1b	p2_1	...	p2_4	p2_9	\
0	01	14	40265	05	1	0	01	55	NaN	NaN	...	NaN	NaN
1	01	14	40265	05	1	0	02	45	2	NaN	...	4	NaN
2	01	14	40299	03	1	0	01	64	2	NaN	...	4	NaN
3	01	14	40299	03	1	0	02	33	1	NaN	...	NaN	NaN
4	01	14	40299	03	1	0	03	35	NaN	NaN	...	NaN	NaN
p2a_anio	p2b	p3	p4a	p5b_thrs	p5b_tdia	fac	code						
0	NaN	NaN	8341	4840	060	6	2715	0140265051001					
1	NaN	NaN	NaN	NaN	NaN	NaN	2715	0140265051002					
2	NaN	NaN	NaN	NaN	NaN	NaN	3396	0140299031001					
3	NaN	NaN	5312	9313	NaN	NaN	3396	0140299031002					
4	NaN	NaN	2512	4681	046	6	3396	0140299031003					

Figura 23. Formato original datos del archivo coel7

Al momento de la elaboración de esta documentación, el *DataFrame* inicial proveniente del archivo "*coel7t*" contenía 23.893 filas por 22 columnas.

ent	con	v_sel	n_hog	h_mud	n_ren	p6b1	p6b2	p6c	p6d	p7	p7a	p7c	\
0	01	40265	05	1	0	01	7	NaN	4	1	7	NaN	NaN
1	01	40265	05	1	0	02	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	01	40299	03	1	0	01	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	01	40299	03	1	0	02	8	NaN	9	1	7	NaN	NaN
4	01	40299	03	1	0	03	7	NaN	9	1	7	NaN	NaN
code													
0	0140265051001												
1	0140265051002												
2	0140299031001												
3	0140299031002												
4	0140299031003												

Figura 24. Formato original datos del archivo coe2t

El *DataFrame* proveniente del archivo "*coe2t*" contenía 23.893 filas por 14 columnas

	ent	con	v_sel	n_hog	h_mud	n_ren	sex	clase1	clase2	clase3	...	\
0	09	40015	01	1	0	01	1	1	1	1	...	\
1	09	40015	01	1	0	02	2	2	4	0	...	\
2	09	40015	01	1	0	03	1	2	4	0	...	\
3	09	40015	01	1	0	04	1	1	2	6	...	\
4	09	40015	01	1	0	05	2	2	4	0	...	\
	cs_p13_2	ingocup	d_ant_lab	d_cexp_est	dur_des	sub_o	s_clasifi	cp_anoc	...	\		
0	5	40000	0	0	0	0	0	0	0	0	0	\
1	3	0	0	0	0	0	0	0	0	0	0	\
2	2	0	0	0	0	0	0	0	0	0	0	\
3	3	0	1	2	3	0	0	0	0	0	0	\
4	3	0	0	0	0	0	0	0	0	0	0	\
	emp_ppal	code										
0	2	0940015011001										
1	0	0940015011002										
2	0	0940015011003										
3	0	0940015011004										
4	0	0940015011005										

Figura 25. Formato original datos del archivo *sdmet*

El *DataFrame* proveniente del archivo "*sdmet*" contenía 29.704 filas por 24 columnas

	ent	con	v_sel	mun	code
0	01	40007	05	001	014000705
1	01	40011	01	001	014001101
2	01	40011	03	001	014001103
3	01	40015	03	001	014001503
4	01	40016	05	001	014001605

Figura 26. Formato original datos del archivo *vivt*

El *DataFrame* proveniente de "*vivt*" contenía 14.185 filas por 5 columnas

Transformación y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Primero, se estandarizan los nombres de las columnas de los dos primeros *DataFrames*, quitando símbolos no deseados y dejando todo en letras minúsculas. Continuando con la estandarización, se reindexan ambos *DataFrames* basándose en valores únicos individuales de sus columnas. Finalmente, se juntan ambos *DataFrames* en uno.

Posteriormente, se cargan los siguientes dos *DataFrames*, aquellos relacionados a datos sociodemográficos y a vivienda, los cuales son sometidos a estandarización de sus datos tal como los anteriores. Luego se juntan estos *DataFrames* con los dos anteriores, generando un solo *DataFrame* que contiene todos los datos.

Se generan los números identificadores geográficos, basados en los municipios y entidades federativas. Luego se reemplazan los valores nulos por valores numéricos para poder realizar agregaciones de los datos. Se generan números identificadores para cada columna y este reemplaza al valor original en el *DataFrame*, por ejemplo; "Posee un trabajo o negocio" tendrá el número 1 si posee trabajo o negocio y 2 si no posee. Esta referencia será guardada por separado en otra tabla, de la forma; "posee trabajo o negocio" = 1; "No" = 2.

Posteriormente creados todos los ID's, se procede a agrupar el *DataFrame* por cada fila única. Luego se integran en el *DataFrame* los valores de ingreso para cada una de las filas.

Finalmente, se estandariza el *DataFrame* por última vez, quitando los valores repetidos e innecesarios, también validando el tipo de dato que es cada columna, dejándolas en un tipo de dato estándar para que sea leído apropiadamente por la base de datos.

Al momento en el que se elaboró esta documentación, el *DataFrame* final contenía 21.315 filas y 42 columnas. La estructura final de la tabla se presenta a continuación:

mun_id	represented_city	age	has_job_or_business	search_job_overseas	\
0	01001	14.0	15.0	2.0	NaN
1	01001	14.0	15.0	NaN	NaN
2	01001	14.0	16.0	2.0	NaN
3	01001	14.0	16.0	2.0	NaN
4	01001	14.0	16.0	2.0	NaN
	search_job_mexico	search_start_business	search_no_search	\	
0	NaN		NaN	1.0	
1	NaN		NaN	NaN	
2	NaN		NaN	1.0	
3	NaN		NaN	1.0	
4	NaN		NaN	1.0	
	search_no_knowledge	search_job_year	... work_history	\	
0	NaN	NaN	...	0.0	
1	NaN	NaN	...	0.0	
2	NaN	NaN	...	0.0	
3	NaN	NaN	...	0.0	
4	NaN	NaN	...	0.0	
	unoccupied_condition	classification_duration_unemployment	\		
0	0.0		0.0		
1	0.0		0.0		
2	0.0		0.0		
3	0.0		0.0		
4	0.0		0.0		
	underemployed_population	underemployed_classification	\		
0	0.0		0.0		
1	1.0		3.0		
2	0.0		0.0		
3	0.0		0.0		
4	0.0		0.0		
	classification_self_employed_unqualified_activities	\			
0		0.0			
1		0.0			
2		0.0			
3		0.0			
4		0.0			
	classification_formal_informal_jobs_first_activity	population	income_id	\	
0		0.0	2534	NaN	
1		1.0	4362	4.0	
2		0.0	3389	NaN	
3		0.0	4352	NaN	
4		0.0	3492	NaN	
	month_id				
0	202004				
1	202004				
2	202004				
3	202004				
4	202004				

Figura 27. Formato final tabla de datos ETOE

4. Salud

A continuación se detalla el proceso de ETL de algunos set de datos relacionados con salud. Se presentan los pipelines que procesan datos de unidades de salud y recursos de salud, los que incluyen consultorios, médicos y especialidades, enfermeras, técnicos, entre otros.

4.1. Health_establishments

Descripción General y Ejecución

El pipeline `health_establishments.py` es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos de establecimientos de salud a nivel de municipio en México. Los datos son descargados desde la plataforma de datos de la [Secretaría de Salud](#) y almacenados en un compartimiento de Google Storage privado para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas, numpy y string y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline de la siguiente forma:

```
(Bamboo) ~/data_etl/etl/health_resources/$ bamboo-cli --folder . --entry
health_establishments
```

Descarga de Datos

Una vez descargados los datos, estos son almacenados en un *DataFrame* de la librería pandas. Al momento en el que se elaboró esta documentación, el *DataFrame* inicial contaba con 41.753 filas y 11 columnas. La siguiente imagen muestra las primeras filas y algunas columnas del *DataFrame* inicial obtenido.

CLUES	CLAVE DE LA ENTIDAD	CLAVE DEL MUNICIPIO	CLAVE DE LA INSTITUCION	CLAVE TIPO ESTABLECIMIENTO	TOTAL DE CONSULTORIOS	TOTAL DE CAMAS	CODIGO POSTAL	LATITUD	LONGITUD	CLAVE ESTRATO UNIDAD
0	ASDIF000011	1	1	DIF	4	0	0	20234.0	21.8677	-102.309
1	ASDIF000023	1	4	DIF	4	0	0	99999.0	22.3399	-102.264
2	ASDIF000035	1	7	DIF	4	0	0	20400.0	22.2028	-102.284
3	ASDIF000040	1	9	DIF	4	0	0	20600.0	22.198	-102.141
4	ASDIF000052	1	3	DIF	4	0	0	20800.0	21.8178	-102.685
...
41748	ZSSSA013172	32	17	SSA	1	2	97	98608.0	22.754194	-102.535118
41749	ZSSSA013184	32	17	SSA	3	0	0	98613.0	22.750741	-102.501783
41750	ZSSSA013196	32	10	SSA	1	8	0	99000.0	23.183786	-102.864666
41751	ZSSSA013201	32	38	SSA	1	4	2	98920.0	22.277664	-101.582062
41752	ZSSSA013213	32	39	SSA	1	3	0	98422.0	23.83211	-103.026027

41753 rows × 11 columns

Figura 28. Formato original datos de unidades de salud

Extracción y Transformación

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Primero, se crea una variable denominada *mun_id* usando las columnas *CLAVE DE LA ENTIDAD* y *CLAVE DEL MUNICIPIO* y se renombran las columnas utilizables, por ejemplo; columnas como "*CLAVE DE LA INSTITUCION*" es reemplazada con el nombre "*institution_id*".

Luego de estandarizar los nombres, se procede a llenar los valores vacíos en el *dataset* y se cambian los tipos de valores de algunas columnas a un tipo de utilidad. (Ej: "*latitud*" pasa a ser una variable "*string*") Se agrega una nueva columna "*count*" para llevar un conteo de los establecimientos disponibles al nivel geográfico al cual pertenecen, para luego realizar un *grouping* de las columnas a utilizar en el *dataset*. Finalmente, se agrega una columna de fecha ("*publication_date*") relacionada a la fecha disponible del cubo.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 41.753 filas y 12 columnas. A continuación puede ser visualizada su estructura:

	mun_id	clues_id	codigo_postal	institution_id	type_id	estrato_id	total_consultorios	total_camas	count	latitud	longitud	publication_time
0	01001	ASDIF000011	20234.0	DIF	4	2	0	0	1	21.8677	-102.309	2022-03
1	01004	ASDIF000023	99999.0	DIF	4	2	0	0	1	22.3399	-102.264	2022-03
2	01007	ASDIF000035	20400.0	DIF	4	2	0	0	1	22.2028	-102.284	2022-03
3	01009	ASDIF000040	20600.0	DIF	4	2	0	0	1	22.198	-102.141	2022-03
4	01003	ASDIF000052	20800.0	DIF	4	2	0	0	1	21.8178	-102.685	2022-03
...
41748	32017	ZSSSA013172	98608.0	SSA	1	2	2	97	1	22.754194	-102.535118	2022-03
41749	32017	ZSSSA013184	98613.0	SSA	3	2	0	0	1	22.750741	-102.501783	2022-03
41750	32010	ZSSSA013196	99000.0	SSA	1	2	8	0	1	23.183786	-102.864666	2022-03
41751	32038	ZSSSA013201	98920.0	SSA	1	2	4	2	1	22.277664	-101.582062	2022-03
41752	32039	ZSSSA013213	98422.0	SSA	1	2	3	0	1	23.83211	-103.026027	2022-03

41753 rows × 12 columns

Figura 29. Formato final tabla de datos de unidades de salud

4.2. Health_resources

Descripción General y Ejecución

El pipeline `health_resources.py` es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos de recursos de salud, desde tipos de camas hasta especialidades médicas disponibles en los distintos recintos de salud. Los datos son descargados desde la plataforma de datos de la [Secretaría de Salud](#) y almacenados en un compartimiento de Google Storage privado para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas, numpy y string y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline de la siguiente forma:

```
(Python) ~/data_etl/etl/health_resources/$ python health_resources.py
```

Descarga de Datos

Una vez descargados los datos, estos son almacenados en un *DataFrame* de la librería pandas. Dadas ciertas discrepancias entre los diferentes años, la lectura de los dataset está en un ciclo *try-except* para sus distintos formatos. Al momento en el que se elaboró esta documentación, como referencia de los distintos dataset, el *DataFrame* para el año 2021 contaba con 14.215 filas y 136 columnas. La siguiente

Imagen muestra las primeras filas y algunas columnas del *DataFrame* inicial obtenido.

CLUES	CLAVE ENTIDAD	CLAVE MUNICIPIO	TIPO DE ESTABLECIMIENTO	E13	C1301	C1302	C1303	C1304	C1305	...	C1725	C1729	C1735	C1738	C1745	C1746
0	ASSSA000030	1	1	HO	15	0.0	1.0	0.0	2.0	1.0	...	8.0	1.0	1.0	0.0	0.0
1	ASSSA000042	1	1	HO	15	0.0	0.0	5.0	1.0	0.0	...	2.0	0.0	0.0	0.0	0.0
2	ASSSA000054	1	1	HO	20	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
3	ASSSA000404	1	3	HO	9	0.0	1.0	1.0	1.0	1.0	...	3.0	0.0	0.0	0.0	0.0
4	ASSSA000614	1	6	HO	5	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	0.0	0.0
...
14210	ZSSSA013160	32	39	CE	6	3.0	0.0	1.0	0.0	0.0	...	2.0	0.0	0.0	0.0	0.0
14211	ZSSSA013172	32	17	CE	2	2.0	0.0	0.0	0.0	0.0	...	2.0	0.0	1.0	0.0	0.0
14212	ZSSSA013196	32	10	CE	8	6.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
14213	ZSSSA013201	32	38	CE	4	3.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0
14214	ZSSSA013213	32	39	CE	3	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

14215 rows × 136 columns

Figura 30. Formato original datos recursos de salud

Extracción y Transformación

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Primero, se crea una variable denominada *mun_id* usando las columnas de *CLAVE DE LA ENTIDAD* y *CLAVE DEL MUNICIPIO* y se renombran las columnas con nombres utilizables, por ejemplo, columnas como "*TIPO DE ESTABLECIMIENTO*" es reemplazada con el nombre "*type_id*".

Luego, por medio de la función *melt*, se desagregan los datos por *mun_id* contra el valor de las subcategorías de los recursos del dataset.

Una vez terminado el paso *melt*, se procede a remover valores no relacionados de la nueva columna *resources_subcategories_id*, y se procede a reemplazar valores atípicos de la data y completar celdas vacías con 0.

Por último, se agrega una columna de año denominada *year* para tener una variable temporal en la data.

Posterior al proceso de transformación, siguiendo el ejemplo del año 2021, se obtiene un *DataFrame* con 1.705.800 filas y 6 columnas. A continuación puede ser visualizada su estructura:

	clues_id	mun_id	type_id	resources_subcategories_id	total	year
0	ASSSA000030	01001	HO		1301	0 2021
1	ASSSA000042	01001	HO		1301	0 2021
2	ASSSA000054	01001	HO		1301	0 2021
3	ASSSA000404	01003	HO		1301	0 2021
4	ASSSA000614	01006	HO		1301	0 2021
...
1705795	ZSSSA013160	32039	CE		1749	0 2021
1705796	ZSSSA013172	32017	CE		1749	0 2021
1705797	ZSSSA013196	32010	CE		1749	0 2021
1705798	ZSSSA013201	32038	CE		1749	0 2021
1705799	ZSSSA013213	32039	CE		1749	0 2021
1705800 rows × 6 columns						

Figura 31. Formato final tabla de datos de recursos de salud

5. Propiedad Industrial

Se presenta a continuación el detalle del proceso de ETL para tres conjuntos de datos relacionados a propiedad industrial. Cuentan con información de patentes, marcas, diseños industriales y modelos de utilidad.

5.1. impi_companies_level

Descripción General y Ejecución

El pipeline *impi_companies_level.py* es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos de titulares de patentes de México por país de origen, entre los años de 2019 a 2021. Este pipeline captura los nombres de las empresas que tienen patentes mexicanas, así como su país de origen y cantidad de las mismas. Los datos son descargados desde la plataforma de datos del [Gobierno de México \(IMPI\)](#) y almacenados en un compartimiento de Google Storage privado para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas, numpy y string y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline de la siguiente forma:

```
(Bamboo)    ~/data_etl/etl/impi/$    bamboo-cli    --folder    .    --entry
impi_companies_level
```

Descarga de Datos

Una vez descargados los datos, estos se leen en conjunto con archivos anexos relacionados a los id's de las empresas y países los cuales no son almacenados, sino que se usan localmente para crear este dataset, para luego, almacenarlos en un *DataFrame* de la librería pandas. Al momento en el que se elaboró esta documentación, el *DataFrame* inicial tiene 376 filas y 9 columnas. La siguiente imagen muestra las primeras filas y algunas columnas del *DataFrame* inicial obtenido.

	Titulares de patentes en México por país de origen (5 o más patentes) a 2021	Unnamed: 2	Unnamed: 3	Principales titulares de patentes en México por país de origen 2020	Unnamed: 6	Unnamed: 7	Principales titulares de patentes en México por país de origen 2019	Unnamed: 10	Unnamed: 11
0	Main Patent Holders in México by Country of Or...	Nan	Nan	Main Patent Holders in Mexico by Country of Or...	Nan	Nan	Main Patent Holders in México by Country of Or...	Nan	Nan
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	País / Country	Titular / Holder	Patentes / Patents	País / Country	Titular / Holder	Patentes / Patents	País / Country	Titular / Holder	Patentes / Patents
3	Alemania / Germany	BASF SE	71	Alemania / Germany	BASF SE	72	Alemania / Germany	BASF SE	54
4	Alemania / Germany	BAYER CROPS SCIENCE AKTIENGESELLSCHAFT	35	Alemania / Germany	EVONIK OPERATIONS GMBH	32	Alemania / Germany	FRAUNHOFER- GESELLSCHAFT ZUR FÖRDERUNG DER ANGE...	30
5	Alemania / Germany	EVONIK OPERATIONS GMBH	28	Alemania / Germany	FRAUNHOFER- GESELLSCHAFT ZUR FÖRDERUNG DER ANGE...	25	Alemania / Germany	BOEHRINGER INGELHEIM INTERNATIONAL GMBH	22
6	Alemania / Germany	FRAUNHOFER- GESELLSCHAFT ZUR FÖRDERUNG DER ANGE...	25	Alemania / Germany	BAYER CROPS SCIENCE AKTIENGESELLSCHAFT	21	Alemania / Germany	BAYER CROPS SCIENCE AG	20
7	Alemania / Germany	BOEHRINGER INGELHEIM INTERNATIONAL GMBH	18	Alemania / Germany	BOEHRINGER INGELHEIM INTERNATIONAL GMBH	20	Alemania / Germany	BAYER PHARMA AKTIENGESELLSCHAFT	17

Figura 32. Formato original datos titulares de patentes

Extracción y Transformación

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Primero, se realiza un paso *ffill* para ir completando las columnas hacia la derecha del *dataset* para las nuevas columnas de la tabla, las cuales son copiadas a una variable llamada *new_header*, luego se debe eliminar las antiguas y utilizar las nuevas columnas asociadas a los años disponibles por categoría. Luego, se renombran las columnas utilizables, por

ejemplo; la columna "*Main Patent Holders in Mexico by Country of Origin 2019*" es reemplazada con el nombre "*2019*". Después de esto, se remueven los valores nulos o N/A.

Una vez estandarizada la tabla original, se crea el esqueleto del cubo creando un *DataFrame* con los nombres utilizados en la data con el fin de poder realizar un *match* entre los datos por año, y agregarlos a este nuevo *DataFrame*. Para ello, iterando sobre los años disponibles para este *dataset*, se toman todas las columnas con el mismo año de referencia, se les asigna el año de su iteración y un contador que permite contar la presencia de las compañías por año, agregándolas al esqueleto creado en el paso anterior.

Por último, se realizan pasos de estandarización de la data, como renombrar las columnas del dataset como "*País / Country*" por "*country_id*", y a su vez, utilizar los archivos anexos que permitirán reemplazar los nombres de compañías y países por id's.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 918 filas y 5 columnas. A continuación puede ser visualizada su estructura:

2	country_id	holder_id	patents	year	count
3	deu	56	54	2019	1
4	deu	188	30	2019	1
5	deu	77	22	2019	1
6	deu	61	20	2019	1
7	deu	64	17	2019	1
8	deu	321	17	2019	1
9	deu	174	15	2019	1
10	deu	420	13	2019	1
11	deu	523	13	2019	1
12	deu	37	11	2019	1
13	deu	223	11	2019	1

Figura 33. Formato final tabla de datos titulares de patentes

5.2. Impi_country_level

Descripción General y Ejecución

El pipeline `impi_country_level.py` es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos de titulares de patentes de México por país de origen, entre los años de 1993 a 2021. Este pipeline captura los países de origen de los propietarios de patentes, diseños industriales, marcas y modelos de utilidad. Los datos son descargados desde la plataforma de datos del [Gobierno de México \(IMPI\)](#) y almacenados en un compartimiento de Google Storage privado para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas, numpy y string y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline de la siguiente forma:

```
(Bamboo)    ~/data_etl/etl/impi/$    bamboo-cli    --folder    .    --entry
impi_country_level
```

Descarga de Datos

Una vez descargados los datos, estos se leen en conjunto con archivos anexos relacionados a los id's de los países los cuales no son almacenados, sino que se usan localmente para crear este *dataset*, para luego, almacenarlos en un *DataFrame* de la librería pandas. El *DataFrame* inicial consiste de la lectura del mismo archivo de

origen en 8 partes, que es el formato de la misma, desagregando entre tipo y estatus del indicador.

Transformación y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Dada la naturaleza de la data, se describirán los pasos en común entre las 8 plantillas que crean este cubo.

Para el primer dataset:

- Primero, se realiza un paso *ffill* para ir completando las columnas hacia la derecha del *dataset* para las nuevas columnas de la tabla, las cuales son copiadas a una variable llamada *new_header*, eliminar las antiguas y utilizar las nuevas columnas asociadas a los años disponibles por categoría.
- Se renombran las columnas utilizables, por ejemplo; columnas como "*Solicitudes presentadas directamente /n Direct Applications*" es reemplazada con el nombre "*Direct Applications*".
- Para la columna "*year*", se agrega un *replace* de los valores trimestrales por el año al cual pertenece para agregarlos en pasos posteriores.
- Por último, se le asigna un "*status_id*" y "*type_id*" acorde al *dataset* tratado.

Para los demás dataset:

- Primero se renombran las columnas utilizables, por ejemplo; columnas como "*País/nCountry*" es reemplazada con el nombre "*country_id*".
- Luego, por medio de la función *melt*, desagregar los datos por año de referencia y valor para el *dataset*.
- Por último, similar al primer cubo se le asigna un "*status_id*" y "*type_id*" acorde al *dataset* tratado.

Una vez realizadas las operaciones necesarias en los 8 dataset, se unen los datos por medio de una función *concat*, y se realizan reemplazos de los nombres de los países y otras variables por id's.

Como últimos pasos, se procede a realizar una operación de agrupar la data (*groupBy*), formateando el *dataset* final, y agregando una última columna llamada "*nacional*" que permita filtrar entre México y el resto de países.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 45.037 filas y 6 columnas. A continuación puede ser visualizada su estructura:

	country_id	year	type_id	status_id	value	national
0	afg	1993	1	1	0	0
1	afg	1993	1	2	0	0
2	afg	1993	2	1	0	0
3	afg	1993	2	2	0	0
4	afg	1993	3	1	0	0
5	afg	1993	3	2	0	0
6	afg	1993	4	1	0	0
7	afg	1993	4	2	0	0
8	afg	1994	1	1	0	0
9	afg	1994	1	2	0	0
10	afg	1994	2	1	0	0

Figura 34. Formato final tabla de datos propiedad industrial por país

5.3. impi_state_level

Descripción General y Ejecución

El pipeline *impi_state_level.py* es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos de patentes de México por entidad federativa, entre los años de 1993 a 2021. Este pipeline captura las entidades de origen de los propietarios de patentes, diseños industriales, marcas y modelos de utilidad. Los datos son descargados desde la plataforma de datos del [Gobierno de México \(IMPI\)](#) y almacenados en un compartimiento de Google Storage privado para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas, numpy y string y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline de la siguiente forma:

```
(Bamboo) ~/data_etl/etl/impi/$ bamboo-cli --folder . --entry impi_state_level
```

Descarga de Datos

Una vez descargados los datos, estos se leen en conjunto con archivos anexos relacionados a los id's de los estados los cuales no son almacenados, sino que se usan localmente para crear este *dataset*, para luego, almacenarlos en un *DataFrame* de la librería pandas. El *DataFrame* inicial consiste de la lectura del mismo archivo de origen en 4 partes, desagregando entre tipo y estatus del indicador.

Transformación y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Dada la naturaleza de la data, se describirán los pasos en común entre las 4 plantillas que crean este cubo.

Para cada *dataset*:

- Primero, se realiza un paso *ffill* para ir completando las columnas hacia la derecha del *dataset* para las nuevas columnas de la tabla, las cuales son copiadas a una variable llamada *new_header*, eliminar las antiguas y utilizar las nuevas columnas asociadas a los años disponibles por categoría.
- Se crea un *DataFrame* vacío con los nombres de las columnas a utilizar.
- Iterando sobre cada elemento distinto entre las columnas de cada *dataset*, se crea un *subset* por año de la data, reemplazando las columnas y agregando cada iteración al *DataFrame* vacío previamente creado.
- Se renombran las columnas con nombres utilizables, por ejemplo; columnas como "*Entidad Federativa /n State*" es reemplazada con el nombre "*ent_id*".
- Para la columna "*year*", se le agrega un *replace* de los valores trimestrales por el año al cual pertenece para agregarlos en pasos posteriores, y se agrupa el *dataset* por las columnas a utilizar.
- Luego, por medio de la función *melt*, se desagregan los datos por el tipo de indicador en el *dataset*.
- Por último, se le asigna un "*status_id*" (y "*type_id*" para los *dataset* de trademarks/marcas) acorde al *dataset* que se está tratando.

Una vez realizado lo anterior, se unen los cubos por medio de una función *concat*, y se realizan reemplazos de los nombres de las entidades federativas y otras variables por id's. Se reemplazan los valores vacíos y de texto con valores 0, y se agrupa finalmente por las columnas a utilizar en el *dataset*.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 5.874 filas y 6 columnas. A continuación puede ser visualizada su estructura:

	ent_id	year	type_id	status_id	value
0	1	1993	1	1	1
1	1	1993	1	2	0
2	1	1993	2	1	12
3	1	1993	2	2	1
4	1	1993	3	1	3
5	1	1993	3	2	0
6	1	1994	1	1	1
7	1	1994	1	2	0
8	1	1994	2	1	3
9	1	1994	2	2	2
10	1	1994	3	1	0

Figura 35. Formato final tabla de datos propiedad industrial por estado

6. Comercio Exterior

A continuación se detallan los pipelines que dan origen a los diferentes cubos de comercio exterior. Los datos de comercio exterior provienen de tres fuentes: Balanza Comercial de Mercancías de México (BCMM), INEGI y BANXICO. En el caso particular de BCMM, debido a la anonimización de los datos, no fue posible agruparlos todos en un único cubo y el pipeline genera un cubo para datos a nivel nacional, otro para el nivel estatal y finalmente un cubo para el nivel municipal.

6.1. economy_foreign_trade_nat/state/mun

Descripción General y Ejecución

El pipeline *foreign_trade_pipeline* procesa los datos facilitados por la Secretaría de Economía del Gobierno de México relacionados con el Comercio Exterior. Los datos son almacenados en Google Storage para ser, posteriormente, procesados mediante un proceso de ETL (extracción, transformación, y carga).

Cabe destacar que los datos poseen diferentes niveles de agregación. Entre ellos, están diferentes periodos de tiempo y profundidad en la clasificación bajo el Sistema Armonizado (HS).

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/foreign_trade/$ python run.py
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y los datos, entregados por la Secretaría de Economía, están distribuidos en múltiples archivos que proporcionan información para diferentes períodos de tiempo, niveles geográficos y códigos de productos: los archivos se encuentran organizados por nivel nacional, estatal y municipal; cada uno de estos niveles contiene los valores organizados por código de productos a dos, cuatro y seis dígitos y estos, a su vez, se encuentran organizados en archivos anuales y mensuales. Por ejemplo, a continuación se muestra un extracto de la data inicial para un archivo de marzo 2020, a nivel municipal y desagregado por productos a 2 dígitos, donde se identifican 15.676 filas y 7 columnas. A continuación, se visualiza la estructura inicial de los datos:

Date	Product_2D	Trade_flow	Foreign_Destination_Origin		Value	Firms	unanonimized_value
2020	33	1		AFG	546.8700000000001	2611	546.87
2020	33	1		AIA	9.64	2611	9.64
2020	33	1		ALA	20.16	2611	20.16
2020	33	1		ALB	6191.88	2611	6191.88
2020	33	1		AND	47.45	2611	47.45
...
2020	97	2		PRI	9536.42	94	9536.42
2020	97	2		SWE	112.13	94	112.13
2020	97	2		TWN	11978.38	94	11978.38
2020	97	2		USA	2157933.179999997	94	2157933.18
2020	97	2		VEN	31.5	94	31.50

Figura 36. Formato original datos Comercio Exterior 2020, nacional, HS2

Limpieza y Transformación

El proceso inicia con la lectura de cada archivo y cada uno sigue el siguiente procesamiento:

- Renombre de las columnas.
- Reemplazo de valores confidenciales (C) en la columna *value* por *np.nan*.
- Localización de datos cuyo valor en la columna *value* es mayor a cero.

- Códigos de países a minúsculas.
- Creación de columnas con clasificación bajo el Sistema Armonizado que no existan en el set de datos. Dichas clasificación pueden ser al nivel: HS2, HS4, y HS6.
- Creación de columnas *month_id* y *year*, las cuales definen el periodo en estudio.
- Agregación de capítulos para todos los niveles de profundidad del Sistema Armonizado. Dicho proceso es realizado con la ayuda de la función *hs6_converter* definida en el script *util.py*.
- Creación de la columna *product_level* que indica el nivel de agregación de los datos según el archivo que contiene los valores por HS2, HS4 o HS6.
- Creación de la columna url, la cual posee la dirección de localización del archivo en Google Storage.
- Homogeneización de algunos códigos de países.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, se obtiene un *DataFrame* con 13.753 filas y 13 columnas. La estructura final de la tabla se presenta a continuación:

hs2_id	flow_id	partner_country	value	firms	unanonimized_value	hs6_id	hs4_id	month_id	year	nat_id	product_level	url
633	1	afg	547	2611	547	0	0	0	2020	mex	2	National/HS_2D/Annual/2D_nat_annual20
633	1	aia	10	2611	10	0	0	0	2020	mex	2	National/HS_2D/Annual/2D_nat_annual20
633	1	ala	20	2611	20	0	0	0	2020	mex	2	National/HS_2D/Annual/2D_nat_annual20
633	1	alb	6192	2611	6192	0	0	0	2020	mex	2	National/HS_2D/Annual/2D_nat_annual20
633	1	and	47	2611	47	0	0	0	2020	mex	2	National/HS_2D/Annual/2D_nat_annual20
...
2197	2	pri	9536	94	9536	0	0	0	2020	mex	2	National/HS_2D/Annual/2D_nat_annual20
2197	2	swe	112	94	112	0	0	0	2020	mex	2	National/HS_2D/Annual/2D_nat_annual20
2197	2	twn	11978	94	11978	0	0	0	2020	mex	2	National/HS_2D/Annual/2D_nat_annual20
2197	2	usa	2157933	94	2157933	0	0	0	2020	mex	2	National/HS_2D/Annual/2D_nat_annual20
2197	2	ven	32	94	32	0	0	0	2020	mex	2	National/HS_2D/Annual/2D_nat_annual20

Figura 37. Formato final tabla de datos Comercio Exterior 2020, nacional, HS2

Al finalizar el proceso, todas las tablas con datos nacionales se unen para generar el cubo *economy_foreign_trade_nat*, las tablas con datos estatales generan el cubo *economy_foreign_trade_ent* y, finalmente, las tablas con datos municipales generan el cubo *economy_foreign_trade_mun*.

6.2. banxico_trade_flow

Descripción General y Ejecución

El pipeline `trade_flow.py` procesa datos a nivel nacional de las exportaciones e importaciones por país de destino u origen, respectivamente, con desagregación mensual. Los datos fueron descargados desde el [Banco de México](#) (BANXICO) y almacenados en Google Storage, desde donde se realiza la descarga de datos. Posteriormente, se realiza un proceso de limpieza y transformación para ingestar los datos en una base de datos de *Clickhouse*.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

(Python) `~/data_etl/etl/banxico/$ python trade_flow.py`

(Bamboo-cli) `~/data_etl/etl/banxico/$ bamboo-cli --folder . --entry trade_flow`

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv; y, en cuanto a su estructura, posee 516 filas y 347 columnas. A continuación se presenta su estructura inicial:

	Título	Periodo disponible	Periodicidad	Cifra	...	01/07/2021	01/08/2021	01/09/2021
Exportación total (Incluye maquila), Total	Ene 1993 - Sep 2021	Mensual	Flujos	...	40887733.0	40313434.0	41680024.0	
Exportación total (Incluye maquila), Total, Am...	Ene 1993 - Sep 2021	Mensual	Flujos	...	36032048.0	35232188.0	37430615.0	
Exportación total (Incluye maquila), Total, Am...	Ene 1993 - Sep 2021	Mensual	Flujos	...	34100168.0	33256485.0	35341564.0	
Exportación total (Incluye maquila), Total, Am...	Ene 1993 - Sep 2021	Mensual	Flujos	...	33011181.0	32139740.0	34474668.0	
Exportación total (Incluye maquila), Total, Am...	Ene 1993 - Sep 2021	Mensual	Flujos	...	1088987.0	1116745.0	866896.0	

Figura 38. Formato original datos de comercio exterior de Banxico

Lectura y Transformación

En esta etapa se sigue el siguiente proceso:

- Lectura de datos de exportaciones e importaciones. Se almacenan los datos en un mismo *DataFrame* pero se identifican mediante la adición de la variable *trade_flow_id*, la cual define si se trata de una exportación (valor igual a 2) o importación (valor igual a 1).
- Transformación de nombres de columnas a minúscula, reemplazando espacios por guión bajo para facilitar el proceso de ETL.
- Eliminación de columnas redundantes.
- Transformación a formato *tidy*. Para ello, se utiliza la función *melt* de Pandas.
- Transformación de la variable *month_id*, que define el periodo en estudio, a formato: YYYYMM.
- Eliminación de columnas que posean agregaciones totales.
- Reemplazo de nombres de países por su código ISO3. Se hace un llamado a la tabla dimensión *dim_shared_country* almacenada en Clickhouse, que contiene dichos valores.
- Eliminación de filas que posean agregaciones por continente.
- Multiplicación de valores en la columna *trade_value* por mil. Esto ya que los valores se encuentran en miles de dólares.
- Agrupación de datos para eliminar duplicados en columnas clave.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 161460 filas y 4 columnas. A continuación puede ser visualizada su estructura:

iso3	trade_flow_id	month_id	trade_value
abw	1	199301	0
abw	1	199302	0
abw	1	199303	0
abw	1	199304	0
abw	1	199305	0

Figura 39. Formato final tabla de datos de comercio exterior de Banxico

6.3. inegi_foreign_trade_country

Descripción General y Ejecución

El pipeline inegi_foreign_trade_country.py procesa datos de la balanza comercial de mercancías por países y zonas geográficas facilitados por el Instituto Nacional de Estadística y Geografía (INEGI). Dichos datos presentan una desagregación mensual. Los datos se almacenan en Google Storage, desde donde son descargados para ejecutar el proceso de limpieza y transformación antes de ser ingestados en una base de datos de *Clickhouse*.

Para ejecutar el proceso de ETL, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

```
(Bamboo-cli) ~/data_etl/$ bamboo-cli --folder etl/inegi_foreign_trade --entry inegi_foreign_trade_country
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xls; y, en cuanto a su estructura, posee 359.136 filas y 3 columnas. A continuación se presenta su estructura inicial:

Bélgica	Sector externo> Balanza comercial de mercancías por países y zonas geográficas> Importaciones FOB> Petroleras> Europa> Unión europea (UE)>
Ruta temática	
Periodo	Dato
1993/01	44
1993/02	76
1993/03	28
1993/04	25
1993/05	55
1993/06	149
1993/07	148
1993/08	91
1993/09	24

Figura 40. Formato original datos de comercio exterior por países, INEGI

Lectura y Transformación

En esta etapa se sigue el siguiente proceso:

- Se leen los datos de cada hoja del archivo .xls. Además, se filtran los valores en la columna *Periodo* para conservar los valores que detallan un periodo temporal.
- Transformación de datos a formato *tidy*. Para ello, se utiliza la función *melt* de Pandas.
- Creación de la variable *month_id*. Se construye a partir de reemplazar el carácter / en la columna *Periodo*.
- Creación de la variable *trade_flow_id*, la cual toma el valor 1 si se detalla una importación, y 2 si se detalla una exportación.
- Creación de la variable *petroleum*, la cual toma el valor 1 para mercancías petroleras y 2 en cualquiera de los otros casos.
- Eliminación de filas que posean el string *Total*. Esto se realiza para no considerar agregaciones totales.
- Posterior a la lectura de la tabla dimensión *dim_shared_country* existente en Clickhouse, se utiliza la función merge de Pandas para agregar una columna con los códigos ISO3.
- Selección de columnas a utilizar: *month_id*, *iso3*, *trade_flow_id*, *petroleum*, *value*.
- Eliminación de valores confidenciales.

- Transformación de valores a dólares (existen originalmente en miles de dólares).

Posterior al proceso de transformación, se obtiene un *DataFrame* con 315.332 filas y 5 columnas. A continuación puede ser visualizada su estructura:

month_id	iso3	trade_flow_id	petroleum	value
199301	reu	2	1	0.0
199301	prt	2	1	9899000.0
199301	brb	2	1	0.0
199301	phl	2	1	0.0
199301	col	2	1	439000.0

Figura 41. Formato final tabla de datos de comercio exterior por países, INEGI

6.4. inegi_foreign_trade_product

Descripción General y Ejecución

El pipeline `inegi_foreign_trade_product.py` procesa datos de exportaciones e importaciones de productos del Sistema Armonizado de Designación y Codificación de Mercancías, facilitados por el Instituto Nacional de Estadística y Geografía (INEGI). Dichos datos presentan una desagregación mensual. Los datos se almacenan en Google Storage, desde donde son descargados para ejecutar el proceso de limpieza y transformación antes de ser ingestados en una base de datos de Clickhouse.

Para ejecutar el proceso de ETL, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

```
(Bamboo-cli) ~/data_etl/$ bamboo-cli --folder etl/inegi_foreign_trade --entry
inegi_foreign_trade_product
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería bamboo-lib se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xls; y, en cuanto a su estructura, posee 514.624 filas y 3 columnas. A continuación se presenta su estructura inicial:

Periodo		name	value
1993/01	Sector externo > Balanza comercial de mercancí...	619493	
1993/01	Sector externo > Balanza comercial de mercancí...	0	
1993/01	Sector externo > Balanza comercial de mercancí...	9899	
1993/01	Sector externo > Balanza comercial de mercancí...	0	
1993/01	Sector externo > Balanza <u>comercial</u> de mercancí...	0	

Figura 42. Formato original datos de comercio exterior por productos, INEGI

Lectura y Transformación

En esta etapa se sigue el siguiente proceso:

- Se leen los datos de cada hoja del archivo .xls. Además, se filtran los valores en la columna *Periodo* para conservar los valores que detallan un periodo temporal.
- Transformación de datos a formato *tidy*. Para ello, se utiliza la función *melt* de Pandas.
- Creación de la variable *month_id*. Se construye a partir de reemplazar el carácter / en la columna *Periodo*.
- Creación de la variable *trade_flow_id*, la cual toma el valor 1 si se detalla una importación, y 2 si se detalla una exportación.
- Creación de las variables *section* y *chapter*, creados a partir de los nombres entregados a las columnas en el archivo fuente.
- Creación de una lista con capítulos únicos relacionados a importaciones, y otra con relación a exportaciones.
- Creación de una lista con capítulos únicos relacionados a importaciones, y otra con relación a exportaciones, pero que no posean agregaciones totales.
- Utilizando las listas anteriores se procede a llenar filas con capítulos faltantes.
- Creación de la variable *hs2_id*, la cual se construye a partir de la agregación de las variables *section_id* y *chapter_id*.
- Selección de columnas a utilizar: *month_id*, *trade_flow_id*, *hs2_id*, *value*.
- Eliminación de valores confidenciales.

- Transformación de valores a dólares (existen originalmente en miles de dólares).

Posterior al proceso de transformación, se obtiene un *DataFrame* con 66.652 filas y 4 columnas. A continuación puede ser visualizada su estructura:

month_id	trade_flow_id	hs2_id	value
199301	2	101	3.119200e+07
199301	2	416	1.531000e+06
199301	2	422	1.591900e+07
199301	2	631	4.273000e+06
199301	2	105	3.690000e+05

Figura 43. Formato final tabla de datos de comercio exterior por productos, INEGI

6.5. inegi_foreign_trade_state

Descripción General y Ejecución

El pipeline `inegi_foreign_trade_state.py` procesa datos de exportaciones de productos del Sistema Armonizado de Designación y Codificación de Mercancías trimestralmente por estados. Los datos son obtenidos del Instituto Nacional de Estadística y Geografía ([INEGI](#)) y almacenados en Google Storage, desde donde son descargados para ejecutar el proceso de limpieza y transformación antes de ser ingestados en una base de datos de *Clickhouse*.

Para ejecutar el proceso de ETL, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

```
(Bamboo-cli) ~/data_etl/$ bamboo-cli --folder etl/inegi_foreign_trade --entry
inegi_foreign_trade_state
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar los datos desde Google Storage. El archivo se encuentra en formato .xlsx y, en cuanto a su estructura, al momento de elaborar esta documentación el archivo contenía 34 filas y 62 columnas de información válida. A continuación se presenta su estructura inicial:

1 Exportaciones trimestrales por Entidad Federativa	2 Serie trimestral I 2007 - I 2022	3 Miles de dólares	4									
5 Entidad Federativa	2007 R/				2008				2009			
	I	II	III	IV	I	II	III	IV	I	II	III	IV
7 Exportaciones totales*	51,250.250	57,823.366	62,611.002	66,125.122	62,193.381	70,478.739	70,100.001	55,195.658	42,154.792	46,254.415	51,204.506	58,620.411
8 Aguascalientes	1,075.959	979.202	1,187.010	1,147.671	975.433	1,168.812	1,284.780	1,027.868	794.218	899.577	1,064.971	1,192.342
9 Baja California	6,816.770	7,187.457	8,233.045	9,621.403	7,219.128	8,310.408	9,439.323	8,020.054	5,812.647	6,401.182	6,847.085	7,680.915
10 Baja California Sur	38,426	28,242	40,754	41,575	37,951	60,856	44,073	40,875	37,264	39,443	32,150	56,419
11 Campeche	5,646.622	6,851.415	7,737.800	8,815.714	8,397.478	9,834.249	9,201.868	4,404.216	3,121.263	4,209.629	4,776.541	5,485.591
12 Coahuila de Zaragoza	3,465.690	3,995.774	4,729.560	5,278.610	5,391.736	5,882.664	5,209.648	5,272.989	3,176.183	2,701.303	3,528.094	4,499.343
13 Colima	11,697	21,330	24,624	21,900	17,457	23,452	23,413	22,286	17,127	16,217	23,017	24,983
14 Chiapas	195,968	227,377	234,577	241,087	225,400	267,096	267,053	141,000	160,045	230,187	230,170	260,100
15 Chihuahua	6,271.050	7,331.152	7,262.263	7,266.033	6,873.342	7,655.237	7,434.055	6,011.597	5,238.910	5,715.059	6,407.813	7,363.84
16 Ciudad de México	682,527	663,713	704,805	662,171	637,332	777,669	799,600	655,469	563,557	581,530	620,854	612,097
17 Durango	172,672	193,467	257,582	254,982	183,070	250,709	287,712	253,412	195,736	205,028	206,491	222,191
18 Guanajuato	1,273,193	1,634,064	1,718,640	1,798,354	1,565,124	1,653,208	1,618,213	1,583,379	1,146,153	1,097,594	1,518,231	1,922,036
19 Guerrero	21,815	28,604	33,312	58,387	91,267	55,245	56,291	45,528	75,340	112,402	118,437	158,685
20 Hidalgo	271,969	277,738	401,734	349,192	474,750	589,231	606,425	374,225	294,058	303,454	327,587	344,406
21 Jalisco	3,234,231	3,539,803	4,058,060	3,613,880	3,227,737	3,984,306	4,371,851	3,776,904	3,523,588	3,795,682	3,752,439	3,987,879
22 México	1,906,915	2,067,288	2,262,213	2,406,721	2,441,340	2,669,844	2,454,713	2,344,001	1,740,505	1,693,939	1,853,281	2,360,628
23 Michoacán de Ocampo	104,111	69,292	64,603	119,281	204,714	259,838	440,637	197,526	308,594	154,459	147,028	195,875
24 Morelos	487,183	476,545	549,734	534,461	476,025	507,006	528,311	448,686	308,604	332,122	406,966	513,992
25 Nayarit	6,068	7,252	12,664	11,118	9,357	7,680	20,273	8,414	10,239	20,448	20,281	8,527
26 Nuevo León	4,594,891	5,069,272	5,046,093	5,091,231	5,451,895	5,697,596	5,730,690	4,885,206	3,895,540	4,031,254	4,504,946	4,885,259
27 Oaxaca	146,397	150,572	176,096	160,346	308,754	380,767	444,165	315,200	207,018	242,541	285,014	308,810
28 Puebla	1,515,507	1,796,900	2,168,141	2,305,765	2,056,475	2,422,075	2,462,427	2,264,498	1,425,175	1,547,403	1,549,731	2,152,952
29 Querétaro	752,823	876,067	887,394	894,223	1,028,695	1,081,024	1,050,093	908,117	715,128	799,895	929,203	1,026,897
30 Quintana Roo	8,856	10,457	11,974	9,280	15,360	11,662	7,651	7,630	8,978	7,400	9,864	9,470
31 San Luis Potosí	1,104,292	1,223,635	1,201,496	1,258,845	1,181,880	1,221,155	1,173,843	952,065	779,014	808,420	929,447	1,042,224
32 Sinaloa	75,100	80,127	84,010	85,591	90,824	93,746	128,439	67,915	53,438	53,700	50,228	50,793
33 Sonora	3,101,094	3,302,125	3,359,478	3,217,688	3,102,754	3,246,048	3,040,319	2,645,994	1,645,107	2,372,461	2,476,347	2,886,343
34 Tabasco	1,629,260	1,979,667	2,192,791	2,470,877	2,628,694	3,067,874	2,895,153	1,399,068	1,233,014	1,665,130	1,895,206	2,175,233
35 Tamaulipas	4,771,448	5,686,332	5,863,435	6,195,215	5,691,336	6,651,352	6,286,229	5,152,351	4,027,304	4,417,867	4,735,915	5,140,000
36 Tlaxcala	103,593	159,207	201,939	191,415	182,196	201,521	241,904	179,199	144,149	155,996	166,993	177,587
37 Veracruz de Ignacio de la Llave	1,057,297	1,079,049	1,053,885	1,078,844	1,280,161	1,581,964	1,521,788	1,048,294	907,942	1,019,049	1,037,215	1,142,592
38 Yucatán	219,738	265,660	318,705	342,446	252,337	259,034	294,683	283,101	210,176	239,541	300,826	282,901
39 Zacatecas	427,526	526,780	580,222	580,042	675,147	624,671	626,442	494,091	358,667	412,357	448,452	414,670

* Las exportaciones totales corresponden a la sumatoria de los subsectores 211 Extracción de petróleo y gas, 212 Minería de minerales metálicos y no metálicos, excepto petróleo y gas y 31-33 Industrias manufactureras. La información del sector 11 Agricultura, cría y explotación de animales.

La suma de los parciales puede no coincidir con el total debido al redondeo.

R/ Resultados revisados a partir de la fecha que se indica, lo anterior debido a la actualización de la metodología de cálculo derivada de la incorporación del Registro Estadístico de Negocios de México (RENEM) y la Encuesta Mensual de la Industria Manufacturera como fuentes estadísticas.

Figura 44. Formato original datos de comercio exterior por estados, INEGI

Lectura y Transformación

Luego de la lectura del archivo, este pasa por un proceso de transformación para obtener el formato deseado. Este proceso consta de los siguientes pasos:

- Se eliminan filas con texto explicativo al inicio y al final de la hoja de datos.
- Se reemplaza el valor *2007 R/* por *2007*.
- Se utiliza el método *ffill* para completar cada columna con el año correspondiente.
- Se une la fila de años con la fila de trimestres para renombrar las columnas.
- Se eliminan filas innecesarias (aquellas con años y trimestres que fueron utilizadas para renombrar las columnas), así como las filas con valores *NaN*. Se reinindexa la tabla.
- Utilizando la función *melt* de Pandas, se transforma la tabla a formato Tidy, dejando solo 3 columnas: *entidad*, *periodo* y *valor*.
- Por medio de la función *replace*, se modifican los valores de la columna *time_id*, cambiando valores como *2007_I* a *2007I*.
- Se renombran las columnas y se cambian los formatos de datos correctos.
- Utilizando una dimensión compartida, se cambia el nombre de las entidades federativas por sus correspondientes ids.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 1920 filas y 3 columnas (las filas irán aumentando a medida que se agreguen nuevos datos al archivo original). A continuación puede ser visualizada su estructura de salida:

ent_id	time_id	value
1	20071	1075959000
2	20071	6816770000
3	20071	38426000
4	20071	5646622000
5	20071	3465690000
6	20071	11697000
...
28	20214	9580143000
29	20214	500024000
30	20214	1920694000
31	20214	338522000
32	20214	1110248000

Figura 45. Formato final tabla de datos de comercio exterior por estados, INEGI

6.6. trade_i_baci_a_12

Descripción General y Ejecución

El pipeline baci_pipeline.py procesa datos sobre flujos comerciales (exportaciones e importaciones) de más de 5000 productos y 200 países. Los datos son obtenidos desde CEPII, quienes construyen la base de datos a partir de los datos que cada país comunica directamente a la División de Estadística de las Naciones Unidas (Comtrade), y, son almacenados en Google Storage. Desde ahí se descargan para ser limpiados y transformados antes de ser ingestados a una base de datos de *Clickhouse*.

Para ejecutar el proceso de ETL, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

(Python) `~/data_etl/etl/baci/$ python baci_pipeline.py`

(Bamboo-cli) `~/data_etl/etl/baci/$ bamboo-cli --folder . --entry baci_pipeline --db_connector='clickhouse-database' --hs_code='12' --year=<year>`

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería bamboo-lib se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv; y, en cuanto a su estructura, posee 10.646.209 filas y 6 columnas. A continuación se presenta su estructura inicial:

year	exporter	importer	hs_original_id	value	quantity
2019	4	31	70310	5.757	22.000
2019	4	31	80211	2.453	0.196
2019	4	31	80620	0.179	0.028
2019	4	31	80711	5.886	37.830
2019	4	31	81310	0.115	0.014

Figura 46. Formato original datos de flujos comerciales mundiales

Lectura y Transformación

En esta etapa se sigue el siguiente proceso:

- Descarga de archivos desde GCP. Para ello, es importante el parámetro `hs_code`, ya que define la revisión que será fuente de los datos. En este caso se utiliza el **Sistema Armonizado con revisión 2012**.
- Extracción de archivos a partir de la clase `ExtractStep`. Dicha clase permite identificar el número de archivos .zip que coinciden con los parámetros entregados, y limita su extracción. Esto ya que sólo un archivo contiene la data .csv requerida.
- Dado que los archivos .csv vienen comprimidos en una carpeta .zip, se utiliza la clase de `bamboo_lib` llamada `UnzipToFolderStep` para su descompresión. Dicha clase descomprime el archivo .zip y crea un directorio bajo el mismo nombre dentro del directorio temporal `temp`.
- Iterando sobre cada archivo .csv, comienza el proceso de transformación.
- Multiplicación de la variable `value` por 1000. Esto se realiza para llevar sus valores a dólares (existen originalmente en miles de dólares).
- Relleno de la columna `hs_original_id` a 6 dígitos.
- Creación de la variable `hs_master_id`. Dicha variable se construye a partir de aplicar la función `hs6_converter` a la variable `hs_original_id`.
- Creación del diccionario `id_num_iso3_map`. Se construye a partir de las variables `id_num` e `iso3` de la tabla dimensión `dim_shared_country` existente en Clickhouse. Esto se realiza con motivo de reemplazar el id de cada país por su código ISO3.
- Limpieza del diccionario `id_num_iso3_map`. Existen valores en la variable `id_num` que tienen múltiples entradas separadas por un string (|). Dado ello, se separan y se concatenan como filas.

- Reemplazo de los *ids* en las columnas *exporter* e *importer* por los códigos ISO3 (países).
- Creación de la variable *hs_revision*. Dicha variable se construye a partir de la revisión del Sistema Armonizado utilizado en la construcción del conjunto de datos.
- Selección de columnas a utilizar.
- Creación de la columna *version*. Dicha variable define la fecha en la cual se ingestaron los datos.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 10.646.209 filas y 9 columnas. A continuación puede ser visualizada su estructura:

year	exporter	importer	hs_master_id	hs_revision	hs_original_id	value	quantity	version
2019	afg	aze	2070310	4	070310	5757.0	22.000	2021-12-22 09:32:58.345104
2019	afg	aze	2080211	4	080211	2453.0	0.196	2021-12-22 09:32:58.345104
2019	afg	aze	2080620	4	080620	179.0	0.028	2021-12-22 09:32:58.345104
2019	afg	aze	2080711	4	080711	5886.0	37.830	2021-12-22 09:32:58.345104
2019	afg	aze	2081310	4	081310	115.0	0.014	2021-12-22 09:32:58.345104

Figura 47. Formato final tabla de datos de flujos comerciales mundiales

7. Inversión Extranjera Directa (IED)

A continuación se detallan los pasos de cada pipeline asociado a datos de Inversión Extranjera Directa (IED). Nuevamente, debido a los niveles de anonimización, no fue posible procesar todos los datos en un único pipeline y se generó un pipeline para cada corte de datos de interés. Cabe mencionar que los cubos que tienen desagregación por industrias, utilizan el Sistema de Clasificación Industrial de América del Norte, SCIAN 2013.

7.1. fdi_10_year_country/country_investment/investment

Descripción General y Ejecución

El pipeline *fdi_10.py* procesa los datos facilitados por la [Secretaría de Economía](#) del Gobierno de México que toman relación con Inversión Extranjera Directa (IED). En específico, procesa datos con relación a la inversión total anual por país de origen, inversión total anual por país de origen y tipo de inversión e inversión total anual por tipo de inversión. Al respecto, son tres los cubos generados a partir del pipeline. Por otro lado, los datos fuente son almacenados en Google Storage.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/foreign_direct_investment/$ python fdi_10.py
```

Descarga de datos

Utilizando el módulo de descarga que facilita la librería bamboo-lib se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y su estructura inicial varía de acuerdo al cubo procesado. A continuación, se visualiza la estructura inicial de los datos. Al momento de escribir esta documentación, para el caso de la inversión total anual por país de origen el archivo contenía 1.152 filas y 5 columnas.

Año	País	Monto	Reuento	Monto C
1999	Alemania	685,711369	266	685,7
1999	Argentina	3,638073	91	3,6
1999	Australia	6,945795	21	6,9
...
2021	Suecia	-31,64759634	13	-31,6
2021	Suiza	95,06947608	30	95,1
2021	Taiwán	0,61283825	7	0,6

Figura 48. Formato original datos IED por año y país

El archivo de inversión total anual por país de origen y tipo de inversión contaba con 2.752 filas y 6 columnas.

Año	País	Inversión	Monto	Reuento	Monto C
1999	Alemania	Cuentas entre compañías	160,518939	70	160,5
1999	Alemania	Nuevas inversiones	308,403768	198	308,4
1999	Alemania	Reinversión de utilidades	216,788662	43	216,8
...
2021	Suiza	Cuentas entre compañías	41,7244782	20	41,7
2021	Suiza	Nuevas inversiones	22,0718243	9	22,1
2021	Suiza	Reinversión de utilidades	31,2731736	9	31,3

Figura 49. Formato original datos IED por año, país y tipo de inversión

Por último, el archivo con datos de inversión total anual por tipo de inversión contaba con 24 filas y 10 columnas.

Año de materialización	Monto cuentas entre compañías	Monto nuevas inversiones	Monto reinversión de utilidades	Reuento cuentas entre compañías	Reuento nuevas inversiones	Reuento reinversión de utilidades	Monto C cuentas entre compañías	Monto C nuevas inversiones	Monto C reinversión de utilidades
1999	4986,21495	6606,924368	2342,739387	2548	6570	586	4.986,2	6.606,9	2.342,7
2000	5637,606776	8704,326873	3905,431695	2551	6332	533	5.637,6	8.704,3	3.905,4
2001	3046,299236	23110,76694	3899,172799	2400	6357	446	3.046,3	23.110,8	3.899,2
...
2019	3016,147825	13228,76851	17998,99319	2336	6229	1007	3.016,1	13.228,8	17.999,0
2020	5098,840477	6622,010814	16064,86588	2441	4159	786	5.098,8	6.622,0	16.064,9
2021	2629,401193	2209,418013	7025,211257	1546	1102	423	2.629,4	2.209,4	7.025,2
Total general	144636,4303	268530,6185	205484,5198	10081	114783	4050	144.636,4	268.530,6	205.484,5

Figura 50. Formato original datos IED por año y tipo de inversión

Limpieza y Transformación

Como se ha mencionado anteriormente, dado que existen diferentes fuentes y salidas de datos, los procesos de ETL difieren entre sí. A continuación se presenta, para cada archivo, el proceso de ETL implementado que genera cada cubo de datos.

a) Inversión total anual por país de origen

- Normalización de nombres de columnas. Se utiliza el módulo de Pandas *norm* para transformar cada *caracter* en minúscula, y también, el módulo *replace* para reemplazar *caracteres*.
- Renombre de columnas.
- Selección de filas que no corresponden a agregados totales.
- Eliminación de valores confidenciales.
- Reemplazo en columna *country* por el código iso3 de cada país.
- Transformación del *dtype* en las columnas *year*, *count*, y *value_c*, a enteros.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, obtenemos un *DataFrame* con 1.074 filas y 4 columnas. La estructura final de la tabla se presenta a continuación:

	year	country	count	value_c
0	1999.0	deu	266.0	685.711369
1	1999.0	arg	91.0	3.638073
2	1999.0	aus	21.0	6.945795
3	1999.0	aut	23.0	12.524428
4	1999.0	bel	31.0	-49.370033
...
1143	2021.0	cze	3.0	0.015951
1146	2021.0	swe	13.0	-31.647596
1147	2021.0	che	30.0	95.069476
1148	2021.0	twn	7.0	0.612838
1150	2021.0	ven	17.0	1.146261

Figura 51. Formato final tabla de datos IED por año y país

b) Inversión total anual por país de origen y tipo de inversión

- Normalización de nombres de columnas. Se utiliza el módulo de Pandas *norm* para transformar cada *caracter* en minúscula, y también, el módulo *replace* para reemplazar *caracteres*.
- Renombre de columnas.
- Selección de filas que no corresponden a agregados totales.
- Eliminación de valores confidenciales.
- Reemplazo en columna *country* por el código iso3 de cada país.
- Transformación del *dtype* en las columnas *year*, *count*, y *value_c*, a enteros.
- Reemplazo en columna *investment_type* for el id de cada tipo de inversión.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, obtenemos un *DataFrame* con 2.159 filas y 5 columnas. A continuación puede ser visualizada su estructura de salida:

	<i>year</i>	<i>country</i>	<i>investment_type</i>	<i>count</i>	<i>value_c</i>
0	1999.0	deu		1	70.0 160.518939
1	1999.0	deu		2	198.0 308.403768
2	1999.0	deu		3	43.0 216.788662
3	1999.0	arg		1	4.0 -2.594482
4	1999.0	arg		2	87.0 6.211129
...
2744	2021.0	che		1	20.0 41.724478
2745	2021.0	che		2	9.0 22.071824
2746	2021.0	che		3	9.0 31.273174
2747	2021.0	twn		1	7.0 0.612838

Figura 52. Formato final tabla de datos IED por año, país y tipo de inversión

c) Inversión total anual por tipo de inversión

- Normalización de nombres de columnas. Se utiliza el módulo de Pandas *norm* para transformar cada *caracter* en minúscula, y también, el módulo *replace* para reemplazar *caracteres*.
- Renombre de columnas por medio de una lista.
- Selección de filas que no corresponden a agregados totales.

- Eliminación de columnas que no serán utilizadas: *value_between_companies*, *value_new_investments*, y *value_re_investments*.
- Transformación del *DataFrame* a formato tidy. Para ello, se crea una nueva columna denominada *investment_type*, la cual posee valores enteros para referirse a los tipos de inversiones (*between_companies*, *new_investments*, y *re_investments*) las cuales anteriormente estaban separadas en columnas individuales.
- Reemplazo en columna *investment_type* por sus id's.

En cuanto a la salida del proceso de ETL, se obtiene un *DataFrame* con 69 filas y 4 columnas. La estructura final de la tabla se presenta a continuación:

	year	count	value_c	investment_type
0	1999.0	2548.0	4986.214950	1
1	2000.0	2551.0	5637.606776	1
2	2001.0	2400.0	3046.299236	1
3	2002.0	2314.0	5868.759710	1
4	2003.0	2285.0	7095.183763	1
...
18	2017.0	696.0	11951.846494	3
19	2018.0	820.0	13245.576041	3
20	2019.0	1007.0	17998.993194	3
21	2020.0	786.0	16064.865878	3
22	2021.0	423.0	7025.211257	3

Figura 53. Formato final tabla de datos IED por año y tipo de inversión

7.2. fdi_2_state_investment

Descripción General y Ejecución

El pipeline *fdi_2.py* procesa los datos facilitados por la [Secretaría de Economía](#) del Gobierno de México que toman relación con Inversión Extranjera Directa (IED). En específico, procesa datos con relación a los tipos de inversión por trimestre y entidades federativas. Los datos son almacenados en Google Storage para ser, posteriormente, procesados mediante un proceso de ETL (extracción, transformación, y carga).

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/foreign_direct_investment/$ python fdi_2.py
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y su estructura inicial se compone de 2.848 filas y 12 columnas. A continuación, se visualiza la estructura inicial de los datos:

	Entidad federativa	Año	Trimestre	Monto Cuentas entre compañías	Monto Nuevas inversiones	Monto Reinversión de utilidades	Recuento Cuentas entre compañías	Recuento Nuevas inversiones	Recuento Reinversión de utilidades	Monto C cuentas entre compañías	Monto C nuevas inversiones	Monto C reinversión de utilidades
0	Aguascalientes	1999	1	20.645964	7.720677	8.860865	58	46	41	20.645964	7.720677	8.860865
1	Aguascalientes	1999	2	24.018523	-0.168652	6.299668	64	36	22	24.018523	-0.168652	6.299668
2	Aguascalientes	1999	3	16.640456	696.548325	2.034928	67	51	18	16.640456	696.548325	2.034928
3	Aguascalientes	1999	4	15.316704	18.769218	2.159396	67	53	23	15.316704	18.769218	2.159396
4	Aguascalientes	2000	1	16.988831	129.370891	61.172054	70	62	53	16.988831	129.370891	61.172054
...
2843	Zacatecas	2020	1	-168.749753	21.751855	168.507596	45	7	44	-168.749753	21.751855	168.507596
2844	Zacatecas	2020	2	19.600942	1.405455	4.525801	32	6	6	19.600942	1.405455	4.525801
2845	Zacatecas	2020	3	-182.763024	0.413678	25.387107	34	4	5	-182.763024	0.413678	25.387107
2846	Zacatecas	2020	4	-270.995383	-1.690795	-22.942449	38	5	8	-270.995383	-1.690795	-22.942449

Figura 54. Formato original datos IED por trimestre, estado y tipo de inversión

Limpieza y Transformación

Esta etapa sigue el siguiente proceso:

- Normalización de nombres de columnas. Se utiliza el módulo de Pandas *norm* para transformar cada *caracter* en minúscula, y también, el módulo *replace* para reemplazar *caracteres*.
- Eliminación de Entidades Federativas que hagan mención al total acumulado por Entidad. Esto se realiza dado que es redundante en los datos.
- Procesamiento de la columna *year* y *quarter_id* para generar correctamente la columna *quarter_id*.
- Eliminación de columnas que no serán utilizadas: *value_between_companies*, *value_new_investments* y *value_re_investments*.
- Transformación del *DataFrame* a formato tidy. Para ello, se crea una nueva columna denominada *investment_type*, la cual posee valores enteros para referirse a los tipos de inversiones (*between_companies*, *new_investments*, y *re_investments*) las cuales anteriormente estaban separadas en columnas individuales.
- Eliminación de datos por estado que posean una cantidad de registros confidenciales mayor a 10%.
- Eliminación de valores confidenciales. Dichos valores se almacenaban en la columna *value_c*.

En cuanto a la salida del proceso de ETL, se obtiene un *DataFrame* con 8.465 filas y 6 columnas. La estructura final de la tabla se presenta a continuación:

	ent_id	year	quarter_id	count	value_c	investment_type
0	1	1999	19991	58	20.645964	1
1	1	1999	19992	64	24.018523	1
2	1	1999	19993	67	16.640456	1
3	1	1999	19994	67	15.316704	1
4	1	2000	20001	70	16.988831	1
...
2843	32	2020	20201	44	168.507596	3
2844	32	2020	20202	6	4.525801	3
2845	32	2020	20203	5	25.387107	3
2846	32	2020	20204	8	-22.942449	3
2847	32	2021	20211	21	60.264757	3

Figura 55. Formato final tabla de datos IED por trimestre, estado y tipo de inversión

7.3. Fdi_3_country_origin

Descripción General y Ejecución

El presente pipeline procesa los datos facilitados por la [Secretaría de Economía](#) del Gobierno de México que toman relación con Inversión Extranjera Directa (IED). En específico, procesa datos relacionados a los orígenes de las inversiones. Los datos son almacenados en Google Storage para ser, posteriormente, procesados mediante un proceso de ETL (extracción, transformación, y carga).

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/foreign_direct_investment/$ python fdi_3.py
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería bamboo-lib se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx; y su estructura inicial se compone de 17.873 filas y 6 columnas. A continuación, se visualiza la estructura inicial de los datos:

	Entidad federativa	Año	País	Monto	Recuento	Monto C
0	Aguascalientes	1999.0	Alemania	2.219975	9	2.219975
1	Aguascalientes	1999.0	Argentina	-0.000000	2	C
2	Aguascalientes	1999.0	Bélgica	-0.115000	2	C
3	Aguascalientes	1999.0	Brasil	-0.010000	2	C
4	Aguascalientes	1999.0	Canadá	0.000175	3	0.000175
...
17868	Zacatecas	2021.0	Japón	-1.700700	2	C
17869	Zacatecas	2021.0	Otros países	307.708870	5	307.70887
17870	Zacatecas	2021.0	Países Bajos	-0.007040	2	C
17871	Zacatecas	2021.0	Reino Unido de la Gran Bretaña e Irlanda del N...	-0.000700	1	C
17872	Total general	NaN	Nan	618651.568548	117518	618651.568548

Figura 56. Formato original datos IED por país, año y estado

Limpieza y Transformación

Esta etapa sigue el siguiente proceso:

- Normalización de nombres de columnas. Se utiliza el módulo de Pandas *norm* para transformar cada *caracter* en minúscula, y también, el módulo *replace* para reemplazar *caracteres*.
- Eliminación de Entidades Federativas que hagan mención al total acumulado por Entidad. Esto se realiza dado que es redundante en los datos.
- Reemplazo de Entidades Federativas por sus id.
- Renombre de columnas a: *ent_id*, *year*, *country*, *value*, *count*, y *value_c*.
- Eliminación de valores confidenciales. Dichos valores se almacenaban en la columna *value_c*.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, obtenemos un *DataFrame* con 10.058 filas y 5 columnas. La estructura final de la tabla se presenta a continuación:

ent_id	year	country	count	value_c
1	1999.0	deu	9	2.219975
1	1999.0	can	3	0.000175
1	1999.0	esp	11	12.304624
1	1999.0	usa	170	87.008375
1	1999.0	fra	12	0.463365
...
32	2020.0	che	3	-0.286696
32	2021.0	can	3	0.829693
32	2021.0	esp	4	6.086107
32	2021.0	usa	17	-110.344698
32	2021.0	xxa	5	307.708870

Figura 57. Formato final tabla de datos IED por país, año y estado

7.4. fdi_4_investment_type

Descripción General y Ejecución

El pipeline *fdi_4.py* procesa los datos facilitados por la [Secretaría de Economía](#) del Gobierno de México que toman relación con Inversión Extranjera Directa (IED). En específico, procesa datos relacionados a los orígenes de las inversiones, considerando el tipo de inversión realizada de manera anual y según el estado receptor. Los datos son almacenados en Google Storage para ser, posteriormente, procesados mediante un proceso de ETL (extracción, transformación, y carga).

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/foreign_direct_investment/$ python fdi_4.py
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería bamboo-lib se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx; y su estructura inicial se compone de 33.907 filas y 7 columnas. A continuación, se visualiza la estructura inicial de los datos:

Entidad federativa	Año	País	Inversión	Monto	Reuento	Monto C
Aguascalientes	1999.0	Alemania	Cuentas entre compañías	1.886538	5	1.886538
Aguascalientes	1999.0	Alemania	Nuevas inversiones	0.182343	3	0.182343
Aguascalientes	1999.0	Alemania	Reinversión de utilidades	[REDACTED]	2	C
Aguascalientes	1999.0	Argentina	Nuevas inversiones	[REDACTED]	2	C
Aguascalientes	1999.0	Bélgica	Cuentas entre compañías	[REDACTED]	1	C
...
Zacatecas	2021.0	Otros países	Reinversión de utilidades	46.401340	3	46.40134
Zacatecas	2021.0	Paises Bajos	Cuentas entre compañías	[REDACTED]	2	C
Zacatecas	2021.0	Paises Bajos	Reinversión de utilidades	[REDACTED]	1	C
Zacatecas	2021.0	Reino Unido de la Gran Bretaña e Irlanda del N...	Reinversión de utilidades	[REDACTED]	1	C
Total general	NaN	NaN	NaN	618651.568548	117518	618651.568548

Figura 58. Formato original datos IED por año, país, tipo y entidad federativa

Limpieza y Transformación

Esta etapa sigue el siguiente proceso:

- Normalización de nombres de columnas. Se utiliza el módulo de Pandas `norm` para transformar cada `caracter` en minúscula, y también, el módulo `replace` para reemplazar caracteres.
- Eliminación de Entidades Federativas que hagan mención al total acumulado por Entidad. Esto se realiza dado que es redundante en los datos.
- Reemplazo de Entidades Federativas por sus id, entre otros reemplazos.
- Renombre de columnas a: `ent_id`, `year`, `country`, `investment_type`, `value`, `count`, y `value_c`.
- Eliminación de valores confidenciales. Dichos valores se almacenaban en la columna `value_c`.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, obtenemos un `DataFrame` con 15.828 filas y 6 columnas. A continuación puede ser visualizada su estructura de salida:

ent_id	year	country	investment_type	count	value_c
1	1999.0	deu		1	5
1	1999.0	deu		2	3
1	1999.0	can		2	3
1	1999.0	esp		2	10
1	1999.0	usa		1	65
...
32	2021.0	esp		3	4
32	2021.0	usa		1	11
32	2021.0	usa		3	9
32	2021.0	xxa		1	4
32	2021.0	xxa		3	3

Figura 59. Formato original tabla de datos IED por año, país, tipo y entidad federativa

7.5. fdi_9_quarter/quarter_investment/_year/_year_investment

Descripción General y Ejecución

El pipeline *fdi_9.py* procesa los datos facilitados por la [Secretaría de Economía](#) del Gobierno de México que toman relación con Inversión Extranjera Directa (IED). En específico, procesa datos con relación a la inversión total por año, inversión total trimestral, inversión total por año y tipo de inversión e inversión total por trimestre y tipo de inversión, todas las anteriores a nivel nacional. Dado ello, son cuatro los cubos generados a partir de este pipeline. En cuanto a los datos, estos son almacenados en Google Storage.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/foreign_direct_investment/$ python fdi_9.py
```

Descarga de datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y su estructura inicial varía de acuerdo al archivo procesado.

El archivo que contiene los datos de inversión total anual contaba con 24 filas y 4 columnas y su estructura inicial es como se muestra en la imagen siguiente:

Año	Monto	Recuento	Monto C
1999	13935.878705	8885	13935.878705
2000	18247.365344	8665	18247.365344
2001	30056.238976	8523	30056.238976
2002	24098.547790	9016	24098.547790
2003	18270.689675	9288	18270.689675

Figura 60. Formato original datos IED anuales a nivel nacional

El archivo con datos de inversión total trimestral contenía 90 filas y 5 columnas y su estructura es la siguiente:

Año	Trimestre	Monto	Recuento	Monto C
1999	1.0	3571.645864	3889	3571.645864
1999	2.0	3395.880570	3952	3395.880570
1999	3.0	3028.447643	3336	3028.447643
1999	4.0	3939.904628	3567	3939.904628
2000	1.0	4599.300171	3631	4599.300171
...
2020	2.0	7329.820611	2403	7329.820611
2020	3.0	1330.397128	2694	1330.397128
2020	4.0	2374.609352	2821	2374.609352
2021	1.0	11864.030463	2764	11864.030463
Total general	NaN	618651.568548	117518	618651.568548

Figura 61. Formato original datos IED trimestrales a nivel nacional

El archivo con datos de inversión total anual por tipo de inversión contenía 24 filas y 10 columnas y su estructura es la siguiente:

Año	Monto cuentas entre compañías	Monto nuevas inversiones	Monto reinversión de utilidades	Recuento cuentas entre compañías	Recuento nuevas inversiones	Recuento reinversión de utilidades	Monto C cuentas entre compañías	Monto C nuevas inversiones	Monto C reinversión de utilidades
1999	4986.214950	6606.924368	2342.739387	2548	6570	586	4986.214950	6606.924368	2342.739387
2000	5637.606776	8704.326873	3905.431695	2551	6332	533	5637.606776	8704.326873	3905.431695
2001	3046.299236	23110.766941	3899.172799	2400	6357	446	3046.299236	23110.766941	3899.172799
2002	5868.759710	15638.660235	2591.127845	2314	6869	313	5868.759710	15638.660235	2591.127845

Figura 62. Formato original datos IED anuales por tipo de inversión a nivel nacional

Finalmente, el archivo con datos de inversión total trimestral por tipo de inversión contenía 90 filas y 11 columnas y su estructura es la siguiente:

Año	Trimestre	Monto cuentas entre compañías	Monto nuevas inversiones	Monto reinversión de utilidades	Recuento cuentas entre compañías	Recuento nuevas inversiones	Recuento reinversión de utilidades	Monto C cuentas entre compañías	Monto C nuevas inversiones	Monto C reinversión de utilidades
1999	1.0	1455.588937	1068.627613	1047.429314	1786	1968	380	1455.588937	1068.627613	1047.429314
1999	2.0	1228.553505	1549.699793	617.627272	1824	2074	270	1228.553505	1549.699793	617.627272
1999	3.0	527.945413	2244.442135	256.060095	1822	1467	221	527.945413	2244.442135	256.060095
1999	4.0	1774.127095	1744.154827	421.622706	1849	1714	216	1774.127095	1744.154827	421.622706

Figura 63. Formato original datos IED trimestral por tipo de inversión a nivel nacional

Limpieza y Transformación

Como se ha mencionado anteriormente, dado que existen diferentes fuentes y salidas de datos, los procesos de ETL difieren entre sí. A continuación se presenta, para cada archivo, el proceso de ETL implementado que genera cada cubo de datos.

a) Inversión total anual

- Normalización de nombres de columnas. Se utiliza el módulo de Pandas `norm` para transformar cada `caracter` en minúscula, y también, el módulo `replace` para reemplazar `caracteres`.
- Renombre de columnas.
- Selección de filas que no corresponden a agregados totales.
- Transformación del `dtype` en las columnas `count` y `year` a enteros.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, obtenemos un `DataFrame` con 23 filas y 3 columnas. A continuación pueden ser visualizadas las primeras filas de su estructura de salida:

year	count	value_c
1999	8885	13935.878705
2000	8665	18247.365344
2001	8523	30056.238976
2002	9016	24098.547790
2003	9288	18270.689675
2004	10120	25032.076843

Figura 64. Formato final tabla de datos IED anuales a nivel nacional

b) Inversión total trimestral

- Normalización de nombres de columnas. Se utiliza el módulo de Pandas *norm* para transformar cada *caracter* en minúscula, y también, el módulo *replace* para reemplazar *caracteres*.
- Renombre de columnas.
- Selección de filas que no corresponden a agregados totales.
- Creación de la columna *quarter_id* que posee los id temporales a nivel trimestral.
- Eliminación de columnas innecesarias.
- Transformación del *dtype* en las columnas *count* y *quarter_id* a enteros.

En cuanto a la salida del proceso de ETL, se obtiene un *DataFrame* con 89 filas y 3 columnas. A continuación puede ser visualizada su estructura de salida:

count	value_c	quarter_id
3889	3571.645864	19991
3952	3395.880570	19992
3336	3028.447643	19993
3567	3939.904628	19994
3631	4599.300171	20001
...
3252	16750.890078	20201
2403	7329.820611	20202
2694	1330.397128	20203
2821	2374.609352	20204
2764	11864.030463	20211

Figura 65. Formato final tabla de datos IED trimestrales a nivel nacional

c) Inversión total por año y tipo de inversión

- Normalización de nombres de columnas. Se utiliza el módulo de Pandas *norm* para transformar cada *caracter* en minúscula, y también, el módulo *replace* para reemplazar *caracteres*.
- Renombre de columnas por medio de una lista.
- Selección de filas que no corresponden a agregados totales.
- Eliminación de columnas que no serán utilizadas: *value_between_companies*, *value_new_investments*, y *value_re_investments*.
- Transformación del *DataFrame* a formato *tidy*. Para ello, se crea una nueva columna denominada *investment_type*, la cual posee valores enteros para referirse a inversiones: *between_companies*, *new_investments*, y *re_investments*; las cuales anteriormente estaban separadas en columnas individuales.
- Reemplazo en columna *investment_type* por sus id's.

En cuanto a la salida del proceso de ETL, se obtiene un *DataFrame* con 69 filas y 4 columnas. A continuación puede ser visualizada su estructura de salida:

year	count	value_c	investment_type
1999.0	2548.0	4986.214950	1
2000.0	2551.0	5637.606776	1
2001.0	2400.0	3046.299236	1
2002.0	2314.0	5868.759710	1
2003.0	2285.0	7095.183763	1
...
2017.0	696.0	11951.846494	3
2018.0	820.0	13245.576041	3
2019.0	1007.0	17998.993194	3
2020.0	786.0	16064.865878	3
2021.0	423.0	7025.211257	3

Figura 66. Formato final tabla de datos IED anuales por tipo de inversión a nivel nacional

d) Inversión total por trimestre y tipo de inversión

- Normalización de nombres de columnas. Se utiliza el módulo de Pandas *norm* para transformar cada *caracter* en minúscula, y también, el módulo *replace* para reemplazar *caracteres*.
- Renombre de columnas por medio de una lista.
- Selección de filas que no corresponden a agregados totales.
- Creación de la columna *quarter_id* que posee los id temporales a nivel trimestral.
- Eliminación de columnas que no serán utilizadas: *year*, *quarter*, *value_between_companies*, *value_new_investments* y *value_re_investments*.
- Transformación del *DataFrame* a formato *tidy*. Para ello, se crea una nueva columna denominada *investment_type*, la cual posee valores enteros para referirse a los tipos de inversiones (*between_companies*, *new_investments* y *re_investments*) las cuales anteriormente estaban separadas en columnas individuales.
- Reemplazo en columna *investment_type* por sus id's.

En cuanto a la salida del proceso de ETL, se obtiene un *DataFrame* con 267 filas y 4 columnas. La estructura final de la tabla se presenta a continuación:

quarter_id	count	value_c	investment_type
19991.0	1786.0	1455.588937	1
19992.0	1824.0	1228.553505	1
19993.0	1822.0	527.945413	1
19994.0	1849.0	1774.127095	1
20001.0	1873.0	980.191516	1
...
20201.0	645.0	14336.759885	3
20202.0	139.0	815.770815	3
20203.0	126.0	801.913478	3
20204.0	99.0	110.421700	3
20211.0	423.0	7025.211257	3

Figura 67. Formato datos IED trimestral por tipo de inversión a nivel nacional

7.6. fdi_quarter_* y fdi_year_*

Descripción General y Ejecución

El pipeline `fdi_additional_tables.py` procesa los datos facilitados por la [Secretaría de Economía](#) del Gobierno de México que toman relación con Inversión Extranjera Directa (IED). En específico, procesa datos relacionados con la inversión total trimestral y anual para cada sector, subsector e industria económica y tipo de inversión, además de inversión total anual por industria y país de origen y la inversión total anual por entidad federativa. De acuerdo a lo anterior, se generan 6 cubos a partir del pipeline. Los problemas de anonimización impiden juntar todos los datos en un único cubo, dado que a medida que se agregan los niveles inferiores se pierden valores en los niveles superiores. Por ellos es necesario procesar los datos por separado y generar 6 cubos diferentes:

- 1) `fdi_quarter_industry_investment`: IED trimestral por sector/subsector/industria y tipo de inversión.
- 2) `fdi_year_industry_country`: IED anual por sector/subsector/industria y países.
- 3) `fdi_year_state_industry`: IED anual por sector/subsector/industria y entidad federativa.
- 4) `fdi_year_industry`: IED anual por sector/subsector/industria.
- 5) `fdi_quarter_industry`: IED trimestral por sector/subsector/industria.
- 6) `fdi_year_investment_industry`: IED anual por sector/subsector/industria y tipo de inversión.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Además, dado que existen diferentes fuentes y combinaciones de parámetros involucrados, se creó el script `run_fdi_additional_tables.py` que almacena los parámetros utilizados para el procesamiento de los datos que generan los seis cubos mencionados.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/foreign_direct_investment/$ python run_fdi_additional_tables.py
```

Descarga de datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx, cada archivo contiene diferentes hojas de datos y su estructura inicial varía de acuerdo al archivo procesado.

Al momento de escribir esta documentación, el archivo con datos de inversión total trimestral por sector industrial y tipo de inversión contaba con 6.141 filas y 7 columnas. Cabe destacar que un archivo de las mismas características existe para el subsector industrial y rama industrial.

Sector	Año	Trimestre	Inversión	Monto	Reuento	Monto C
11 Agricultura, cría y explotación de animales...	1999	1	Cuentas entre compañías	0.435078	4	0.435078
21 Minería	1999	1	Cuentas entre compañías	6.720601	26	6.720601
22 Generación, transmisión y distribución de e...	1999	1	Cuentas entre compañías	6.893001	7	6.893001
23 Construcción	1999	1	Cuentas entre compañías	29.682373	18	29.682373
31 Industrias manufactureras	1999	1	Cuentas entre compañías	590.494225	270	590.494225
...

Figura 68. Formato original datos IED por sector, trimestre y tipo de inversión

El archivo con datos de inversión total anual por industria y país de origen se estructuraba en 11.566 filas y 6 columnas.

Año de materialización	País de Origen_otros	Sector	Suma de Monto en millones	Recuento distinto de Expediente	Monto C
1999	Alemania	11 Agricultura, cría y explotación de animales...	0.004700	2	C
1999	Alemania	23 Construcción	61.988283	7	61.988283
1999	Alemania	31 Industrias manufactureras	2.723321	11	2.723321
1999	Alemania	32 Industrias manufactureras	199.682383	33	199.682383
1999	Alemania	33 Industrias manufactureras	180.533753	91	180.533753
...

Figura 69. Formato original datos IED por sector, año y país

El archivo con datos de inversión total anual por estados e industrias contaba con 12.908 filas y 6 columnas.

Año de materialización	Entidad federativa	Sector	Suma de Monto en millones	Recuento distinto de Expediente	Monto C
1999	Aguascalientes	11 Agricultura, cría y explotación de animales...	0.000000	1	C
1999	Aguascalientes	22 Generación, transmisión y distribución de e...	0.385442	4	0.385442
1999	Aguascalientes	23 Construcción	0.002578	3	0.002578
1999	Aguascalientes	31 Industrias manufactureras	37.647984	43	37.647984
1999	Aguascalientes	32 Industrias manufactureras	21.530571	24	21.530571
...

Figura 70. Formato original datos IED por sector, año y estado

El archivo con datos de inversión total por año e industria se estructura en 529 filas y 5 columnas.

Año	Sector	Monto	Recuento	Monto C
1999	11 Agricultura, cría y explotación de animales...	87.791614	63	87.791614
1999	21 Minería	229.078375	104	229.078375
1999	22 Generación, transmisión y distribución de e...	345.558844	39	345.558844
1999	23 Construcción	206.980556	158	206.980556
1999	31 Industrias manufactureras	1642.160354	724	1642.160354
...

Figura 71. Formato original datos IED por año e industria



El archivo con datos de inversión total trimestral por industrias contaba de 2.047 filas y 6 columnas.

	Sector	Año	Trimestre	Monto	Recuento	Monto C
11 Agricultura, cría y explotación de animales...	1999	1	28.431735	19	28.431735	
21 Minería	1999	1	120.914406	51	120.914406	
22 Generación, transmisión y distribución de e...	1999	1	12.942499	16	12.942499	
23 Construcción	1999	1	80.416861	61	80.416861	
31 Industrias manufactureras	1999	1	791.493028	399	791.493028	
...

Figura 72. Formato original datos IED por industria y trimestre

Finalmente, el archivo con datos de inversión total anual por tipo de inversión e industrias se estructuraba en 1.587 filas y 6 columnas.

	Sector	Inversión	Año	Monto	Recuento	Monto C
11 Agricultura, cría y explotación de animales...	Cuentas entre compañías	1999	2.301394	8	2.301394	
21 Minería	Cuentas entre compañías	1999	19.418630	33	19.41863	
22 Generación, transmisión y distribución de e...	Cuentas entre compañías	1999	32.162537	11	32.162537	
23 Construcción	Cuentas entre compañías	1999	67.081704	29	67.081704	
31 Industrias manufactureras	Cuentas entre compañías	1999	794.708157	400	794.708157	
...

Figura 73. Formato original datos IED por industria, año y tipo de inversión

Limpieza y Transformación

Como se ha mencionado anteriormente, dado que existen diferentes fuentes y salidas de datos, los procesos de ETL difieren entre sí. Por ello, a continuación se presenta, para cada cubo generado, el proceso de ETL implementado.

a) Inversión total trimestral por sector/subsector/rama industrial y tipo de inversión

- Renombre de columnas.
- Eliminación de filas que contienen totales. Esto dado que es información redundante.
- Reformulación de variable *quarter_id*, donde su id corresponde a una combinación de la variable *year* y *quarter_id*.
- Creación de columnas faltantes: *sector_id*, *subsector_id* e *industry_group_id*. Dado que existe un archivo para cada nivel industrial, en cada archivo se agregan columnas con ceros para los niveles faltantes.
- Reemplazo del nombre de sectores industriales por sus id.
- Eliminación de datos por tipo de inversión que posean un número de valores confidenciales mayor a 10%.
- Reemplazo de tipos de inversión por su id.
- Transformación de columnas de datos a tipo *float*.

En cuanto a la salida del proceso de ETL y considerando el archivo con valores para sectores industriales, se obtiene un *DataFrame* con 4.049 filas y 8 columnas. A continuación puede ser visualizada su estructura de salida:

<i>sector_id</i>	<i>quarter_id</i>	<i>investment_type</i>	<i>value</i>	<i>count</i>	<i>value_c</i>	<i>subsector_id</i>	<i>industry_group_id</i>
11	19991.0	1.0	0.435078	4.0	0.43507799999999996	0.0	0.0
21	19991.0	1.0	6.720601	26.0	6.7206009999999985	0.0	0.0
22	19991.0	1.0	6.893001	7.0	6.893001000000003	0.0	0.0
23	19991.0	1.0	29.682373	18.0	29.682372999999995	0.0	0.0
31-33	19991.0	1.0	590.494225	270.0	590.4942249999998	0.0	0.0
...
31-33	20211.0	3.0	808.532526	66.0	808.5325263500009	0.0	0.0
31-33	20211.0	3.0	1685.157338	157.0	1685.1573375899982	0.0	0.0
43	20211.0	3.0	703.078245	54.0	703.0782450699996	0.0	0.0
55	20211.0	3.0	0.000000	0.0	0	0.0	0.0
93	20211.0	3.0	0.000000	0.0	0	0.0	0.0

Figura 74. Formato final tabla de datos IED por sector, trimestre y tipo de inversión

b) Inversión total anual por industria y país de origen

- Renombrar columnas
- Selección de claves primarias. Dichas claves serán guardadas en una lista llamada *pk_id*.
- Eliminación de filas cuya clave primaria sea nula.
- Creación de columnas faltantes: *sector_id*, *subsector_id* e *industry_group_id*. Dado que existe un archivo para cada nivel industrial, en cada archivo se agregan columnas con ceros para los niveles faltantes.
- Reemplazo del nombre de sectores industriales por sus id.
- Reemplazo de países por sus *ids*.
- Selección de filas con valores no confidenciales.
- Transformación de columnas de datos a tipo *float*.

En cuanto a la salida del proceso de ETL, y considerando el archivo con valores para sectores industriales, se obtiene un *DataFrame* con 6.040 filas y 8 columnas. A continuación puede ser visualizada su estructura de salida:

year	country_id	sector_id	value	count	value_c	subsector_id	industry_group_id
1999.0	deu	23	61.988283	7.0	61.988283	0.0	0.0
1999.0	deu	31-33	2.723321	11.0	2.7233209999999994	0.0	0.0
1999.0	deu	31-33	199.682383	33.0	199.68238300000002	0.0	0.0
1999.0	deu	31-33	180.533753	91.0	180.53375299999993	0.0	0.0
1999.0	deu	43	188.638393	57.0	188.63839300000003	0.0	0.0
...
2021.0	che	31-33	-0.297268	4.0	-0.29726779000000025	0.0	0.0
2021.0	che	43	-0.297364	4.0	-0.29736410999999885	0.0	0.0
2021.0	che	48-49	16.277043	5.0	16.2770426	0.0	0.0
2021.0	twn	31-33	5.010208	5.0	5.01020845	0.0	0.0
2021.0	ven	72	1.146261	17.0	1.14626135	0.0	0.0

Figura 75. Formato final tabla de datos IED por sector, año y país

c) Inversión total anual por estado e industria

- Renombrar columnas.
- Selección de claves primarias. Dichas claves serán guardadas en una lista llamada *pk_id*.
- Eliminación de filas cuya clave primaria sea nula.
- Reemplazo de entidades federativas por sus ids.
- Selección de filas con valores no confidenciales.
- Creación de columnas faltantes: *sector_id*, *subsector_id* e *industry_group_id*. Dado que existe un archivo para cada nivel industrial, en cada archivo se agregan columnas con ceros para los niveles faltantes.
- Reemplazo del nombre de sectores industriales por sus id.
- Reemplazo de países por sus id.
- Transformación de columnas de datos a tipo *float*.

En cuanto a la salida del proceso de ETL, y considerando el archivo con valores para sectores industriales, se obtiene un *DataFrame* con 10.682 filas y 8 columnas. A continuación puede ser visualizada su estructura de salida:

year	ent_id	sector_id	value	count	value_c	subsector_id	industry_group_id
1999.0	1	22	0.385442	4.0	0.385442	0.0	0.0
1999.0	1	23	0.002578	3.0	0.002578	0.0	0.0
1999.0	1	31-33	37.647984	43.0	37.647984	0.0	0.0
1999.0	1	31-33	21.530571	24.0	21.530571	0.0	0.0
1999.0	1	31-33	737.684528	59.0	737.684528	0.0	0.0
...
2021.0	32	31-33	-0.063735	5.0	-0.063735	0.0	0.0
2021.0	32	31-33	-4.742262	4.0	-4.742262	0.0	0.0
2021.0	32	46	4.597508	3.0	4.597508	0.0	0.0
2021.0	32	52	6.857873	7.0	6.857873	0.0	0.0
2021.0	32	54	0.234930	3.0	0.234930	0.0	0.0

Figura 76. Formato final tabla de datos IED por sector, año y estado

d) Inversión total por año y sector industrial

- Renombrar columnas.
- Selección de claves primarias. Dichas claves serán guardadas en una lista llamada *pk_id*.
- Eliminación de filas cuya clave primaria sea nula.
- Eliminación de datos que agrupados por clave primaria posean un número de valores confidenciales mayor a 10%.
- Creación de columnas faltantes: *sector_id*, *subsector_id* e *industry_group_id*. Dado que existe un archivo para cada nivel industrial, en cada archivo se agregan columnas con ceros para los niveles faltantes.
- Reemplazo del nombre de sectores industriales por sus id.
- Selección de valores en columna *value_c* que no posean valores confidenciales o igual a *false*.
- Transformación de columnas de datos a tipo *float*.

En cuanto a la salida del proceso de ETL, y considerando el archivo con valores para sectores industriales, se obtiene un *DataFrame* con 528 filas y 7 columnas. A continuación puede ser visualizada su estructura de salida:

year	sector_id	value	count	value_c	subsector_id	industry_group_id
1999.0	11	87.791614	63.0	87.791614	0.0	0.0
1999.0	21	229.078375	104.0	229.078375	0.0	0.0
1999.0	22	345.558844	39.0	345.558844	0.0	0.0
1999.0	23	206.980556	158.0	206.980556	0.0	0.0
1999.0	31-33	1642.160354	724.0	1642.160354	0.0	0.0
...
2021.0	62	4.110380	5.0	4.110380	0.0	0.0
2021.0	71	42.076211	11.0	42.076211	0.0	0.0
2021.0	72	375.273216	906.0	375.273216	0.0	0.0
2021.0	81	16.236747	6.0	16.236747	0.0	0.0
2021.0	93	0.000000	0.0	0.000000	0.0	0.0

Figura 77. Formato final tabla de datos IED por año e industria

e) Inversión total trimestral por sector industrial

- Renombrar columnas.
- Selección de claves primarias. Dichas claves serán guardadas en una lista llamada *pk_id*.
- Eliminación de filas cuya clave primaria sea igual a *Total general*.
- Reformulación de columna *quarter_id*. Se agrega el año previo a la referencia trimestral.
- Eliminación de datos que agrupados por clave primaria posean un número de valores confidenciales mayor a 10%.
- Creación de columnas faltantes: *sector_id*, *subsector_id* e *industry_group_id*. Dado que existe un archivo para cada nivel industrial, en cada archivo se agregan columnas con ceros para los niveles faltantes.
- Reemplazo del nombre de sectores industriales por sus id.
- Selección de valores en columna *value_c* que no posean valores confidenciales o igual a *false*.
- Transformación de columnas de datos a tipo *float*.

En cuanto a la salida del proceso de ETL, y considerando el archivo con valores para sectores industriales, se obtiene un *DataFrame* con 1.952 filas y 7 columnas. A continuación puede ser visualizada su estructura de salida:

sector_id	quarter_id	value	count	value_c	subsector_id	industry_group_id
11	19991.0	28.431735	19.0	28.431735	0.0	0.0
21	19991.0	120.914406	51.0	120.914406	0.0	0.0
22	19991.0	12.942499	16.0	12.942499	0.0	0.0
23	19991.0	80.416861	61.0	80.416861	0.0	0.0
31-33	19991.0	791.493028	399.0	791.493028	0.0	0.0
...

Figura 78. Formato final tabla de datos IED por industria y trimestre

f) Inversión total anual por tipo de inversión y sector industrial

- Renombrar columnas.
- Selección de claves primarias. Dichas claves serán guardadas en una lista llamada *pk_id*.
- Eliminación de filas cuya clave primaria sea igual a *Total general*.
- Eliminación de datos que agrupados por tipo de inversión posean un número de valores confidenciales mayor a 10%.
- Creación de columnas faltantes: *sector_id*, *subsector_id* e *industry_group_id*. Dado que existe un archivo para cada nivel industrial, en cada archivo se agregan columnas con ceros para los niveles faltantes.
- Reemplazo del nombre de sectores industriales por sus id.
- Selección de valores no confidenciales.
- Reemplazo de tipos de inversión por sus ids.
- Transformación de columnas de datos a tipo *float*.

En cuanto a la salida del proceso de ETL, y considerando el archivo con valores para sectores industriales, se obtiene un *DataFrame* con 1352 filas y 7 columnas. La estructura final de la tabla se presenta a continuación:

sector_id	investment_type	year	value	count	subsector_id	industry_group_id
11	1.0	1999.0	2.301394	8.0	0.0	0.0
21	1.0	1999.0	19.418630	33.0	0.0	0.0
22	1.0	1999.0	32.162537	11.0	0.0	0.0
23	1.0	1999.0	67.081704	29.0	0.0	0.0
31-33	1.0	1999.0	794.708157	400.0	0.0	0.0
...

Figura 79. Formato final tabla de datos IED por industria, año y tipo de inversión

8. Asociación Nacional de Universidades e Instituciones de Educación Superior (ANUIES)

A continuación se detalla el procesamiento aplicado a los datos provenientes de la Asociación Nacional de Universidades e Instituciones de Educación Superior (ANUIES). En este caso, se procesan los datos en tres pipelines diferentes que agrupan los datos según matriculados, nivel de estudios y origen de los estudiantes.

8.1. Anuies_enrollment

Descripción General y Ejecución

El pipeline *anuies_enrollment.py* procesa los datos facilitados por la Asociación Nacional de Universidades e Instituciones de Educación Superior (ANUIES) que toman relación con los procesos de inscripción de estudiantes. Los datos son descargados desde la plataforma de la [Asociación Nacional de Universidades e Instituciones de Educación Superior](#), y son almacenados en Google Storage para su posterior procesamiento. Cabe destacar que los datos vienen desagregados a nivel municipal.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de Clickhouse, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
(bamboo-cli) ~/data_etl/etl/anuies/$ python enrollment_pipeline.py
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y se identifican 66.593 filas y 76 columnas. A continuación, se visualiza la estructura inicial de los datos:

ENTIDAD	MUNICIPIO	CVE	CAMPO UNITARIO	...	Suma de PNI-SA	Suma de PNI-EUR	Suma de PNI-AAO
AGUASCALIENTES		001	11000	...	0	0	0
	NaN	NaN	NaN	...	0	0	0
	NaN	NaN	NaN	...	0	0	0
	NaN	NaN	NaN	...	0	0	0
	NaN	NaN	NaN	...	0	0	0

	NaN	NaN	NaN	...	0	0	0
	NaN	NaN	Total	84100	0	0	0
	NaN	Total	056	NaN	0	0	0
Total ZACATECAS		NaN	NaN	...	2	0	0
Total general		NaN	NaN	...	2042	118	68

Figura 80. Formato original datos ANUIES - matriculados

Extracción y Transformación

Este paso del pipeline sigue la secuencia mencionada a continuación:

- Renombre de columnas
- Se completan celdas *NaN* con los valores correspondientes haciendo uso de la función *ffill()*.
- Reemplazo de las entidades federativas (*ent_id*) por sus ids.
- Limpieza de celdas que incluyen el string *Total* acompañando a valores numéricos. Dicho string es eliminado para facilitar agregaciones y futuros análisis. Así también, se eliminan puntos (.) y dos puntos (:) para mejorar el formato del set de datos.
- Reemplazo de ids para los municipios. Para ello, se hace una query a Clickhouse con motivo de traer la dimensión *dim_shared_geography_mun* que contiene dichos ids.
- Uso de la función *melt* de pandas para llevar el set de datos a un formato *tidy*.
- Reemplazo de los valores en las columnas *sex*, *type*, y *age* por medio de diccionarios definidos en el script *static.py*.

- Se añaden los ids para las columnas *program* y *campus*. Esto se realiza posterior a la lectura de archivos almacenados en Google Cloud Storage y que contienen dicha información.
- Selección de variables a utilizar: *mun_id*, *campus_id*, *program*, *type*, *sex*, *value* y *age*.
- Procesamiento del nombre de instituciones, campus, carreras y programas.
- Creación de la columna *year* la cual contiene información con respecto al periodo de tiempo en estudio.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 87.580 filas y 8 columnas. La estructura final de la tabla se presenta a continuación:

<i>mun_id</i>	<i>campus_id</i>	<i>program</i>	<i>type</i>	<i>sex</i>	<i>value</i>	<i>age</i>	<i>year</i>
1001.0	1600001.0	11500021.0	13.0	1.0	2.0	5.0	2017
1001.0	1100001.0	31100173.0	13.0	1.0	2.0	5.0	2017
1001.0	1600001.0	42100257.0	13.0	1.0	1.0	5.0	2017
1001.0	1600001.0	41200055.0	13.0	1.0	1.0	5.0	2017
1001.0	1100004.0	41400021.0	13.0	1.0	1.0	5.0	2017
...
32056.0	347400024.0	95000012.0	13.0	2.0	1.0	32.0	2017
32056.0	347400030.0	92204002.0	12.0	2.0	21.0	32.0	2017
32056.0	347400030.0	92207004.0	12.0	2.0	6.0	32.0	2017
32056.0	5900013.0	94200015.0	13.0	2.0	3.0	32.0	2017
32056.0	350000001.0	104100059.0	12.0	2.0	1.0	32.0	2017

Figura 81. Formato final tabla de datos ANUIES - matriculados

8.2. Anuies_origin

Descripción General y Ejecución

El pipeline `origin_pipeline.py` procesa los datos facilitados por la Asociación Nacional de Universidades e Instituciones de Educación Superior (ANUIES) que toman relación con la entidad federativa de origen de los estudiantes. Los datos son descargados desde la plataforma de la [Asociación Nacional de Universidades e Instituciones de Educación Superior](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
(bamboo-cli) ~/data_etl/etl/anuies/$ python origin_pipeline.py
```

Descarga de datos

Utilizando el módulo de descarga que facilita la librería `bamboo-lib` se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y se identifican 66.593 filas y 76 columnas. A continuación, se visualiza la estructura inicial de los datos:

ENTIDAD	MUNICIPIO	...	Suma de MAT-M-31	Suma de MAT-M-32
AGUASCALIENTES	1.0	...	0	0
	NaN	NaN	0	3
	NaN	NaN	0	0
	NaN	NaN	9	7
	NaN	NaN	1	9

Figura 82. Formato original datos ANUIES - origen de estudiantes

Extracción y Transformación

Este paso del pipeline sigue la secuencia mencionada a continuación:

- Renombre de columnas
- Se completan celdas *NaN* con los valores correspondientes haciendo uso de la función *ffill()*.
- Reemplazo de las entidades federativas (*ent_id*) por sus id.
- Limpieza de celdas que incluyen el *string Total* acompañando a valores numéricos. Dicho *string* es eliminado para facilitar agregaciones y futuros análisis. Así también, se eliminan los puntos (.) y dos puntos (:) para mejorar el formato del set de datos.
- Reemplazo de ids para los municipios. Para ello, se hace una *query* a Clickhouse con motivo de traer la dimensión *dim_shared_geography_mun* que contiene dichos ids.
- Uso de la función *melt* de pandas para llevar el set de datos a un formato *tidy*.
- Reemplazo de los valores en las columnas *type*, y *origin* por medio de diccionarios definidos en el script *static.py*.
- Procesamiento del nombre de instituciones, campus, carreras y programas. Se añaden los ids para las columnas *program*, *campus*. Esto se realiza posterior a la lectura de archivos almacenados en Google Cloud Storage y que contienen dicha información.
- Selección de variables a utilizar: *mun_id*, *campus_id*, *program*, *type*, *origin* y *value*.
- Creación de la columna *year* la cual contiene información con respecto al periodo de tiempo en estudio.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 18.212 filas y 7 columnas. La estructura final de la tabla se presenta a continuación:

mun_id	campus_id	program	type	origin	value	year
1001.0	1200001.0	11500042.0	11.0	1.0	12.0	2017
1001.0	1400001.0	11500039.0	11.0	1.0	24.0	2017
1001.0	800001.0	11500042.0	11.0	1.0	53.0	2017
1001.0	700001.0	11100077.0	11.0	1.0	53.0	2017
1001.0	200001.0	11100085.0	11.0	1.0	81.0	2017

Figura 83. Formato final tabla de datos ANUIES - origen de estudiantes

8.3. Anuies_status

Descripción General y Ejecución

El pipeline *status_pipeline.py* procesa los datos facilitados por la Asociación Nacional de Universidades e Instituciones de Educación Superior (ANUIES) que toman relación con el grado académico de los estudiantes (egresados, graduados y titulados). Los datos son descargados desde la plataforma de la [Asociación Nacional de Universidades e Instituciones de Educación Superior](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
(bamboo-cli) ~/data_etl/etl/anuies/$ python status_pipeline.py
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y se identifican 40.549 filas y 49 columnas. A continuación, se visualiza la estructura inicial de los datos:

ENTIDAD	MUNICIPIO	...	Suma de MAT-M-31	Suma de MAT-M-32
AGUASCALIENTES	1.0	...	0	0
	NaN	NaN	0	3
	NaN	NaN	0	0
	NaN	NaN	9	7
	NaN	...	1	9

Figura 84. Formato original datos ANUIES - grados académicos

Extracción y Transformación

Este paso del pipeline sigue la secuencia mencionada a continuación:

- Renombre de columnas
- Se completan celdas *NaN* con los valores correspondientes haciendo uso de la función *ffill()*.
- Reemplazo de las entidades federativas (*ent_id*) por sus ids.
- Limpieza de celdas que incluyen el string *Total* acompañando a valores numéricos. Dicho string es eliminado para facilitar agregaciones y futuros análisis. Así también, se eliminan los puntos (.) y dos puntos (:) para mejorar el formato del set de datos.
- Reemplazo de ids para los municipios. Para ello, se hace una query a Clickhouse con motivo de traer la dimensión *dim_shared_geography_mun* que contiene dichos ids.
- Cambios menores en el nombre de columnas para posteriormente usar la función *melt* de pandas para llevar el set de datos a un formato *tidy*.
- Reemplazo de los valores en las columnas *stat*, *sex* y *type* por medio de diccionarios definidos en el script *static.py*.
- Procesamiento del nombre de instituciones, campus, carreras y programas. Se añaden los ids para las columnas *program*, *campus*. Esto se realiza posterior a la lectura de archivos almacenados en Google Cloud Storage y que contienen dicha información.
- Selección de variables a utilizar: *mun_id*, *campus_id*, *program*, *type*, *sex*, *value* y *stat*.
- Creación de la columna *year* la cual contiene información con respecto al periodo de tiempo en estudio.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 26.109 filas y 8 columnas. A continuación puede ser visualizada su estructura de salida:

mun_id	campus_id	program	type	sex	value	stat	year
1001.0	1500001.0	11500039.0	11.0	1.0	1.0	1.0	2017
1001.0	1200001.0	11500042.0	11.0	1.0	2.0	1.0	2017
1001.0	1400001.0	11500039.0	11.0	1.0	7.0	1.0	2017
1001.0	800001.0	11500042.0	11.0	1.0	3.0	1.0	2017
1001.0	700001.0	11100077.0	11.0	1.0	1.0	1.0	2017

Figura 85. Formato final tabla de datos ANUIES - grados académicos

9. Población y Vivienda

A continuación se detalla el procesamiento aplicado a los datos del Censo de Población, tanto para personas como viviendas, además de datos de población de CONAPO y la proyección de la población.

9.1. Conapo_metro_area_population

Descripción General y Ejecución

El pipeline *metro_areas_population.py* procesa los datos facilitados por el Consejo Nacional de Población (CONAPO) que toman relación con la población total a nivel de Zona Metropolitana. Los datos son descargados desde la plataforma del [Consejo Nacional de Población](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
(Bamboo-cli) ~/data_etl/etl/metro_areas_population/$ bamboo-cli --folder .
--entry metro_areas_population
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y, con respecto a su estructura, se identifican 415 filas y 13 columnas. A continuación, se visualiza la estructura inicial de los datos:

	zona_metropolitana	mun_id	1900	...	DMU2 (hab/ha)		mun	zona_metropolitana_id
0	Aguascalientes	1001	506274	...	108.2		Aguascalientes	99101
1	Aguascalientes	1005	41092	...	75.0		Jesús María	99101
2	Aguascalientes	1011	n.a.	...	83.3	San Francisco de los Romo		99101
3	Ensenada	2001	259979	...	54.3		Ensenada	99201
4	Mexicali	2002	601938	...	59.3		Mexicali	99202

Figura 86. Formato original datos población CONAPO

Lectura y Transformación

Esta etapa sigue el siguiente proceso:

- Lectura de datos.
- Reemplazo de valores *n.a* por el valor 0.
- Transformación de las columnas a un formato *tidy*. Se crean dos nuevas columnas: la primera será el año en estudio, y la segunda, la población.
- Se renombran las columnas para tener mejor definición de sus atributos.
- Se elimina la variable *zm_id*. Esto ya que el municipio puede ser conocido a partir de la variable *zm_name*.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 1.245 filas y 5 columnas. A continuación puede ser visualizada su estructura de salida:

	zm_name	mun	mun_id	year	population
0	Aguascalientes	Aguascalientes	1001	2000	643419
1	Aguascalientes	Jesús María	1005	2000	64097
2	Aguascalientes	San Francisco de los Romo	1011	2000	20066
3	Ensenada	Ensenada	2001	2000	370730
4	Mexicali	Mexicali	2002	2000	764602

Figura 87. Formato final tabla de datos población CONAPO

9.2. Inegi_population_total

Descripción General y Ejecución

El pipeline *population_total_pipeline.py* procesa los datos facilitados por el [Instituto Nacional de Estadística y Geografía](#) de México (INEGI) que toman relación con la Población Total extraída del Censo de Población y Vivienda de los años 2010 y 2020. Los datos, una vez descargados desde su sitio, son almacenados en Google Storage para ser, posteriormente, procesados mediante un proceso de ETL (extracción, transformación, y carga).

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/inegi_census/$ bash population_total_ingest.sh
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y separada en archivos por entidades federativas para el 2010 y 2020. A modo de ejemplo, en el caso de la entidad federativa de Aguascalientes en 2020, la estructura inicial de archivo se compone de 2.058 filas y 232 columnas. A continuación, se visualiza la estructura inicial de estos datos:

ENTIDAD	NOM_ENT	MUN	NOM_MUN	LOC	NOM_LOC	LONGITUD	LATITUD	ALTITUD	POBTOT	...	VPH_CEL	VPH_INTER
1	Aguascalientes	0	Total de la entidad Aguascalientes	0	Total de la Entidad	NaN	NaN	NaN	1425607	...	359895	236003
1	Aguascalientes	0	Total de la entidad Aguascalientes	9998	Localidades de una vivienda	NaN	NaN	NaN	3697	...	732	205
1	Aguascalientes	0	Total de la entidad Aguascalientes	9999	Localidades de dos viviendas	NaN	NaN	NaN	3021	...	470	146
1	Aguascalientes	1	Aguascalientes	0	Total del Municipio	NaN	NaN	NaN	948990	...	251719	178619
1	Aguascalientes	1	Aguascalientes	1	Aguascalientes	102°17'45.768" W	21°52'47.362" N	1878.0	863893	...	232793	169675
...

Figura 88. Formato original datos del Censo Poblacional 2020 - Aguascalientes

Limpieza y Transformación

Esta etapa sigue el siguiente proceso:

- Selección de columnas a utilizar, las cuales son: *entidad, mun, loc, nom_mun, nom_loc, pobfem, y pobmas*.
- Eliminación de filas que, en el valor de localidad, posean el string *Total del Municipio*. Esto debido a que las filas con valores totales son redundantes.
- Reformulación de la columna *mun_id*. Se agrega el id de la entidad previo al id del municipio.
- Renombre de columnas.
- Creación de columna *year*, la cual define en año de realización del Censo.
- Transformación del *DataFrame* hacia un formato *tidy*. Para ello, se utiliza el módulo *melt* de Pandas. En ello, se crea la variable *sex* que define el sexo, y la variable *population* que define el total de la población.
- Reemplazo de la variable *sex* por sus ids.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, se obtiene un *DataFrame* con 22 filas y 4 columnas. El proceso es iterativo para cada entidad federativa y finalmente se unen todas las tablas en una única tabla que contendrá los valores para las 32 entidades federativas. A continuación puede ser visualizado un extracto de la estructura de salida de la tabla para Aguascalientes:

mun_id	year	sex	population
1001	2020	2	486917
1002	2020	2	26275
1003	2020	2	29687
1009	2020	1	11114
1010	2020	1	10446
1011	2020	1	30705

Figura 89. Formato final tabla de datos del Censo de Población - Aguascalientes

9.3. Inegi_population

Descripción General y Ejecución

El pipeline `population_pipeline.py` procesa los datos facilitados por el [Instituto Nacional de Estadística y Geografía](#) de México que toman relación con datos a nivel de Personas desde la Encuesta Intercensal 2015 y el cuestionario ampliado del Censo de Población y Vivienda de 2020. Los datos, una vez descargados desde su sitio, son almacenados en Google Storage para ser, posteriormente, procesados mediante un proceso de ETL (extracción, transformación, y carga).

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/inegi_intercensal_survey/$ bash population_ingest.sh
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería bamboo-lib se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y organizada por entidades federativas. A modo de ejemplo, el archivo con datos para la entidad federativa de Aguascalientes en 2020 presenta una estructura de 95.983 filas y 92 columnas. A continuación, se visualiza la estructura inicial de los datos:

ENT	MUN	LOC50K	ID_VIV	ID_PERSONA	COBERTURA	ESTRATO	UPM	FACTOR	CLAVIP	...	HIJOS_SOBREVIV	FECHA_NAC_M
1	1	1	10010000001	1001000000100001	2	01-001-0001-00	1	59	1	...	2.0	9.0
1	1	1	10010000001	1001000000100002	2	01-001-0001-00	1	59	1	...	NaN	NaN
1	1	1	10010000001	1001000000100003	2	01-001-0001-00	1	59	1	...	NaN	NaN
1	1	1	10010000002	1001000000200001	2	01-001-0001-00	1	59	1	...	NaN	NaN
1	1	1	10010000002	1001000000200002	2	01-001-0001-00	1	59	1	...	2.0	5.0
...

Figura 90. Formato original datos cuestionario ampliado Censo Poblacional - Aguascalientes

Limpieza y Transformación

Esta etapa sigue el siguiente proceso:

- Creación de variables definidas en la lista `extra_columns_low` con el valor `numpy.nan`. En dicha lista existen variables que no existen en la Encuesta Intercensal 2015, o el Censo de Población y Vivienda de 2020, o en ambos. Esto se realiza con motivo de integrar los datos en un mismo `DataFrame`.
- Reformulación de la variable `mun_id`. Se define mediante la agregación de: entidad, y municipio. En ese orden.
- Creación de la columna `mun_id_trab` mediante la agregación de las variables `ent_pais_trab`. La variable describe la zona geográfica en la que desarrolla su trabajo.
- Reemplazo de valores `numpy.nan` con el valor 0. Se utiliza el módulo `fillna` de Pandas.
- Se transforma el `dtype` de las variables `mun_id`, `mun_id_trab`, `factor` y `edad` a `int`.
- Creación del `DataFrame df_I`, el cual guarda transformaciones para los ids actuales de variables cualitativas, hacia id's de base común. Dichas transformaciones son obtenidas desde un archivo llamado *Intercensal Census IDs.xlsx*.
- Reemplazo de ids actuales para variables cualitativas por los ids de base común contenidos en el `DataFrame df_I`.
- Renombre de variables a inglés.

- Creación de la variable *filtered_age*, la cual es una variable binaria que denota el valor unitario cuando la edad es mayor o igual a 12 años, y cero en otro caso.
- Se asigna el valor *numpy.nan* a las variables en la lista *params_translated_add1*, cuando presentan un valor igual a 888888.
- Creación de la variable *year*, la cual define el periodo de realización del Censo.
- Reemplazo de valores en las columnas *state_of_birth* y *state_of_residency* por sus ids.
- Definición de la variable *country_of_residency_5_years*. Dicha variable toma el valor definido en la variable *state_of_residency* cuando su *dtype* es un *string*. En caso contrario, se le asigna el valor *numpy.nan*.
- Definición de la variable *state_of_residency_5_years*. Dicha variable toma el valor definido en la variable *state_of_residency* cuando su *dtype* no es un *string*. En caso de ser *string*, se le asigna el valor *numpy.nan*.
- Definición de la variable *country_of_birth*. Dicha variable toma el valor definido en la variable *state_of_birth* cuando su *dtype* es un *string*. En caso contrario, se le asigna el valor *numpy.nan*.
- Definición de la variable *state_of_birth*. Dicha variable toma el valor definido en la variable *state_of_birth* cuando su *dtype* no es un *string*. En caso de ser *string*, se le asigna el valor *numpy.nan*.
- Transformación de variables a objeto.
- Eliminación de la variable *state_of_residency*.
- Definición de la variable *foreign_migrant*. Dicha variable toma el valor unitario cuando la variable *country_of_residency_5_years* es diferente al valor *pd.isnull* (nulo), y cero en caso contrario.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, se obtiene un *DataFrame* con 95.983 filas y 49 columnas. A continuación puede ser visualizada un extracto de su estructura de salida:

sex	parent	sersalud	dhsersal1	laboral_condition	time_to_work	transport_mean_work	...	state_of_residency_5_years	country_of_birth	foreign_migrant
1	1	1	1	1.0	1.0	1.0	...	1.0	NaN	0
1	1	1	1	1.0	1.0	1.0	...	1.0	NaN	0
1	1	1	1	1.0	1.0	1.0	...	1.0	NaN	0
1	1	1	1	1.0	1.0	1.0	...	1.0	NaN	0
1	1	1	1	1.0	1.0	1.0	...	1.0	NaN	0
...

Figura 91. Formato final tabla de datos cuestionario ampliado Censo Poblacional - Aguascalientes

9.4. Population_basic_quest_by_age

Descripción General y Ejecución

El pipeline `population_basic_quest_by_age.py` procesa los datos facilitados por el [Instituto Nacional de Estadística y Geografía](#) de México que toman relación con el Censo de Población y Vivienda en 2020 - Cuestionario Básico. En específico, define características poblacionales por rango etario y sexo. Los datos fueron facilitados por INEGI, y se almacenaron en Google Storage para ser, posteriormente, procesados mediante un proceso de ETL (extracción, transformación, y carga).

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
(Bamboo-cli) ~/data_etl/etl/inegi_census/$ bamboo-cli --folder . --entry population_basic_quest_by_age
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería `bamboo-lib` se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato `.xlsx`; y su estructura inicial se compone de 150.120 filas y 15 columnas. A continuación, se visualiza la estructura inicial de los datos:

	Entidad federativa	...	Población con algún problema o condición mental	Unnamed: 14_level_1
0	Estados Unidos Mexicanos	1590583
1	Estados Unidos Mexicanos	90161
2	Estados Unidos Mexicanos	153402
3	Estados Unidos Mexicanos	166114
4	Estados Unidos Mexicanos	151910

Figura 92. Formato inicial tabla de datos cuestionario básico Censo Poblacional

Lectura y Transformación

Esta etapa sigue el siguiente proceso:

- Lectura de datos desde Google Cloud Platform.
- Asignación de nuevos nombres a columnas.
- Eliminación de filas donde las variables *municipality*, *sex*, y *age*, sean igual a *Total*. Esto se realiza para eliminar valores que corresponden a totales agregados, dado que pueden ser calculados.
- Creación de la variable *entity_code*, compuesta de los valores numéricos en la variable *entity*.
- Creación de la variable *municipality_code*, compuesta de los valores numéricos en la variable *municipality*.
- Creación de la variable *mun_id*. Dicha variable será compuesta por la agregación de los id a nivel de entidad y municipio. Para ello, se agregan las variables *entity_code* y *municipality_code*.
- Eliminación de variables que no serán utilizadas: *entity*, *municipality*, *entity_code*, y *municipality_code*.
- Reemplazo de valores en variables *age* y *sex* por sus id. Dichos reemplazos están definidos en diccionarios en el script *utils.py*.
- Creación de la variable *year*, la cual almacena el periodo en estudio.

En cuanto a la salida del proceso de ETL, se obtiene un *DataFrame* con 93.822 filas y 15 columnas. A continuación puede ser visualizada su estructura de salida:

	sex	age	total_population	afro_desc_population	...	talk_imperiment	mental_imperiment	mun_id	year
141	1	1	38966	535	...	577	471	1001	2020
142	1	6	41347	555	...	364	888	1001	2020
143	1	11	41849	502	...	259	911	1001	2020
144	1	16	43222	608	...	242	811	1001	2020
145	1	21	42528	785	...	197	718	1001	2020

Figura 93. Formato final tabla de datos cuestionario básico Censo Poblacional

9.5. Population_projection

Descripción General y Ejecución

El pipeline *population_projection.py* procesa los datos facilitados por el Consejo Nacional de Población (CONAPO) que toman relación con la proyección de la población. Los datos son descargados desde la plataforma del [Consejo Nacional de Población](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
(Bamboo-cli) ~/data_etl/etl/population_projection/$ bamboo-cli --folder .
--entry population_projection
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y, con respecto a su estructura, se identifican 1.100.736 filas y 9 columnas. A continuación, se visualiza la estructura inicial de los datos:

	RENGLON	CLAVE	CLAVE_ENT	NOM_ENT	MUN	SEXO	ANO	EDAD_QUIN	POB
0	1	1001	1	Aguascalientes	Aguascalientes	Mujeres	2015	pobm_00_04	39403
1	2	1001	1	Aguascalientes	Aguascalientes	Mujeres	2016	pobm_00_04	39204
2	3	1001	1	Aguascalientes	Aguascalientes	Mujeres	2017	pobm_00_04	38891
3	4	1001	1	Aguascalientes	Aguascalientes	Mujeres	2018	pobm_00_04	38581
4	5	1001	1	Aguascalientes	Aguascalientes	Mujeres	2019	pobm_00_04	38272

Figura 94. Formato original datos proyección de población

Lectura y Transformación

Esta etapa sigue el siguiente proceso:

- Lectura de datos. Se utiliza la función `read_csv` de Pandas.
- Transformación de nombres de columnas a minúscula. Se utiliza la función `str.lower()`.
- Selección de columnas a utilizar: `clave`, `sexo`, `año`, `edad_quin` y `pob`.
- Renombre de columnas al inglés. Los nombres anteriores se transforman a: `mun_id`, `sex`, `year`, `age` y `population`.
- Reemplazo de valores en columnas `sex` y `age` por sus `id's`.

Posterior al proceso de transformación, se obtiene un `DataFrame` con 1.100.736 filas y 5 columnas . A continuación puede ser visualizada su estructura de salida:

	mun_id	sex	year	age	population
0	1001	2	2015	1	39403
1	1001	2	2016	1	39204
2	1001	2	2017	1	38891
3	1001	2	2018	1	38581
4	1001	2	2019	1	38272

Figura 95. Formato final tabla de datos proyección de población

9.6. Inegi_housing

Descripción General y Ejecución

Para generar el cubo de datos *inegi_housing* se procesan los datos del 2010 y 2020 en pipelines separados debido a sus estructuras originales. Los pipeline trabajados son *housing_pipeline_2010.py* y *housing_pipeline_2020.py* y procesan los datos facilitados por el [Instituto Nacional de Estadística y Geografía](#) de México que toman relación con el cuestionario ampliado de viviendas del Censo Poblacional. Los datos, una vez descargados desde su sitio, son almacenados en Google Storage para ser, posteriormente, procesados mediante un proceso de ETL (extracción, transformación, y carga).

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

A) ETL para datos del 2010

Ejecución:

Para ejecutar el pipeline *housing_pipeline_2010.py* se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/inegi_census/$ bash housing_pipeline_2010.sh
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .dbf y su estructura inicial se compone de 16.572 filas y 56 columnas. A continuación, se visualiza la estructura inicial de los datos:

ENT	NOM_ENT	MUN	NOM_MUN	...	UPM	TAM_LOC	CERTEZA	IDH125
01	Aguascalientes	001	Aguascalientes	...	0002	01	0	0
01	Aguascalientes	001	Aguascalientes	...	0002	01	0	0
01	Aguascalientes	001	Aguascalientes	...	0002	01	0	0
01	Aguascalientes	001	Aguascalientes	...	0006	01	0	0
01	Aguascalientes	001	Aguascalientes	...	0023	01	0	0

Figura 96. Formato original datos de vivienda Censo Económico 2010

Limpieza y Transformación

Esta etapa sigue el siguiente proceso:

- Selección de columnas a utilizar, las cuales se encuentran definidas en la lista *labels*.
- Transformación de *dtypes*. Esto se realiza, debido a la gran cantidad de variables, con el diccionario *dtypes*.
- Creación del *DataFrame* *data*, el cual guarda transformaciones para los *id's* actuales de variables cualitativas, hacia *id's* de base común. Dichas transformaciones son obtenidas desde un archivo llamado *Intercensal Census IDs.xlsx*.
- Creación de la variable *loc_id*, la cual se compone por la agregación de los siguientes id: *entidad*, *municipio* y *localidad*.
- Reemplazo de intervalos de ingreso por sus *id's*.
- Reemplazo de *id's* actuales para variables cualitativas por los *id's* de base común.
- Renombre de columnas a inglés.
- Creación de columna *year*, la cual define el periodo de tiempo en estudio.
- Se agregan con valor *numpy.nan*, todas aquellas variables nuevas en el Censo de Población y Vivienda de 2020, y la Encuesta Intercensal 2015, que no fueron definidas en el censo actual. Esto se realiza con motivo de poder generar un *dataset* que posea la agregación de todos los datos existentes.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, se obtiene un *DataFrame* con 15.893 filas y 40 columnas. A continuación puede ser visualizada su estructura de salida:

loc_id	home_type	acquisition	...	age	national_financial_aid	retirement_financial_aid
10010000.0	1.0	1.0	...	NaN	NaN	NaN
10010000.0	1.0	1.0	...	NaN	NaN	NaN
10010000.0	1.0	1.0	...	NaN	NaN	NaN
10010000.0	1.0	1.0	...	NaN	NaN	NaN
10010000.0	1.0	1.0	...	NaN	NaN	NaN

Figura 97. Formato final tabla de datos de vivienda Censo Económico 2010

B) ETL para datos del 2020

Ejecución:

Para ejecutar el pipeline *housing_pipeline_2010.py* se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/inegi_census/$ python housing_pipeline_2020.py
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv; y su estructura inicial se compone de 24.349 filas y 83 columnas. A continuación, se visualiza la estructura inicial de los datos:

ENT	MUN	LOC50K	ID_VIV	COBERTURA	...	TIPOHOG	INGTRHOG	JEFE_SEXO	JEFE_EDAD	TAMLOC
1	1	1	10010000001	2	...	1	NaN	3	55	5
1	1	1	10010000002	2	...	1	30100.0	1	45	5
1	1	1	10010000003	2	...	5	20000.0	1	60	5
1	1	1	10010000004	2	...	1	66000.0	1	62	5
1	1	1	10010000005	2	...	1	25000.0	1	52	5

Figura 98. Formato original datos de vivienda Censo Económico 2020

Limpieza y Transformación

Esta etapa sigue el siguiente proceso:

- Selección de columnas a utilizar, las cuales se encuentran definidas en las listas *labels* y *extra_labels*. Se agrega *extra_labels* por separado ya que son las variables del Censo que no fueron incluidas en el Censo previo, es decir, del año 2010.

- Transformación de *dtypes*. Esto se realiza, debido a la gran cantidad de variables, con el diccionario *dtypes*.
- Creación del *DataFrame* data, el cual guarda transformaciones para los *id's* actuales de variables cualitativas, hacia *id's* de base común. Dichas transformaciones son obtenidas desde un archivo llamado *Intercensal Census IDs.xlsx*.
- Creación de la variable *loc_id*, la cual se compone por la agregación de los siguientes *id's*: *entidad*, *municipio* y *localidad*.
- Reemplazo de intervalos de ingreso por sus id.
- Reemplazo de *id's* actuales para variables cualitativas por los id de base común definidos en el dataframe data.
- Renombre de columnas a inglés.
- Creación de columna *year*, la cual define el periodo de tiempo en estudio.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, se obtiene un *DataFrame* con 24.347 filas y 40 columnas. A continuación puede ser visualizada su estructura de salida:

loc_id	home_type	acquisition	...	retirement_financial_aid	households	year
10010000.0	1.0	1.0	...	8.0	12.0	2020
10010000.0	1.0	1.0	...	8.0	11.0	2020
10010000.0	1.0	1.0	...	7.0	11.0	2020
10010000.0	1.0	1.0	...	8.0	11.0	2020
10010000.0	1.0	1.0	...	8.0	12.0	2020

Figura 99. Formato final tabla de datos de vivienda Censo Económico 2020

9.7. Inegi_housing_basic

Descripción General y Ejecución

El pipeline `inegi_housing_basic.py` procesa los datos facilitados por el [Instituto Nacional de Estadística y Geografía](#) de México que toman relación con el Censo de Población y Vivienda en 2020 - Cuestionario básico, en su apartado de vivienda, proporcionando datos de la disponibilidad de servicios y tecnologías en las viviendas mexicanas. Los datos fueron facilitados por INEGI y almacenados en Google Storage para ser, posteriormente, procesados mediante un proceso de ETL (extracción, transformación, y carga).

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y numpy, y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/inegi_census/$ python inegi_housing_basic.py
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería `bamboo-lib` se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx; y su estructura inicial se compone de 2.507 filas y 15 columnas. A continuación, se visualiza la estructura inicial de los datos:

		Unnamed: 0	Unnamed: 1	...	Unnamed: 13	Unnamed: 14
0		Estados Unidos Mexicanos	Total	...	6616141.0	4047100.0
1		01 Aguascalientes	Total	...	98724.0	70126.0
2		01 Aguascalientes	001 Aguascalientes	...	80951.0	56131.0
3		01 Aguascalientes	002 Asientos	...	596.0	556.0
4		01 Aguascalientes	003 Calvillo	...	1382.0	1352.0

Figura 100. Formato inicial datos de vivienda Censo Económico 2020 - Cuestionario Básico 1

Limpieza y Transformación

Esta etapa sigue el siguiente proceso:

- Lectura de datos desde Google Cloud Platform. En dicha etapa, se renombran las variables al inglés, habiendo previamente identificado la razón de sus valores.
- Eliminación de filas donde la variable *ent_name* sea igual a *Estados Unidos Mexicanos*, o donde la variable *mun_name* sea igual a *Total*. Esto se realiza para eliminar valores que corresponden a totales agregados, dado que pueden ser calculados.
- Creación de la variable *ent_id*, compuesta de los valores numéricos en la variable *ent_name*.
- Creación de la variable *mun_id*, compuesta de los valores numéricos en la variable *mun_name*.
- Modificación de la variable *mun_id*. Dicha variable será compuesta por la agregación de los id a nivel de entidad y municipio. Para ello, se agregan las variables *ent_id* y *mun_id*.
- Eliminación de variables que no serán utilizadas: *ent_id*, *ent_name*, y *mun_name*.
- Creación de la variable *year*, la cual almacena el periodo en estudio.
- Reorganización del conjunto de datos.

En cuanto a la salida del proceso de ETL, se obtiene un DataFrame con 2.469 filas y 15 columnas. A continuación puede ser visualizada su estructura de salida:

	mun_id	year	private_homes_inhab	fridge	...	internet	tv_service	movie_service	video_game_console	
0	1001	2020	266508	254959	...	178619	130290	80951	56131	
1	1002	2020		12522	11071	...	4526	3882	596	556
2	1003	2020		<u>15520</u>	14659	...	6553	4749	1382	1352
3	1004	2020		3931	3518	...	1741	1664	223	191
4	1005	2020		33171	31414	...	19920	13483	9296	6582

Figura 101. Formato final datos de vivienda Censo Económico 2020 - Cuestionario Básico 1

9.8. inegi_housing_ranges_basic

Descripción General y Ejecución

El presente pipeline procesa los datos facilitados por el [Instituto Nacional de Estadística y Geografía](#) de México que toman relación con el Censo de Población y Vivienda en 2020 - Cuestionario básico. En específico, define características de los jefes de hogar o persona de referencia por sexo y rango etario. Los datos fueron facilitados por INEGI y almacenados en Google Storage para ser, posteriormente, procesados mediante un proceso de ETL (extracción, transformación, y carga).

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
~/data_etl/etl/inegi_census/$ python inegi_housing_ranges_basic.py
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx; y su estructura inicial se compone de 135.112 filas y 5 columnas. A continuación, se visualiza la estructura inicial de los datos:

	Entidad federativa	...	Hogares censales
0	NaN	...	NaN
1	Estados Unidos Mexicanos	...	35219141.0
2	Estados Unidos Mexicanos	...	2029.0
3	Estados Unidos Mexicanos	...	163453.0
4	Estados Unidos Mexicanos	...	1094102.0

Figura 102. Formato inicial datos de vivienda Censo Económico 2020 - Cuestionario Básico 2

Limpieza y Transformación

Esta etapa sigue el siguiente proceso:

- Lectura de datos desde Google Cloud Platform.
- Asignación de nuevos nombres a columnas: *ent_name*, *mun_name*, *sex*, *age_range*, y *households*.
- Eliminación de filas donde las variables sean igual a *Total*. Esto se realiza para eliminar valores que corresponden a totales agregados, dado que pueden ser calculados.
- Creación de la variable *ent_id*, compuesta de los valores numéricos en la variable *ent_name*.
- Creación de la variable *mun_id*, compuesta de los valores numéricos en la variable *mun_name*.
- Modificación de la variable *mun_id*. Dicha variable será compuesta por la agregación de los id a nivel de entidad y municipio. Para ello, se agregan las variables *ent_id* y *mun_id*.
- Reemplazo de valores en columna *sex* por sus id.
- Renombre de rangos etarios.
- Eliminación de variables que no serán utilizadas: *ent_id*, *ent_name*, y *mun_name*.
- Creación de la variable *year*, la cual almacena el periodo en estudio.
- Reorganización del conjunto de datos.

En cuanto a la salida del proceso de ETL, y considerando el ejemplo mencionado previamente, obtenemos un *DataFrame* con 83.946 filas y 5 columnas. A continuación puede ser visualizada su estructura de salida:

	mun_id	year	sex	age_range	households
0	1001	2020	1	12 a 14 años	6
1	1001	2020	1	15 a 19 años	695
2	1001	2020	1	20 a 24 años	6315
3	1001	2020	1	25 a 29 años	15549
4	1001	2020	1	30 a 34 años	20267

Figura 103. Formato final datos de vivienda Censo Económico 2020 - Cuestionario Básico 2

10. Consejo Nacional de Evaluación de Política de Desarrollo Social (CONEVAL)

A continuación se detalla el procesamiento aplicado a los datos provenientes del Consejo Nacional de Evaluación de Política de Desarrollo Social (CONEVAL). En este caso, se encuentran tres pipelines que procesan los valores del índice de GINI a diferentes niveles geográficos y otros tres pipelines para procesar datos de brechas sociales y pobreza.

10.1. Coneval_gini_nat

Descripción General y Ejecución

El pipeline *gini_pipeline_nat.py* procesa los datos facilitados por el Consejo Nacional de Evaluación de la Política de Desarrollo Social (CONEVAL) que toman relación con el cálculo del Coeficiente de Gini. En ello, el pipeline se refiere a los datos a nivel nacional. Los datos son descargados desde la plataforma del [Consejo Nacional de Evaluación de la Política de Desarrollo Social](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
(Bamboo-cli) ~/data_etl/etl/coneval/$ bamboo-cli --folder . --entry
gini_pipeline_nat
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y, con respecto a su estructura, se identifican 35 filas y 7 columnas. Es importante señalar que del archivo inicial, se agregan en un mismo conjunto de datos las hojas de cálculo “Coeficiente de Gini 2008-2014” y “Coeficiente de Gini 2016-2018”, dado que contienen el Coeficiente Gini para diferentes años. A continuación, se visualiza la estructura inicial de los datos:

	Unnamed: 0	Entidad federativa	...	Unnamed: 5	Unnamed: 6
0	NaN		NaN	2012.000000	2014.000000
1	NaN	Aguascalientes	...	0.479191	0.486294
2	NaN	Baja California	...	0.464538	0.433568
3	NaN	Baja California Sur	...	0.492758	0.454270
4	NaN	Campeche	...	0.533018	0.499936

Figura 104. Formato original datos GINI nivel nacional

Lectura y Transformación

Esta etapa sigue el siguiente proceso:

- Lectura de las hojas de cálculo en el archivo que poseen información del Coeficiente Gini para diferentes períodos de tiempo. Dichos conjuntos de datos temporales son concatenados en un único *DataFrame*. Para ello, se utiliza la función *append* de Pandas.
- Selección de entidades con nombre Estados Unidos Mexicanos. De esta forma, se obtienen sólo los datos a nivel nacional.
- Reemplazo de la entidad Estados Unidos Mexicanos con su *id*.
- Transformación del tipo de dato para las columnas: *ent_id*, *year* y *gini*.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 6 filas y 3 columnas. A continuación puede ser visualizada su estructura de salida:

	ent_id	year	gini
0	0	2008	0.505337
1	0	2010	0.508857
2	0	2012	0.497666
3	0	2014	0.503281
4	0	2016	0.498373
5	0	2018	0.468821

Figura 105. Formato final tabla de datos GINI nivel nacional

10.2. Coneval_gini_ent

Descripción General y Ejecución

El pipeline *gini_pipeline_ent.py* procesa los datos facilitados por el Consejo Nacional de Evaluación de la Política de Desarrollo Social (CONEVAL) que toman relación con el cálculo del Coeficiente de Gini. En ello, el pipeline se refiere a los datos a nivel de Entidad Federativa. Los datos son descargados desde la plataforma del [Consejo Nacional de Evaluación de la Política de Desarrollo Social](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso de limpieza y transformación, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
(Bamboo-cli) ~/data_etl/etl/coneval/$ bamboo-cli --folder . --entry gini_pipeline_ent
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y, con respecto a su estructura, se identifican 35 filas y 7 columnas. Es importante señalar que del archivo inicial, se agregan en un mismo conjunto de datos las hojas de cálculo “Coeficiente de Gini 2008-2014” y “Coeficiente de Gini

2016-2018”, dado que contienen el Coeficiente Gini para diferentes años. A continuación, se visualiza la estructura inicial de los datos:

	Unnamed: 0	Entidad federativa	...	Unnamed: 5	Unnamed: 6
0	NaN	NaN	...	2012.000000	2014.000000
1	NaN	Aguascalientes	...	0.479191	0.486294
2	NaN	Baja California	...	0.464538	0.433568
3	NaN	Baja California Sur	...	0.492758	0.454270
4	NaN	Campeche	...	0.533018	0.499936

Figura 106. Formato original datos GINI nivel estatal

Lectura y Transformación

Esta etapa sigue el siguiente proceso:

- Lectura de las diferentes hojas de cálculo que poseen datos para distintos períodos de tiempo.
- Selección de entidades cuyo nombre difiere a *Estados Unidos Mexicanos*. Esto permite eliminar datos a nivel nacional.
- Reemplazo de entidades por sus *id's*. Dicha operación se realiza haciendo uso del archivo *countries-EF-codes.xlsx*.
- Transformación del *dtype* para las columnas: *ent_id*, *year* y *gini*.

Posterior al proceso de transformación, obtenemos un *DataFrame* con 192 filas y 3 columnas. A continuación puede ser visualizada su estructura de salida:

	ent_id	year	gini
0	1	2008	0.515696
1	2	2008	0.453242
2	3	2008	0.495741
3	4	2008	0.523707
4	5	2008	0.469642

Figura 107. Formato final tabla de datos GINI nivel estatal

10.3. Coneval_gini_mun

Descripción General y Ejecución

El pipeline *gini_pipeline_mun.py* procesa los datos facilitados por el Consejo Nacional de Evaluación de la Política de Desarrollo Social (CONEVAL) que toman relación con el cálculo del Coeficiente de Gini. En ello, el pipeline se refiere a los datos a nivel de municipio. Los datos son descargados desde la plataforma del [Consejo Nacional de Evaluación de la Política de Desarrollo Social](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso de limpieza y transformación, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
(Bamboo-cli) ~/data_etl/etl/coneval/$ bamboo-cli --folder . --entry gini_pipeline_mun
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx y, con respecto a su estructura, se identifican 2.463 filas y 7 columnas. Es importante señalar que del archivo inicial, se agregan en un mismo conjunto de datos las hojas de cálculo 2010 y 2015, dado que contienen el Coeficiente Gini para diferentes años. A continuación, se visualiza la estructura inicial de los datos:

	Unnamed: 0	Clave de entidad	... Coeficiente de Gini	Razón de ingreso 1
0	NaN	NaN ...	NaN	NaN
1	NaN	NaN ...	NaN	NaN
2	NaN	01 ...	0.426187	0.117213
3	NaN	01 ...	0.442576	0.119099
4	NaN	01 ...	0.426041	0.117333

Figura 108. Formato original datos GINI nivel municipal

Lectura y Transformación

Esta etapa sigue el siguiente proceso:

- Lectura de hojas de cálculo que poseen datos a nivel municipal.
- Selección de datos cuyo valor en la variable *Clave de municipio* difiere a *NaN*.
- Selección de columnas a utilizar: *Clave de municipio*, *Coeficiente de Gini* y *Razón de ingreso 1*.
- Creación de la columna *year*. Dicha columna se refiere al periodo de tiempo en estudio medido en años.
- Renombre de columnas a: *mun_id*, *gini* y *income_rate*.

Posterior al proceso de transformación, obtenemos un *DataFrame* con 4.913 filas y 4 columnas. A continuación puede ser visualizada su estructura de salida:

	mun_id	gini	income_rate	year
2	1001	0.426187	0.117213	2010
3	1002	0.442576	0.119099	2010
4	1003	0.426041	0.117333	2010
5	1004	0.427528	0.122823	2010
6	1005	0.449637	0.125425	2010

Figura 109. Formato final tabla de datos GINI nivel municipal

10.4. Coneval_poverty

Descripción General y Ejecución

El pipeline *poverty_pipeline.py* procesa los datos facilitados por el Consejo Nacional de Evaluación de la Política de Desarrollo Social (CONEVAL) que toman relación con la medición de la pobreza a nivel municipal. Los datos son descargados desde la plataforma del [Consejo Nacional de Evaluación de la Política de Desarrollo Social](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso de limpieza y transformación, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando la siguiente forma:

```
(Bamboo-cli)  ~/data_etl/etl/coneval/$ bamboo-cli --folder . --entry
poverty_pipeline --year=<year>
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y, con respecto a su estructura, se identifican 2.456 filas y 37 columnas. A continuación, se visualiza la estructura inicial de los datos:

	clave_entidad	entidad_federativa	clave_municipio	municipio	...	plb	plb_pob	plbm	plbm_pob
0	1	Aguascalientes	1001	Aguascalientes	...	39.2	314,300	9.9	79,703
1	1	Aguascalientes	1002	Asientos	...	70.8	34,247	33.8	16,326
2	1	Aguascalientes	1003	Calvillo	...	71.9	41,428	33.9	19,519
3	1	Aguascalientes	1004	Cosío	...	59.9	8,950	22.3	3,331
4	1	Aguascalientes	1005	Jesús María	...	49.8	50,888	16.9	17,287

Figura 110. Formato original datos de pobreza

Lectura y Transformación

Esta etapa sigue el siguiente proceso:

- Lectura del archivo .csv.
- Renombre de columnas a inglés.
- Creación de la columna *year*, la cual hace referencia al periodo de estudio.
- Reemplazo de caracteres no deseados: comas (,) y textos (n.d).

Posterior al proceso de transformación, se obtiene un *DataFrame* con 2.456 filas y 19 columnas. A continuación puede ser visualizada su estructura de salida:

	mun_id	population	poverty	...	income_below_welfare_line	income_below_min_welfare_line	year
0	1001	801807.0	242317.0	...	314300.0		79703.0 2010
1	1002	48358.0	31694.0	...	34247.0		16326.0 2010
2	1003	57627.0	39419.0	...	41428.0		19519.0 2010
3	1004	14929.0	8091.0	...	8950.0		3331.0 2010
4	1005	102211.0	43315.0	...	50888.0		17287.0 2010

Figura 111. Formato final tabla de datos de pobreza

11. Seguridad Pública

A continuación se presentan el detalle de los pipelines que procesan los datos de percepción de seguridad provenientes de la Encuesta Nacional de Victimización y Percepción de Seguridad Pública (ENVIPE), además del pipeline que procesa los datos facilitados por el Secretariado Ejecutivo del Sistema Nacional de Seguridad Pública (SESNSP) que toman relación con los reportes de incidencia delictiva.

11.1. Inegi_envipe

Descripción General y Ejecución

Este pipeline *envipe_pipeline.py* procesa los datos facilitados por el Instituto Nacional de Estadística y Geografía (INEGI) que toman relación con la Encuesta Nacional de Victimización y Percepción de Seguridad Pública (ENVIPE). Los datos de dicha encuesta son descargados desde la plataforma del [Instituto Nacional de Estadística y Geografía](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

```
~/data_etl/etl/envipe/$ source envipe_ingest.sh
```

```
(bamboo-cli)    ~/data_etl/etl/envipe/$ bamboo-cli --folder . --entry
envipe_pipeline --year=<year_value>
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .dbf por lo tanto, posterior a su organización en un *DataFrame* de la librería Pandas, se identifican 186 columnas y más de 90.000 filas. A continuación, se visualiza un extracto de la estructura inicial de los datos:

ID_VIV	ID_HOG	ID_PER	UPM	VIV_SEL	HOGAR	RESUL_H	R_SEL	SEXO	EDAD	AP5_6_10	AP5_8	FAC_HOG	FAC_ELE	FAC_HOG
100013.01	0100013.01.01	0100013.01.01.01	100013	1	1	B	1	2	53	...	NaN	4	184	184
100013.02	0100013.02.01	0100013.02.01.02	100013	2	1	A	2	2	42	...	3.0	3	184	368
100013.03	0100013.03.01	0100013.03.01.01	100013	3	1	B	1	2	44	...	3.0	4	184	184
100013.04	0100013.04.01	0100013.04.01.02	100013	4	1	B	2	2	62	...	NaN	2	184	551
100013.05	0100013.05.01	0100013.05.01.03	100013	5	1	B	3	1	19	...	NaN	2	184	551
...
3260731.17	3260731.17.01	3260731.17.01.01	3260731	17	1	B	1	1	75	...	4.0	1	246	491
3260731.18	3260731.18.01	3260731.18.01.01	3260731	18	1	A	1	2	26	...	NaN	3	246	246

Figura 112. Formato original datos de ENVIPE

Lectura

En esta etapa es donde se transforma el archivo .dbf a un *DataFrame* de Pandas, y además, se estandarizan los nombres de las columnas a caracteres en minúscula.

Transformación

En esta etapa se sigue el siguiente proceso:

- Seleccionar las columnas relevantes para el análisis de datos.
- Modificar el nombre de las columnas en el *DataFrame*, y también, el *dtype* de: *homes_factor*, *people_factor*, *expenses_in_protection_against_crime*.
- Reemplazar los valores en la columna *expenses_in_protection_against_crime* por su *id*. Este proceso se lleva a cabo leyendo un archivo .xlsx desde Google Drive que contiene el nivel de gasto en protección para diferentes niveles de ingreso (*income*).
- Modificación de la columna *mun_id* que toma relación con la geografía a nivel municipal de los datos. En ello, se agrega el *id* del estado en estudio.
- Reorganizar el orden de las columnas en el *DataFrame*.

- Creación de una nueva columna que hace referencia al año llamada *year*.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 23 columnas y más de 90.000 filas. A continuación puede ser visualizada su estructura de salida:

security_perception_in_their_state	sociodemographic_stratum	expenses_in_protection_against_crime	neighbors_trust	coworkers_trust	family_trust	friends_trust	ti
1.0	1.0		1.0	1.0	1.0	1.0	1.0
1.0	1.0		1.0	1.0	1.0	1.0	1.0
1.0	1.0		1.0	1.0	1.0	1.0	1.0
1.0	1.0		1.0	1.0	1.0	1.0	1.0
1.0	1.0		1.0	1.0	1.0	1.0	1.0

Figura 113. Formato final tabla de datos de ENVIPE

11.2. Sensnsp_crimes

Descripción General y Ejecución

El pipeline *crimes_pipeline.py* procesa los datos facilitados por el Secretariado Ejecutivo del Sistema Nacional de Seguridad Pública (SESNSP) que toman relación con los reportes de incidencia delictiva. Los datos son descargados desde la plataforma del [Secretariado Ejecutivo del Sistema Nacional de Seguridad Pública](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

```
~/data_etl/etl/crime/$ source crimes_ingest.sh  
(bamboo-cli) ~/data_etl/etl/crime/$ bamboo-cli --folder . --entry crimes_pipeline
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y se identifican 1.346.618 filas y 21 columnas. A continuación, se visualiza un extracto de la estructura inicial de los datos:

Año	Clave_Ent	Entidad	Cve. Municipio	Municipio	Bien jurídico afectado	Tipo de delito	Subtipo de delito	Modalidad	Enero	...	Marzo	Abril	Mayo	Junio	Ju
0	2015	1	Aguascalientes	1001	Aguascalientes	La vida y la Integridad corporal	Homicidio	Homicidio doloso	Con arma de fuego	2	...	1	1	0	1
1	2015	1	Aguascalientes	1001	Aguascalientes	La vida y la Integridad corporal	Homicidio	Homicidio doloso	Con arma blanca	1	...	0	0	0	1
2	2015	1	Aguascalientes	1001	Aguascalientes	La vida y la Integridad corporal	Homicidio	Homicidio doloso	Con otro elemento	0	...	1	1	3	2

Figura 114. Formato original datos del SESNSP

Lectura y Transformación

En esta etapa se desarrollan los siguientes pasos:

- Eliminación de columnas no relevantes en el proceso de ETL.
- Renombre de columnas. En ello, para hacer renombre de las columnas que hacen referencia a los meses se utiliza un diccionario definido como *dict_months*.
- Transformación del *DataFrame* mediante la función *melt* de Pandas. Con ello, se obtiene un *DataFrame* en formato *tidy*.
- Creación de la columna *month_id*. Dicha columna se crea mediante la agregación del año y el mes en estudio.
- Reemplazo de valores en la columna *crime_modality*. Dichos valores se reemplazan con el *id* del tipo de crimen. El *id* se obtiene desde un Google Spreadsheet, y se utiliza la función *replace* de Pandas para llevar a cabo el proceso.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 16.159.414 filas y 5 columnas. A continuación puede ser visualizada su estructura de salida:

	mun_id	crime_subtype_id	crime_modality_id	value	month_id
0	1001	50302		8	2 201501
1	1001	50302		7	1 201501
2	1001	50302		9	0 201501
3	1001	50302		27	1 201501
4	1001	50301		8	0 201501

Figura 115. Formato final tabla de datos del SESNSP

12. Economía

A continuación se presenta el detalle de los pipelines que procesan datos relacionados con economía general. Se incorpora el pipeline que procesa los datos del Directorio Estadístico Nacional de Unidades Económicas (DENU), el pipeline que procesa la información de Producto Interno Bruto y el pipeline que agrupa información de indicadores mundiales del Banco Mundial.

12.1. Inegi_denue

Descripción General y Ejecución

El pipeline `companies_pipeline.py` es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos del Directorio Estadístico Nacional de Unidades Económicas, DENU. Cada dataset cuenta con datos relacionados a número de trabajadores, código postal, ubicación geográfica y fecha de inscripción de las unidades económicas. Los datos son descargados desde Google Storage, se someten a un proceso de limpieza y transformación, y luego son ingestados en una base de datos *ClickHouse* siguiendo un modelo relacional para formar el cubo de datos.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando:

```
(bamboo-cli) ~/data_etl/etl/denue/$ python run.py
```

Descarga de Datos

Una vez obtenidos los datos desde el sitio de INEGI y almacenados en un compartimiento privado de GCP storage, se establece una conexión privada validada con credenciales fuertemente encriptadas, para ejecutar una descarga segura hasta el servidor de procesamiento de los datos.

Lectura

Una vez descargados los datos, estos se leen y almacenan en un *DataFrame* de la librería pandas. La estructura de este pipeline agrega unidades económicas a la fecha de creación de un nuevo módulo, por lo que no se puede dar una descripción de las tablas de datos que represente a todas las bases de datos disponibles. La siguiente imagen muestra las primeras filas y algunas columnas del *DataFrame* inicial obtenido de ejemplo, el cual posee 730.133 filas y 41 columnas.

	id	nom_estab	raz_social	codigo_act	...	tipoUniEco	latitud	longitud	fecha Alta	
0	6861538	3ITEC	TRESITEC SC	811312	...	Fijo	21.932796	-102.338508	2019-04	
1	6173	5A SEC	VIKAL MAQUINARIA, S.A. DE C.V.	812210	...	Fijo	21.858925	-102.289707	2010-07	
2	39995	5A SEC	VIKAL MAQUINARIA, S.A. DE C.V.	812210	...	Fijo	21.900844	-102.306392	2014-12	
3	12110	5A SEC	VIKAL MAQUINARIA, S.A. DE C.V.	812210	...	Fijo	21.920297	-102.297837	2010-07	
4	7344	AAA SERVICIOS Y AUTOPERTES		NaN	811116	...	Fijo	21.891568	-102.272175	2010-07
...	
730128	4627618	ZONA COMPUCELL		NaN	811499	...	Fijo	22.648415	-102.996615	2014-12
730129	4603077	ZONDA		NaN	811410	...	Fijo	23.830999	-103.032196	2010-07
730130	4630290	ZTILLEN		NaN	811121	...	Fijo	22.770066	-102.595239	2014-12
730131	4634747	ZUMBA SIN NOMBRE		NaN	812110	...	Fijo	22.756846	-102.496715	2014-12
730132	4616119	ZUÑIGA HERMANOS AUTOMOTRIZ		NaN	811116	...	Fijo	23.585186	-103.255779	2010-07

Figura 116. Formato inicial tabla de datos DENU

Transformación y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Esta etapa sigue el siguiente proceso:

- Se crea una columna relacionada al municipio al que pertenece la unidad económica.

- Se estandarizan los rangos de valores para los datos de personal ocupado, eliminando ciertos caracteres especiales.
- Por medio de un diccionario mensual, se estandariza el mes de incorporación de la unidad económica para crear un *id de fecha* por unidad.
- Se reemplaza el rango de personas en la empresa por un *id por tramo*.
- Se reemplaza el tipo de establecimiento de la empresa por un *id por categoría*.
- Se renombran las columnas de la base de datos a nombres interpretables por columna.
- Se procesan los códigos postales por unidad económica.
- Se aplica un ciclo *for* para cambiar el formato de las columnas a los correspondientes por tipo de valor en la columna (*int, float*).
- Por último, se capture la fecha de publicación de la base de datos por el archivo.

Al terminar el proceso de transformación y limpieza, se crea una tabla formateada lista para su ingestión. A continuación, se muestra una tabla ejemplo de la versión final antes de su ingestión:

	id	national_industry_id	n_workers	postal_code	mun_id	cve_loc	establishment	latitude	longitude	directory_added_date	lower	upper
0	8195	931610	3	20190	1001	0001	1	21.879282	-102.272162	201007	11	30
1	27184	931410	2	20660	1006	0001	1	22.147843	-102.275354	201007	6	10
2	4258	931410	1	20210	1001	0001	1	21.868154	-102.309784	201007	0	5
3	20349	931410	6	20259	1001	0001	1	21.880022	-102.280883	201007	101	250
4	21217	931410	2	20400	1007	0001	1	22.225426	-102.323554	201007	6	10
...
81661	4624534	931610	1	99030	32010	0001	1	23.182442	-102.881740	201412	0	5
81662	4624540	931610	1	99030	32010	0001	1	23.182442	-102.881740	201412	0	5
81663	4624542	931610	1	99030	32010	0001	1	23.182442	-102.881740	201412	0	5
81664	4624538	931610	1	99030	32010	0001	1	23.182442	-102.881740	201412	0	5
81665	4617965	931610	1	99200	32049	0001	1	22.779705	-103.560514	201007	0	5

Figura 117. Formato final tabla de datos DENUE

12.2. Inegi_gdp

Descripción General y Ejecución

El pipeline *inegi_gdp.py* procesa los datos del Instituto Nacional de Estadística y Geografía (INEGI) relacionados con el producto interno bruto de cada unidad económica a través de periodos temporales trimestrales. El pipeline es alimentado por un documento en formato Excel, el cual es sometido a un proceso de limpieza y transformación, y luego, los datos son ingestados en una base de datos *ClickHouse* siguiendo un modelo relacional para formar el cubo de datos.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando:

```
(bamboo-cli) ~/data_etl/etl/inegi_gdp/$ python inegi_gdp.py
```

Descarga de Datos

Una vez obtenidos los datos desde el sitio de INEGI y almacenados en un compartimiento privado de GCP storage, se establece una conexión privada validada con credenciales fuertemente encriptadas, para ejecutar una descarga segura hasta el servidor de procesamiento de los datos.

Lectura

Una vez descargados los datos, estos se leen y almacenan en un *DataFrame* de la librería pandas. Al momento en el que se elaboró esta documentación, el

DataFrame inicial contenía 182 filas y 204 columnas. La siguiente imagen muestra las primeras filas y algunas columnas del *DataFrame* inicial obtenido:

	Concepto	1993	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	1994
0	NaN	T1	T2	T3	T4	6 Meses	9 Meses	Anual	T1
1	__abB.1bP - Producto interno bruto	1502180.505	1553803.154	1553768.129	1630621.357	1527991.829	1536583.929	1560093.286	1669869.542
2	D.21-D.31 - Impuestos sobre los productos...	68635.191	68168.205	65257.342	72201.25	68401.698	67353.579	68565.497	82505.449
3	__cdB.1bV - Valor agregado bruto	1433545.313	1485634.949	1488510.787	1558420.106	1459590.131	1469230.35	1491527.789	1587364.093
4	Actividades primarias	76185.057	77963.609	75552.257	88589.301	77074.333	76566.974	79572.556	77350.774
...
180	RCifras revisadas	NaN							
181	PCifras preliminares	NaN							

Figura 118. Formato inicial tabla de datos Producto Interno Bruto

Transformación y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Esta etapa sigue el siguiente proceso:

- Se eliminan espacios en blanco en los nombres de todas las columnas del *DataFrame*.
- Se inicia la primera celda del *DataFrame* (posición 0,0) como periodo, transponer la tabla y reiniciar el índice de la misma.
- Convertir elementos innecesarios del *DataFrame* (textos con la palabra *unnamed*) en la columna *index* a valores *NaN*, para poder ser reemplazados con la función *ffill()* por los años disponibles en la columna.
- Por medio de un ciclo *for* sobre un arreglo llamado *levels*, se crea un nuevo arreglo llamado *mask*.

- Haciendo uso de *mask*, se reemplazan los nombres de las columnas del *DataFrame* y se remueven los NaN. Además, se formatean los nombres de las columnas eliminando textos vacíos en los extremos, pasar los nombres a minúsculas y eliminar espacios entre palabras con guión bajo (...).
- Remover en la columna de periodo trimestral los caracteres dejando solo los números por trimestre.
- Aplicar una función *melt()* al *DataFrame*, de manera de tener la data en un formato *tidy*, agrupando por concepto y periodo, con una nueva columna para el sector por ID (*sector_id*).
- A la nueva columna *sector_id*, se le separa por el carácter especial _- con el fin de conservar solo el ID del sector, agregando el nombre en el posterior esquema.
- Se reemplazan los nombres de las columnas del *DataFrame* por *quarter_id*, *quarter*, *sector_id* y *value*.
- Por último, se crea un nuevo *quarter_id* a base de las columnas de *quarter* y *quarter_id*, y eliminar la columna *quarter*.

Al terminar el proceso de transformación y limpieza, se crea una tabla formateada lista para su ingestión. A continuación, se muestra una tabla ejemplo de la versión final de la misma antes de su ingestión.

	quarter_id	sector_id	value
0	19931	11	76185.1
1	19932	11	77963.6
2	19933	11	75552.3
3	19934	11	88589.3
4	19941	11	77350.8
...
2255	20201	93	962247
2256	20202	93	928973
2257	20203	93	898630
2258	20204	93	923265
2259	20211	93	963792

Figura 119. Formato final tabla de datos Producto Interno Bruto

12.3. Indicators_i_wdi_a

Descripción General y Ejecución

El pipeline *wdi_indicators_pipeline.py* procesa los datos de indicadores de desarrollo provenientes del Banco Mundial, compilados a partir de fuentes internacionales reconocidas oficialmente. Los datos son descargados desde la plataforma del [Banco Mundial](#), y son almacenados en Google Storage para su posterior procesamiento.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando:

```
(bamboo-cli) ~/data_etl/etl/wdi_indicators/$ bamboo-cli --folder . --entry
wdi_indicators_pipeline
```

Descarga de Datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y son dos los archivos utilizados para el procesamiento: el primer archivo posee información sobre los indicadores de desarrollo mundial en distintos períodos de tiempo, y el segundo, sobre la población agrupada por país.

El archivo de indicadores de Desarrollo Mundial posee una estructura de 378.575 filas y 66 columnas:

	Country Name	Country Code	Indicator Name	...	2019	2020	Unnamed: 65
0	Arab World	ARB	Access to clean fuels and technologies for coo...	...	NaN	NaN	NaN
1	Arab World	ARB	Access to electricity (% of population)	...	NaN	NaN	NaN
2	Arab World	ARB	Access to electricity, rural (% of rural popul...	...	NaN	NaN	NaN
3	Arab World	ARB	Access to electricity, urban (% of urban popul...	...	NaN	NaN	NaN
4	Arab World	ARB	Account ownership at a financial institution o...	...	NaN	NaN	NaN

Figura 120. Formato inicial tabla de indicadores de Desarrollo Mundial

El archivo de Indicadores de Desarrollo Mundial Adicionales posee una estructura de 61 filas y 5 columnas.

	year	country	pop_combined	pop_female	pop_male
0	1960	asxxb	10 876	5 334	5 542
1	1961	asxxb	11 258	5 518	5 740
2	1962	asxxb	11 655	5 707	5 949
3	1963	asxxb	12 065	5 899	6 166
4	1964	asxxb	12 482	6 091	6 391
..
56	2016	asxxb	23 618	11 835	11 783
57	2017	asxxb	23 675	11 876	11 798
58	2018	asxxb	23 726	11 914	11 812
59	2019	asxxb	23 774	11 950	11 824
60	2020	asxxb	23 817	11 982	11 834

Figura 121. Formato inicial tabla de Indicadores de Desarrollo Mundial Adicionales

Lectura y Transformación

Esta etapa sigue el siguiente proceso:

- Lectura de archivo que almacena los Indicadores de Desarrollo Mundial.
- Renombre de columnas a conveniencia.
- Eliminación de filas cuyo valor en la variable *geo_id* existe en la lista *EXCLUSIONS*. Los valores eliminados corresponden a organizaciones que no son países.
- Reemplazo de valores erróneos en la columna *geo_id* mediante el diccionario *CORRECTIONS*.
- Transformación del conjunto de datos a formato *tidy*. La idea es transformar los años que existen como columnas, a filas dentro del conjunto de datos, dado que después se podrán hacer agregaciones bajo este nuevo formato.
- Creación de la columna *year*, la cual almacena el periodo temporal en estudio.
- Lectura del archivo Indicadores de Desarrollo Mundial Adicionales.

- Creación de la medida `total_pop`. Dicha medida se calcula a partir del conjunto de datos adicional, y su valor es el de la variable `pop_combined` multiplicado por 1000. La medida es agregada bajo el indicador `SP.POP.TOTL` a la lista `extra_data`, haciendo uso de la función `append` de Pandas.
- Creación de la medida `total_female_pop`. Dicha medida se calcula a partir del conjunto de datos adicional, y su valor es el de la variable `pop_female` multiplicado por 1000. La medida es agregada bajo el indicador `SP.POP.TOTL.FE.IN` a la lista `extra_data`, haciendo uso de la función `append` de Pandas.
- Creación de la medida `total_male_pop`. Dicha medida se calcula a partir del conjunto de datos adicional, y su valor es el de la variable `pop_male` multiplicado por 1000. La medida es agregada bajo el indicador `SP.POP.TOTL.FE.ZS` a la lista `extra_data`, haciendo uso de la función `append` de Pandas.
- Creación de la medida `pct_female_pop`. Dicha medida se calcula a partir del conjunto de datos adicional, y su valor es la división de la variables `total_female_pop` por `total_pop`, multiplicado por 100. La medida es agregada bajo el indicador `SP.POP.TOTL.MA.IN` a la lista `extra_data`, haciendo uso de la función `append` de Pandas.
- Creación de la medida `pct_male_pop`. Dicha medida se calcula a partir del conjunto de datos adicional, y su valor es la división de la variables `total_male_pop` por `total_pop`, multiplicado por 100. La medida es agregada bajo el indicador `SP.POP.TOTL.MA.ZS` a la lista `extra_data`, haciendo uso de la función `append` de Pandas.
- Creación del dataframe `extra_df`, el cual se construye a partir de los valores recientemente calculados, y que están almacenados en la lista `extra_data`.
- Concatenación de datos en `extra_df` al conjunto de datos inicial (df). Se utiliza la función `concat` de Pandas.
- Creación de la variable `version`, la cual almacena el momento temporal en el cual fueron ingestados los datos.

Posterior al proceso de transformación, obtenemos un *DataFrame* con 6.301.090 filas y 5 columnas . A continuación puede ser visualizada su estructura de salida:

	geo_id	indicator_id	year	measure	version	
48	xxa	SP.ADO.TFRT	1960	1.477086e+02	2021-07-07 12:45:28.987704	
54	xxa	SP.POP.DPND	1960	9.031121e+01	2021-07-07 12:45:28.987704	
55	xxa	SP.POP.DPND.OL	1960	7.854604e+00	2021-07-07 12:45:28.987704	
56	xxa	SP.POP.DPND.YG	1960	8.204993e+01	2021-07-07 12:45:28.987704	
88	xxa	ER.FSH.AQUA.MT	1960	0.000000e+00	2021-07-07 12:45:28.987704	
...
300	twn	SP.POP.TOTL	2020	2.381700e+07	2021-07-07 12:45:28.987704	
301	twn	SP.POP.TOTL.FE.IN	2020	1.198200e+07	2021-07-07 12:45:28.987704	
302	twn	SP.POP.TOTL.FE.ZS	2020	5.030860e+01	2021-07-07 12:45:28.987704	
303	twn	SP.POP.TOTL.MA.IN	2020	1.183400e+07	2021-07-07 12:45:28.987704	
304	twn	SP.POP.TOTL.MA.ZS	2020	4.968720e+01	2021-07-07 12:45:28.987704	

Figura 122. Formato final tabla de Indicadores de Desarrollo Mundial

13. Infonavit

A continuación se presenta el detalle de los pipelines que procesan datos proporcionados por el Instituto del Fondo Nacional de la Vivienda para los Trabajadores (INFONAVIT) y hacen referencia a montos de créditos y subsidios Infonavit entregados a la población mexicana para la adquisición de viviendas.

13.1. Infonavit_age_range_credits

Descripción General y Ejecución

El pipeline *age_range_credits.py* procesa los datos facilitados por el Instituto del Fondo Nacional de la Vivienda para los Trabajadores (INFONAVIT) que toman relación con los créditos otorgados según el rango etario de los beneficiarios.

Los datos fueron facilitados por la organización, y se almacenaron en Google Storage para su posterior procesamiento.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, las librerías *pandas*, *numpy*, y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

(Python) `~/data-etl/etl/infonavit/$ python age_range_credits.py`

(Bamboo-cli) `~/data-etl/etl/infonavit/$ bamboo-cli --folder . --entry age_range_credits`

Descarga y lectura de datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y a la fecha de creación de este documento se identificaron 1.380 filas y 11 columnas. A continuación, se visualiza una muestra de la estructura inicial de los datos.

FH_PRCO	FH_CRTE	ID_GRPO_EDAD	ID_PRDO	ID_ANIO	...	CRDS_FRMS	IMPE_CHQE	MINTO_CRDO_INF	NMRO_SBSS	MNTO_SBSS
2021-09-05 00:10:05	2021-07-31	R01	201201	2012	...	131	2.550812e+07	25202072.27	2	63131.49
2021-09-05 00:10:05	2021-07-31	R01	201202	2012	...	367	7.229754e+07	72969411.03	205	9577110.38
2021-09-05 00:10:05	2021-07-31	R01	201203	2012	...	437	8.713927e+07	88528977.08	229	10895277.14
2021-09-05 00:10:05	2021-07-31	R01	201204	2012	...	287	5.819311e+07	58545986.59	108	5209839.15
2021-09-05 00:10:05	2021-07-31	R01	201205	2012	...	337	7.099124e+07	71623681.77	113	5192185.57

Figura 123. Formato inicial tabla de datos Infonavit (rango de edad)

Transformación y limpieza

En esta etapa se desarrollan los siguientes pasos:

- Renombre de columnas a conveniencia.
- Creación de la variable *month_id*, la cual se constituye a partir de la agregación de las variables *year* y *month*.
- Reemplazo de valores en la columna *age_range_id* por sus *ids*.
- Selección de columnas a utilizar.
- Transformación del tipo de dato en las columnas: *age_range_id*, *month_id*, *credits_number*, y *subsidy_number* a *int*.
- Creación de la variable *nat_id*, la cual denotará el nivel de desagregación nacional.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 1.380 filas y 8 columnas. A continuación puede ser visualizada la estructura de salida:

age_range_id	month_id	credits_number	check_amount	infonavit_credit_amount	subsidy_number	subsidy_amount	nat_id
1	201201	131	2.550812e+07	25202072.27	2	63131.49	mex
1	201202	367	7.229754e+07	72969411.03	205	9577110.38	mex
1	201203	437	8.713927e+07	88528977.08	229	10895277.14	mex
1	201204	287	5.819311e+07	58545986.59	108	5209839.15	mex
1	201205	337	7.099124e+07	71623681.77	113	5192185.57	mex

Figura 124. Formato final tabla de datos Infonavit (rango de edad)

13.2. Infonavit_credit_line

Descripción General y Ejecución

El pipeline *credit_line.py* procesa los datos facilitados por el Instituto del Fondo Nacional de la Vivienda para los Trabajadores (INFONAVIT) que toman relación con el tipo de crédito otorgado. Los datos fueron facilitados por la organización, y se almacenaron en Google Storage para su posterior procesamiento.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, las librerías *pandas*, *numpy*, y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

- (Python) `~/data-etl/etl/infonavit/$ python credit_line.py`
- (Bamboo-cli) `~/data-etl/etl/infonavit/$ bamboo-cli --folder . --entry credit_line`

Descarga y lectura de datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y al momento del desarrollo de esta documentación se identificaron 575 filas y 11 columnas. A continuación, se visualiza la estructura inicial de los datos:

FH_PRCO	FH_CRTE	ID_TIPO_LNEA_TIPO_VVNA	ID_PRDO	...	IMPE_CHQE	MNTO_CRD0_INFT	NMRO_SBSS	MNTO_SBSS
2021.09.05 00:10:05	2021.07.31	L2ETE	201508	...	4.075829e+09	2.864387e+09	314	1.851091e+07
2021.09.05 00:10:05	2021.07.31	L2ETE	201507	...	4.354818e+09	3.059489e+09	255	1.490076e+07
2021.09.05 00:10:05	2021.07.31	L2ETE	201506	...	4.810476e+09	3.371802e+09	1237	6.950963e+07
2021.09.05 00:10:05	2021.07.31	L2ETE	201505	...	4.203254e+09	3.019503e+09	2027	1.089263e+08
2021.09.05 00:10:05	2021.07.31	L2ETE	201504	...	3.782488e+09	2.798966e+09	2549	1.381086e+08

Figura 125. Formato inicial tabla de datos Infonavit (líneas de crédito)

Transformación y limpieza

En esta etapa se desarrollan los siguientes pasos:

- Renombre de columnas a conveniencia.
- Creación de la variable *month_id*, la cual se constituye a partir de la agregación de las variables *year* y *month*.
- Reemplazo de valores en la columna *credit_line_id* por sus *ids*.
- Selección de columnas a utilizar.
- Transformación del tipo de dato en las columnas: *credit_line_id*, *month_id*, *credit_number*, y *subsidy_number* a *int*.
- Creación de la variable *nat_id*, la cual denotará el nivel de desagregación nacional.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 575 filas y 8 columnas. A continuación puede ser visualizada su estructura de salida:

credit_line_id	month_id	credit_number	check_amount	infonavit_credit_amount	subsidy_number	subsidy_amount	nat_id
1	201508	10417	4.075829e+09	2.864387e+09	314	1.851091e+07	mex
1	201507	11125	4.354818e+09	3.059489e+09	255	1.490076e+07	mex
1	201506	12506	4.810476e+09	3.371802e+09	1237	6.950963e+07	mex
1	201505	11115	4.203254e+09	3.019503e+09	2027	1.089263e+08	mex
1	201504	10310	3.782488e+09	2.798966e+09	2549	1.381086e+08	mex

Figura 126. Formato final tabla de datos Infonavit (líneas de crédito)

13.3. Entity_credits

Descripción General y Ejecución

El pipeline `entity_credits.py` procesa los datos facilitados por el Instituto del Fondo Nacional de la Vivienda para los Trabajadores (INFONAVIT) que toman relación con los créditos otorgados por Entidad Federativa. Los datos fueron facilitados por la organización, y se almacenaron en Google Storage para su posterior procesamiento.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, las librerías `pandas`, `numpy`, y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

- (Python) `~/data-etl/etl/infonavit/$ python entity_credits.py`
- (Bamboo-cli) `~/data-etl/etl/infonavit/$ bamboo-cli --folder . --entry entity_credits`

Descarga y lectura de datos

Utilizando el módulo de descarga que facilita la librería `bamboo-lib` se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato `.csv` y al momento del desarrollo de esta documentación se identificaron 3.795 filas y 11 columnas. A continuación, se visualiza la estructura inicial de los datos:

FH_PRCO	FH_CRTE	CV_ENTD_FDRA	ID_PRDO	ID_ANIO	...	CRDS_FRMS	IMPE_CHOE	MNTO_CRDO_INF	NMRO_SBSS	MNTO_SBSS
2021-09-05 00:10:05	2021-07-31	1000	202107	2021	...	913	3.244137e+08	2.588586e+08	0	0.000000e+00
2021-09-05 00:10:05	2021-07-31	1000	202106	2021	...	927	3.468622e+08	2.595214e+08	0	0.000000e+00
2021-09-05 00:10:05	2021-07-31	1000	202105	2021	...	1039	3.682306e+08	2.936209e+08	0	0.000000e+00
2021-09-05 00:10:05	2021-07-31	1000	202104	2021	...	891	2.985938e+08	2.423346e+08	0	0.000000e+00
2021-09-05 00:10:05	2021-07-31	1000	202103	2021	...	1070	3.464668e+08	2.909128e+08	0	0.000000e+00

Figura 127. Formato inicial tabla de datos Infonavit (créditos entregados por estado)

Transformación y limpieza

En esta etapa se desarrollan los siguientes pasos:

- Renombre de columnas a conveniencia.
- Reformulación de la variable *id_entity*. Se construye con los *ids* referentes a la entidad en cuestión.
- Eliminación de datos a nivel nacional. Dichos datos poseen el valor 99 en la columna *id_entity*.
- Creación de la variable *month_id*, la cual se constituye a partir de la agregación de las variables *year* y *month*.
- Selección de columnas a utilizar.
- Transformación del tipo de dato en las columnas: *id_entity*, *month_id*, *credit_number*, y *subsidy_number* a *int*.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 3.680 filas y 7 columnas. A continuación puede ser visualizada su estructura de salida:

id_entity	month_id	credit_number	check_amount	infonavit_credit_amount	subsidy_number	sbsidy_amount
1	202107	913	3.244137e+08	2.588586e+08	0	0.00
1	202106	927	3.468622e+08	2.595214e+08	0	0.00
1	202105	1039	3.682306e+08	2.936209e+08	0	0.00
1	202104	891	2.985938e+08	2.423346e+08	0	0.00
1	202103	1070	3.464668e+08	2.909128e+08	0	0.00

Figura 128. Formato final tabla de datos Infonavit (créditos entregados por estado)

13.4. house_credits

Descripción General y Ejecución

El pipeline *house_credits.py* procesa los datos facilitados por el Instituto del Fondo Nacional de la Vivienda para los Trabajadores (INFONAVIT) que toman relación con los créditos otorgados por tipo de vivienda. Los datos fueron facilitados por la organización, y se almacenaron en Google Storage para su posterior procesamiento.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, las librerías *pandas*, *numpy*, y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

- (Python) `~/data-etl/etl/infonavit/$ python house_credits.py`
- (Bamboo-cli) `~/data-etl/etl/infonavit/$ bamboo-cli --folder . --entry house_credits`

Descarga y lectura de datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y al momento del desarrollo de esta documentación se identificaron 805 filas y 11 columnas. A continuación, se visualiza la estructura inicial de los datos:

FH_PRCO	FH_CRTE	ID_CLSN_VVNA	ID_PRDO	ID_ANIO	...	CRDS_FRMS	IMPE_CHOE	MNTO_CRDO_INFIT	NMRO_SBSS	MNTO_SBSS
2021-09-05 00:10:05	2021-07-31	SD	201607	2016	...	0	0.000000e+00	0.000000e+00	0	0.00
2021-09-05 00:10:05	2021-07-31	SD	201606	2016	...	0	0.000000e+00	0.000000e+00	0	0.00
2021-09-05 00:10:05	2021-07-31	SD	201605	2016	...	0	0.000000e+00	0.000000e+00	0	0.00
2021-09-05 00:10:05	2021-07-31	SD	201604	2016	...	0	0.000000e+00	0.000000e+00	0	0.00
2021-09-05 00:10:05	2021-07-31	RSDLP	202001	2020	...	168	2.088607e+08	5.346172e+07	0	0.00

Figura 129. Formato inicial tabla de datos Infonavit (créditos por tipo de vivienda)

Transformación y limpieza

En esta etapa se desarrollan los siguientes pasos:

- Renombre de columnas a conveniencia.
- Creación de la variable *month_id*, la cual se constituye a partir de la agregación de las variables year y month.
- Reemplazo de valores en la columna *id_viv* por sus *ids*.
- Selección de columnas a utilizar.
- Transformación del tipo de dato en las columnas: *id_viv*, *month_id*, *credit_number*, y *subsidy_number* a *int*.
- Creación de la variable *nat_id*, la cual denotará el nivel de desagregación nacional.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 805 filas y 8 columnas. A continuación puede ser visualizada su estructura de salida:

id_viv	month_id	credit_number	check_amount	infonavit_credit_amount	subsidy_number	subsidy_amount	nat_id
6	201607	0	0.000000e+00	0.000000e+00	0	0.00	mex
6	201606	0	0.000000e+00	0.000000e+00	0	0.00	mex
6	201605	0	0.000000e+00	0.000000e+00	0	0.00	mex
6	201604	0	0.000000e+00	0.000000e+00	0	0.00	mex
5	202001	168	2.088607e+08	5.346172e+07	0	0.00	mex

Figura 130. Formato final tabla de datos Infonavit (créditos por tipo de vivienda)

13.5. income_level_credits

Descripción General y Ejecución

El pipeline `income_level_credits.py` procesa los datos facilitados por el Instituto del Fondo Nacional de la Vivienda para los Trabajadores (INFONAVIT) que toman relación con los créditos y subsidios Infonavit otorgados según rangos de ingresos de los beneficiarios. Los datos fueron facilitados por la organización, y se almacenaron en Google Storage para su posterior procesamiento.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, las librerías `pandas`, `numpy`, y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

- (Python) `~/data-etl/etl/infonavit/$ python income_level_credits.py`
- (Bamboo-cli) `~/data-etl/etl/infonavit/$ bamboo-cli --folder . --entry income_level_credits`

Descarga y lectura de datos

Utilizando el módulo de descarga que facilita la librería `bamboo-lib` se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato `.csv` y al momento del desarrollo de esta documentación se identificaron 3.105 filas y 11 columnas. A continuación, se visualiza la estructura inicial de los datos:

FH_PRCO	FH_CRTE	TX_ID_NVEL_INGO	ID_PRDO	...	IMPE_CHQE	MNT0_CRDO_INF	NMRO_SBS	MNT0_SBS
2021.09.05 00:10:05	2021.07.31	NVL01	201201	...	2.587812e+07	2.179703e+07	22	1296602.23
2021.09.05 00:10:05	2021.07.31	NVL01	201202	...	1.361178e+08	1.275066e+08	694	40863887.99
2021.09.05 00:10:05	2021.07.31	NVL01	201203	...	2.147057e+08	2.015620e+08	1080	63903900.85
2021.09.05 00:10:05	2021.07.31	NVL01	201204	...	1.247520e+08	1.147878e+08	566	33078390.26
2021.09.05 00:10:05	2021.07.31	NVL01	201205	...	1.572941e+08	1.454396e+08	685	39567123.58

Figura 131. Formato inicial tabla de datos Infonavit (créditos por ingreso de beneficiarios)

Transformación y limpieza

En esta etapa se desarrollan los siguientes pasos:

- Renombre de columnas a conveniencia.
- Creación de la variable *month_id*, la cual se constituye a partir de la agregación de las variables *year* y *month*.
- Reemplazo de valores en la columna *income_id* por sus *ids*.
- Selección de columnas a utilizar.
- Transformación del tipo de dato en las columnas: *income_id*, *month_id*, *credit_number*, y *subsidy_number* a *int*.
- Creación de la variable *nat_id*, la cual denotará el nivel de desagregación nacional.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 3.105 filas y 8 columnas . A continuación puede ser visualizada su estructura de salida:

income_id	month_id	credit_number	check_amount	infonavit_credit_amount	subsidy_number	subsidy_amount	nat_id
1	201201	242	2.587812e+07	2.179703e+07	22	1296602.23	mex
1	201202	918	1.361178e+08	1.275066e+08	694	40863887.99	mex
1	201203	1392	2.147057e+08	2.015620e+08	1080	63903900.85	mex
1	201204	908	1.247520e+08	1.147878e+08	566	33078390.26	mex
1	201205	1230	1.572941e+08	1.454396e+08	685	39567123.58	mex

Figura 132. Formato final tabla de datos Infonavit (créditos por ingreso de beneficiarios)

13.6. payment_entity_credits

Descripción General y Ejecución

El pipeline *payment_entity_credits.py* procesa los datos facilitados por el Instituto del Fondo Nacional de la Vivienda para los Trabajadores (INFONAVIT) que entregan información de las contribuciones realizadas por las compañías registradas en Infonavit. Los datos fueron facilitados por la organización, y se almacenaron en Google Storage para su posterior procesamiento.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, las librerías *pandas*, *numpy*, y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

- (Python) `~/data-etl/etl/infonavit/$ python payment_entity_credits.py`
- (Bamboo-cli) `~/data-etl/etl/infonavit/$ bamboo-cli --folder . --entry payment_entity_credits`

Descarga y lectura de datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y al momento del desarrollo de esta documentación se identificaron 94 filas y 9 columnas. A continuación, se visualiza la estructura inicial de los datos:

ID_PRDO	ANIO_PAGO	BMSE_PAGO	FH_PRCO	CV_ENTD_FDRA	NU_EMPS	PAGO_AMRN	PAGO_APRN_CON_CRD0	PAGO_APRN_SIN_CRD0
200505	2005	5	2021-05-17 12:33:27	99999	754311	4.469337e+09	1.301324e+09	5.672244e+09
200506	2005	6	2021-05-17 12:33:27	99999	756954	4.556845e+09	1.289409e+09	5.620854e+09
200601	2006	1	2021-05-17 12:33:27	99999	752768	4.644184e+09	1.323898e+09	5.671915e+09
200602	2006	2	2021-05-17 12:33:27	99999	753314	4.846824e+09	1.349781e+09	5.714510e+09
200603	2006	3	2021-05-17 12:33:27	99999	760070	5.010211e+09	1.412464e+09	5.989573e+09

Figura 133. Formato inicial tabla de datos Infonavit (contribuciones de compañías)

Transformación y limpieza

En esta etapa se desarrollan los siguientes pasos:

- Renombre de columnas a conveniencia.
- Creación de la variable *bimester_id*, la cual se constituye a partir de la agregación de las variables *year* y *bimester*.
- Reestructuración de la variable *entity_id*. Se estructura con los *ids* de las entidades federativas. (Este paso se realiza para procesar la columna con datos de la entidad, sin embargo, al momento de la redacción de esta documentación, la columna no presentaba valores diferentes a 99999, por lo que al final del pipeline, esta columna es omitida).
- Selección de columnas a utilizar.
- Transformación del tipo de dato en las columnas: *bimester_id*, *entity_id*, y *number_companies* a *int*.
- Creación de la variable *nat_id*, la cual denotará el nivel de desagregación nacional.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 94 filas y 6 columnas. A continuación puede ser visualizada su estructura de salida:

bimester_id	number_companies	harmonization_payment	payment_contribution_with_credit	payment_contribution_without_credit	nat_id
200505	754311	4.469337e+09	1.301324e+09	5.672244e+09	mex
200506	756954	4.556845e+09	1.289409e+09	5.620854e+09	mex
200601	752768	4.644184e+09	1.323898e+09	5.671915e+09	mex
200602	753314	4.846824e+09	1.349781e+09	5.714510e+09	mex
200603	760070	5.010211e+09	1.412464e+09	5.989573e+09	mex

Figura 134. Formato final tabla de datos Infonavit (contribuciones de compañías)

13.7. product_credits

Descripción General y Ejecución

El pipeline *product_credits.py* procesa los datos facilitados por el Instituto del Fondo Nacional de la Vivienda para los Trabajadores (INFONAVIT) que toman relación con los créditos otorgados por tipo de producto. Los datos fueron facilitados por la organización, y se almacenaron en Google Storage para su posterior procesamiento.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, las librerías *pandas*, *numpy*, y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

- (Python) `~/data-etl/etl/infonavit/$ python product_credits.py`
- (Bamboo-cli) `~/data-etl/etl/infonavit/$ bamboo-cli --folder . --entry product_credits`

Descarga y lectura de datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv y al momento del desarrollo de esta documentación se identificaron 920 filas y 11 columnas. A continuación, se visualiza la estructura inicial de los datos:

	FH_PRCO	FH_CRTE	CLSPRDOPR	ID_PRDO	ID_ANIO	...	CRDS_FRMS	IMPE_CHQE	MNTO_CRD0_INFT	NMRO_SBSS	MNTO_SBSS
2021-09-05 00:10:05	2021-07-31	6	201209	2012	...		0	0.00000e+00	0.000000e+00	0	0.000000e+00
2021-09-05 00:10:05	2021-07-31	6	201208	2012	...		0	0.00000e+00	0.000000e+00	0	0.000000e+00
2021-09-05 00:10:05	2021-07-31	6	201207	2012	...		0	0.00000e+00	0.000000e+00	0	0.000000e+00
2021-09-05 00:10:05	2021-07-31	6	201206	2012	...		0	0.00000e+00	0.000000e+00	0	0.000000e+00
2021-09-05 00:10:05	2021-07-31	6	201205	2012	...		0	0.00000e+00	0.000000e+00	0	0.000000e+00

Figura 135. Formato inicial tabla de datos Infonavit (créditos por tipo de producto)

Transformación y limpieza

En esta etapa se desarrollan los siguientes pasos:

- Renombre de columnas a conveniencia.
- Creación de la variable *month_id*, la cual se constituye a partir de la agregación de las variables *year* y *month*.
- Selección de columnas a utilizar.
- Transformación del tipo de dato en las columnas: *product_id*, *month_id*, *credit_number*, y *subsidy_number* a *int*.
- Creación de la variable *nat_id*, la cual denotará el nivel de desagregación nacional.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 920 filas y 8 columnas. A continuación puede ser visualizada su estructura de salida:

product_id	month_id	credit_number	check_amount	infonavit_credit_amount	subsidy_number	subsidy_amount	nat_id
6	201209	0	0.000000e+00	0.000000e+00	0	0.000000e+00	mex
6	201208	0	0.000000e+00	0.000000e+00	0	0.000000e+00	mex
6	201207	0	0.000000e+00	0.000000e+00	0	0.000000e+00	mex
6	201206	0	0.000000e+00	0.000000e+00	0	0.000000e+00	mex
6	201205	0	0.000000e+00	0.000000e+00	0	0.000000e+00	mex

Figura 136. Formato final tabla de datos Infonavit (créditos por tipo de producto)

14. Gasto Público

A continuación se presenta el detalle del pipeline que procesa datos relacionados con el Presupuesto de Egresos de la Federación (PEF) y entrega información del presupuesto aprobado y ejecutado anualmente según diferentes clasificaciones del gasto.

14.1. budget_transparency_annual_pipeline

Descripción General y Ejecución

El pipeline *budget_transparency_annual_pipeline.py* realiza el procesamiento de los datos de transparencia presupuestaria, los cuales contienen información del presupuesto aprobado y ejecutado anualmente según diferentes categorías del gasto.

Los datos fueron facilitados por la Secretaría de Economía, y se almacenaron en Google Storage para su posterior procesamiento.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, la librería *pandas* y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Antes de ingestar los datos en una base de datos de *Clickhouse*, estos son llamados desde Google Storage para ser sometidos a un proceso de limpieza y transformación.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

```
(Bamboo-cli) ~/data_etl/$ bamboo-cli --folder etl/budget_transparency/ --entry budget_transparency_annual_pipeline
```

Procesamiento

1. Descarga

Utilizando el módulo *WildCardDownloadStep* que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .xlsx (excel). Existe un archivo por año comenzando en el 2013 siendo el último año disponible el 2020 (hasta el momento de la elaboración de esta documentación), por lo que 8 archivos son descargados y procesados.

2. Lectura

Una vez descargados todos los archivos, estos son leídos en el paso *ReadSteap* y almacenados en un mismo *DataFrame* de Pandas del cual se conservan solo las columnas relacionadas a *ciclo*, *ramo*, *tipo de gasto*, *entidad federativa*, *función* y *montos aprobados* y *ejecutados*. En este paso también se formatean los valores de tipo texto presentes en el *DataFrame*, se eliminan filas con identificadores nulos y se reemplazan los identificadores de entidades federativas correspondientes a lugares "en el extranjero". El *DataFrame* resultante tiene 2.242.127 filas y 15 columnas, la imagen muestra sus primeras filas.

ciclo	id_ramo	desc_ramo	id_tipogasto	desc_tipogasto	gpo_funcional	...	id_subfuncion	desc_subfuncion	id_entidad_federativa	entidad_federativa	monto_ejercicio	monto_aprobado
2013.0	2	Presidencia de la república	1.0	Gasto corriente	1.0	...	1.0	Presidencia / gubernatura	9	Distrito Federal	16704186.38	14023385.0
2013.0	2	Presidencia de la república	1.0	Gasto corriente	1.0	...	1.0	Presidencia / gubernatura	9	Distrito Federal	72254.49	95100.0
2013.0	2	Presidencia de la república	1.0	Gasto corriente	1.0	...	1.0	Presidencia / gubernatura	9	Distrito Federal	510542.93	391517.0
2013.0	2	Presidencia de la república	1.0	Gasto corriente	1.0	...	1.0	Presidencia / gubernatura	9	Distrito Federal	5663724.50	1564355.0
2013.0	2	Presidencia de la república	1.0	Gasto corriente	1.0	...	1.0	Presidencia / gubernatura	9	Distrito Federal	2247893.94	1952128.0

Figura 137. Formato inicial tabla de datos gasto público

3. Creación de dimensión "Función"

En este pipeline, para la creación de las dimensiones se utiliza el *DataFrame* generado en el paso *Lectura*. En este *DataFrame* se agrupa por los valores únicos de las columnas relacionadas al grupo funcional. Luego, se trabaja sobre los datos de tipo texto (*strings*), donde se corrige ortografía y se obtienen sus traducciones a inglés. La dimensión resultante está compuesta por 92 filas y 10 columnas, las cuales corresponden a los 92 tipos de subfunciones que existen, éstas se agrupan en 28 funciones y 4 grupos funcionales.

4. Creación de dimensión "Ramo"

Este paso, al igual que todos los pasos de creación de dimensiones, sigue la lógica del paso 3. La dimensión resultante está compuesta por 57 filas y 4 columnas, correspondientes a los 57 tipos de ramos existentes.

5. Creación de dimensión "Tipo de Gasto"

La dimensión resultante está compuesta por 10 filas y 4 columnas, correspondientes a los 10 tipos de ramos existentes.

6. Creación del cubo

Primero se importan los *DataFrames* resultantes de todos los pasos anteriores usando la funcionalidad de *Bamboo* para poder usar resultados de pasos no consecutivos como *input* de un paso cualquiera. Luego, se juntan todos los *DataFrames* importados uniéndose por sus nombres, para así poder incluir los *ID's* provenientes de las dimensiones. Posteriormente, se renombran columnas y se eliminan las que no son necesarias, se reemplazan por 0 todos los valores nulos o vacíos de las columnas de montos y se ajustan los tipos de datos de las columnas que lo requieren. El *DataFrame* resultante queda compuesto por 2.242.127 filas y 9 columnas. La imagen muestra las primeras filas del *DataFrame*.

year	ent_id	functional_group_id	function_id	subfunction_id	department_id	exp_type_id	amount_executed	amount_approved
2013	9		1	103	10301	45	1	16704186.38
2013	9		1	103	10301	45	1	72254.40
2013	9		1	103	10301	45	1	510542.93
2013	9		1	103	10301	45	1	5663724.50
2013	9		1	103	10301	45	1	2247893.94

Figura 138. Formato final tabla de datos gasto público

15. Parques Industriales

A continuación se presenta el detalle del pipeline que procesa datos de parques industriales de México proporcionados por la Asociación Mexicana de Parques Industriales Privados (AMPIP). Contiene el listado de parques industriales registrados en AMPIP, tipo de parque y número de empresas en su interior.

15.1. industrial_parks

Descripción General y Ejecución

El pipeline *industrial_park.py* es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los registros de parques industriales de México, proporcionados por la Asociación Mexicana de Parques Industriales Privados (AMPIP).

Los datos fueron facilitados por AMPIP y se almacenaron en Google Storage para su posterior procesamiento.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, la librería *pandas* y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

```
(Phyton) ~/data-etl/etl/industrial_parks/$ python industrial_park.py
```

Descarga

Los datos fueron enviados internamente por AMPIP en un archivo excel (.xlsx) y almacenados en un compartimiento privado de GCP storage, se estableció una

conexión privada validada con credenciales fuertemente encriptadas, para ejecutar una descarga segura hasta el servidor de procesamiento de los datos.

Lectura

Una vez descargados los datos, estos se leen y almacenan en un *DataFrame* de la librería pandas. Según la última actualización enviada a la fecha de creación de esta documentación, la estructura del archivo constaba de 855 filas y 35 columnas de información. La siguiente imagen muestra las primeras filas y algunas columnas del *DataFrame* inicial obtenido:

f1d	NOMBRE	ADMINISTRADOR	DESARROLLADOR	SOCIO	TIPO_INVERSION	PLANMTRO	VIALIDADES	DESARROLLO	ADMINISTRACION	...	MASTER_REGION_PI
735	Parque Industrial Bruno Pagliai		NaN	NaN	0.0	Pública	1.0	1.0	5.0	0.0 ...	Sureste
623	Parque Industrial Logistik		Artha Capital	Artha Capital	1.0	Privada	1.0	1.0	2.0	1.0 ...	Centro Occidente
387	Parque industrial Tizayuca	COFOIN, Gov't State of Hidalgo\ñ		NaN	1.0	Pública	1.0	1.0	4.0	0.0 ...	Centro
341	Parque Tecnológico Castro del Río		Marabis	Marabis	1.0	Privada	1.0	1.0	4.0	1.0 ...	Centro Occidente
556	Parque Industrial Puebla 2000		NaN	Secretaría de Competitividad, Trabajo y Desar...	1.0	NaN	1.0	1.0	4.0	1.0 ...	Centro

Figura 139. Formato inicial tabla de datos de parques industriales

Transformación y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Esta etapa sigue el siguiente proceso:

- Se crea un diccionario de datos para reemplazar los nombres de los parques por sus *ids*.
- Se estandarizan los nombres de columnas dejándolas en minúsculas y eliminando espacios adicionales.
- Se seleccionan las columnas de interés: *cvegeo*, *master_nombre_ampip*, *master_tipo_de_asentamiento*, *número_de_empresas*, se renombran y se eliminan duplicados.
- Se crea un diccionario para reemplazar el tipo de parque por su *id*.

- Se reemplazan los nombres y tipos de parques por sus *ids* utilizando los diccionarios creados anteriormente.
- Se crea la columna *count_parks* con el valor 1. Esta columna servirá para hacer el conteo de parques en diferentes niveles de agregación de la data.
- Se reemplazan los valores nulos por ceros en la columna *companies*

Al terminar el proceso de transformación y limpieza, se obtiene una tabla en formato tidy lista para su ingestión. A continuación, se muestra una tabla ejemplo de la versión final de la misma antes de su ingestión, la cual se compone de 852 filas y 5 columnas.

	mun_id	park_name_id	park_type_id	companies	count_parks
0	30193	734	2	97	1
1	24050	622	1	93	1
2	13069	387	2	88	1
3	11017	341	1	73	1
4	21114	555	1	59	1
5	29031	726	1	57	1
6	24028	619	1	48	1
7	2004	76	1	46	1

Figura 140. Formato final tabla de datos de parques industriales

16. Remesas

A continuación se detallan los pipelines que procesan datos relacionados con remesas. Estos datos son obtenidos desde el sitio web de [BANXICO](#) y contienen información de los países de origen/destino de las remesas, tipos de operaciones y montos recibidos a nivel de municipios.

16.1. banxico_countries_remittances

Descripción General y Ejecución

El pipeline *countries_remittances.py* procesa datos a nivel nacional de las remesas provenientes o enviadas a países extranjeros. Los datos fueron descargados desde el [Banco de México](#) (BANXICO) y almacenados en Google Storage, desde donde se realiza la descarga de datos. Posteriormente, se realiza un proceso de limpieza y transformación para ingestar los datos en una base de datos de Clickhouse.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, la librería *pandas* y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

(Python) `~/data_etl/etl/banxico/$ python countries_remittances.py`

(Bamboo CLI) `~/data_etl/etl/banxico/$ bamboo-cli --folder . --entry countries_remittances`

Descarga

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en

formato .csv; y, en cuanto a su estructura, posee 15.120 filas y 44 columnas. A continuación se presenta su estructura inicial:

	Título	Periodo disponible	...	01/04/2021	01/07/2021
País de origen de los ingresos por remesas, Total	Ene-Mar 2013 – Jul-Sep 2021	...	13031.627083	13686.837095	
País de origen de los ingresos por remesas, Af...	Ene-Mar 2013 – Jul-Sep 2021	...	0.001866	0.000422	
País de origen de los ingresos por remesas, Al...	Ene-Mar 2013 – Jul-Sep 2021	...	0.004203	0.002285	
País de origen de los ingresos por remesas, Al...	Ene-Mar 2013 – Jul-Sep 2021	...	8.852029	12.174753	
País de origen de los ingresos por remesas, An...	Ene-Mar 2013 – Jul-Sep 2021	...	0.000323	0.000799	

Figura 141. Formato inicial tabla de datos remesas según país de origen/destino

Transformación y limpieza

En esta etapa se sigue el siguiente proceso:

- Lectura de datos. Aquí se transforman los nombres de variables a minúsculas y se reemplazan espacios por guión bajo para estandarizar los nombres.
- Transformación de los datos a formato *tidy*. Para ello, se utiliza la función *melt* de Pandas.
- Creación de la variable *move*. Dicha variable tendrá valor igual a *origin* o *destination*, dependiendo si las remesas se envían desde México o provienen desde el extranjero.
- Selección de filas cuyo valor en la columna *iso3*, referente al país de origen, no posea el string *Total*.
- Transformación de la variable temporal (trimestral) a formato: YYYY-MM
- Reemplazo de nombres de países por su código *iso3*.
- Asignación de meses a su trimestre correspondiente. Dicha transformación se efectúa con la ayuda del diccionario *trimester_map*.
- Multiplicación de los valores de remesas por un millón. Esto se realiza ya que los valores vienen en millones de dólares, y son transformados a dólares para mayor facilidad de trabajo.
- Se redondean los valores de remesas a dos decimales.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 15.050 filas y 4 columnas. A continuación puede ser visualizada su estructura:

iso3	quarter_id	remittance_amount	move
afg	20131	960.0	1
alb	20131	5375.0	1
deu	20131	808244.0	1
ago	20131	49502.0	1
aia	20131	0.0	1

Figura 142. Formato final tabla de datos de remesas según país de origen/destino

16.2. banxico_income_remittances

Descripción General y Ejecución

El pipeline *income_remittances.py* procesa datos a nivel nacional del ingreso por remesas según tipo de operación. Dichos datos presentan una desagregación mensual. Los datos fueron descargados desde el [Banco de México](#) (BANXICO) y almacenados en Google Storage, desde donde se realiza la descarga de datos. Posteriormente, se realiza un proceso de limpieza y transformación para ingestar los datos en una base de datos de Clickhouse.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, la librería *pandas* y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

(Python) `~/data_etl/etl/banxico/$ python income_remittances.py`

(Bamboo CLI) `~/data_etl/etl/banxico/$ bamboo-cli --folder . --entry income_remittances`

Descarga

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv; y, en cuanto a su estructura, posee 15 filas y 330 columnas. A continuación se presenta su estructura inicial:

	Título	Periodo disponible	Periodicidad	...	01/07/2021	01/08/2021	01/09/2021
	Remesas Familiares, Total	Ene 1995 - Sep 2021	Mensual	...	4540.255252	4743.564412	4403.017431
	Remesas Familiares, Money Orders	Ene 1995 - Sep 2021	Mensual	...	19.553449	17.616829	13.778504
	Remesas Familiares, Cheques Personales	Ene 1995 - Sep 2021	Mensual	...	0.000000	0.000000	0.000000
	Remesas Familiares, Transferencias Electrónicas	Ene 1995 - Sep 2021	Mensual	...	4492.644199	4706.727432	4371.597559
	Remesas Familiares, Efectivo y Especie	Ene 1995 - Sep 2021	Mensual	...	28.057604	19.220151	17.641368
	Remesas Familiares, Total	Ene 1995 - Sep 2021	Mensual	...	11626.778000	12261.513000	11544.918000

Figura 143. Formato inicial tabla de datos remesas según tipo de operación

Transformación y limpieza

En esta etapa se sigue el siguiente proceso:

- Lectura de datos. Aquí se transforman los nombres de variables a minúsculas y se reemplazan espacios por guión bajo para estandarizar los nombres.
- Transformación a formato *tidy*. Para ello, se utiliza la función *melt* de Pandas.
- Selección y renombre de columnas a utilizar.
- Selección de filas cuyo valor en la columna *operation_type*, referente al tipo de operación, no posea el string *Total* o *Promedio*.
- Creación de dos *DataFrames*. El primero seleccionará datos que se relacionen con el monto de las remesas (*df_usd*) y el segundo, con la cantidad de remesas enviadas (*df_ope*).
- Ambos *DataFrames* son posteriormente unidos por la función *merge* de Pandas, siendo unidos bajo criterios de tipo de operación y periodo de tiempo. De esta forma se dividen dichas medidas en dos variables.
- Transformación de la variable temporal (trimestral) a formato: YYYY-MM.
- Reemplazo de los tipos de operación por sus *ids*.
- Multiplicación de los valores de monto de remesas por un millón. Esto se realiza ya que los valores vienen en millones de dólares, y son transformados a dólares para mayor facilidad de trabajo.
- Multiplicación de los valores de cantidad de remesas por un mil. Esto se realiza ya que los valores vienen en miles de operaciones, y son transformados a número de operaciones para mayor facilidad de trabajo.
- Se redondean los valores de monto de remesas a dos decimales.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 1.284 filas y 4 columnas. A continuación puede ser visualizada su estructura:

operation_type	month_id	remittance_amount	remittance_quantity
1	199501	9.814270e+07	344927.0
2	199501	2.064788e+06	5435.0
3	199501	1.328102e+08	<u>380433.0</u>
4	199501	2.155467e+07	37098.0
1	199502	9.079906e+07	307813.0

Figura 144. Formato final tabla de datos de remesas según tipo de operación

16.3. banxico_mun_income_remittances

Descripción General y Ejecución

El pipeline *mun_income_remittances.py* procesa datos del total de remesas a nivel de municipio. Dichos datos se presentan con periodicidad trimestral. Los datos fueron descargados desde el [Banco de México](#) (BANXICO) y almacenados en Google Storage, desde donde se realiza la descarga de datos. Posteriormente, se realiza un proceso de limpieza y transformación para ingestar los datos en una base de datos de *Clickhouse*.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, la librería *pandas* y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

(Python) `~/data_etl/etl/banxico/$ python mun_income_remittances.py`

(Bamboo CLI) `~/data_etl/etl/banxico/$ bamboo-cli --folder . --entry mun_income_remittances`

Descarga

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar la data desde Google Storage. Cabe destacar que la data se encuentra en formato .csv; y, en cuanto a su estructura, posee 2.520 filas y 44 columnas. A continuación se presenta su estructura inicial:

	Título	Periodo disponible	...	01/04/2021	01/07/2021
Ingresos por Remesas Familiares, Aguascalientes	Ene-Mar 2003 – Jul-Sep 2021	...	177.718402	183.974273	
Ingresos por Remesas, Distribución por Municipio...	Ene-Mar 2013 – Jul-Sep 2021	...	99.922888	102.032788	
Ingresos por Remesas, Distribución por Municipio...	Ene-Mar 2013 – Jul-Sep 2021	...	3.059740	3.167073	
Ingresos por Remesas, Distribución por Municipio...	Ene-Mar 2013 – Jul-Sep 2021	...	30.244071	31.414294	
Ingresos por Remesas, Distribución por Municipio...	Ene-Mar 2013 – Jul-Sep 2021	...	0.956827	0.964759	

Figura 145. Formato inicial tabla de datos remesas según municipios

Transformación y limpieza

En esta etapa se sigue el siguiente proceso:

- Lectura de datos. Aquí se transforman los nombres de variables a minúsculas y se reemplazan espacios por guión bajo para estandarizar los nombres.
- Selección de columnas a utilizar.
- Transformación de los datos a formato *tidy*. Para ello, se utiliza la función *melt* de Pandas.
- Eliminación de filas que poseen datos agrupados a nivel de entidad.
- Reemplazo de municipios por sus *ids*. Para ello se realiza una query a Clickhouse, ya que los *ids* existen en la tabla dimensión *dim_shared_geography_mun*.
- Transformación de la variable temporal (trimestral) a formato: YYYY-MM.
- Asignación de meses a su trimestre correspondiente. Dicha transformación se efectúa con la ayuda del diccionario *trimester_map*.
- Multiplicación de los valores de remesas por un millón. Esto se realiza ya que los valores vienen en millones de dólares, y son transformados a dólares para mayor facilidad de trabajo.
- Se redondean los valores de remesas a dos decimales.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 87.080 filas y 3 columnas. A continuación puede ser visualizada su estructura:

mun_id	quarter_id	remittance_amount
1001	20131	39091951.0
1002	20131	1224811.0
1003	20131	9772749.0
1004	20131	366325.0
1010	20131	1236990.0

Figura 146. Formato final tabla de datos de remesas según municipio

17. Complejidad Económica

A continuación se presentan los pipelines que procesan datos relacionados con los índices de complejidad económica. Parte de estos indicadores son calculados con datos propios de México obtenidos desde DENUE y utilizando una librería de Datawheel que facilita el cálculo de estos indicadores (Canon-stats). Un último cubo relacionado con el Índice de Complejidad Económica a nivel de países es obtenido desde el Observatorio de Complejidad Económica (oec.world).

17.1. complexity_eci

Descripción General y Ejecución

El pipeline `eci_pipeline.py` procesa datos sobre el Índice de Complejidad Económica (ECI por sus siglas en inglés). El índice es calculado en base a los datos provenientes de DENUE utilizando como medida el punto medio del número de trabajadores a nivel de industria SCIAN. Los datos poseen desagregación a nivel de municipio, metro-áreas y entidades federativas. Los datos almacenados en Google Storage, son descargados para ser limpiados y transformados antes de ser ingestados a una base de datos de *Clickhouse*.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, las librerías `pandas` y `request`, además de dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

(Phyton) `~/data_etl/etl/complexity/$ python eci_pipeline.py`

```
(Bamboo CLI) ~/data_etl/etl/complexity/$ bamboo-cli --folder . --entry eci_pipeline
```

Descarga

Los datos son obtenidos mediante un *request* a la API de Tesseract. Dicho proceso se realiza ya que los datos son facilitados por DENUE y existen en el cubo llamado *inegi_denue*. Por lo tanto, simplemente se hace una consulta para llamar a los datos requeridos que se encuentran limpios y en formato TIDY. En cuanto a su estructura, la respuesta posee 26.243 filas y 8 columnas. A continuación se presenta su estructura inicial:

State ID	Number of Employees	Midpoint ECI	Number of Employees	Midpoint ECI	Ranking	...	Latest Metro Area ID	Municipality ID
19.0		1.629146			1	...	True	NaN
22.0		1.543553			2	...	True	NaN
2.0		1.278828			3	...	True	NaN
5.0		1.142296			4	...	True	NaN
8.0		1.132704			5	...	True	NaN

Figura 147. Formato inicial tabla de datos provenientes del cubo *inegi_denue*

Transformación y limpieza

Para conocer el rango de años disponibles, se realiza una llamada al cubo de datos *inegi_denue*:

- Desde el cubo *inegi_denue*, agregando como *drilldown* la variable *Month*, la *measure Companies* y el parámetro *parents=true* (el cual incorpora niveles superiores de los *drilldown* especificados); se realiza un *request* a la API de Tesseract: <https://dev-api.datamexico.org/tesseract/data> para obtener todos los periodos de tiempo disponibles en el cubo de datos.
- Transformación de la respuesta a un conjunto de datos JSON. Dicha acción permite, posteriormente, utilizar la función *dataframe* de Pandas para transformar los datos a un *DataFrame* llamado *df_time*.
- Se ordena el conjunto de datos *df_time* de forma descendente. La variable que rige el ordenamiento es *Month ID*.

Ahora que se tiene los periodos disponibles, se procede a realizar cálculos de complejidad económica. Es importante destacar que para construir el conjunto de datos final, se extraerá información del cubo *inegi_denue* a tres niveles geográficos: *State*, *Metro Area*, y *Municipality*; incorporando todos los periodos temporales disponibles, considerando el nivel industrial *NAICS Industry*, y agregando como medida la variable *Number of Employees Midpoint*. Dado lo anterior, se procede como sigue:

- Creación de la lista *time*. Dicha lista se construye a partir de seleccionar los valores únicos de la variable *Month ID* existente en el conjunto de datos *df_time*. Para facilitar el formato de lista, se utiliza la función *list* de Pandas.
- Definición de parámetros *threshold*: *agg=3* (retraso utilizado para medir ventajas comparativas), *threshold_geo=300* (valor mínimo de la medida en la agregación geográfica para que la observación sea considerada en los cálculos), *threshold_industry=300* (valor mínimo de la medida en la agregación industrial para que la observación sea considerada en los cálculos), *cube=inegi_denue* (conjunto de datos fuente), *level_industry=Industry Group* (nivel industrial sobre el cual se calcularán los índices de complejidad) y, *measure=Number of Employees Midpoint* (medida sobre la cual se calculan los índices de complejidad).

Iterando sobre los niveles geográficos: *State*, *Metro Area*, y *Municipality*; para todos los periodos temporales definidos en la lista *time*, se realizan los siguientes pasos:

- Creación de la variable *time_agg*, la cual se construye a partir de un subconjunto de la lista *time*. El subconjunto se compone del periodo actual (en proceso), sumado a los dos periodos anteriores en la lista. Por ejemplo, si se considera el año 2017 en proceso, se agrega también a la lista los años 2016, y 2015.
- Creación de la variable *n*, la cual se define por el largo de la lista *time_agg*.

- Creación del String `time_param`, el cual se compone al agregar, separadamente por una coma (,), los valores en la lista `time_agg`. Por ejemplo, dado `time_agg=[2017, 2016, 2015]`, se obtiene `time_param='2017,2016,2015'`. Esto se realiza ya que el parámetro será utilizado en la llamada de datos a la API de Tesseract.

Para realizar el `request` a la API de **Canon-stats** (librería de *node* que ejecuta *scripts* the Python para realizar cálculos de Complejidad Económica), se especifican los parámetros de entrada. Dichos parámetros se especifican ya que **Canon-stats** obtiene las matrices de entrada (datos) desde Tesseract, y por lo tanto, se debe definir con detalle cuales son los datos a ser procesados. Por lo anterior:

- Se definen los parámetros del `requests` (son parámetros dinámicos): `cube=inegi_denu, Month=time_param` (string definido anteriormente, varía de acuerdo al periodo en estudio), `ranking=true` (una vez realizados los cálculos, ordena de mayor a menor el índice de complejidad calculado, y crea una nueva variable ranking con valores de 1,..,m; con m el largo del conjunto de elementos), `rca=f'{level_geo},{level_industry},{measure}'` (define las dimensiones sobre las cuáles regir el cálculo del RCA, y la medida sobre la cual hacer el cálculo), y `threshold=f'{level_industry}:{threshold_industry * n},{level_geo}:{threshold_geo * n}'` (define el límite inferior de la medida al agrupar por las variables `level_geo` y `level_industry`, respectivamente).

Consideraciones:

1. Dado que se considera en el cálculo de complejidad el nivel geográfico e industrial, se está calculando el índice de complejidad económica de la industria j en el nivel geográfico i, de acuerdo al resultado en la variable Number of Employees Midpoint.
- El motivo de agregar límites inferiores como: `threshold_industry`, y `threshold_geo` es minimizar la posibilidad de obtener resultados erróneos debido a que se consideran muestras poco representativas. En ello, se establece que tanto agregaciones por nivel geográfico e industrias tengan un

número de empleados mayor o igual a 300 para ser incorporados en el cálculo de RCA. Recordemos que el *Economic Complexity Index (ECI)* se construye a partir del indicador RCA.

Una vez definidos los parámetros:

- Se realiza un *request* a la API de **Canon-stats**: <https://dev.datamexico.org/api/stats/eci>. Este *endpoint* recibe un JSON (parámetros definidos anteriormente), y realiza los cálculos de complejidad sobre las dimensiones y medidas especificadas. Entrega un *payload* de respuesta.
- Transformación del *payload* a formato JSON, el cual es transformado a un *DataFrame* de Pandas.
- Creación de la variable *Time ID*, la cual se construye a partir del periodo en estudio especificado por la variable *time_id*.
- Creación de la variable *Level*, la cual se construye a partir del nivel industrial definido en la variable *level_geo*.
- Creación de la variable *latest*, la cual se construye a partir del *index i*. En este caso, *i* es iterador temporal que toma el valor 0 en relación al valor temporal más reciente disponible. En ello, si *i*=0, entonces la variable *latest* toma el valor True, y 0 en todos los otros casos.
- Se concatenan todas las consultas al *DataFrame* global: df. Para ello se utiliza la función *concat* de Pandas.
- Renombre de columnas a conveniencia. Para ello se utiliza la función *rename* de Pandas.
- Transformación de valores *boolean* en variable *latest* a *integers*. Dado ello, valores *True* serán reemplazados con el valor 1, y valores *False* con 0.

- Se rellena con 0 los valores vacíos en las columnas: `ent_id`, `zm_id`, `mun_id` y se transforma su tipo de dato a `integer`.
- Creación de las variables `year` y `nation_id` con el valor 0. Esto se realiza con motivo de poder unir el conjunto de datos que retorna de la función `find_missing_values`. Dicha función realiza una llamada al cubo de datos `complexity_eci_a_hs12_hs6` para solicitar el Índice de Complejidad Económica anual a nivel nacional y ordenados bajo un ranking. Una vez obtenidos los datos estos son agregados al conjunto de datos final haciendo uso de la función `append` de Pandas.

Posterior al proceso de transformación, se obtiene un `DataFrame` con 26.243 filas y 10 columnas. A continuación puede ser visualizada su estructura:

<code>ent_id</code>	<code>eci</code>	<code>eci_ranking</code>	<code>time_id</code>	<code>level</code>	<code>latest</code>	<code>zm_id</code>	<code>mun_id</code>	<code>year</code>	<code>nation_id</code>
19	1.629146	1	20210510	State	1	0	0	0	0
22	1.543553	2	20210510	State	1	0	0	0	0
2	1.278828	3	20210510	State	1	0	0	0	0
5	1.142296	4	20210510	State	1	0	0	0	0
8	1.132704	5	20210510	State	1	0	0	0	0

Figura 148. Formato final tabla de datos con el Índice de Complejidad Económica (ECI)

17.2. complexity_pci

Descripción General y Ejecución

El pipeline `pci_pipeline.py` procesa datos sobre el Índice de Complejidad Productos (PCI por sus siglas en inglés). El índice es calculado en base a los datos provenientes de DENUA utilizando como medida el punto medio del número de trabajadores a nivel de zona metropolitana. Los datos poseen desagregación a nivel de industria Nacional, industria SCIAN y grupo industrial. Luego de almacenarlos en Google Storage, son descargados para ser limpiados y transformados antes de ser ingestados a una base de datos de *Clickhouse*.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, las librerías `pandas` y `request` y dos librerías desarrolladas por Datawheel: `bamboo-lib` y `dw-bamboo-cli`.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

(Python) `~/data_etl/etl/complexity/$ python pci_pipeline.py`

(Bamboo CLI) `~/data_etl/etl/complexity/$ bamboo-cli --folder . --entry pci_pipeline`

Descarga

Los datos son obtenidos mediante un `request` a la API de Tesseract debido a que los datos facilitados por DENUA ya existen en el cubo llamado `inegi_denua`. Por lo tanto, simplemente se hace una consulta para conseguir los datos necesarios. En cuanto a su estructura, la respuesta posee 20.995 filas y 11 columnas. A continuación se presenta su estructura inicial:

Industry Group ID	Number of Employees	Midpoint PCI	Number of Employees	Midpoint PCI	Ranking	Industry Group
3343		2.706597			1	Fabricación de Equipo de Audio y de Video
3352		2.400403			2	Fabricación de Aparatos Eléctricos de Uso Doméstico
4353		2.335651			3	Comercio al por Mayor de Maquinaria y Equipo p...
3329		2.289565			4	Fabricación de otros Productos Metálicos
3321		2.253934			5	Fabricación de Productos Metálicos Forjados y ...

Figura 149. Formato inicial tabla de datos por industrias provenientes del cubo *inegi_denu*

Transformación y limpieza

Para conocer el rango de años disponibles, se realiza una llamada al cubo de datos *inegi_denu*:

- Desde el cubo *inegi_denu*, agregando como *drilldown* la variable *Month*, la *measure Companies* y el parámetro *parents=true* (el cual incorpora niveles superiores de los *drilldown* especificados); se realiza un *request* a la API de Tesseract: <https://dev-api.datamexico.org/tesseract/data>.
- Transformación de la respuesta a un conjunto de datos JSON. Dicha acción permite, posteriormente, utilizar la función *dataframe* de Pandas para transformar los datos a un *DataFrame* llamado *df_time*.
- Se ordena el conjunto de datos *df_time* de forma descendente. La variable que rige el ordenamiento es *Month ID*.

Ahora que se tiene los periodos disponibles, se procede a realizar los cálculos de complejidad económica. Es importante destacar que para construir el conjunto de datos final, se extraerá información del cubo *inegi_denu* a tres niveles industriales de desagregación: *Industry Group*, *NAICS Industry*, y *National Industry*, incorporando todos los periodos temporales disponibles, considerando la desagregación geográfica metropolitana, y agregando como medida la variable *Number of Employees Midpoint*. Dado lo anterior, se procede como sigue:

- Creación de la lista *time*. Dicha lista se construye a partir de seleccionar los valores únicos de la variable *Month ID* existente en el conjunto de datos *df_time*. Para facilitar el formato de lista, se utiliza la función *list* de Pandas.
- Definición de parámetros *threshold*: *agg=3* (retraso utilizado para medir ventajas comparativas), *threshold_geo=300* (valor mínimo de la medida en la

agregación geográfica para que la observación sea considerada en los cálculos), `threshold_industry=300` (valor mínimo de la medida en la agregación industrial para que la observación sea considerada en los cálculos), `cube=inegi_denue` (conjunto de datos fuente), `level_geo=Metro Area` (nivel geográfico sobre el cual se calculan los índices de complejidad), y `measure=Number of Employees Midpoint` (medida sobre la cual se calculan los índices de complejidad).

Iterando sobre los niveles industriales: *Industry Group*, *NAICS Industry*, y *National Industry*; para todos los periodos temporales definidos en la lista `time`, se realiza lo siguiente:

- Creación de la variable `time_agg`, la cual se construye a partir de un subconjunto de la lista `time`. El subconjunto se compone del periodo actual (en proceso), sumado a los períodos entre el año actual y dos años siguientes en la lista. Por ejemplo, si consideramos el año 2017 en proceso, se agrega a la lista los años 2016, y 2015.
- Creación de la variable `n`, la cual se define por el largo de la lista `time_agg`.
- Creación del string `time_param`, el cual se compone al agregar, separadamente por una coma (,), los valores en la lista `time_agg`. Por ejemplo, dado `time_agg=[2017, 2016, 2015]`; se obtiene `time_param='2017,2016,2015'`. Esto se realiza ya que el parámetro será utilizado en la llamada de datos a la API de Tesseract.

Para realizar el *request* a la API de **Canon-stats** (librería de node que ejecuta scripts the Python para realizar cálculos de Complejidad Económica) se especifican los parámetros de entrada. Dichos parámetros se especifican ya que **Canon-stats** obtiene las matrices de entrada (datos) desde Tesseract, y por lo tanto, se deben detallar cuales son los datos a ser procesados. Por lo anterior:

- Se definen los parámetros del *requests* (son parámetros dinámicos): `cube=inegi_denue`, `Month=time_param` (*string* definido anteriormente, varía

de acuerdo al periodo en estudio), *ranking=true* (una vez realizados los cálculos, ordena de mayor índice a menor, y crea una nueva variable ranking con valores de 1,..,m; con m el largo del conjunto de elementos), *rca=f'{level_geo},{level_industry},{measure}'* (define las dimensiones sobre las cuáles regir el cálculo RCA, y la medida sobre la cual hacer el cálculo), y *threshold=f'{level_industry}:{threshold_industry * n},{level_geo}:{threshold_geo * n}'* (define el límite inferior de la medida al agrupar por las variables *level_geo* y *level_industry*, respectivamente).

Consideraciones:

1. Dado que se considera en el cálculo de complejidad el área metropolitana y el nivel industrial, se está calculando el índice de complejidad de producto de la industria j en el área metropolitana i, de acuerdo al resultado en la variable *Number of Employees Midpoint*; y esto aplicado a los diferentes niveles industriales.
2. El motivo de agregar límites inferiores como: *threshold_industry*, y *threshold_geo*; es minimizar la posibilidad de obtener resultados erróneos debido a que se consideran muestras poco representativas. En ello, se establece que tanto metro áreas como industrias tengan un número de empleados mayor o igual a 300 para ser incorporados en el cálculo de RCA. Recordemos que el *Product Complexity Index (PCI)* se construye a partir del indicador RCA.

Una vez definidos los parámetros:

- Se realiza un *request* a la API de **Canon-stats**: <https://dev.datamexico.org/api/stats/pci>. Este endpoint recibe un JSON (parámetros definidos anteriormente), y realiza los cálculos de complejidad sobre las dimensiones y medidas especificadas. Entrega un *payload* de respuesta.

- Transformación del *payload* a formato JSON, para, haciendo uso de la función *dataframe* de Pandas; crear un *DataFrame*.
- Creación de la variable *Time ID*, la cual se construye a partir del periodo en estudio especificado por la variable *time_id*.
- Creación de la variable Level, la cual se construye a partir del nivel industrial definido en la variable *level_geo*.
- Creación de la variable *latest*, la cual se construye a partir del index i. En este caso, i es iterador temporal que toma el valor 0 en relación al valor temporal más reciente disponible. En ello, si i=0, entonces la variable *latest* toma el valor True, y 0 en todos los otros casos.
- Se concatenan todas las consultas al dataframe global: df. Para ello se utiliza la función *concat* de Pandas.
- Renombre de columnas a conveniencia. Para ello se utiliza la función *rename* de Pandas.
- Selección de columnas a utilizar.
- Transformación de valores *boolean* en variable *latest* a *integers*. Dado ello, valores True serán reemplazados con el valor 1, y valores False con 0.
- Se rellena con 0 los valores nulos en las columnas: *national_industry_id*, *industry_group_id*, *naics_industry_id*; y se transforma su tipo de dato a *integer*.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 20.995 filas y 8 columnas. A continuación puede ser visualizada su estructura:

<i>national_industry_id</i>	<i>industry_group_id</i>	<i>naics_industry_id</i>	<i>time_id</i>	<i>latest</i>	<i>pci</i>	<i>pci_ranking</i>	<i>level</i>
0	3343	0	20210510	1	2.706597	1	Metro Area
0	3352	0	20210510	1	2.400403	2	Metro Area
0	4353	0	20210510	1	2.335651	3	Metro Area
0	3329	0	20210510	1	2.289565	4	Metro Area
0	3321	0	20210510	1	2.253934	5	Metro Area

Figura 150. Formato final tabla de datos con el Índice de Complejidad de Productos (PCI)

17.3. complexity_eci_a_hs12_hs6

Descripción General y Ejecución

El pipeline `eci_ranking_pipeline.py` procesa datos del Índice de Complejidad Económica (ECI por sus siglas en inglés) a nivel de países. Los datos son obtenidos del Observatorio de Complejidad Económica ([oec.world](#)) y transformados para ser ingestados a una base de datos de *Clickhouse*.

Para ejecutar este proceso de ETL, se utiliza el lenguaje de programación Python, la librería pandas y json y dos librerías desarrolladas por Datawheel: bamboo-lib y dw-bamboo-cli.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías requeridas:

```
~/data_etl/$ source venv/bin/activate
```

Luego se puede ejecutar el pipeline utilizando una de estas dos formas:

(Python) `~/data_etl/etl/eci_ranking/$ python eci_ranking_pipeline.py.py`

(Bamboo CLI) `~/data_etl/etl/eci_ranking/$ bamboo-cli --folder . --entry eci_ranking_pipeline.py`

Descarga

Los datos son obtenidos mediante un *request* a la API de Tesseract. El proceso se realiza ya que los datos son extraídos desde el cubo de flujos comerciales *trade_i_baci_a_12*, disponible en el Observatorio de Complejidad Económica. Es importante mencionar que los datos provenientes de esta fuente ya tienen los indicadores de complejidad, por lo tanto, no es necesario hacer este tipo de cálculos complejos en los pasos del pipeline que se describen a continuación.

En cuanto a su estructura, la respuesta para un año en particular posee 226 filas y 4 columnas. A continuación se presenta su estructura inicial:

Country ID	Trade Value ECI	Trade Value ECI Ranking	Country
asjpn	2.197547	1	Japan
euche	2.055590	2	Switzerland
eudeu	2.034291	3	Germany
naant	1.961102	4	Netherlands Antilles
askor	1.927689	5	South Korea

Figura 151. Formato inicial tabla de datos ECI por países

Transformación y limpieza

En esta etapa se sigue el siguiente proceso:

- Transformación de datos desde formato JSON a un *DataFrame* de Pandas. Se realiza utilizando la función *dataframe*.
- Renombre de columnas a conveniencia. Esto es posible con la función *rename* de Pandas.
- Creación de la variable *year*. Dicha variable almacena periodo anual en estudio.
- Selección de columnas a utilizar: *year*, *country_id*, *eci_rank*, y *eci*

Es importante mencionar que este proceso es iterativo para cada año desde el 2012 hasta el último año disponible.

Posterior al proceso de transformación, se obtiene un *DataFrame* con 226 filas y 4 columnas para el caso particular del año 2018. A continuación puede ser visualizada su estructura:

year	country_id	eci_rank	eci
2018	jpn	1	2.197547
2018	che	2	2.055590
2018	deu	3	2.034291
2018	ant	4	1.961102
2018	kor	5	1.927689

Figura 152. Formato final tabla de datos ECI por países

18. COVID-19

Se presentan los pipelines que procesan datos relacionados con la pandemia de COVID-19. Uno de los pipelines procesa los datos generales, mientras que los otros procesan los mismos datos para obtener diferentes indicadores a niveles geográficos diferentes.

18.1. Gobmx_covid

Descripción general y ejecución

El pipeline de datos de la situación mexicana frente al COVID-19, *covid_pipeline.py*, es una herramienta de ETL (Extracción, Transformación y Carga de datos). Estos datos ofrecen información respecto al último balance de los casos de contagios reportados en México, considerando datos geográficos y características de los contagiados, fallecidos y casos sospechosos (edad, sexo y comorbilidades).

Los datos son descargados desde [el repositorio de datos del gobierno de México](#) y pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos Clickhouse siguiendo un modelo relacional.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de la siguiente forma:

(bamboo-cli)

```
~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_pipeline
```

Si el archivo de datos ya se encuentra descargado, entonces se puede pasar como parámetro del pipeline el directorio donde está localizado el archivo.

(bamboo-cli)

```
~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_pipeline
--file_path='/path/to/file'
```

Descarga de datos

Para efectuar la descarga de datos, el pipeline se conecta directamente a la página fuente, usando la clase *DownloadStep* de *Bamboo*. El archivo viene comprimido en formato ZIP, por lo que es descomprimido y puesto a disposición del siguiente paso del pipeline.

Lectura de datos

Una vez descargados los datos, estos se leen y almacenan en un *DataFrame* de la librería pandas.

Al momento de la elaboración de esta documentación, el *DataFrame* inicial contenía 7.228.942 filas por 40 columnas, sin embargo este valor crece día a día ya que el pipeline es ejecutado a diario para mantener actualizados los datos. La siguiente imagen muestra las primeras filas y algunas columnas del *DataFrame* inicial.

	FECHA_ACTUALIZACION	ID_REGISTRO	ORIGEN	SECTOR	ENTIDAD_UM	SEXO	ENTIDAD_NAC	ENTIDAD_RES	MUNICIPIO_RES	TIPO_PACIENTE	...
0	2021-06-13	z482b8	1	12	9	2	9	9	12	1	...
1	2021-06-13	z49a69	1	12	23	1	23	23	4	2	...
2	2021-06-13	z23d9d	1	12	22	2	24	22	9	1	...
3	2021-06-13	z24953	1	12	9	1	9	9	10	1	...
4	2021-06-13	zz8e77	1	12	9	2	9	9	2	1	...
...

Figura 153. Formato original reporte diario COVID-19

Transformación y limpieza

Posterior a la lectura de los datos, estos son sometidos al proceso de transformación y limpieza para poder obtenerlos en el formato deseado. Primero, se estandarizan los nombres de las columnas del *DataFrame*, quitando símbolos no deseados de estos,

renombrando las columnas con nombres en inglés fáciles de identificar y dejando en letras minúsculas todas las palabras. Continuando con la estandarización, se estandarizan los formatos de las columnas relacionadas a entidad y municipio haciendo que sus valores sean del tipo *string* (caracteres).

Las columnas que informan si el paciente falleció, fecha de fallecimiento, país de origen y nacionalidad son estandarizadas en el siguiente paso, quitando sus valores nulos y reemplazándolos por valores numéricos donde corresponde. Por ejemplo, en la columna *is_dead*, donde había un valor NaN, ahora habrá un 0.

Finalmente, se reemplazan los valores no conocidos de los números identificadores de municipalidades por un valor estándar y se estandarizan los ID de *covid_positive* donde 1 indica que el paciente dio positivo al covid, 2 si es negativo y 3 si no se ha obtenido resultado (caso sospechoso).

Al momento en el que se elaboró esta documentación, el *DataFrame* final contenía 7.228.942 filas y 39 columnas. La estructura final de la tabla se presenta a continuación:

	updated_date	origin	type_health_institution_attended	health_institution_attended_ent	sex	patient_origin_ent_id	patient_type	ingress_date	symptoms_date	death_date	...	country_nationality	country_origin	required_ICU	clasificacion_final	patient_residence_mun_id	is_dead	country_nationality_old	country_origin_id	time_id	id	
0	20200813	1		12	9	1	20200106	20200106	NaN	...	xxa	xxa	97	1	9012	0	México	97	20210613	1		
1	20200813	1		12	23	2	20200720	20200717	20200721,0	...	xxa	xxa	1	2	23004	1	México	97	20210613	1		
2	20200813	1		12	22	1	20210105	20210105	NaN	...	xxa	xxa	97	6	22009	0	México	97	20210613	1		
3	20200813	1		12	9	2	9	1	20200915	20201015	NaN	...	xxa	xxa	97	7	9010	0	México	97	20210613	1
4	20200813	1		12	9	1	20200416	20200410	NaN	...	xxa	xxa	97	6	9002	0	México	97	20210613	1		
...		
7228937	20210613	2		4	25	2	25	1	20210611	20210610	NaN	...	xxa	xxa	97	7	25008	0	México	97	20210613	1
7228938	20210613	2		12	24	1	24	1	20210611	20210611	NaN	...	xxa	xxa	97	6	24028	0	México	97	20210613	1
7228939	20210613	2		12	27	1	27	1	20210611	20210610	NaN	...	xxa	xxa	97	6	27002	0	México	97	20210613	1
7228940	20210613	2		4	30	2	30	1	20210611	20210610	NaN	...	xxa	xxa	97	7	30193	0	México	97	20210613	1
7228941	20210613	2		4	31	2	31	1	20210610	20210609	NaN	...	xxa	xxa	97	3	31101	0	México	97	20210613	1

Figura 154. Formato final tabla de datos COVID-19

18.2. Gobmx_covid_stats_nation

Descripción general y ejecución

El pipeline de estadísticas de la situación mexicana frente al COVID-19, *covid_stats_nation.py*, es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos del cubo *gobmx_covid* para obtener diferentes métricas asociadas a la evolución de la pandemia a nivel nacional.

Estos datos ofrecen información respecto de la evolución de los casos de positivos, fallecidos, sospechosos y hospitalizados, informando los valores diarios, acumulados, promedios móviles a 7 días y tasas de contagios sobre el total de la población.

Los datos son obtenidos desde la [API de tesseract](#) construida por Datawheel para el sitio DataMéxico. También se usan los datos del último reporte COVID-19 entregado por el gobierno mexicano en su [repositorio](#). Pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos Clickhouse siguiendo un modelo relacional.

Cabe destacar que este pipeline es necesario para obtener las métricas correctas a diferentes niveles geográficos, en este caso, a nivel nacional. Al momento de ingestar los datos, cuando hay diferentes niveles geográficos estos se agrupan desde los niveles más pequeños a los niveles superiores en sumas o promedios simples. En el caso de las estadísticas del COVID-19 se calcula, por ejemplo, el promedio móvil a 7 días, medida que al ser agregada a diferentes niveles geográficos no representaría el valor real del indicador.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de la siguiente forma:

(bamboo-cli)

```
~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_stats_nation
```

Si el archivo de datos ya se encuentra descargado, entonces se puede pasar como parámetro del pipeline el directorio donde está localizado el archivo.

(bamboo-cli)

```
~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_stats_nation
--file_path='/path/to/file'
```

Lectura de datos

Los datos son extraídos desde la API usando la librería *requests* de python, la que permite extraer directamente los datos desde la API y recibirlas en formato JSON, estos se leen y almacenan en un *DataFrame* de la librería pandas. Son 4 las llamadas a la API que se realizan para obtener el *DataFrame* inicial. Una vez recibida cada llamada, se guarda en un *DataFrame* y se junta con los datos anteriores. Cabe destacar que las llamadas obtienen datos que toman relación con: casos, muertes, hospitalizados, y sospechosos.

Al momento de la elaboración de esta documentación, el *DataFrame* inicial contenía 1 fila y 9 columnas. La siguiente imagen muestra un extracto del *DataFrame* inicial. Cabe destacar que posee sólo una fila ya que son los agregados a nivel nacional para la última fecha disponible.

date	accum_cases_report	new_cases_report	accum_deaths_report	new_deaths_report	accum_hospitalized_report	new_hospitalized_report	accum_suspect_report	new_suspect_report
20210613	2454176	0	230150	0	459470	0	434331	0

Figura 155. Formato original datos COVID-19 a nivel nacional, última fecha disponible

En cuanto al último reporte de COVID-19 este es descargado por el pipeline *covid_pipeline* y, al momento de la elaboración de esta documentación, el

DataFrame inicial contenía 7.228.942 filas por 40 columnas. Para mayor información ver sección 18.1.

Transformación y limpieza

Primero, se estandarizan los ID de *covid_positive* donde 1 indica si el paciente dio positivo al COVID-19, 2 si es negativo, y 3 si no se ha obtenido resultado. Luego se agregan los datos, agrupando por casos positivos, negativos y no reportados, obteniendo la suma de casos sospechosos, casos hospitalizados y casos totales, esto en *DataFrames* separados. Posteriormente se vuelve a juntar toda la información en un solo *DataFrame*.

Luego se procede a reemplazar los datos vacíos o nulos en las columnas de tiempo y municipio, para agregar consistencia y validez a los datos.

Finalmente, se juntan los datos del último reporte del gobierno mexicano con los datos ya pre procesados obtenidos desde la API, para luego obtener agregaciones de este *DataFrame*, tales como los casos promedio por día, acumulación total de casos, porcentaje de nuevos casos, entre otros.

Al momento en el que se elaboró esta documentación, el *DataFrame* final contiene 473 filas y 45 columnas. La estructura final de la tabla se presenta a continuación:

	date	daily_cases	daily_hospitalized	daily_suspect	resultado	daily_deaths	days_between_ingress_and_death	accum_cases_report	new_cases_report	accum_deaths_report	...	rate_daily_cases	rate_accum_cases	rate_new_cases_report	rate_accum_deaths_report	rate_daily_deaths	rate_accum_deaths	rate_new_deaths_report	rate_accum_deaths_report	day_from_50_cases	day_from_10_deaths
0	20200227	3	1	0	Nan	0	Nan	0	0	0	...	0.002348	0.000348	0.0	0.000000	0.000000	0.0	0.000000	0	0	
1	20200228	2	0	0	Nan	0	Nan	0	0	0	...	0.001565	0.000313	0.0	0.000000	0.000000	0.0	0.000000	0	0	
2	20200229	1	0	0	Nan	0	Nan	0	0	0	...	0.000783	0.0004695	0.0	0.000000	0.000000	0.0	0.000000	0	0	
3	20200301	0	0	0	Nan	0	Nan	0	0	0	...	0.000000	0.0004695	0.0	0.000000	0.000000	0.0	0.000000	0	0	
4	20200302	1	1	0	Nan	0	Nan	0	0	0	...	0.000783	0.0004748	0.0	0.000000	0.000000	0.0	0.000000	0	0	
...	
468	20200609	806	129	2817	639.0	39	8.769731	0	0	0	...	0.630211	1325.094067	0.0	0.000000	0.030619	1427.006861	0.0	0.000000	456	441
469	20201010	540	130	3867	1289.0	38	10.394737	0	0	0	...	0.423561	1325.519428	0.0	0.000000	0.027938	1427.365857	0.0	0.000000	457	442
470	20201011	380	92	6105	2035.0	31	10.193548	0	0	0	...	0.297358	1325.815985	0.0	0.000000	0.024258	1427.209845	0.0	0.000000	458	443
471	20201012	15	6	2862	954.0	10	6.500000	0	0	0	...	0.011738	1325.826723	0.0	0.000000	0.007825	1427.88670	0.0	0.000000	459	444
472	20201013	0	0	39	13.0	0	Nan	2454176	0	230150	...	0.000000	1325.826723	0.0	1920.441426	0.000000	1427.88670	0.0	180.096943	460	445

Figura 156. Formato final tabla de datos y estadísticas COVID-19 a nivel nacional

18.3. Gobmx_covid_stats_state

Descripción general y ejecución

El pipeline de estadísticas de la situación mexicana frente al COVID-19, *covid_stats_state.py*, es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos del cubo *gobmx_covid* para obtener diferentes métricas asociadas a la evolución de la pandemia a nivel estatal.

Estos datos ofrecen información respecto de la evolución de los casos de positivos, fallecidos, sospechosos y hospitalizados, informando los valores diarios, acumulados, promedios móviles a 7 días y tasas de contagios sobre el total de la población.

Los datos son obtenidos desde la [API de tesseract](#) construida por Datawheel para el sitio DataMéxico. También se usan los datos del último reporte COVID-19 entregado por el gobierno mexicano en su [repositorio](#). Pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos Clickhouse siguiendo un modelo relacional.

Cabe destacar que este pipeline es necesario para obtener las métricas correctas a diferentes niveles geográficos, en este caso, para cada entidad federativa. Al momento de ingestar los datos, cuando hay diferentes niveles geográficos estos se agrupan desde los niveles más pequeños a los niveles superiores en sumas o promedios simples. En el caso de las estadísticas del COVID-19 se calcula, por ejemplo, el promedio móvil a 7 días, medida que al ser agregada a diferentes niveles geográficos no representaría el valor real de indicador.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline a través de *bamboo-cli* de la siguiente forma:

```
~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_stats_state
```

Si el archivo de datos ya se encuentra descargado, entonces se puede pasar como parámetro del pipeline el directorio donde está localizado el archivo.

```
~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_stats_state  
--file_path='/path/to/file'
```

Lectura de datos

Los datos son extraídos desde la API usando la librería *requests* de python, la que permite extraer directamente los datos desde la API y recibirlas en formato JSON, estos se leen y almacenan en un *DataFrame* de la librería pandas. Son 4 las llamadas a la API que se realizan para obtener el *DataFrame* inicial. Una vez recibida cada llamada, se guarda en un *DataFrame* y se junta con los datos anteriores.

Al momento de la elaboración de esta documentación, el *DataFrame* inicial contiene 32 filas por 10 columnas. La siguiente imagen muestra las primeras filas y algunas columnas del *DataFrame* inicial.

	date	ent_id	accum_cases_report	new_cases_report	accum_deaths_report	new_deaths_report	accum_hospitalized_report	new_hospitalized_report	accum_suspect_report	new_suspect_report
0	20210613	1	26649	0	2446	0	5403	0	10310	0
1	20210613	2	49997	0	8617	0	15239	0	33664	0
2	20210613	3	34180	0	1461	0	2960	0	3036	0
3	20210613	4	10674	0	1243	0	2549	0	4639	0
4	20210613	5	69220	0	6365	0	11967	0	14004	0
5	20210613	6	12115	0	1204	0	2636	0	2120	0
6	20210613	7	11827	0	1647	0	3542	0	38841	0
7	20210613	8	57224	0	7450	0	13919	0	5121	0

Figura 157. Formato original datos COVID-19 a nivel estatal, última fecha disponible

En cuanto al último reporte de COVID-19 este es descargado por el pipeline *covid_pipeline* y, al momento de la elaboración de esta documentación, el *DataFrame* inicial contenía 7.228.942 filas por 40 columnas. Para mayor información ver sección 18.1.

Transformación y limpieza

Primero, se estandarizan los ID de *covid_positive* donde 1 indica si el paciente dio positivo al COVID-19, 2 si es negativo, y 3 si no se ha obtenido resultado. Luego se agregan los datos, agrupando por casos positivos, negativos y no reportados, obteniendo la suma de casos sospechosos, casos hospitalizados y casos totales, esto en *DataFrames* separados. Posteriormente se vuelve a juntar toda la información en un solo *DataFrame*.

Luego se procede a reemplazar los datos vacíos o nulos en las columnas de tiempo y entidad federativa, para agregar consistencia y validez a los datos.

Finalmente, se juntan los datos del último reporte del gobierno mexicano con los datos ya pre procesados obtenidos desde la API, para luego obtener agregaciones de este *DataFrame*, tales como los casos promedio por día, acumulación total de casos, porcentaje de nuevos casos, entre otros.

Al momento en el que se elaboró esta documentación, el *DataFrame* final contenía 14.713 filas y 46 columnas. La estructura final de la tabla se presenta a continuación:

date	ent_id	daily_cases	daily_hospitalized	daily_suspect	resultado	daily_deaths	days_between_ingress_and_death	acum_cases_report	new_cases_report	...	rate_daily_cases	rate_acum_cases	rate_new_cases_report	rate_acum_cases_report	rate_daily_deaths	rate_acum_deaths	rate_new_deaths_report	rate_acum_deaths_report	day_from_50_cases	day_from_10_deaths
20200314	1	0	0	Nan	0	Nan	0	0	0	0.00000004	0.00000004	0.0	0.00000000	0.0	0.0	0.00000000	0.0	0	0	
20200315	1	0	0	0	Nan	0	Nan	0	0	0.00000000	0.00000000	0.0	0.00000000	0.0	0.0	0.00000000	0.0	0	0	
20200316	1	0	0	0	Nan	0	Nan	0	0	0.00000000	0.00000000	0.0	0.00000000	0.0	0.0	0.00000000	0.0	0	0	
20200317	1	1	0	Nan	0	Nan	0	0	0	0.00000004	0.00000004	0.0	0.00000000	0.0	0.0	0.00000000	0.0	0	0	
20200318	1	3	0	0	Nan	0	Nan	0	0	0.209112	0.348851	0.0	0.00000000	0.0	0.00000000	0.0	0.00000000	0	0	
...	
20210609	32	7	2	0	Nan	0	Nan	0	0	0.420081	1690.56117	0.0	0.000000	0.0	0.000000	0.0	0.000000	416	408	
20210610	32	6	2	0	3.0	0	Nan	0	0	0.380052	1690.924169	0.0	0.000000	0.0	0.000000	0.0	0.000000	417	408	
20210611	32	4	0	21	7.0	0	Nan	0	0	0.240035	1691.16204	0.0	0.000000	0.0	0.000000	0.0	0.000000	418	407	
20210612	32	0	0	0	3.0	0	Nan	0	0	0.000000	1691.164204	0.0	0.000000	0.0	0.000000	0.0	0.000000	419	408	

Figura 158. Formato final tabla de datos y estadísticas COVID-19 a nivel estatal

18.4. Gobmx_covid_stats.metroarea

Descripción general y ejecución

El pipeline de estadísticas de la situación mexicana frente al COVID-19, *covid_stats_metroarea.py*, es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos del cubo *gobmx_covid* para obtener diferentes métricas asociadas a la evolución de la pandemia a nivel de área metropolitana..

Estos datos ofrecen información respecto de la evolución de los casos de positivos, fallecidos, sospechosos y hospitalizados, informando los valores diarios, acumulados, promedios móviles a 7 días y tasas de contagios sobre el total de la población.

Los datos son obtenidos del último reporte COVID-19 entregado por el gobierno mexicano en su [repositorio](#). Pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos Clickhouse siguiendo un modelo relacional.

Cabe destacar que este pipeline es necesario para obtener las métricas correctas a diferentes niveles geográficos, en este caso, para cada área metropolitana. Al momento de ingestar los datos, cuando hay diferentes niveles geográficos estos se agrupan desde los niveles más pequeños a los niveles superiores en sumas o promedios simples. En el caso de las estadísticas del COVID-19 se calcula, por ejemplo, el promedio móvil a 7 días, medida que al ser agregada a diferentes niveles geográficos no representaría el valor real del indicador.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de la siguiente forma:

(bamboo-cli)

```
~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_stats.metroarea
```

Si el archivo de datos ya se encuentra descargado, entonces se puede pasar como parámetro del pipeline el directorio donde está localizado el archivo.

```
(bamboo-cli) ~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_stats.metroarea --file_path='/path/to/file'
```

Lectura de datos

En el presente ETL son utilizados tres conjuntos de datos extraídos desde diferentes fuentes. Primero que todo, se importan datos de población a nivel de área metropolitana, desde el cubo de datos *population_projection*, y son almacenados un *DataFrame*. Segundo, se lee un conjunto de datos que posee los id para las áreas metropolitanas. Tercero, se agregan los datos del último reporte de COVID-19, los cuales son descargados a través del pipeline *covid_pipeline*, y son puestos a disposición para ejecutar el procesamiento de los datos.

A continuación se visualiza la estructura inicial de los tres conjuntos de datos mencionados anteriormente. Los datos de población a nivel de área metropolitana poseen una estructura inicial de 74 filas y 3 columnas.

	Metro Area ID	Metro Area	Projected Population
0	99101	Aguascalientes	1143729
1	99201	Ensenada	536143
2	99202	Mexicali	1087478
3	99203	Tijuana	2011247
4	99301	La Paz	301961
...
69	993006	Poza Rica	570057
70	993007	Veracruz	956463
71	993008	Xalapa	817501
72	993101	Mérida	1237697
73	993201	Zacatecas-Guadalupe	399055

Figura 159. Formato original datos proyección de población por zona metropolitana

Los datos de id para áreas metropolitanas, poseen una estructura inicial de 417 filas y 3 columnas.

	zm_id	NOM_ZM	mun_id
0	99101	Aguascalientes	1001
1	99101	Aguascalientes	1005
2	99101	Aguascalientes	1011
3	99201	Ensenada	2001
4	99202	Mexicali	2002
...
412	993201	Zacatecas-Guadalupe	32017
413	993201	Zacatecas-Guadalupe	32032
414	993201	Zacatecas-Guadalupe	32050
415	993201	Zacatecas-Guadalupe	32056
416	993201	Zacatecas-Guadalupe	32057

Figura 160. Formato original datos municipios pertenecientes a cada zona metropolitana

En cuanto al último reporte de COVID-19 este es descargado por el pipeline `covid_pipeline` y, al momento de la elaboración de esta documentación, el `DataFrame` inicial contenía 7.228.942 filas por 40 columnas. Para mayor información ver sección 18.1.

Limpieza y Transformación

Primero, se estandarizan los ID de *covid_positive* donde 1 indica si el paciente dio positivo al COVID-19, 2 si es negativo, y 3 si no se ha obtenido resultado. Luego se agregan los datos, agrupando por casos positivos, negativos y no reportados, obteniendo la suma de casos sospechosos, casos hospitalizados y casos totales, esto en *DataFrames* separados. Posteriormente se vuelve a juntar toda la información en un solo *DataFrame*.

Segundo, se procede a reemplazar los datos vacíos o nulos en las columnas de tiempo y área metropolitana, para agregar consistencia y validez a los datos.

Finalmente, se obtienen las estadísticas tales como casos promedio por día, acumulación total de casos, porcentaje de nuevos casos, entre otros..

Al momento en el que se elaboró esta documentación, el *DataFrame* final contiene 33.257 filas y 20 columnas. La estructura final de la tabla se presenta a continuación:

date	zm_id	daily_cases	daily_deaths	accum_cases	accum_deaths	avg7_daily_cases	avg7_accum_cases	avg7_daily_deaths	avg7_accum_deaths	sum_last7_daily_cases	sum_last7_accum_cases	sum_last7_daily_deaths	sum_last7_accum_deaths	rate_daily_cases	rate_accum_cases	rate_daily_deaths	rate_accum_deaths	day_from_50_cases	day_from_10_deaths
20200314	99101	1	0	1	0	NaN	NaN	NaN	NaN	0	0	0	0	0.087433	0.087433	0.000000	0.000000	0	0
20200315	99101	0	0	1	0	NaN	NaN	NaN	NaN	0	0	0	0	0.000000	0.087433	0.000000	0.000000	0	0
20200316	99101	0	0	1	0	NaN	NaN	NaN	NaN	0	0	0	0	0.000000	0.087433	0.000000	0.000000	0	0
20200317	99101	1	0	2	0	NaN	NaN	NaN	NaN	0	0	0	0	0.087433	0.174867	0.000000	0.000000	0	0
20200318	99101	2	0	4	0	NaN	NaN	NaN	NaN	0	0	0	0	0.174867	0.349733	0.000000	0.000000	0	0
...
20210608	993201	5	1	13731	965	4.265714	13721.142857	0.426571	953.714286	30	96048	3	6676	1.252960	3440.879077	0.250992	239.315383	404	369
20210609	993201	4	0	13735	965	3.575429	13724.742857	0.288714	954.000000	25	96073	2	6678	1.002368	3441.891445	0.000000	239.315383	405	370
20210610	993201	4	0	13739	965	3.265714	13728.000000	0.288714	954.285714	23	96096	2	6680	1.002368	3442.883813	0.000000	239.315383	406	371
20210611	993201	3	0	13742	965	3.142857	13731.142857	0.288714	954.571429	22	96118	2	6682	0.797179	3443.859589	0.000000	239.315383	407	372
20210612	993201	0	0	13742	955	2.857143	13734.000000	0.142857	954.714286	20	96138	1	6683	0.000000	3443.855689	0.000000	239.315383	408	373

Figura 161. Formato final tabla de datos y estadísticas COVID-19 a nivel de zona metropolitana

18.5. Gobmx_covid_stats_mun

Descripción general y ejecución

El pipeline de estadísticas de la situación mexicana frente al COVID-19, *covid_stats_mun.py*, es una herramienta de ETL (Extracción, Transformación y Carga de datos) que procesa los datos del cubo *gobmx_covid* para obtener diferentes métricas asociadas a la evolución de la pandemia a nivel municipal.

Estos datos ofrecen información respecto de la evolución de los casos positivos, fallecidos, sospechosos y hospitalizados, informando los valores diarios, acumulados, promedios móviles a 7 días y tasas de contagios sobre el total de la población.

Los datos son obtenidos desde la [API de tesseract](#) construida por Datawheel para el sitio DataMéxico. También se usan los datos del último reporte COVID-19 entregado por el gobierno mexicano en su [repositorio](#). Pasan por un proceso de limpieza y transformación, para luego ser ingestados en una base de datos Clickhouse siguiendo un modelo relacional.

Cabe destacar que este pipeline es necesario para obtener las métricas correctas a diferentes niveles geográficos, en este caso, para cada municipio. Al momento de ingestar los datos, cuando hay diferentes niveles geográficos estos se agrupan desde los niveles más pequeños a los niveles superiores en sumas o promedios simples. En el caso de las estadísticas del COVID-19 se calcula, por ejemplo, el promedio móvil a 7 días, medida que al ser agregada a diferentes niveles geográficos no representaría el valor real del indicador.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de la siguiente forma:

```
(bamboo-cli) ~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_stats_mun
```

Si el archivo de datos ya se encuentra descargado, entonces se puede pasar como parámetro del pipeline el directorio donde está localizado el archivo.

```
(bamboo-cli) ~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_stats_mun --file_path='/path/to/file'
```

Lectura de datos

Los datos son extraídos desde la API usando la librería *requests* de python, la que permite extraer directamente los datos desde la API y recibirlas en formato JSON, estos se leen y almacenan en un *DataFrame* de la librería pandas. Son 4 las llamadas a la API que se realizan para obtener el *DataFrame* inicial. Una vez recibida cada llamada, se guarda en un *DataFrame* y se junta con los datos anteriores. Cabe destacar que las llamadas obtienen datos que toman relación con: casos positivos, fallecidos, hospitalizados, y sospechosos.

Al momento de la elaboración de esta documentación, el *DataFrame* inicial contenía 2.400 filas por 10 columnas. La siguiente imagen muestra algunas filas y columnas del *DataFrame* inicial.

date	mun_id	accum_cases_report	new_cases_report	accum_deaths_report	new_deaths_report	accum_hospitalized_report	new_hospitalized_report	accum_suspect_report	new_suspect_report
20210613	1001	22037.0	0.0	2133.0	0.0	4577.0	0.0	9162.0	0.0
20210613	1002	464.0	0.0	36.0	0.0	94.0	0.0	38.0	0.0
20210613	1003	982.0	0.0	26.0	0.0	95.0	0.0	54.0	0.0
20210613	1004	129.0	0.0	12.0	0.0	31.0	0.0	19.0	0.0
20210613	1005	778.0	0.0	62.0	0.0	155.0	0.0	364.0	0.0
...
20210613	20436	NaN	NaN	NaN	NaN	NaN	NaN	1.0	0.0
20210613	20480	NaN	NaN	NaN	NaN	NaN	NaN	2.0	0.0
20210613	20491	NaN	NaN	NaN	NaN	NaN	NaN	1.0	0.0
20210613	20514	NaN	NaN	NaN	NaN	NaN	NaN	1.0	0.0
20210613	20541	NaN	NaN	NaN	NaN	NaN	NaN	1.0	0.0

Figura 162. Formato original datos COVID-19 a nivel municipal, última fecha disponible

En cuanto al último reporte de COVID-19 este es descargado por el pipeline *covid_pipeline* y, al momento de la elaboración de esta documentación, el

DataFrame inicial contenía 7.228.942 filas por 40 columnas. Para mayor información ver sección 4.2.

2. Transformación y limpieza

Primero, se estandarizan los ID de *covid_positive* donde 1 indica si el paciente dio positivo al COVID-19, 2 si es negativo, y 3 si no se ha obtenido resultado. Luego se agregan los datos, agrupando por casos positivos, negativos y no reportados, obteniendo la suma de casos sospechosos, casos hospitalizados y casos totales, esto en *DataFrames* separados. Posteriormente se vuelve a juntar toda la información en un solo *DataFrame*.

Luego se procede a reemplazar los datos vacíos o nulos en las columnas de tiempo y municipio, para agregar consistencia y validez a los datos.

Finalmente, se juntan los datos del último reporte del gobierno mexicano con los datos ya pre procesados obtenidos desde la API, para luego obtener agregaciones de este *DataFrame*, tales como los casos promedio por día, acumulación total de casos, porcentaje de nuevos casos, entre otros..

Al momento en el que se elaboró esta documentación, el *DataFrame* final contenía 909.735 filas y 46 columnas. La estructura final de la tabla se presenta a continuación:

date	mun_id	daily_cases	daily_hospitalized	daily_suspect	resultado	daily_deaths	days_between_ingress_and_death	accum_cases_report	new_cases_report	...	rate_daily_cases	rate_accum_cases	rate_new_cases_report	rate_accum_cases_report	rate_daily_deaths	rate_accum_deaths	rate_new_deaths_report	rate_accum_deaths_report	day_from_50_cases	day_from_10_deaths
20200314	1001	1	0	0	Nan	0	Nan	0	0	...	0.03993	0.03993	0.0	0.0	0.0	0.0	0.0	0	0	0
20200315	1001	0	0	0	Nan	0	Nan	0	0	...	0.00000	0.03993	0.0	0.0	0.0	0.0	0.0	0	0	0
20200316	1001	0	0	0	Nan	0	Nan	0	0	...	0.00000	0.03993	0.0	0.0	0.0	0.0	0.0	0	0	0
20200317	1001	1	1	0	Nan	0	Nan	0	0	...	0.03993	0.207905	0.0	0.0	0.0	0.0	0.0	0	0	0
20200318	1001	2	0	0	Nan	0	Nan	0	0	...	0.03993	0.415810	0.0	0.0	0.0	0.0	0.0	0	0	0
...	
20210409	32999	0	0	0	Nan	0	Nan	0	0	...	Nan	Nan	Nan	Nan	Nan	Nan	Nan	0	0	
20210410	32999	0	0	0	Nan	0	Nan	0	0	...	Nan	Nan	Nan	Nan	Nan	Nan	Nan	0	0	
20210411	32999	0	0	0	Nan	0	Nan	0	0	...	Nan	Nan	Nan	Nan	Nan	Nan	Nan	0	0	
20210412	32999	0	0	0	Nan	0	Nan	0	0	...	Nan	Nan	Nan	Nan	Nan	Nan	Nan	0	0	
20210413	32999	0	0	0	Nan	0	Nan	0	0	...	Nan	Nan	Nan	Nan	Nan	Nan	Nan	0	0	

Figura 163. Formato final tabla de datos y estadísticas COVID-19 a nivel municipal

19. Precios de productos

El siguiente pipeline procesa datos de precios de productos agropecuarios.

19.1. sniim

Descripción general y ejecución

El pipeline `sniim_pipeline.py` procesa datos de precios de productos agroalimentarios provenientes del Sistema Nacional de Información e Integración de Mercados (SNIIM). Los datos contienen registros diarios de precios promedios de distintos productos en diferentes estados del país. La información de cada producto es entregada en un archivo .csv y almacenada en Google Storage, desde donde son descargados para ejecutar el proceso de limpieza y transformación antes de ser ingestados en una base de datos de *Clickhouse*.

Para llevar a cabo el proceso mencionado anteriormente, se utiliza el lenguaje de programación Python, donde son fundamentales las librerías pandas y dos librerías desarrolladas por Datawheel: *bamboo-lib* y *dw-bamboo-cli*.

Ejecución

Para ejecutar el pipeline se debe activar un entorno virtual de Python 3 que cuente con las librerías necesarias:

```
~/data-etl/$ source venv/bin/activate
```

Luego, se puede ejecutar el pipeline de la siguiente forma:

(bamboo-cli)

```
~/data-etl/etl/covid/$ bamboo-cli --folder . --entry covid_pipeline
```

(Python)

```
~/data-etl/etl/sniim/$ python sniim_pipeline.py
```

Descarga de datos

Utilizando el módulo de descarga que facilita la librería *bamboo-lib* se procede a descargar los datos desde Google Storage. Todos los archivos utilizados se encuentran en formato .csv y, en relación a su estructura, cada archivo presenta diferentes formatos (distinto número de columnas y filas, diferentes nombres de columnas, diferentes nomenclaturas en los datos). El objetivo del pipeline es homogeneizar los diferentes archivos e integrarlos en una única tabla de datos.

Lectura de datos

Todos los archivos de datos y dimensiones de datos (dimensión de productos, marcas y estados creadas especialmente para este pipeline) son almacenados en la misma carpeta en GCP. Luego, el pipeline lee todos los archivos y se crean dos arreglos con nombres de archivos, el primero (*data*) guarda todos los archivos con datos (para ello identifica todos los archivos que no contienen el string *_dimension* en su nombre) y, el segundo (*dim*) guarda los archivos de dimensiones (en este caso identifica los archivos que si contienen el string *_dimension* en su nombre).

Luego de la lectura de los archivos, se crean los diccionarios de datos que permitirán reemplazar valores de texto por sus correspondientes ids (ids creados en los archivos de dimensiones). Se crean cuatro diccionarios: *dict_states*, *dict_products*, *dicto_hs2* y *dict_marks*, los cuales serán utilizados en pasos posteriores del pipeline.

Transformación y limpieza

Para procesar los archivos de datos y unificarlos en una única tabla, el pipeline contiene un ciclo *for* que itera por cada archivo renombrando columnas, agregando o eliminando columnas, agregando una columna con el nombre del archivo (la cual será útil para procesamientos posteriores). En algunos casos se crean columnas con valores *NAN*. Todo depende del archivo que se esté procesando y la información disponible en él. Al final el *loop*, se tiene una tabla con las columnas necesarias y que agrupa todos los productos disponibles.

El procesamiento que sigue permite limpiar la tabla resultante del ciclo for. Los pasos son:

- Se agrega la columna *unit_id* que toma el valor 1 para litros, 2 para kilogramo y 3 para tonelada. Esto para identificar la unidad de medida de cada producto.
- Se eliminan filas no útiles. En este caso, el archivo de huevos contenía registros que debían ser eliminados (las filas con los strings *Huevo clasificado* y *Caja 24 Docenas* fueron eliminados).
- Se crea la columna *product_temp* a partir de las columnas *product* y *subProduct*. El contenido de la columna creada hace match con la tabla de dimensiones de productos y se utiliza el diccionario *dict_products* para reemplazar los valores por sus ids.
- Nuevamente se utiliza la columna *product_temp*, ahora para crear la columna *hs2_id* con los ids de la clasificación armonizada de productos a dos dígitos.
- Utilizando el diccionario *dict_states* se reemplazan los valores de la columna *state_id* por los *id's* correspondientes a las entidades federativas.
- Se crea la columna *mark_id*. Para ello se reemplazan los valores nulos de la columna *mark* por *No aplica*, se eliminan los espacios sobrantes y se reemplazan algunos valores. Con esto es posible utilizar el diccionario de marcas *dict_marks* y obtener la columna con sus ids correspondientes.
- Se corrige el tipo de dato de la columna *value* a *float*.
- Se aplica la función *to_datetime* para homogeneizar el formato de la columna *time_id*. Posteriormente se eliminan los guiones en las fechas para obtener un formato como *20200101* (año, mes, día).
- Finalmente se eliminan las columnas que no son de interés.

Posterior al proceso de transformación, se obtiene un DataFrame con 15.485.52 filas y 7 columnas (las filas irán aumentando a medida que se agreguen nuevos datos a los archivos originales). A continuación puede ser visualizada su estructura de salida:

value	unit_id	product_id	hs2_id	state_id	time_id	mark_id
125.0	2	315	416	9	20180103	999
125.0	2	315	416	9	20180104	999
170.0	2	315	416	19	20180104	999
125.0	2	315	416	9	20180105	999
125.0	2	315	416	9	20180108	999
170.0	2	315	416	19	20180108	999
125.0	2	315	416	9	20180109	999
125.0	2	315	416	9	20180110	999
...
200.0	2	352	103	4	20220428	999
275.0	2	352	103	30	20220428	999
200.0	2	352	103	4	20220429	999
210.0	2	352	103	30	20220429	999

Figura 164. Formato final tabla de datos precios de productos SNIIM.

ANEXO

Proceso de Backend

Visión general de ETL

Un pipeline de ETL analiza múltiples fuentes de datos para modelar e implementar Cubos ROLAP (Relational Online Analytical Processing). Estos cubos de datos son el método preferencial para el trabajo con big data, debido a su capacidad para procesar un alto volumen de datos sobre la marcha.

Cada cubo contiene observaciones donde se identifican ciertas cantidades que se quieren medir y agregar (medidas), y categorías que se utilizarán para filtrar, agrupar, o agregar las medidas (dimensiones). El proceso de ETL sigue esta secuencia general:

- 1. Análisis exploratorio de datos:** Análisis de las fuentes de datos para encontrar el mejor diseño posible para los cubos. Cada cubo tiene un modelo relacional asociado, y se requiere que las tablas en la base de datos sigan este modelo. En esta etapa el modelo es preliminar. No se opera sobre los datos, sólo se trata de aprovechar al máximo las características de las fuentes de datos en papel.
- 2. Diseño de pipelines de ETL con Bamboo:** Diseño de Pipelines de ETL (Extracción, Transformación y Carga). Los datos se extraen de diversas fuentes, se llevan al formato *tidy* mediante distintas transformaciones siguiendo un modelo [esquema de estrella](#), y se cargan en una base de datos local o remota. Los pipelines de ETL corresponden a una serie de *scripts* de Python 3 que utilizan la librería [bamboo-lib](#), y se pueden ejecutar modificando sus parámetros en tiempo real con la herramienta *dw-bamboo-cli*. Ambas librerías fueron desarrolladas en Datawheel.
- 3. Implementación de Tesseract:** Escribir un esquema de Tesseract. Esto es un archivo .xml donde se define un modelo lógico de los cubos. Incluye las dimensiones, jerarquías, niveles, miembros, y medidas, y se convierte en un modelo físico de la base de datos. Tesseract lee este esquema y genera una API donde se permite hacer consultas a esta base de datos multidimensional y operar con ella, incluyendo Desgloses (*Drilldowns*) y Cortes (*Cuts*) sobre sus dimensiones.

4. Deployment en una máquina virtual:

Desplegar la aplicación en un servidor.

Una vez que el mejor modelo ha sido validado junto a los pipelines y el esquema de Tesseract, es hora de desplegar la aplicación haciendo que Tesseract se ejecute desde una máquina virtual con una IP estática y su propio dominio. El paso de despliegue contempla la instalación del software requerido en la máquina virtual y su correcta configuración para alimentar cualquier solución del front end.

Estructura de la base de datos

Cada uno de los Cubos ROLAP requiere una Tabla de Hechos (*Fact Table*) única y un número cualquiera de Tablas de Dimensión (*Dimension Tables*) conectadas en un modelo relacional. A continuación, se presenta un ejemplo de Tabla de Hechos para un cubo:

Dimension Columns			Measure Columns			
dim_1_id	dim_2_id	dim_3_id	dim_n_id	mea_1	mea_2	mea_m
1	1	2	0	0.15	0.2	101.00
1	1	2	1	0.21	0.4	121.00
1	2	4	0	0.33	0.3	201.00
1	2	8	1	1.23	0.1	32.00
1	3	3	0	2.34	0.5	43.00
2	3	2	1	4.00	0.6	210.00
2	3	1	1	2.12	0.7	325.00
...			...			

Figura 1: Ejemplo de Tabla de Hechos

Como se puede ver, la Tabla de Hechos puede contener tantas dimensiones y medidas como se necesiten. Es importante que las medidas no tengan ningún valor *NULL*, dado que al agregarlas junto con números válidos retornarán un resultado *NULL*.

Ahora se indican algunos ejemplos de dimensiones comunes con las que se trabaja usualmente.

- **Dimensión simple**

La dimensión más simple posible sólo necesita su ID para indicar la relación con la tabla de hechos y una columna de nombres para cada uno de los posibles IDs. Es importante definir un ID que se relacione a valores faltantes o "no definidos". Se sugiere usar el 0, como muestra a continuación:

dim_1_id	dim_1_name
0	Undefined
1	A
2	B
3	C
4	D
5	E
6	F

Figura 2: Tabla de dimensiones

La columna de nombres de la dimensión actuará como etiqueta para cualquier herramienta de *front end* que utilice estos datos, tales como exploradores de datos y visualizaciones.

- **Dimensión jerárquica con niveles**

Otro tipo de dimensión es la que contiene niveles anidados. Las jerarquías no pueden ser irregulares, cada miembro en el nivel más bajo de la tabla solo puede pertenecer a un grupo único en los niveles más altos. Esto es necesario para que funcione el proceso de agregación ROLAP sobre la marcha en la API.

higher_level_id	higher_level_name	lower_level_id	lower_level_name	dim_2_id	dim_2_name
0	Undefined	0	Undefined	0	Undefined
1	A	1	AP	1	APX
1	A	1	AP	2	APY
1	A	2	AQ	3	AQX
2	B	3	BP	4	BPX
2	B	4	BQ	5	BQX
2	B	4	BQ	6	BQY

Figura 3: Tabla de dimensión jerárquica con niveles

Al igual que antes, se sugiere crear una categoría para representar valores faltantes o indefinidos para cada nivel. Además, la columna ID de la dimensión debe relacionarse con el nivel más detallado de la tabla (no hay necesidad de pre-calcular las medidas para hacer agregaciones en niveles superiores).

- **Dimensión en línea**

El último tipo de dimensión analizado ni siquiera necesita ser una tabla en la base de datos. En la imagen de la tabla de hechos, se puede ver que la n-ésima dimensión solo tiene valores 0 y 1. Si hay muy pocos IDs para una dimensión, es posible definir la dimensión línea por línea en el archivo de configuración de Tesseract sin tener que ingestar una tabla en la base de datos con muy pocas filas de información.

Modelo esquema de estrella resultante

El modelo esquema de estrella resultante se vería como muestra la figura 4.

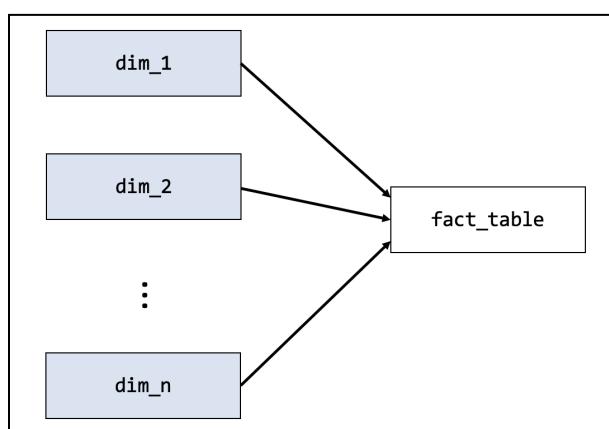


Figura 4: Esquema de Estrella Resultante

Ejemplo 1: Conjunto de datos ficticio

Suponga que existe una empresa internacional que distribuye frutas y verduras mediante un proceso de teletransportación. Todas las frutas y verduras se venden en una caja unitaria estándar, una venta puede incluir una o varias cajas. Hay datos sobre cada venta, una lista de países de origen y destino y una lista de los productos que se teletransportan. La tarea consiste en encontrar un modelo relacional para estos datos, que se ilustran a continuación:

country_name	iso2_code	iso3_code
Afghanistan	AF	AFG
Albania	AL	ALB
Algeria	DZ	DZA
American Samoa	AS	ASM
Andorra	AD	AND
Angola	AO	AGO
Anguilla	AI	AIA
Antarctica	AQ	ATA
Antigua and Barbuda	AG	ATG
Argentina	AR	ARG
Armenia	AM	ARM
Aruba	AW	ABW
Australia	AU	AUS
Austria	AT	AUT
Azerbaijan	AZ	AZE
Bahamas (the)	BS	BHS
Bahrain	BH	BHR
Bangladesh	BD	BGD
Barbados	BB	BRB

product_name	product_category	product_color
Red Apples	fruit	red
Blood Oranges	fruit	red
Cherries	fruit	red
Cranberries	fruit	red
Red Grapes	fruit	red
Pink/Red Grapefruit	fruit	red
Red Pears	fruit	red
Pomegranates	fruit	red
Raspberries	fruit	red
Strawberries	fruit	red
Watermelon	fruit	red
Beets	vegetable	red
Red Peppers	vegetable	red
Radishes	vegetable	red
Radicchio	vegetable	red
Red Onions	vegetable	red
Red Potatoes	vegetable	red
Rhubarb	vegetable	red
Tomatoes	vegetable	red
Yellow Apples	fruit	yellow
Apricots	fruit	yellow
Cape Gooseberries	fruit	yellow

date	origin_country	destination_country	product	duration	units	amount
1/1/19	ML	KG	nan	13.64	85	425
1/1/19	CG	ET	Yellow Pears	57.25	35	315
1/1/19	CR	CR	Broccoli	92.16	93	1395
1/1/19	JO	KZ	Raspberries	45.56	24	168
1/1/19	CZ	BH	Pomegranates	26.77	92	552
1/1/19	KP	US	Mushrooms	22.17	67	603
1/1/19	AS	MU	Spinach	40.01	51	408
1/1/19	PT	PW	Sweet Potatoes	37.89	62	310
1/1/19	NF	VC	nan	80.66	58	696
1/1/19	PA	BZ	White Peaches	0.36	10	60
1/1/19	FK	MP	Garlic	9.98	37	518
1/1/19	PE	MC	Peas	26.79	55	275
1/1/19	UY	PH	Yellow Potatoes	99.03	28	280
1/1/19	NP	RO	Green Peas	96.69	17	204
1/1/19	AO	KE	Yellow Figs	86.75	26	182
1/1/19	SX	GN	Green Cabbage	41.85	91	1274
1/1/19	MP	YE	Yellow Figs	28.46	95	1235
1/1/19	CU	IM	nan	33.08	19	171
1/1/19	PS	BY	Blackberries	26.7	66	594
1/1/19	PA	KE	Jerusalem Artichokes	2.86	99	1089
1/1/19	PN	GL	Leeks	88.18	81	405
1/1/19	CK	UG	Chinese Cabbage	59.06	15	120
1/1/19	UG	GL	Purple Grapes	75.87	88	616

A partir de estas fuentes de datos, se pueden reconocer las siguientes dimensiones y medidas para construir el cubo:

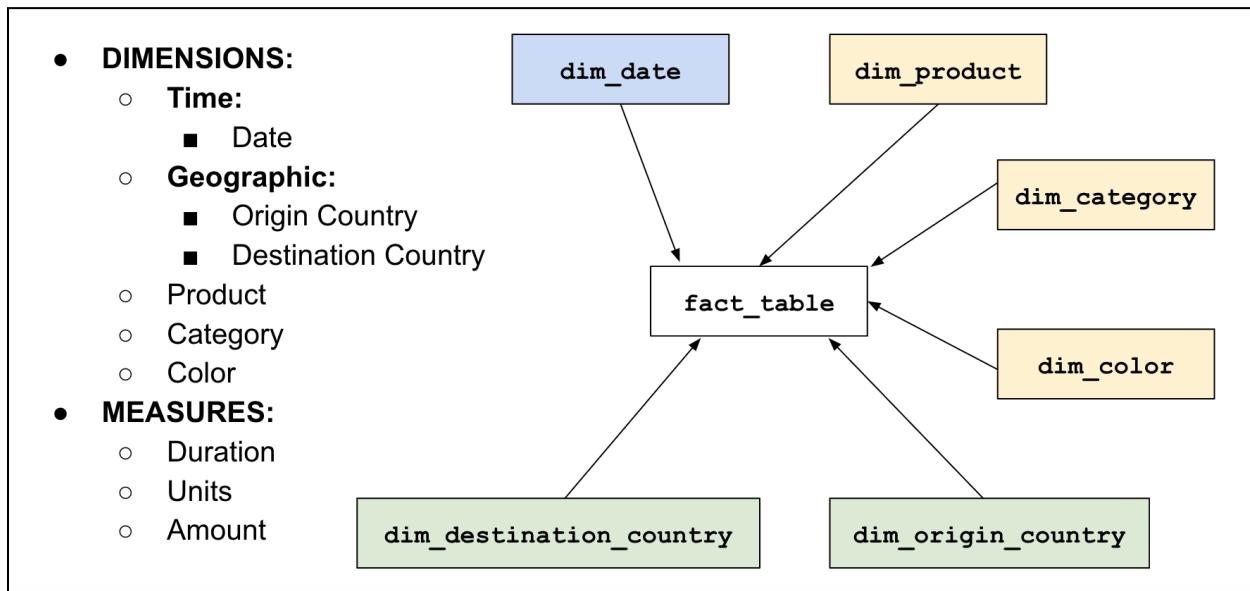


Figura 5: Dimensiones y medidas

Existen algunos puntos importantes respecto a la transformación de los datos:

La columna *date* en la fuente de datos puede ser trabajada para generar una dimensión de tiempo completa, es decir, tener agregaciones por días, semanas, meses, trimestres y años.

Se observan columnas para *product_name*, *product_category* y *product_color* en la misma fuente de datos. Sin embargo, no generan una jerarquía de niveles donde se puedan agregar a miembros de un nivel inferior a uno superior, funcionan mejor como dimensiones aparte, por eso es necesario separar este archivo y generar tres tablas de dimensión a partir de él.

En este ejemplo es posible hacer una dimensión compartida para los países desde el archivo de fuente, pero eso no permite hacer desgloses o cortes correctamente a la dimensión porque no se podría diferenciar entre origen y destino. Es mejor crear dos tablas de país, especificando las dimensiones de país de origen y destino.

Resultado

El modelo lógico final para el cubo quedará como sigue:

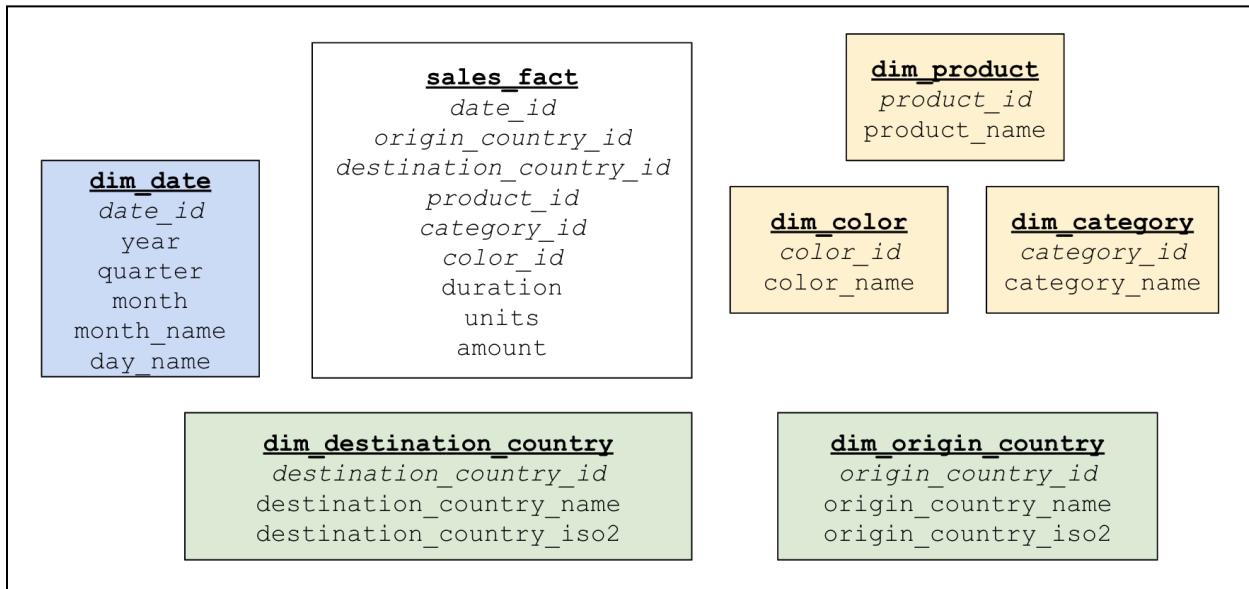


Figura 6: Modelo lógico

Se observa que todas las tablas de dimensión están apuntando a la tabla de hechos. Se puede ver en cursiva cuales son las llaves primarias que conectan cada tabla. Una vez que se tiene una base de datos con esta clase de modelo se puede escribir el archivo de esquema de Tesseract fácilmente y configurar *Tesseract-UI + Vizbuilder* para explorar los datos a través de los cubos *ROLAP* generados.

Ejemplo 2: Cubo de datos INEGI-DENUE

1. Análisis Exploratorio de Datos

Este conjunto de datos se encuentra en la sección del Directorio Estadístico Nacional de Unidades Económicas de INEGI. La fuente de datos corresponde a una serie de archivos en formato CSV que han sido previamente almacenados en un Google Cloud Bucket (GCB).

Cada uno de estos archivos posee una estructura similar como se observa en la figura 7 (por un tema de espacios, la imagen omite algunas columnas):

id	nom_estab	raz_social	codigo_act	nombre_act	per_ocu	tipo_vial	nom_via	tipo_v_e_1	nom_v_e_1
6186029	MAQUINARIA Y CONSTRUCCIONES CAFA, S.A. DE C.V.	MAQUINARIA Y CONSTRUCCIONES CAFA, S.A. DE C.V.	212321	Minería de arena y grava para la construcción	11 a 30 personas	CARRETERA	CARRERA FEDERAL LIBRE TRAMO AGUASCALIENTES J...	CALLE	NINGUNO
6186016	MATERIALES GERO	MATERIALES GERO S.A. DE C.V.	212321	Minería de arena y grava para la construcción	11 a 30 personas	AVENIDA	LIBRAMIENTO CALVILLO-JALPA	CALLE	NINGUNO
6186020	MATERIALES PARA LA CONSTRUCCIÓN SIN NOMBRE	NaN	212321	Minería de arena y grava para la construcción	0 a 5 personas	BOULEVARD	BOULEVARD RODOLFO LANDEROS GALLEGOS	CALLE	NINGUNO
6186028	MATERIALES PUENTE NEGRO DE COSIO, S.A. DE C.V.	MATERIALES PUENTE NEGRO DE COSIO, S.A. DE C.V.	212321	Minería de arena y grava para la construcción	51 a 100 personas	AVENIDA	MORELOS	CALLE	IGNACIO ALLENDE PONIENTE
6186021	MATERIALES TRITURADOS DE AGUASCALIENTES, S.A. ...	MATERIALES TRITURADOS DE AGUASCALIENTES, S.A. ...	212321	Minería de arena y grava para la construcción	11 a 30 personas	CALLE	LIBERTAD	AVENIDA	CONVENCIÓN DE 1914 NORTE
...
6188944	TRITURADOS SOL Y LUNA, S.A. DE C.V.	TRITURADOS SOL Y LUNA, S.A. DE C.V.	212311	Minería de piedra caliza	11 a 30 personas	CARRETERA	SALTILLO - ZACATECAS KILOMÉTRICO 223	NaN	NINGUNO
6703430	UNIDAD MINERA SAN JOSÉ	ARIAN SILVER DE MEXICO, S.A. DE C.V.	212222	Minería de plata	31 a 50 personas	CALLE	DEL MINERAL	NaN	NINGUNO
6417577	UNIDAD SAN MARTÍN	INDUSTRIAL MINERA MEXICO SA DE CV	212231	Minería de cobre	101 a 250 personas	PROLONGACIÓN	MIGUEL HIDALGO	NINGUNO	NINGUNO
6188970	YACIMIENTO DE GEODAS EL PANDITO	NaN	212399	Minería de otros minerales no metálicos	0 a 5 personas	CALLE	LÁZARO CÁRDENAS	NaN	CERRO EL PANDITO

Figura 7: Columnas del archivo

Una vez analizado el contenido de los archivos fuente, se deben llevar al formato tidy para facilitar su análisis, pero los archivos ya cuentan con los siguientes requerimientos:

- Cada fila en los archivos corresponde a una observación.
- Cada columna en los archivos es una dimensión.

Por lo tanto los archivos fuentes ya vienen en formato tidy. El siguiente paso es identificar las dimensiones y medidas para el modelo relacional de Tesseract, pero antes hay que decidir cuáles son las columnas que realmente se van a utilizar.

A continuación se entrega una explicación breve de cada columna:

- **id:** Corresponde al ID de la empresa, se mantendrá dentro de la *Fact Table* como una dimensión del cubo de datos.
- **cod_act:** Se cambiará su nombre a *national_industry_id* y corresponderá a la dimensión de Industria (*Industry*) en el cubo. Se utilizará una tabla de dimensión externa para aprovechar esta información.
- **per_ocu:** Se cambiará su nombre a *n_workers*, corresponderá a la dimensión Tamaño de Empresa (*Company Size*). Como las categorías de esta dimensión son pocas, se definirá como una tabla de dimensión en línea (escrita directamente en el esquema, sin necesidad de crearla en la base de datos).
- **cod_postal:** Se cambiará su nombre a *postal_code*, se mantendrá en la *Fact Table* pero no será asociada a ninguna dimensión.
- **cve_ent y cve_mun:** Estas columnas se utilizarán para generar un ID de Municipio único llamado *mun_id*, con el que se traerá la dimensión compartida de Geografía (*Geography*).
- **ageb, manzana y cve_loc:** Estas columnas serán eliminadas en el proceso de limpieza y no se asociarán a ninguna dimensión.
- **tipoUniEco:** Corresponde al tipo de unidad económica, se cambiará su nombre a *establishment* para indicar si la dimensión de Establecimiento (*Establishment*) corresponde a Fijo o Semifijo según el glosario del INEGI. Se creará una tabla de dimensión en línea en el esquema de Tesseract.
- **latitud y longitud:** Se cambiará su nombre a *latitude* y *longitude* y se mantendrán en la *Fact Table* pero no serán asociadas a ninguna dimensión.

- **fecha_alta:** Se cambiará su nombre a *directory_added_date* y se mantendrá en la *Fact Table* sin asociar a ninguna dimensión.

Además de las columnas presentes en el archivo, se construirán algunas adicionales:

- **publication_date:** Se construirá a partir de la fecha que se entregue como parámetro en el pipeline, se utilizará como llave para la dimensión de Fecha (*Date*).
- **lower, middle, upper:** Se realizará un cálculo adicional para el tamaño de la empresa, llenando estas columnas con el número mínimo y máximo de empleados de un rango, además de su punto medio. Se convertirán en las medidas del cubo (*Number of Employees LCI, Midpoint y UCI*).
- **Companies:** Se definirá una medida extra en el cubo que contará el total de empresas dentro de la respuesta posterior a los desgloses y cortes en el cubo.

El modelo relacional quedará como muestra a continuación:

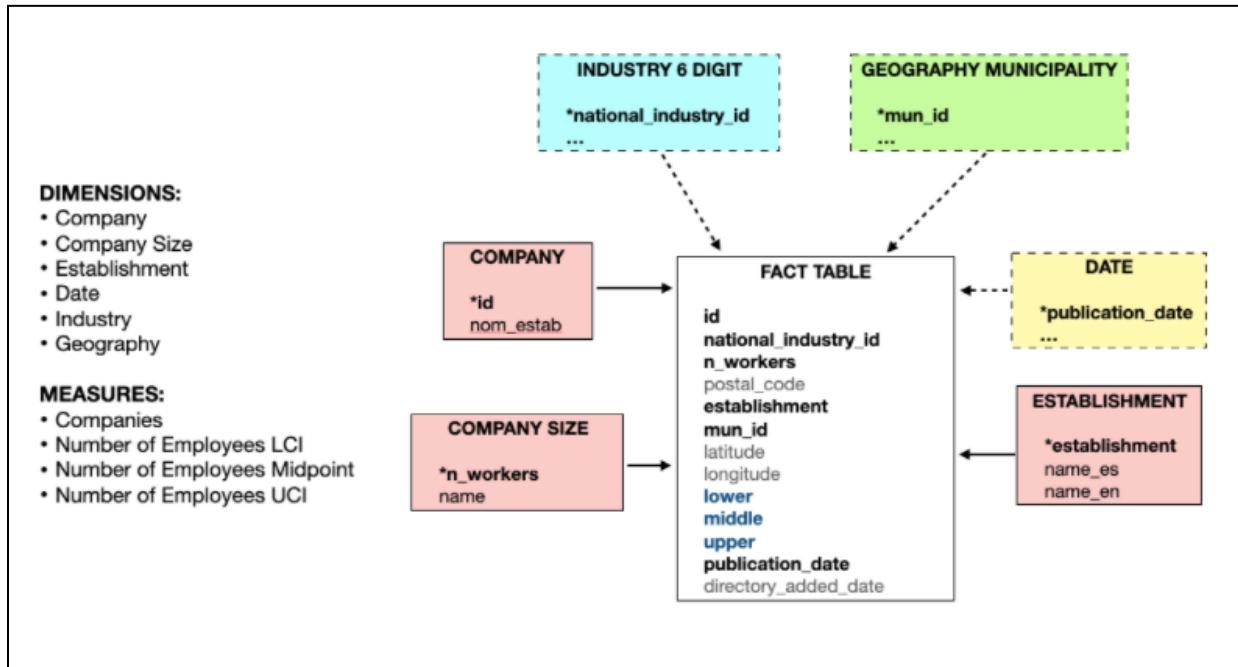


Figura 8: Modelo relacional

2. Diseño de Pipelines de ETL

Dado que se deben procesar varios archivos CSV en distintas carpetas, es una buena práctica tener un *script* que organice la ejecución secuencial del pipeline para cada archivo:

run.py

```
import os
import glob
import pandas as pd
from google.cloud import storage

storage_client =
storage.Client.from_service_account_json(os.environ.get('GOOGLE_APPLICATION_CREDENTIALS'))
bucket = storage_client.get_bucket('datamexico-data')
print('Check for available folders in "datamexico-data/denue", leave blank to ingest all or just "exit"')
folder = input('Enter folder: ')
blobs = bucket.list_blobs(prefix='denue/{}/'.format(folder))
routes = [x.name for x in blobs if x.name != 'denue/{}/'.format(folder)]

if folder != 'exit':
    for url in routes:
        if 'index' not in url:
            date = url.split('denue/')[1].split('/')[0].replace('_', '-')
            date = (date.split('-')[2] + '-' + date.split('-')[1] + '-' +
date.split('-')[0]).replace('-', '')
            # runs pipeline
            os.system("bamboo-cli --folder . --entry companies_pipeline --date=" +
date + " --url=" + url)

else:
    pass
```

Luego comienza el diseño del pipeline de ETL, en tres partes principales:

1. Bloque de Importación

Importación de librerías de Python necesarias para el diseño del pipeline, en particular pandas y las clases de Bamboo.

```
import numpy as np
import pandas as pd
from bamboo_lib.connectors.models import Connector
from bamboo_lib.models import EasyPipeline, PipelineStep, Parameter
from bamboo_lib.steps import DownloadStep, LoadStep
```

2. Pasos Personalizados

Se definen dos pasos personalizados, uno para lectura de datos llamado *ReadStep()* y otro que realiza la transformación de los datos llamado *TransformStep()*.

```
class ReadStep(PipelineStep):
    def run_step(self, prev, params):
        # read data
        try:
            df = pd.read_csv(prev, encoding='utf-8', dtype='str', usecols=[0, 3, 5,
25, 26, 28, 30, 32, 33, 37, 38, 39, 40])
            df.columns = ['id', 'codigo_act', 'per_ocu', 'cod_postal', 'cve_ent',
'cve_mun', 'cve_loc', 'ageb', 'manzana', 'tipounieco', 'latitud', 'longitud',
'fecha_alta']
        except:
            df = pd.read_csv(prev, encoding='latin-1', dtype='str', usecols=[0, 3, 5,
25, 26, 28, 30, 32, 33, 37, 38, 39, 40])
            df.columns = ['id', 'codigo_act', 'per_ocu', 'cod_postal', 'cve_ent',
'cve_mun', 'cve_loc', 'ageb', 'manzana', 'tipounieco', 'latitud', 'longitud',
'fecha_alta']
        return df

class TransformStep(PipelineStep):
    def run_step(self, prev, params):
        df = prev
```

```

# format
df.cve_mun = df.cve_mun.str.zfill(3)
df.cve_loc = df.cve_loc.str.zfill(4)

# create columns
df.cve_mun = df.cve_ent + df.cve_mun
df['cve_mun'] = df['cve_mun'].astype('int')

df.drop(columns=['ageb', 'manzana', 'cve_ent'], inplace=True)

df.per_ocu = df.per_ocu.str.replace('personas', '').str.strip()
df.per_ocu = df.per_ocu.str.replace(' a ', ' - ')
df.per_ocu = df.per_ocu.str.replace(' y más', ' +')
df.per_ocu = df.per_ocu.str.replace(' y més', ' +')
df.fecha_alta = df.fecha_alta.str.replace('-', '')

# date processing
df.fecha_alta = df.fecha_alta.str.upper()
months = {'ENERO': '01',
          'FEBRERO': '02',
          'MARZO': '03',
          'ABRIL': '04',
          'MAYO': '05',
          'JUNIO': '06',
          'JULIO': '07',
          'AGOSTO': '08',
          'SEPTIEMBRE': '09',
          'OCTUBRE': '10',
          'NOVIEMBRE': '11',
          'DICIEMBRE': '12'}

for key, val in months.items():
    for date in df.fecha_alta.unique().tolist():
        try:
            if key in date:
                temp = date.replace(key, val).split()[1] + date.replace(key,
val).split()[0]
                df.fecha_alta = df.fecha_alta.str.replace(date, temp)
        except:

```

```

        continue

#range creation
df['lower'] = np.nan
df['upper'] = np.nan
df['middle'] = np.nan

for ele in df.per_ocu.unique():
    try:
        if '-' in ele:
            df.loc[df.per_ocu == ele, 'lower'] = int(ele.split(' - ')[0])
            df.loc[df.per_ocu == ele, 'upper'] = int(ele.split(' - ')[1])
            df.loc[df.per_ocu == ele, 'middle'] = (float(ele.split(' - ')[1]) +
+ float(ele.split(' - ')[0]))/2.0
        else:
            df.loc[df.per_ocu == ele, 'lower'] = int(ele.split(' +')[0])
            df.loc[df.per_ocu == ele, 'upper'] = int(ele.split(' +')[0])
            df.loc[df.per_ocu == ele, 'middle'] = int(ele.split(' +')[0])
    except:
        continue

# replace values
workers = {
    '0 - 5': 1,
    '6 - 10': 2,
    '11 - 30': 3,
    '31 - 50': 4,
    '51 - 100': 5,
    '101 - 250': 6,
    '251 +': 7
}
df.per_ocu.replace(workers, inplace=True)

place = {
    'Fijo': 1,
    'Semifijo': 2,
    'Actividad en vivienda': 3
}
df.tipounieco.replace(place, inplace=True)

```

```

# rename column names
column_names = {
    'codigo_act': 'national_industry_id',
    'per_ocu': 'n_workers',
    'cod_postal': 'postal_code',
    'tipounieco': 'establishment',
    'fecha_alta': 'directory_added_date',
    'cve_mun': 'mun_id',
    'latitud': 'latitude',
    'longitud': 'longitude'
}
df.rename(columns=column_names, inplace=True)
df.postal_code = df.postal_code.str.replace('0', '0')
for code in df.postal_code.unique():
    try:
        float(code)
    except:
        df.postal_code.replace(code, np.nan, inplace=True)

# data types conversion
dtypes = {
    'id': 'int',
    'national_industry_id': 'str',
    'directory_added_date': 'int',
    'n_workers': 'int',
    'postal_code': 'str',
    'mun_id': 'int',
    'establishment': 'int',
    'latitude': 'float',
    'longitude': 'float',
    'lower': 'int',
    'middle': 'float',
    'upper': 'int'
}
df['directory_added_date'] =
df['directory_added_date'].astype(str).str.replace(' ', '')

for key, val in dtypes.items():
    try:
        # string column check

```

```

        if (df.loc[:, key].isnull().sum() > 0) & (key == 'str'):
            df.loc[:, key] = df.loc[:, key].astype('object')
        else:
            df.loc[:, key] = df.loc[:, key].astype(val)
    except Exception as e:
        if val == 'int':
            df.loc[:, key] = df.loc[:, key].astype('float')
        else:
            print(e)

    df['publication_date'] = int(params['date'])

return df

```

3. Esqueleto del Pipeline

En esta sección se definen los parámetros del pipeline, las conexiones a la base de datos donde se realizará la carga de datos, instancias de los pasos definidos arriba o importados y la lógica de funcionamiento del pipeline.

```

class DENUEPipeline(EasyPipeline):
    @staticmethod
    def parameter_list():
        return [
            Parameter(label="Date", name="date", dtype=str),
            Parameter(label="URL", name="url", dtype=str)
        ]

    @staticmethod
    def steps(params):

        db_connector = Connector.fetch('clickhouse-database', open('../conns.yaml'))

        dtypes = {
            'id': 'UInt32',
            'national_industry_id': 'String',
            'n_workers': 'UInt8',
            'postal_code': 'String',
            'establishment': 'UInt8',

```

```

'mun_id': 'UInt16',
'latitude': 'Float32',
'longitude': 'Float32',
'lower': 'UInt8',
'middle': 'Float32',
'upper': 'UInt8',
'publication_date': 'UInt32',
'directory_added_date': 'UInt32'
}

download_step = DownloadStep(
    connector='data',
    connector_path="conns.yaml"
)

read_step = ReadStep()
transform_step = TransformStep()
load_step = LoadStep('inegi_denue', connector=db_connector,
if_exists='append', pk=['id', 'mun_id', 'national_industry_id'], dtype=dtypes,
nullable_list=['n_workers', 'postal_code', 'establishment', 'latitude', 'longitude',
'directory_added_date', 'lower', 'middle', 'upper'])

return [download_step, read_step, transform_step, load_step]

```

Una vez finalizados los tres pasos anteriores, se puede testear el resultado en un entorno de desarrollo local usando Docker.

Entorno de Desarrollo Local

Utilizando [Docker Desktop](#) y otras herramientas, es posible generar un entorno de desarrollo local y ejecutar el pipeline ahí. A continuación se detallan los pasos para configurar este entorno local.

Configurar un entorno de desarrollo local de ETL usando Docker

En el pasado, han existido problemas con la instalación de algunas tecnologías (Bamboo, Bamboo-CLI, ClickHouse, Tesseract, Tesseract-UI) en los entornos locales de desarrollo y con la realización de tareas de ETL. En esta guía, se muestra cómo instalar todo utilizando Docker, de forma que todas las tecnologías funcionen en forma transversal en todos los sistemas operativos y bajo distintas configuraciones locales.

Un [Contenedor de Docker](#) es una unidad estándar de software que envuelve el código en un paquete junto a sus dependencias, de forma que la aplicación se ejecute rápida y confiablemente desde un entorno computacional a otro. Una imagen de contenedor de Docker es un paquete ejecutable de software, liviano y autónomo, que incluye todo lo necesario para ejecutar una aplicación: código, tiempo de ejecución, herramientas del sistema, librerías del sistema y configuraciones.

Para la instalación, se utilizan tres contenedores de Docker:

- **Python 3.8:** Este contenedor será creado a partir de una [imagen de Docker](#) y tendrá una instalación de Python 3.8, donde se utiliza *bamboo-lib* y *dw-bamboo-cli* sin problemas. Se debe pasar cualquier repositorio de ETL con el que se esté trabajando como un [volumen](#) (método de preferencia para usar data persistente), y de este modo ser capaces de ingestar en cualquier base de datos (local o remota) desde allí.

- **ClickHouse:** Va a contener la base de datos ClickHouse donde se va a ingestar los datos, existe una [imagen de Docker](#) para ella. Es importante publicar los puertos 8123 y 9000 al host (tu sistema), de forma que queden expuestos y se puedan ingestar los datos. Al mirar desde el host al contenedor, la base de datos se encontrará en `localhost:9000`, pero si se mira al contenedor de la BD desde otro contenedor (Python o Tesseract), se encontrará en `host.docker.internal:9000`.
- **Tesseract:** Para construir la imagen, se necesita clonar el siguiente [repositorio](#). Durante la creación del contenedor, se pasará el repositorio de configuración de Tesseract como un volumen, para encontrar el esquema de Tesseract allí, y además publicar el puerto 7777 al host.

Aparte de estos contenedores, se creó un proyecto de prueba para que se pueda instalar el software necesario, clonar estos repositorios listos, crear y ejecutar los contenedores y visualizar la data resultante en Tesseract-UI.

- **`dw-localenv-etl`:** Un repositorio de ETL con un pipeline único de Bamboo que ingesta datos de PIB por país y por año.
- **`dw-localenv-tesseract`:** El repositorio donde se almacena el esquema de Tesseract para la data de PIB.

Además de eso, se asumirá que ya está instalado [Docker](#) para poder trabajar con los contenedores, [Visual Studio Code](#) para escribir código y usar [Git](#), [DBeaver](#) para monitorear la base de datos, [npm and node](#) son necesarios para utilizar Tesseract-UI en el último paso.

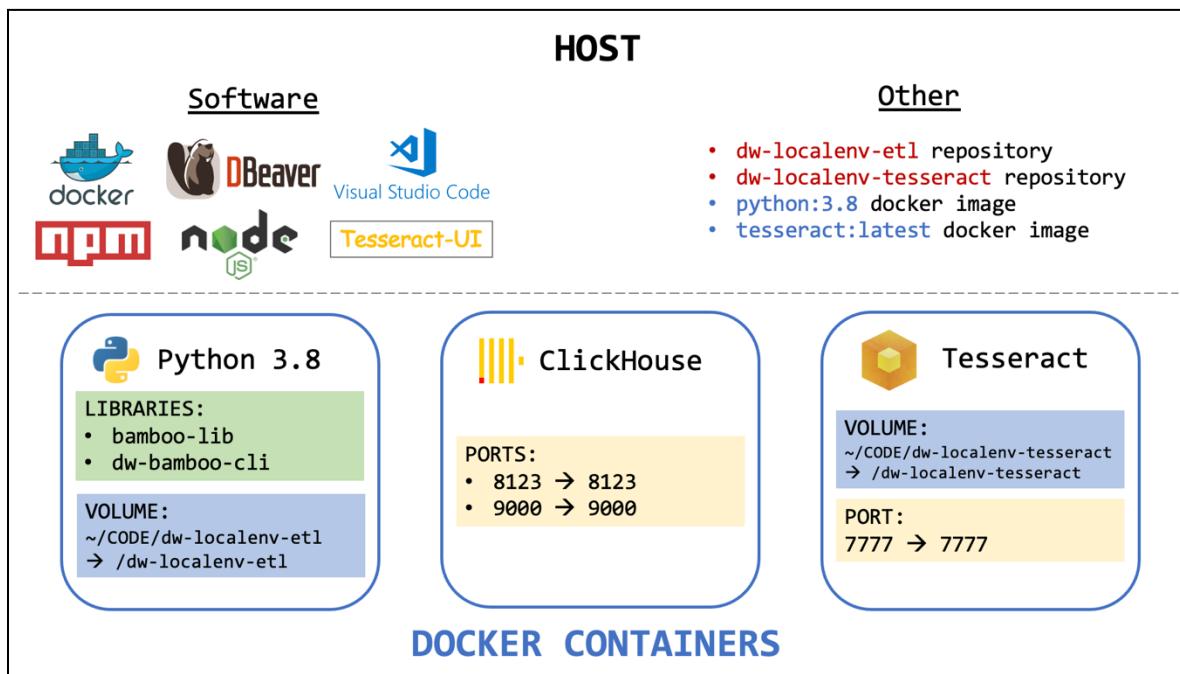


Figura 9: Contenedores Docker

Pasos de instalación

1. Contenedor de ClickHouse Server

Abrir una terminal e iniciar el contenedor con:

```
docker run -d --name clickhouse-local -p 8123:8123 -p 9000:9000
--ulimit nofile=262144:262144 yandex/clickhouse-server:19
```

Si nunca se ha hecho el pull de la imagen de Docker, el proceso será realizado automáticamente, se utiliza la versión 19.17, la última versión 21.2 no funciona con Tesseract.

Nótese que se abren los puertos 8123 y 9000 con **-p** y también que el contenedor se ejecutará en modo desprendido (**-d**) de forma que no se detenga automáticamente. Una vez que el proceso ha finalizado, se puede abrir Docker Desktop y verificar que la instancia de *ClickHouse* está en ejecución. Debería verse como en la imagen siguiente:

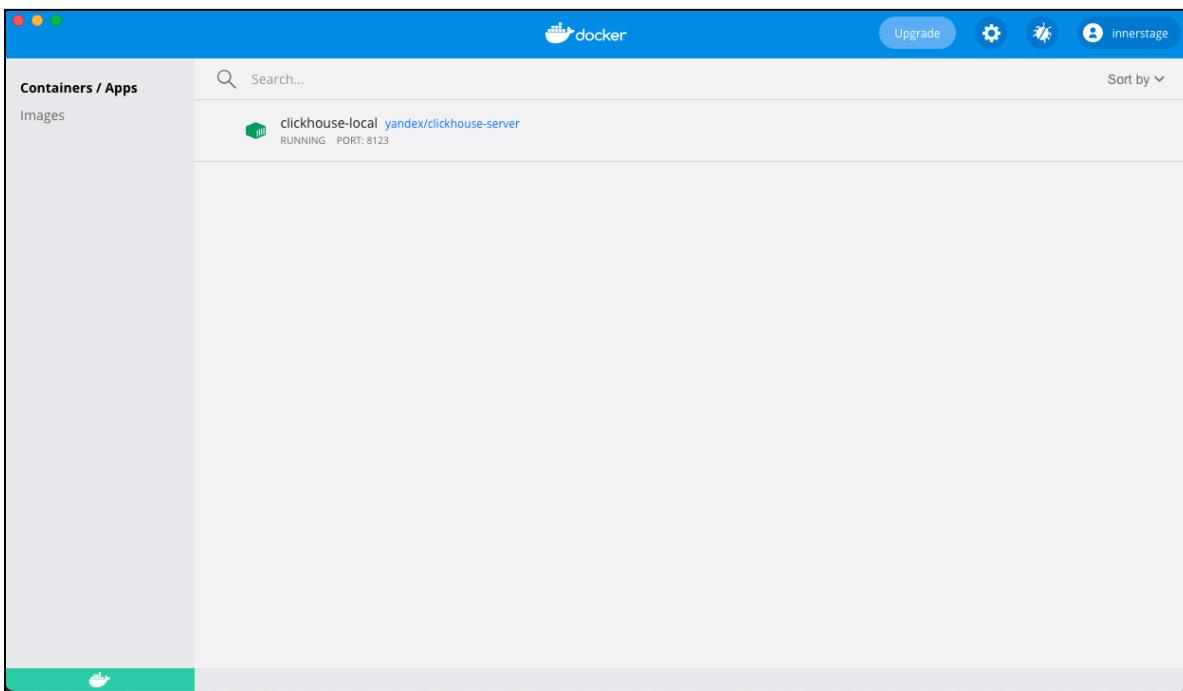


Figura 10: Docker Desktop con instancia de ClickHouse en ejecución

El nombre de usuario por defecto en el servidor es *default* y la contraseña está vacía. En este punto, se puede usar *DBeaver* para configurar una conexión hacia el servidor y verificar que está vacío.

Ahora se muestran algunas operaciones básicas que se necesitan para ejecutar un pipeline de ETL o durante el debugging.

Ingresar al clickhouse-client

Entrar al *clickhouse-client* dentro del contenedor para realizar toda clase de operaciones con bases de datos: realizar queries con SQL, eliminar tablas, crear y eliminar bases de datos, etc. Para poder acceder, se ejecuta el contenedor en modo interactivo:

```
docker exec -it clickhouse-local bash
```

Una vez ingresado, serás el usuario o usuaria *root* del sistema, teniendo poder absoluto en cada comando que se utilice.

Ahora se puede acceder al *clickhouse-client*, crear una base de datos llamada *my_test_database* y luego eliminarla. Es necesario salir del *clickhouse-client* y luego salir de nuevo para abandonar el contenedor y volver a tu sistema:

```
clickhouse-client
CREATE DATABASE my_test_database
DROP DATABASE my_test_database
exit
exit
```

Se utiliza la base de datos *default* en el ejemplo, así que todo está listo para el siguiente paso.

2. Contenedor de Python

En este paso se clona el repositorio de ETL. Se debe ingresar a la carpeta *CODE* o cualquier directorio en el que se guarden repositorios y clonarlo ahí.

Al ingresar estos comandos en una terminal de VSCode se puede abrir el repositorio y trabajar con los archivos de inmediato:

```
cd ~/CODE
git clone https://github.com/innerstage/dw-localenv-etl.git
cd dw-localenv-etl
code -r .
```

Se creó una plantilla para las variables de entorno que se requieren para la ingestión en un archivo llamado *template.env*. Se recomienda utilizarla para crear el propio archivo *.env* con las variables.

Es necesario modificar *localhost:9000* en *DB_URL* por *host.docker.internal:9000* para que pueda encontrar la base de datos.

Ahora es posible ejecutar el contenedor de Docker, utilizando el siguiente comando:

```
docker run -it -v ~/CODE/dw-localenv-etl:/dw-localenv-etl
--name=python3-local python:3.8 bash
```

Nótese que se ejecuta en modo interactivo (`-it`) y pasando la carpeta del repositorio en el host al contenedor (`-v <host_dir>:<container_dir>`). Cualquier cambio que se haga de ahora en adelante en la carpeta del repositorio va a persistir en el contenedor, esto es muy útil para testear, porque se pueden editar los pipelines en VSCode y estos cambios van a estar disponibles de inmediato en el contenedor donde se pueden tratar de ejecutar nuevamente.

Si no se ha hecho pull de la imagen `python:3.8` con anterioridad, Docker va a realizar el proceso antes de ejecutar el contenedor. Una vez en el contenedor, se puede chequear que la versión de Python es correcta, y luego instalar las librerías necesarias.

```
python -V
pip install bamboo-lib dw-bamboo-cli
```

Si se olvida añadir una carpeta necesaria como volumen, o aparece algún otro problema con el contenedor de Python, siempre se puede ir a la ventana de Docker Desktop y eliminarlo, luego se crea uno nuevo.

Ahora, asumiendo que las variables de entorno están correctas, se puede continuar e ingestar la data:

```
cd dw-localenv-etl
source .env
bamboo-cli --folder . --entry gdp_fact_pipeline
--input-file=gdp-source --output-db=clickhouse-local --ingest=True
```

Se puede revisar en DBeaver, que los datos se ingestaron correctamente en la base de datos. Deben haber dos tablas (`dim_country` y `gdp_fact`) en la base de datos `default`. Ahora se puede continuar con el siguiente paso.

3. Contenedor de Tesseract

De forma similar, se inicia preparando este paso con la clonación del repositorio de configuración de Tesseract, el que trae el esquema:

```
cd ~/CODE
git clone https://github.com/innerstage/dw-localenv-tesseract.git
```

Ahora se clona el repositorio de Tesseract-OLAP para construir la imagen desde la que se va a ejecutar el contenedor:

```
git clone https://github.com/tesseract-olap/tesseract.git
cd tesseract
docker build -t tesseract:latest .
```

La construcción de la imagen puede tomar un par de minutos. Si al terminar se observa un montón de líneas en rojo, no es para preocuparse, la mayoría son advertencias o información de los logs, pero ninguna es un error real.

Luego de que la imagen ha sido construida, se puede ejecutar un contenedor de Tesseract, este es un poco más complejo por la cantidad de elementos que se deben cubrir:

- Publicar el puerto 7777 (-p 7777:7777).
- Pasar el repositorio de configuración de Tesseract como un volumen (-v ~/CODE/dw-localenv-tesseract:/dw-localenv-tesseract).
- Pasar múltiples variables de entorno, dos de ellas obligatorias:
 - URL de la base de datos:
(-e TESSERACT_DATABASE_URL='clickhouse://host.docker.internal:9000/default')
 - Directorio del esquema:
(-e TESSERACT_SCHEMA_FILEPATH='/dw-localenv-tesseract/schema.xml')
- Si el esquema está mal por algún motivo y se necesita hacer debugging, se puede eliminar el contenedor y ejecutar uno nuevo añadiendo estas variables, las cuales dan un mayor detalle sobre los errores:
 - Tesseract Debug (-e TESSERACT_DEBUG=true)
 - Rust Log (-e RUST_LOG=info o -e RUST_LOG=debug para mayor detalle)

En conclusión, para ejecutar este contenedor se tiene que hacer algo como esto:

```
docker run -p 7777:7777 -v ~/CODE/dw-localenv-tesseract:/dw-localenv-tesseract -e
TESSERACT_DATABASE_URL='clickhouse://default:@host.docker.internal:9000/default' -e
TESSERACT_SCHEMA_FILEPATH='/dw-localenv-tesseract/schema.xml' -e TESSERACT_DEBUG=true
-e RUST_LOG=debug --name=tesseract-local tesseract:latest
```

Si la terminal se ve como en la imagen siguiente luego de ejecutar la línea de código anterior, Tesseract se está ejecutando correctamente:

```

INFO tesseract_olap > Geoservice URL not provided
INFO tesseract_olap::logic_layer::cache > Populating cache...
INFO clickhouse_rs > try to connect to tcp://default@host.docker.internal:9000
INFO clickhouse_rs > [hello] <- ClickHouse 19.17.54428 (Etc/UTC)
INFO clickhouse_rs > [ping]
INFO clickhouse_rs > [pong]
INFO clickhouse_rs::types::query_result > [send query] select distinct country_id from dim_country
INFO tesseract_clickhouse > Time for sql execution: 0.018
INFO clickhouse_rs > [ping]
INFO clickhouse_rs > [pong]
INFO clickhouse_rs::types::query_result > [send query] select distinct year from gdp_fact
INFO tesseract_clickhouse > Time for sql execution: 0.016
INFO clickhouse_rs > [ping]
INFO clickhouse_rs > [pong]
INFO clickhouse_rs::types::query_result > [send query] select distinct year from gdp_fact
INFO tesseract_clickhouse > Time for sql execution: 0.013
INFO tesseract_olap::logic_layer::cache > Cache ready! (Time elapsed: 0.050)
INFO actix_net::server::server > Starting 2 workers
INFO actix_net::server::server > Starting server on 0.0.0.0:7777
Tesseract listening on: 0.0.0.0:7777
Tesseract database: default:@host.docker.internal:9000/default, Clickhouse
Tesseract schema path: /dw-localenv-tesseract/schema.xml
Tesseract JWT token protection: OFF
Tesseract debug mode: ON

```

Figura 11: Terminal indicando que Tesseract se ejecuta de forma correcta

Usualmente, se utiliza mucho tiempo haciendo debugging sobre un esquema, así que no hay que desanimarse si no funciona inmediatamente al trabajar con otro proyecto.

Una vez que se esté ejecutando, se puede configurar Tesseract-UI para visualizar los datos y tal vez hacer más tests o modificar el esquema.

Para verificar que todo funciona bien y Tesseract se encuentra activo y esperando requests se puede visitar <http://localhost:7777/> en el navegador y se debe recibir la siguiente respuesta en JSON:

```
{
    "status": "ok",
    "tesseract_version": "0.14.13"
}
```

4. Tesseract-UI

La instalación de Tesseract-UI se realiza a través de npm, el instalador va a crear la carpeta del proyecto automáticamente. Para mantener el orden, se pondrá el proyecto dentro de la carpeta CODE:

```
cd CODE
npm init @datawheel/tesseract-ui dw-localenv-ui
```

El instalador hará algunas preguntas:

```
? Enter the full URL for the OLAP server > http://localhost:7777
? Where will this instance run? > - Use arrow-keys. Return to submit.
> A local computer
```

Si hay algún problema con esta instalación, probablemente se debe a las versiones de node y npm, generalmente la última versión de node funciona bien, siempre se puede eliminar y luego reinstalar si el problema sigue.

Ahora, asumiendo que esta instalación funcionó, es posible iniciar el servidor del front end para visualizar los datos usando:

```
cd dw-localenv-ui
npm start
```

Luego se abre la siguiente URL en el navegador para ver el resultado: <http://localhost:4000>. Si el puerto está ocupado, la misma terminal indicará el puerto en el que se está ejecutando Tesseract.

Una vez logrado esto, se pueden usar estos contenedores, o nuevos contenedores creados a necesidad del usuario para hacer debug de otros pipelines de ETL, ingestar los datos en *ClickHouse*, testear esquemas de Tesseract y visualizar los datos; sin importar el sistema operativo utilizado.

Links de acceso a los recursos mencionados en el anexo

- Contenedor de Docker: <https://www.docker.com/resources/what-container>
- Imagen de Docker: https://hub.docker.com/_/python
- Volumen de Docker: <https://docs.docker.com/storage/volumes/>
- Clickhouse Server Docker: <https://hub.docker.com/r/yandex/clickhouse-server>
- Repositorio: <https://github.com/tesseract-olap/tesseract>
- Instalación de Docker: <https://www.docker.com/products/docker-desktop>
- Instalación de VSC: <https://code.visualstudio.com/>
- Instalación de Git: <https://git-scm.com/>
- Instalación de DBeaver: <https://dbeaver.io/>
- Instalación de npm y node: <https://nodejs.org/en/>