Normalización de Datos

In [37]:
```
from typing import List, Iterable, Set
from itertools import chain, combinations
from pathlib import Path
import pandas as pd
import joblib
import matplotlib.pyplot as plt
from enum import StrEnum
from tqdm import tqdm, trange
```

In [4]:
```
DATA_DIR = Path('data').resolve()

MARZO_PATH = DATA_DIR / 'Marzo-2021.xlsx'
ABRIL_PATH = DATA_DIR / 'Abril-2021.xlsx'

PKLS_PATH = DATA_DIR / 'pkls'

MES = 'Abril'
PATH_MES = {
    'Marzo': MARZO_PATH,
    'Abril': ABRIL_PATH
}
```

In [5]:
```
class TableName(StrEnum):
    SERVICIOS = 'Servicios'
    REP_LEGALES = 'RepLegales'
    VEHICULOS = 'Vehiculos'
    O_D_RECORRIDOS = 'O-D Recorridos'
    TRAZADOS = 'Trazados'

    @classmethod
    def get_sheet_names(cls) -> list[str]:
        return [table_name.value for table_name in cls]

def get_data(path: str, sheet_name: str) -> pd.DataFrame:
    """Get the data from an Excel file."""
    return pd.read_excel(path, sheet_name=sheet_name)
```

In [6]:
```
# read as CSVs

# servicios = get_data(PATH_MES[MES], TableName.SERVICIOS.value)
# rep_legales = get_data(PATH_MES[MES], TableName.REP_LEGALES.value)
# vehiculos = get_data(PATH_MES[MES], TableName.VEHICULOS.value)
# recorridos = get_data(PATH_MES[MES], TableName.O_D_RECORRIDOS.value)
# trazados = get_data(PATH_MES[MES], TableName.TRAZADOS.value)
```

In [7]:
```
# save as pickles for faster loading

# joblib.dump(servicios, PKLS_PATH / f'{MES}_servicios.pkl')
# joblib.dump(rep_legales, PKLS_PATH / f'{MES}_rep_legales.pkl')
# joblib.dump(vehiculos, PKLS_PATH / f'{MES}_vehiculos.pkl')
# joblib.dump(recorridos, PKLS_PATH / f'{MES}_recorridos.pkl')
# joblib.dump(trazados, PKLS_PATH / f'{MES}_trazados.pkl')
```

In [182…
```
# load pickles
servicios = joblib.load(PKLS_PATH / f'{MES}_servicios.pkl')
rep_legales = joblib.load(PKLS_PATH / f'{MES}_rep_legales.pkl')
vehiculos = joblib.load(PKLS_PATH / f'{MES}_vehiculos.pkl')
recorridos = joblib.load(PKLS_PATH / f'{MES}_recorridos.pkl')
trazados = joblib.load(PKLS_PATH / f'{MES}_trazados.pkl')
```

Identificación de llaves

In [27]:
```
def is_candidate_key(df: pd.DataFrame, cols: List[str]) -> bool:
    """Check if the given columns are a candidate key for the given DataF
    return df[cols].drop_duplicates().shape[0] == df.shape[0]
```

Servicios

In [28]:
```
servicios.head(3)
```

Out[28]:

	REGION	FOLIO	TIPO_SERVICIO	FLOTA VIGENTE	NRO_LINEA	RUT_RESPONSABLE	NC
0	1	10	TAXI COLECTIVO RURAL	1	NaN	NaN	
1	1	4608	TAXI COLECTIVO RURAL	1	NaN	NaN	
2	1	27	TAXI COLECTIVO RURAL	1	NaN	NaN	

Lo más lógico es que la llave primaria de la tabla de servicios sea el número de folio. Veamos primero si {FOLIO} es llave candidata.

In [ ]:
```
is_candidate_key(servicios, ['FOLIO'])
```

```
False
```

Dado que hay valores duplicados para {FOLIO}, no puede ser super llave y por lo tanto no puede ser llave candidata. Lo siguiente más lógico es que, si el número de región se encuentra presente, entonces en realidad hay un número de folio único por región. Veamos si {FOLIO, REGION} es llave candidata.

In [ ]:
```
is_candidate_key(servicios, ['FOLIO', 'REGION'])
```

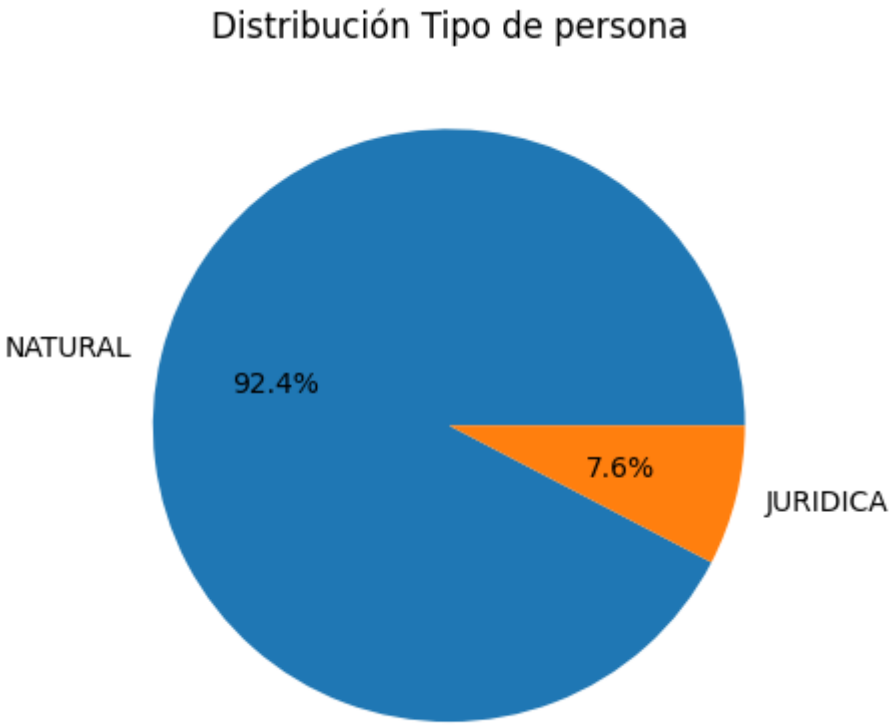
```
True
```

Confirmamos que {FOLIO, REGION} es llave candidata.

In [ ]:
```
is_candidate_key(servicios, ['FOLIO', 'NOMBRE_RESPONSABLE'])
```

```
True
```

In [39]:
```
(
    servicios['TIPO_PERSONA']
    .value_counts()
    .plot(kind='pie', autopct='%1.1f%%')
);
plt.title('Distribución Tipo de persona')
plt.ylabel('')
plt.show();
```



In [68]:
```
servicios_juridicos = servicios[servicios['TIPO_PERSONA'] == 'JURIDICA']
print(
    'Cantidad de servicios por personas jurídicas: '
    f'{servicios_juridicos.shape[0]}'
)
servicios_juridicos.notna().sum()
```

```
Cantidad de servicios por personas jurídicas: 3174
```

Out[68]:
```
REGION                3174
FOLIO                 3174
TIPO_SERVICIO         3174
FLOTA VIGENTE         3174
NRO_LINEA             1533
RUT_RESPONSABLE       3174
NOMBRE_RESPONSABLE    3174
NOMBRE_FANTASIA       1779
TIPO_PERSONA          3174
DOMICILIO             3171
COMUNA                3167
REGION.1              3167
TELEFONO              3006
FAX                    611
EMAIL                 2201
dtype: int64
```

In [69]:
```
servicios_naturales = servicios[servicios['TIPO_PERSONA'] == 'NATURAL']
print(
    'Total de registros para personas naturales: '
    f'{servicios_naturales.shape[0]}'
)
servicios_naturales.notna().sum()
```

```
Total de registros para personas naturales: 38351
```

Out[69]:
```
REGION                38351
FOLIO                 38351
TIPO_SERVICIO         38351
FLOTA VIGENTE         38351
NRO_LINEA               510
RUT_RESPONSABLE           0
NOMBRE_RESPONSABLE    38351
NOMBRE_FANTASIA           5
TIPO_PERSONA          38351
DOMICILIO                 0
COMUNA                    0
REGION.1                  0
TELEFONO                  0
FAX                       0
EMAIL                     0
dtype: int64
```

Como analizamos en el notebook de exploración de datos, la tabla de servicios en realidad se compone de 2 tablas según tipo de persona (natural o jurídica).

Esto lo podemos ver ya que si filtramos para tipo de persona natural, las columnas [DOMICILIO, ..., EMAIL] tienen únicamente valores nulos, ya que esta información se encontraría realmente en la tabla "RepLegales", a la cual se le puede hacer join con el atributo "RUT_RESPONSABLE" de la tabla "Servicios" (de hecho podemos ver que no falta ningún RUT en la tabla de servicios).

En resumen, las últimas 6 columnas de la tabla de servicios (DOMICILIO, COMUNA, REGION.1, TELEFONO, FAX, EMAIL) son en realidad atributos para personas juridicas ya que no tienen valores para personas naturales.

In [71]:
```
cols_datos_personales = ['DOMICILIO', 'COMUNA', 'REGION.1', 'TELEFONO', '
```

In [76]:
```
cols_irrelevantes = ['RUT_RESPONSABLE', 'NOMBRE_FANTASIA', 'TIPO_PERSONA'
servicios_naturales = (
    servicios_naturales
    .drop(columns=cols_datos_personales + cols_irrelevantes)
)
```

In [78]:
```
servicios_naturales.head(3)
```

Out[78]:

	REGION	FOLIO	TIPO_SERVICIO	FLOTA VIGENTE	NRO_LINEA	NOMBRE_RESPONSABLE
0	1	10	TAXI COLECTIVO RURAL	1	NaN	FRANCISCO R. CARLOS MAMAN
1	1	4608	TAXI COLECTIVO RURAL	1	NaN	JORGE ANDRES SOTC LARA
2	1	27	TAXI COLECTIVO RURAL	1	NaN	RENE RUPERTC OYANADEL RIVEROS

In [79]:
```
servicios_naturales.notna().sum()
```

Out[79]:
```
REGION                38351
FOLIO                 38351
TIPO_SERVICIO         38351
FLOTA VIGENTE         38351
NRO_LINEA               510
NOMBRE_RESPONSABLE    38351
dtype: int64
```

Ahora podemos ver que los servicios de personas naturales, tras el tratamiento de datos, ya no tiene valores nulos (excepto por NRO_LINEA que era el que más valores no nulos tenía y lo dejamos por si acaso pero se puede borrar eventualmente).

Tras analizar estos datos, nos damos cuenta de que en realidad distinguir la tabla de servicios en 2 tablas es muy problemático ya que la tabla que tendría realmente datos sería la de personas jurídicas pero esta representa un porcentaje muy pequeño de los servicios, por lo que en realidad no vale la pena separar las tablas, y nos podemos quedar sólo con los atributos que si tienen valores para personas naturales.

In [183…
```
servicios = servicios[servicios_naturales.columns].drop(columns=['NRO_LIN
```

In [184…
```
servicios.head(3)
```

Out[184]:

	REGION	FOLIO	TIPO_SERVICIO	FLOTA VIGENTE	NOMBRE_RESPONSABLE
0	1	10	TAXI COLECTIVO RURAL	1	FRANCISCO R. CARLOS MAMANI
1	1	4608	TAXI COLECTIVO RURAL	1	JORGE ANDRES SOTO LARA
2	1	27	TAXI COLECTIVO RURAL	1	RENE RUPERTO OYANADEL RIVEROS

In [189…
```
is_candidate_key(servicios, ['FOLIO', 'REGION'])
```

Out[189]:
```
True
```

Donde la PK es: {FOLIO, REGION}

Represenantes Legales

In [83]:
```
rep_legales.head(3)
```

Out[83]:

	REGION	FOLIO	TIPO_SERVICIO	NOMBRE_REPLEGAL
0	3	6000	TAXI COLECTIVO RURAL	MIGUEL ANGEL GUAJARDO SANTANA
1	3	6003	TAXI COLECTIVO RURAL	JORGE EDUARDO CAMPILLAY HURTADO
2	3	6005	TAXI COLECTIVO RURAL	OMAR EDDIE MU?OZ GONZALEZ

In [94]:
```
print(
    is_candidate_key(rep_legales, ['NOMBRE_REPLEGAL']),
    is_candidate_key(rep_legales, ['REGION']),
    is_candidate_key(rep_legales, ['NOMBRE_REPLEGAL', 'REGION']),
    is_candidate_key(rep_legales, ['FOLIO']),
    is_candidate_key(rep_legales, ['FOLIO', 'REGION']),
)
print('Ninguna de las combinaciones de columnas es llave candidata.')
print(is_candidate_key(rep_legales, ['FOLIO', 'NOMBRE_REPLEGAL']))
```

```
False False False False False
Ninguna de las combinaciones de columnas es llave candidata.
True
```

La combinación de {FOLIO, NOMBRE_REPLEGAL} no tiene duplicados, pero como sabemos que un folio se identifica con su número de folio y de región, entonces {FOLIO, REGION, NOMBRE_REPLEGAL} es la llave primaria de la tabla de representantes legales.

Así sería, sin embargo, dado que ya no nos interesan los datos personales de los representantes, podemos deshacernos de la tabla completamente.

Vehiculos

In [116…
```
vehiculos.shape[0] == vehiculos.drop_duplicates().shape[0]
```

Out[116]: False

La tabla de vehiculos trae valores duplicados así que los eliminamos.

In [117…
```
init_n = vehiculos.shape[0]
vehiculos = vehiculos.drop_duplicates()
print(f'Cantidad de registros duplicados: {init_n - vehiculos.shape[0]}')
```

```
Cantidad de registros duplicados: 20
```

In [119…
```
is_candidate_key(vehiculos, ['PPU'])
```

Out[119]: True

In [120…
```
is_candidate_key(vehiculos, ['FOLIO', 'REGION'])
```

Out[120]: False

Un vehiculo puede tener sólo un folio pero un folio puede tener varios. vehiculos. Podemos ver que su PK es {PPU}, es decir la patente del vehiculo.

Recorridos

In [122…
```
is_candidate_key(recorridos, recorridos.columns)
```

Out[122]: False

Podemos ver que la tabla de recorridos tiene valores duplicados, por lo que los eliminamos.

In [123…
```
init_n = recorridos.shape[0]
recorridos = recorridos.drop_duplicates()
print(f'Cantidad de registros duplicados: {init_n - recorridos.shape[0]}'
```

```
Cantidad de registros duplicados: 18
```

In [127…
```
recorridos.head(3)
```

Out[127]:

	REGION	FOLIO	TIPO_SERVICIO	NOMBRE_RECORRIDO	TIPO_TRAZADO	NOMB
0	1	800012	BUS INTERURBANO CORRIENTE	T	PRINCIPAL	
1	1	200002	BUS URBANO CORRIENTE	9	PRINCIPAL	TER
2	1	200002	BUS URBANO CORRIENTE	94	PRINCIPAL	TER

Lo primero que notamos es que la tabla trae los lugares de origen y destino en la misma tabla, cuando tendría más sentido tener una tabla de lugares y una tabla de recorridos que la relacione con lugares de origen y destino.

Por lo que en realidad, quiséramos que la tabla luzca así:

Recorrido: REGION | FOLIO | TIPO_SERVICIO | NOMBRE_RECORRIDO | LUGAR_ORIGEN | LUGAR_DESTINO

Lugar: ID_LUGAR | NOMBRE_LUGAR | COMUNA | DOMICILIO

Veamos cómo construir la tabla "Lugar"

In [ ]:
```
nombre_origen = set(recorridos['NOMBRE_ORIGEN'])
nombre_destino = set(recorridos['NOMBRE_DESTINO'])
print(f'Cantidad de nombres de origen: {len(nombre_origen)}')
print(f'Cantidad de nombres de destino: {len(nombre_destino)}')
```

```
print(f'Cantidad de nombres de origen y destino: {len(nombre_origen & nom
print(f'Cantidad de nombres de origen o destino: {len(nombre_origen | nom
```

```
Cantidad de nombres de origen: 2745
Cantidad de nombres de destino: 2671
Cantidad de nombres de origen y destino: 936
Cantidad de nombres de origen o destino: 4480
```

Del análisis anterior, podemos ver que se puede construir la tabla lugar simplemente tomando la union de los nombres, junto a sus detalles. Sin embargo, los datos parecen estar un poco sucios aquí, por lo que puede ser que haya que corregirlos un poco más y/o cruzarlos con datos externos. De todas formas, en el peor de los casos, basta quedarse sólo con el nombre de la comuna ya que debería ser información suficiente para la aplicación.

En otras palabras, si se vuelve muy complicado tener el detalle del lugar, se puede olvidar por completo la tabla "Lugares" y en la tabla "Recorridos", en lugar de apuntar a un id_lugar, se puede poner en nombre de una comuna directamente.

In [190…
```
nombres = nombre_origen | nombre_destino
comunas = set(recorridos['COMUNA_ORIGEN']) | set(recorridos['COMUNA_DESTI
domicilios = set(recorridos['DOMICILIO_ORIGEN']) | set(recorridos['DOMICI
```

Trazados

In [146…
```
trazados.head(3)
```

Out[146]:

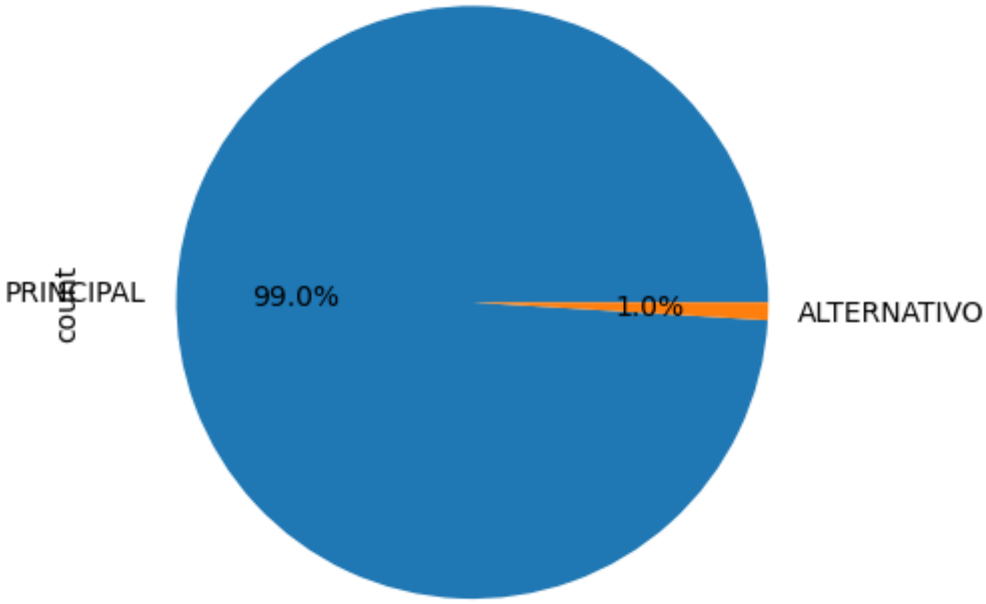
	REGION	FOLIO	TIPO_SERVICIO	NOMBRE_RECORRIDO	TIPO_TRAZADO	ORDEN
0	1	10	TAXI COLECTIVO RURAL	T	PRINCIPAL	1
1	1	10	TAXI COLECTIVO RURAL	T	PRINCIPAL	2
2	1	10	TAXI COLECTIVO RURAL	T	PRINCIPAL	3

In [171…
```
is_candidate_key(
    trazados,
    [
        'REGION',
        'FOLIO',
        'NOMBRE_RECORRIDO',
        'ORDEN',
        # 'SENTIDO',
        # 'CALLE',
        # 'COMUNA',
        # 'TIPO_SERVICIO',
        # 'TIPO_TRAZADO', # ANTES ERA UN ATRIBUTO NECESARIO
    ]
    )
```

Out[171]: True

Primero confirmamos que la tabla de trazados no tenía valores duplicados. Luego probamos combinaciones para ver si encontramos una llave candidata, sin embargo notamos que la llave candidata encontrada era: {FOLIO, REGION, NOMBRE_RECORRIDO, ORDEN, TIPO_TRAZADO}, lo cual tiene sentido, pero veamos cual es la distribución real de TIPO_TRAZADO.

In [167…
```
(
    trazados['TIPO_TRAZADO']
    .value_counts()
    .plot(kind='pie', autopct='%1.1f%%')
);
```



Vemos que se trata principalmente de "PRINCIPAL", representando a un 99% de los datos, por lo que para simplificar el esquema, eliminaremos esta columna dejando solo los valores "PRINCIPAL" y por lo tanto la llave primaria de la tabla de trazados será: {FOLIO, REGION, NOMBRE_RECORRIDO, ORDEN}.

In [170…
```
trazados = (
    trazados[trazados['TIPO_TRAZADO'] == 'PRINCIPAL']
    .drop(columns=['TIPO_TRAZADO'])
)
```

In [155…
```
is_candidate_key(trazados, trazados.columns)
```

Out[155]: True

Confirmamos que sigue siendo llave primaria.