BOOK_PAPER

Jane Doe

2/9/23

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Introduction

Aquí va la introducción y algunos antecedentes Tener muy claro el planteamiento del problema Definir subsecciones

2 Data Acquisition and Preprocessing

Censo de poblacion 2020

Descarga masiva Manzanas

3 Preprocessing

4 Preprocessing

4.1 Índice de concentración de desventajas

Construido a partir de cuatro dimensiónes (con base a censo 2010) y reducidad a una componente principal por PCA. Las dimensiones que se exponen en el articulo son: - Porcentaje de masculinos de 15 a 29 - Porcentaje de población sin servicios a salud - Promedio de habitantes que ocupan un hogar privado - Porcentaje de personas que hablan una lengua indigena

Con base al Censo de población de 2020, las dimensiones se resumen de la forma:

Clave	Descripción
P_15A17_M	Población masculina de 15 a 17 años
P_18A24_M	Población masculina de 18 a 24 años
PSINDER	Población sin afiliación a servicios de salud
PROM_OCUP	Promedio de ocupantes en viviendas particulares habitadas
P3YM_HLI	Población de 3 años y más que habla alguna lengua indígena

> La primer parte consta de cargar la base de datos de censo, seleccionar las dimensiones, limpiar la información y preparala para poder hacerla únion de estas en la geometry de la unidd geografica manzanas. Se cargan las librerias necesarias.

```
## Librerias

import numpy as np
import pandas as pd
import geopandas as gpd
import contextily as ctx
import matplotlib.pyplot as plt
from IPython.display import Markdown
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

### Warnings
import warnings
```

```
warnings.filterwarnings('ignore')
```

Se carga la información del Censo por unidad manzanas conservando unicamente los campos de interes: **ENTIDAD, NOM_ENT, MUN, NOM_MUN LOC, NOM_LOC, AGEB, MZA, POBTOT, P_15A17_M, P_18A24_M, PSINDER, PROM_OCUP y P3YM_HLI**

```
CENSO_2020 =  pd.read_csv(Entradas + 'MANZANAS.csv', encoding = 'Latin-1')
CENSO_2020 = CENSO_2020 [['ENTIDAD','NOM_ENT','MUN',
                          'NOM_MUN','LOC','NOM_LOC',
                          'AGEB','MZA','POBTOT',
                          'P_15A17_M','P_18A24_M','PSINDER',
                          'PROM_OCUP', 'P3YM_HLI']]

CENSO_2020.head(2)
```

De la base de datos filtramos eliminando las filas que contienen los totales

```
Values = ['Total de la entidad','Total del municipio',
          'Total de la localidad urbana', 'Total AGEB urbana']
CENSO_2020_USE = CENSO_2020.query("NOM_LOC != @Values")

CENSO_2020_USE.head(2)
```

Dentro de la base de datos existe la presencia de filas donde los valores son simbolos o caracteres especiales(*, N/D, 99999, etc), por lo que es necesario remplazarlos por valores NAN.

```
CENSO_2020_USE = CENSO_2020_USE.replace({'999999999': np.nan,
                                         '99999999': np.nan,
                                         '*': np.nan,
                                         'N/D': np.nan})

DIM_NUM = CENSO_2020_USE.iloc[: , -6:].columns.tolist()
DIM_TEXT = CENSO_2020_USE.iloc[:, :8].columns.tolist()

CENSO_2020_USE[DIM_NUM] = CENSO_2020_USE[DIM_NUM].astype('float')
CENSO_2020_USE[DIM_TEXT] = CENSO_2020_USE[DIM_TEXT].astype(str)
```

De igual manera la información referida a a las columnas de “Entidad, Municipio, Localidad, AGEB y Manzana”, no presetan el formato necesario para crear la

columna CVEGEO, por lo que debemos completar la información de la forma correcta.

Creada la columna CVEGEO, calculamos la dimensión población masculina de 15 a 24 años como la suma de "P_15A17_M y P_18A24_M". Una vez que hemos creado la dimensión se hacen poco necesarias "P_15A17_M y P_18A24_M, por lo que se eliminan.

```
## Corrección de información
CENSO_2020_USE['ENTIDAD'] = CENSO_2020_USE['ENTIDAD'].str.zfill(2)
CENSO_2020_USE['MUN'] = CENSO_2020_USE['MUN'].str.zfill(3)
CENSO_2020_USE['LOC'] = CENSO_2020_USE['LOC'].str.zfill(4)
CENSO_2020_USE['AGEB'] = CENSO_2020_USE['AGEB'].str.zfill(4)
CENSO_2020_USE['MZA'] = CENSO_2020_USE['MZA'].str.zfill(3)

CENSO_2020_USE['CVEGEO'] = CENSO_2020_USE[['ENTIDAD', 'MUN',
                                           'LOC','AGEB','MZA']].agg(''.join, axis=1)

## Cálculo de Población masculina de 15 a 24 años
CENSO_2020_USE['P_15A24_M'] = CENSO_2020_USE[["P_15A17_M",
                                              "P_18A24_M"]].sum(axis=1,
                                                                min_count=1)

## Eliminación de dimensiones
CENSO_2020_USE = CENSO_2020_USE.drop(['P_15A17_M','P_18A24_M'], axis = 1)

CENSO_2020_USE.head(2)
```

Se carga la base de datos geoespacial que corresponde a la unidad geografica de "manzanas". Para este caso, el archivo se encuentra en formato "json". Del archivo, unicamente consideramos las columnas "CVEGEO y geometry"

```
### Se carga el archivo espacial de Manzanas

MANZA_CDMX = gpd.read_file(Entradas +'MANZA_CDMX.json')
MANZA_CDMX = MANZA_CDMX[['CVEGEO','geometry']]
MANZA_CDMX.head(2)
```

Union (merge) de las unidades geoespaciales con la información del Censo de Población y vivienda 2020. En este punto a cada unidad geografica le asignamos la información del censo de población.

```
### Union a la izquierda con campo llave primaria "CVEGEO"
MERGE = MANZA_CDMX.merge( CENSO_2020_USE,
                          left_on = 'CVEGEO',
                          right_on = 'CVEGEO',
                          how = 'inner')
MERGE.head(2)
```

Siempre es importante saber en que sistema de proyección se encuentran nuestros datos, para eso usamos "crs"

```
MERGE.crs
```

Hacemos un mapa por que nos gustan los mapitas.

```
fig, ax = plt.subplots(figsize=(8, 8))
ax = MERGE.plot(ax = ax, column='P_15A24_M',
                legend=False,
                alpha=0.8,
                scheme='NaturalBreaks',
                cmap='copper',
                classification_kwds={'k':6})
ax.set(title='Población Masculina 15 a 24 años, Ciudad de México')

ax.set_axis_off()

plt.show()
```

Hasta el punto anterior tenemos la información contenida dentro de las manzanas, el paso que sigue es llevar las manzanas a colonias. Para esto es necesario entender que ambos elementos son poligonales y que los centroides de manzanas no necesariamente refieren a la solución contenida dentro de un poligono mayor.

Por eso es necesario usar el criterio de maxima área de la sobreposición de poligonos. Al hablar de área el sistema de proyección debe estar en metros, por lo que si no lo esta se debe cambiar. Para este caso se cambio a EPSG:6362

Se cargan las colonias y se valida que ambos crs se encuentren en metros "6362 o 6362" , en caso contrario es necesrio llevar a cabo una reproyección.

```
COLONIAS_CDMX = gpd.read_file(Entradas +'COLONIAS.json')
print("Colonias CRS", COLONIAS_CDMX.crs)
print("Manzanas CRS", MERGE.crs)
```

Los archivos estan en coordenadas geograficas, por lo que se reproyecta

```
MANZANA_METROS = MERGE.to_crs(6362)
COLONIAS_METROS = COLONIAS_CDMX.to_crs(6362)

print("Crs Manzanas", MANZANA_METROS.crs )
print("Crs Colonias", COLONIAS_METROS.crs )
```

Buscamos en este punto identificar la intersección entre colonias y manzanas para asignar a cada manzana (base al criterio de área maxima) la clave de la colonia a la que pertence.

```
INTERSECCION = gpd.overlay(COLONIAS_METROS,
                           MANZANA_METROS,
                            how = 'intersection')
```

Se calcula el valor de área para cada poligono intersectado

```
## Se calcula el area
INTERSECCION['area'] = INTERSECCION.geometry.area
```

Para el overlay se reordena la información del área de manera descendente y se eliminan los duplicados con base a la "CVEGEO" manteniendo unicamente el primer valor

```
INTERSECCION = (INTERSECCION.sort_values('area', ascending = False).
                drop_duplicates(subset="CVEGEO", keep = 'first').
                drop(['geometry','area'], axis = 1))

### Se eliminan columnas no necesarias
INTERSECCION_USE = INTERSECCION.drop(['ENT', 'CVEDT', 'NOMDT', 'DTTOLOC'], axis = 1)
```

En la base de colonias se identificaron caracteres especiales, por lo que se procede a remplazarlos, por su valor correspondiente.

```
Dic_Ca = {'Ã‘': 'Ñ'}
INTERSECCION_USE.replace(Dic_Ca, inplace=True, regex=True)
INTERSECCION_USE.columns = INTERSECCION_USE.columns.to_series().replace(Dic_Ca, regex=True

INTERSECCION_USE.shape
```

Se une la información de overly de las Manzanas ya alineadas con colonias en la geometry de las manzanas para tener la base final. En la base final podemos observar la información a nivel: manzana, colonia y alcaldia, donde esta ultima se extraer en razon directa de la informacion contenida en la base de manzanas.

```
DATA_FINAL_USE = MANZA_CDMX.merge(INTERSECCION_USE,
                            left_on = 'CVEGEO',
                            right_on = 'CVEGEO',
                            how = 'inner').rename({"CVEUT": "CVE_COL", "NOMUT": "NOM_COL", "
DATA_FINAL_USE.shape
```

Se valida que cada manzana este asociada a cada una de las colonias
Recorndado que la relación es una colonia a muchas manzanzanas
“ ” Por lo que no deben existir manzanas repetididas”

$$M_{n-1} = f(C)$$
$$C \leftarrow M_{n-1}$$

Se valida que no existan manzanas repetidas

```
DATA_FINAL_USE.CVEGEO.value_counts()
```

La relacion de muchas manzanas a una colonia, se valida para cada clave de colonias se repite tantas veces existan manzanas

```
DATA_FINAL_USE.CVE_COL.value_counts()
```

Aqui validamos como las claves de las manzanas son diferentes para una misma colonia y se entiende la relacion, muchas manzanas a una colonia

```
DATA_FINAL_USE.query('CVE_COL == "07-320"').head(3)
```

Reordenamos la información de la forma “Regional a local” es decir:

$$Alcaldia \rightarrow Colonia \rightarrow Manzana$$

1. **Alcaldia**: *ENTIDAD, NOM_ENT, MUN, NOM_MUN, LOC, NOM_LOC,*

2. **Colonia**: *ID_COL, CVE_COL, NOM_COL,*
3. **Manzana**: *AGEB, MZA, CVEGEO,*

4. **Dimensiones**: *POBTOT, PSINDER, PROM_OCUP, P3YM_HLI, P_15A24_M*
5. **Geometry**

Reordenamiento de la información

```
DATA_FIN_USE = DATA_FINAL_USE[['ENTIDAD','NOM_ENT', 'MUN', 'NOM_MUN', 'LOC', 'NOM_LOC',
                               'ID_COL','CVE_COL', 'NOM_COL', 'AGEB', 'MZA',
                               'POBTOT', 'PSINDER', 'PROM_OCUP', 'P3YM_HLI', 'P_15A24_M',
                               'geometry']]

DATA_FIN_USE.head(3)
```

Hacemos un mapita porque nos gustan los mapitas

```
fig, ax = plt.subplots(figsize=(8, 8))
ax = DATA_FIN_USE.plot(ax = ax, column='P_15A24_M',
                       legend= False,
                       alpha=0.8,
                       scheme='NaturalBreaks',
                       cmap='copper',
                       classification_kwds={'k':6})
ax.set(title='Población Masculina 15 a 24 años, Ciudad de México')

ax.set_axis_off()

plt.show()
```

4.2 Análisis de Componentes Principales (PCA)

En esta sección se cálcula el indice de ***Concetración de desventajas*** mediante la reducción de las dimensiónes por componentes principales (PCA). Esto se hace a nivel **Alcaldias y Delegaciones**. La información a nivel alcaldia y delegacion es un proceso de reagrupacion y nuevos calculos de los valores.

4.2.1 NIVEL COLONIAS

Para este punto agrupamos los datos por nivel colonia para extraer el valor del índice por “PCA”

```
COLONIA_PCA = pd.DataFrame(DATA_FIN_USE.groupby(['CVE_COL']).agg({'POBTOT': 'sum',
                                                                  'PSINDER':'sum',
                                                                  'PROM_OCUP': 'mean',
                                                                  'P3YM_HLI':'sum',
                                                                  'P_15A24_M': 'sum'}).res
COLONIA_PCA.head(2)
```

Impotante

Para que componentes principales tenga un alto rendimiento la información sdebe estar normalizada por **Z-SCORE (StandardScaler)**

$$Z = \frac{x - \mu}{\sigma}$$

Para calcular la componente principal, separamos nuestra base de datos con el fin de tener las dimensiones que contruyen el índice.

```
### Hacemos un copia por si necesitamos un proceso con la base original
PCA_COLONIAS = COLONIA_PCA.copy()

### Seleccion de las dimensiones con las que se calcula el indice de desventajas "PSINDER;

PCA_X = PCA_COLONIAS.drop(['CVE_COL','POBTOT'], axis = 1)
PCA_y = PCA_COLONIAS[['CVE_COL']]
```

Se normaliza la informacion por Z-Score, determinamos el número de componentes y aplicamos la función para calcular

```
### Se normaliza la informacion por Z-Score

S_TRANSF = StandardScaler()
PCA_X_SCALER = pd.DataFrame(S_TRANSF.fit_transform(PCA_X),
                            columns = PCA_X.columns)

### Se determina el número de componentes
PCA_N = PCA(n_components = 1)
PCA_COMPONENTE = PCA_N.fit_transform(PCA_X_SCALER)
```

Se calcula la varianza total por respecto al numero de componentes

```
### Se calcula la varianza total por respecto al numero de componentes
VARIANZA_TOTA = PCA_N.explained_variance_ratio_.sum() * 100
print("\n Total de la variancia explicada \n", round(VARIANZA_TOTA,3), "%")
```

Con una componente (PC1) se explica 66.82 % de la varianza total, lo cual implica que mas de la mitad de la información e los datos puede encapsularse en ese componente principal.

finalmente se indexan los resultados a la base de datos como una nueva columna con clave DIS_COL = concentrated disadvantage component

```
### Pegamos los valores de PCA en la base de datos
PCA_COLONIAS['DIS_COL'] = PCA_COMPONENTE
PCA_COLONIAS.head(5)
```

Impotante

Lo anterior lo transformamos a una funcion para optimizar el proceso de trabajo. Funcion que podemos llamar despues

```
def CONC_DIS (TABLA, DIM_CLAVE, DIM_POBLA):

    PCA_TABLA = TABLA.copy()

    ### Seleccion de las dimensiones con las que se calcula el indice de desventajas "PSIN

    PCA_X = PCA_TABLA.drop([DIM_CLAVE, DIM_POBLA], axis = 1)
    PCA_y = PCA_TABLA[[DIM_CLAVE]]

    ### Se normaliza la informacion por Z-Score

    S_TRANSF = StandardScaler()
    PCA_X_SCALER = pd.DataFrame( S_TRANSF.fit_transform(PCA_X),
                                 columns = PCA_X.columns)

    ### Se determina el número de componentes

    PCA_N = PCA(n_components = 1)
    PCA_COMPONENTE = PCA_N.fit_transform(PCA_X_SCALER)

    ### Se calcula la varianza total por respecto al numero de componentes
```

```
    VARIANZA_TOTA = PCA_N.explained_variance_ratio_.sum() * 100

    print("\n Total de la variancia explicada \n", round(VARIANZA_TOTA,3), "%")
    ### Se indexan los resultados a la base de datos como una nueva columna con clave DIS_

    PCA_TABLA['DISAD'] = PCA_COMPONENTE

    PCA_TABLA = PCA_TABLA[[DIM_CLAVE, 'DISAD' ]]

    return (PCA_TABLA)
```

Podemos usar la función creada y aplicarla en los datos para revalidar los resultados.

```
### Revalidación de información

DESVE_COL = CONC_DIS (COLONIA_PCA, 'CVE_COL','POBTOT')
DESVE_COL.head(2)
```

4.2.2 NIVEL ALCALIDAS

El Agrupamiento de datos por nivel Alcaldia para "PCA". Recordando que la clave de Alcaldia == Municipio la podemos observar de la forma: DATA_FIN_USE.NOM_MUN.value_counts()

```
ALCALDIA_PCA = pd.DataFrame(DATA_FIN_USE.groupby(['MUN']).agg({'POBTOT': 'sum',
                                                               'PSINDER':'sum',
                                                               'PROM_OCUP': 'mean',
                                                               'P3YM_HLI':'sum',
                                                               'P_15A24_M': 'sum'}).res
ALCALDIA_PCA.head(2)
```

Aplicamos la funcion creada con anterioridad

```
### Aplicando la función creada arriba

DESVE_ALCA = CONC_DIS (ALCALDIA_PCA, 'MUN','POBTOT')
DESVE_ALCA.head(2)
```

En este punto, podemos unir toda la información a la tabla original y renombramos las columnas de desventajas en cada nivel

```
MERGE_DESVENTAJAS = DATA_FIN_USE.merge(DESVE_COL,
                                       left_on = 'CVE_COL',
                                       right_on = 'CVE_COL',
                                       how = 'inner').merge(DESVE_ALCA,
                                                            left_on ='MUN',
                                                            right_on = 'MUN' ,
                                                            how = 'inner').rename({"DISAD_

MERGE_DESVENTAJAS.head(2)
```

Hacemos un mapita nuevamente

```
fig, ax = plt.subplots(figsize=(8, 8))
ax = MERGE_DESVENTAJAS.plot(ax = ax, column='DIS_COL',
                            legend= True,
                            alpha=0.8,
                            scheme='NaturalBreaks',
                            cmap='copper',
                            classification_kwds={'k':6})
ax.set(title='Desventajas a nivel Colonia, Ciudad de México')

ax.set_axis_off()

plt.show()
```

```
fig, ax = plt.subplots(figsize=(8, 8))
ax = MERGE_DESVENTAJAS.plot(ax = ax, column='DIS_MUN',
                            legend= True,
                            alpha=0.8,
                            scheme='NaturalBreaks',
                            cmap='copper',
                            classification_kwds={'k':6})
ax.set(title='Desventajas a nivel Alcaldias, Ciudad de México')

ax.set_axis_off()

plt.show()
```

5 Methods

6 Summary

In summary, this book has no content whatsoever.

References