

✓ Import Required Libraries

Import libraries such as pandas, matplotlib, and seaborn for data manipulation and visualization.

```
# Instalar librerías
!pip install -q -U Dash

# Import necessary libraries for data manipulation and visualization
import pandas as pd # For data manipulation
import matplotlib.pyplot as plt # For creating plots
import seaborn as sns # For advanced visualization styles

import ipywidgets as widgets
from IPython.display import display

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

from wordcloud import WordCloud
import json
import geopandas as gpd

import numpy as np
import folium

#Librerías para DashBoard
from dash import Dash, dcc, html, Input, Output, State
from dash import Dash, html, dash_table, dcc, callback, Output, Input
from dash.dependencies import Input, Output

import io
import base64

# Configure default styles for seaborn
sns.set(style="whitegrid") # Set a white grid background for plots
```

✓ Load Data

Load the dataset 'df_all_years_filtered.csv' into a pandas

```
# Load the dataset into a pandas DataFrame
l_path='/content/'
l_file='df_all_years_filtered.csv'
df = pd.read_csv(l_path + l_file, index_col=0)

# --- Load GeoJSON ---
with open("/content/comunas.geojson", "r", encoding='utf-8') as f:
    geojson = json.load(f)
```

✓ Create additional variables

```

# Diccionario de reemplazo
rama_educacional_dict = {
    "H1": "Humanista científico Diurno",
    "H2": "Humanista científico Nocturno",
    "H3": "Humanista científico - validación de estudios",
    "H4": "Humanista científico - Reconocimiento de estudios",
    "T1": "Técnico profesional comercial",
    "T2": "Técnico profesional Industrial",
    "T3": "Técnico profesional Servicios y técnica",
    "T4": "Técnico profesional agrícola",
    "T5": "Técnico profesional Marítima"
}

# Aplicar el reemplazo en el atributo 'rama_educacional'
df["COD_RAMA_EDUCACIONAL"] = df["RAMA_EDUCACIONAL"]
df["RAMA_EDUCACIONAL"] = df["RAMA_EDUCACIONAL"].replace(rama_educacional_dict)

# Diccionario de reemplazo para GRUPO_DEPENDENCIA
grupo_dependencia_dict = {
    1: "Particular pagado",
    2: "Particular subvencionado",
    3: "Municipal",
    4: "Servicio Local de Educación"
}

# Aplicar el reemplazo en el atributo 'GRUPO_DEPENDENCIA'
df["COD_GRUPO_DEPENDENCIA"] = df["GRUPO_DEPENDENCIA"]
df["GRUPO_DEPENDENCIA"] = df["GRUPO_DEPENDENCIA"].replace(grupo_dependencia_dict)

# Diccionario de reemplazo para CODIGO_REGION
codigo_region_dict = {
    1: "REGION DE TARAPACA",
    2: "REGION DE ANTOFAGASTA",
    3: "REGION DE ATACAMA",
    4: "REGION DE COQUIMBO",
    5: "REGION DE VALPARAISO",
    6: "REGION DEL LIBERTADOR GENERAL BERNARDO O'HIGGINS",
    7: "REGION DEL MAULE",
    8: "REGION DEL BIOBIO",
    9: "REGION DE LA ARAUCANIA",
    10: "REGION DE LOS LAGOS",
    11: "REGION AISEN DEL GENERAL CARLOS IBAÑEZ DEL CAMPO",
    12: "REGION DE MAGALLANES Y DE LA ANTARTICA CHILENA",
    13: "REGION METROPOLITANA DE SANTIAGO",
    14: "REGION DE LOS RIOS",
    15: "REGION DE ARICA Y PARINACOTA",
    16: "REGION DE ÑUBLE"
}


# Aplicar el reemplazo en el atributo 'CODIGO_REGION'
df["codregion"] = df["CODIGO_REGION"]
df["CODIGO_REGION"] = df["CODIGO_REGION"].replace(codigo_region_dict)

# Convertir la columna CODIGO_COMUNA a tipo numérico, forzando NaN para valores no válidos
df["CODIGO_COMUNA"] = pd.to_numeric(df["CODIGO_COMUNA"], errors='coerce')

# Eliminar filas con valores NaN en la columna CODIGO_COMUNA
df = df.dropna(subset=["CODIGO_COMUNA"])

# Convertir la columna CODIGO_COMUNA a tipo int
df["CODIGO_COMUNA"] = df["CODIGO_COMUNA"].astype(int)

```

 <ipython-input-8-29eb50d4efa4>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
df["CODIGO_COMUNA"] = df["CODIGO_COMUNA"].astype(int)

Diccionario de reemplazo para CODIGO_COMUNA

```
codigo_comuna_dict = {
    1101: "IQUIQUE",
    1107: "ALTO HOSPICIO",
    1401: "POZO ALMONTE",
    1402: "CAMIÑA",
    1403: "COLCHANE",
    1404: "HUARA",
    1405: "PICA",
    2101: "ANTOFAGASTA",
    2102: "MEJILLONES",
    2103: "SIERRA GORDA",
    2104: "TALTAL",
    2201: "CALAMA",
    2202: "OLLAGUE",
    2203: "SAN PEDRO DE ATACAMA",
    2301: "TOCOPILLA",
    2302: "MARIA ELENA",
    3101: "COPIAPO",
    3102: "CALDERA",
    3103: "TIERRA AMARILLA",
    3201: "CHAÑARAL",
    3202: "DIEGO DE ALMAGRO",
    3301: "VALLENAR",
    3302: "ALTO DEL CARMEN",
    3303: "FREIRINA",
    3304: "HUASCO",
    4101: "LA SERENA",
    4102: "COQUIMBO",
    4103: "ANDACOLLO",
    4104: "LA HIGUERA",
    4105: "PAIGUANO",
    4106: "VICUÑA",
    4201: "ILLAPEL",
    4202: "CANELA",
    4203: "LOS VILOS",
    4204: "SALAMANCA",
    4301: "OVALLE",
    4302: "COMBARBALA",
    4303: "MONTE PATRIA",
    4304: "PUNITAQUI",
    4305: "RIO HURTADO",
    5101: "VALPARAISO",
    5102: "CASABLANCA",
    5103: "CON CON",
    5104: "JUAN FERNANDEZ",
    5105: "PUCHUNCAVI",
    5107: "QUINTERO",
    5109: "VIÑA DEL MAR",
    5201: "ISLA DE PASCUA",
    5301: "LOS ANDES",
    5302: "CALLE LARGA",
    5303: "RINCONADA",
    5304: "SAN ESTEBAN",
```

5401: "LA LIGUA",
5402: "CABILDO",
5403: "PAPUDO",
5404: "PETORCA",
5405: "ZAPALLAR",
5501: "QUILLOTA",
5502: "CALERA",
5503: "HIJUELAS",
5504: "LA CRUZ",
5506: "NOGALES",
5601: "SAN ANTONIO",
5602: "ALGARROBO",
5603: "CARTAGENA",
5604: "EL QUISCO",
5605: "EL TABO",
5606: "SANTO DOMINGO",
5701: "SAN FELIPE",
5702: "CATEMU",
5703: "LLAILLAY",
5704: "PANQUEHUE",
5705: "PUTAENDO",
5706: "SANTA MARIA",
5801: "QUILPUE",
5802: "LIMACHE",
5803: "OLMUE",
5804: "VILLA ALEMANA",
6101: "RANCAGUA",
6102: "CODEGUA",
6103: "COINCO",
6104: "COLTAUCO",
6105: "DOÑIHUE",
6106: "GRANEROS",
6107: "LAS CABRAS",
6108: "MACHALI",
6109: "MALLOA",
6110: "MOSTAZAL",
6111: "OLIVAR",
6112: "PEUMO",
6113: "PICHIDEGUA",
6114: "QUINTA DE TILCOCO",
6115: "RENGO",
6116: "REQUINOA",
6117: "SAN VICENTE",
6201: "PICHILEMU",
6202: "LA ESTRELLA",
6203: "LITUECHE",
6204: "MARCIHUE",
6205: "NAVIDAD",
6206: "PAREDONES",
6301: "SAN FERNANDO",
6302: "CHEPICA",
6303: "CHIMBARONGO",
6304: "LOLOL",
6305: "NANCAGUA",
6306: "PALMILLA",
6307: "PERALILLO",
6308: "PLACILLA",
6309: "PUMANQUE",
6310: "SANTA CRUZ",
7101: "TALCA",
7102: "CONSTITUCION",
7103: "CUREPTO",
7104: "EMPEDRADO",
7105: "MAULE",
7106: "PELARCO",

7107: "PENCAHUE",
7108: "RIO CLARO",
7109: "SAN CLEMENTE",
7110: "SAN RAFAEL",
7201: "CAUQUENES",
7202: "CHANCO",
7203: "PELLUHUE",
7301: "CURICO",
7302: "HUALAÑE",
7303: "LICANTEN",
7304: "MOLINA",
7305: "RAUCO",
7306: "ROMERAL",
7307: "SAGRADA FAMILIA",
7308: "TENÓ",
7309: "VICHUQUEN",
7401: "LINARES",
7402: "COLBUN",
7403: "LONGAVI",
7404: "PARRAL",
7405: "RETIRO",
7406: "SAN JAVIER",
7407: "VILLA ALEGRE",
7408: "YERBAS BUENAS",
8101: "CONCEPCION",
8102: "CORONEL",
8103: "CHIGUAYANTE",
8104: "FLORIDA",
8105: "HUALQUI",
8106: "LOTA",
8107: "PENCO",
8108: "SAN PEDRO DE LA PAZ",
8109: "SANTA JUANA",
8110: "TALCAHUANO",
8111: "TOME",
8112: "HUALPEN",
8201: "LEBU",
8202: "ARAUCO",
8203: "CAÑETE",
8204: "CONTULMO",
8205: "CURANILAHUE",
8206: "LOS ALAMOS",
8207: "TIRUA",
8301: "LOS ANGELES",
8302: "ANTUCO",
8303: "CABRERO",
8304: "LAJA",
8305: "MULCHEN",
8306: "NACIMIENTO",
8307: "NEGRETE",
8308: "QUILACO",
8309: "QUILLECO",
8310: "SAN ROSENDO",
8311: "SANTA BARBARA",
8312: "TUCAPEL",
8313: "YUMBEL",
8314: "ALTO BIO-BIO",
9101: "TEMUCO",
9102: "CARAHUE",
9103: "CUNCO",
9104: "CURARREHUE",
9105: "FREIRE",
9106: "GALVARINO",
9107: "GORBEA",
9108: "LAUTARO",

9109: "LONCOCHE",
9110: "MELIPEUCO",
9111: "NUEVA IMPERIAL",
9112: "PADRE LAS CASAS",
9113: "PERQUENCO",
9114: "PITRUFQUEN",
9115: "PUCON",
9116: "SAAVEDRA",
9117: "TEODORO SCHMIDT",
9118: "TOLTEN",
9119: "VILCUN",
9120: "VILLARRICA",
9121: "CHOLCHOL",
9201: "ANGOL",
9202: "COLLIPULLI",
9203: "CURACAUTIN",
9204: "ERCILLA",
9205: "LONQUIMAY",
9206: "LOS SAUCES",
9207: "LUMACO",
9208: "PUREN",
9209: "RENAICO",
9210: "TRAIGUEN",
9211: "VICTORIA",
10101: "PUERTO MONTT",
10102: "CALBUCO",
10103: "COCHAMO",
10104: "FRESIA",
10105: "FRUTILLAR",
10106: "LOS MUERMOS",
10107: "LLANQUIHUE",
10108: "MAULLIN",
10109: "PUERTO VARAS",
10201: "CASTRO",
10202: "ANCUD",
10203: "CHONCHI",
10204: "CURACO DE VELEZ",
10205: "DALCAHUE",
10206: "PUQUELDON",
10207: "QUEILEN",
10208: "QUELLON",
10209: "QUEMCHI",
10210: "QUINCHAO",
10301: "OSORNO",
10302: "PUERTO OCTAY",
10303: "PURRANQUE",
10304: "PUYEHUE",
10305: "RIO NEGRO",
10306: "SAN JUAN DE LA COSTA",
10307: "SAN PABLO",
10401: "CHAITEN",
10402: "FUTALEUFU",
10403: "HUALAIHUE",
10404: "PALENA",
11101: "COIHAIQUE",
11102: "LAGO VERDE",
11201: "AISEN",
11202: "CISNES",
11203: "GUAITECAS",
11301: "COCHRANE",
11302: "O'HIGGINS",
11303: "TORTEL",
11401: "CHILE CHICO",
11402: "RIO IBAÑEZ",
12101: "PUNTA ARENAS",

12102: "LAGUNA BLANCA",
12103: "RIO VERDE",
12104: "SAN GREGORIO",
12201: "CABO DE HORNO (EX NAVARINO)",
12202: "ANTARTICA",
12301: "PORVENIR",
12302: "PRIMAVERA",
12303: "TIMAUKELE",
12401: "NATALES",
12402: "TORRES DEL PAINE",
13101: "SANTIAGO",
13102: "CERRILLOS",
13103: "CERRO NAVIA",
13104: "CONCHALI",
13105: "EL BOSQUE",
13106: "ESTACION CENTRAL",
13107: "HUECHURABA",
13108: "INDEPENDENCIA",
13109: "LA CISTERNA",
13110: "LA FLORIDA",
13111: "LA GRANJA",
13112: "LA PINTANA",
13113: "LA REINA",
13114: "LAS CONDES",
13115: "LO BARNECHEA",
13116: "LO ESPEJO",
13117: "LO PRADO",
13118: "MACUL",
13119: "MAIPU",
13120: "ÑUÑO",
13121: "PEDRO AGUIRRE CERDA",
13122: "PEÑALOEN",
13123: "PROVIDENCIA",
13124: "PUDAHUEL",
13125: "QUILICURA",
13126: "QUINTA NORMAL",
13127: "RECOLETA",
13128: "RENCA",
13129: "SAN JOAQUIN",
13130: "SAN MIGUEL",
13131: "SAN RAMON",
13132: "VITACURA",
13201: "PUENTE ALTO",
13202: "PIRQUE",
13203: "SAN JOSE DE MAIPO",
13301: "COLINA",
13302: "LAMP",
13303: "TILIT",
13401: "SAN BERNARDO",
13402: "BUIN",
13403: "CALERA DE TANGO",
13404: "PAINE",
13501: "MELIPILLA",
13502: "ALHUE",
13503: "CURACAVI",
13504: "MARIA PINTO",
13505: "SAN PEDRO",
13601: "TALAGANTE",
13602: "EL MONTE",
13603: "ISLA DE MAIPO",
13604: "PADRE HURTADO",
13605: "PEÑAFLO",
14101: "VALDIVIA",
14102: "CORRAL",
14103: "LANCO",

```

14104: "LOS LAGOS",
14105: "MAFIL",
14106: "MARIQUINA",
14107: "PAILLACO",
14108: "PANGUIPULLI",
14201: "LA UNION",
14202: "FUTRONO",
14203: "LAGO RANCO",
14204: "RIO BUENO",
15101: "ARICA",
15102: "CAMARONES",
15201: "PUTRE",
15202: "GENERAL LAGOS",
16101: "CHILLAN",
16102: "BULNES",
16103: "CHILLAN VIEJO",
16104: "EL CARMEN",
16105: "PEMUCO",
16106: "PINTO",
16107: "QUILLON",
16108: "SAN IGNACIO",
16109: "YUNGAY",
16201: "QUIRIHUE",
16202: "COBQUECURA",
16203: "COELEMU",
16204: "NINHUE",
16205: "PORTEZUELO",
16206: "RANQUIL",
16207: "TREGUACO",
16301: "SAN CARLOS",
16302: "COIHUECO",
16303: "ÑIQUEN",
16304: "SAN FABIAN",
16305: "SAN NICOLAS"
}

# Aplicar el reemplazo en el atributo 'CODIGO_COMUNA'
df["cod_comuna"] = df["CODIGO_COMUNA"]
df["CODIGO_COMUNA"] = df["CODIGO_COMUNA"].replace(codigo_comuna_dict)

```

✓ Dashboard

✓ Variables for Graphics and Auxiliary functions

```

# Definir los bins para los histogramas
bins = [0, 200, 400, 600, 800, 1000]
bin_labels = ['0-200', '200-400', '400-600', '600-800', '800-1000']

# Función para crear los bins y filtrar los datos
def prepare_data(df, selected_regions=None, selected_comunas=None, selected_years=None):
    # Hacer una copia para no modificar el original
    filtered_df = df.copy()

    # Aplicar filtros
    if selected_regions:
        filtered_df = filtered_df[filtered_df['CODIGO_REGION'].isin(selected_regions)]
    if selected_comunas:
        filtered_df = filtered_df[filtered_df['CODIGO_COMUNA'].isin(selected_comunas)]
    if selected_years:
        filtered_df = filtered_df[filtered_df['YEAR'].isin(selected_years)]

```



```

# Crear columnas de bins
filtered_df['CLEC_REG_BIN'] = pd.cut(filtered_df['CLEC_REG_ACTUAL'], bins=bins, labels=bin_labels, right=False)
filtered_df['MATE1_REG_BIN'] = pd.cut(filtered_df['MATE1_REG_ACTUAL'], bins=bins, labels=bin_labels, right=False)
filtered_df['CLEC_INV_BIN'] = pd.cut(filtered_df['CLEC_INV_ACTUAL'], bins=bins, labels=bin_labels, right=False)
filtered_df['MATE1_INV_BIN'] = pd.cut(filtered_df['MATE1_INV_ACTUAL'], bins=bins, labels=bin_labels, right=False)

# Eliminar filas con NaN en las columnas de bins
filtered_df = filtered_df.dropna(subset=['CLEC_REG_BIN', 'MATE1_REG_BIN', 'CLEC_INV_BIN', 'MATE1_INV_BIN'], how='all')

return filtered_df

# --- Binning ---
def bin_score_label(score):
    if score <= 100: return 'Cien'
    elif score <= 200: return 'Doscientos'
    elif score <= 300: return 'Trescientos'
    elif score <= 400: return 'Cuatrocientos'
    elif score <= 500: return 'Quinientos'
    elif score <= 600: return 'Seiscientos'
    elif score <= 700: return 'Setecientos'
    elif score <= 800: return 'Ochocientos'
    elif score <= 900: return 'Novecientos'
    elif score <= 1000: return 'Mil'
    else: return 'Error'

# Obtener opciones únicas para los filtros
l_valores_filtros_regiones = df['CODIGO_REGION'].sort_values().unique()
l_valores_filtros_comunas = df['CODIGO_COMUNA'].sort_values().unique()
l_valores_filtros_years = df['YEAR'].sort_values().unique()

# Valores Default de Filtros
l_default_region = "REGION DEL BIOBIO"
l_default_comuna = ''
l_default_year = 2022

# Filter to BioBío region (CODIGO_REGION = 8)
df_biobio = df[df['codregion'] == 8].copy()

# Remove zeros and NaNs
df_biobio = df_biobio[df_biobio['CLEC_REG_ACTUAL'].notna() & (df_biobio['CLEC_REG_ACTUAL'] > 0)]
df_biobio = df_biobio[df_biobio['MATE1_REG_ACTUAL'].notna() & (df_biobio['MATE1_REG_ACTUAL'] > 0)]

# --- Calculate averages ---
avg_scores = df_biobio.groupby(['cod_comuna', 'CODIGO_COMUNA']).agg({
    'CLEC_REG_ACTUAL': 'mean',
    'MATE1_REG_ACTUAL': 'mean'
}).reset_index()

# Rename columns and format
avg_scores.rename(columns={
    'cod_comuna': 'cod_comuna',
    'CODIGO_COMUNA': 'Comuna',
    'CLEC_REG_ACTUAL': 'Prom_CLEC',
    'MATE1_REG_ACTUAL': 'Prom_MATE1'
}, inplace=True)

# Convert to int for GeoJSON match
avg_scores['cod_comuna'] = avg_scores['cod_comuna'].astype(int)

```

✓ Layout and Callback

```
# Aplicación Dash
app = Dash(__name__)

#####
# Sección Diseño del dashboard (Layout)
#####

app.layout = html.Div(style={'backgroundColor': 'black'},
    children=[
        # Sección de Título, Descripción y Detalles
        html.H1("Rendición de PAES 2022 a 2025", style={'textAlign': 'center', 'color': 'white'}),
        html.Div([
            # Título Principal
            html.Div([
                html.H1("Comparación de período regular e invierno para las pruebas base de Comprensión de Lectura (CLEC) y
                    style={'textAlign': 'center', 'color': 'white', 'fontSize': '28px', 'marginBottom': '20px'})
            ], style={'width': '100%'}),

            # Descripción y Detalles en dos columnas
            html.Div([
                # Columna izquierda - Descripción
                html.Div([
                    html.H3("Descripción Dashboard", style={'color': 'white', 'marginBottom': '10px'}),
                    html.P("Este dashboard muestra las diferencias entre la rendición de la PAES entre 2022 y 2025 en ambos
                        \"Buscamos responder la hipótesis de si las rendiciones en el período de invierno tienen mayor pun
                        style={'color': 'white', 'textAlign': 'justify'})
                ], style={'width': '48%', 'display': 'inline-block', 'verticalAlign': 'top', 'padding': '10px'}),

                # Columna derecha - Detalles
                html.Div([
                    html.H3("Detalle de visualizaciones", style={'color': 'white', 'marginBottom': '10px'}),
                    html.Ul([
                        html.Li("Evolución de los puntajes promedio de ambas pruebas.", style={'color': 'white'}),
                        html.Li("La rama educacional y grupo de dependencia de origen.", style={'color': 'white'}),
                        html.Li("Distribuciones de puntajes promedio por región y comuna (bins de 200 puntos).", style={'color': 'white'}),
                        html.Li("Nube de palabras de puntajes promedio (bins de 100 puntos).", style={'color': 'white'}),
                        html.Li("Histograma de puntajes promedio.", style={'color': 'white'}),
                        html.Li("Distribución de puntajes de CLEC versus MATE1.", style={'color': 'white'}),
                        html.Li("Filtros disponibles: Región, Comuna y Año de rendición.", style={'color': 'white'})
                    ], style={'paddingLeft': '20px'})
                ], style={'width': '48%', 'display': 'inline-block', 'float': 'right', 'verticalAlign': 'top', 'padding': '10px'})
            ], style={'marginBottom': '30px'}),
        ]),

        #####
        # Filtros Globales
        html.Div([
            html.Div([
                html.Label('Región', style={'color': 'white'}),
                dcc.Dropdown(
                    id='region-filter',
                    options=[{'label': reg, 'value': reg} for reg in l_valores_filtros_regiones],
                    value=None,
                    multi=True,
                    placeholder='Seleccione región(es)'
                )
            ], style={'width': '30%', 'display': 'inline-block', 'padding': '10px'}),

            html.Div([
                html.Label('Comuna', style={'color': 'white'}),
                dcc.Dropdown(
                    id='comuna-filter',
```

```

options=[{'label': com, 'value': com} for com in l_valores_filtros_comunas],
value=None,
multi=True,
placeholder='Seleccione comuna(s)'
)
], style={'width': '30%', 'display': 'inline-block', 'padding': '10px'}),

html.Div([
    html.Label('Año', style={'color': 'white'}),
    dcc.Dropdown(
        id='year-filter',
        options=[{'label': str(int(year)), 'value': year} for year in l_valores_filtros_years],
        value=None,
        multi=True,
        placeholder='Seleccione año(s)'
    )
], style={'width': '30%', 'display': 'inline-block', 'padding': '10px'})
], style={'backgroundColor': '#333333', 'padding': '10px', 'borderRadius': '5px'}),

#####
# Gráficos

#1. gráficas de series de tiempo
html.Div([
    html.Div([
        dcc.Graph(id='serie-prom-clecymat')
    ], style={'width': '100%', 'display': 'inline-block'})
], style={'marginTop': '20px'}),

#2. gráficas de composición
html.Div([
    html.Div([
        dcc.Graph(id='grafica-pie')
    ], style={'width': '48%', 'display': 'inline-block'}),
    html.Div([
        dcc.Graph(id='grafica-barra')
    ], style={'width': '48%', 'display': 'inline-block', 'float': 'right'})
], style={'marginTop': '20px'}),

#3. gráficas de distribución estadística
html.Div([
    html.Div([
        dcc.Graph(id='clec-reg-hist')
    ], style={'width': '48%', 'display': 'inline-block'}),

    html.Div([
        dcc.Graph(id='clec-inv-hist')
    ], style={'width': '48%', 'display': 'inline-block', 'float': 'right'})
], style={'marginTop': '20px'}),

html.Div([
    html.Div([
        dcc.Graph(id='mate1-reg-hist')
    ], style={'width': '48%', 'display': 'inline-block'}),

    html.Div([
        dcc.Graph(id='mate1-inv-hist')
    ], style={'width': '48%', 'display': 'inline-block', 'float': 'right'})
], style={'marginTop': '20px'}),

#4. nube de palabras clave
html.Div([
    html.Div([
        dcc.Graph(id='nube-reg')
    ], style={'width': '48%', 'display': 'inline-block'}),
    html.Div([

```

```

        dcc.Graph(id='nube-inv')
    ], style={'width': '48%', 'display': 'inline-block', 'float': 'right'})
], style={'marginTop': '20px'}),

#5. mapas con marcadores o coropléticos
html.Div([
    html.Div([
        dcc.Graph(id='map-clec')
    ], style={'width': '100%', 'display': 'inline-block'}),
    html.Div([
        dcc.Graph(id='map-mate1')
    ], style={'width': '100%', 'display': 'inline-block'})
]),

# Filtros locales para el gráfico de comparación
html.Div([
    html.Div([
        html.Label('Tipo de Dependencia 1', style={'color': 'white'}),
        dcc.Dropdown(
            id='tipo1-filter',
            options=[{'label': tipo, 'value': tipo} for tipo in df['GRUPO_DEPENDENCIA'].unique()],
            value=None,
            placeholder='Seleccione primer tipo'
        )
    ], style={'width': '48%', 'display': 'inline-block', 'padding': '10px'}),

    html.Div([
        html.Label('Tipo de Dependencia 2', style={'color': 'white'}),
        dcc.Dropdown(
            id='tipo2-filter',
            options=[{'label': tipo, 'value': tipo} for tipo in df['GRUPO_DEPENDENCIA'].unique()],
            value=None,
            placeholder='Seleccione segundo tipo'
        )
    ], style={'width': '48%', 'display': 'inline-block', 'padding': '10px'})
], style={'backgroundColor': '#333333', 'padding': '10px', 'borderRadius': '5px', 'marginTop': '20px'}),

#6. gráficas comparativas de dos variables en el tiempo + #7. gráficas comparativas de dos variables en el tiempo
html.Div([
    html.Div([
        dcc.Graph(id='graf-puntajes')
    ], style={'width': '100%', 'display': 'inline-block'})
], style={'marginTop': '20px'}),

#8. gráficas de correlación
html.Div([
    html.Div([
        dcc.Graph(id='graf-corr')
    ], style={'width': '100%', 'display': 'inline-block'})
], style={'marginTop': '20px'})
])

#####
# Sección Callback con funciones que retornan gráficos actualizados
#####

# Callbacks para actualizar los filtros y gráficos
@app.callback(
    Output('comuna-filter', 'options'),
    Input('region-filter', 'value')
)
def update_comuna_options(selected_regions):

```

```

if not selected_regions:
    return [{ 'label': com, 'value': com} for com in l_valores_filtros_comunas]
filtered_df = df[df['CODIGO_REGION'].isin(selected_regions)]
filtered_comunas = sorted(filtered_df['CODIGO_COMUNA'].unique())
return [{ 'label': com, 'value': com} for com in filtered_comunas]
#####

# Tipo de Gráfico:
# Gráfica de distribución estadística
# Objetivo:
# Histograma con frecuencia de comunas que llegan a ciertos puntajes en CLEC y MATE1
# separados en bins (0-200, 200-400, 400-600, 600-800, 800-1000)
@app.callback(
    [Output('clec-reg-hist', 'figure'),
     Output('clec-inv-hist', 'figure'),
     Output('mate1-reg-hist', 'figure'),
     Output('mate1-inv-hist', 'figure')],
    [Input('region-filter', 'value'),
     Input('comuna-filter', 'value'),
     Input('year-filter', 'value')]
)

def update_histograms(selected_regions, selected_comunas, selected_years):
    # Preparar los datos con los filtros aplicados
    filtered_df = prepare_data(df, selected_regions, selected_comunas, selected_years)

    # Verificar si hay datos después de filtrar
    if filtered_df.empty:
        empty_fig = go.Figure()
        empty_fig.update_layout(
            title="No hay datos disponibles para los filtros seleccionados",
            xaxis={"visible": False},
            yaxis={"visible": False},
            annotations=[{
                "text": "No hay datos",
                "xref": "paper",
                "yref": "paper",
                "showarrow": False,
                "font": {
                    "size": 28
                }
            }]
        )
    )
    return empty_fig, empty_fig, empty_fig, empty_fig

# Crear histogramas
clec_reg_fig = px.histogram(
    filtered_df.dropna(subset=['CLEC_REG_BIN']),
    x='CLEC_REG_BIN',
    color='CODIGO_COMUNA',
    title='Distribución de Puntajes CLEC Regular',
    category_orders={'CLEC_REG_BIN': bin_labels},
    labels={'CLEC_REG_BIN': 'Rango de Puntaje', 'count': 'Frecuencia'}
)

clec_inv_fig = px.histogram(
    filtered_df.dropna(subset=['CLEC_INV_BIN']),
    x='CLEC_INV_BIN',
    color='CODIGO_COMUNA',
    title='Distribución de Puntajes CLEC Invierno',
    category_orders={'CLEC_INV_BIN': bin_labels},
    labels={'CLEC_INV_BIN': 'Rango de Puntaje', 'count': 'Frecuencia'}
)

mate1_reg_fig = px.histogram(
    filtered_df.dropna(subset=['MATE1_REG_BIN']),

```

```

    filtered_df.dropna(subset=['MATE1_REG_BIN']),
    x='MATE1_REG_BIN',
    color='CODIGO_COMUNA',
    title='Distribución de Puntajes MATE1 Regular',
    category_orders={'MATE1_REG_BIN': bin_labels},
    labels={'MATE1_REG_BIN': 'Rango de Puntaje', 'count': 'Frecuencia'})

mate1_inv_fig = px.histogram(
    filtered_df.dropna(subset=['MATE1_INV_BIN']),
    x='MATE1_INV_BIN',
    color='CODIGO_COMUNA',
    title='Distribución de Puntajes MATE1 Invierno',
    category_orders={'MATE1_INV_BIN': bin_labels},
    labels={'MATE1_INV_BIN': 'Rango de Puntaje', 'count': 'Frecuencia'})

# Ajustar diseño de los gráficos
for fig in [clec_reg_fig, mate1_reg_fig, clec_inv_fig, mate1_inv_fig]:
    fig.update_layout(
        bargroupmode='group',
        xaxis_title='Rango de Puntaje',
        yaxis_title='Frecuencia',
        legend_title='Comuna')

return clec_reg_fig, clec_inv_fig, mate1_reg_fig, mate1_inv_fig
#####

# Tipo de Gráfico:
# Gráfica de series de tiempo
# Objetivo:
# puntajes obtenidos en cada período las pruebas obligatorias de CLEC y MATE1
@app.callback(
    [Output('serie-prom-clecymat', 'figure')],
    [Input('region-filter', 'value')])

def update_timeseries(selected_regions):
    # Para testing:
    #fig = go.Figure(go.Scatter(x=[1,2,3], y=[4,1,2]))
    #return [fig] # Devuelve la figura directamente

    # Step 0: Calculate average scores (tu código original)
    avg_score_df = []

    for test, columns in {
        'CLEC': ['CLEC_REG_ACTUAL', 'CLEC_INV_ACTUAL'],
        'MATE1': ['MATE1_REG_ACTUAL', 'MATE1_INV_ACTUAL']
    }.items():
        for period, column in zip(['REG', 'INV'], columns):
            filtered = df[df[column].notna() & (df[column] > 0)]
            avg = filtered.groupby('YEAR')[column].mean().round(0).reset_index()
            avg['YEAR'] = avg['YEAR'].astype(int)
            avg['TEST'] = test
            avg['TEST_PERIOD'] = period
            avg.rename(columns={column: 'AVG_SCORE'}, inplace=True)
            avg_score_df.append(avg)

    avg_score_df = pd.concat(avg_score_df, ignore_index=True)

    # Step 1: Separate average scores
    clec_scores = avg_score_df[avg_score_df['TEST'] == 'CLEC']
    mate_scores = avg_score_df[avg_score_df['TEST'] == 'MATE1']

```

```

# Create subplots with shared x-axis
fig = make_subplots(rows=2, cols=1,
                    subplot_titles=('Promedio de puntajes CLEC por año y periodo',
                                    'Promedio de puntajes MATE1 por año y periodo'),
                    vertical_spacing=0.15)

# Add CLEC traces
for period in ['REG', 'INV']:
    period_data = clec_scores[clec_scores['TEST_PERIOD'] == period]
    fig.add_trace(
        go.Scatter(
            x=period_data['YEAR'],
            y=period_data['AVG_SCORE'],
            name=f'CLEC {period}',
            mode='lines+markers+text',
            text=period_data['AVG_SCORE'].astype(int),
            textposition='top center',
            marker=dict(color='seagreen' if period == 'REG' else 'steelblue'),
            line=dict(color='seagreen' if period == 'REG' else 'steelblue')
        ),
        row=1, col=1
    )

# Add MATE1 traces
for period in ['REG', 'INV']:
    period_data = mate_scores[mate_scores['TEST_PERIOD'] == period]
    fig.add_trace(
        go.Scatter(
            x=period_data['YEAR'],
            y=period_data['AVG_SCORE'],
            name=f'MATE1 {period}',
            mode='lines+markers+text',
            text=period_data['AVG_SCORE'].astype(int),
            textposition='top center',
            marker=dict(color='seagreen' if period == 'REG' else 'steelblue'),
            line=dict(color='seagreen' if period == 'REG' else 'steelblue')
        ),
        row=2, col=1
    )

# Update layout
fig.update_layout(
    height=800,
    showlegend=True,
    hovermode='x unified'
)

# Update y-axes titles
fig.update_yaxes(title_text='Puntaje promedio', row=1, col=1)
fig.update_yaxes(title_text='Puntaje promedio', row=2, col=1)

# Update x-axes titles
fig.update_xaxes(title_text='Año', row=2, col=1)

return [fig]
#####

# Tipo de Gráfico:
# gráficas de composición Barras "apiladas" por grupo de dependencia (>= 3 intentos)
# Objetivo:
# gráficas de composición Barras "apiladas" por grupo de dependencia (>= 3 intentos)
@app.callback(
    [Output('grafica-pie', 'figure')],

```

```

    Output('grafica-barra', 'figure']],
    [Input('region-filter', 'value'),
    Input('year-filter', 'value']]
)
def update_graphs_composition(selected_regions, selected_years):
    dff = df.copy()
    if selected_regions:
        dff = dff[dff['CODIGO_REGION'].isin(selected_regions)] # Cambiado == por isin()
    if selected_years:
        dff = dff[dff['YEAR'].isin(selected_years)] # Cambiado == por isin() para consistencia

    fig_pie = px.pie(dff, names='GRUPO_DEPENDENCIA', title='Composición por Grupo Dependencia')

    fig_bar = px.bar(dff.groupby('RAMA_EDUCACIONAL').size().reset_index(name='count'),
                     x='RAMA_EDUCACIONAL', y='count',
                     title='Distribución por Rama Educacional')

    fig_bar_apilado = px.bar(
        dff.groupby(['GRUPO_DEPENDENCIA', 'YEAR']).size().reset_index(name='count'),
        x='YEAR',
        y='count',
        color='GRUPO_DEPENDENCIA',
        title='Composición Apilada por Año y Grupo Dependencia',
        barmode='stack'
    )

    return fig_bar, fig_bar_apilado
#####

# Tipo de Gráfico:
# Wordcloud REGULAR
# Objetivo:
# Nube de palabras con ocurrencia de puntajes por bins
@app.callback(
    Output('nube-reg', 'figure'), # Elimina los corchetes
    [Input('region-filter', 'value')]
)
def update_wordcloud_reg(selected_regions):
    # --- REG Data ---
    score_cols_reg = ['CLEC_REG_ACTUAL', 'MATE1_REG_ACTUAL']
    reg_data = df[score_cols_reg].copy()
    reg_data['ID'] = df.index
    reg_data = reg_data.melt(id_vars='ID', value_vars=score_cols_reg, var_name='TEST', value_name='SCORE')
    reg_data = reg_data[(reg_data['SCORE'].notna()) & (reg_data['SCORE'] > 0)]

    # Aplicar la función de binning (asegúrate de definir bin_score_label)
    reg_data['SCORE_BIN'] = reg_data['SCORE'].apply(bin_score_label)

    # --- Count and Format ---
    reg_counts = reg_data.groupby('SCORE_BIN')['ID'].count().reset_index(name='COUNT')
    reg_counts = reg_counts.sort_values(by='COUNT', ascending=False)
    reg_counts['COUNT_FMT'] = reg_counts['COUNT'].apply(lambda x: f"{x:,}".replace(",", "."))

    # --- Crear WordCloud ---
    wc = WordCloud(width=800, height=400, background_color='white', colormap='plasma')
    wc.generate_from_frequencies(dict(zip(reg_counts['SCORE_BIN'], reg_counts['COUNT'])))

    # --- Crear figura de Plotly ---
    fig = go.Figure()

    # Añadir WordCloud como imagen
    fig.add_layout_image(
        dict(
            source=wc.to_image(), # Convertir WordCloud a imagen
            xref="paper",
            yref="paper",
            x=0,

```



```

        y=1,
        size=1,
        sizey=1,
        sizing="stretch",
        layer="below"
    )
)

# Añadir tabla de datos
fig.add_trace(
    go.Table(
        header=dict(
            values=['<b>Rango</b>', '<b>Cantidad</b>'],
            font=dict(size=12, color='white'),
            fill=dict(color='#4a4a4a'),
            align='center'
        ),
        cells=dict(
            values=[reg_counts['SCORE_BIN'], reg_counts['COUNT_FMT']],
            align='center',
            font=dict(size=11)
        ),
        domain=dict(x=[0, 1], y=[0, 0.3]) # Posición de la tabla (30% inferior)
    )
)

# Configurar layout
fig.update_layout(
    title=dict(
        text='WordCloud de Puntajes - REG',
        x=0.5,
        y=0.95,
        font=dict(size=18)
    ),
    xaxis=dict(showgrid=False, zeroline=False, visible=False),
    yaxis=dict(showgrid=False, zeroline=False, visible=False),
    margin=dict(l=20, r=20, t=80, b=20),
    height=700,
    plot_bgcolor='white',
    paper_bgcolor='white'
)

return fig # Devuelve la figura directamente (sin lista)
#####

# Tipo de Gráfico:
# Wordcloud INVIERNO
# Objetivo:
# Nube de palabras con ocurrencia de puntajes por bins
@app.callback(
    Output('nube-inv', 'figure'), # Elimina los corchetes
    [Input('region-filter', 'value')]
)
def update_wordcloud_inv(selected_regions):
    # --- INV Data ---
    score_cols_inv = ['CLEC_INV_ACTUAL', 'MATE1_INV_ACTUAL']
    inv_data = df[score_cols_inv].copy()
    inv_data['ID'] = df.index
    inv_data = inv_data.melt(id_vars='ID', value_vars=score_cols_inv, var_name='TEST', value_name='SCORE')
    inv_data = inv_data[(inv_data['SCORE'].notna()) & (inv_data['SCORE'] > 0)]

    # Aplicar la función de binning
    inv_data['SCORE_BIN'] = inv_data['SCORE'].apply(bin_score_label)

    # --- Count and Format ---
    inv_counts = inv_data.groupby(['SCORE_BIN'])['ID'].count().reset_index(name='COUNT')

```

```

inv_counts = inv_counts.groupby('SCORE_BIN')[ 'COUNT'].count().reset_index(name='COUNT')
inv_counts = inv_counts.sort_values(by='COUNT', ascending=False)
inv_counts['COUNT_FMT'] = inv_counts['COUNT'].apply(lambda x: f"{x:,}".replace(",","."))

# --- Crear WordCloud ---
wc = WordCloud(width=800, height=400, background_color='white', colormap='cool')
wc.generate_from_frequencies(dict(zip(inv_counts['SCORE_BIN'], inv_counts['COUNT'])))

# --- Crear figura de Plotly ---
fig = go.Figure()

# Añadir WordCloud como imagen
fig.add_layout_image(
    dict(
        source=wc.to_image(),
        xref="paper",
        yref="paper",
        x=0,
        y=1,
        size=1,
        sizey=1,
        sizing="stretch",
        layer="below"
    )
)

# Añadir tabla de datos
fig.add_trace(
    go.Table(
        header=dict(
            values=['<b>Rango</b>', '<b>Cantidad</b>'],
            font=dict(size=12, color='white'),
            fill=dict(color='#4a4a4a'),
            align='center'
        ),
        cells=dict(
            values=[inv_counts['SCORE_BIN'], inv_counts['COUNT_FMT']],
            align='center',
            font=dict(size=11)
        ),
        domain=dict(x=[0, 1], y=[0, 0.3])
    )
)

# Configurar layout
fig.update_layout(
    title=dict(
        text='WordCloud de Puntajes - INV',
        x=0.5,
        y=0.95,
        font=dict(size=18)
    ),
    xaxis=dict(showgrid=False, zeroline=False, visible=False),
    yaxis=dict(showgrid=False, zeroline=False, visible=False),
    margin=dict(l=20, r=20, t=80, b=20),
    height=700,
    plot_bgcolor='white',
    paper_bgcolor='white'
)

return fig

```

```
#####
```

```

# Tipo de Gráfico:
# gráfico de comparación de puntajes promedio
# Objetivo:

```

```

# gráfico de comparación de puntajes promedio
@app.callback(
    Output('graf-puntajes', 'figure'),
    [Input('region-filter', 'value'),
     Input('comuna-filter', 'value'),
     Input('year-filter', 'value'),
     Input('tipo1-filter', 'value'),
     Input('tipo2-filter', 'value')]
)
def update_comparison_chart(selected_regions, selected_comunas, selected_years, tipo1, tipo2):
    """
    Función para crear gráfico de comparación de puntajes promedio

    Args:
        selected_regions: Lista de regiones seleccionadas (None si es "Todos")
        selected_comunas: Lista de comunas seleccionadas (None si es "Todos")
        selected_years: Lista de años seleccionados (None si es "Todos")
        tipo1: Primer tipo de dependencia a comparar (None si es "Todos")
        tipo2: Segundo tipo de dependencia a comparar (None si es "Todos")

    Returns:
        plotly.graph_objects.Figure: Figura con el gráfico de comparación
    """
    # Filtrar datos
    df_filtrado = df.copy()

    # Aplicar filtros globales si no es "Todos" (None en Dash)
    if selected_regions:
        df_filtrado = df_filtrado[df_filtrado["CODIGO_REGION"].isin(selected_regions)]
    if selected_comunas:
        df_filtrado = df_filtrado[df_filtrado["CODIGO_COMUNA"].isin(selected_comunas)]
    if selected_years:
        df_filtrado = df_filtrado[df_filtrado["YEAR"].isin(selected_years)]

    if df_filtrado.empty:
        # Crear figura vacía con mensaje
        fig = go.Figure()
        fig.update_layout(
            title="No hay datos disponibles para los filtros seleccionados",
            xaxis={"visible": False},
            yaxis={"visible": False},
            annotations=[{
                "text": "No hay datos",
                "xref": "paper",
                "yref": "paper",
                "showarrow": False,
                "font": {"size": 28}
            }]
        )
        return fig

    # Validar columnas necesarias
    required_cols = ["CLEC_REG_ACTUAL", "MATE1_REG_ACTUAL", "CLEC_INV_ACTUAL",
                    "MATE1_INV_ACTUAL", "GRUPO_DEPENDENCIA"]
    if not all(col in df_filtrado.columns for col in required_cols):
        fig = go.Figure()
        fig.update_layout(
            title="Faltan columnas necesarias en los datos",
            xaxis={"visible": False},
            yaxis={"visible": False},
            annotations=[{
                "text": "Datos incompletos",
                "xref": "paper",
                "yref": "paper",
                "showarrow": False,
                "font": {"size": 28}
            }]

```

```

    })
    )
    return fig

# Validar que tipo1 y tipo2 sean distintos, excepto si son None ("Todos")
if tipo1 and tipo2 and tipo1 == tipo2:
    fig = go.Figure()
    fig.update_layout(
        title="Error en selección de tipos",
        xaxis={"visible": False},
        yaxis={"visible": False},
        annotations=[{
            "text": "Selecciona valores distintos para Tipo 1 y Tipo 2",
            "xref": "paper",
            "yref": "paper",
            "showarrow": False,
            "font": {"size": 20}
        }]
    )
    return fig

categorias = []
valores = []
colores = []

# Si no se selecciona ningún tipo, graficar promedio general
if not tipo1 and not tipo2:
    categorias = ["MATE1 (Total)", "CLEC (Total)", "MATE1_INV (Total)", "CLEC_INV (Total)"]
    valores = [
        df_filtrado["MATE1_REG_ACTUAL"][df_filtrado["MATE1_REG_ACTUAL"] != 0].mean(skipna=True),
        df_filtrado["CLEC_REG_ACTUAL"][df_filtrado["CLEC_REG_ACTUAL"] != 0].mean(skipna=True),
        df_filtrado["MATE1_INV_ACTUAL"][df_filtrado["MATE1_INV_ACTUAL"] != 0].mean(skipna=True),
        df_filtrado["CLEC_INV_ACTUAL"][df_filtrado["CLEC_INV_ACTUAL"] != 0].mean(skipna=True)
    ]
    colores = ['blue', 'orange', 'green', 'red']
else:
    # Agregar datos para tipo1 si está seleccionado
    if tipo1:
        sub_df1 = df_filtrado[df_filtrado["GRUPO_DEPENDENCIA"] == tipo1]
        if not sub_df1.empty:
            categorias += [f"MATE1 ({tipo1})", f"CLEC ({tipo1})", f"MATE1_INV ({tipo1})", f"CLEC_INV ({tipo1})"]
            valores += [
                sub_df1["MATE1_REG_ACTUAL"][sub_df1["MATE1_REG_ACTUAL"] != 0].mean(skipna=True),
                sub_df1["CLEC_REG_ACTUAL"][sub_df1["CLEC_REG_ACTUAL"] != 0].mean(skipna=True),
                sub_df1["MATE1_INV_ACTUAL"][sub_df1["MATE1_INV_ACTUAL"] != 0].mean(skipna=True),
                sub_df1["CLEC_INV_ACTUAL"][sub_df1["CLEC_INV_ACTUAL"] != 0].mean(skipna=True)
            ]
            colores += ['blue', 'orange', 'green', 'red']

    # Agregar datos para tipo2 si está seleccionado
    if tipo2:
        sub_df2 = df_filtrado[df_filtrado["GRUPO_DEPENDENCIA"] == tipo2]
        if not sub_df2.empty:
            categorias += [f"MATE1 ({tipo2})", f"CLEC ({tipo2})", f"MATE1_INV ({tipo2})", f"CLEC_INV ({tipo2})"]
            valores += [
                sub_df2["MATE1_REG_ACTUAL"][sub_df2["MATE1_REG_ACTUAL"] != 0].mean(skipna=True),
                sub_df2["CLEC_REG_ACTUAL"][sub_df2["CLEC_REG_ACTUAL"] != 0].mean(skipna=True),
                sub_df2["MATE1_INV_ACTUAL"][sub_df2["MATE1_INV_ACTUAL"] != 0].mean(skipna=True),
                sub_df2["CLEC_INV_ACTUAL"][sub_df2["CLEC_INV_ACTUAL"] != 0].mean(skipna=True)
            ]
            colores += ['purple', 'brown', 'pink', 'gray']

if not valores:
    fig = go.Figure()
    fig.update_layout(
        title="No hay datos suficientes",

```

```

        xaxis={"visible": False},
        yaxis={"visible": False},
        annotations=[{
            "text": "No hay datos para los filtros",
            "xref": "paper",
            "yref": "paper",
            "showarrow": False,
            "font": {"size": 28}
        }]
    )
    return fig

# Crear figura de Plotly
fig = go.Figure()

# Añadir barras
for cat, val, color in zip(categorias, valores, colores):
    fig.add_trace(go.Bar(
        x=[cat],
        y=[val],
        name=cat,
        marker_color=color,
        text=[f"{val:.1f}"],
        textposition='inside',
        textfont=dict(color='white', size=14, family='Arial'),
        hoverinfo='y+name'
    ))

# Configurar layout
fig.update_layout(
    title='Comparación de Puntajes Promedio',
    xaxis_title='Pruebas y Tipos',
    yaxis_title='Puntaje Promedio',
    barmode='group',
    uniformtext_minsize=8,
    uniformtext_mode='hide',
    height=600,
    showlegend=False
)

return fig
#####

# Tipo de Gráfico:
# gráfico de correlación
# Objetivo:
# gráfico de correlación Puntajes CLE versus MATE1
# Tipo de Gráfico:
# gráfico de correlación
# Objetivo:
# gráfico de correlación Puntajes CLEC versus MATE1
@app.callback(
    Output('graf-corr', 'figure'),
    [Input('region-filter', 'value'),
     Input('year-filter', 'value')]
)
def update_correlation_chart(selected_regions, selected_years):
    """
    Función para crear gráfico de correlación entre CLEC y MATE1

    Args:
        selected_regions: Lista de regiones seleccionadas (None si es "Todos")
        selected_years: Lista de años seleccionados (None si es "Todos")

    Returns:
        plotly.graph objects.Figure: Figura con el gráfico de correlación

```

```

"""
# Filtrar datos
df_filtrado = df.copy()

# Aplicar filtros globales si no es "Todos" (None en Dash)
if selected_regions:
    df_filtrado = df_filtrado[df_filtrado["CODIGO_REGION"].isin(selected_regions)]
if selected_years:
    df_filtrado = df_filtrado[df_filtrado["YEAR"].isin(selected_years)]

if df_filtrado.empty:
    # Crear figura vacía con mensaje
    fig = go.Figure()
    fig.update_layout(
        title="No hay datos disponibles para los filtros seleccionados",
        xaxis={"visible": False},
        yaxis={"visible": False},
        annotations=[{
            "text": "No hay datos",
            "xref": "paper",
            "yref": "paper",
            "showarrow": False,
            "font": {"size": 28}
        }]
    )
    return fig

# Seleccionar columnas relevantes
columnas_seleccionadas = ['CLEC_REG_ACTUAL', 'MATE1_REG_ACTUAL']
df_cor = df_filtrado[columnas_seleccionadas].copy()

# Eliminar ceros y NaNs
df_cor = df_cor[(df_cor['CLEC_REG_ACTUAL'] > 0) & (df_cor['MATE1_REG_ACTUAL'] > 0)]
df_cor = df_cor.dropna()

if df_cor.empty:
    # Crear figura vacía con mensaje
    fig = go.Figure()
    fig.update_layout(
        title="No hay datos válidos para mostrar la correlación",
        xaxis={"visible": False},
        yaxis={"visible": False},
        annotations=[{
            "text": "Datos insuficientes",
            "xref": "paper",
            "yref": "paper",
            "showarrow": False,
            "font": {"size": 28}
        }]
    )
    return fig

# Crear figura de Plotly
fig = go.Figure()

# Añadir gráfico de densidad
fig.add_trace(
    go.Histogram2dContour(
        x=df_cor['CLEC_REG_ACTUAL'],
        y=df_cor['MATE1_REG_ACTUAL'],
        colorscale='Viridis',
        ncontours=20,
        showscale=True,
        name='Densidad'
    )
)

```

```

# Añadir dispersión de puntos
fig.add_trace(
    go.Scatter(
        x=df_cor['CLEC_REG_ACTUAL'],
        y=df_cor['MATE1_REG_ACTUAL'],
        mode='markers',
        marker=dict(
            color='rgba(255, 255, 255, 0.3)',
            size=3,
            line=dict(width=1, color='DarkSlateGrey')
        ),
        name='Puntos',
        showlegend=False
    )
)

# Calcular línea de regresión
try:
    m, b = np.polyfit(df_cor['CLEC_REG_ACTUAL'], df_cor['MATE1_REG_ACTUAL'], 1)
    fig.add_trace(
        go.Scatter(
            x=df_cor['CLEC_REG_ACTUAL'],
            y=m*df_cor['CLEC_REG_ACTUAL'] + b,
            mode='lines',
            line=dict(color='red', width=2),
            name=f'Regresión (r = {df_cor.corr().iloc[0,1]:.2f})'
        )
    )
except:
    pass

# Configurar layout
fig.update_layout(
    title=f'Correlación entre CLEC y MATE1',
    xaxis_title='Puntaje CLEC Regular',
    yaxis_title='Puntaje MATE1 Regular',
    height=600,
    showlegend=True,
    plot_bgcolor='white',
    paper_bgcolor='white'
)

return fig

#####
@app.callback(
    [Output('map-clec', 'figure'),
     Output('map-mate1', 'figure')],
    Input('region-filter', 'value')
)
def update_maps(region_filter):
    # Static Biobio map only
    fig_clec = px.choropleth_mapbox(
        avg_scores,
        geojson=geojson,
        locations='cod_comuna',
        color='Prom_CLEC',
        featureidkey='properties.cod_comuna',
        hover_name='Comuna',
        hover_data=['Prom_CLEC', 'Prom_MATE1'],
        mapbox_style='carto-positron',
        center={"lat": -37.5, "lon": -72.5},
        zoom=6.5,
        color_continuous_scale="RdBu_r",

```

```

    title="Puntaje Promedio CLEC por Comuna - Region del Biobío",
    opacity=0.7
)

fig_mate1 = px.choropleth_mapbox(
    avg_scores,
    geojson=geojson,
    locations='cod_comuna',
    color='Prom_MATE1',
    featureidkey='properties.cod_comuna',
    hover_name='Comuna',
    hover_data=['Prom_CLEC', 'Prom_MATE1'],
    mapbox_style='carto-positron',
    center={"lat": -37.5, "lon": -72.5},
    zoom=6.5,
    color_continuous_scale="YlGnBu",
    title="Puntaje Promedio MATE1 por Comuna - Región del Biobío",
    opacity=0.7
)

# Update layout for both figures
for fig in [fig_clec, fig_mate1]:
    fig.update_layout(
        margin={"r": 0, "t": 40, "l": 0, "b": 0},
        coloraxis_colorbar={
            'title': 'Puntaje'
        }
    )

return fig_clec, fig_mate1

#####
# Ejecutar la aplicación
if __name__ == '__main__':
    app.run(debug=True)

```




Rendición de PAES 2022 a 2025

Comparación de período regular e invierno para las pruebas base de Comprensión de Lectura (CLEC) y Matemáticas (MATE1)

Descripción Dashboard

Este dashboard muestra las diferencias entre la rendición de la PAES entre 2022 y 2025 en ambos periodos: regular e invierno para las pruebas base de Comprensión de Lectura (CLEC) y Matemáticas (MATE1). Buscamos responder la hipótesis de si las rendiciones en el periodo de invierno tienen mayor puntaje promedio que en la rendición regular.

Detalle de visualizaciones

- Evolución de los puntajes promedio de ambas pruebas.
- La rama educacional y grupo de dependencia de origen.
- Distribuciones de puntajes promedio por región y comuna (bins de 200 puntos).
- Nube de palabras de puntajes promedio (bins de 100 puntos).
- Histograma de puntajes promedio.
- Distribución de puntajes de CLEC versus MATE1.
- Filtros disponibles: Región, Comuna y Año de rendición.

Región

REGION DEL BIOBIO

Comuna

Seleccione comuna(s)

Año

2022202320242025

Promedio de puntajes CLEC por año y periodo

Año	CLEC REG	CLEC INV
2022	500	500
2023	643	630
2024	599	672
2025	596	636

Promedio de puntajes MATE1 por año y periodo

Año	MATE1 REG	MATE1 INV
2022	501	501
2023	562	480
2024	604	636
2025	625	695

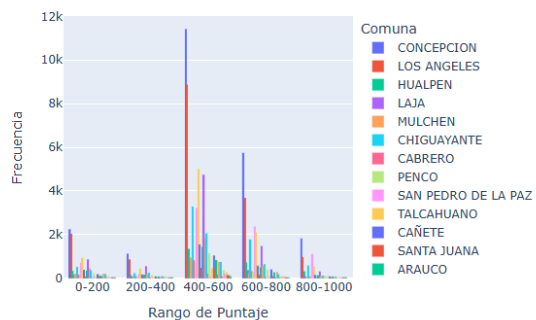
Distribución por Rama Educacional

RAMA_EDUCACIONAL	count
Humanista científico Diurno	80000
Humanista científico Nocturno	10000
Técnico profesional Industrial	10000
Técnico profesional Marítima	10000
Técnico profesional Servicios y técnica	10000
Técnico profesional agrícola comercial	10000

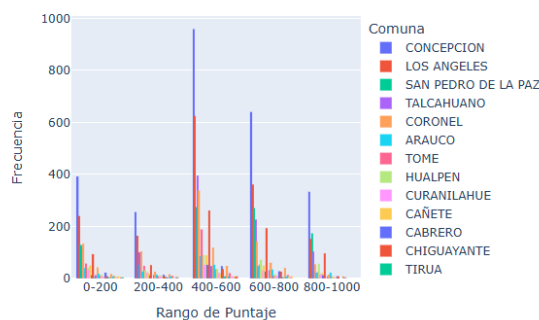
Composición Apilada por Año y Grupo Dependencia

YEAR	Municipal	Particular pagado	Particular subvencionado	Servicio Local de Educación
2022	10000	2000	12000	1000
2023	10000	2000	12000	1000
2024	10000	2000	12000	1000
2025	10000	2000	12000	1000

Distribución de Puntajes MATE1 Regular



Distribución de Puntajes MATE1 Invierno



WordCloud de Puntajes - REG



WordCloud de Puntajes - INV



Puntaje Promedio CLEC por Comuna - Región del Biobío

