

## Adatbányászat a Gyakorlatban

### 4. Gyakorlat: Pontdiagramok és viselkedésük

Kuknyó Dániel  
Budapesti Gazdasági Egyetem

2024/25  
1.félév

## 1 Bevezetés

## 2 Színek kezelése

## 3 Kiugró értékek

## 4 Csúszkák

## 5 Tematikus térképek

## 6 Pontdiagramok térképekkel

## 7 Markdown

## 1 Bevezetés

## 2 Színek kezelése

## 3 Kiugró értékek

## 4 Csúszkák

## 5 Tematikus térképek

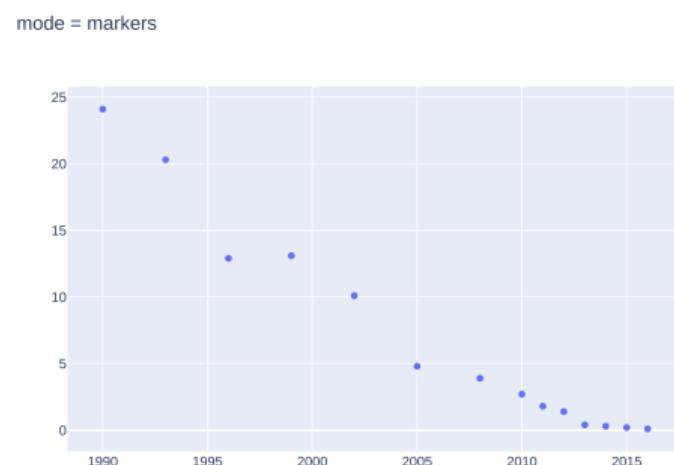
## 6 Pontdiagramok térképekkel

## 7 Markdown

# Alapvető pontdiagramok

A plotly könyvtárban a pontdiagramokat a `graph_objects` valósítja meg. Az alapvető jelölőtípusok:

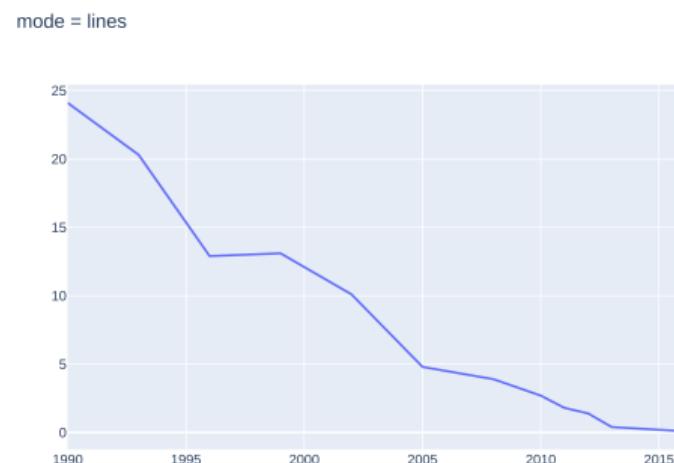
- `markers`: Csak jelölők
- `lines`: Csak vonalak
- `text`: Csak szöveget
- `markers+lines`: Jelölők és vonalak
- `markers+text`: Jelölők és szöveg
- `lines+text`: Vonalak és szöveg
- `markers+lines+text`: Jelölők, vonalak és szöveg



# Alapvető pontdiagramok

A plotly könyvtárban a pontdiagramokat a `graph_objects` valósítja meg. Az alapvető jelölőtípusok:

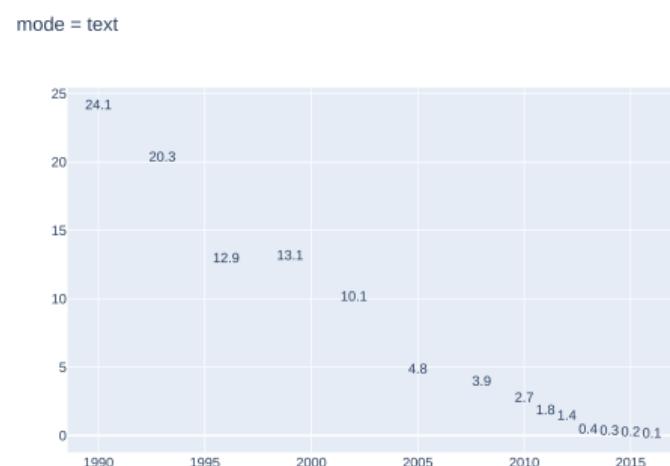
- `markers`: Csak jelölők
- `lines`: Csak vonalak
- `text`: Csak szöveget
- `markers+lines`: Jelölők és vonalak
- `markers+text`: Jelölők és szöveg
- `lines+text`: Vonalak és szöveg
- `markers+lines+text`: Jelölők, vonalak és szöveg



# Alapvető pontdiagramok

A plotly könyvtárban a pontdiagramokat a `graph_objects` valósítja meg. Az alapvető jelölőtípusok:

- `markers`: Csak jelölők
- `lines`: Csak vonalak
- `text`: Csak szöveget
- `markers+lines`: Jelölők és vonalak
- `markers+text`: Jelölők és szöveg
- `lines+text`: Vonalak és szöveg
- `markers+lines+text`: Jelölők, vonalak és szöveg



# Alapvető pontdiagramok

A plotly könyvtárban a pontdiagramokat a `graph_objects` valósítja meg. Az alapvető jelölőtípusok:

- `markers`: Csak jelölők
- `lines`: Csak vonalak
- `text`: Csak szöveget
- `markers+lines`: Jelölők és vonalak
- `markers+text`: Jelölők és szöveg
- `lines+text`: Vonalak és szöveg
- `markers+lines+text`: Jelölők, vonalak és szöveg

mode = markers+lines

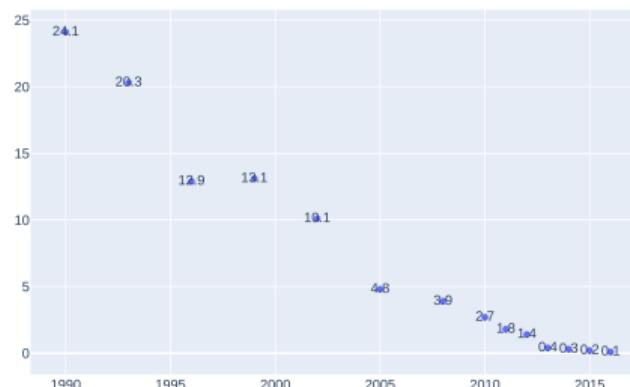


# Alapvető pontdiagramok

A plotly könyvtárban a pontdiagramokat a `graph_objects` valósítja meg. Az alapvető jelölőtípusok:

- `markers`: Csak jelölők
- `lines`: Csak vonalak
- `text`: Csak szöveget
- `markers+lines`: Jelölők és vonalak
- `markers+text`: Jelölők és szöveg
- `lines+text`: Vonalak és szöveg
- `markers+lines+text`: Jelölők, vonalak és szöveg

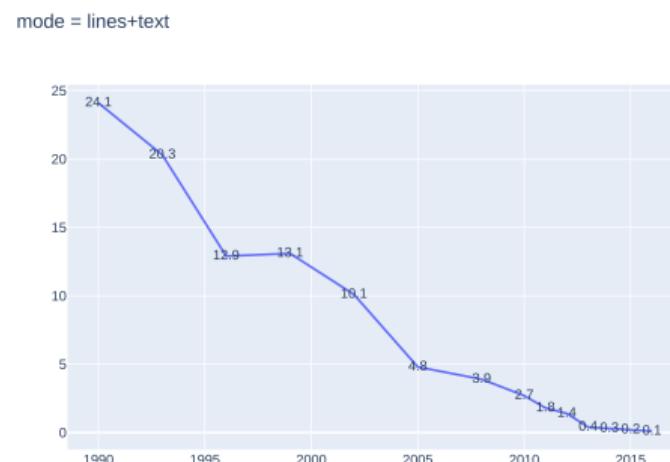
mode = markers+text



# Alapvető pontdiagramok

A plotly könyvtárban a pontdiagramokat a `graph_objects` valósítja meg. Az alapvető jelölőtípusok:

- `markers`: Csak jelölők
- `lines`: Csak vonalak
- `text`: Csak szöveget
- `markers+lines`: Jelölők és vonalak
- `markers+text`: Jelölők és szöveg
- `lines+text`: Vonalak és szöveg
- `markers+lines+text`: Jelölők, vonalak és szöveg

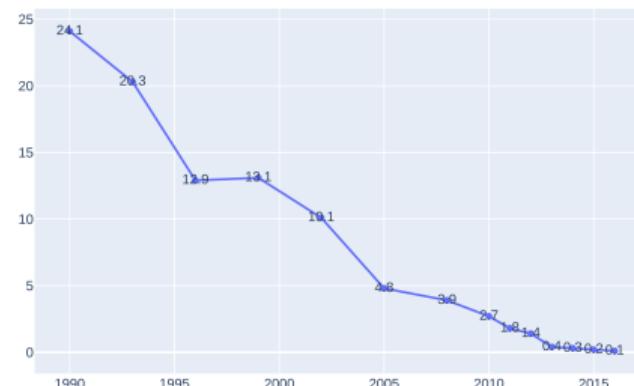


# Alapvető pontdiagramok

A plotly könyvtárban a pontdiagramokat a `graph_objects` valósítja meg. Az alapvető jelölőtípusok:

- `markers`: Csak jelölők
- `lines`: Csak vonalak
- `text`: Csak szöveget
- `markers+lines`: Jelölők és vonalak
- `markers+text`: Jelölők és szöveg
- `lines+text`: Vonalak és szöveg
- `markers+lines+text`: Jelölők, vonalak és szöveg

mode = `markers+lines+text`



# Több nyom egy diagramon

Ahhoz, hogy egyszerre több nyomvonal szerepeljen egy diagramon, ezeket egymás után kell hozzáadni a Figure objektumhoz.

```
1 for country in countries:  
2     fig.add_scatter(x=df_country['year'],  
                         y=df_country[perc_pov_19], name=  
                         country, mode='markers+lines')
```



## 1 Bevezetés

## 2 Színek kezelése

## 3 Kiugró értékek

## 4 Csúszkák

## 5 Tematikus térképek

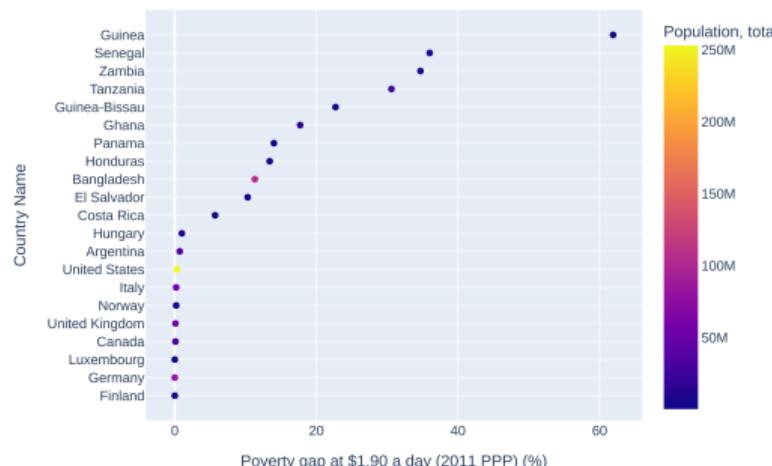
## 6 Pontdiagramok térképekkel

## 7 Markdown

# Alapértelmezett színezés folytonos változóval

A színezést a px.scatter() függvény hívásakor a color paraméterének állításával lehet elérni. Folytonos változó esetén a vászon jobb oldalán egy folytonos színskála fog megjelenni.

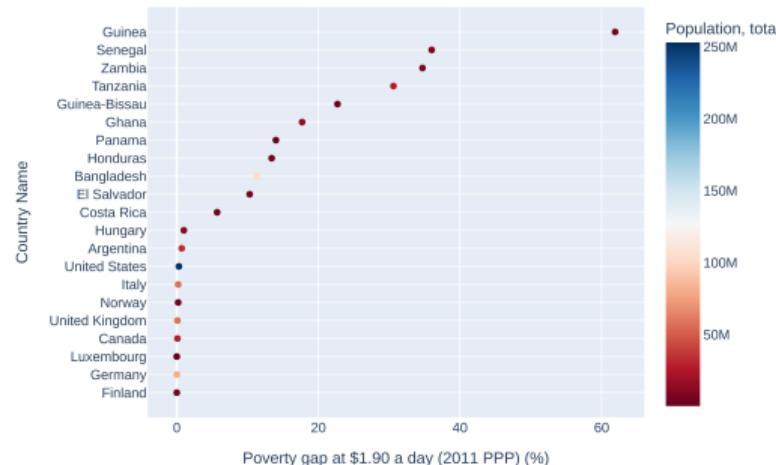
```
1 fig = px.scatter(df, x=indicator, y='Country Name', color='Population, total')
```



# Személyre szabott színezés folytonos változóval

A színskála megadható a `color_continuous_scale` paraméter állításával. A színskálák listája ezen a linken érhető el.

```
1 fig = px.scatter(df, x=indicator, y='Country Name', color='Population, total', color_continuous_scale='RdBu')
```

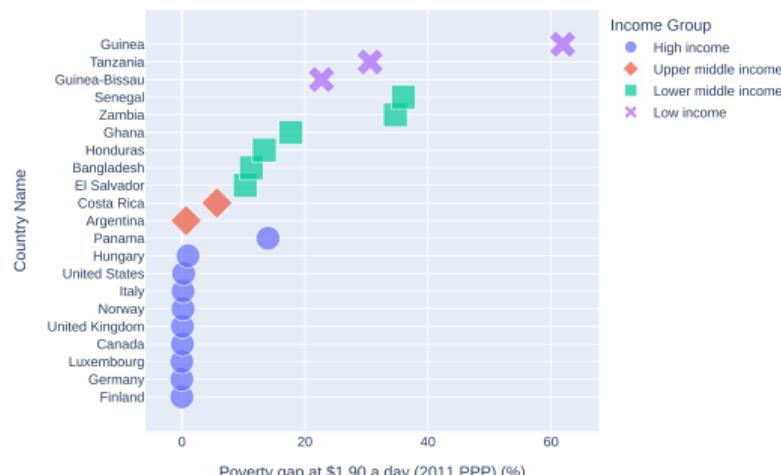


# Színek diszkrét változókkal

Ha a diagramon ábrázolt függő változó diszkrét, nem egy színskála jön létre, hanem a diagram jobb oldalán a lehetséges kategóriák lesznek felsorolva, amelyek közül kattintással lehet kiválasztani a megjelenítendő nyomot.

```
1 fig = px.scatter(  
2     df,  
3     x=indicator,  
4     y='Country Name',  
5     color='Income Group',  
6     ...  
7 )
```

Poverty gap at \$1.90 a day (2011 PPP) (%) - 1991 -  
color='Income Group'

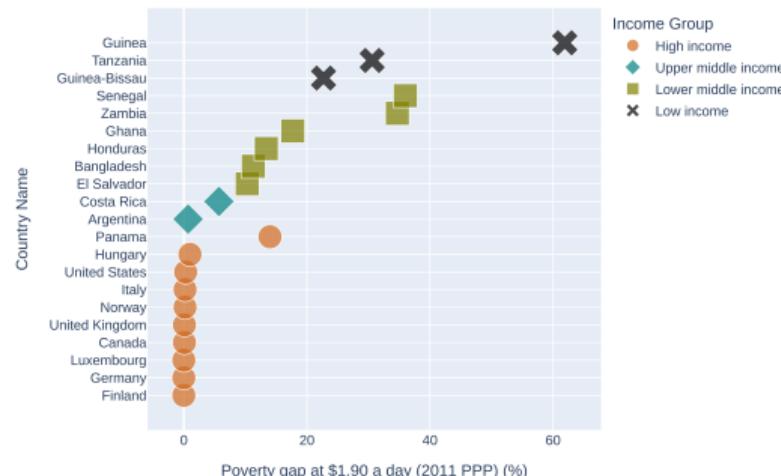


# Egyéni színek diszkrét változókkal

A `color_discrete_sequence` paraméter egy listát fogad, ahol minden elem egy színnevet tartalmazó szöveges érték.

```
1 fig = px.scatter(  
2     df,  
3     x=indicator,  
4     y='Country Name',  
5     color='Income Group',  
6     symbol='Income Group',  
7     color_discrete_sequence=['chocolate',  
8         'teal', 'olive', 'black'],  
9     ...  
)
```

Poverty gap at \$1.90 a day (2011 PPP) (%) - 1991 -  
`color_discrete_sequence=['chocolate', 'teal', 'olive', 'black']`



## 1 Bevezetés

## 2 Színek kezelése

## 3 Kiugró értékek

## 4 Csúszkák

## 5 Tematikus térképek

## 6 Pontdiagramok térképekkel

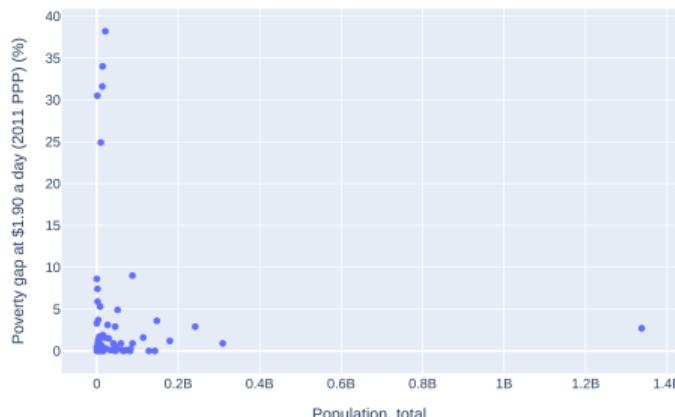
## 7 Markdown

# Kiugró értékek a gyakorlatban

A következő példában egyetlen outlier érték, Kína a maga 1.4 milliárd lélekszámával az összes többi országot egy kis területbe szorítja a diagram bal oldalán.

A következő fejezet azzal fog foglalkozni, hogy hogyan lehet ilyen jelenségeket kezelní.

Poverty gap at \$1.90 a day (2011 PPP) (%) - 2010

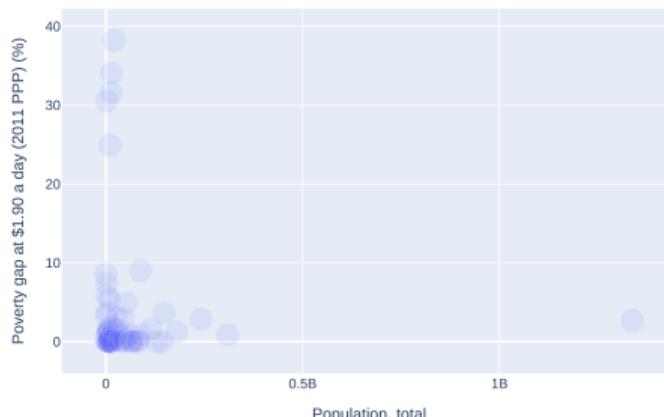


# Markerek áttetszőségének és méretének állítása

Az opacity paraméter egy  $[0, 1]$  intervallumba eső tizedes törtet vár el paraméterül. A 0 érték egy teljesen áttetsző, az 1 pedig egy teljesen átlátszatlan markert fog eredményezni.

Mivel a markerek nagyon kicsik, a size paraméter növelésével jobban láthatóvá lehet őket tenni.

Poverty gap at \$1.90 a day (2011 PPP) (%) - 2010 -  
opacity=0.1, size=[5]\*len(df), size\_max=15

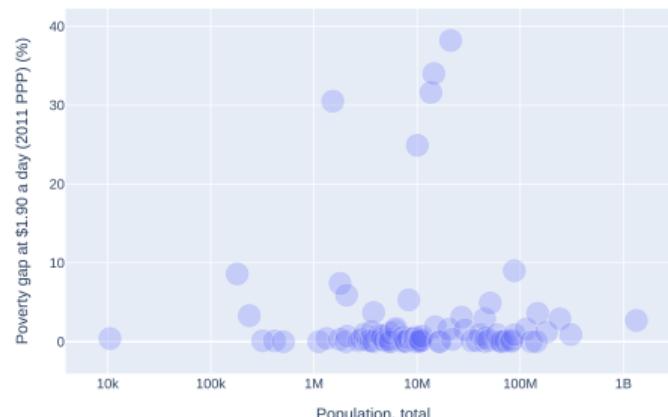


# Logaritmikus skála használata

A logaritmikus skála olyan skála, amelyen az értékek nem egyenletesen, hanem logaritmikus arányban vannak elosztva. Például egy 10-es alapú logaritmikus skálán a jelölések 1, 10, 100, 1000 stb. lehetnek.

Logaritmikus skálát egy plotly diagramon a `log_x` paraméter bekapcsolásával lehet létrehozni.

Poverty gap at \$1.90 a day (2011 PPP) (%) - 2010 -  
opacity=0.25, size=[5]\*len(df), size\_max=15 log\_x=True



## 1 Bevezetés

## 2 Színek kezelése

## 3 Kiugró értékek

## 4 Csúszkák

## 5 Tematikus térképek

## 6 Pontdiagramok térképekkel

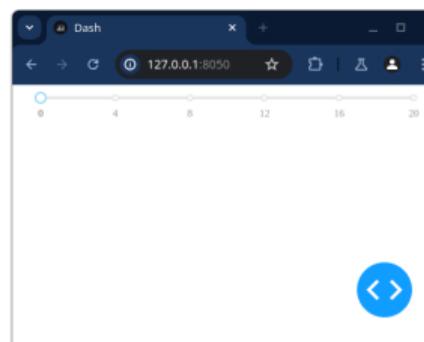
## 7 Markdown

# Dash csúszkák

A Slider és RangeSlider komponensek körök, amelyeket a felhasználók húzhatnak egy érték beállításához, és folyamatos vagy kategorikus értékekhez is használhatók.

A Slider egyetlen értéket állít, míg a RangeSlider komponensen két csúszka szerepel, és az ezek által határolt tartományt adja meg.

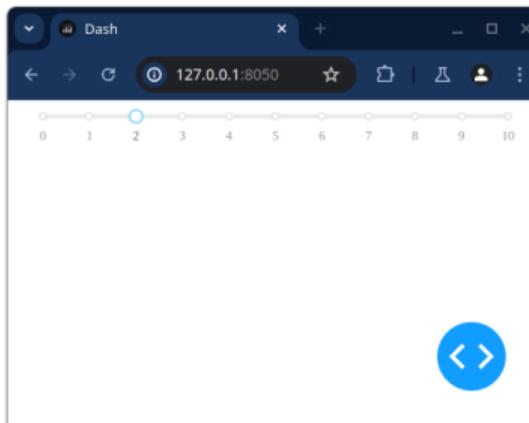
```
1 app = dash.Dash(__name__)
2 app.layout = html.Div([
3     dcc.Slider(
4         id='slider',
5         min=0,
6         max=20
7     )
8 ])
9 app.run_server(mode='inline')
```



# Slider komponensek paraméterei

- min: A csúszka minimum értéke
- max: A csúszka maximum értéke
- step: A legkisebb állítható lépték
- dots: Szerepeljenek-e markerek a csúszkán
- included: Ha az értéke False, a markerek közötti értéket nem veheti fel a csúszka állapota
- marks: Egy szótár, ahol a kulcsok a csúszka értékei, és az értékek azok a címkék, amelyek az adott értékeknél jelennek meg.

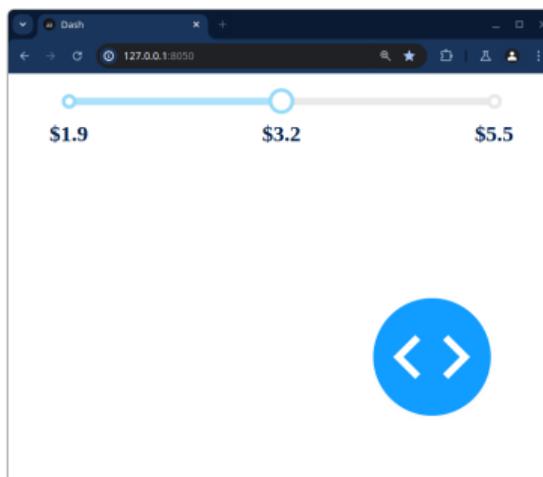
```
1dcc.Slider(  
2    min=0,  
3    max=10,  
4    step=1,  
5    dots=True,  
6    included=False  
7 )
```



# Markerek testreszabása

A `marks` paraméter egy szótárat fogad, aminek a kulcsa a címke, és a hozzá tartozó érték egy másik szótár, amiben a címkéhez tartozó tulajdonságok vannak megadva.

```
1 marks={  
2   0: {'label': '$1.9', 'style': {'color': cividis0, 'fontWeight': 'bold'}},  
3   1: {'label': '$3.2', 'style': {'color': cividis0, 'fontWeight': 'bold'}},  
4   2: {'label': '$5.5', 'style': {'color': cividis0, 'fontWeight': 'bold'}}},  
5 }
```



# Alkalmazás csúszkákkal (app\_v3\_1.py)

Az alkalmazás következő iterációjában két csúszka került hozzáadásra, az egyikkel a szegénységi szintet, a másikkal pedig az évet lehetséges kiválasztani.

A hozzá tartozó callback függvény a csúszkák állapotának változásának hatására elindul, és leszűri a megfelelő adatkészletet. Az adatkészletből egy plotly diagramot állít elő és tériti vissza a megfelelő Output attribútumba.

A teljes alkalmazásba való beépítést az app\_v3\_2.py valósítja meg.



## 1 Bevezetés

## 2 Színek kezelése

## 3 Kiugró értékek

## 4 Csúszkák

## 5 Tematikus térképek

## 6 Pontdiagramok térképekkel

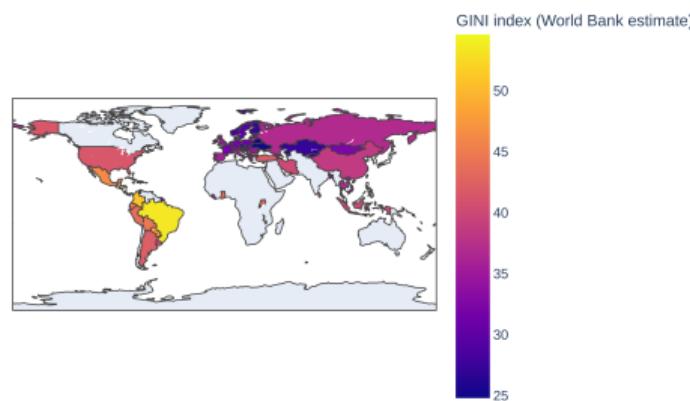
## 7 Markdown

# Egyeszerű tematikus térkép

Egy tematikus térképhez szükség van egy érték oszlopra, és egy országkód oszlopra a rendelkezésre álló adatkészletben.

Az országkódokat általában ISO 3166-1 alpha-3 formátumban használja, ami hárombetűs kódokat jelent (pl. Magyarország esetében "HUN").

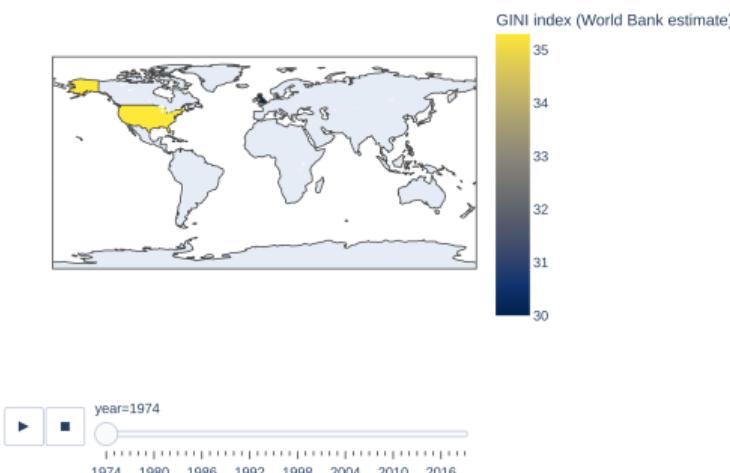
```
1 fig = px.choropleth(df, locations="Country Code", color=indicator)
```



# Animációs réteg tematikus térképekkel

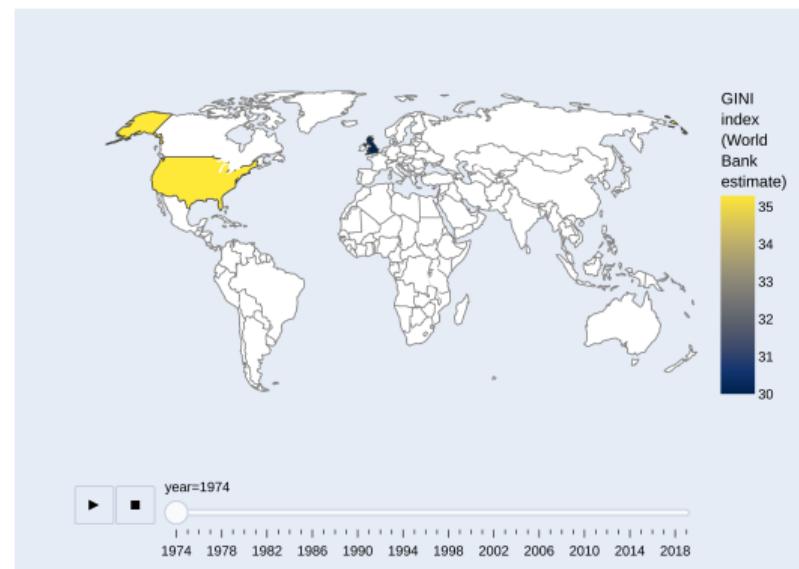
Az `animation_frame` paraméterrel lehetséges bevezetni egy új, interaktív réteget adó komponenst, ami a választott változó alapján képes szekvenciálisan változtatni a megjelenített diagramot.

```
1 fig = px.choropleth(  
2     poverty[poverty['is_country']],  
3     color_continuous_scale='cividis',  
4     locations='Country Code',  
5     color=indicator,  
6     animation_frame='year'  
7 )
```



# Fontosabb paraméterek tematikus térképekkel

- `fig.layout.geo.showframe:`  
Eltünteti a keretet a térképdobozról
- `fig.layout.geo.showcountries:`  
Mutatja az országok keretezővonalait
- `fig.layout.geo.projection.type:`  
Térkép projekció állítása
- `fig.layout.geo.landcolor:` A föld színének állítása
- `fig.layout.geo.bgcolor:`  
Háttérszín a térképen
- `fig.layout.paper_bgcolor:`  
Háttérszín a kereteződobozban



# Callback függvények térképekkel

## ① Új DropDownList komponens létrehozása:

```
1 dcc.Dropdown(  
2     id='indicator_dropdown',  
3     value='GINI index (World Bank  
        estimate)',  
4     options=[{'label': indicator, '  
          value': indicator} for  
            indicator in poverty.columns  
            [3:54]]  
5 )
```

## ② Graph komponens létrehozása a térfénpnek:

```
1 dcc.Graph(id='indicator_map_chart',  
2 )
```

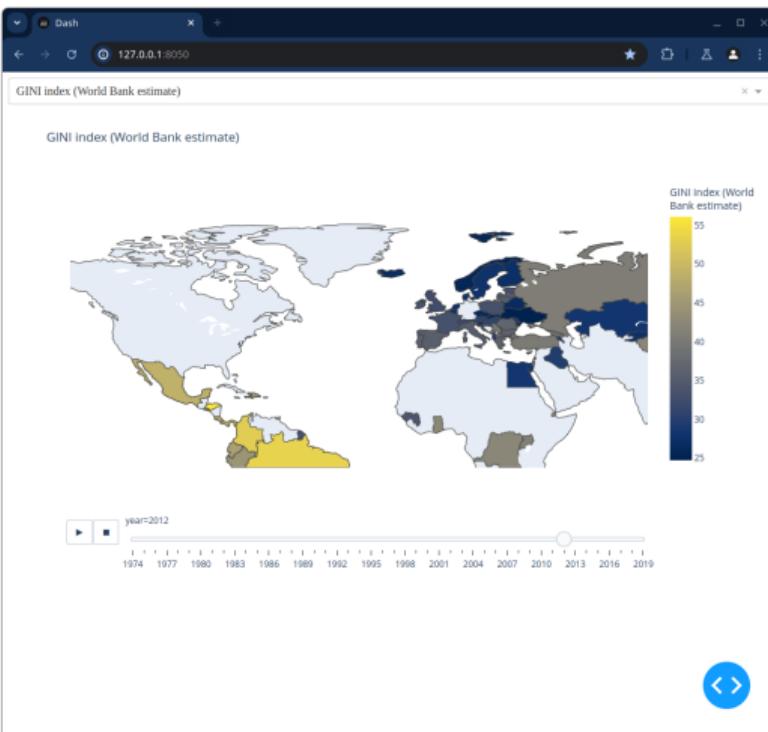
## ③ Callback függvény létrehozása

```
1 @app.callback(  
2     Output('indicator_map_chart', '  
        figure'),  
3     Input('indicator_dropdown', '  
        value'))  
4 def display_generic_map_chart(  
        indicator):  
5     df = poverty[poverty['is_country  
        ']]  
6     fig = px.choropleth(  
7         df,  
8         locations='Country Code',  
9         color=indicator,  
10        ...  
11    )  
12    fig.layout.geo.showframe = False  
13    ...  
14    return fig
```

# Alkalmazás legördülő menüvel és tematikus térképpel (map\_app\_v1.py)

Az alkalmazás működésének megfelelően a felhasználó kiválaszthat egy indikátort a Dropdown menüből, ez elindít egy callback függvényt, aminek átadódik az indikátor értéke.

A legördülő menü állapotát felhasználva a callback függvény renderel egy tematikus térképet, és felülírja a Graph komponens figure attribútumát.



# dcc.Store komponensek

A tároló komponenseket arra lehet használni, hogy a kliens oldalon tároljon el adatot anélkül, hogy azt visszaküldené a szervernek.

Dash alkalmazásokban a dcc.Store komponensek tartalmát callback függvények segítségével lehet manipulálni.

```
1 app.layout = html.Div([
2     dcc.Store(id='store', storage_type='
3         session'),
4 )
5 @app.callback(
6     Output('my-store', 'data'),
7     Input('save-button', 'n_clicks')
8 )
9 def save_data(n_clicks):
10     if n_clicks:
11         return {'key': 'value'}
12     return dash.no_update
```

# dcc.Store komponensek

A Store komponenseknek 3 típusa

létezik:

- **memory**: Az adat a böngésző memóriájában tárolódik, és törlődik, amikor az oldalt frissíti a felhasználó
- **local**: Az adat a böngésző helyi tárhelyén tárolódik el, és frissítés után nem törlődik
- **session**: Az adat a böngésző munkamenete során marad meg, és akkor törlődik, amikor a felhasználó bezárja a megfelelő lapot

```
1 app.layout = html.Div([
2     dcc.Store(id='store', storage_type='
3         session'),
4 )
5
5 @app.callback(
6     Output('my-store', 'data'),
7     Input('save-button', 'n_clicks')
8 )
9 def save_data(n_clicks):
10    if n_clicks:
11        return {'key': 'value'}
12    return dash.no_update
```

# dcc.Interval komponensek

A dcc.Interval komponenseket arra lehet használni, hogy egy adott callback függvényt elindítson az alkalmazás adott időközönként. Ilyen például adatkészletek, diagramok frissítése, vagy folyamatok állapotának ellenőrzése. Fontosabb paraméterei:

- interval: Az intervallum (ms) hossza, ami két callback indítás között eltelik
- n\_intervals: Az eltelt intervallumok számát tartalmazó attribútum
- disabled: Ha értéke True, a callback függvény nem indul el

```
1 app.layout = html.Div([
2     dcc.Interval(
3         id='interval-component',
4         interval=1 * 1000,
5         n_intervals=0,
6     ),
7 ])
8
9 @app.callback(
10     Output('output', 'children'),
11     Input('interval-component', 'n_intervals')
12 )
13 def update_output(n):
14     return f'Interval has triggered {n} times.'
```

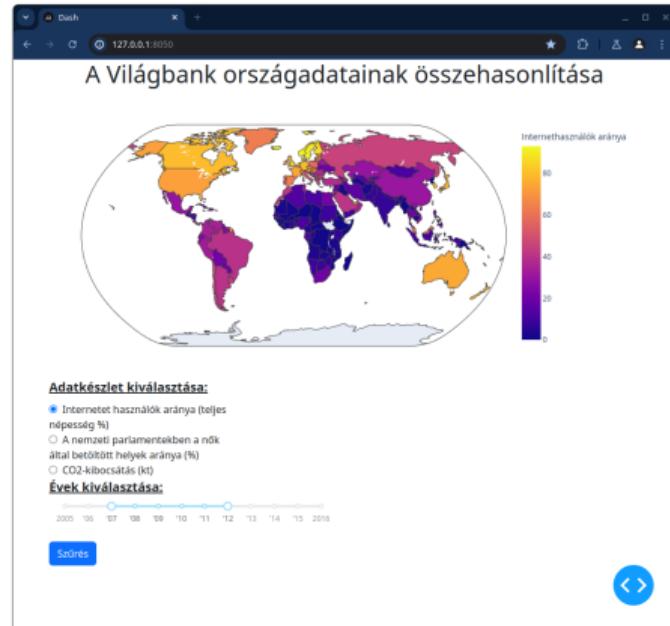
# Alkalmazás Storage és Interval komponensekkel (map\_app\_v2.py)

dcc.Storage:

- A `store_data` callback frissíti a `dcc.Store` komponens `data` attribútumát a Világbank adataival minden alkalommal, amikor a `dcc.Interval` komponens frissítést indít.

dcc.Interval:

- A `store_data` callback minden alkalommal aktiválódik, amikor a `dcc.Interval` komponens növeli az `n_intervals` attribútum értékét. Ez biztosítja, hogy a Világbank adatai minden percben frissüljenek és tárolódjjanak a `dcc.Store` komponensben.



## 1 Bevezetés

## 2 Színek kezelése

## 3 Kiugró értékek

## 4 Csúszkák

## 5 Tematikus térképek

## 6 Pontdiagramok térképekkel

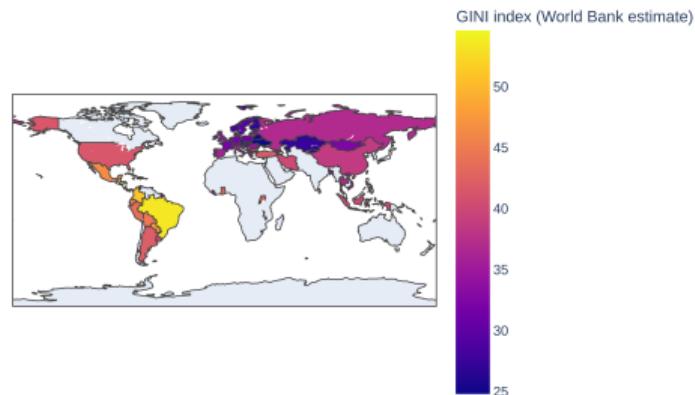
## 7 Markdown

# Térkép projekciók

A térkép projekció egy matematikai módszer melynek feladata, hogy a Föld gömbölyű felületét egy síkra vetítse.

A folyamat során szükségszerűen torzulások keletkeznek.

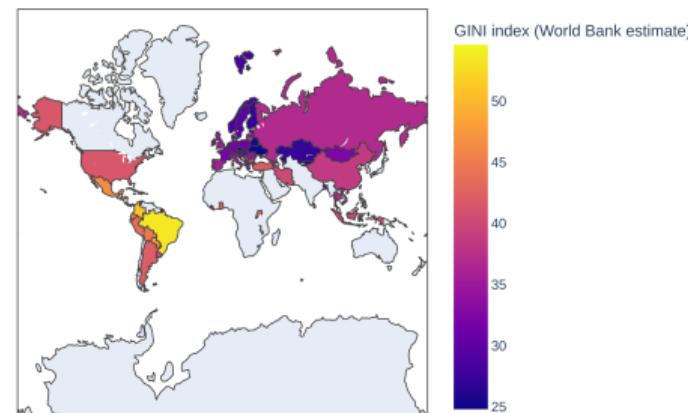
projection = equirectangular



# Térkép projekciók

- **Equirectangular:** A legegyszerűbb projekció, ahol a földrajzi szélesség és hosszúság egyenesen arányosan van ábrázolva a síkon.
- **Mercator:** Szögtartó, tehát a szögek és az irányok helyesek maradnak, így hasznos a navigációban.
- **Orthographic:** Háromdimenziós gömböt ábrázol síkban úgy, mintha egy távoli pontból néznénk.
- **Sinusoidal:** Ez az egyenlő területű projekció, amely megőrzi a területek arányait.

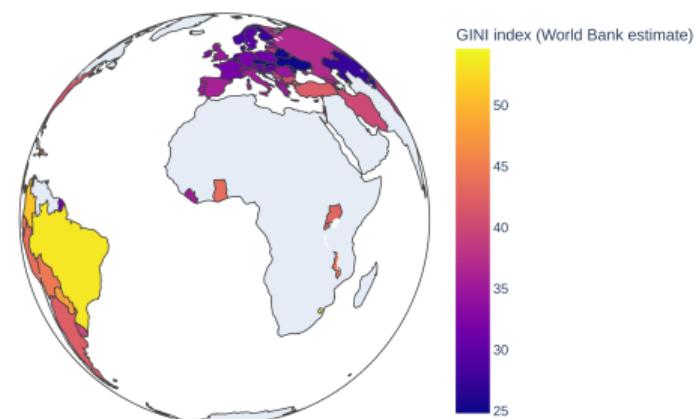
projection = mercator



# Térkép projekciók

- **Equirectangular:** A legegyszerűbb projekció, ahol a földrajzi szélesség és hosszúság egyenesen arányosan van ábrázolva a síkon.
- **Mercator:** Szögtartó, tehát a szögek és az irányok helyesek maradnak, így hasznos a navigációban.
- **Orthographic:** Háromdimenziós gömböt ábrázol síkban úgy, mintha egy távoli pontból néznénk.
- **Sinusoidal:** Ez az egyenlő területű projekció, amely megőrzi a területek arányait.

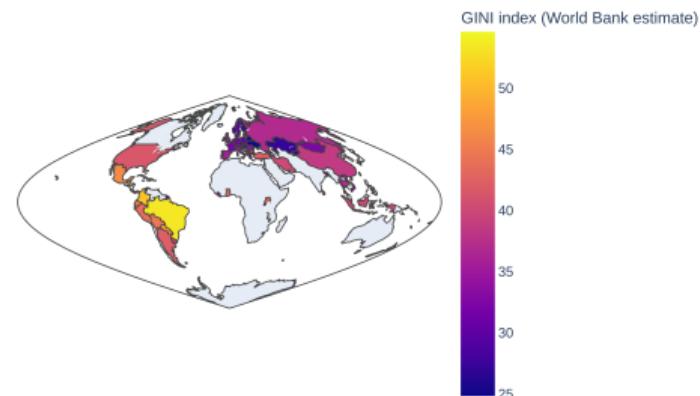
projection = orthographic



# Térkép projekciók

- **Equirectangular:** A legegyszerűbb projekció, ahol a földrajzi szélesség és hosszúság egyenesen arányosan van ábrázolva a síkon.
- **Mercator:** Szögtartó, tehát a szögek és az irányok helyesek maradnak, így hasznos a navigációban.
- **Orthographic:** Háromdimenziós gömböt ábrázol síkban úgy, mintha egy távoli pontból néznénk.
- **Sinusoidal:** Ez az egyenlő területű projekció, amely megőrzi a területek arányait.

projection = sinusoidal



# Pontdiagramok térképekkel

A plotly könyvtár alapértelmezetten támogatja az ISO-alpha3 országkódok pontos pozíójának ábrázolását, ezért amikor paraméterül megkapja a locations='Country Code' értéket innen ki tudja olvasni az adott ország hosszúsági és szélességi koordinátáit.

```
1 df = poverty[poverty['year'].eq(2010) &  
     poverty['is_country']]  
2 fig = px.scatter_geo(df, locations=  
     'Country Code')
```



# Mapbox térképek

A `zoom` paraméter állításával lehetséges ráközelíteni a diagramra, a `center` paraméter a közelítés központját adja meg, és a `mapbox_style` pedig a térképstílust.

```
1 px.scatter_mapbox(  
2     lon=[5, 10, 15, 20],  
3     lat=[10, 7, 18, 5],  
4     zoom=2,  
5     center={'lon': 5, 'lat': 10},  
6     size=[5]*4,  
7     color_discrete_sequence=['darkred'],  
8     mapbox_style='stamen-watercolor'  
)
```



## 1 Bevezetés

## 2 Színek kezelése

## 3 Kiugró értékek

## 4 Csúszkák

## 5 Tematikus térképek

## 6 Pontdiagramok térképekkel

## 7 Markdown

# Markdown alapjai

A Markdown nyelv segítségével egyszerűen lehet HTML struktúrát létrehozni. Az outputot úgy jeleníti meg, mint bármelyik HTML dokumentum, de a megírása sokkal egyszerűbb.

## HTML

```
1 <h1>Főcím</h1>
2 <h2>Alcím</h2>
3 <ul>
4   <li>Első elem</li>
5   <li>Második elem</li>
6   <li>Harmadik elem</li>
7 </ul>
```

## Markdown

```
1 # Főcím
2 ## Alcím
3 * Első elem
4 * Második elem
5 * Harmadik elem
```

# Markdown szabályai

- Főcímek:

```
1 # Első főcím
2 ## Második főcím
3 ### Harmadik főcím
```

- Formázások:

```
1 *Dölt*
2 **Félkövér**
3 ***Félkövér és dölt***
```

- Linkek:

```
1 [Link szöveg](URL)
```

- Képek:

```
1 !*[Alternatív szöveg](Elérési út)
```

- Számozatlan lista:

```
1 * Első elem
2 * Második elem
```

- Számozott lista:

```
1 1. Első elem
2 2. Második elem
```

- Programkód:

```
1 'print(Hello, World!)'
```

- Táblázat:

	Főcím 1	Főcím 2
1	-----	-----
2	Sor 1	Adat

# dcc.Markdown komponensek frissítése callback függvényel

Dash keretrendszer alatt dcc.Markdown komponenseket lehetséges definiálni. Ezeket callback függvények segítségével dinamikusan lehet frissíteni.

Markdown komponens létrehozása:

```
1  dcc.Markdown(  
2      id='indicator_map_details_md',  
3      style={'backgroundColor': '#E5ECF6'})
```

Callback dekorátor:

```
1 @app.callback(  
2     Output('indicator_map_chart', 'figure'),  
3     Output('indicator_map_details_md', 'children'),  
4     Input('indicator_dropdown', 'value'))
```

```
1 def update(indicator):  
2     ...  
3     markdown = f"""  
4     ---  
5     ## {series_df['Indicator Name'].values[0]}  
6     {series_df['Long definition'].values[0]}  
7     * **Mértékegység:** {series_df['Unit of measure'].fillna('count').values[0]}  
8     * **Periodicitás:** {series_df['Periodicity'].fillna('N/A').values[0]}  
9     * **Forrás:** {series_df['Source'].values[0]}  
10    ### Limitációk és kivételek:  
11    {limitations}  
12    """  
13    return markdown
```

# Alkalmazás térképpel és Markdown komponenssel (app\_v3\_3.py)

