

Interaktív alkalmazások Python segítségével

Plotly, Dash

Kuknyó Dániel
Budapesti Gazdasági Egyetem

2025. nov. 4.

1 Dash alapok

2 Diagramok

3 Pontdiagramok

4 Gyakorisági adatok

1 Dash alapok

2 Diagramok

3 Pontdiagramok

4 Gyakorisági adatok

Órai környezet telepítése

- ① Új Anaconda környezet létrehozása dash néven:

```
1 $ conda create --name dash python=3.12
```

- ② Környezet aktiválása:

```
1 $ conda activate dash
```

- ③ Órai tárhely klónozása:

```
1 $ git clone https://github.com/basictask/Adatbanyaszat.git
```

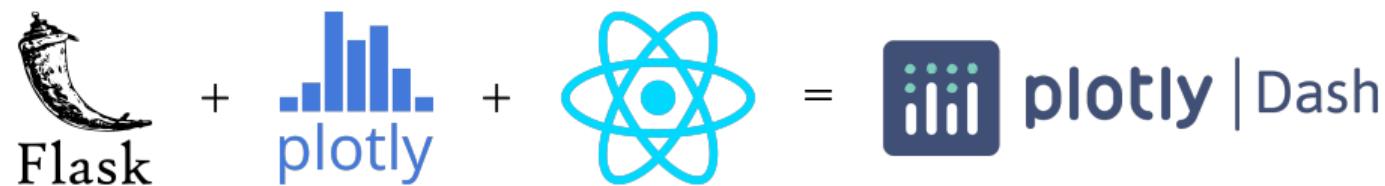
- ④ Függőségek telepítése:

```
1 $ cd Adatbanyaszat  
2 $ pip install -r requirements.txt
```

A Dash keretrendszer

A Dash keretrendszer segítségével lehetséges interaktív, dinamikus adatalapú műszerfalakat és alkalmazásokat készíteni szintisztán Python nyelvben.

A Dash a **Flask** mikrokeretrendszert használja backend szerverként, **Plotly** segítségével jelenítui meg a diagramokat és **React** komponenseket használ a felhasználói interakció kezelésére.



A Dash könyvtár komponensei

Dash

Ez a fő csomag, amely bármely alkalmazás gerincét biztosítja a `dash.Dash` objektumon keresztül.

Emellett néhány más eszközt is biztosít az interaktivitás és kivételek kezeléséhez, amelyekről később fogunk beszélni, amikor építjük az alkalmazásunkat.

A Dash könyvtár komponensei

Dash Core Components

Egy csomag, amely interaktív komponensek készletét biztosítja, amelyeket a felhasználók manipulálhatnak.

Legördülő menük, dátumválasztók, csúszkák és sok más komponens is megtalálható ebben a csomagban.

A Dash könyvtár komponensei

Dash HTML Components

Ez a csomag az összes elérhető HTML címkét Python osztályként biztosítja.

Egyszerűen átalakítja a Pythont HTML-re.

Például, Pythonban a `dash_html_components.H1('Hello, World')` kód átalakul `<h1>Hello, World</h1>` HTML kóddá.

A Dash könyvtár komponensei

Dash Bootstrap Components

Ez egy harmadik féltől származó csomag, amely Bootstrap funkcionalitást ad a Dash-hez. Ez a csomag és annak komponensei számos elrendezéssel és vizuális jelekkel kapcsolatos lehetőséget kezelnek.

Az elemek egymás mellé vagy egymás fölé helyezése, méretük meghatározása a böngésző képernyőmérete alapján, valamint kódolt színek készletének biztosítása a jobb kommunikáció érdekében a felhasználókkal.

Egy Dash alkalmazás struktúrája

- Importálások:

```
1 import dash
2 import dash_html_components as
3     html
3 import dash_core_components as dcc
```

- Elrendezés:

```
1 app.layout = html.Div([
2     dcc.Dropdown()
3     dcc.Graph()
4     ...
5 ])
```

- Visszahívási függvények:

```
1 @app.callback()
2 ...
3 @app.callback()
4 ...
```

- Alkalmazás példányosítása:

```
1 app = dash.Dash(__name__)
```

- Alkalmazás futtatása:

```
1 if __name__ == '__main__':
2     app.run_server()
```

Egy kezdeti alkalmazás

- ① Egy új, app.py fájlban a következő csomagok importálásával:

```
1 import dash
2 import dash_core_components as dcc
```

- ② Alkalmazás példányosítása:

```
1 app = dash.Dash(__name__)
```

- ③ Alkalmazás elrendezésének létrehozása:

```
1 app.layout = html.Div([
2     html.H1('Hello, World!')
3 ])
```

- ④ Futtatás:

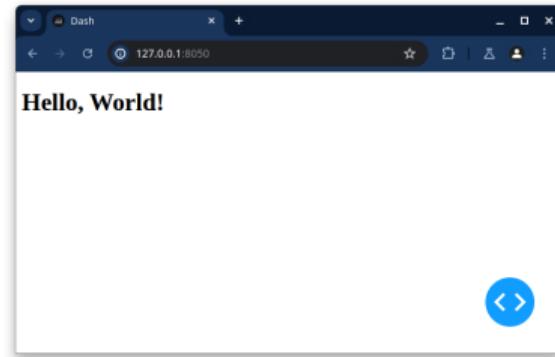
```
1 if __name__ == '__main__':
2     app.run_server(debug=True)
```

Az alkalmazás futtatása (app_v1_1.py)

A Python értelmező segítségével a megfelelő könyvtárban állva az app_v1_1.py fájl futtatásával az eredmény a következő:

```
1 (dash) daniel@neptune:~/Documents/BGE/  
      Adatbanyaszat/2_dash/code$ python  
      app_v1_1.py  
2 Dash is running on http  
      ://127.0.0.1:8050/  
3  
4 * Serving Flask app 'app_v1_1'  
5 * Debug mode: on
```

A böngészőben a 127.0.0.1:8050 címre navigálva a következő eredmény látható:



Komponensek hozzáadása az alkalmazáshoz

Komponensek hozzáadása az alkalmazás elrendezésének szerkesztésével érhető el (app.layout). Ez meglehetősen egyszerű, csak a legfelső szintű html.Div komponens children attribútumához kell hozzáfűzni a megfelelő elemeket.

```
1 html.Div(children=[component_1, component_2, component_3, ...])
```

Div

A Div a HTML-ben egy blokk szintű elem, amely képes egy dokumentum különböző komponenseit csoportosítani. A Div nem rendelkezik semmilyen alapértelmezett stílussal vagy viselkedéssel.

HTML komponensek hozzáadása

children

Ez az első, és a fő konténere a komponenseknek. Paraméterül kaphatja elemek listáját vagy egyetlen elemet is.

className

Ez megegyezik a HTML class attribútumával.

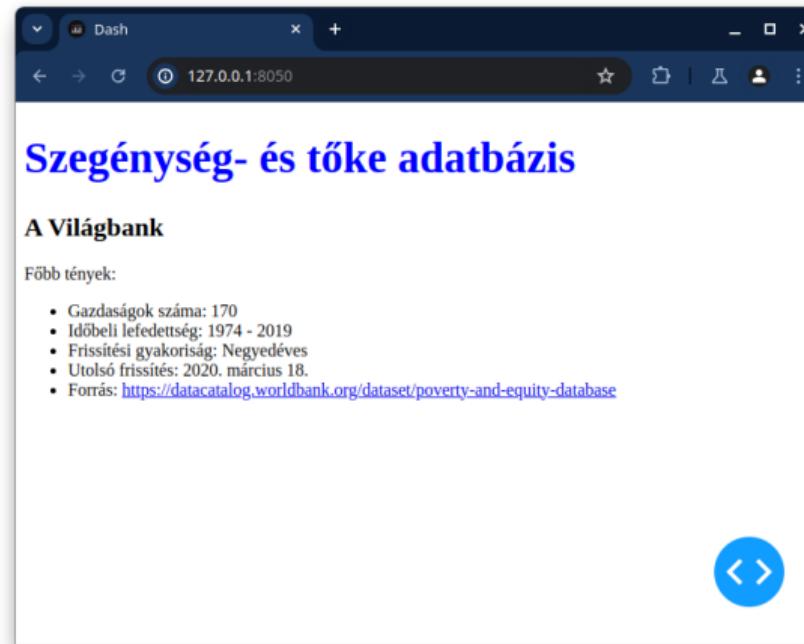
id

A komponens azonosítója. Az interaktivitás megvalósításában van kulcsfontosságú szerepe

style

Ez megfelel az azonos nevű HTML attribútumnak azzal a különbséggel, hogy camelCase stílust használ a változók elnevezésére.

A Dash alkalmazás HTML komponensekkel (app_v1_2.py)

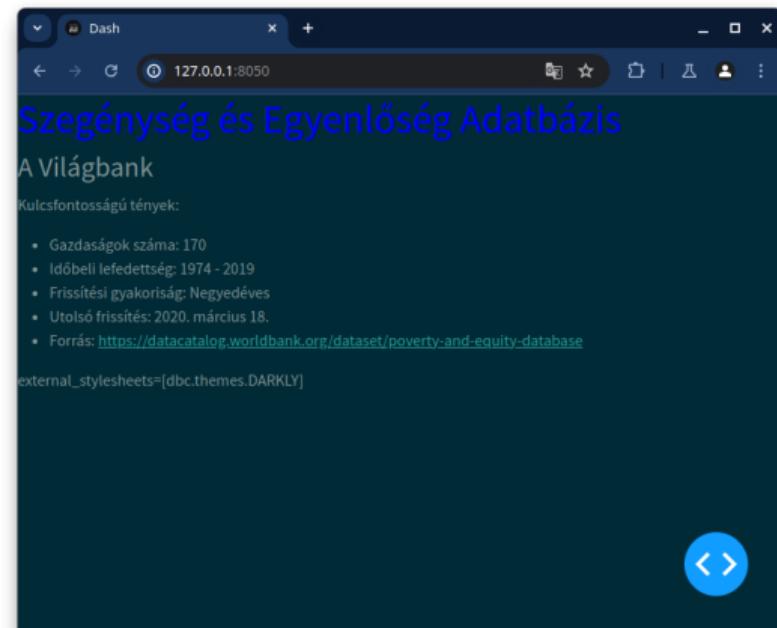
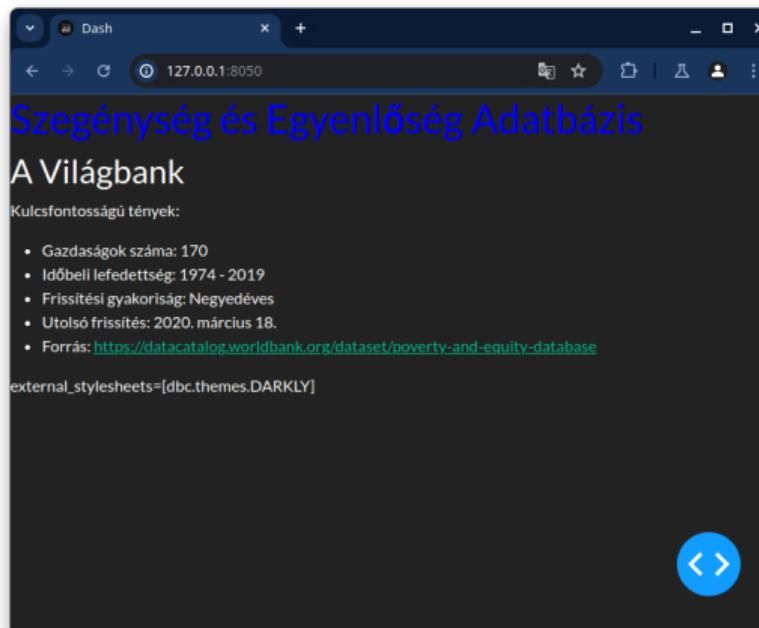


Témák

Egy Dash alkalmazás témájának megváltoztatása rendkívül egyszerű: a Dash objektum létrehozásakor kell egy új téma argumentumot bevinni a konstruktur függvénybe.

```
1 import dash_bootstrap_components as dbc  
2 ...  
3 app = dash.Dash(__name__ , external_stylesheets=[dbc.themes.BOOTSTRAP])  
4 ...
```

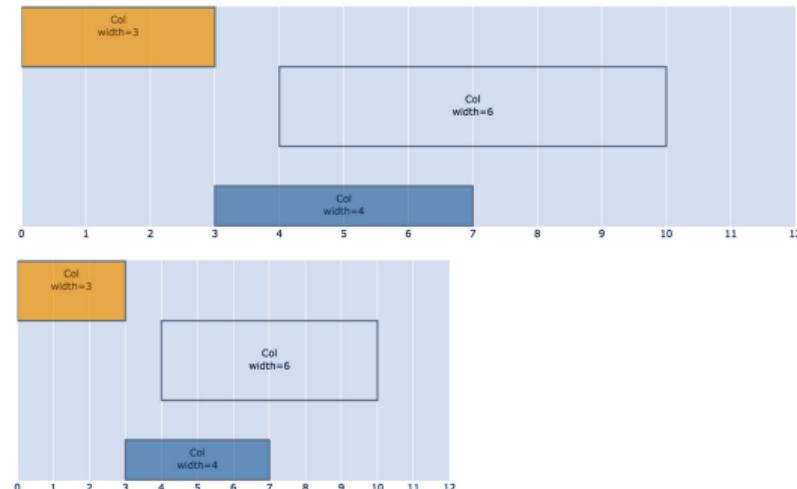
Témák az előző alkalmazásban (app_v1_3.py)



A rács rendszer

A Bootstrap segítségével lehetséges oszlopokat definiálni, ami egy független képernyőként viselkedik, egymás fölött megjelenítve az elemeket.

A rács rendszer 12 oszlopra bontja a képernyőt, és egy komponens szélessége oszlopok számában adható meg.

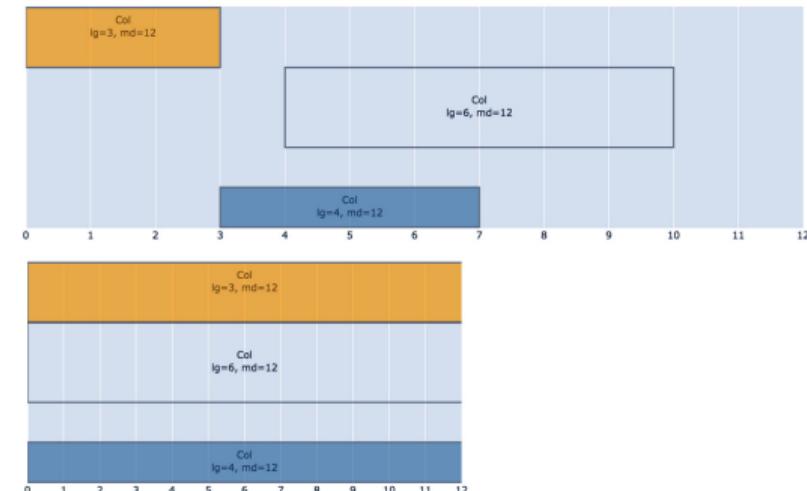


```
1 import dash_bootstrap_components as dbc  
2 dbc.Col(children=[child1, child2, ...])
```

Rácsok dinamikus képernyő méreten

Vannak olyan esetek, amikor nem kívánatos az elemek méretezése a képernyővel együtt. Amikor a képernyő kisebb lesz, némelyik komponenseknek jó, ha kiterjednek méretükben.

Öt különböző méretet lehet definiálni: xs (extra-small), md (medium), lg (large), xl (extra-large).

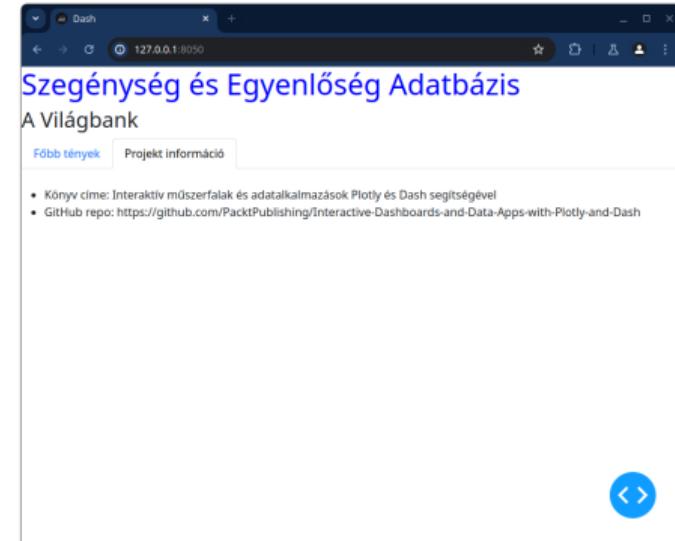


```
1 import dash_bootstrap_components as dbc  
2 dbc.Col(children=[child1, child2, ...], lg=6, md=12)
```

Bootstrap komponensek hozzáadása az alkalmazáshoz (app_v1_4.py)

Az alkalmazás következő verziójában két új komponens kerül hozzáadásra, a Tabs és Tab. Ezek szorosan kapcsolódnak egymáshoz. A Tabs a Tab konténere.

Ennek eredménye egy informatívabb és jobban elrendezett alkalmazás.



Dash alkalmazások Jupyter Notebookban

Kevés programkód változtatással az alkalmazás Jupyter Notebook környezetben is futtathatóvá válik. Ezt a `jupyter_dash` csomag teszi lehetővé.

A használatához a Dash helyett a `JupyterDash` csomagot kell importálni, és ennek mentén kell példányosítani az alkalmazást:

```
1 from jupyter_dash import JupyterDash  
2 app = JupyterDash(__name__)
```

A `JupyterDash` három módot biztosít az alkalmazás futtatására:

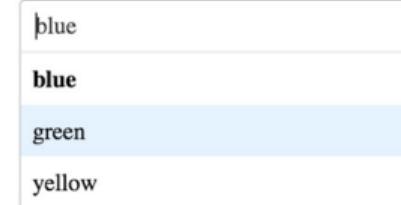
- `external`: Külön böngésző ablakban
- `inline`: A kód output helyen a notebookban
- `jupyterlab`: Külön böngészőfülben (csak JupyterLab szerveren)

Komponensek id paramétere

Az id paraméter elengedhetetlen a Dash alkalmazások interaktivitásához.

Ez egy, a komponensekhez rendelt egyedi azonosító, amelynek segítségével az alkalmazás megkülönbözteti és kezeli a különböző vezérlőelemeket, például grafikonokat vagy szövegdobozokat.

`id='color_dropdown'`



```
1 app.layout = html.Div([
2     dcc.Dropdown(
3         id='color_dropdown',
4         options=[{'label': x, 'value': x} for x in ['blue', 'green', 'yellow']]
5     )
6 ])
```

Callback függvények

A callback egy Dash alkalmazásban egy olyan függvény, amely akkor hívódik meg, amikor egy adott esemény bekövetkezik, például egy felhasználói interakció.

Így dinamikusan frissíthetők az alkalmazás komponensei. A következők szükségesek egy callback függvényhez:

- Output: Az a komponens attribútum, amelyik meg fog változni a függvény hatására.
- Input: Az az alkalmazás elem vagy esemény, amelyik elindítja a függvényt.
- A függvény fejléc és definíció.

Back end

Input('color_dropdown', 'value')

```
def display_selected_color(color):
    if color is None:
        color = 'nothing'
    return 'You selected ' + color
```

Output('color_output', 'children')

Front end

`id='color_dropdown'`

blue
blue
green
yellow

`id='color_output'`

You selected <color>

Callback függvény implementálása

A callback működés egy dekorátor segítségével valósítható meg a függvény fejléce fölött. Itt definiálni kell az Input és Output komponenseket, ilyen sorrendben.

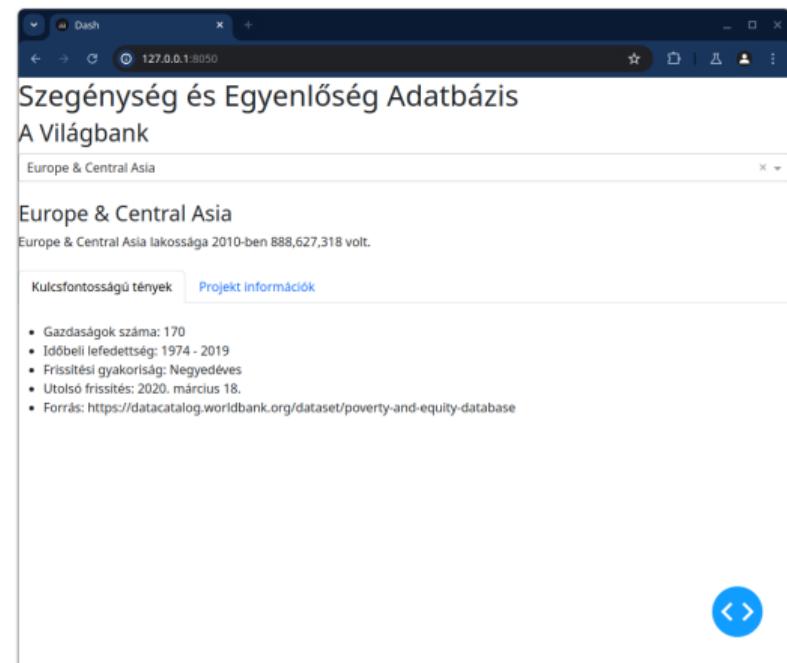
Egy callback függvénynek több inputja és outputja lehet.

```
1 @app.callback(
2     Output('color_output', 'children'),
3     Input('color_dropdown', 'value')
4 )
5 def display_selected_color(color):
6     if color is None:
7         color = 'nothing'
8     return 'You selected ' + color
```

Callback függvény implementálása az alkalmazásba (app_v1_5.py)

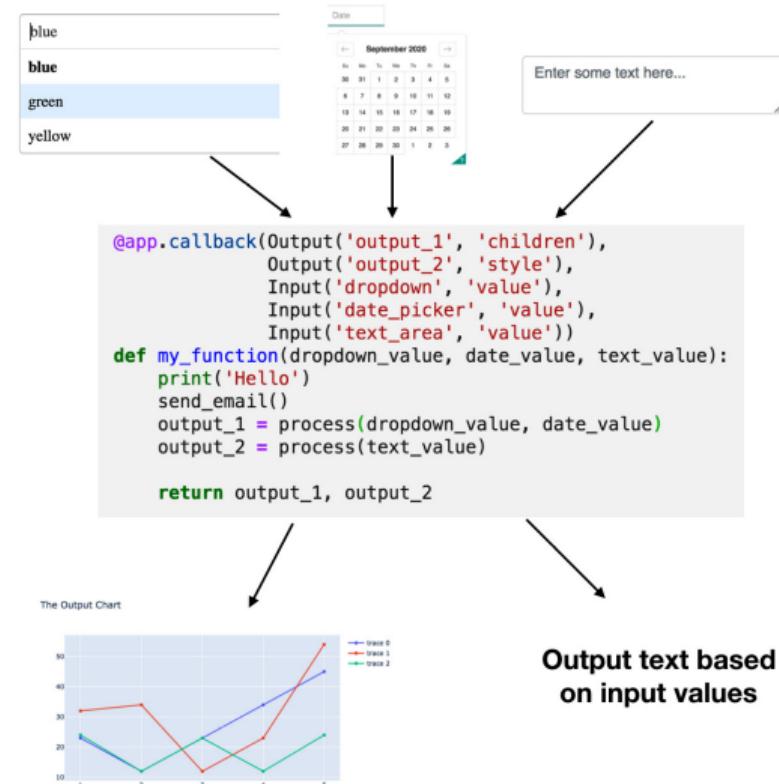
Egy callback implementálásának lépései:

- ① Új lenyíló lista létrehozása egy adathalmazban megtalálható országok segítségével
- ② Egy új callback függvény létrehozása amely megkapja a választott országot, leszűri az adathalmazt majd megtalálja az ország népességi adatait (az összes forrásfájl a data mappában található).
- ③ Egy riport készítése a megtalált adatokról.



Callback függvények tulajdonságai

- A visszatérés előtt szinte bármilyen műveletet elvégezhetnek, mint pl. egy gépi tanulás modell tanítása
- A callback függvények harmadik attribútuma a (State). Az állapottal definiált objektum attribútumok nem indítják el a callback függvényt, de a futás során a függvény hozzáfér az értékükhez.
- A callback dekorátor definíciós sorrendje: [Output, Input, State].
- Az input és állapot sorrend a callback dekorátorban meg kell feleljen a paraméterek sorrendjének..



1 Dash alapok

2 Diagramok

3 Pontdiagramok

4 Gyakorisági adatok

A Figure objektum

A Plotly-ban a Figure objektum egy magasszintű adatstruktúra, amely egyesíti a diagram adatait és elrendezését egyetlen objektumban. Olyan attribútumokat tartalmaz, mint a data és layout.

```
1 import plotly.graph_objects as go  
2  
3 fig = go.Figure()
```

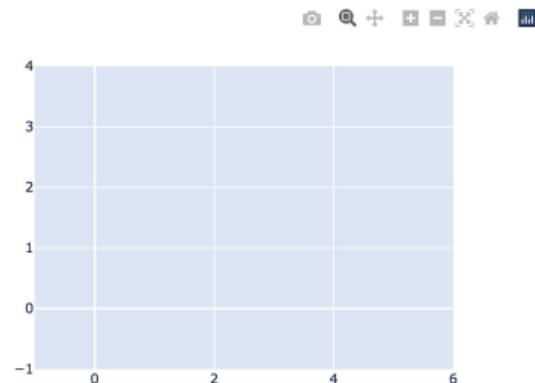


Figure objektumok attribútumai

data

Ez az attribútum tartalmazza a diagramon megjelenítendő adatokat. Ez egy lista, amely különböző nyomokat (trace) tartalmaz.

trace

Egy trace vagy nyom egy adatcsoportot képvisel a diagramon belül. minden nyomnak megvan a maga típusa (pl. kör, szórás, oszlop) és különböző tulajdonságokkal rendelkezik, mint az x és y tengely értékei, színek, név stb...

layout

Ez az attribútum határozza meg a diagram elrendezését és stílusát. Ide tartoznak a tengelyek címkei, a diagram címe, a háttérszínek, a margók, a legendák és egyéb vizuális elemek. A layout attribútum egy szótár, amely különböző kulcs-érték párokat tartalmaz.

A data attribútum

A Figure objektumot a `plotly.graph_objects` könyvtár tartalmazza. Példányosítás után az `add_scatter()` függvény meghívásával hozzáadódik egy új nyom a vászonhoz. A nyom az *x* és *y* koordinátákat tartalmazza a pontdiagramhoz.

```
1 import plotly.graph_objects as go
2
3 fig = go.Figure()
4 fig.add_scatter(x=[1, 2, 3], y=[4, 2, 3])
5 fig.show()
```

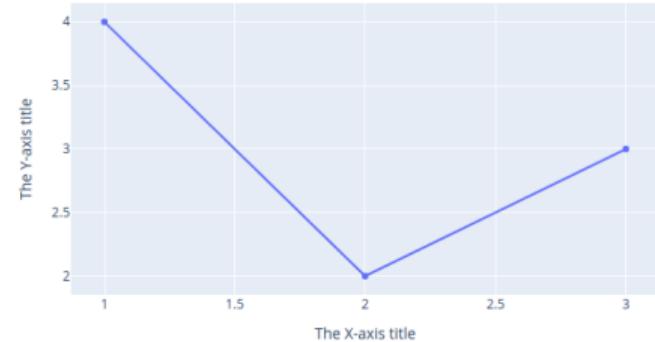


A layout attribútum

A layout attribútum módosítása közvetlen hozzáféréssel lehetséges. Ez egy fastruktúrájú adatszerkezet, ahol az attribútumoknak alárendelt attribútumai vannak.

```
1 fig.layout.title = 'The Figure Title'  
2 fig.layout.xaxis.title = 'The X-axis  
    title'  
3 fig.layout.yaxis.title = 'The Y-axis  
    title'
```

The Figure Title



Grafikon renderelése json formátumba

A `show()` függvény több módot is biztosít egy függvény megjelenítésére. Az egyik ilyen a `fig.show('json')`.

Ennek segítségével meg lehet vizsgálni a diagram renderelése közben létrejövő fa struktúrát.

```
1 fig.show('json')  
└ root:  
  └ data: [] 2 items  
  └ layout:
```

```
1 fig.show('json')  
└ root:  
  └ data: [] 2 items  
  └ layout:  
    └ template:  
      └ title:  
        └ text: "The Figure Title"  
    └ xaxis:  
      └ title:  
        └ text: "The X-axis title"  
    └ yaxis:  
      └ title:  
        └ text: "The Y-axis title"
```

```
1 fig.show('json')  
└ root:  
  └ data: [] 2 items  
  └ 0:  
    └ type: "scatter"  
    └ x: [] 3 items  
      0: 1  
      1: 2  
      2: 3  
    └ y: [] 3 items  
      0: 1  
    └ layout:  
      └ template:  
      └ title:  
        └ text: "The Figure Title"  
    └ xaxis:  
    └ yaxis:
```

Grafikon konfigurálása

A config paraméter egy dict objektumot vár el, és több tulajdonságát is képes vezérelni a diagramnak:

```
1 fig.show(  
2     config={  
3         'displaylogo': False,  
4         'modeBarButtonsToAdd': ['drawrect',  
5             'drawcircle', 'eraseshape']  
6     }  
7 )
```

- **displayModeBar**: A teljes menüsáv mutatása, alapértelmezése True
- **responsive**: Változzon-e a diagram mérete a böngésző méretnek megfelelően. Alapértelmezése a True.
- **toImageButtonOptions**: A kép letöltésének alapértelmezett formátumát adja meg.
- **modeBarButtonsToRemove**: Azon menügombok listája, amiket ne jelenítsen meg a diagram.

Vezérlőkkel összekapcsolt diagramok

Egy interaktív diagram létrehozása az elrendezésben:

```
1 app.layout = html.Div([
2     dcc.Dropdown(
3         id='year_dropdown',
4         value='2010',
5         options=[{'label': year, 'value': str(year)} for year in range
6             (1974, 2019)])
7     ),
8     dcc.Graph(id='population_chart')
9 ])
```

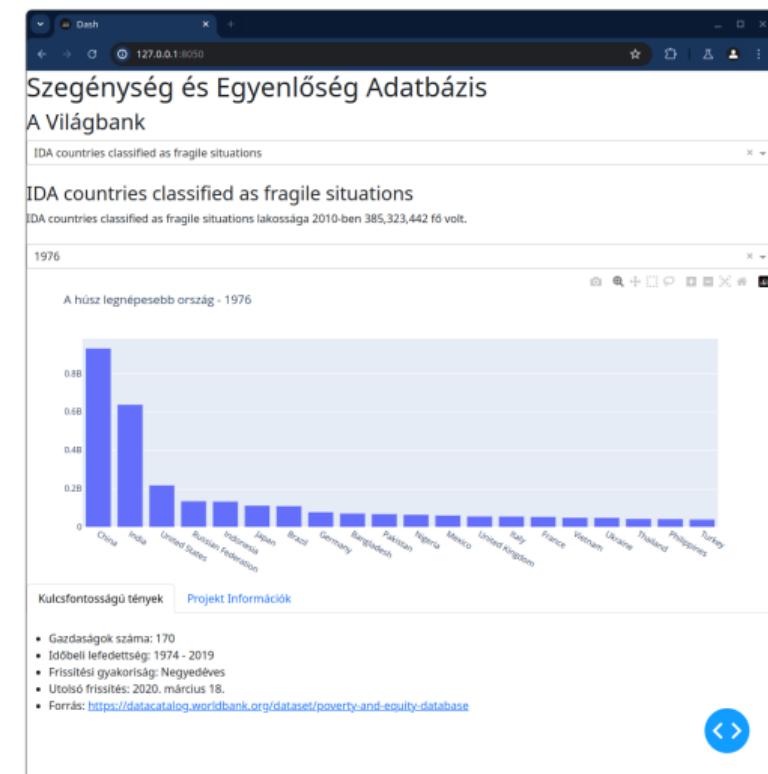
A Dropdown komponensben definiálva vannak a lehetséges értékek, és a Graph egy egyedi id adattaggal van ellátva az interaktivitás miatt.

Az ehhez a diagramhoz tartozó callback függvény a legördülő menü értékét kapja meg paraméterül, az implementációjában leszűri a táblát, majd egy Figure objektumot térít vissza, ami felülírja a population_chart figure attribútumát:

```
1 @app.callback(
2     Output('population_chart', 'figure'),
3     Input('year_dropdown', 'value')
4 )
5 def plot_countries_by_population(year):
6     year_df = ...
7     fig = go.Figure()
8     ...
9     return fig
```

Alkalmazás felépítése interaktív diagrammal (app_v2_1.py)

- 1 Pandas importálása, és a szegénységi adatokat tartalmazó fájl megnyitása.
- 2 Régiók kizárása az adatkészletből, hogy csak országok maradjanak.
- 3 Egy DataFrame létrehozása, amely csak az országok teljes népességét tartalmazza.
- 4 Egy legördülő menü segítségével ki lehet választani az évet, amely leszűri az adathalmazt, hogy az abból az évből legnépesebb országokat reprezentálja.
- 5 Oszlopdiagram létrehozása, amely a legnépesebb országokat tartalmazza.



Dash vizuális debugger

A vizuális debugger a Dash-ben egy eszköz, amely segít a fejlesztőknek nyomon követni és hibakeresni a Dash alkalmazásaiat. A vizuális debugger lehetővé teszi, hogy valós időben lássuk az alkalmazás állapotát, a komponensek közötti adatáramlást és az eseményeket:

- Komponensek hierarchiája
- Állapot és tulajdonságok
- Események és változások
- Hibák és figyelmeztetések

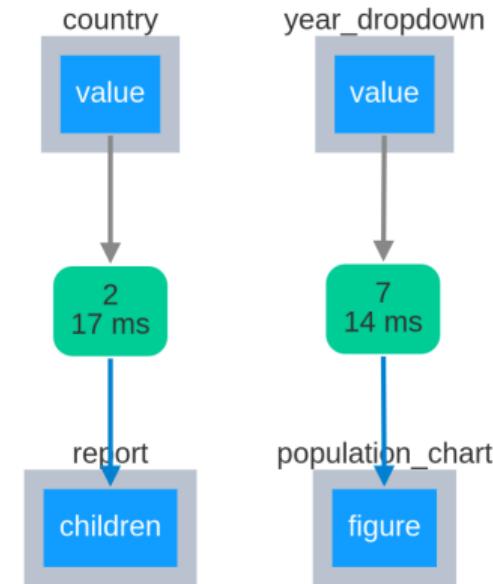
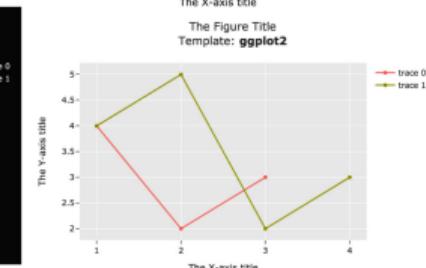
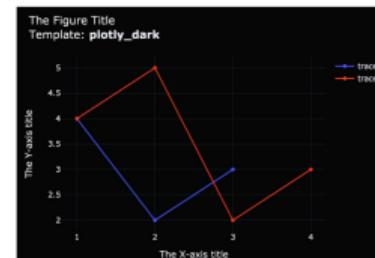
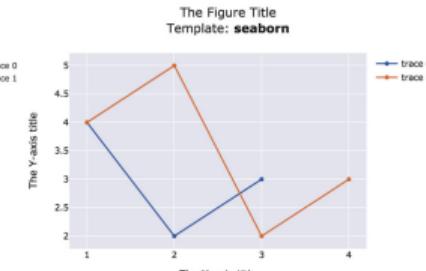
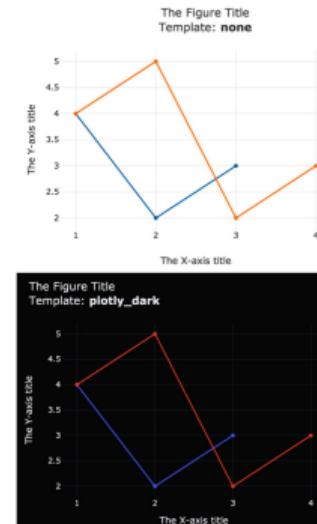


Diagram témák szerkesztése

A diagramok témájának szerkesztése nagyon sok időt megspórolhat, és egy általános megoldást nyújt arra, hogy minden diagram témáját egyszerre lehessen változtatni.

Ez a layout alatt a template attribútum módosításával érhető el.

```
1 fig.layout.template = template_name
```



Példa adathalmaz definiálása

A következőkben a következő egyszerű adattáblával készült diagramok lesznek láthatók:

```

1 df = pd.DataFrame({
2     'numbers': [1, 2, 3, 4, 5, 6,
3                 7, 8],
4     'colors': ['blue', 'green', ,
5                 'orange', 'yellow', 'black
6                 , 'gray', 'pink', 'white
7                 ],
8     'floats': [1.1, 1.2, 1.3,
9                 2.4, 2.1, 5.6, 6.2, 5.3],
10    'shapes': ['rectangle', ,
11                 'circle', 'triangle', ,
12                 'rectangle', 'circle', ,
13                 'triangle', 'rectangle', ,
14                 'circle'],
15    'letters': list('AAABBCCC')
16 })
17 
```

	numbers	colors	floats	shapes	letters
1	0	1	1.1	rectangle	A
2	1	2	1.2	circle	A
3	2	3	1.3	triangle	A
4	3	4	2.4	rectangle	B
5	4	5	2.1	circle	B
6	5	6	5.6	triangle	C
7	6	7	6.2	rectangle	C
8	7	8	5.3	circle	C

Pontdiagram

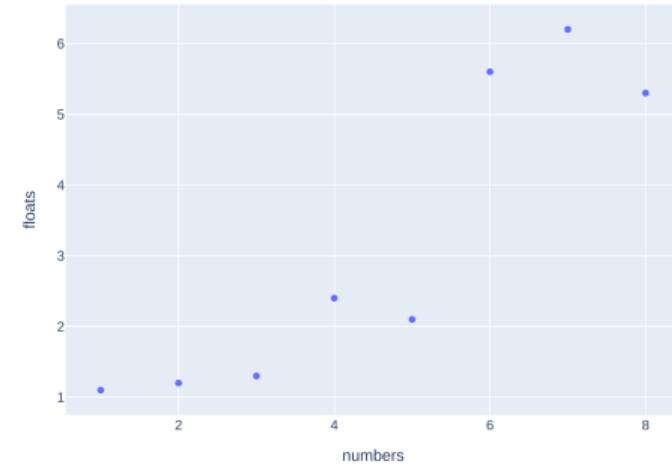
Egy `plotly express` diagramnak kétféleképpen is át lehet adni az adathalmazt.

Az első esetben a `DataFrame` kerül átadásra, és az `x`, `y` paraméterek a `DataFrame` oszlopaira hivatkoznak:

```
1 px.scatter(data_frame=df, x='numbers',  
             y='floats')
```

A másik esetben pedig közvetlenül vannak hivatkozva az oszlopok:

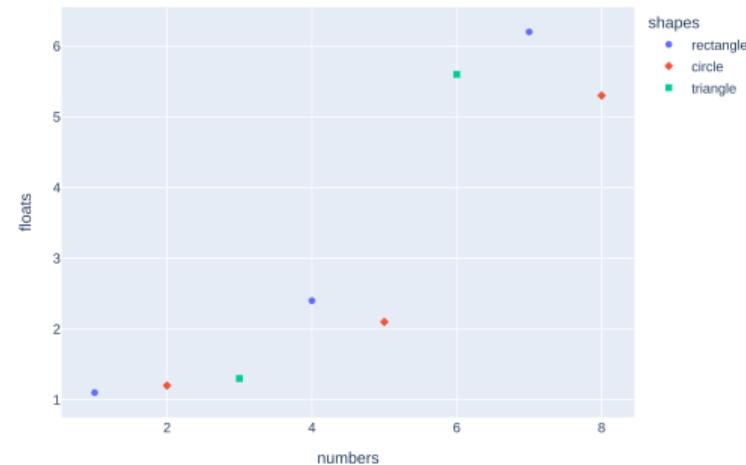
```
1 px.scatter(x=df['numbers'], y=df['  
floats'])
```



Pontdiagram kategóriákkal

Ebben az esetben minden adatosztály egy külön nyomként jelenik meg.

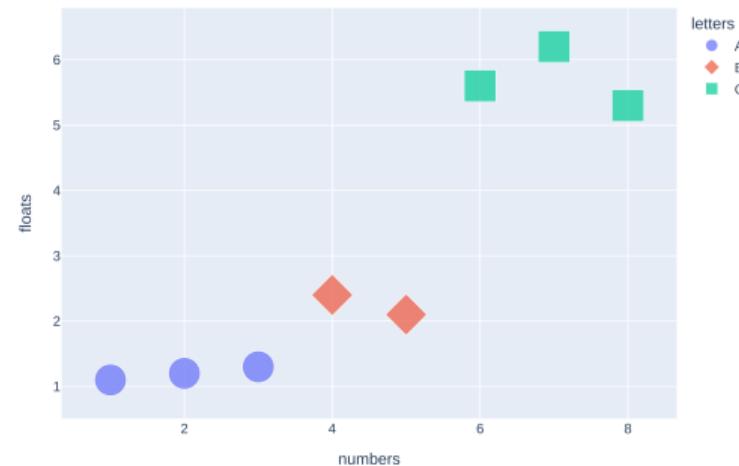
```
1 px.scatter(df, x='numbers', y='floats',
             color='shapes', symbol='shapes')
```



Pontdiagram jelölőkkel

Minden (x, y) adatpont jelölőjét lehetséges külön állítani. Ezeknek testre lehet szabni a színét, méretét:

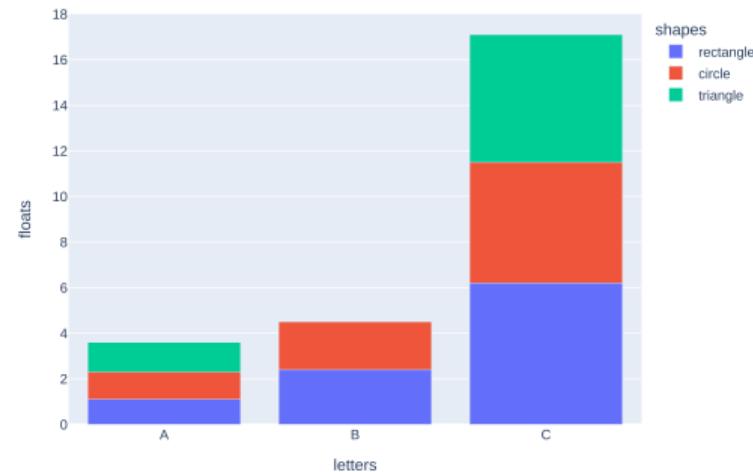
```
1 px.scatter(df, x='numbers', y='floats',
    color='letters', symbol='letters',
    size=[35] * 8)
```



Rakott oszlopdiagram

A rakott oszlopdiagram több oszlopdiagram együttese. Ebben az esetben is minden adatcsoport egy külön nyomként jelenik meg az adatszerkezetben. A csoportosítási változót a `color` attribútum adja meg.

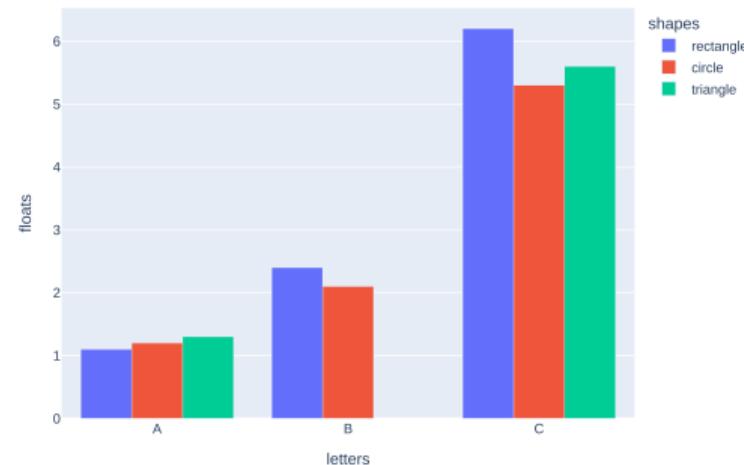
```
1 px.bar(df, x='letters', y='floats',  
        color='shapes')
```



Csoportosított oszlopdiagram

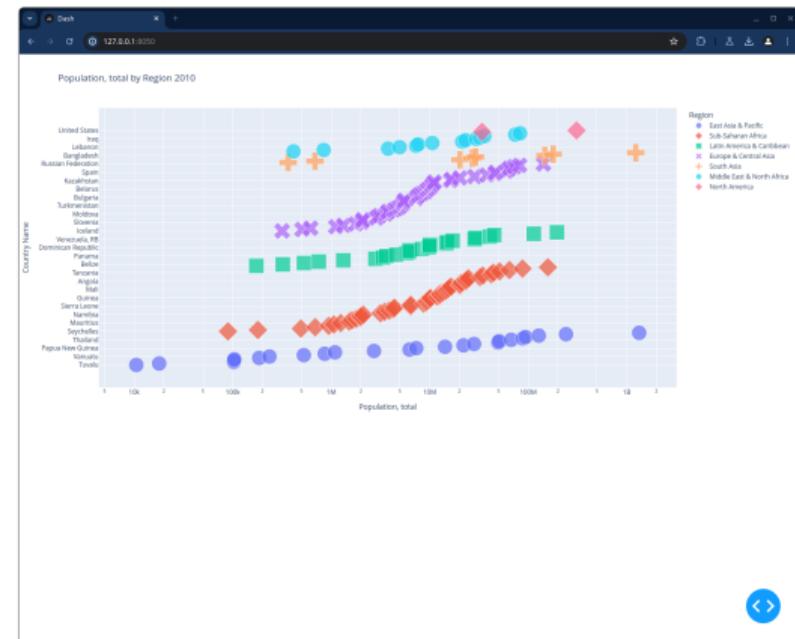
Csoportosítás esetén az adatcsoportok nem egymáson, hanem egymás mellett foglalnak helyet. A csoportosítás attribútuma itt is a `color`, és a csoportosítási típust a `barmode=color` adja meg.

```
1 px.bar(df, x='letters', y='floats',
      color='shapes', barmode='group')
```



Összetett plotly express diagram létrehozása (px_app.py)

- 1 Adathalmazok beolvasása, transzformációja és megfelelő formára hozása
- 2 Változók létrehozása, amik megadják a szűrési kritériumokat: year, indicator, grouper
- 3 Adathalmaz leszűrése a változók alapján
- 4 A px.scatter() meghívása egy Dash objektum Div komponensén



Feketetett oszlopdiagram

Vannak olyan esetek, amikor a tengelycímek

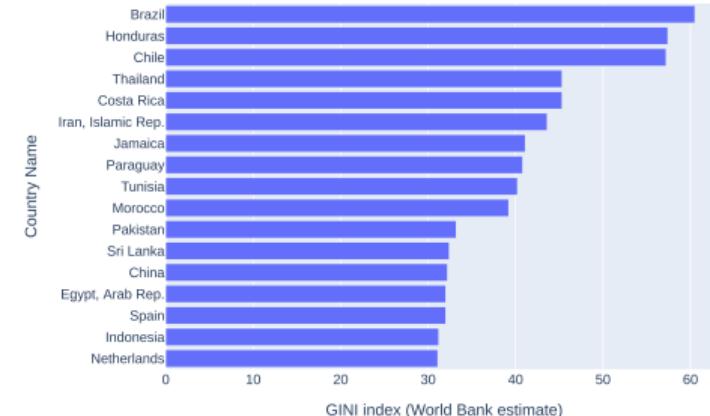
hosszúak, ezért fontos a jó olvashatóság.

Ebben az esetben a vízszintes
oszlopdiagram a megfelelő választás.

Ehhez az x és y paramétereket meg kell
cserélni és az orientation='h'
paramétert be kell állítani:

```
1 fig = px.bar(  
2     df,  
3     x=gini,  
4     y='Country Name',  
5     title=' - '.join([gini, str(year)]),  
6     orientation='h'  
7 )
```

GINI index (World Bank estimate) - 2000



Dinamikus méretű oszlopdiagram

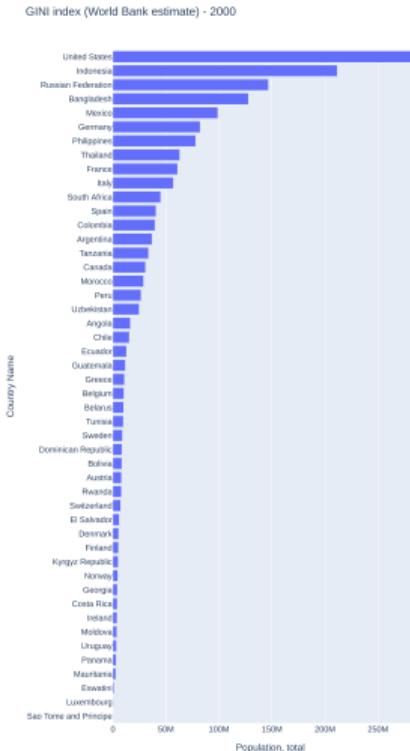
Alapértelmezés szerint a plotly express diagramok maximális méretének intervalluma [20.2, 65.8]. Ezt manuálisan is lehet állítani, de amikor kevés rekord van a diagram eltorzulhat.

Egy megoldás erre, ha országunként 20 pixellel nő meg a height paraméter.

```

1 fig = px.bar(
2   df,
3   x=indicator,
4   y='Country Name',
5   title=' - '.join([gini, str(year)]),
6   height=200 + (20 * n_countries),
7   orientation='h',
8 )

```



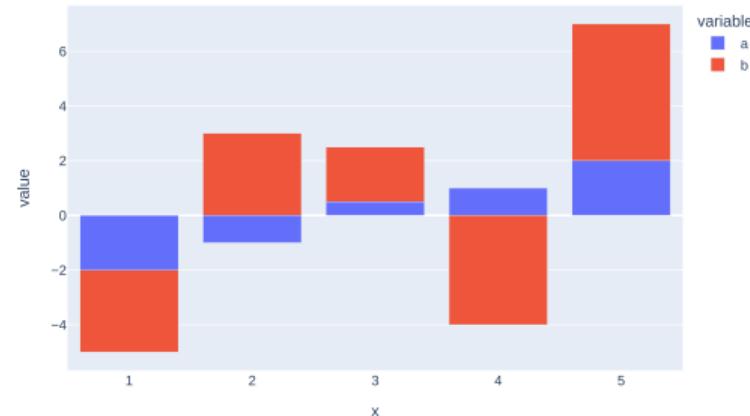
Oszlopmodok

Ugyanazt az oszlopdiagramot többféleképpen is meg lehet jeleníteni. Ennek a felelőse a barmode paraméter.

relative

Az oszlopok egymás mellett jelennek meg, és az értékek relatív különbségeit mutatják.

barmode=relative



Oszlopmodok

Ugyanazt az oszlopdiagramot többféleképpen is meg lehet jeleníteni. Ennek a felelőse a barmode paraméter.

group

Az oszlopok csoportosítva jelennek meg, különböző kategóriák szerint.

barmode=stack



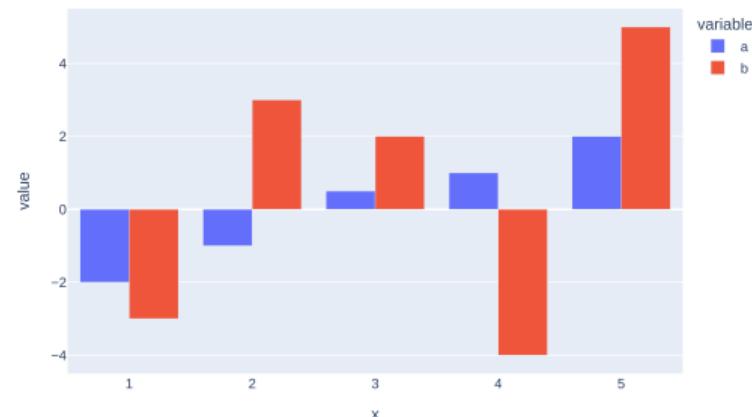
Oszlopmodok

Ugyanazt az oszlopdiagramot többféleképpen is meg lehet jeleníteni. Ennek a felelőse a `barmode` paraméter.

overlay

Az oszlopok egymásra helyezve jelennek meg, átfedésben.

`barmode=group`



Oszlopmodok

Ugyanazt az oszlopdiagramot többféleképpen is meg lehet jeleníteni. Ennek a felelőse a barmode paraméter.

stack

Az oszlopok egymásra rakva jelennek meg, az értékek összeadódnak.

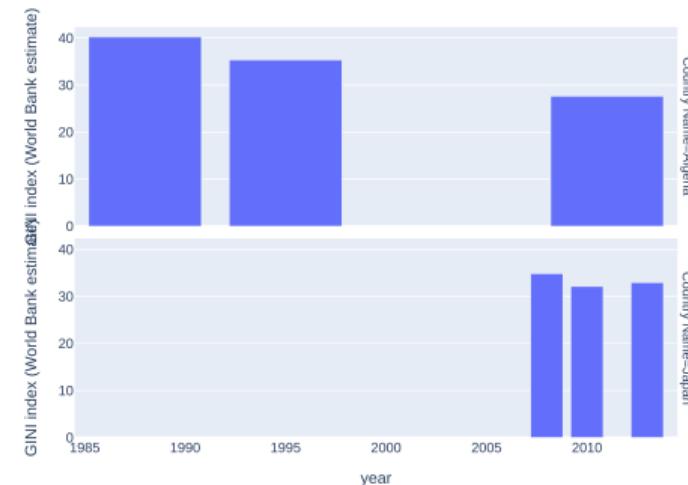
barmode=overlay



Diagramok szétbontása felületekre

Szétbontással új dimenziókat lehet felvenni egy műszerfalra. Ki lehet választani egy jellemzőt, ami mentén a szeletelés történik. A megfelelő paraméterek erre a plotly express könyvtárban a facet_col és facet_row attól függően, hogy új oszlop vagy sor fog létrejönni a diagramban.

```
1 fig = px.bar(df, x='year', y=gini,  
               facet_row='Country Name')
```



Legnépesebb országok régiónként

Egy legördülő listán ki lehet választani az évek közül a megfelelőt, ez elindít egy callback függvényt, ami egy dcc.Graph objektum figure adattagját frissíti egy plotly express diagrammal.

```
1  dcc.Dropdown(  
2      id='year_dropdown',  
3      value='2010',  
4      options=[{'label': year, 'value': str  
           (year)} for year in range(1974,  
2019)]  
5 ),  
6 ...  
7  dcc.Graph(id='population_chart'),
```

```
1 @param_app.callback(  
2     Output('population_chart', 'figure'),  
3     Input('year_dropdown', 'value'))  
4 )  
5 def plot_countries_by_population(year):  
6     fig = go.Figure()  
7     year_df = population_df[['Country  
        Name', year]].sort_values(year,  
        ascending=False)[:20]  
8     fig.add_bar(x=year_df['Country Name'  
        ], y=year_df[year])  
9     fig.layout.title = f'A húsz legné  
        pesebb ország - {year}'  
10    return fig
```

Többváltozós legördülő lista

A Dropdown komponensnek van egy extra, opcionális paramétere, a multi, ami egy logikai változót vár el értékként, és ha be van kapcsolva lehetővé teszi több érték kiválasztását egy változóból.

```
1dcc.Dropdown(  
2    id='gini_country_dropdown',  
3    multi=True,  
4    options=[{'label': country, 'value':  
5        country} for country in gini_df['  
Country Name'].unique()])
```



Helykitöltő szöveg legördülő listához

Egy további opcionális paramétere a Dropdown objektumoknak a placeholder, amivel lehetséges helykitöltő szöveget hozzáadni a legördülő listához.

Ezzel megjelenít egy szöveget, ami azelőtt látszik, hogy a felhasználó belekattintana dobozba.

```
1dcc.Dropdown(  
2    id='gini_country_dropdown',  
3    placeholder='Válasszon egy vagy több  
4        országot',  
5    multi=True,  
6    options=[{'label': country, 'value':  
7        country} for country in gini_df['  
8            Country Name'].unique()]  
9 ),
```

Országok

Válasszon egy vagy több országot

Elrendezés komponensek újrakezelése

Egy python szkriptben inicializált layout változót lehetséges egy másik szkriptben importálni.

Ebben az esetben a layout.children komponenst egy listaként kell átvenni, és hozzá kell fűzni az aktuális szkript elrendezés definíójához.

```
1 app.layout = html.Div([
2     *app_v2_3.app.layout.children[:-1],
3     dbc.Row([
4         ...
5     ]),
6     app_v2_3.app.layout.children[-1]
7 ])
```

A lista kicsomagolás operátor

Az előző példában a listára értelmezett * operátor a python nyelvben arra használható, hogy egy lista értékeit kicsomagolja valamelyen argumentumban vagy másik adatstruktúrában.

```
1 In [1]: a = [1, 2, 3]
2 In [2]: b = 5
3 In [3]: [*a, 4, b]
4 Out[3]: [1, 2, 3, 4, 5]
```

Ez a funkcionalitás akkor használatos, ha arra van szükség, hogy egy függvény tetszőleges számú paramétert legyen képes átvenni:

```
1 def f(*argv):
2     ...
3
4 f('Hello', 'World')
```

Abban az esetben ha nevesített argumentumokra van szükség a ** operátor teszi ezt lehetővé:

```
1 def f(**kwargs):
2     ...
3
4 f(school='bge', spec='data')
```

Callback függvények újrafelhasználása

Callback függvényeket is lehetséges felhasználni szkriptek között, viszont ennek az eljárása eltér az elrendezés komponensek újrafelhasználásának módjától.

Ahhoz, hogy egy alkalmazásba be lehessen importálni egy másik alkalmazás callback függvényeit, a definíció helyén egy külső függvénybe kell ezeket beágyazni, majd ezt a függvényt később meghívni egy paraméterezett Dash alkalmazás objektummal.

```
1 def register_callbacks(param_app):
2     @param_app.callback(
3         ...
4     )
5     def callback1(...):
6         ...
7
8     @param_app.callback(
9         ...
10    )
11    def callback2(...):
12        ...
```

A meghívás egy másik alkalmazásból:

```
1 import app_v2_1
2
3 app = dash.Dash(__name__)
4 ...
5 app_v2_1.register_callbacks(app)
```

1 Dash alapok

2 Diagramok

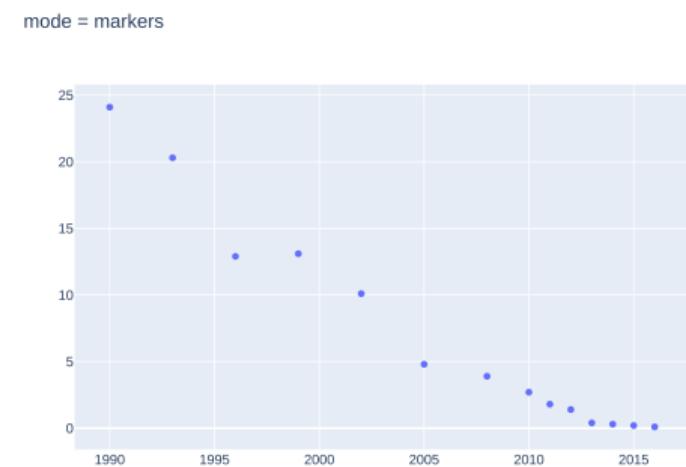
3 Pontdiagramok

4 Gyakorisági adatok

Alapvető pontdiagramok

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

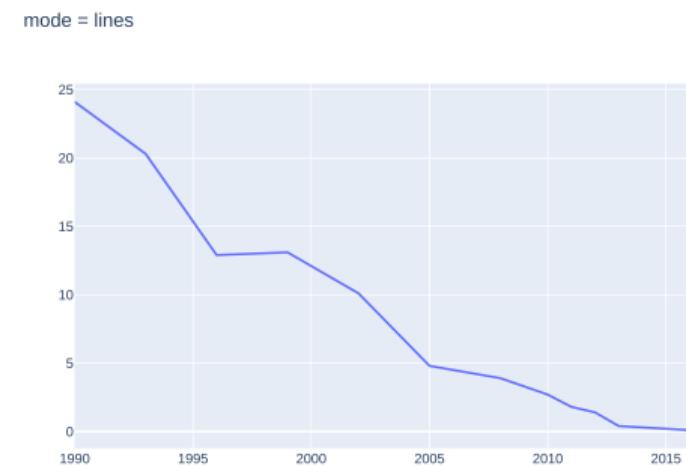
- `markers`: Csak jelölők
- `lines`: Csak vonalak
- `text`: Csak szöveget
- `markers+lines`: Jelölők és vonalak
- `markers+text`: Jelölők és szöveg
- `lines+text`: Vonalak és szöveg
- `markers+lines+text`: Jelölők, vonalak és szöveg



Alapvető pontdiagramok

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

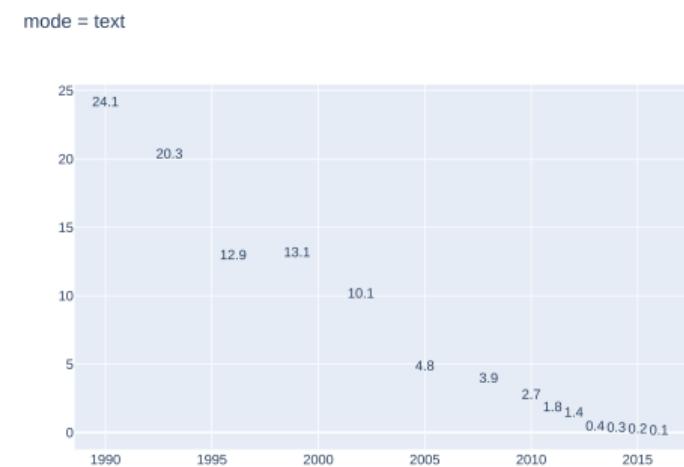
- `markers`: Csak jelölők
- `lines`: Csak vonalak
- `text`: Csak szöveget
- `markers+lines`: Jelölők és vonalak
- `markers+text`: Jelölők és szöveg
- `lines+text`: Vonalak és szöveg
- `markers+lines+text`: Jelölők, vonalak és szöveg



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

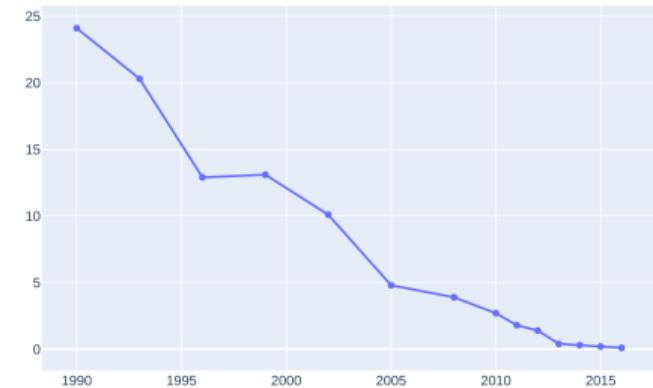


Alapvető pontdiagramok

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

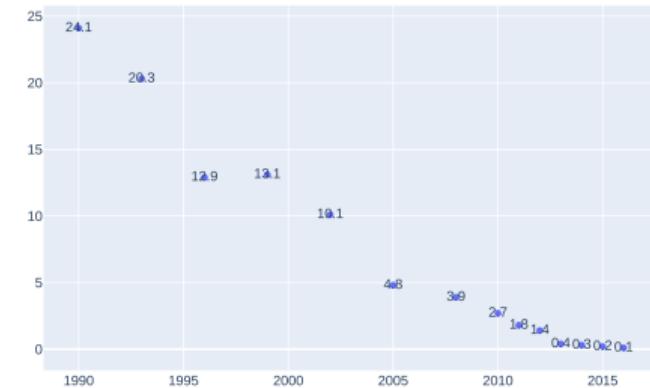


Alapvető pontdiagramok

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

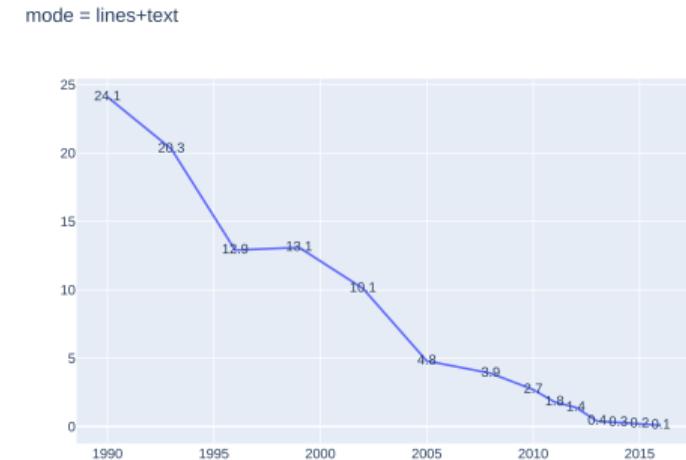
mode = markers+text



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

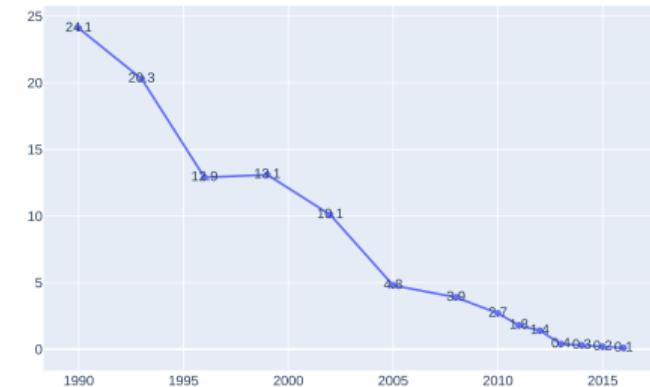


Alapvető pontdiagramok

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.

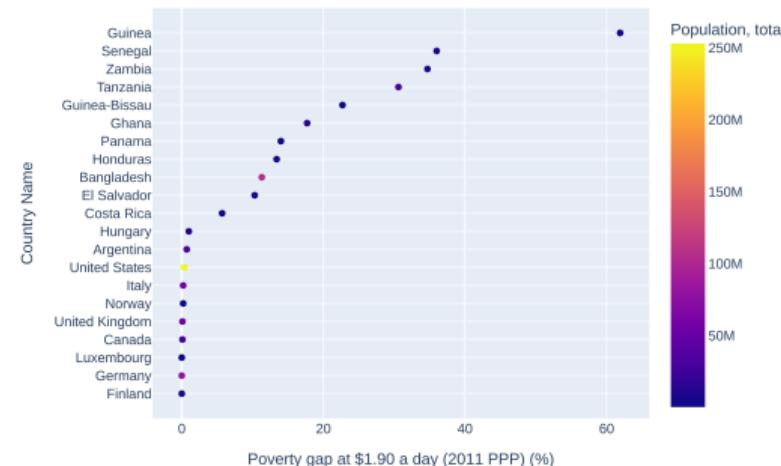
```
1 for country in countries:  
2     fig.add_scatter(x=df_country['year'],  
                         y=df_country[perc_pov_19], name=  
                         country, mode='markers+lines')
```



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.

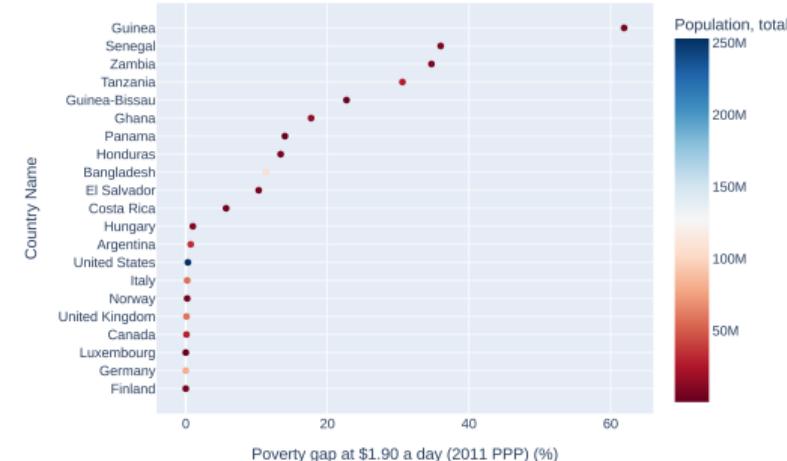
```
1 fig = px.scatter(df, x=indicator, y='Country Name', color='Population, total')
```



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'



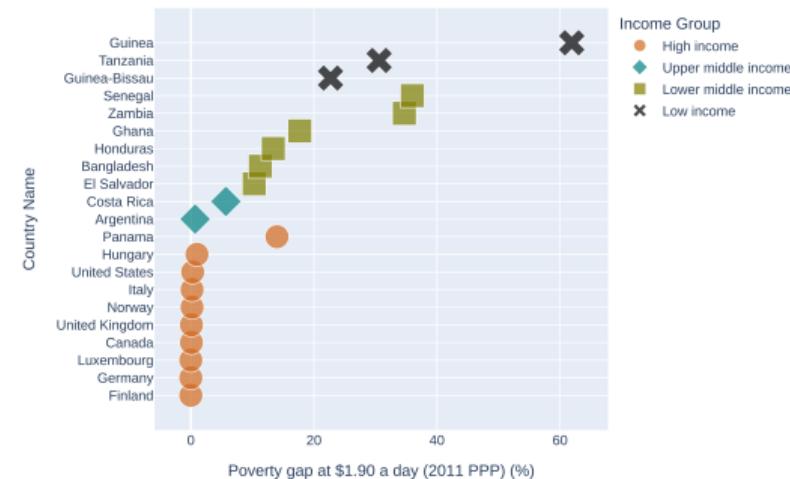
Egyéni színek diszkrét változókkal

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']`

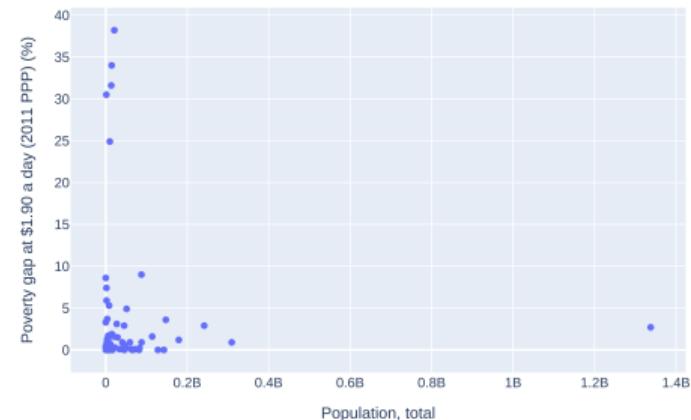


Kiugró értékek a gyakorlatban

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

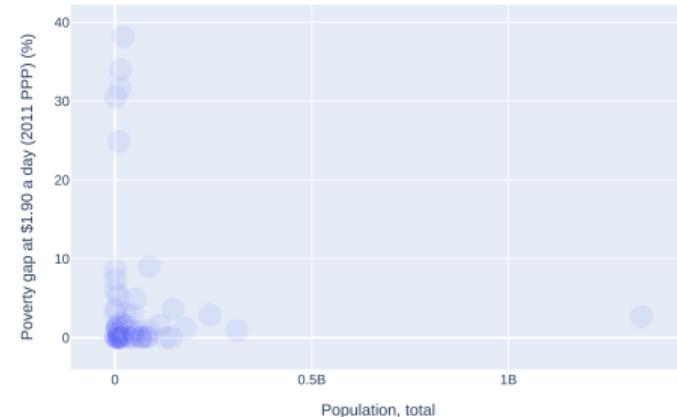


Markerek áttetszőségének és méretének állítása

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



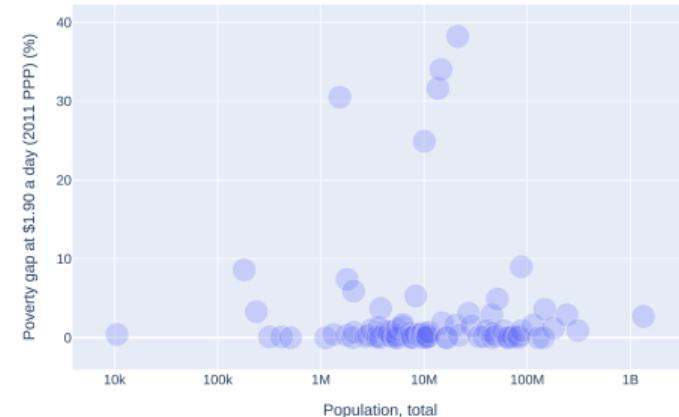
Logaritmikus skála használata

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

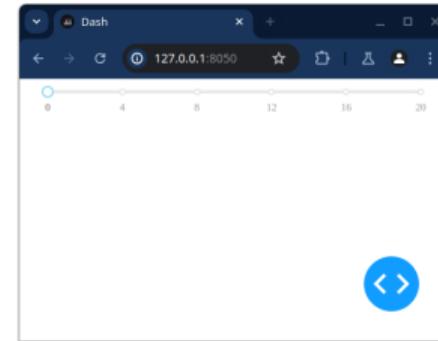


Dash csúszkák

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.

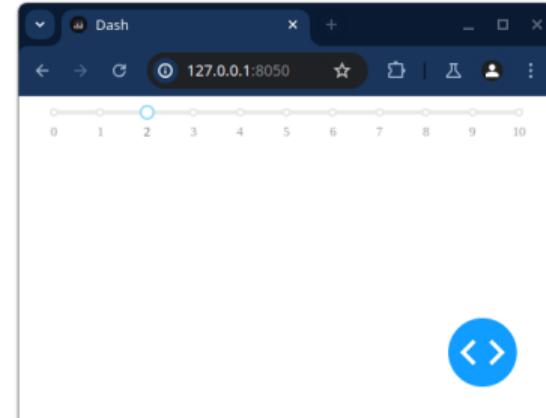
```
1 app = dash.Dash(__name__)
2 app.layout = html.Div([
3     dcc.Slider(
4         id='slider',
5         min=0,
6         max=20
7     )
8 ])
9 app.run_server(mode='inline')
```



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.

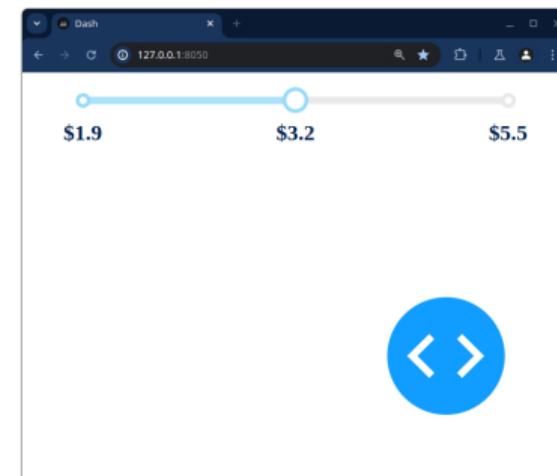
```
1dcc.Slider(  
2    min=0,  
3    max=10,  
4    step=1,  
5    dots=True,  
6    included=False  
7 )
```



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)

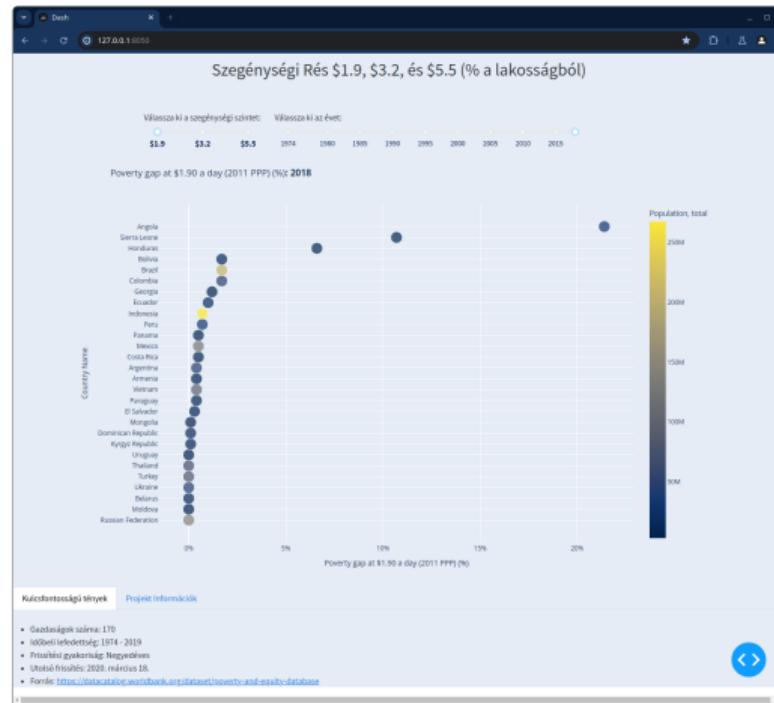
Az alkalmazás következő iterációjában két csúszka került hozzáadásra, az egyikkel a szegénységi szintet, a másikkal pedig az évet lehetséges kiválasztani.

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.



Csúszkák beépítése az alkalmazásba (app_v3_2.py)

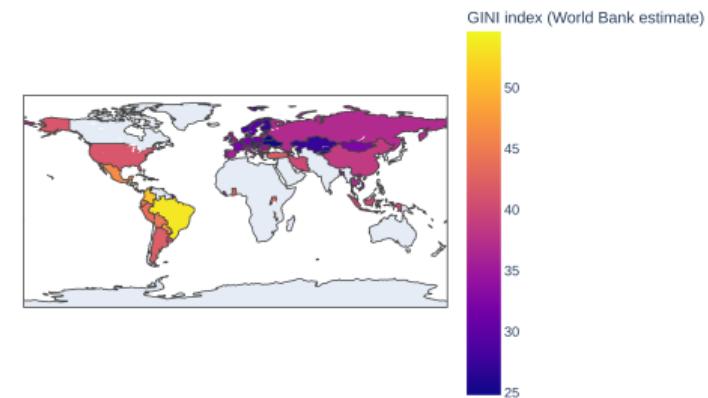


Egyeszerű tematikus térkép

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").

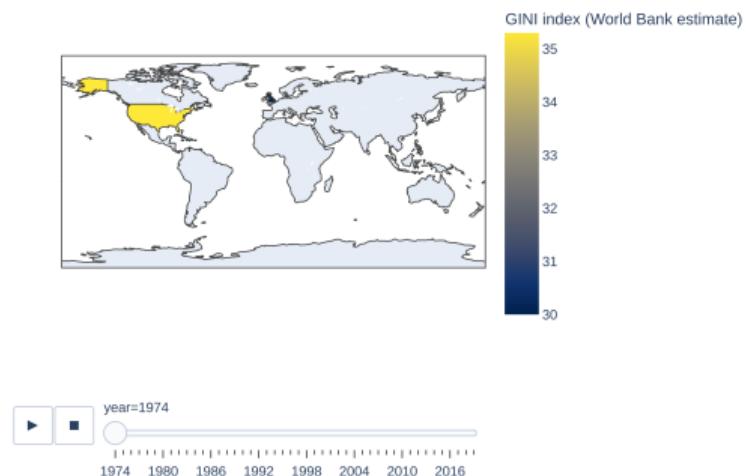
```
1 fig = px.choropleth(df, locations="Country Code", color=indicator)
```



Animációs réteg tematikus térképekkel

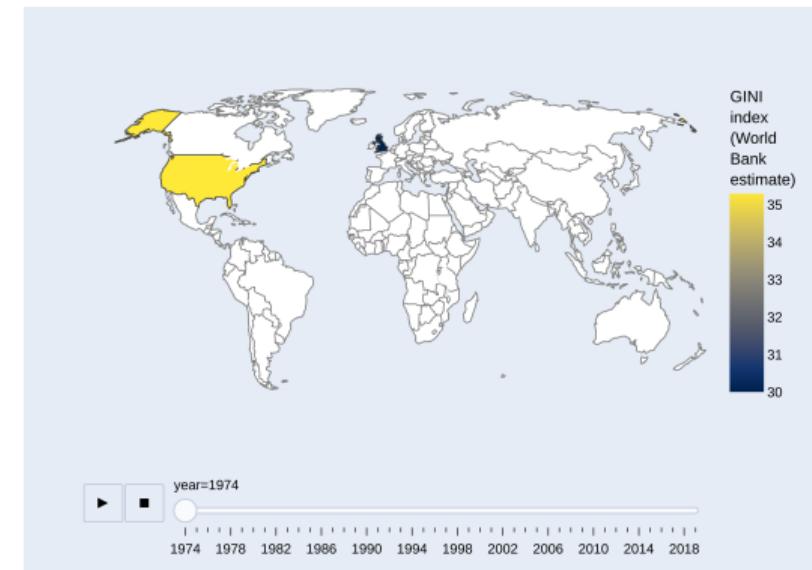
Az `animation_frame` paraméterrel
lehetséges bevezetni egy új, interaktív
réteget adó komponenst, ami a választott
változó alapján képes szekvenciálisan
változtatni a megjelenített diagramot.

```
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 DropDown komponens létrehozása:

```
1  dcc.Dropdown(  
2      id='indicator_dropdown',  
3      value='GINI index (World Bank  
4          estimate)',  
5      options=[{'label': indicator,  
6                  'value': indicator} for  
7                  indicator in poverty.  
8                  columns[3:54]]  
9  )
```

2 Graph komponens létrehozása a térképnél:

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

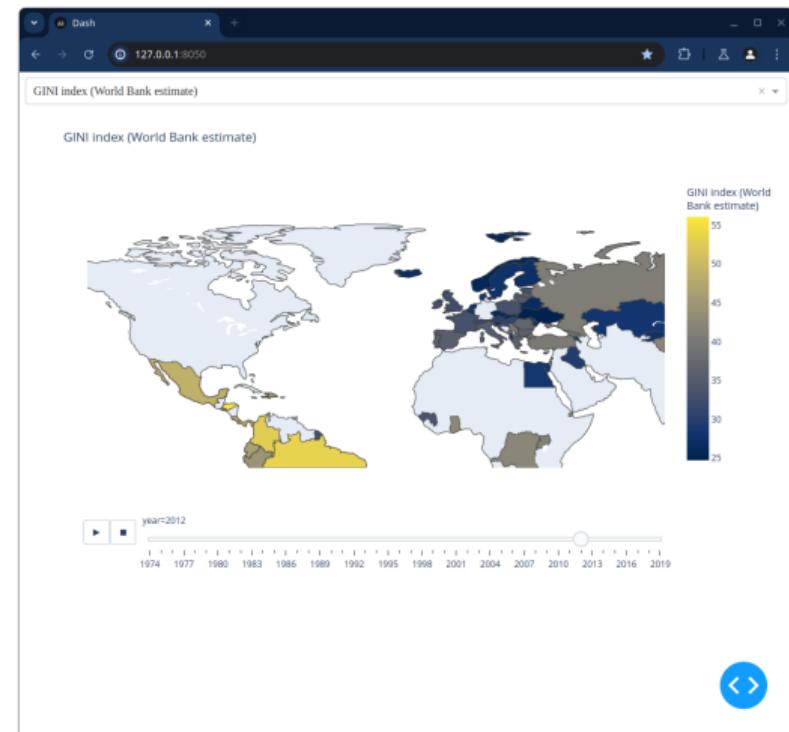
3 Callback függvény létrehozása

```
1  @app.callback(  
2      Output('indicator_map_chart', '  
3              figure'),  
4      Input('indicator_dropdown', '  
5              value'))  
6  def display_generic_map_chart(  
7      indicator):  
8      df = poverty[poverty['is_country  
9          ',']]  
10     fig = px.choropleth(  
11         df,  
12         locations='Country Code',  
13         color=indicator,  
14         ...  
15     )  
16     fig.layout.geo.showframe = False  
17     ...  
18     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
6 @app.callback(
7     Output('my-store', 'data'),
8     Input('save-button', 'n_clicks')
9 )
10 def save_data(n_clicks):
11     if n_clicks:
12         return {'key': 'value'}
13     return dash.no_update
```

dcc.Store komponensek

A Store komponenseknek 3 típusa

létezik:

- **memory:** Az adat a böngésző memoriájában 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.'
```

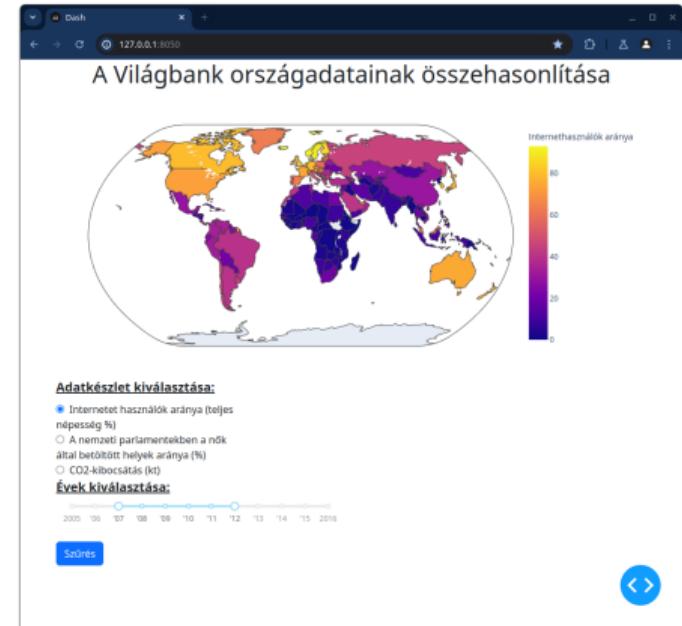
Alkalmazás Storage és Interval komponensekkel (map_app_v2.py)

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.

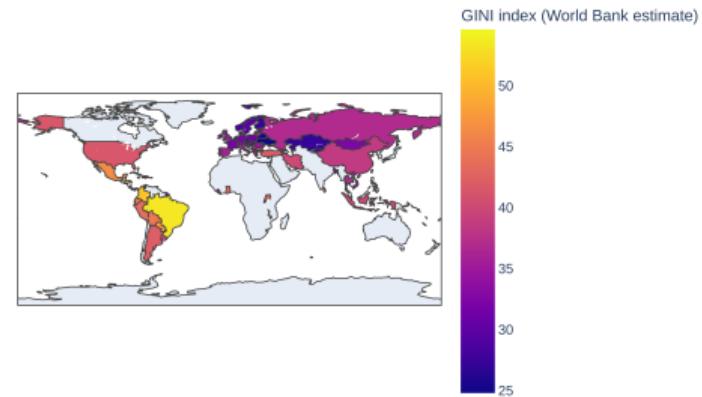


Térkép projekciók

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

A folyamat során szükségszerűen torzulások keletkeznek.

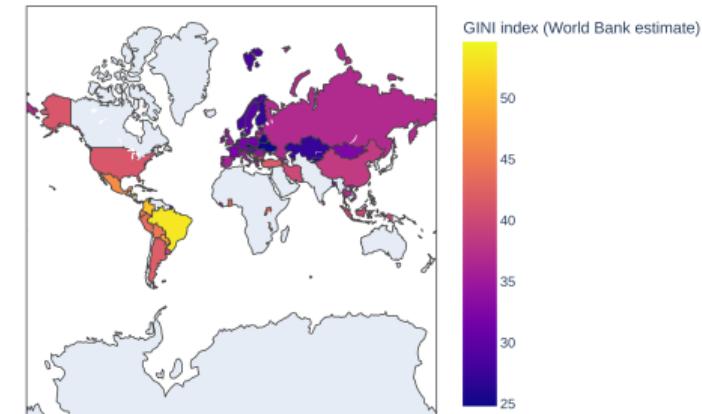
projection = equirectangular



Térkép projekciók

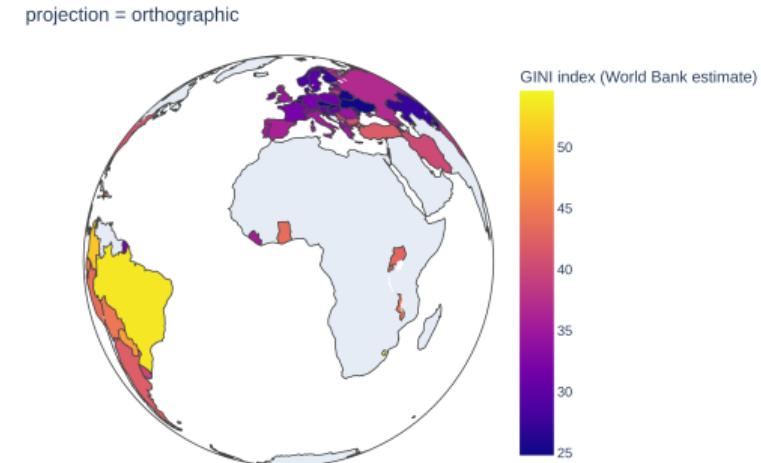
- **Equirectangular:** A legegyszerűbb projekció, ahol a földrajzi szélesség és hosszúság egyenesen arányosan van ábrázolva a síkon.
- **Mercator:** Szögtartó, tehát a szögek és az irányok helyesek maradnak, így hasznos a navigációban.
- **Orthographic:** Háromdimenziós gömböt ábrázol síkban úgy, mintha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

projection = mercator



Térkép projekciók

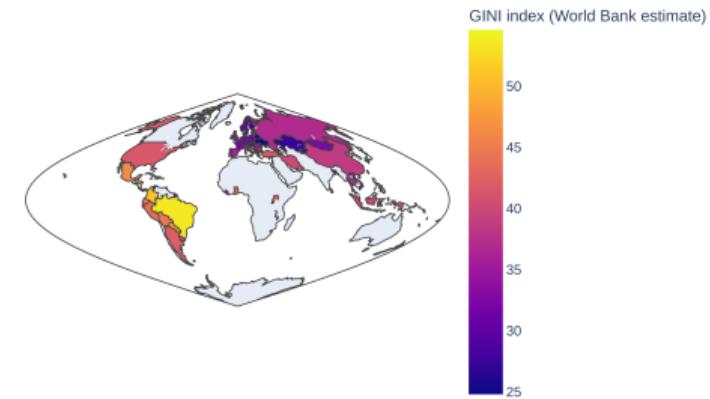
- **Equirectangular:** A legegyszerűbb projekció, ahol a földrajzi szélesség és hosszúság egyenesen arányosan van ábrázolva a síkon.
- **Mercator:** Szögtartó, tehát a szögek és az irányok helyesek maradnak, így hasznos a navigációban.
- **Orthographic:** Háromdimenziós gömböt ábrázol síkban úgy, mintha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



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'  
9 )
```



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 *Dőlt*  
2 **Félkövér**  
3 ***Félkövér és dőlt***
```

• 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
14     return markdown
```

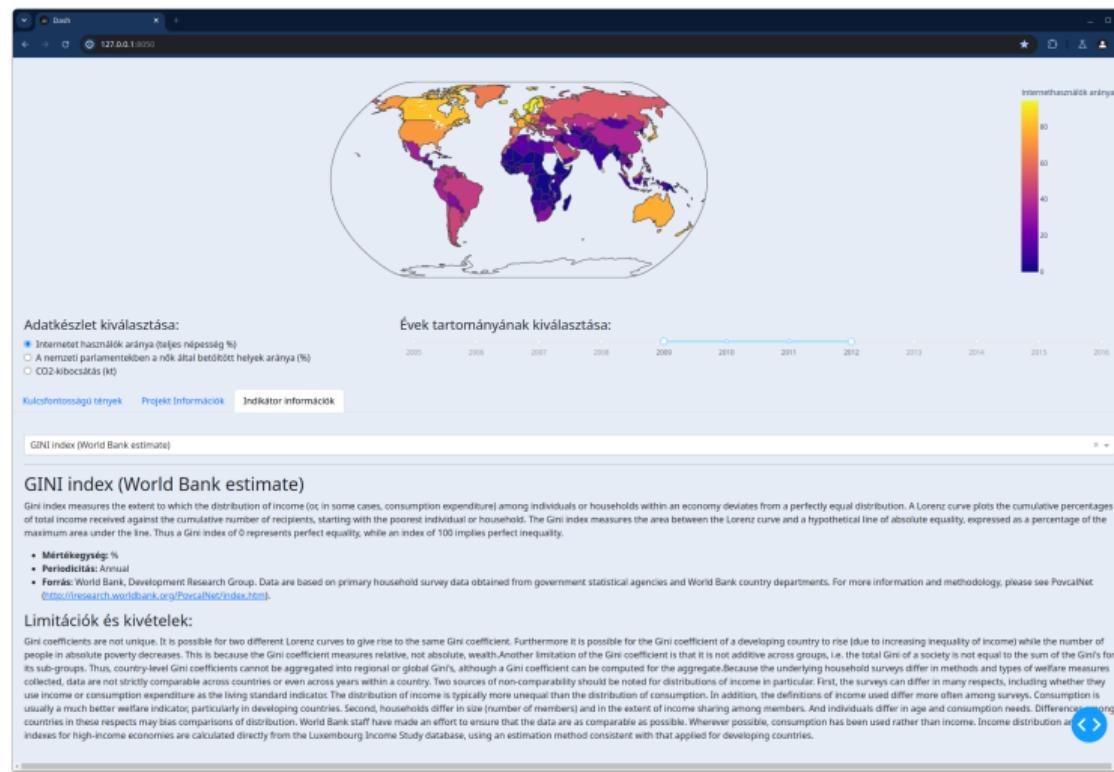
Markdown komponens beépítése a térkép alkalmazásba (map_app_v3.py)

Egy legördülő menü segítségével választható az indikátor neve. Az állapot változása elindít egy callback függvényt, ami kiolvassa az indikátornak megfelelő információt egy adatfájlból.

A kiolasztott tartalmat az alkalmazás Markdown formátumban jeleníti meg.



Alkalmazás térképpel és Markdown komponenssel (app_v3_3.py)



1 Dash alapok

2 Diagramok

3 Pontdiagramok

4 Gyakorisági adatok

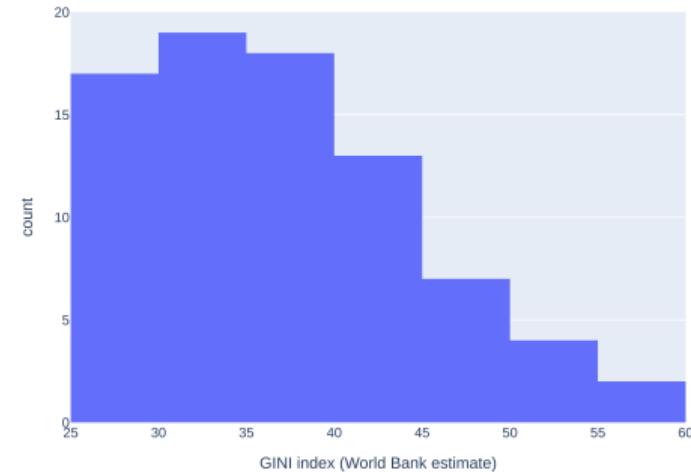
Hisztogramok létrehozása

Hisztogram

A hisztogram egy statisztikai grafikon, amely az adatok eloszlását mutatja be. Oszlopdiagram formájában ábrázolja, hogy az adatok milyen gyakorisággal fordulnak elő különböző intervallumokban.

Hisztogram létrehozása plotly segítségével:

```
1 px.histogram(data_frame=df, x=gini)
```



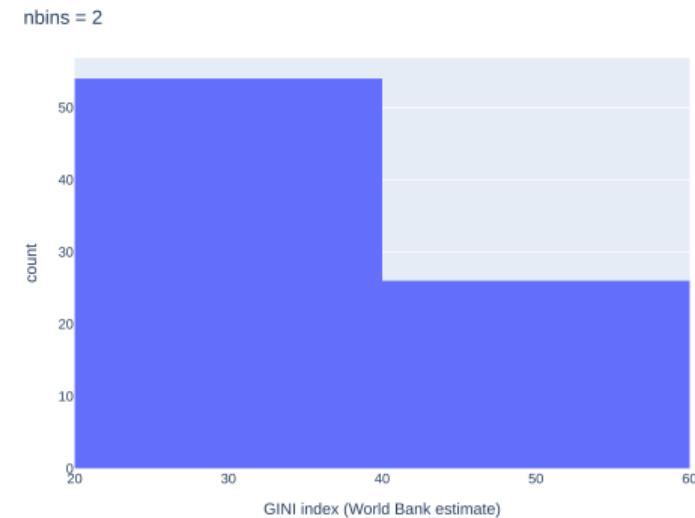
Hisztogramok felbontása

Osztályköz

Az osztályközök határozzák meg, hogy az adatok milyen tartományokba kerülnek, és ezek az intervallumok határozzák meg a hisztogram oszlopainak szélességét.

Az osztályközök száma az nbins paraméter segítségével állítható.

```
1 for n in [2, 45, 500]:  
2     px.histogram(data_frame=df, x=gini,  
                     nbins=n)
```



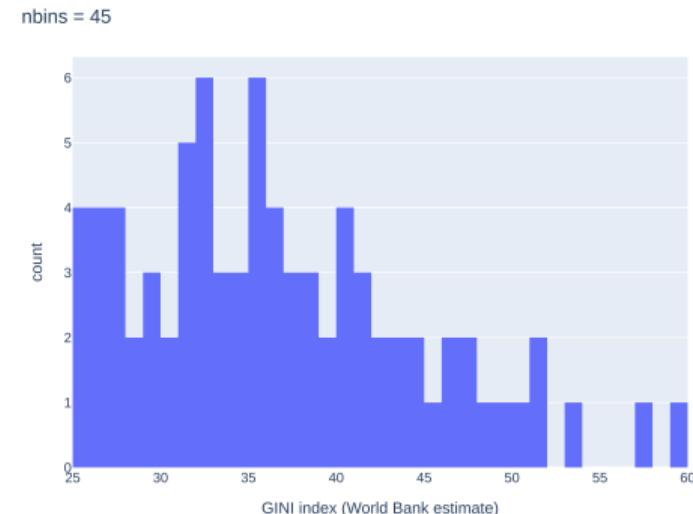
Hisztogramok felbontása

Osztályköz

Az osztályközök határozzák meg, hogy az adatok milyen tartományokba kerülnek, és ezek az intervallumok határozzák meg a hisztogram oszlopainak szélességét.

Az osztályközök száma az nbins paraméter segítségével állítható.

```
1 for n in [2, 45, 500]:  
2     px.histogram(data_frame=df, x=gini,  
                     nbins=n)
```



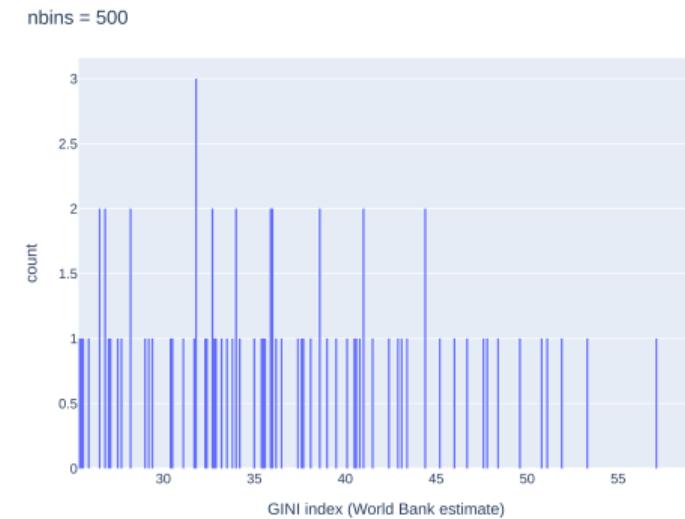
Hisztogramok felbontása

Osztályköz

Az osztályközök határozzák meg, hogy az adatok milyen tartományokba kerülnek, és ezek az intervallumok határozzák meg a hisztogram oszlopainak szélességét.

Az osztályközök száma az nbins paraméter segítségével állítható.

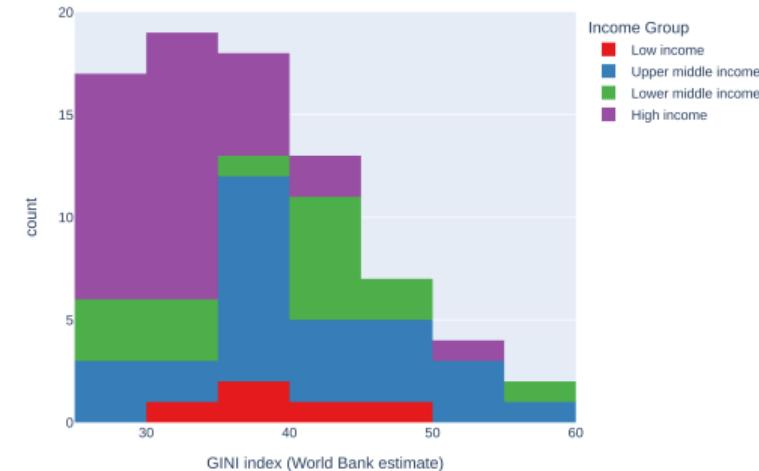
```
1 for n in [2, 45, 500]:  
2     px.histogram(data_frame=df, x=gini,  
                     nbins=n)
```



Hisztogram hasítása színekkel

Plotly express diagramokat lehetséges változón belüli csoportonként meghasítani. Ennek eléréséhez a color paramétert kell a megfelelő változóra állítani.

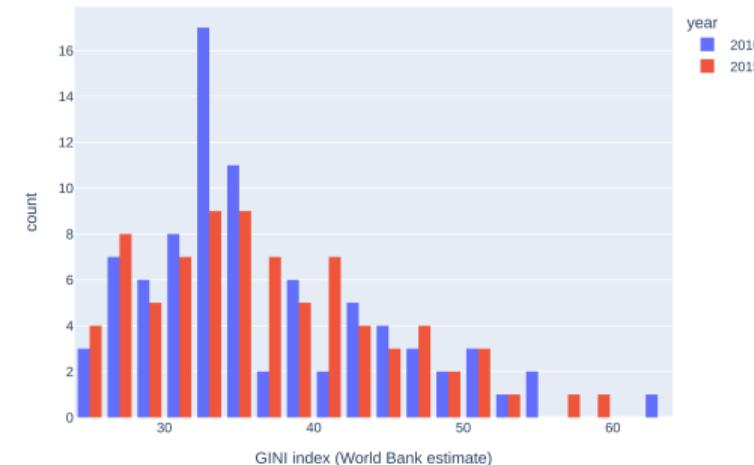
```
1 px.histogram(data_frame=df, x=gini,  
    color='Income Group',  
    color_discrete_sequence=px.colors.  
    qualitative.Set1)
```



Csoportosított hisztogramok

Vannak olyan esetek, amikor egy változónak több csoportját egymás mellett szükséges megmutatni. Ekkor a hisztogramokat lehetséges csoportosítani adott értékek szerint, a `color` és a `barmode='group'` paraméterek állításával.

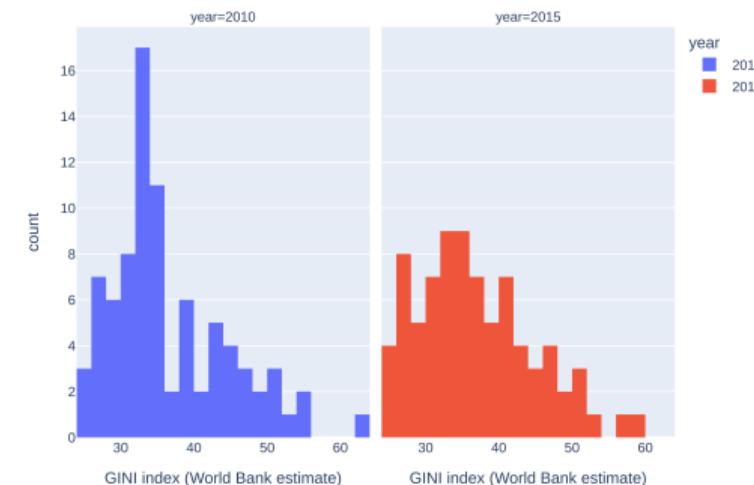
```
1 px.histogram(df, x=gini, color='year',  
               barmode='group')
```



Hasított hisztogramok

A diagramok hasítása adott változó értékei szerint lehetséges úgy is, hogy minden, a változóhoz tartozó értékre szűrt adathalmaz egy külön diagramon jelenik meg, a `facet_col` paraméter állításával.

```
1 px.histogram(df, x=gini, color='year',  
               facet_col='year')
```

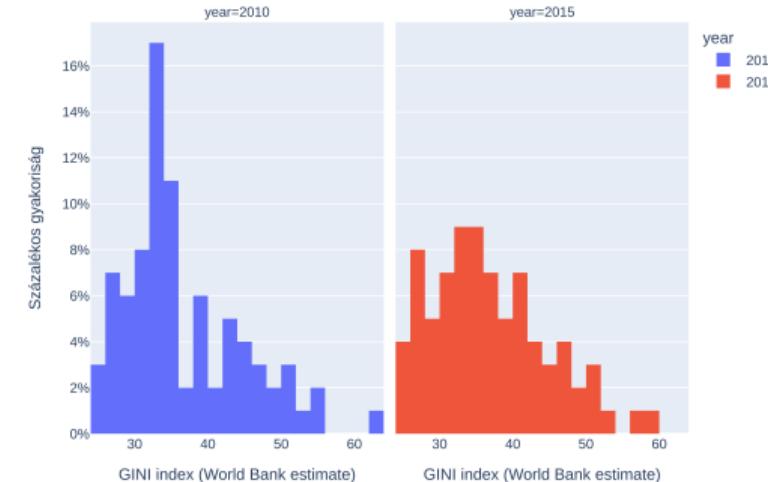


Hisztogramok normalizálása

Normalizált hisztogram

Olyan grafikon, ahol az egyes oszlopok az adott intervallumba eső adatok gyakoriságát jelzi olyan módon, hogy az oszlopok összege 1 legyen.

```
1 fig = px.histogram(df, x=gini, color='year', facet_col='year')
2 fig.layout.yaxis.ticksuffix = '%'
3 fig.layout.yaxis.title = 'Százalékos gyakoriság'
4 fig.show()
```

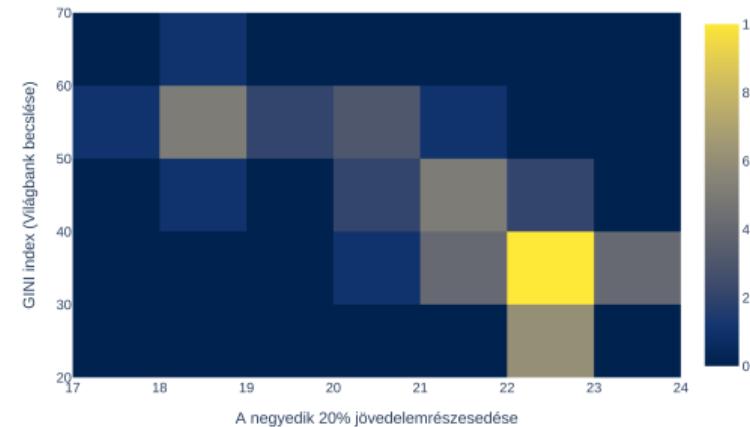


Hisztogramok több dimenzióban

2D hisztogram

Két dimenzióban osztja fel az adatokat, és minden cella (osztályköz) azt mutatja meg, hogy hány adatpont esik az adott tartományba mindkét változó esetében.

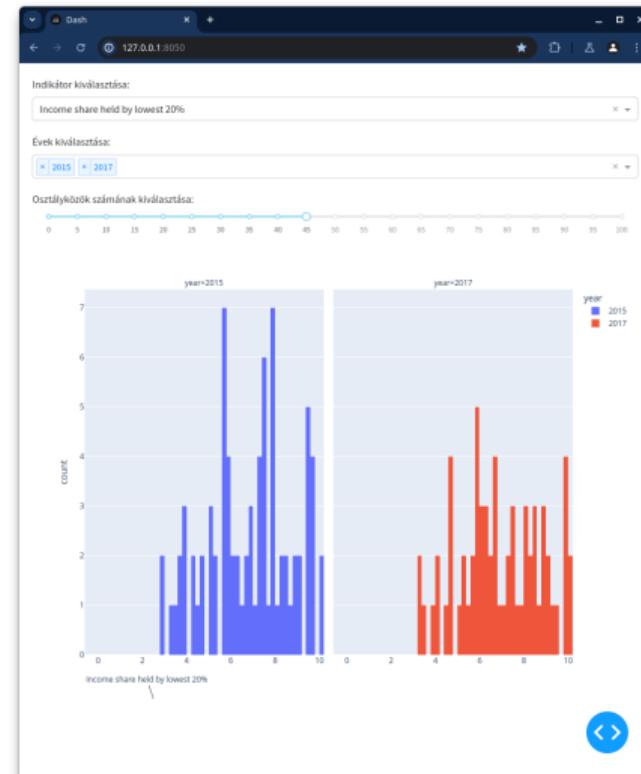
```
1 fig = go.Figure()
2 fig.add_histogram2d(
3     x=df['Income share held by fourth 20%'],
4     y=df['GINI index (World Bank estimate)'],
5     colorscale='cividis'
6 )
```



Alkalmazás interaktív hisztogramokkal (freq_app_v1.py)

A callback függvények a felhasználói interakciók alapján frissítik a grafikonokat. A `display_histogram` függvény három bemeneti elemet figyel (`years`, `indicator`, `bins`), és ezek alapján frissíti a hisztogram ábrát.

Ha nincs kiválasztott év vagy indikátor, a függvény nem frissíti az ábrát (`PreventUpdate`). Az adatok szűrése után a Plotly Express segítségével készül el a hisztogram.



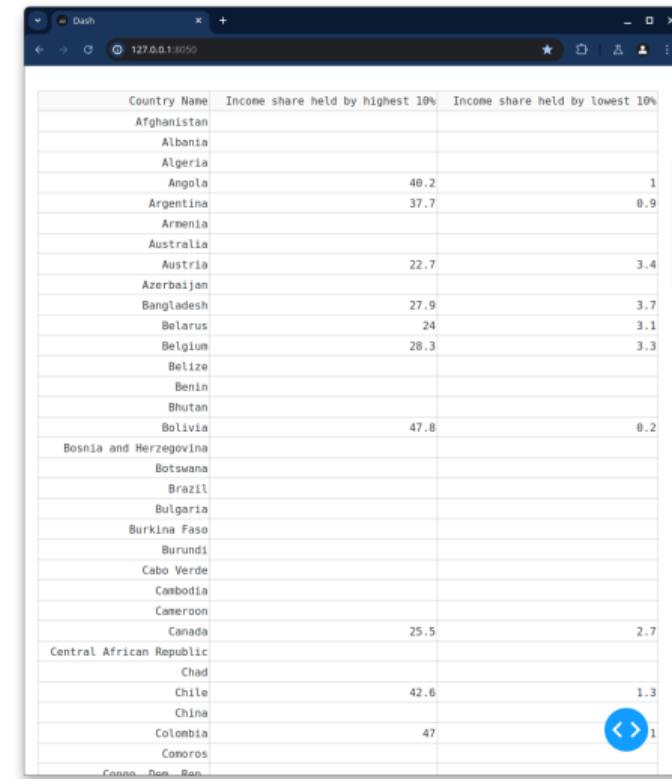
Interaktív hisztogramok beépítése az alkalmazásba (app_v4_1.py)



Adattábla létrehozása

A Dash keretrendszerben interaktív táblázatokat a dash_table könyvtárral lehet létrehozni.

```
1 from dash import html, dash_table
2
3 app.layout = html.Div([
4     ...
5     dash_table.DataTable(
6         data=pov_df.to_dict('records'),
7         columns=[{'name': col, 'id': col}
8                  for col in pov_df.columns]
9     )
10    ...])
```

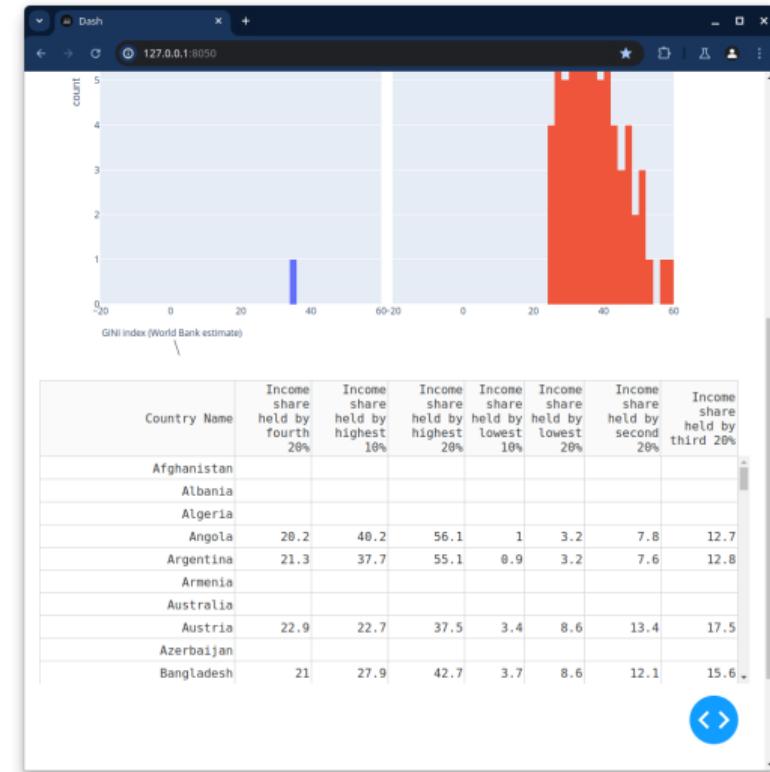


The screenshot shows a web browser window titled 'Dash' with the URL '127.0.0.1:8050'. The page displays an interactive table with three columns: 'Country Name', 'Income share held by highest 10%', and 'Income share held by lowest 10%'. The table lists various countries with their corresponding income share percentages. At the bottom right of the table, there is a blue circular button with a white double-headed arrow icon and the number '1'.

Country Name	Income share held by highest 10%	Income share held by lowest 10%
Afghanistan		
Albania		
Algeria		
Angola	48.2	1
Argentina	37.7	0.9
Armenia		
Australia		
Austria	22.7	3.4
Azerbaijan		
Bangladesh	27.9	3.7
Belarus	24	3.1
Belgium	28.3	3.3
Belize		
Benin		
Bhutan		
Bolivia	47.8	0.2
Bosnia and Herzegovina		
Botswana		
Brazil		
Bulgaria		
Burkina Faso		
Burundi		
Cabo Verde		
Cambodia		
Cameroon		
Canada	25.5	2.7
Central African Republic		
Chad		
Chile	42.6	1.3
China		
Colombia		
Comoros	47	
Conor. Dem. Rep.		

Adattábla személyre szabása

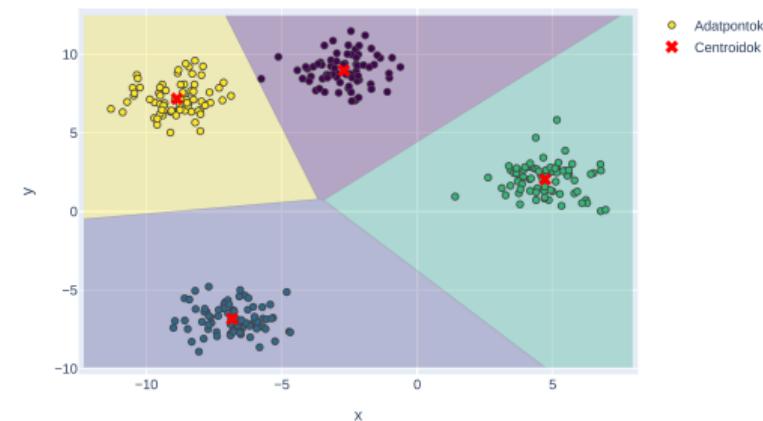
```
1 dash_table.DataTable(  
2     data=pov_df.to_dict('records'),  
3     columns=[{'name': col, 'id': col} for  
4             col in pov_df.columns],  
5     style_header={'whiteSpace': 'normal'  
6                   },  
7     fixed_rows={'headers': True},  
8     style_table={'height': '400px'},  
9     virtualization=True,  
10    )
```



K-közép klaszterezés

Az algoritmus eljárása:

- ➊ K számú klaszter centroid inicializálása véletlenszerűen:
 $\mu_1, \mu_2, \dots, \mu_K$
- ➋ minden x_i adatpont a hozzá legközelebb eső klaszterhez rendelése az euklideszi távolságot használva:
 $c_i = \arg \min_j \|x_i - \mu_j\|^2$
- ➌ Klaszterközéppontok újraszámítása úgy, hogy az adott klaszterhez tartozó pontok várható értékét tükrözzék:
 $\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$
- ➍ Ismétlés a kilépési kritériumig



Optimális klaszterszám megtalálása

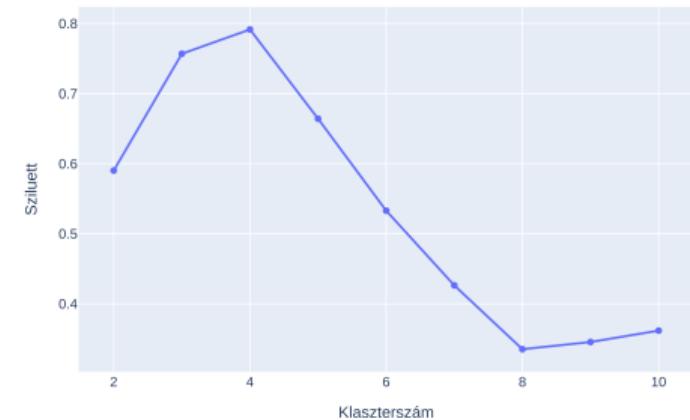
Egy klaszter konfiguráció annál jobb, minél szorosabban helyezkednek el az egy klaszterben lévő egyedek, és minél jobban elkülönülnek a más klaszterben lévő egyedektől. Ezt tükrözi a sziluett együttható:

Sziluett

$$S(x) \in [-1, 1] = \frac{a(x) - b(x)}{\max\{a(x), b(x)\}}$$

Ahol:

- $a(x)$: x mintaegyed és minden vele nem egy klaszterben lévő mintaegyed távolsága
- $b(x)$: x mintaegyed és minden vele egy klaszterben lévő mintaegyed távolsága



Scikit-learn K-közép

K-közép modul importálása és tanítása a make_blobs adathalmazon:

```
1 from sklearn.cluster import KMeans  
2  
3 kmeans = KMeans(n_clusters=n_clusters,  
                  random_state=random_state)  
4 kmeans.fit(X)  
5  
6 labels = kmeans.labels_  
7 centroids = kmeans.cluster_centers_
```

Klaszter centroidek x és y koordinátái:

```
1 In [1]: print(kmeans.cluster_centers_)  
2 Out [1]: [[-2.70981136  8.97143336]  
3           [-6.83235205 -6.83045748]  
4           [ 4.7182049   2.04179676]  
5           [-8.87357218  7.17458342]]
```

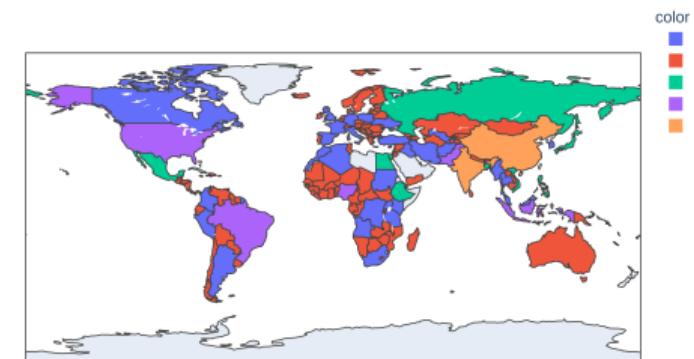
Becsült címkék az adathalmaz első 10 elemére:

```
1 In [2]: print(kmeans.labels_)  
2 Out [2]: [3 3 0 1 3 1 2 1 0 2]
```

Országok klaszterezése

A következő program az országokat 5 klaszterbe sorolja be, majd ennek a kimenetét felhasználva hoz létre egy tematikus térképet:

```
1 year = 2018
2 indicators = ['Population', 'total']
3 kmeans = KMeans(n_clusters=5)
4 df = poverty[poverty['year'].eq(year) &
               poverty['is_country']]
5 data = df[indicators].values
6 kmeans.fit(data)
7
8 fig = px.choropleth(
9     df,
10    locations='Country Name',
11    locationmode='country names',
12    color=[str(x) for x in kmeans.labels_
13        ])
```



Hiányzó értékek kezelése

Az `sklearn.impute.SimpleImputer` lehetővé teszi a hiányzó értékek pótlását különböző stratégiák alkalmazásával.

Néhány gyakori stratégia:

- `mean` (áttag): A hiányzó értékeket az adott oszlop átlagával helyettesíti
- `median` (medián): A hiányzó értékeket az adott oszlop mediánjával helyettesíti
- `most_frequent` (leggyakoribb): A hiányzó értékeket az adott oszlop leggyakoribb értékével
- `constant` (állandó): Egy megadott állandó értékkel helyettesíti a hiányzó értékeket.

`SimpleImputer` használata:

```
1 from sklearn.impute import  
    SimpleImputer  
  
2  
3 data = np.array([1, 2, 1, 2, np.nan]).  
    reshape(-1, 1)  
4 imp = SimpleImputer(strategy='mean')  
5 imp.fit(data)  
  
6  
7 print(imp.transform(data))
```

```
1 [[1. ]  
2 [2. ]  
3 [1. ]  
4 [2. ]  
5 [1.5]]
```

Adatok méretezése

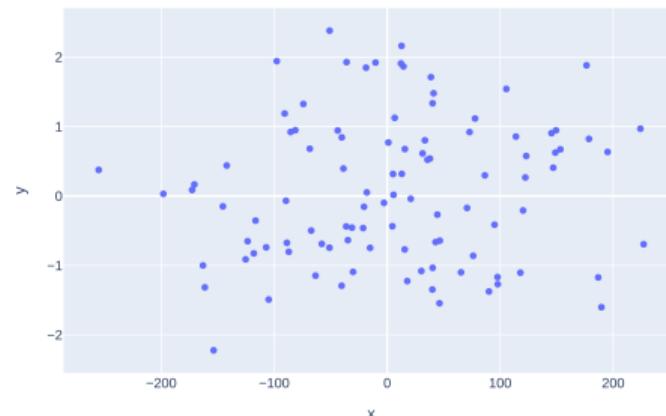
```
1 from sklearn.preprocessing import StandardScaler  
2  
3 data = np.array([1, 2, 3, 4, 5]).  
    reshape(-1, 1)  
4  
5 scaler = StandardScaler()  
6 print(scaler.fit_transform(data))
```

```
1 [[-1.41421356]  
2 [-0.70710678]  
3 [ 0. ]  
4 [ 0.70710678]  
5 [ 1.41421356]]
```

Méretezés

Adat előkészítési technika, mely során az adatok értékei egy adott tartományba transzformálódnak.

Méretezés előtt



Adatok méretezése

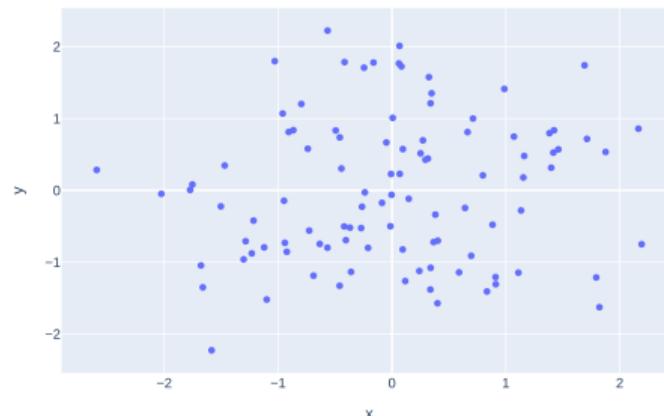
```
1 from sklearn.preprocessing import StandardScaler  
2  
3 data = np.array([1, 2, 3, 4, 5]).  
    reshape(-1, 1)  
4  
5 scaler = StandardScaler()  
6 print(scaler.fit_transform(data))
```

```
1 [[-1.41421356]  
2 [-0.70710678]  
3 [ 0. ]  
4 [ 0.70710678]  
5 [ 1.41421356]]
```

Méretezés

Adat előkészítési technika, mely során az adatok értékei egy adott tartományba transzformálódnak.

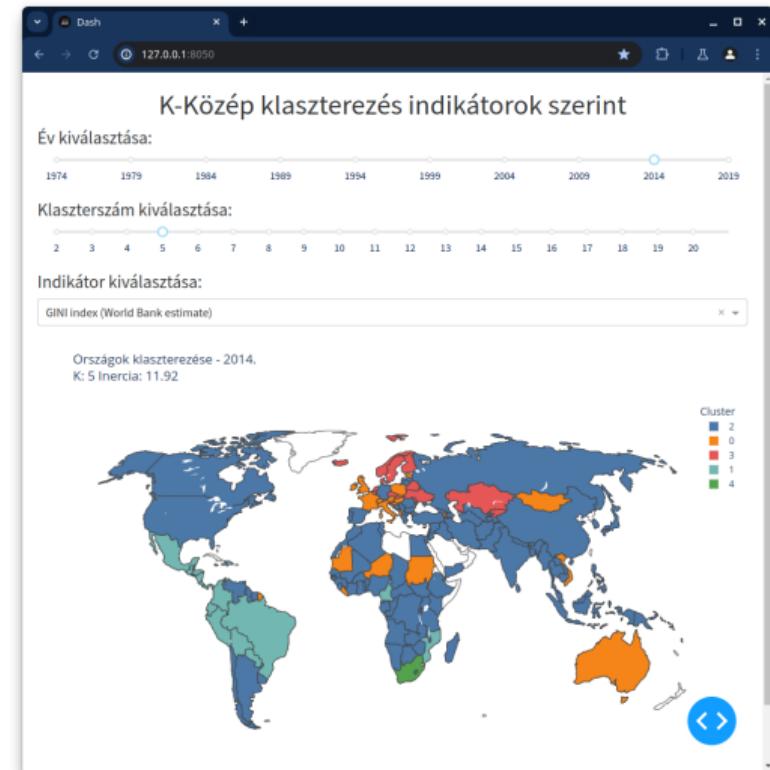
Méretezés után



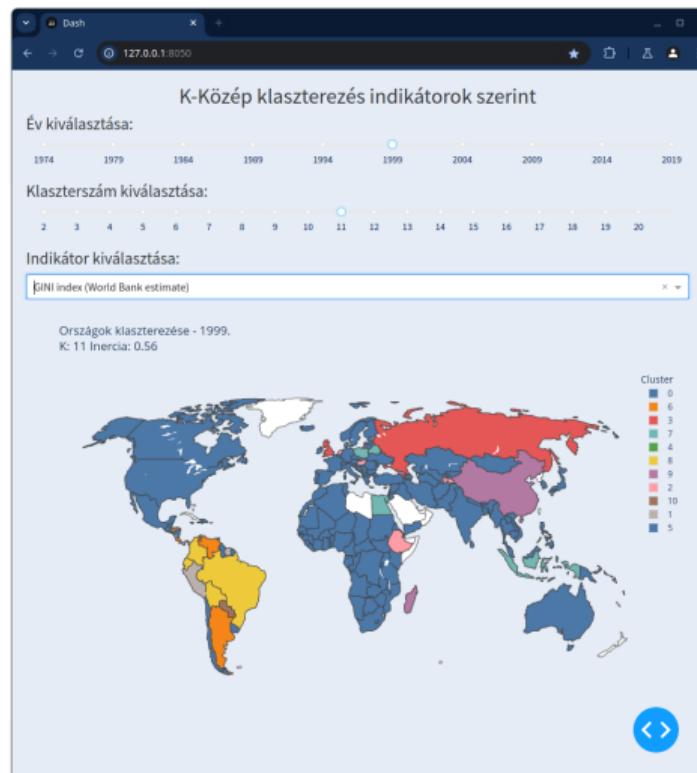
Alkalmazás interaktív térképpel, K -közép klaszterezéssel (`kmeans_app.py`)

A callback függvény frissíti a térképet a kiválasztott év, klaszterszám és indikátor alapján.

A hiányzó értékeket az átlaggal pótolja, majd az adatokat skálázza. A K-Közép algoritmust alkalmazza a transzformált adatokra, és a klaszterszámot a rendelkezésre álló adatok alapján korlátozza. Végül létrehozza a térképet, ahol az országokat a klaszter címkék alapján színezi, és finomhangolja a megjelenést.



K-közép beépítése a teljes alkalmazásba (app_v4_2.py)



Komponensek állapota

Az Output és Input mellett a harmadik callback argumentum a State.

- A sorrend a callback dekorátoron belül [Input, Output, State].
- Az Input komponens állapotának megváltoztatása elindítja a callback függvényt, a State nem.
- Ha egy Input elem módosul, a callback függvény State bemenő paraméterének értéke az lesz, amire az legutóbb módosult.

```
1 ...
2 html.Button('Futtatás', id='
3     kmeans_button')
4 ...
```

```
1 @app.callback(
2     Output('clustered_map_chart', 'figure')
3     ,
4     Input('kmeans_button', 'n_clicks'),
5     State('year_cluster_slider', 'value')
6     ,
7     State('ncluster_slider', 'value'),
8     State('indicator_dropdown', 'value'),
9 )
10 def clustered_map(n_clicks, year,
11     n_clusters, indicator):
12     if n_clicks > n_clusters:
13         return no_update
14     ...
```

Interaktív töltés indikátor

A `dcc.Loading` képes visszajelzést adni egy felhasználónak, amíg a bele ágyazott komponens frissül.

Egy diagram beágyazása Loading komponensbe:

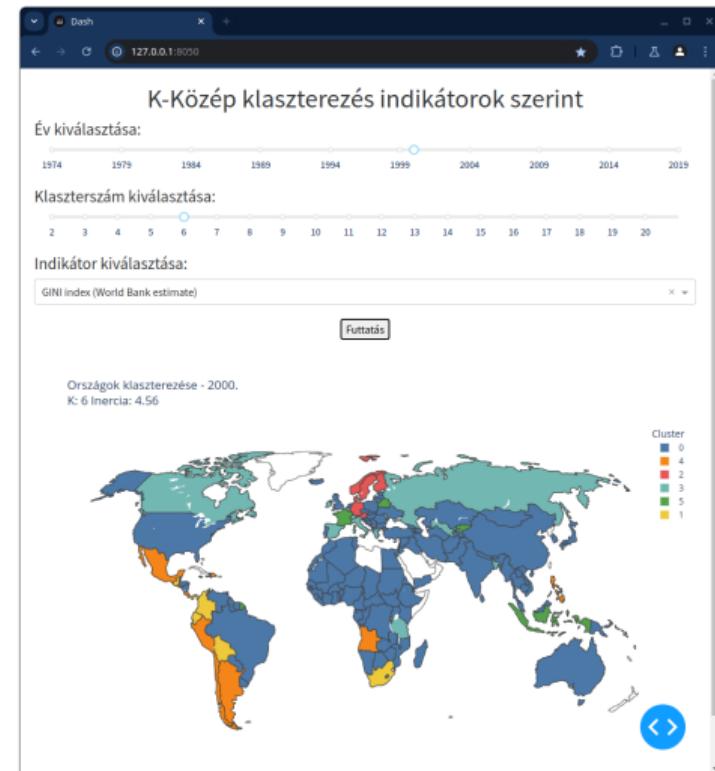
```
1dcc.Loading([
2  dcc.Graph(
3    id='clustered_map_chart',
4  ),
5])
```



K-közép alkalmazás állapottal

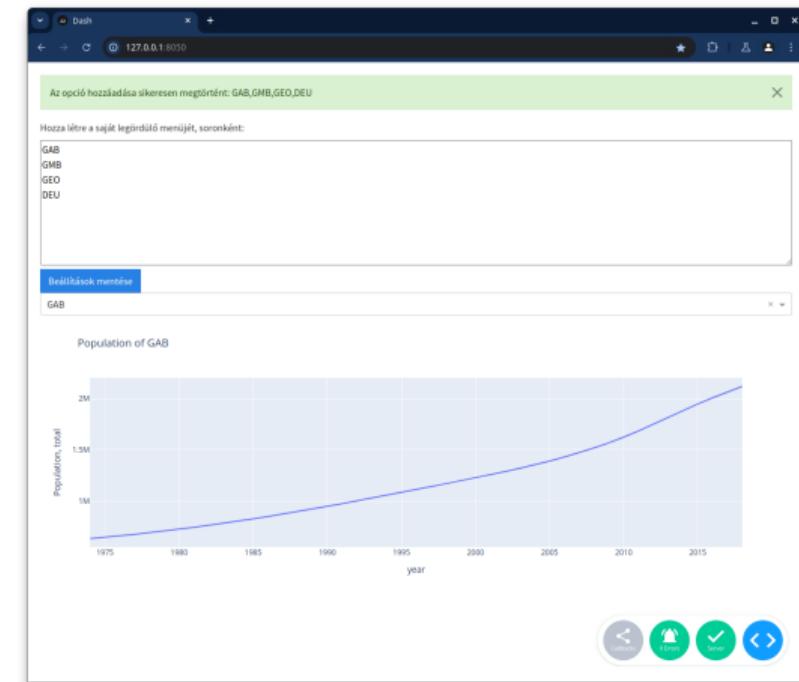
Az alkalmazás funkcionálisában
megegyezik az előző verzióval, viszont
ebben az esetben a callback függvény a
Futtatás gombra kattintással indul el.

A vezérlőelemek állapota State
argumentumon keresztül adódik át a
függvénynek, ezért nem indul el az elemek
állapotának módosításakor.



Komponensvezérlő alkalmazás (component_app_v1.py)

- Elrendezés:** Az alkalmazás HTML és Dash komponensekből épül fel, beleérte információs üzenetet, egy szövegmezőt, gombot, legördülő menüt és grafikont.
- Callback függvények:**
 - Legördülő menü frissítése:** A gomb megnyomására frissíti a legördülő menü opciót a szövegmező tartalma alapján, és visszajelző üzenetet jelenít meg.
 - Diagram frissítése:** A legördülő menü kiválasztott értéke alapján frissíti a népesség diagramot.



Információs üzenetek

A dbc.Alert információs üzenetek megjelenítésére használhatunk a Dash alkalmazásokban:

- primary: Információ kék színnel
- danger: Figyelmeztetés piros színnel
- success: Siker zöld színnel

Div definiálása az elrendezésben:

```
1 app.layout = html.Div([
2 ...
3     html.Div(id='component_feedback'),
4 ...
5])
```

Callback függvény ami a Div.children attribútumot módosítja:

```
1 @app.callback(
2     Output('component_feedback', 'children')
3 )
4 def set_feedback():
5     return dbc.Alert(
6         f"Az opció hozzáadása sikeresen megtörtént: {', '.join(text)}",
7         color='success',
8         dismissable=True,
9     )
```

Az opció hozzáadása sikeresen megtörtént: GAB,GMB,GEO,DEU



Elrendezés létrehozása

- ➊ Div létrehozása az üzenetnek:

```
1 html.Div(id='component_feedback'),
```

- ➋ dcc.TextArea szövegdoboz:

```
1 dcc.Textarea(  
2     id='component_text',  
3     cols=40,  
4     rows=5,  
5     style={'width': '100%', 'height'  
6             : 200},  
7 ),
```

- ➌ Gomb, ami a callback függvényt indítja:

```
1 dbc.Button('Beállítások mentése',  
            id='component_button'),
```

- ➍ Legördülő lista:

```
1 dcc.Dropdown(id='  
                component_dropdown'),
```

- ➎ Grafikon:

```
1 dcc.Graph(id='component_chart'),
```

Legördülő menü frissítése

A függvény a component_button gomb megnyomására indul, és állapotként a szövegmező aktuális értékét fogadja.

A szövegmező tartalmát a logika felbontja szóközök mentén különálló szavakra, majd egy sikeres visszajelző üzenettel tér vissza azáltal, hogy a component_feedback Div állapotát változtatja meg.

Az új opciókat egy listába rendezi, majd visszatér az új legördülő menü opciókkal és a diagrammal.

```
1 @app.callback(
2     Output('component_dropdown', 'options'),
3     Output('component_feedback', 'children'),
4     Input('component_button', 'n_clicks'),
5     State('component_text', 'value')
6 )
7 def set_dropdown_options(n_clicks, options):
8     if not n_clicks:
9         return no_update
10    text = options.split()
11    message = dbc.Alert(
12        f"Az opció hozzáadása sikeresen megtörtént
13        : {', '.join(text)}",
14        color='success',
15        dismissable=True
16    )
17    options = [{ 'label': t, 'value': t} for t in
18                text]
19    return options, message
```

Diagram frissítése

A függvény a component_chart legördülő menü kiválasztott értékét (country_code) veszi bemenetként. Ha nincs kiválasztott érték a diagram nem frissül.

A poverty adatkészlet szűrése után létrehoz egy vonaldiagramot, amely az adott ország népességének változását mutatja meg az évek során.

```
1 @app.callback(
2     Output('component_chart', 'figure'),
3     Input('component_dropdown', 'value')
4 )
5 def create_population_chart(
6     country_code):
7     if not country_code:
8         return no_update
9     # Adatkészlet szűrése
10    df = poverty[poverty['Country Code']
11                  == country_code]
12    # Diagram létrehozása
13    return px.line(
14        df,
15        x='year',
16        y='Population, total',
17        title=f"Population of {country_code}
18              }"
19    )
```

Az alkalmazás callback gráfja

A callback gráf vizuálisan ábrázolja a különböző komponensek közötti kapcsolatokat, segít lekövetni felhasználói események sorozatait balról elindulva a nyilak mentén, és hogy melyik komponens melyik attribútuma indította el a folyamatot.

Ez megkönnyíti a Dash alkalmazásokban való hibakeresést.

