Adatbányászat a Gyakorlatban

3. Gyakorlat: Dash diagramok

Kuknyó Dániel
Budapesti Gazdasági Egyetem

2024/25
1.félév

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()
```

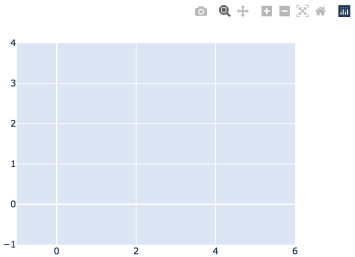


Figure objektumok attribútumai

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...

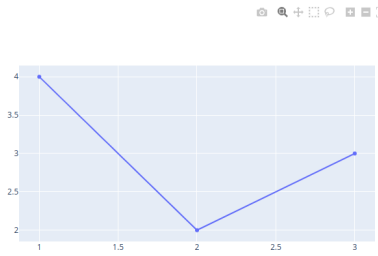
layout

Ez az attribútum határozza meg a diagram elrendezését és stílusát. Ide tartoznak a tengelyek címkéi, 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.

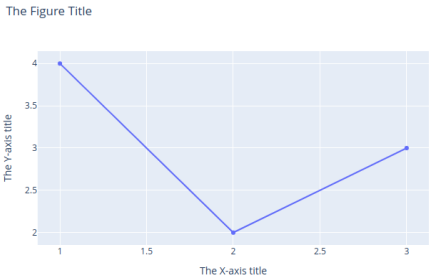
```
1 import plotly.graph_objects as go
2
3 fig = go.Figure()
4 fig.add_scatter(x=[1, 2, 3], y=[4, 2, 3])
5 fig.show()
```



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'
```



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.

Grafikon konfigurálása

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

```
fig.show(
  config={
    'displaylogo': False,
    'modeBarButtonsToAdd': ['drawrect',
        'drawcircle', 'eraseshape']
  }
)
```

- `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:

```
app.layout = html.Div([
  dcc.Dropdown(
    id='year_dropdown',
    value='2010',
    options=[{'label': year, 'value':
        str(year)} for year in range
        (1974, 2019)]
  ),
  dcc.Graph(id='population_chart')
])
```

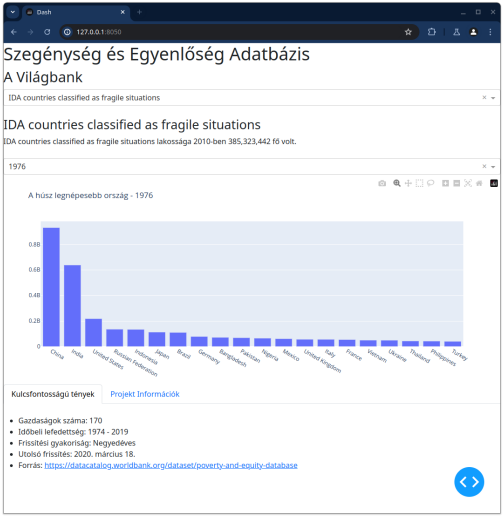
A `Dropdown` komponensben definiálva vannak a lehetséges értékek, és a `Graph` egy egyedi `id` adattaggal van ellátva az interaktivitás miatt.

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:

```
@app.callback(
  Output('population_chart', 'figure'),
  Input('year_dropdown', 'value')
)
def plot_countries_by_population(year):
  year_df = ...
  fig = go.Figure()
  ...
  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 alkalmazásaikat. 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

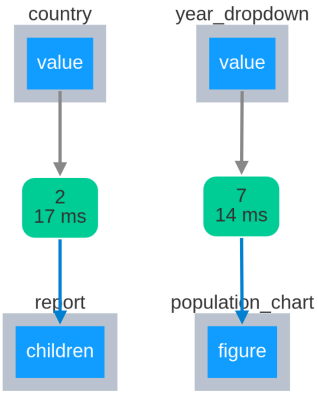
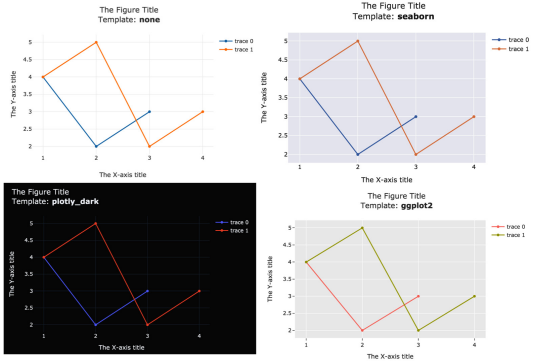


Diagram témák szerkesztése

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:

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

	numbers	colors	floats	shapes	letters
0	1	blue	1.1	rectangle	A
1	2	green	1.2	circle	A
2	3	orange	1.3	triangle	A
3	4	yellow	2.4	rectangle	B
4	5	black	2.1	circle	B
5	6	gray	5.6	triangle	C
6	7	pink	6.2	rectangle	C
7	8	white	5.3	circle	C

Pontdiagram

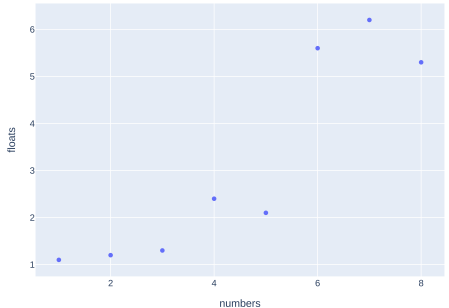
Egy plotly express diagramnak kétféleképpen is át lehet adni az adathalmazt.

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:

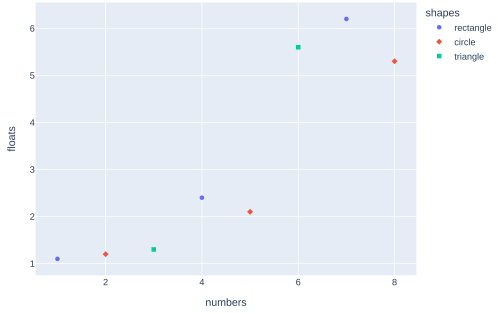
```
1 px.scatter(x=df['numbers'], y=df['
      floats'])
```



Pontdiagram kategóriákkal

Ebben az esetben minden adatosztály egy külön nyomként jelenik meg.

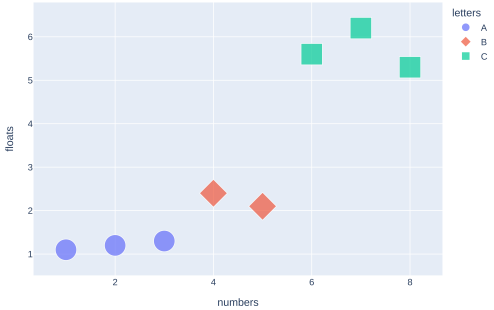
```
1 px.scatter(df, x='numbers', y='floats',
      color='shapes', symbol='shapes')
```



Pontdiagram jelölőkkel

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

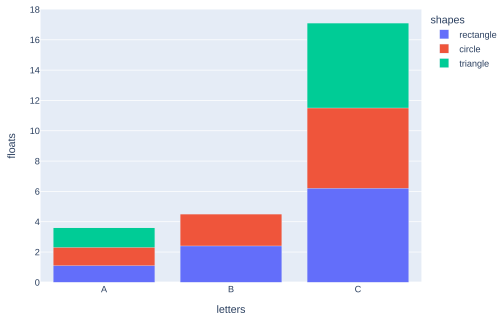
```
1 px.scatter(df, x='numbers', y='floats',
      color='letters', symbol='letters',
      size=[35] * 8)
```



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.

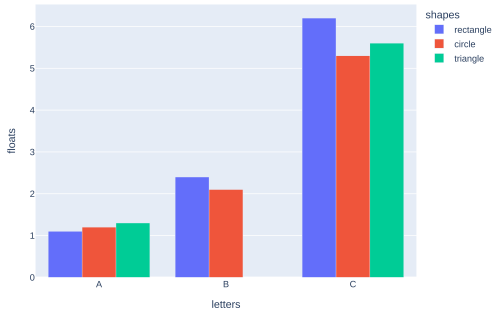
```
1 px.bar(df, x='letters', y='floats',
      color='shapes')
```



Csoportosított oszlopdiagram

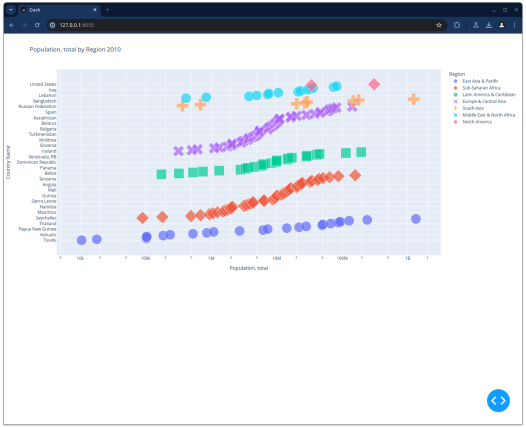
Csoportosítás esetén az adatcsoportok nem egymáson, hanem egymás mellett foglalnak helyet. A csoportosítás attribútuma itt is a `color`, és a csoportosítási típust a `barmode=color` adja meg.

```
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

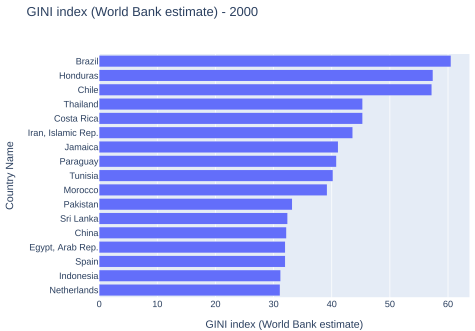


Fektetett oszlopdiagram

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:

```
fig = px.bar(
  df,
  x=gini,
  y='Country Name',
  title=' - '.join([gini, str(year)]),
  orientation='h'
)
```

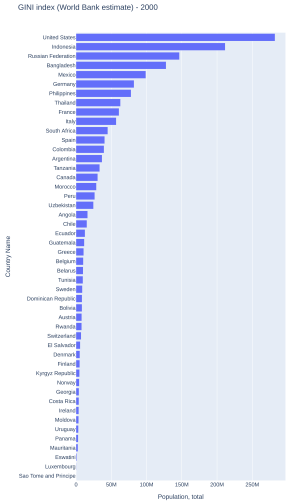


Dinamikus méretű oszlopdiagram

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 országonként 20 pixellel nő meg a `height` paraméter.

```
fig = px.bar(
  df,
  x=indicator,
  y='Country Name',
  title=' - '.join([gini, str(year)]),
  height=200 + (20 * n_countries),
  orientation='h',
)
```



Oszlopmódok

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

`relative`

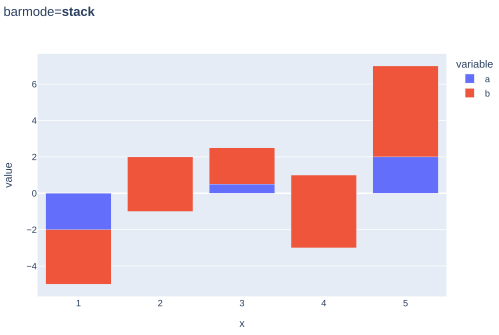
Az oszlopok egymás mellett jelennek meg, és az értékek relatív különbségeit mutatják.

Oszlopmódok

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

`group`

Az oszlopok csoportosítva jelennek meg, különböző kategóriák szerint.

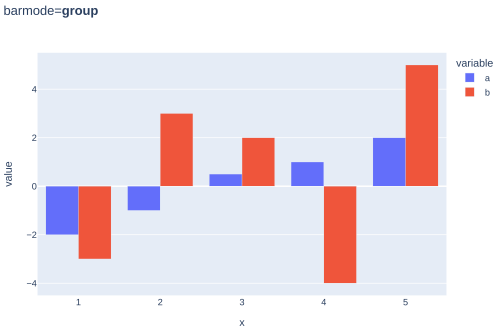


Oszlopmódok

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

`overlay`

Az oszlopok egymásra helyezve jelennek meg, átfedésben.

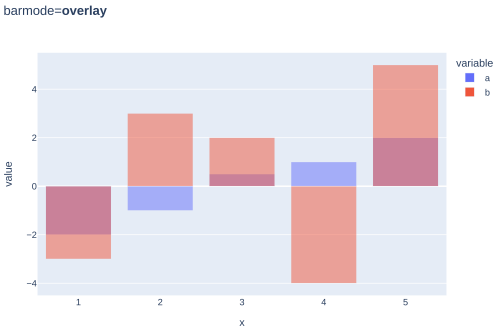


Oszlopmódok

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

`stack`

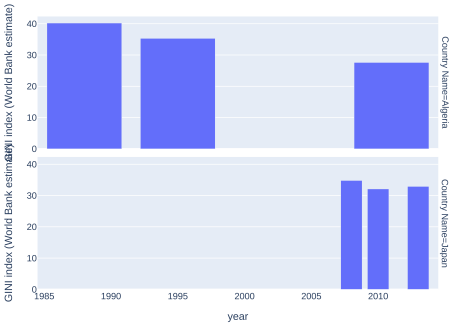
Az oszlopok egymásra rakva jelennek meg, az értékek összeadódnak.



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` objektun `figure` adattagját frissíti egy plotly express diagrammal.

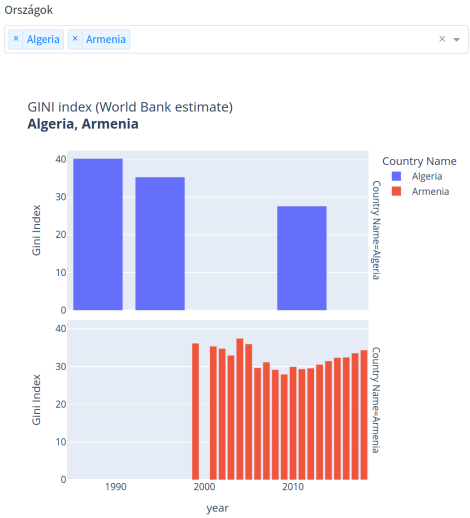
```
dcc.Dropdown(
  id='year_dropdown',
  value='2010',
  options=[{'label': year, 'value': str
      (year)} for year in range(1974,
      2019)]
),
...
dcc.Graph(id='population_chart'),
```

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

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

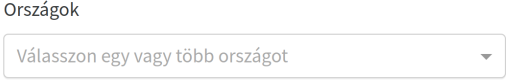


Helykitöltő szöveg legördülő listákhoz

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.

```
dcc.Dropdown(
  id='gini_country_dropdown',
  placeholder='Válasszon egy vagy több országot',
  multi=True,
  options=[{'label': country, 'value': country} for country in gini_df['Country Name'].unique()]
),
```



Elrendezés komponensek újrafelhasználása

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 definíciójához.

```
app.layout = html.Div([
  *app_v2_3.app.layout.children[:-1],
  dbc.Row([
    ...
  ]),
  app_v2_3.app.layout.children[-1]
])
```

A lista kicsomagolás operátor

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

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

```
def f(*argv):
  ...

f('Hello', 'World')
```

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

```
def f(**kwargs):
  ...

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.

```
def register_callbacks(param_app):
  @param_app.callback(
    ...
  )
  def callback1(...):
    ...

  @param_app.callback(
    ...
  )
  def callback2(...):
    ...
```

A meghívás egy másik alkalmazásból:

```
import app_v2_1

app = dash.Dash(__name___)
...
app_v2_1.register_callbacks(app)
```