

Adatbányászat a Gyakorlatban

5. Gyakorlat: Gyakorisági adatok kezelése

Kuknyó Dániel
Budapesti Gazdasági Egyetem

2024/25
1.félév

1 Bevezetés

2 Adattáblák

3 Gépi tanulás

4 Összetett callback függvények

5 Komponenseket irányító komponensek

1 Bevezetés

2 Adattáblák

3 Gépi tanulás

4 Összetett callback függvények

5 Komponenseket irányító komponensek

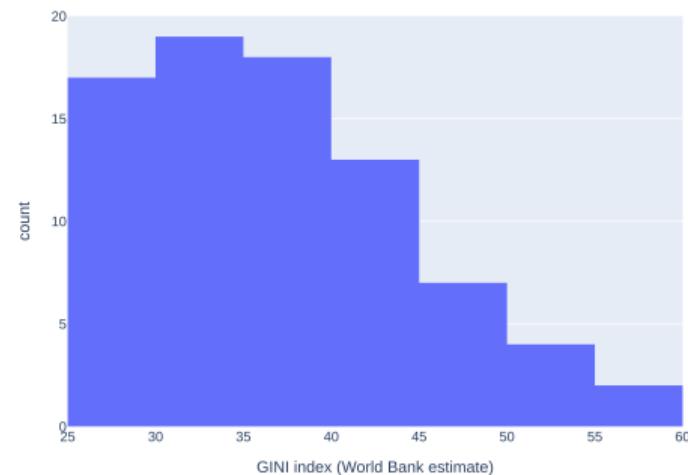
Hisztogramok létrehozása

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)
```



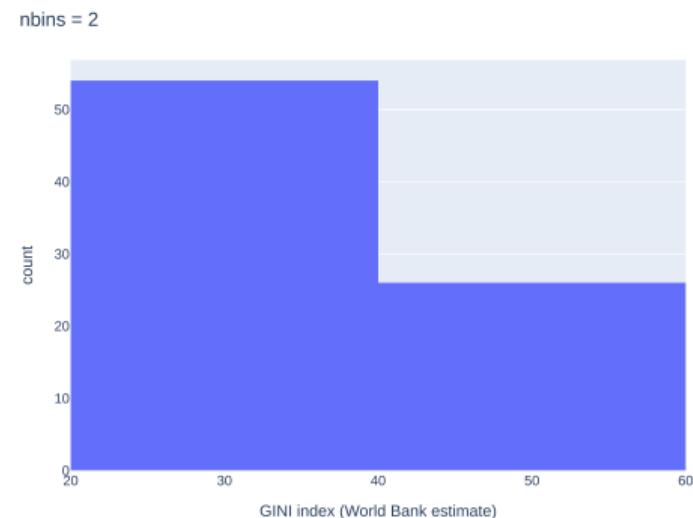
Hiszistogramok felbontása

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)
```



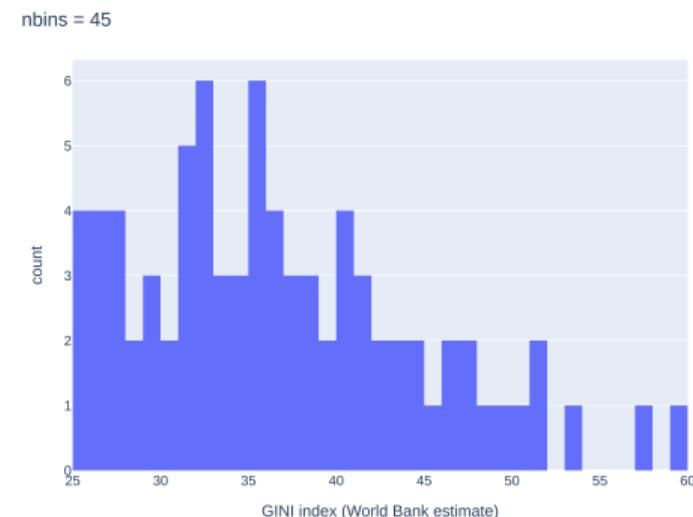
Hisztorogramok felbontása

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)
```



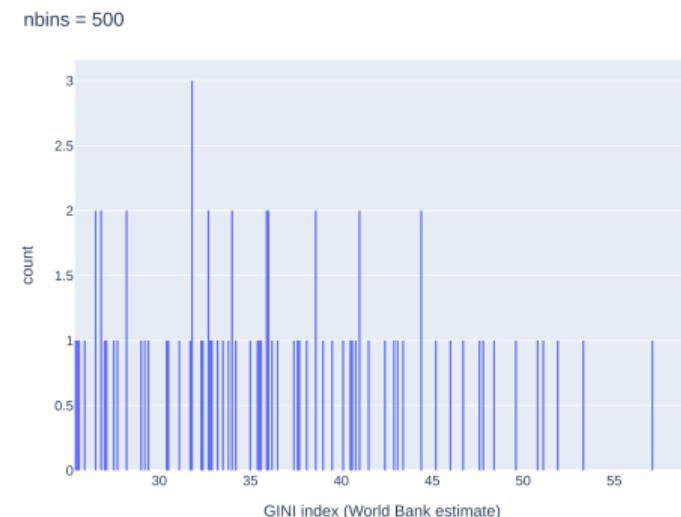
Hiszistogramok felbontása

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.

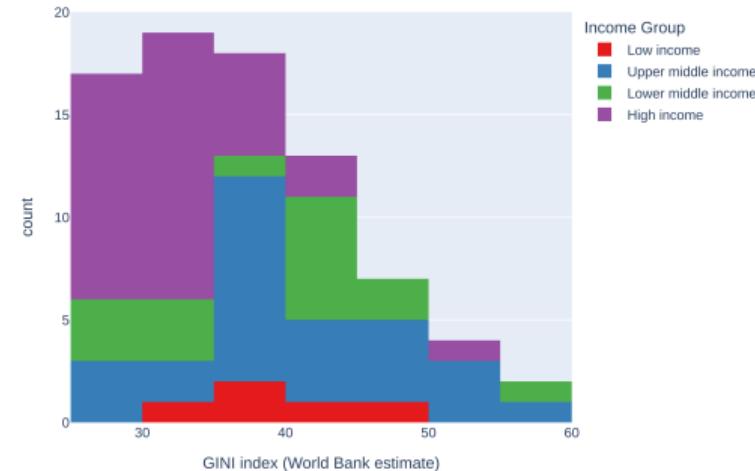
```
1 for n in [2, 45, 500]:  
2     px.histogram(data_frame=df, x=gini,  
                     nbins=n)
```



Hisztogram hasítása színekkel

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

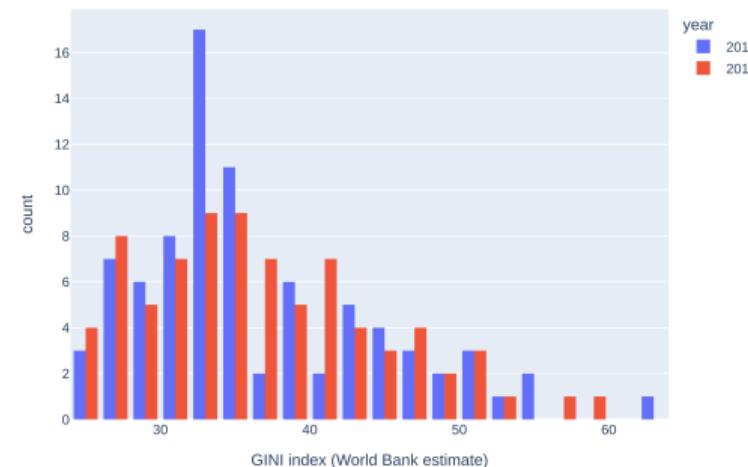
```
1 px.histogram(data_frame=df, x=gini,  
    color='Income Group',  
    color_discrete_sequence=px.colors.  
    qualitative.Set1)
```



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.

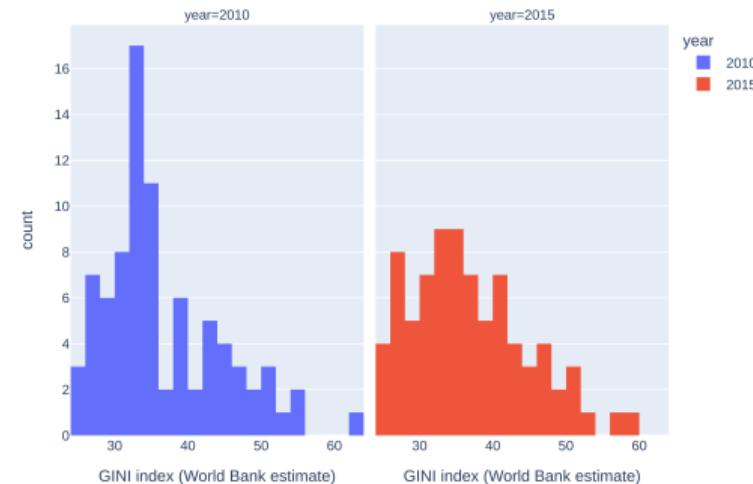
```
1 px.histogram(df, x=gini, color='year',  
   barmode='group')
```



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')
```

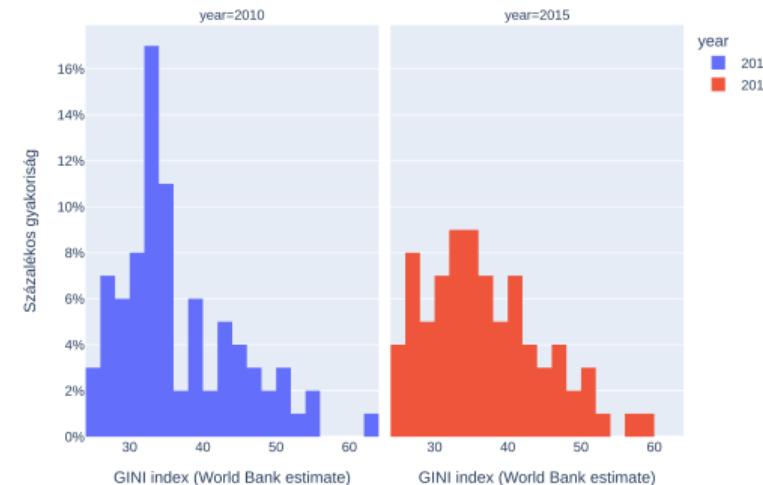


Hiszistogramok normalizálása

Normalizált hiszistogram

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()
```

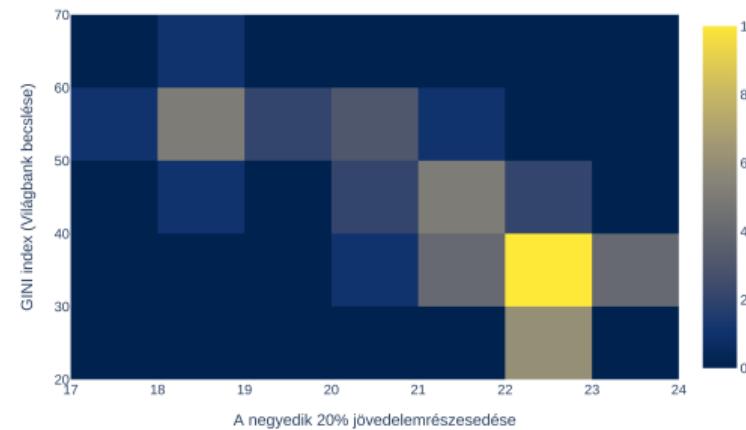


Hisztogramok több dimenzióban

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.

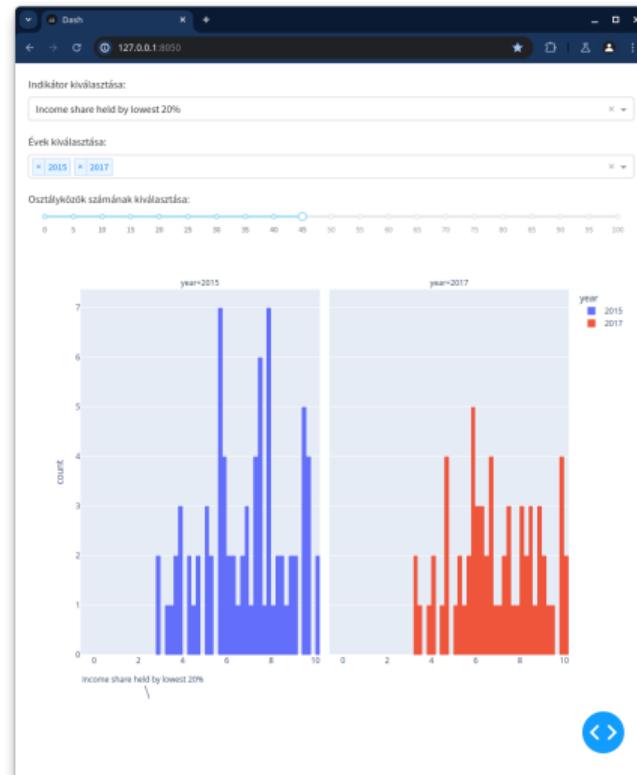
```
1 fig = go.Figure()
2 fig.add_histogram2d(
3     x=df['Income share held by fourth 20%'],
4     y=df['GINI index (World Bank estimate)'],
5     colorscale='cividis'
6 )
```



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)



1 Bevezetés

2 Adattáblák

3 Gépi tanulás

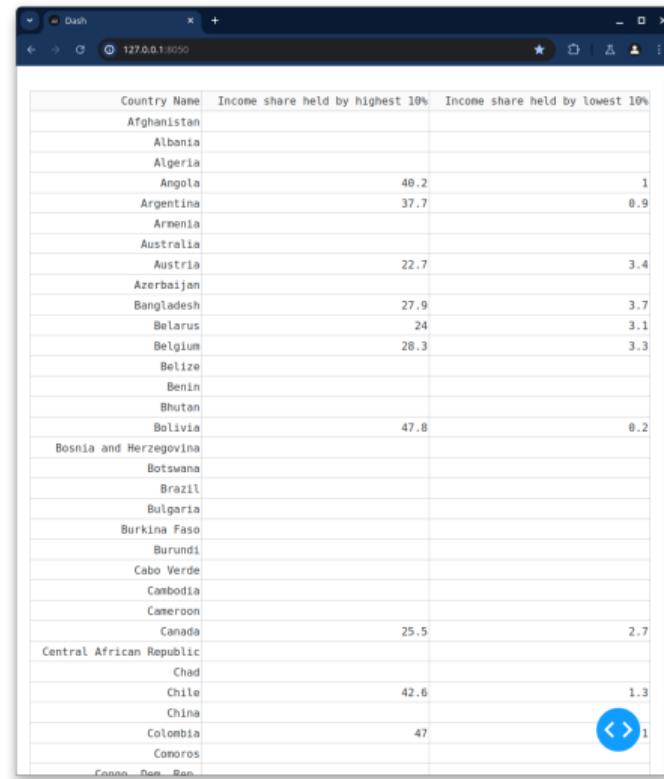
4 Összetett callback függvények

5 Komponenseket irányító komponensek

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    ...
11])
```

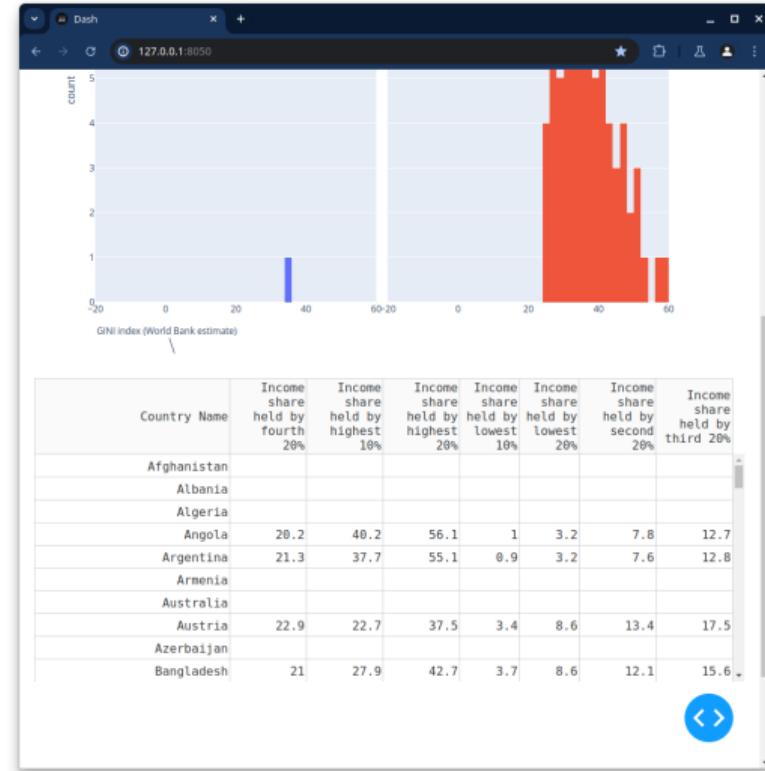


The screenshot shows a Dash application window titled 'Dash' running on '127.0.0.1:8050'. The table displays data from a DataFrame named 'pov_df'. The columns are 'Country Name', 'Income share held by highest 10%', and 'Income share held by lowest 10%'. The data includes 30 countries with their respective income shares. A blue circular navigation button with a double arrow is visible at the bottom right of the table.

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	47	
Comoros		
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    )
```



1 Bevezetés

2 Adattáblák

3 Gépi tanulás

4 Összetett callback függvények

5 Komponenseket irányító komponensek

K-közép klaszterezés

Az algoritmus eljárása:

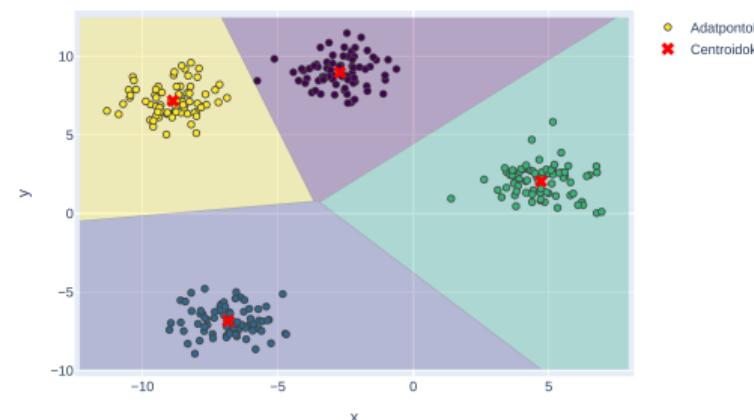
- 1 K számú klaszter centroid inicializálása véletlenszerűen:
 $\mu_1, \mu_2, \dots, \mu_K$

- 2 minden x_i adatpont a hozzá legközelebb eső klaszterhez rendelése az euklideszi távolságot használva:
 $c_i = \arg \min_j \|x_i - \mu_j\|^2$

- 3 Klaszterközpontok újraszámítása úgy, hogy az adott klaszterhez tartozó pontok várható értékét tükrözzék:

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

- 4 Ismétlés a kilépési kritériumig



Optimális klaszterszám megtalálása

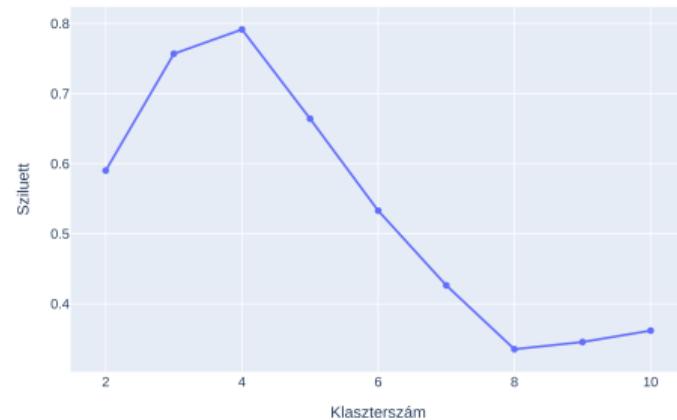
Egy klaszter konfiguráció annál jobb, minél szorosabban helyezkednek el az egy klaszterben lévő egyedek, és minél jobban elkülönülnek a más klaszterben lévő egyedektől. Ezt tükrözi a sziluett együttható:

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,
4   random_state=random_state)
5 kmeans.fit(X)
6
7 labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

Klaszter centroidok 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]]
```

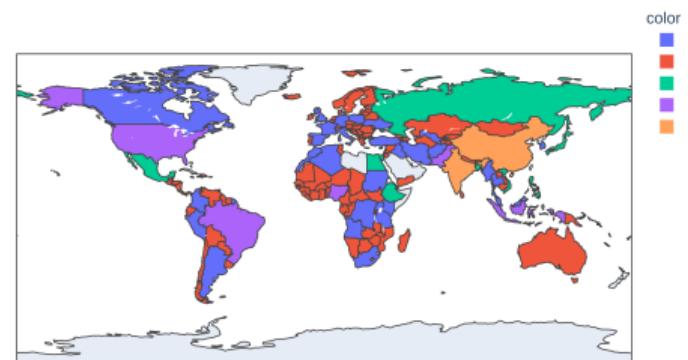
Becsült címkék az adathalmaz első 10 elemére:

```
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) &
5   poverty['is_country']]
6 data = df[indicators].values
7 kmeans.fit(data)
8
9 fig = px.choropleth(
10   df,
11   locations='Country Name',
12   locationmode='country names',
13   color=[str(x) for x in kmeans.labels_]
```



Hiányzó értékek kezelése

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

Néhány gyakori stratégia:

- `mean` (átlag): 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

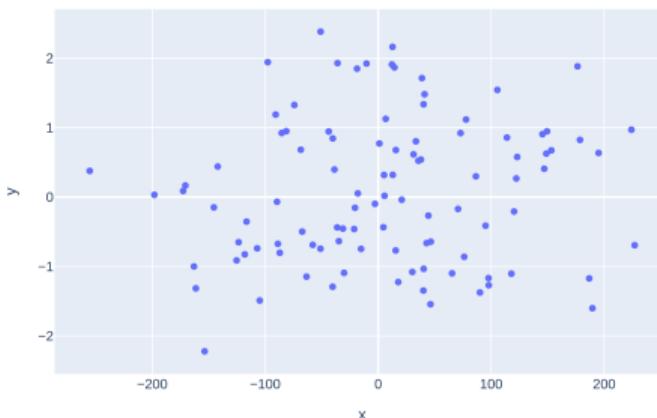
```
1 from sklearn.preprocessing import StandardScaler  
2  
3 data = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)  
4  
5 scaler = StandardScaler()  
6 print(scaler.fit_transform(data))
```

```
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

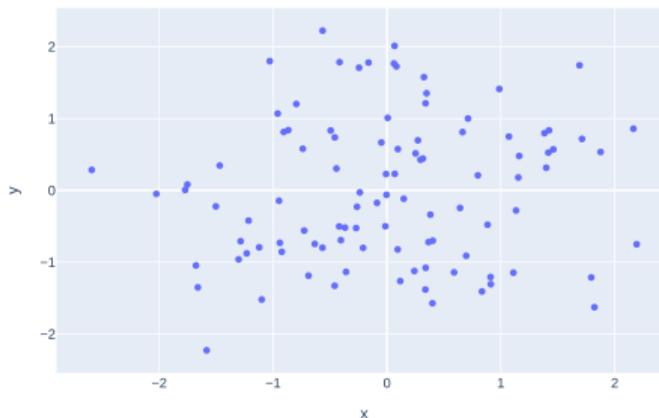
```
1 from sklearn.preprocessing import StandardScaler  
2  
3 data = np.array([1, 2, 3, 4, 5]).  
    reshape(-1, 1)  
4  
5 scaler = StandardScaler()  
6 print(scaler.fit_transform(data))
```

```
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.

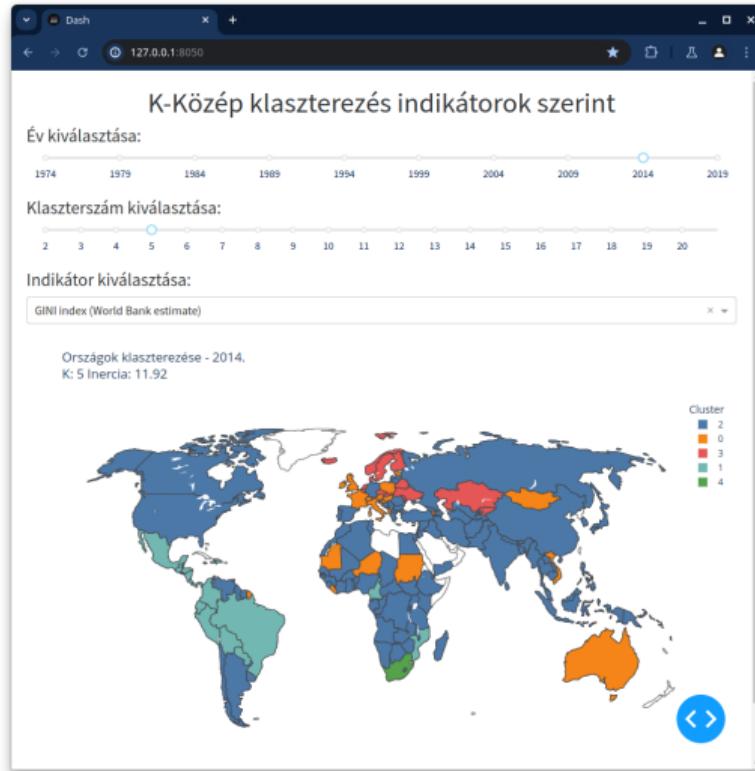
Méretezés után



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)



1 Bevezetés

2 Adattáblák

3 Gépi tanulás

4 Összetett callback függvények

5 Komponenseket irányító komponensek

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='
    kmeans_button')
3 ...
```

```
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:

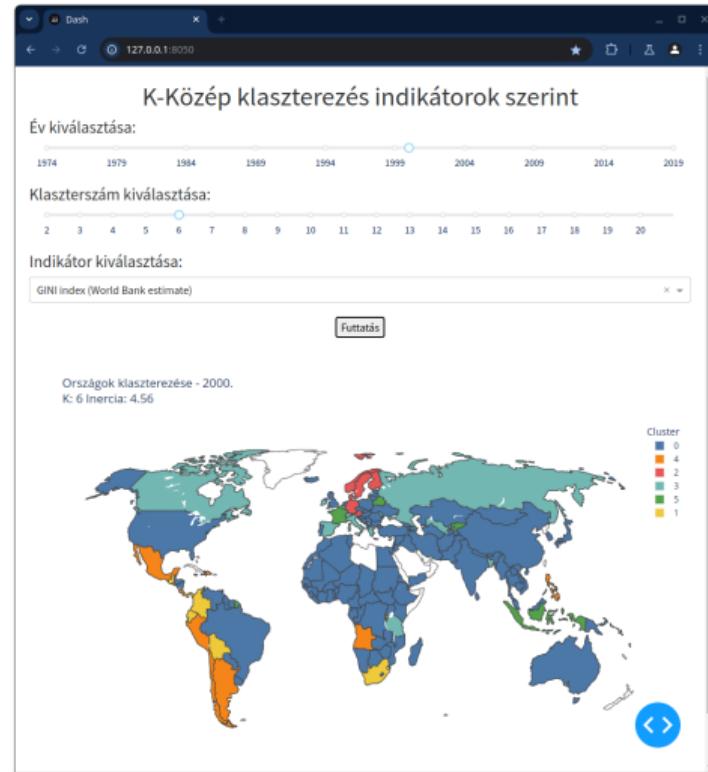
```
1  dcc.Loading([
2      dcc.Graph(
3          id='clustered_map_chart',
4      ),
5  ])
```



K-közép alkalmazás állapottal

Az alkalmazás funkcionálitásában 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.



1 Bevezetés

2 Adattáblák

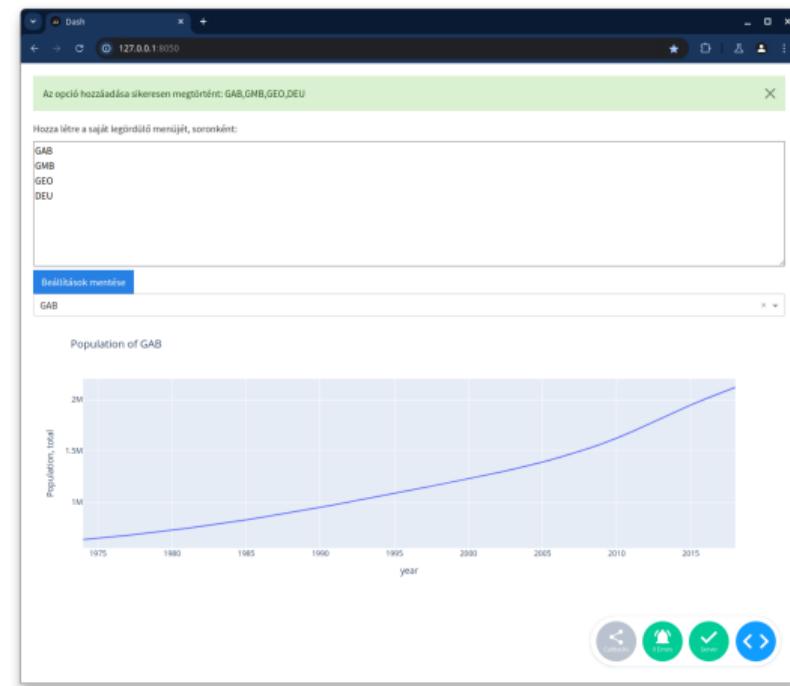
3 Gépi tanulás

4 Összetett callback függvények

5 Komponenseket irányító komponensek

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

- 1 Div létrehozása az üzenetnek:

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

- 2 dcc.TextArea szövegdoboz:

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

- 3 Gomb, ami a callback függvényt indítja:

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

- 4 Legördülő lista:

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

- 5 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.

