9. Előadás

Python kurzus

Tárgyfelelős:
Dr. Tejfel Máté

Előadó:
Dr. Király Roland

9. Előadás tematikája

Big Data és adatfeldolgozás alapjai

1. Big Data alapfogalmak
2. Adatelemzés Python eszközökkel
3. Pandas adatstruktúrák: DataFrame, Series
4. Adatok beolvasása és előfeldolgozása
5. Adatmanipulációk (szűrés, rendezés, aggregálás)
6. Adatfeldolgozási feladatok és példák

1. Big Data alapfogalmak

1.a Az adatok jelentősége, értéke mind az üzleti, mind a tudományos életben egyre fontosabbá válik. Az adatelemzés segít a vállalatoknak és szervezeteknek döntéshozataluk javításában, hatékonyságuk növelésében. Az adatvezérelt döntéshozatal ügyfél és versenytárs elemzésen és a piac változásainak gyorsabb, pontosabb követésén alapul.

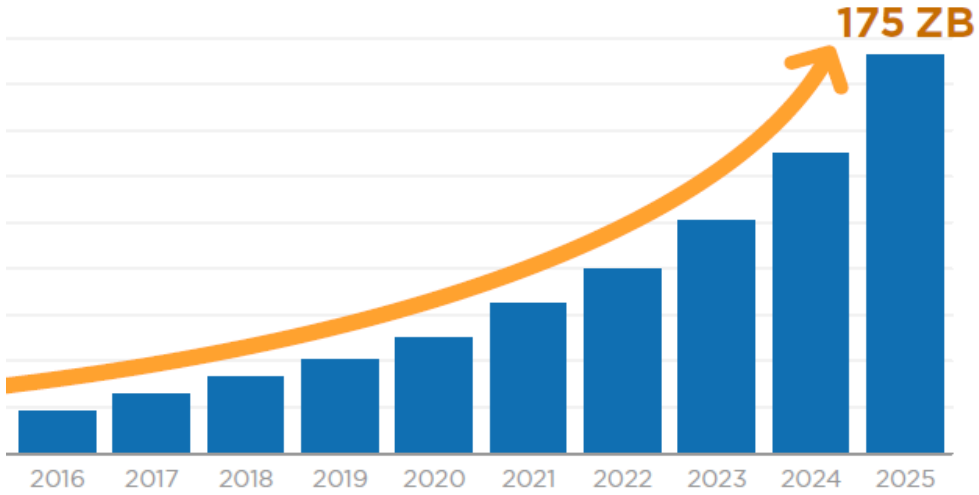
1.b Az adat típusai:

- **Strukturált adatok:** táblázatok, adatbázisok
- **Strukturálatlan adatok:** szöveges dokumentumok, e-mailek, képek és videók

1.c Adatok exponenciális növekedése:

Naponta trillió (exa) bájt adat keletkezik az internet, a szenzorok és a digitális eszközök révén.

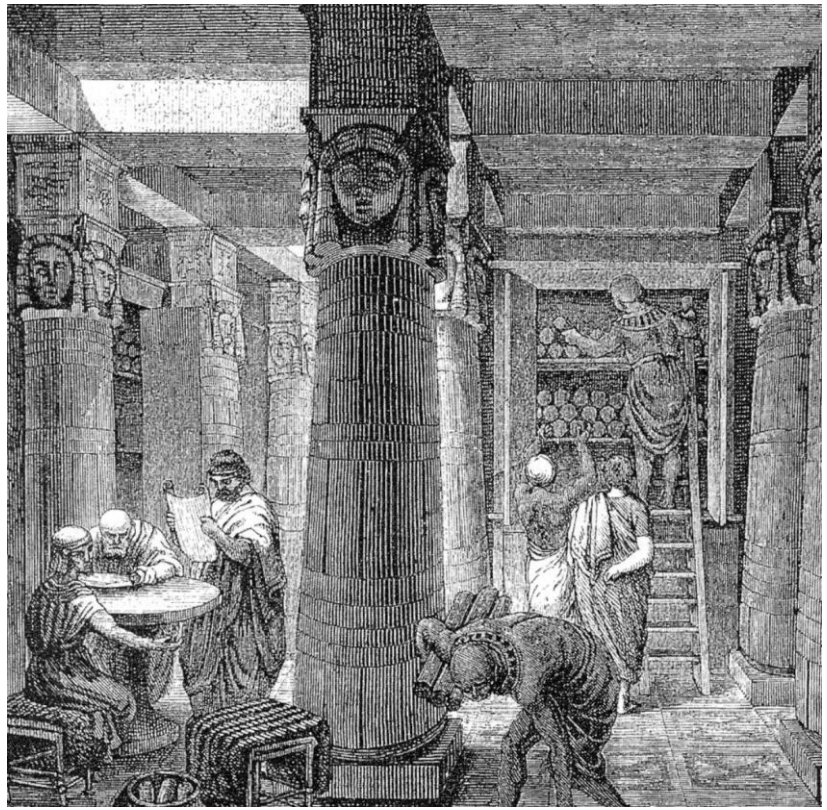
A globális adatállomány évente több, mint 50%-kal nő, és az előrejelzések szerint 2025-re eléri a 175 zettabájtot az IDC szerint.



International Data Corporation

1. Információ

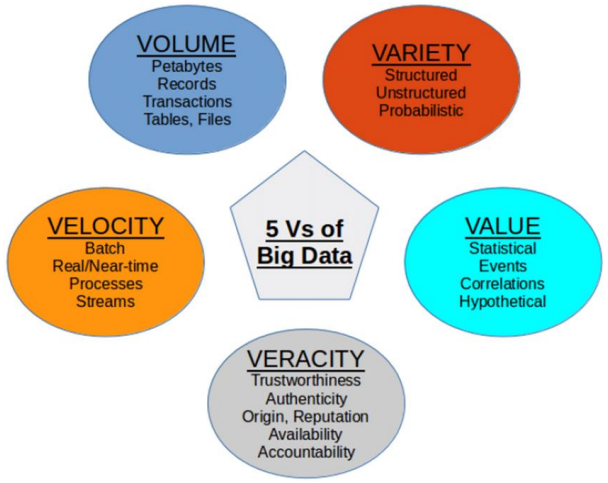
1. Beszéd, alapvető kommunikáció
2. Írás, generációk közötti tudásátadás (i.e. 8. század)
3. Könyvtárak, írott tudás megőrzése és rendszerezés (Alexandria, Muszeion, i.e 3. század)
4. Nyomtatás (1450)
5. 2020 – internet forradalma
6. Web1 (dot.com), Web2 kommunikációs forradalmak
7. 2010+ Big Data, adatelemzés automatizálása
8. Adatbányászati módszerek, HPC
9. AI alapú adatbányászati módszerek



1.d A Big Data definíciója: Olyan adathalmazok, amelyek túl nagyok vagy összetettek ahhoz, hogy a hagyományos adatfeldolgozási módszerekkel kezelhetőek legyenek.
A Big Data rendszerek lehetővé teszik a nagy mennyiségű, gyorsan áramló és különböző forrásokból származó adatok valós idejű elemzését.

Big Data jellemzői: az 5V modell szerint:

- **Volume (Mennyiség)**: óriási adatmennyiségek.
- **Velocity (Sebesség)**: az adatok gyors beérkezése és feldolgozása.
- **Variety (Változatosság)**: különböző típusú és formátumú adatok.
- **Veracity (Hitelesség)**: az adatok minősége és pontossága.
- **Value (Érték)**: az adatokból kinyerhető hasznos információ.



https://www.researchgate.net/figure/The-five-dimensions-of-Big-Data-6_fig2_358990445

Hozzáférés szerinti csoportosítás:

- **Open Data**: Nyílt adatok, amelyek mindenki számára hozzáférhetők és szabadon használhatók, pl. kormányzati statisztikák, közlekedési adatok.
- **Private Data**: Privát adatok, amelyek egy szervezet vagy egyén tulajdonában vannak, és korlátozott hozzáférésűek, pl. ügyféladatbázisok, pénzügyi jelentések.

1.e Adatkezelési eszközök:

Hagyományos adatok esetén:

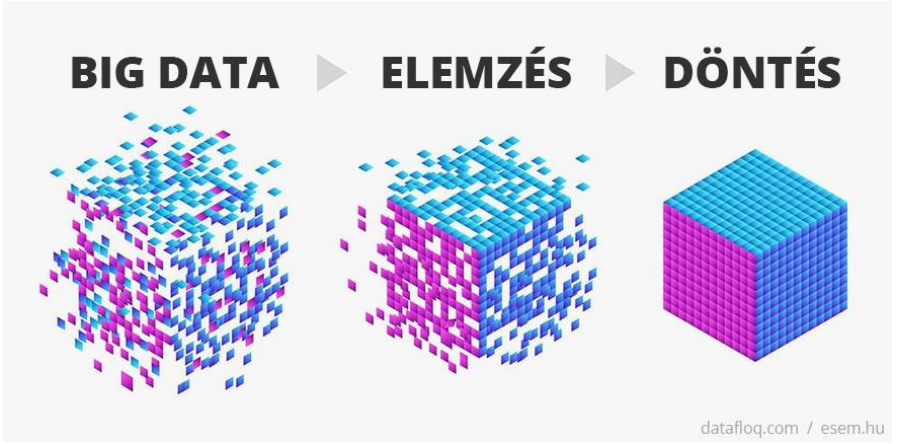
A relációs adatbázis-kezelő rendszerek (RDBMS), mint az Oracle, MySQL, SQL Server (kis- és közepes méretű adatok kezelése).

Big Data adatkezelési technológiára példák:

Strukturálatlan adatok: A Hadoop és a NoSQL adatbázisok (pl. MongoDB, Cassandra).
Strukturált adatok: SQLite, beágyazott adatbázis-kezelő rendszer.

Az adatelemzés lépései:

1. Adatok gyűjtése.
2. Adatok előkészítése és tisztítása.
3. Adatok vizualizációja.
4. Adatok elemzése.
5. Értelmezés és következtetések levonása.
6. Döntéshozatal és intézkedések végrehajtása.



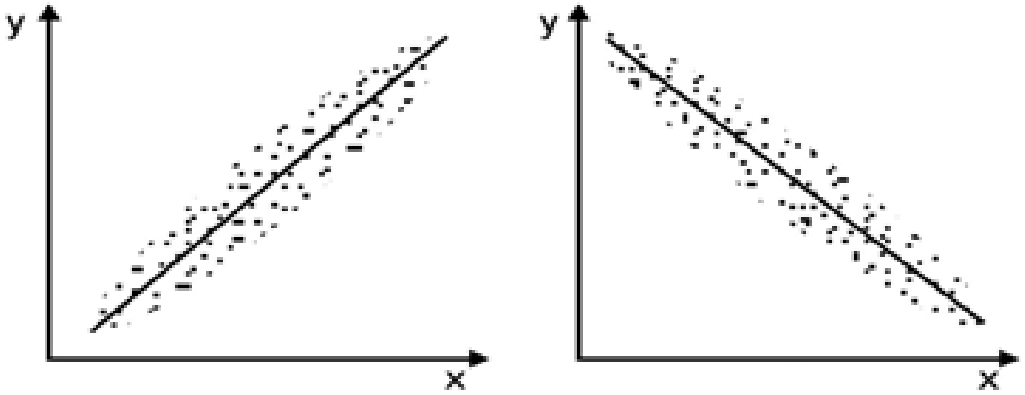
1.f Adatelemzési módszerek

Descriptive (leíró): az adatok összegzése, a meglévő információk bemutatása:

- Statisztikai elemzések (átlag, medián, szórás).
- Adatvizualizációs technikák, pl. grafikonok, diagramok.
- Adatbányászati eszközök, az adatok közti összefüggések kimutatására.

Prediktív (előrejelző): a jövőbeli események előrejelzése:

- Regressziós elemzések, klasszifikáció.
- Gépi tanulási modellek, döntési fák, neurális hálózatok.



http://www.jgypk.hu/tamop15e/tananyag_html/spss/alapfogalmak3.html

https://developers.google.com/machine-learning/clustering/overview

2. Adatelemzés Python eszközökkel

A **Python nyelvű adatelemzés és adatfeldolgozás** során a NumPy és Pandas könyvtárak alapvető eszközök. Ezek hatékony és gyors adatfeldolgozást tesznek lehetővé nagy mennyiségű adattal.

- **NumPy:** Egy hatékony, tömbökkel dolgozó könyvtár, amely nagy mennyiségű numerikus adat kezelésére és számítások végrehajtására alkalmas.
- **Pandas:** Az adatok hatékony feldolgozására és elemzésére szolgál, két fő adatstruktúrája van: **Series** (egydimenziós) és **DataFrame** (kétdimenziós).

```
# Series példa
series = pd.Series([10, 20, 30, 40],
index=['a', 'b', 'c', 'd’])

 # DataFrame példa
df = pd.DataFrame({
   'Column1': [1, 2, 3],
   'Column2': ['a', 'b', 'c']
})
```

2.1. NumPy – Numerikus számításokhoz

- A NumPy könyvtár a numerikus számítások alapja
- Tömbök használata, amelyek különböző dimenziókat támogatnak (egy-, két- vagy többdimenziós).
- Hatékony matematikai műveletek tömbökkel.
- Beépített függvények a lineáris algebrára, statisztikai számításokra, stb.

NumPy példa:

```
import numpy as np
# Egydimenziós tömb létrehozása
arr = np.array([1, 2, 3, 4, 5])
print("Egydimenziós tömb:", arr)
# Többdimenziós (mátrix) tömb
matrix = np.array([[1, 2, 3], [4, 5, 6]])
print("Mátrix:\n", matrix)
```

```
# Alapvető műveletek
arr_sum = np.sum(arr)
arr_mean = np.mean(arr)
print("Összeg:", arr_sum)
print("Átlag:", arr_mean)
```

2.2. Pandas – Adatfeldolgozás és elemzés

- A Pandas könyvtár lehetővé teszi a komplex adatstruktúrák kezelését és elemzését, legfontosabb elemei a Series és DataFrame objektumok.
- Ezt a könyvtárat használjuk a táblázatos adatok kezelésére, amelyek sorokból és oszlopokból állnak.
- Az adatok előfeldolgozását, manipulálását és megjelenítését is lehetővé teszi.

Pandas főbb elemei:

- **Series:** Egy dimenziós adatszerkezet, amely egy indexelt adattömb, hasonló a Python listákhoz vagy szótárakhoz.
- **DataFrame:** Kétdimenziós, táblázatos adatszerkezet, amely több sorból és oszlopból áll.

Pandas – Series példák

Egydimenziós adatszerkezet létrehozása és kezelése:

```
import pandas as pd
# Series létrehozása
series = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
print("Series példa:\n", series)
```

Pandas – DataFrame példák

Kétdimenziós adatszerkezetek létrehozása és kezelése:

```
# DataFrame létrehozása
data = {
   'Name': ['Anna', 'Béla', 'Cecília'],
   'Age': [29, 34, 22],
   'City': ['Budapest', 'Debrecen', 'Szeged']
}
df = pd.DataFrame(data)
print("\nDataFrame példa:\n", df)
```

```
# Adatokhoz való hozzáférés
print("\nElső sor adatai:\n", df.iloc[0])
print("\nÉletkor oszlop adatai:\n", df['Age'])
```

2.3. Műveletek – Gyakorlatok

Alapvető műveletek a DataFrame-ekkel:

- **Adatok importálása és exportálása:**
  - CSV beolvasása DataFrame-be: df = pd.read_csv('fajl_neve.csv')
  - CSV exportálás: df.to_csv('output.csv', index=False)
- **Alapvető adatvizsgálat:**
  - Az első néhány sor megtekintése: df.head()
  - Adatstatisztika: df.describe()
  - Adattípusok ellenőrzése: df.dtypes
- **Adatok manipulálása:**
  - Sorok és oszlopok kiválasztása: df[['Name', 'Age']]
  - Adatok szűrése: df[df['Age'] > 30]
  - Oszlopok hozzáadása vagy módosítása: df['NewColumn'] = df['Age'] * 2

Pandas példa

```
df = pd.read_csv('fajl_neve.csv’)
df.head()
df.describe()
df.dtypes
df[['Name', 'Age']]
df[df['Age'] > 30]

# Új oszlop hozzáadása
df['Age x2'] = df['Age'] * 2

# Szűrés: csak a 'Debrecen'-ben élők
debrecen_df = df[df['City'] == 'Debrecen']

# Átlagéletkor kiszámítása
average_age = df['Age'].mean()
print("\nÁtlagéletkor:", average_age)

df.to_csv('output.csv', index=False)
```

Az index=False eredménye, hogy az output.csv fájlban nem lesz benne az első index oszlop.

3. Pandas adatstruktúrák: DataFrame, Series

3.1. Series

A **Series** egy egydimenziós adatszerkezet, amely hasonló a Python listához vagy szótárhoz, de tartalmaz egy kapcsolódó indexet, amely minden értéket azonosít.

```
import pandas as pd

# Series létrehozása
jegyek = pd.Series([5, 4, 3, 5, 2], index=['tanuló1', 'tanuló2', 'tanuló3', 'tanuló4', 'tanuló5'])
print("Series példa:\n", jegyek)

# Adathozzáférés index alapján
print("\n'Tanuló1' jegye:", jegyek['tanuló1'])

# Adatszűrés (csak azok a jegyek, amelyek nagyobbak 3-nál)
szűrt_jegyek = jegyek[jegyek > 3]
print("\nSzűrt jegyek:\n", szűrt_jegyek)
```

```
Series példa:
tanuló1    5
tanuló2    4
tanuló3    3
tanuló4    5
tanuló5    2
dtype: int64

'Tanuló1' jegye: 5

Szűrt jegyek:
tanuló1    5
tanuló2    4
tanuló4    5
```

3.2. DataFrame

A **DataFrame** egy kétdimenziós, táblázatos adat-struktúra, sorok és oszlopok. Hasonló egy táblázathoz vagy SQL adatbázishoz, ahol az **oszlopok különböző típusú adatokat** tartalmazhatnak.

```
# DataFrame létrehozása
adatok = {
   'név': ['Anna', 'Béla', 'Cecília', 'Dávid', 'Emma'],
   'kor': [29, 34, 22, 30, 25],
   'város': ['Budapest', 'Debrecen', 'Szeged', 'Pécs', 'Miskolc']
}
diákok_df = pd.DataFrame(adatok)
print("\nDataFrame példa:\n", diákok_df)
# Oszlop kiválasztása
print("\n'Kor' oszlop:\n", diákok_df['kor'])
# Sor kiválasztása index alapján
print("\nElső diák adatai:\n", diákok_df.iloc[0])
# Adatszűrés (csak 30 év feletti diákok)
idősebb_diákok = diákok_df[diákok_df['kor'] > 30]
print("\n30 év feletti diákok:\n", idősebb_diákok)
```

```
DataFrame példa:
     név      kor  város
0  Anna     29    Budapest
1  Béla      34    Debrecen
2  Cecília 22    Szeged
3  Dávid    30    Pécs
4  Emma   25   Miskolc

'Kor' oszlop:
0   29
1   34
2   22
3   30
4   25
Name: kor, dtype: int64

Első diák adatai:
név     Anna
kor      29
város   Budapest
Name: 0, dtype: object

30 év feletti diákok:
     név      kor    város
1  Béla     34    Debrecen
3  Dávid   30    Pécs
```

3.3. DataFrame műveletek

Példa: Új oszlop hozzáadása:

```
# Új oszlop hozzáadása a diákok DataFrame-hez
diákok_df['jegyek átlag'] = [4.5, 3.8, 4.0, 4.7, 3.9]
print("\nÚj oszloppal bővített DataFrame:\n", diákok_df)
```

Példa: Adatok rendezése:

```
# Adatok rendezése 'kor' szerint csökkenő sorrendben
rendezett_df = diákok_df.sort_values(by='kor', ascending=False)
print("\nRendezett DataFrame:\n", rendezett_df)
```

3.4. Alapvető statisztikai műveletek

Példa: Átlag, maximum, szórás és módusz

```
# Átlagéletkor kiszámítása
átlag_kor = diákok_df['kor'].mean()
print("\nÁtlagéletkor:", átlag_kor)

# Legmagasabb életkor megkeresése
max_kor = diákok_df['kor'].max()
print("\nLegidősebb diák életkora:", max_kor)

# Életkorok szórásának kiszámítása
szoras = diákok_df['kor’].std()
print("\nÉletkorok szórása:", szoras)

# Életkorok móduszának kiszámítása
modusz = diákok_df['kor’].mode()
print("\nÉletkorok módusza:", modusz)
```

4. Adatok beolvasása és előfeldolgozása

Az adatok hatékony beolvasása és előfeldolgozása az adatelemzési folyamat egyik legfontosabb lépése. A **Pandas** könyvtár számos módszert biztosít különböző formátumú adatok importálására és azok előkészítésére az elemzéshez.

4.1. Adatok beolvasása

A leggyakoribb adatforrások közé tartoznak a **CSV**, **Excel** és **SQL** adatbázisok. Ezek beolvasására a Pandas különböző függvényeket biztosít:

CSV fájl beolvasása:

```
import pandas as pd
# CSV fájl beolvasása
adatok_df = pd.read_csv('pelda_adatok.csv')
print("Beolvasott adatok:\n", adatok_df.head())
```

Excel fájl beolvasása:

```
# Excel fájl beolvasása
adatok_df = pd.read_excel('pelda_adatok.xlsx', sheet_name='Adatlap1')
print("Beolvasott Excel adatok:\n", adatok_df.head())
```

Adatok betöltése SQL adatbázisból:

```
# SQL adatbázisból történő beolvasás
import sqlite3
kapcsolat = sqlite3.connect('adatbazis.db')
adatok_df = pd.read_sql_query('SELECT * FROM diakok', kapcsolat)
kapcsolat.close()
```

4.2. Hiányzó adatok kezelése

Az adatok előfeldolgozásának egyik lépése a hiányzó adatok kezelése.
A hiányzó adatok miatt **pontatlan elemzések és hibák léphetnek fel**, ezért fontos, hogy kezeljük őket.

Hiányzó adatok eltávolítása és hiányzó adatok kitöltése:

```
# Sorok eltávolítása, ahol hiányzó adat van
adatok_df = adatok_df.dropna()
print("Hiányzó adatok eltávolítása után:\n", adatok_df.head())

# Hiányzó értékek kitöltése egy adott értékkel
adatok_df['kor'] = adatok_df['kor'].fillna(30)

# Hiányzó értékek kitöltése az oszlop átlagával
adatok_df['jegyek'] = adatok_df['jegyek'].fillna(adatok_df['jegyek'].mean())
print("Hiányzó értékek kitöltése átlaggal:\n", adatok_df.head())
```

4.3. Adatok típusának konverziója

Az adatelemzési folyamat során előfordulhat, hogy az adatok típusát konvertálni kell, például szöveget számmá vagy dátumformátummá.

Típuskonverzió egész számokká és dátumformátum konverzió:

```
# 'kor' oszlop konvertálása egész számokká
adatok_df['kor'] = adatok_df['kor'].astype(int)
print("Adatok típusa konvertálás után:\n", adatok_df.dtypes)

# 'datum' oszlop konvertálása dátumformátummá
adatok_df['datum'] = pd.to_datetime(adatok_df['datum'])
```

4.4. Adatok előkészítése és adattisztítás

Az adatok előkészítéséhez gyakran szükséges az adatok szűrése, duplikált sorok eltávolítása, illetve az adatok rendezése.

Duplikált sorok eltávolítása, adatok rendezése, szűrése:

```
# Duplikált sorok eltávolítása
adatok_df = adatok_df.drop_duplicates()

# Adatok rendezése 'kor' szerint
adatok_df = adatok_df.sort_values(by='kor', ascending=True)

# Adatok szűrése, ahol a 'kor' oszlop értéke 25 feletti
fiatal_felnőttek = adatok_df[adatok_df['kor'] > 25]
print("Szűrt adatok:\n", fiatal_felnőttek.head())
```

Összegzés

Az adatok előfeldolgozása biztosítja, hogy az **adatok tiszták**, **konzisztens formátumúak** és előkészítettek legyenek a további feldolgozásra és elemzésre.

5. Adatmanipulációk (szűrés, rendezés, aggregálás)

Az adatmanipuláció a Pandas egyik legerősebb funkciója, amely lehetővé teszi az adatok hatékony feldolgozását, szűrését, rendezését és aggregálását. Ezeket a funkciókat alkalmazzuk az adatok előkészítésében és az elemzési folyamatban.

5.1. Szűrés

```
import pandas as pd
# Minta DataFrame létrehozása
adatok = {
   'név': ['Anna', 'Béla', 'Cecília', 'Dávid', 'Emma'],
   'kor': [29, 34, 22, 30, 25],
   'város': ['Budapest', 'Debrecen', 'Szeged', 'Pécs', 'Miskolc']
}
diákok_df = pd.DataFrame(adatok)

# Szűrés, ahol a 'kor' oszlop értéke 30 év felett van
idősebb_diákok = diákok_df[diákok_df['kor'] > 30]
print("30 év feletti diákok:\n", idősebb_diákok)
```

Az adatok szűrése lehetővé teszi, hogy csak a megadott feltételnek megfelelő sorokat tartsuk meg egy új DataFrame-ben.

```
30 év feletti diákok:
    név   kor    város
1   Béla  34     Debrecen
```

5.2. Rendezés

A rendezés lehetővé teszi az adatok sorainak újra sorrendezését egy vagy több oszlop alapján.

```
# Adatok rendezése 'kor' szerint növekvő sorrendben
rendezett_diákok = diákok_df.sort_values(by='kor')
print("Rendezett diákok (kor szerint növekvő sorrendben):\n", rendezett_diákok)
```

```
Rendezett diákok (kor szerint növekvő sorrendben):
    név      kor   város
2   Cecília  22    Szeged
4   Emma     25    Miskolc
0   Anna     29    Budapest
3   Dávid    30    Pécs
1   Béla     34    Debrecen
```

```
# Adatok rendezése 'név' szerint csökkenő sorrendben
rendezett_diákok_név = diákok_df.sort_values(by='név', ascending=False)
print("Rendezett diákok (név szerint csökkenő sorrendben):\n", rendezett_diákok_név)
```

5.3. Aggregálás

Az aggregálás lehetővé teszi az **adatok összesítését** különböző metrikák segítségével, például **átlag, összeg, minimum, maximum**.

```
# Adatok aggregálása a 'kor' átlagának kiszámításával
átlag_kor = diákok_df['kor'].mean()
print("Átlagos életkor:", átlag_kor)

# Adatok csoportosítása város szerint és a kor átlagának kiszámítása
városi_átlag_kor = diákok_df.groupby('város')['kor'].mean()
print("Városi átlagos életkor:\n", városi_átlag_kor)
```

```
Átlagos életkor:  28.0
Városi átlagos életkor:
város
Budapest    29.0
Debrecen    34.0
Miskolc     25.0
Pécs        30.0
Szeged      22.0
Name: kor, dtype: float64
```

5.4. Összetett példa

```
# Adatok szűrése, ahol a 'kor' nagyobb 25-nél, majd rendezés 'név' szerint
szűrt_rendezett_diákok = diákok_df[diákok_df['kor'] > 25].sort_values(by='név')
print("Szűrt és rendezett diákok:\n", szűrt_rendezett_diákok)

# Csoportosítás város szerint, majd a diákok számának megszámolása
városi_diák_szám = diákok_df.groupby('város').size()
print("Diákok száma városonként:\n", városi_diák_szám)
```

```
Szűrt és rendezett diákok:
     név      kor    város
0   Anna     29     Budapest
1   Béla     34     Debrecen
3   Dávid    30     Pécs

Diákok száma városonként:
 város
 Budapest    1
 Debrecen    1
 Miskolc     1
 Pécs        1
 Szeged      1
```

A szűrés, rendezés és aggregálás kombinálható a mélyebb elemzés érdekében.

6. Adatfeldolgozási feladatok és példák

Az adatfeldolgozási feladatok során különböző műveleteket végezhetünk a **Pandas** könyvtár segítségével, amelyek segítségével az adatokat **előkészíthetjük, módosíthatjuk**, és értékes **információkat nyerhetünk ki** belőlük.

6.1. Adattranszformációk

Az adatok transzformációja során például új oszlopok hozzáadását és a meglévő oszlopok értékeinek módosítását végezzük el.

Példa: Új oszlop létrehozása

```
import pandas as pd
# Minta DataFrame létrehozása
adatok = {
   'név': ['Anna', 'Béla', 'Cecília', 'Dávid', 'Emma'],
   'kor': [29, 34, 22, 30, 25],
   'város': ['Budapest', 'Debrecen', 'Szeged', 'Pécs', 'Miskolc']
}
diákok_df = pd.DataFrame(adatok)
# Új oszlop hozzáadása, amely a kor kétszerese
diákok_df['kor_kétszer'] = diákok_df['kor'] * 2
print("Új oszloppal bővített DataFrame:\n", diákok_df)
```

	név	kor	város	kor_kétszer
0	Anna	29	Budapest	58
1	Béla	34	Debrecen	68
2	Cecília	22	Szeged	44
3	Dávid	30	Pécs	60
4	Emma	25	Miskolc	50

6.2. Adatok normalizálása

Az adatok normalizálása során az értékeket egy közös skálára helyezzük, ami segíthet az adatok összehasonlíthatóságában.

Példa: Értékek skálázása 0 és 1 közé

```
diákok_df['kor_normalizált'] = (diákok_df['kor'] - diákok_df['kor'].min()) /
(diákok_df['kor'].max() - diákok_df['kor'].min())
print("Normalizált 'kor' oszlop:\n", diákok_df)
```

Eredmény:

	név	kor	város	kor_kétszer	kor_normalizált
0	Anna	29	Budapest	58	0.636364
1	Béla	34	Debrecen	68	1.000000
2	Cecília	22	Szeged	44	0.000000
3	Dávid	30	Pécs	60	0.727273
4	Emma	25	Miskolc	50	0.272727

6.3. Feltételes adatmanipuláció

Az adatok manipulációja során gyakran szükséges egyes sorokat vagy oszlopokat feltételek alapján módosítani.

Példa: Új oszlop létrehozása feltételes értékekkel

```
# Új oszlop, amely megjelöli, ha a kor 30 év felett van
diákok_df['30_fölött'] = diákok_df['kor'].apply(lambda x: 'Igen' if x > 30 else 'Nem')
print("Feltételes oszloppal bővített DataFrame:\n", diákok_df)
```

	név	kor	város	kor_kétszer	kor_normalizált	30_fölött
0	Anna	29	Budapest	58	0.636364	Nem
1	Béla	34	Debrecen	68	1.000000	Igen
2	Cecília	22	Szeged	44	0.000000	Nem
3	Dávid	30	Pécs	60	0.727273	Nem
4	Emma	25	Miskolc	50	0.272727	Nem

6.4. Adatok összekapcsolása és egyesítése

Az adatok összekapcsolása több forrásból származó adatok egyesítését jelenti, például **merge, join** vagy **concat** használatával.

Példa: DataFrame-ek egyesítése

```
# Második DataFrame létrehozása
tantárgyak_df = pd.DataFrame({
   'név': ['Anna', 'Béla', 'Cecília', 'Dávid', 'Emma'],
   'tantárgy': ['Matematika', 'Fizika', 'Kémia', 'Biológia', 'Irodalom']
})
# DataFrame-ek egyesítése a 'név' oszlop alapján
összekapcsolt_df = pd.merge(diákok_df, tantárgyak_df, on='név')
print("Összekapcsolt DataFrame:\n", összekapcsolt_df)
```

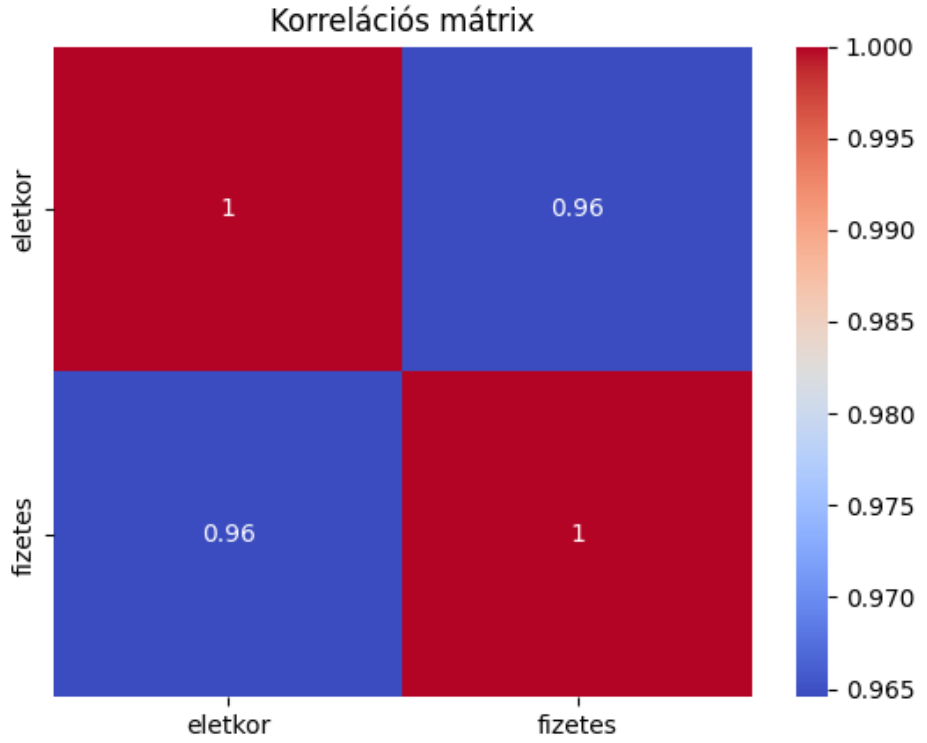
	név	kor	város	kor_kétszer	kor_normalizált	30_fölött	tantárgy
0	Anna	29	Budapest	58	0.636364	Nem	Matematika
1	Béla	34	Debrecen	68	1.000000	Igen	Fizika
2	Cecília	22	Szeged	44	0.000000	Nem	Kémia
3	Dávid	30	Pécs	60	0.727273	Nem	Biológia
4	Emma	25	Miskolc	50	0.272727	Nem	Irodalom

Példa: Adatok korrelációjának vizsgálata

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
adatok = {
   'eletkor': [23, 45, 34, 25, 32, 40, 29, 48, 37, 22],
   'fizetes': [250, 500, 300, 260, 320, 480, 290, 520, 410, 240]
}
adat_df = pd.DataFrame(adatok)
# Korrelációs mátrix kiszámítása
korrelacio = adat_df.corr()
print("\nKorrelációs mátrix:")
print(korrelacio)
# Korrelációs mátrix megjelenítése
sns.heatmap(korrelacio, annot=True, cmap='coolwarm')
plt.title('Korrelációs mátrix')
plt.show()
```

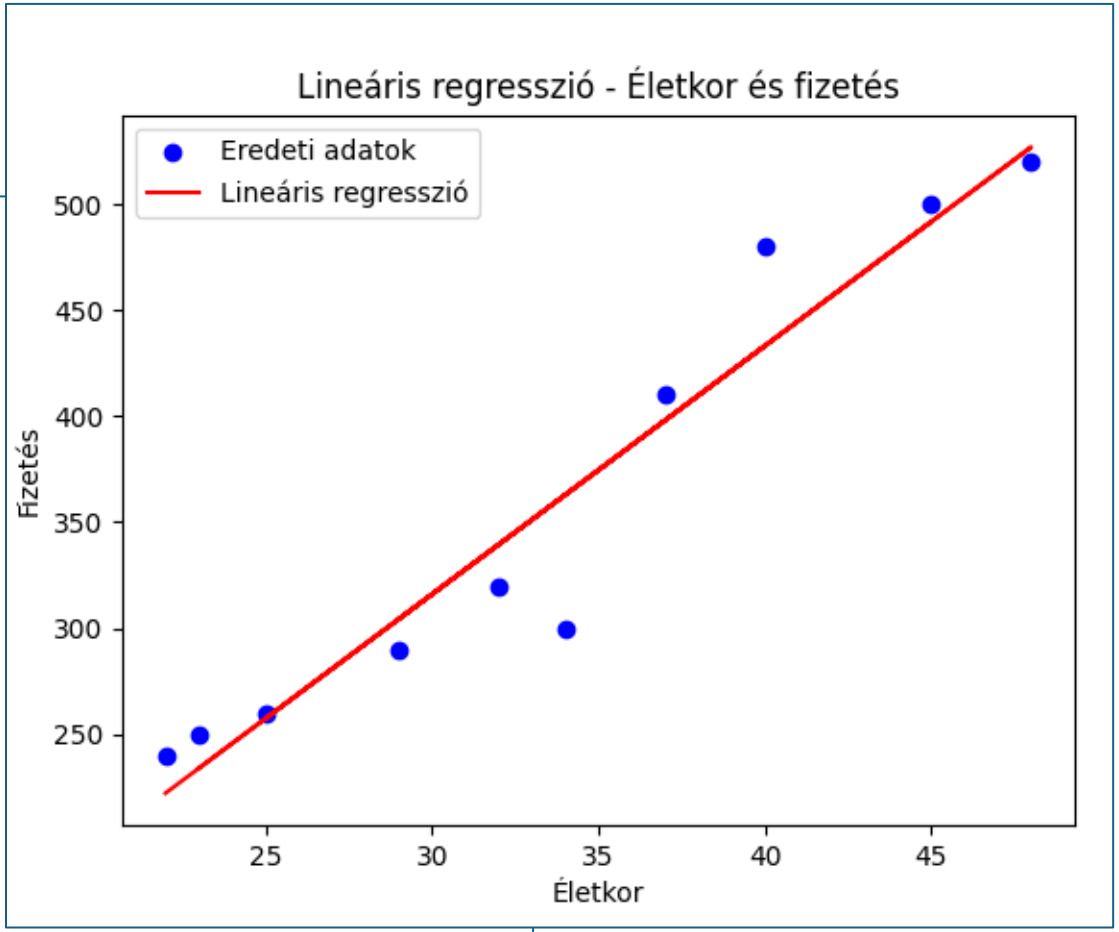
Korrelációs mátrix:

	eletkor	fizetes
eletkor	1.000000	0.964549
fizetes	0.964549	1.000000



Példa: Lineáris regresszió végrehajtása

```
from sklearn.linear_model import LinearRegression
# Független (x) és függő (y) változók kiválasztása
x = adat_df[['eletkor']]
y = adat_df['fizetes']
# Lineáris regressziós modell létrehozása
modell = LinearRegression()
modell.fit(x, y)
# Egyenes egyenletének kiírása
print(f'Y = {modell.coef_[0]:.2f} * X + {modell.intercept_:.2f}')
# Előrejelzés az adatokra
y_elorejelzes = modell.predict(x)
# Diagram rajzolása
plt.scatter(adat_df['eletkor'], adat_df['fizetes'], color='blue', label='Eredeti adatok')
plt.plot(adat_df['eletkor'], y_elorejelzes, color='red', label='Lineáris regresszió')
plt.xlabel('Életkor')
plt.ylabel('Fizetés')
plt.title('Lineáris regresszió - Életkor és fizetés')
plt.legend()
plt.show()
```



Y = 11.73 * X + -35.92

6.5. Big Data példák különböző területekről

Keresőmotorok – Google

- Napi több milliárd keresés feldolgozása
- Big Data technológiák: Hadoop, Spark, saját rendszerek
- Alkalmazások: webindexelés, keresési relevancia, gépi tanulás

E-kereskedelem – Amazon

- Vásárlói szokások elemzése és ajánlások
- Big Data technológiák: DynamoDB, Amazon ML, Hadoop, Spark
- Alkalmazások: termékajánlás, ügyfélszegmentáció, árképzés

Pénzügyi szektor – Hitelkártya-csalások

- Valós idejű tranzakciók elemzése csalások felderítésére
- Big Data technológiák: Apache Kafka, Apache Flink, gépi tanulás
- Alkalmazások: kockázatelemzés, csalásellenes rendszerek

Egészségügy – Járványok előrejelzése

- Adatok gyűjtése kórházi nyilvántartásokból és szenzorokból
- Big Data technológiák: Cloudera, Hadoop, gépi tanulás
- Alkalmazások: betegségmodellezés, állapotfigyelés, gyógyszerkutatás

Közösségi média – Facebook, Twitter

- Felhasználói tevékenységek adatainak feldolgozása
- Big Data technológiák: Hadoop, Spark, Cassandra, Storm
- Alkalmazások: sentiment analízis, trendfigyelés, hirdetéscélzás

Közlekedési szektor – Forgalomfigyelő rendszerek

- GPS, mobil adatok és közlekedési kamerák elemzése
- Big Data technológiák: Hadoop, Spark, IoT-platformok
- Alkalmazások: valós idejű forgalom-előrejelzés, útvonal-optimalizálás

Energiaipar – Intelligens mérési rendszerek

- Energiafogyasztási adatok gyűjtése és elemzése
- Big Data technológiák: Hadoop, Spark, IoT eszközök
- Alkalmazások: energiaelosztás optimalizálása, anomáliaészlelés

6.6. Klasszikus példa – A Titanic-projekt

Ez egy adatfeldolgozási gyakorlat valós adatállományon a **Big Data** és **gépi tanulás** területén. A **Titanic katasztrófa adathalmaza** a Kaggle platformon is elérhető, és tartalmazza az utasok adatait, például:

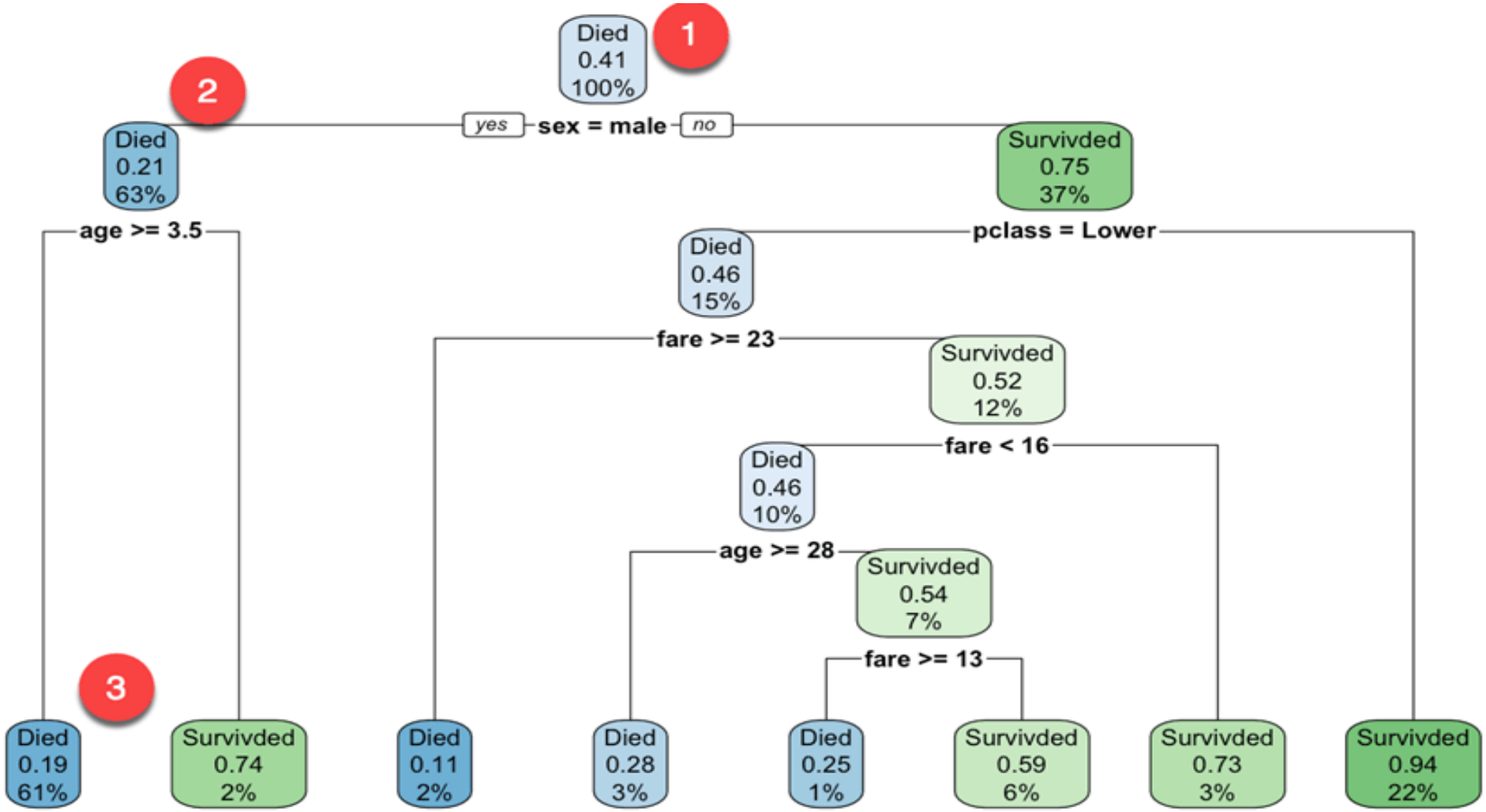
- **Név** (name)
- **Osztály** (pclass)
- **Nem** (sex)
- **Életkor** (age)
- **Testvérek/házastársak száma** a fedélzeten (sibsp)
- **Szülők/gyermekek száma** a fedélzeten (parch)
- **Jegy ára** (fare)
- **Beszállás kikötője** (embarked)
- **Túlélés** (survived, ahol 1 a túlélőket, 0 a nem túlélőket jelenti)

A feladat során az adatok elemzésével válaszokat kaphatunk arra, hogy milyen tényezők növelték az utasok túlélési esélyeit.

Adatfeldolgozási lépések a Titanic-projekthez:

1. **Adatok beolvasása és előfeldolgozása**:
   - Hiányzó adatok kezelése (pl. életkor kitöltése az átlaggal).
   - Kategóriák numerikus értékekké alakítása (pl. a „férfi” és „nő” szöveges adatok 0-ra és 1-re cserélése).
2. **Adatok elemzése**:
   - Alapvető statisztikai elemzések készítése.
   - Különböző tényezők (pl. osztály, nem) hatásának vizsgálata a túlélési arányra.
3. **Vizualizáció**:
   - Diagramok készítése a túlélési arányok bemutatására különböző csoportokban.
   - Heatmap-ek és diagramok a jellemzők közötti korrelációk megjelenítésére.
4. **Gépi tanulási modellek alkalmazása**:
   - Egyszerű algoritmusok, például döntési fa használata a túlélési esélyek meghatározására.
   - A modell kiértékelése különböző metrikákkal.

Titanic katasztrófa döntési fa



https://www.guru99.com/hu/r-decision-trees.html

Összegzés

1. Megbeszéltük a Big Data alapfogalmait
2. Bemutattuk a Pandas és a NumPy könyvtárak használatát
3. Példákat néztünk adatstruktúrák: DataFrame, Series alkalmazására
4. Bemutattuk az adatok beolvasási és előfeldolgozási lehetőségeit
5. Példákat néztünk adatmanipulációkra (szűrés, rendezés, aggregálás)
6. Adatfeldolgozási feladatokat mutattunk be

Konzultáció

Minden héten csütörtökön 18:00 – 20:00

Köszönöm a figyelmet!