

# Következtető Statisztika Python Jegyzet

Kovács László

2025-06-24



# Tartalomjegyzék

<b>1. Előhang</b>	<b>5</b>
<b>2. Statisztikához szükséges Python nyelvi alapok</b>	<b>7</b>
2.1. Programozási alapelvek . . . . .	7
2.2. A Pythonról általában . . . . .	8
2.3. A Spyder felülete . . . . .	9
2.4. Working Directory . . . . .	11
2.5. Alapvető Python adattípusok és adatszerkezetek . . . . .	11
2.6. Vezérlési szerkezetek . . . . .	27
2.7. A Pandas data frame objektum . . . . .	31
2.8. Aggregálás data frame-ben . . . . .	48
2.9. Egyszerű leíró statisztika data frame-ben . . . . .	52
2.10. Adatminőségi problémák felismerése és kezelése leíró statisztika segítségével . . . . .	55
2.11. Data frame-k összekapcsolása . . . . .	59
2.12. Kilógó értékek keresése és kezelése . . . . .	64
2.13. Korrelációs elemzések data frame-ben . . . . .	69
Gyakorló feladatok . . . . .	76
Gyakorló feladatok megoldása . . . . .	76
<b>3. Leíró Statisztika ismételés és Valószínűségyszámítás alapok</b>	<b>83</b>
3.1. Leíró statisztikai mutatók . . . . .	83
3.2. A normális eloszlás és sűrűségfüggvénye . . . . .	92

3.3. Az Exponenciális eloszlás . . . . .	106
3.4. A Varianciahányados Pythonban - Kokain a Balatonban . . . . .	111

# 1. fejezet

## Előhang

**!!!FRISSÍTENI szükséges a teljes jegyzethez!**

Ez a jegyzet hivatalosan a Budapesti Corvinus Egyetem gazdaságinformatikus hallgatóinak készült abból a célból, hogy a Statisztika II. tárgy sikeres abszolválásához szükséges Python programnyelvi elemek felelevenítésre kerüljenek.

Viszont, a jegyzet előkészítése során arra jutottam, hogy egy kicsit általánosabb célú anyagot szeretnék készíteni: egy **olyan bevezető jegyzetet a Python nyelvhez**, ami kimondottan a **statisztikai és adatelemzési feladatok elvégzéséhez szükséges elemeit és kiegészítő csomagjait mutatja be** teljesen kezdő szinten, minden **programozási előismeret feltételezése nélkül**.

Ebből adódóan szeretném kicsit tételesen összefoglalni, hogy mire számíthat az olvasó a jegyzetben. Nem szeretek zsákba macskát árulni, így szeretném már előre letisztázni, hogy ez az anyag mivel foglalkozik a Python nyelven belül, és ami talán még fontosabb, hogy mivel *nem*.

A jegyzet **bemutatja**:

- A Python nyelv legalapvetőbb utasításait és elemeit.
- A Python nyelv statisztikai-adatelemzési feladatok megoldására könnyen használható adatszerkezetait (**numpy** és **pandas** csomagok).
- A Python nyelv legalapvetőbb adatvizualizációs képességeit a **matplotlib** csomagon keresztül.
- Egy egyszerűbb, leíró statisztikai és *nem* modellezési célú adatelemzési folyamatban felmerülő leggyakoribb adatminőségi problémák azonosítási és megoldási módjait Python nyelven
- A *Spyder* fejlesztőkörnyezet működését és lehetőségeit adatelemzési feladatok megoldása során.

A jegyzetnek **nem célja**:

- Teljes körű áttekintést adni a Python nyelv elemeiről és adatszerkezetéről. A Pythont végig csak *szkriptnyelvként* használjuk, nem pedig általános célú programnyelvként.
- A Python minden lehetséges fejlesztőkörnyezetét (Jupyter Notebook, Visual Studio Code, replit.com, stb.) bemutatni.
- A `numpy`, `pandas` és `matplotlib` csomagok teljeskörű működéséről. Csak az alapvető leíró statisztikai és adatkezelési funkciókat tekintjük át.
- Teljes körű bemutatót adni a Python képességeiről az adatminőségi kihívások azonosítása és kezelése területén. Tényleg csak a legalapvetőbb és leggyakrabban előforduló problémákat tekintjük át, a legegyszerűbb kezelési módokkal.
- A Python képességeinek áttekintése a következő statisztika és a statisztikai modellezés vagy éppen gépi tanulás területén (`statmodels`, `sklearn`, `tensorflow`, stb. csomagok)

Ezen kívül a jegyzet **feltételez némi leíró statisztikai előismeretet**. Konkrétan a következő fogalmak ismeretét veszem adottnak:

- ismérv és azok mérési skálái
- átlag, szórás
- medián és egyéb kvantilisok
- hisztogram, doboz ábra
- eloszlások alakja
- korreláció és pontdiagram

**Ha** egy lelkes **gazdaságinformatikus hallgató forgatja a jegyzetet**, aki a *Programozás alapjai* tárgyban a Python nyelv és a `pandas` data frame-k alapjait már tökéletesen elsajátította biztosan sok „uncsi” részt fog találni a jegyzetben. Számára **leginkább csak a 8-13. fejezetek gyors átprögetését ajánlom**, hogy a statisztikai számítások elvégzéséhez és a statisztikai adattáblák kezeléséhez szükséges Python megoldásokat felelevenítse. Ugyanakkor fontos megjegyeznem, hogy a **Statisztika II. tárgy tanóráin a jelen jegyzetben szereplő tudást ismertnek feltételezzük, és nem fogunk rá külön kitérni az oktatás során!!** Szóval, ha valaki egy kicsit is bizonytalan a Python ismereteiben, azért olvassa át ezt az ismétlő jegyzetet, és ha bármi nem világos, **KÉRDEZZEN!!**

## 2. fejezet

# Statisztikához szükséges Python nyelvi alapok

### 2.1. Programozási alapelvek

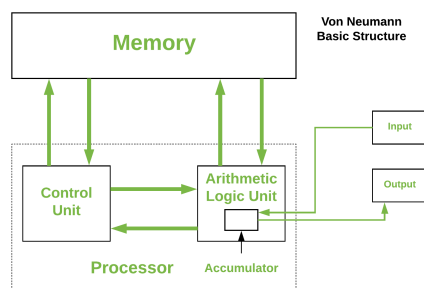
Mivel a Python egy programnyelv, így elengedhetetlen, hogy a használata előtt némi programozási alapvetésekkel tisztában legyünk.

Talán az már kijelenthető, hogy közismert a tény, mi szerint a mai számítógépek alapvetően a Neumann-elvek szerint működnek.

A mi szempontunkból ez csak annyit jelent, hogy a számítógépet alapvetően *utasítások* végrehajtására használjuk programozás során: pl. számold ki ezt, vagy rajzold ki amaszt. A programozás kihívása, hogy a gépállat felfogása nagyon nehéz, ezért az utasításokat nagyon konkrétan meg kell neki fogalmazni. Ehhez a megfogalmazáshoz adnak segítséget a különböző programnyelvek, így a Python is.

A Neumann-elvek szerint a programnyelven kiadott utasításokat a számítógépben a *processzor* (Central Processing Unit, CPU) hajtja végre. Ugyanakkor az utasítások végrehajtásához a gépnek adatokat is fejben kellhet tartania (mondjuk átlag számítás során nem árt tudnia milyen számok átlagát számoljuk ki). Ezeket az adatokat nem meglepő módon a *memóriájában* (Random Access Memory, RAM) tárolja a gép. A gépállattal való kommunikációhoz szükség van valami beviteli = input eszközre (billentyűzet, egér) és az utasítások eredményének megjelenítéséhez kell egy kimeneti = output eszköz (monitor) is.

És...ennyi! Alapvetően a modern számítógépek ennyi alkotóelemből állnak (a háttértár programozás szempontjából irreleváns). Mindez egy cuki ábrán (a processzor belső felépítése minket jelenleg nem érdekel):



Számítógép vásárlás szempontjából is alapvetően a CPU és a RAM határozza meg mennyire gyors a gép: minél nagyobb a CPU órajele (GHz) és minél több magja van, annál több utasítást tud végrehajtani a gép egy adott idő alatt, és minél nagyobb a RAM mérete (GB) annál több adatot tud egyszerre fejben tartani. Talán nem ér minket meglepetésként, ha azt mondom, hogy a *statisztikai számítások alapvetően RAM igényesek* (mert sok adattal dolgoznak). 16-32 GB már kell, hogy komolyabb statisztikai modelleket gyorsan tudjunk futtatni egy valós vállalati adattáblán (ami általában több, mint 1 millió rekorddal és minimum 30-40 oszloppal = változóval rendelkezik).

## 2.2. A Pythonról általában

Az Python a jegyzet írásakor a legnépszerűbb általános célú programozási nyelv, 2024 januárjában a TIOBE index alapján a legtöbb sor programkódot Python nyelven írják a fejlesztők.

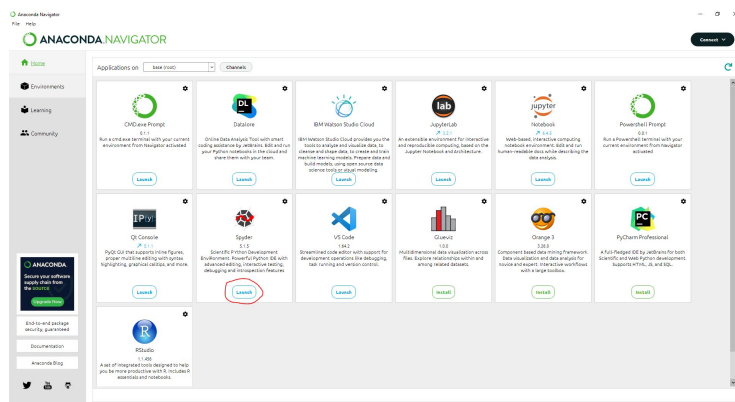
A mi szempontunkból a Python olyan szempontból vonzó, hogy a külső kiegészítő csomagjai segítségével a valószínűségszámítás, statisztika és általánosabb adatelemzés műveletei könnyen és gyorsan elvégezhetők a segítségével. Tehát a Python használható olyan matematikai, statisztikai modellezési és elemzési feladatok elvégzésére alkalmas szkriptnyelvként, mint például az R vagy a Matlab. A Python előnye ezekkel a nyelvekkel szemben, hogy mivel általános célú programnyelv, így a matematikai-statisztikai számítások eredményei sokkal könnyebben integrálhatók egy üzleti célú alkalmazásba, ami mondjuk felhasználói felülettel rendelkezik. Ahogy az *Előhangban* már jeleztem, a jegyzet kimondottan a Python statisztikai és adatelemzési funkcióinak alapszintű bemutatásával foglalkozik. Tehát alkalmazást fejleszteni itt nem fogunk, a Pythont szkriptnyelvként működtetjük: elküldjük a programkódban megírt számítási igényeinket a gépállatnak, és az visszaköpi a számítások eredményeit a képernyőre, és mi egyrészt gyönyörködünk bennük, másrészt értelmezzük az eredményeket. Viszont, jó tudni, hogy a Python az eredmények további felhasználására is képes programnyelv. Ebben több, mint egy matematikusi körökben szintén népszerű R vagy Matlab. Hátránya a felsorolt nyelvekkel szemben, hogy



mivel általános célú programnyelv, és nem kimondottan a matematikai-statisztikai számításokra optimalizált, így számos számítás lekódolása sokkal körülményesebb Pythonban, mint R-ben vagy Matlabban. De hát *valamit valamiért.* :)

A Python, mi az **Anaconda keretrendszer**ből működtetjük, ami innen letölthető. Az Anaconda számos fejlesztőkörnyezetet biztosít a Python nyelvhez. Itt megjegyzendő, hogy a **Python és a Python fejlesztőkörnyezete nem összekeverendő!** A Python maga a programnyelv, amiben kódot írunk a számítógépünknek, hogy hajtsa végre, míg a fejlesztőkörnyezet az a program, amiben ezt a kódot megírjuk! Mi a Python kódjainkat **Spyder fejlesztőkörnyezetben** írjuk majd, mivel ez a fejlesztőkörnyezet az, ami leginkább a Python matematikai-statisztikai műveleteket végrehajtó, szkriptnyelv-szerű használatára van optimalizálva.

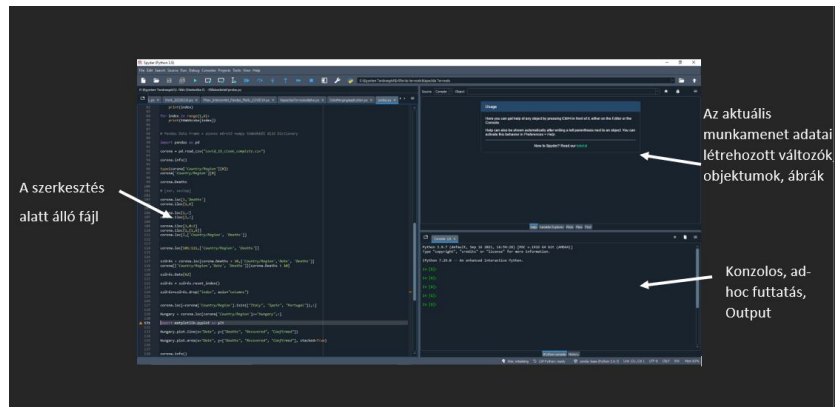
Miután telepítettük és elindítottuk az Anaconda keretrendszert, a kezdőképernyőről rögtön indíthatjuk is a Spydert:



## 2.3. A Spyder felülete


A Spyder fejlesztőkörnyezet indítása után az alábbihoz hasonló képernyőkép fogad minket:

## 102. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK




A Python kódokat a Spyder-ben `.py` kiterjesztésű szkriptfájlokban fogjuk írni. Egy ilyen az alább látható módon lehet létrehozni:

A szkriptfájlba írhatjuk a gépállatnak szóló utasításainkat Python nyelven.

Az utasítások végrehajtását a Spyder felület felső részén lakó  gomb megtaposásával tudjuk kérni a géptől, aki az utasítás eredményét alul, a *Console* felületen köpi ki. Ekkor a Spyder mindig azt a Python utasítást hajtja végre a gomb megnyomásakor, amiben éppen a villogó kurzorral álltunk. Egy példában számoltassuk ki a Pythonnal, hogy mennyi  $3 + 2$ :

Több utasítást is végre tudunk hajtani a géppel egyszerre. Csak jelöljük ki a szkriptben a végrehajtandó utasításokat, és így kijelölés után tapossuk meg a

Spyder felső menüsorában található  gombot! Egy utasítást több sorba is írhatunk, de egy új utasítás mindig új sorban kezdődjön! Érdemes egy üres sort is hagyni az előző utasítás vége után!

Számoltassunk akkor most ki a Pythonnal egyszerre két dolgot is: mennyi  $3 + 2$  és mennyi  $3 \times 2$ :

Ezek után a jegyzetek további részében a feladatok elvégzéséhez szükséges Python kódrészleteket és azok eredményét az alábbi módon jelölöm:

```
3+2
```

```
## 5
```

```
3*2
```

```
## 6
```

Ha az összes `.py` fájlunkban lévő kódot le szeretnénk futtatni abban a sorrendben,



ahogy a fájlban szerepelnek, akkor a Spyder felső menüsorán a gombot kell megütni. De vigyázzunk, ilyenkor a Python nem írja ki az utasításaink eredményét a konzolra, csak akkor, ha külön beágyazzuk őket egy `print` nevű extra utasítás zárójelei közé!

```
print(3+2)
```

```
## 5
```

```
print(3*2)
```

```
## 6
```

A művelet videón:

## 2.4. Working Directory

Mielőtt belevágunk a Python mélyebb rejtelseibe van még egy fontos dolog, amiről még szót kell ejteni: a *Working Directory* kérdéséről. A *Working Directory* az a mappa, ahonnan a *pythonállat* alapértelmezés szerint minden fájlt innen akar a memóriába tölteni és ide akar visszaírni. A Spyder jobb felső sarkában lévő részen lehet kiválasztani és beállítani, hogy melyik mappa legyen a *Working Directory*. Ezek után **minden fájlunk alpból ide fog mentődni, és minden adattáblát ide rakjunk be, amivel majd a Pythonban dolgozni akarunk!**

A Spyder-ben alapértelmezett *Working Directory*-t is be tudunk állítani, ha elbandukolunk a **Tools** -> **Preferences** -> **Current working directory** menübe, és ott a **Console directory / The following directory** című résznél beállítjuk a kívánt fix mappát alapértelmezett *Working Directory*-nak.

Az egész alapértelmezett *Working Directory*-val kapcsolatos okfejtés működésben megnézhető a következő videón:

## 2.5. Alapvető Python adattípusok és adatszerkezetek

Eddig a Pythonban csak utasításokat hajtottunk végre, de a memóriában (RAM-ban) nem tároltattunk el még vele semmit. Most itt az idő! Az utasítások eredményét a `=` szimbólummal tudjuk a memóriába valamilyen szimpatikus néven elmenteni.

### 2.5.1. Egyszerű adattípusok

Mentsük el a  $3+2$  eredményét egy `összeg` névre hallgató R objektumba: `összeg = 3+2`. Az utasítás végrehajtásának hatására az `összeg` objektum megjelenik az Spyder *jobb felső* sarkában lévő résznél, a *Variable Explorer* fülön. (A Python alapjáraton karakterkészletben elég bő, így simán tudunk ékezetes objektumneveket is adni. De néha én a biztonság kedvéért megmaradok az ékezet nélküli elnevezéseknél. Öreg vagyok már, na! :)). A Spyder képernyőnek ezen a *Variable Explorer* részén látjuk mindig azt, hogy éppen milyen Python objektumok élnek a RAM-ban:

A Python memória-objektumoknak több fajtája van. A legegyszerűbbek azok, amik csak egy értéket tartalmaznak (mint nekünk az előbb az `összeg`). Ezeket szokás **változónak** is hívni. Én nem szeretem ezt az elnevezést, mert keverhető a statisztikai értelemben vett változóval, ami mindig egy statisztikai megfigyelést leíró tulajdonságot/ismérvet jelent (pl. munkavállaló jövedeleme). Ennek ellenére én is gyakran használom a Python memória objektumokra a változó elnevezést. :)

A Python objektumoknak mindig van **adattípusa** is, ami **leírja, hogy az adott objektumban számértékű, szöveges, dátum vagy valami egyéb jellegű adatot tárolunk-e**. Ez azért marha fontos, mert az adattípustól függ, hogy mennyi hely szükséges a RAM-ban az objektum tárolásához. Érzésre megmondható talán, hogy egy szöveges adat tárolására több hely kell, mint egy egész szám tárolásához.

Az adattípusokat Pythonban a `type` névre hallgató **beépített függvénnyel** lehet lekérdezni.

Ezen a ponton érdemes megemlékezni arról, hogy a Pythonban léteznek **függvényként működő beépített utasítások** is. Ezek az úgynevezett Python függvények olyan utasítások, amik a matekban megszokott  $f(x)$  függvény alakot veszik fel. A függvény neve leírja, hogy a függvény milyen műveletet végeztet el a gépállattal, és a zárójelek között pedig megadjuk, hogy milyen bemeneti paramétereken (adatokon) kell elvégezni a kijelölt műveletet. Pl. ilyen függvény volt már a `print` is.

Gyakorlati példaként lássuk akkor **Python függvényekre a `type` működését**:

```
összeg = 3+2
type(összeg)
```

```
## <class 'int'>
```

Ez a `type` nevű jószág azt csiripeli nekünk, hogy ez az `összeg` című változó egy `int` adattípusú, azaz **egész szám**, leánykori angolos nevén **integer**. Tehát, ha egy Python objektum `int` adattípusú, akkor az azt jelenti, hogy ő bizony csak egész számokat tud elképzelni a világban, tört számokat nem tud tárolni.

## 2.5. ALAPVETŐ PYTHON ADATTÍPUSOK ÉS ADATSZERKEZETEK 13

Törtszámok tárolására vannak a `float` adattípusú objektumok.

```
tört = 3/2
type(tört)
```

```
## <class 'float'>
```

A Statisztika II. tárgyban a kétféle számítpus közti különbségnek nem igazán lesz jelentősége, de ha igazi *big data*-val foglalkozik az ember, akkor a RAM takarékoság miatt számít, hogy valami csak egy egész számnyi, vagy egy törtszámnyi helyet foglal sok-sok tizedeshellyel!

Néhány egyéb fontosabb adattípus és megadási módjuk:

```
szoveg = "Hello There!" # Figyeljünk rá, hogy szöveget a kódban csak idézőjelek közé rakjunk! Mi
type(szoveg)
```

```
## <class 'str'>
```

```
igazhamis = True
type(igazhamis) # a bool típusnak csak két értéke lehet: True vagy False
```

```
## <class 'bool'>
```

A fenti kódrészletben szereplő `#` jel a komment jele a Pythonban. A `#` mögötti részeket a gépállat nem fogja végrehajtani, olyan lesz neki, mintha ott sem lenne. Ezzel magunknak írhatunk a Python szkriptbe hasznos megjegyzéseket.

Az egyes adattípusok között tudunk konvertálni, ha van ennek van értelme. A konverzóra függvényeket tudunk használni, amik neve kivétel nélkül megegyezik azzal a kulcsszóval, amit a `type` függvény visszaad adattípusnak. Tehát a függvény, ami mondjuk az objektumot szöveggé, azaz stringgé konvertálja az `str` névre hallgat.

Tehát akkor számból tudunk szöveget csinálni:

```
szam = 1992
type(szam)
```

```
## <class 'int'>
```

```
nemszam = str(szam)
type(nemszam)
```

```
## <class 'str'>
```

## 142. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
nemszam
```

```
## '1992'
```

Láthatjuk, hogy amikor kiíratjuk a `nemszam` változót, akkor ott az 1992 már aposztrófok között van, ami azt jelöli, hogy ez beza már szöveges, azaz *string* adat.

Olyan szövegből tudunk számot csinálni, aminek tartalma tényleg egy valid szám:

```
szöveg = "1992"  
type(szöveg)
```

```
## <class 'str'>
```

```
nemszöveg = int(szöveg)  
type(nemszöveg)
```

```
## <class 'int'>
```

Tizedestörtekekkel vigyázzunk! A **Python angol lokalizációt feltételez mindig**, így tizedes pontot kell alkalmazni! A tizedes vesszővel írt számot nem fogja felismerni, és hisztis hibaüzenetet dob. :( A `nemjo` változó pedig nem jön létre a memóriában.

```
nemjo = float("3,14")
```

```
## ValueError: could not convert string to float: '3,14'
```

```
nemjo
```

```
## NameError: name 'nemjo' is not defined
```

```
type(nemjo)
```

```
## NameError: name 'nemjo' is not defined
```

De ha a törtszám tizedes ponttal adott a string változóban, akkor az minden további nélkül `float` adattípusra konvertálható.

```
jo = float("3.14")
jo
```

```
## 3.14
```

```
type(jo)
```

```
## <class 'float'>
```

Ha törtszámot (`float`) etetünk meg vacsorára az `int` függvénnyel, akkor annak az egészrészét veszi.

```
fura = int(3.14)
fura
```

```
## 3
```

```
type(fura)
```

```
## <class 'int'>
```

Pythonban a rosszul beállított adattípusokból születhet pár baleset. Íme a leggyakoribb példák.

A `+` két string között az összefűzést jelenti, így az alábbi kód tökéletesen működőképes.

```
szoveg1 = "Hello"
szoveg2 = "There"

szoveg1+szoveg2
```

```
## 'HelloThere'
```

Viszont a string és integer összege nem értelmezhető, így hibaüzi lesz a vége...mily meglepő :) Ellenben a szorzatuk értelmes eredményt mutat: a stringet összefűzi annyiszor amennyi az integer típusú változó értéke!

```
szoveg = "3"
szam = 4

szoveg+szam
```

## 162. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
## TypeError: can only concatenate str (not "int") to str
```

```
szoveg*szam
```

```
## '3333'
```

Érdeemes megnézni mi történik, ha egy stringként tárolt egész számot és egy integerként tárolt egész számot úgy „adunk össze” és „szorzunk össze”, hogy előtte a stringet integerré, vagy floattá konvertáljuk.

```
szoveg = "3"  
szam = 4  
  
int(szoveg)+szam
```

```
## 7
```

```
int(szoveg)*szam
```

```
## 12
```

```
float(szoveg)+szam
```

```
## 7.0
```

```
float(szoveg)*szam
```

```
## 12.0
```

Ekkor igazából semmi galiba nem történik, minden szituáció értelmes eredményre vezet. Annyi, hogy amikor `float`-ra konvertáltuk a stringben tárolt egész számot, akkor az eredmény is `float` típusú objektum lesz. Ezt onnan látni a `type` függvény nélkül, hogy pl. a `float(szoveg)+szam` eredménye 7.0 lesz a `int(szoveg)+szam`-féle 7 helyett.

Viszont, ha a stringben egy tizedes törtet tárolok el (rendesen tizedes ponttal), akkor az már összeveszik az `int` függvénnyel, és ezekben a csillagálásokban hibát dob a pythonállat.

```
szoveg = "3.5"  
szam = 4  
  
szoveg+szam
```



## 2.5. ALAPVETŐ PYTHON ADATTÍPUSOK ÉS ADATSZERKEZETEK 17

```
## TypeError: can only concatenate str (not "int") to str
```

```
szoveg*szam
```

```
## '3.53.53.53.5'
```

```
int(szoveg)+szam
```

```
## ValueError: invalid literal for int() with base 10: '3.5'
```

```
int(szoveg)*szam
```

```
## ValueError: invalid literal for int() with base 10: '3.5'
```

```
float(szoveg)+szam
```

```
## 7.5
```

```
float(szoveg)*szam
```

```
## 14.0
```

Ha egészrészt szeretnék venni a stringként tárolt törtszámomból, akkor az `int` alkalmazása előtt beza `float`-tá kell konvertálni.

```
szoveg = "3.5"
```

```
szam = 4
```

```
szoveg+szam
```

```
## TypeError: can only concatenate str (not "int") to str
```

```
szoveg*szam
```

```
## '3.53.53.53.5'
```

```
int(float(szoveg))+szam
```

```
## 7
```

```
int(float(szoveg))*szam
```

```
## 12
```

```
float(szoveg)+szam
```

```
## 7.5
```

```
float(szoveg)*szam
```

```
## 14.0
```

## 2.5.2. Összetett adatszerkezetek

### 2.5.2.1. A Python list

Egyszerre több értéket tartalmazó objektumot is fel tudunk venni a Python memóriájába, ha [] zárójelek között vesszővel felsoroljuk az eltárolandó értékeket. Ennek az objektumnak a neve `list`.

```
sokszam = [3.14, 2.71, 88, 1234]
type(sokszam)
```

```
## <class 'list'>
```

Írassuk ki a teljes listát egyben az outputra!

```
sokszam
```

```
## [3.14, 2.71, 88, 1234]
```

Kérjük le a lista 1. és 3. elemeit! Egy elemet a listából a sorszámaival tudunk kinyerni, ha ezt a sorszámot [] zárójelek között megadjuk. Ugyanakkor figyeljünk, hogy a Pitonállat 0-tól indexel! Azaz, az 1. elem a 0.; 2. az 1.; 3. a 2. és stb.

```
sokszam[0]
```

```
## 3.14
```

```
sokszam[2]
```

```
## 88
```

Nézzük meg az adattípusait is ezeknek a listaelemeknek.

```
type(sokszam[0])
```

```
## <class 'float'>
```

```
type(sokszam[2])
```

```
## <class 'int'>
```

Láthatjuk, hogy a lista megőrzi az elemeinek eredeti adattípusát. Tehát, a 3.14 adattípusa `float`, míg a 88-é `int`. Ezzel sokat spórol a memóriánkon, hogy nem kényszeríti át a 88-at is `float`-ba az egységesség jegyében.

Ez a logika szövegekkel is működik. Ha felveszek egy szöveges értéket is a listába, akkor annak az adattípusa string, azaz `str` lesz. A számértékű adatok pedig maradnak annak rendje és módja szerint `float` és `int` típusban, ami éppen kell. :)

```
sokszam_sokszoveg = [88, 42, "Hello", 1992, 9, "There", "Friend", 11]
```

```
type(sokszam_sokszoveg[0])
```

```
## <class 'int'>
```

```
type(sokszam_sokszoveg[2])
```

```
## <class 'str'>
```

Kérjük le, hogy egy lista hány elemet tartalmaz. Ezt a `len` nevű függvény intézi nekünk.

```
len(sokszam_sokszoveg)
```

```
## 8
```

## 202. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

8 elemű a lista, szupszi!

Viszont, ezt az elemszám lekérdezést meg lehet oldani úgynevezett **metódus** segítségével is! A **metódusok olyan függvények, amik egy konkrét memóriában élő objektumon hajtanak végre műveleteket**. Ez a spéci logika a Python nyelvben úgy jelenik meg, hogy **nem azt mondjuk**, hogy  $f(x)$  módon végrehajtom az  $f$  műveletet az  $x$  objektumon. Mint ahogy a `len(sokszam_sokszoveg)` is működik, \*\*hanem úgy gondolkodunk, hogy  $x.f()$  módon végrehajtjuk az  $x$  objektumon az  $f$  műveletet. Ez a gyakorlatban a `sokszam_sokszoveg` nevű lista elemszámának lekérdezésénél az alábbi módon működik.

```
sokszam_sokszoveg.__len__()
```

```
## 8
```

Királyság, az eredmény így is tök 8. :) A legtöbb metódus nevében amúgy nincsenek ilyen hosszúságú `__` részek. Illetve, a metódusok zárójelei közé lehet majd egyéb paramétereket is írni, amik szabályozzák a metódus működését. Ilyen például a lista elemeinek sorbarendeziési művelete.

Egy listát sorba rendezni ugyanis már csak metódussal tudunk, ami `sort` néven fut. Ezt kell elsűtni a listánkon egy kis pontocskával megtámogatva.

```
sokszam.sort()
sokszam
```

```
## [2.71, 3.14, 88, 1234]
```

Szépen növekvő sorban vannak már itt a számaink. Viszont **BRÉKÓ** van, mert a `sort` metódus **felülírta az eredeti listát, tehát az értékek eredeti sorrendje elveszett!** Ha **szükségünk van az eredeti sorrendre**, akkor beza **másolatot kell készíteni az eredeti listából a rendezés előtt!** Ezt a másolat készítést a `copy` metódussal tudjuk megtenni. Ha ezt nem alkalmazzuk, akkor a gépállat olyan szinten kezeli az új objektumot is, hogy mindent megcsinál vele, amit az eredetivel! Ha ezt a kapcsolatot a másolat és az eredeti objektum között *el akarjuk vágni*, akkor kell a `copy` metódus.

```
sokszam = [3.14, 2.71, 1234, 88]
sokszam_copy = sokszam.copy()
sokszam.sort()
sokszam
```

```
## [2.71, 3.14, 88, 1234]
```

```
sokszam_copy
```

```
## [3.14, 2.71, 1234, 88]
```

Láthatjuk, hogy a fenti példában minden oké, megvan az eredeti sorrend is a `sokszam_copy`-ban. De itt lentebb, ha leahagyom a `copy`-t, akkor GázGéza van!

```
sokszam = [3.14, 2.71, 1234, 88]
sokszam_copy = sokszam
sokszam.sort()
sokszam
```

```
## [2.71, 3.14, 88, 1234]
```

```
sokszam_copy
```

```
## [2.71, 3.14, 88, 1234]
```

Viszont, ha csökkenő és nem növekvő sorrendet akarok a listában, akkor azt a `sort` metódus zárójelei között, **paraméterként tudom megadni** `reverse=True` módon.

```
sokszam.sort(reverse=True)
sokszam
```

```
## [1234, 88, 3.14, 2.71]
```

Oké, ez működik! :) Azt, hogy egy metódusnak vagy általános függvénynek milyen paraméterei vannak, azt pl. a Python nyelv `w3schools-on` található online dokumentációjából lehet kideríteni. Itt a metódus/függvény nevére kell rákeresni. Szép szóval azt szokás mondani, hogy a dokumentáció megadja, hogy az egyes Python függvényeket milyen paraméterezéssel (más néven argumentumokkal) lehet *meghívni*.

A sorbarendezés működik csak stringeket tartalmazó listára is.

```
soknév = ["Kovács", "László", "Balázsné", "Mócsai", "Andrea", "Musa"]
soknév.sort()
soknév
```

```
## ['Andrea', 'Balázsné', 'Kovács', 'László', 'Musa', 'Mócsai']
```

## 222. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
soknév.sort(reverse=True)
soknév
```

```
## ['Mócsai', 'Musa', 'László', 'Kovács', 'Balázs', 'Andrea']
```

Ellenben, ha a listában vegyesen vannak stringek és valami számértéket jelölő adattípusok (int és float), akkor a rendezés vége egy szép kis hibaüzenet lesz.

```
sokszam_sokszoveg = [88, 42, "Hello", 1992, 9, "There", "Friend", 11]
sokszam_sokszoveg.sort()
```

```
## TypeError: '<' not supported between instances of 'str' and 'int'
```

Na ezt a rendezősdit vegyes adattípusokon már tényleg nem érti a gépállat! Tanulság: rendezés esetén nem iszunk kevertet! :)

Nézzük meg hogyan tudunk több, mint 1 elemet kiválasztani a listákból!

Kérjünk le minden elemet 2-től 4-ig. Ezt úgy tudjuk megtenni, hogy a lista neve után [] zárójelek között :-tal elválasztva megadjuk a kiválasztás kezdeti és végső sorsszámát: *kezdet:vége*. Azonban **vigyázzunk! A kezdeti végpontot zárt, a végsőt nyílt** intervallumként értelmezi a Pitonállat! Tehát ennek szellemében, ha figyelembe vesszük a 0-val kezdődő indexszálást is, akkor a 2-től 4-ig tartó listaelemeket (2-1):4 = 1:4 módon kell megadni.

```
sokszam_sokszoveg[1:4]
```

```
## [42, 'Hello', 1992]
```

A hecc kedvéért nézzük meg mit ad vissza gépállat, ha nem létező elemet kérünk le.

```
sokszam_sokszoveg[9]
```

```
## IndexError: list index out of range
```

Csak, hogy emlékezzünk arra, hogy az `IndexError: list index out of range` hibaüzenet nem létező listelem kiválasztását jelenti. :)

### 2.5.2.2. A Python Dictionary

A Python Dictionary típusú (`dict`) objektuma nemes egyszerűséggel egy olyan list, amiben **szöveges kulcsokkal és nem sorszámmal indexeljük a listaelemeket**.

Létrehozni az értékek és a szöveges azonosítók (azaz kulcsok) megadásával tudjuk "kulcs" : "érték" módon `{}` zárójelek között.

Hozzunk létre egy `Laci` nevű `dict`-et, ami tartalmazza Laci 3 legfontosabb ismervét: vezeté- és keresztnévét, valamint születési évét.

```
Laci = {"vezetek": "Kovács",
"kereszt": "László",
"year": 1992}
Laci
```

```
## {'vezetek': 'Kovács', 'kereszt': 'László', 'year': 1992}
```

```
type(Laci)
```

```
## <class 'dict'>
```

Kérjük le a 2. elemet a szótárból.

```
Laci[1]
```

```
## KeyError: 1
```

Teljesen jogosan nyávog a pitonkénk, hogy ezt nem érti, hiszen itt a 2. elem nem értelmezhető, mivel nem sorszámmal azonosíthatók a szótár elemei.

De ezt az alábbi hivatkozást a szöveges kulcson keresztül érteni fogja.

```
Laci["kereszt"]
```

```
## 'László'
```

Milyenek az adattípusok?

```
type(Laci["vezetek"])
```

```
## <class 'str'>
```

```
type(Laci["kereszt"])
```

```
## <class 'str'>
```

```
type(Laci["year"])
```

```
## <class 'int'>
```

Nagyszerű! Mint a listában, minden elem őrzi szépen az eredeti adattípusát. :)

### 2.5.3. A numpy tömb

Nekünk azért jó, mert úgy van optimalizálva ez az adatszerkezet, hogy az elemein a **statisztikai számítások** (átlag, medián, szórás, stb.) **nagy adattömegben is gyorsan** fussanak!

Ehhez már külön csomag kell, ami **numpy** névre hallgat.

Külső csomagokat a Pythonhoz a **pip install** utasítás segítségével tudunk telepíteni. A **numpy** csomagot tehát a következő kóddal lehet felvarázsolni Pitonkánknak.

```
pip install numpy
```

Ezt a **fenti kódot csak egyszer kell lefutatni**, utána a Python mindig **emlékezni fog** van már neki egy **numpy** névre hallgató **kiegészítő csomagja**.

Viszont, a **következő kódot mindig futtassuk le** **mielőtt egy kódban a numpy csomagot használni akarjuk!**

```
import numpy as np
```

Minden **numpy** függvényt a fenti kódsor miatt egy **np** előtaggal tudunk majd csak használni. Szakkifejezéssel élve, a fenti kódsorral a **numpy** függvényeket az **np névtérbe** töltöttük be a gépállat számára *úgymond*.

Hozzunk is létre **numpy tömböt!** Ezt úgy tudjuk megtenni, hogy egy **[]** zárójelekkel létrehozott listát berakunk egy az **np** névtérben lakó **array** nevű függvény zárójelei közé.

```
tömböcske = np.array([3.14, 2.67, 88, 1234])
type(tömböcske)
```



## 2.5. ALAPVETŐ PYTHON ADATTÍPUSOK ÉS ADATSZERKEZETEK 25

```
## <class 'numpy.ndarray'>
```

```
tömböcske
```

```
## array([ 3.14, 2.67, 88. , 1234. ])
```

Láthatjuk is, hogy a `tömböcske` adattípusa `numpy`-féle `ndarray`, azaz tömb. :)

Egy `numpy` tömböt amúgy lehet már létező listából történő klónozással is létrehozni, szintén az `np.array` függvénnyel.

```
sokszam = [3.14, 2.67, 88, 1234]
tömböcske = np.array(sokszam)
type(tömböcske)
```

```
## <class 'numpy.ndarray'>
```

```
tömböcske
```

```
## array([ 3.14, 2.67, 88. , 1234. ])
```

Az elemek kiválasztása szerencsénkre ugyanúgy megy, mint `list`-ben.

Pl. az első és negyedik elemek lekérdezése az alábbi.

```
tömböcske[0]
```

```
## 3.14
```

```
tömböcske[3]
```

```
## 1234.0
```

Másodiktól Negyedik elemig történő kiválasztás.

```
tömböcske[1:4]
```

```
## array([ 2.67, 88. , 1234. ])
```

És itt olyat is lehet, hogy csak a 2. és 4. elemet szedjük ki! A sima `list` ezt pl. nem igazán tudja! Ehhez az kell, hogy a kiválasztott sorszámokat egy `list`-ként, `[]` zárójelekkel létrehozva adjuk meg az indexszeléshez használt szögletes zárójelek között!

## 262. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
tömböcske[[1,3]]
```

```
## array([ 2.67, 1234.  ])
```

Tehát, a fenti példában azért van `[[[]]]` használat, mert a külső `[]` a tömb indexelése miatt van, a belső `[]` pedig a kiválasztott sorszámok listája miatt kerül a képbe.

De mik itt az egyes elemek adattípusai? Lessük meg őket!

```
tömböcske[0]
```

```
## 3.14
```

```
tömböcske[3]
```

```
## 1234.0
```

```
type(tömböcske[0])
```

```
## <class 'numpy.float64'>
```

```
type(tömböcske[3])
```

```
## <class 'numpy.float64'>
```

**BRÉKÓ! A numpy tömbök elemeinek mindig azonos adattípusúnak kell lenniük!** Ha alapból nem azok, akkor a gépállat átkonvertál mindent a legáltalánosabb adattípusra. Az intek és floatok esetében ez a törtszámokat is elviselő float.

Ellenben ha van szöveg is a dologban...

```
szöveges_tömb = np.array(sokszam_sokszoveg)  
type(szöveges_tömb)
```

```
## <class 'numpy.ndarray'>
```

```
szöveges_tömb
```

```
## array(['88', '42', 'Hello', '1992', '9', 'There', 'Friend', '11'],  
##       dtype='<U11'>)
```

```
type(szoveges_tomb[0])
```

```
## <class 'numpy.str_'>
```

```
type(szoveges_tomb[2])
```

```
## <class 'numpy.str_'>
```

...akkor bizony a legáltalánosabb adattípus, amit minden elem megörököl az a string, vagyis `str`!

## 2.6. Vezérlési szerkezetek

### 2.6.1. Elágazás (if)

Az elágazások arra az esetre vannak, ha **bizonyos utasításokat a kódunkban csak akkor akarunk végrehajtani, ha előtte valamiféle logikai feltétel teljesül.**

Például, ha egy egész szám nagyobb, mint 10 kiírjuk, hogy *'Hatalmas'*.

Vagyis létrehozunk egy új változót `szam` néven, megnézzük, hogy az értéke nagyobb-e mint 10, és ha igen, akkor egy `print` függvénnyel kiírjuk tényleg a *'Hatalmas'* szócskát. Mindez Pythonul az alábbi módon néz ki.

```
szam = 13

if szam > 10:
    print("Hatalmas")
```

```
## Hatalmas
```

Mivel a számunk 13 volt, és  $13 > 10$ , így ki lett írva, hogy *'Hatalmas'*. De ha a számnak 8-at adunk meg, akkor értelemszerűen nincs kiírás.

```
szam = 8

if szam > 10:
    print("Hatalmas")
```

## 282. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

Azt figyeljük meg a fenti két kódrészletben, hogy a vizsgálandó logikai feltételt egy `if` kulcsszóval adjuk meg, majd utána egy `:`-ot írunk, és a feltétel esetén futtatandó kódot egy TAB billentyűs behúzással kezdjük a következő sorban.

**Figyelem!** Ha **nincs behúzás, akkor a pythonka mindenképp végrehajtja az utasítást, ami következik** az `if`-el kezdődő sor után!

```
szam = 8

if szam > 10:
    print("Hatalmas")

print("Ezt mindenképp kiírjuk!")
```

## Ezt mindenképp kiírjuk!

Még olyat is tudunk csinálni egy `else` kulcsszóval, hogy ha az `if`-ben megadott logikai feltétel nem teljesül akkor is kiírunk valamit. Pl. most a feltétel nem teljesülése esetén írjuk ki azt, hogy *'Törpe'*

```
szam = 3

if szam > 10:
    print("Hatalmas")
else:
    print("Törpe")
```

## Törpe

```
print("Ezt mindenképp kiírjuk!")
```

## Ezt mindenképp kiírjuk!

Ahogy a fenti kódrészlet eredménye is mutatja, **az `else` esetén is kell a behúzás az egyéb esetben végrehajtandó kódokhoz**, hogy azt csinálja nekünk a pythonállat, amit szeretnénk.

Az elvégezhető logikai összehasonlító műveletek az `if` feltételekben egy `a` és `b` objektum között a következők a Python nyelvén:

- Egyenlő: `a == b`
- Nem egyenlő: `a != b`
- Kisebb: `a < b`
- Nagyobb: `a > b`

- Legalább: `a <= b`
- Legfeljebb: `a >= b`

Az *egyenlő* és *nem egyenlő* természetesen működik `str`-ek esetében is, a többi viszont csak `int` és `float` adattípusú objektumokra értelmes, amúgy *hibára futnak*.

### 2.6.2. Ciklusok (for)

Alapvetően többféle ciklus van a programnyelvekben, de minekünk igazából csak a `for` ciklusra lesz szükségünk.

A `for` ciklus egy általunk éppen `aktualis_elem`-nek elnevezett objektumot pörget végig egy lista vagy `numpy` tömb minden elemén. Ezzel így ki tudjuk olvasni egyesével egy tömb vagy lista minden értékét. A „*végigpörgetés*” alias „*ciklizálás*” során végrehajtandó kódot hívjuk **ciklusmagnak**, és ezt a kódrészletet az `if`-hez hasonló módon **behúzással kell elválasztani** a kód többi részétől!

Lássuk hát a dolgot akció közben!

```
sokszam_sokszoveg = [88, 42, "Hello", 1992, 9, "There", "Friend", 11]

tömböcske = np.array([3.14, 2.67, 88, 1234])

for aktualis_elem in sokszam_sokszoveg:
    aktualis_elem
```

```
## 88
## 42
## 'Hello'
## 1992
## 9
## 'There'
## 'Friend'
## 11
```

```
print("-----")
```

```
## -----
```

```
for aktualis_elem in tömböcske:
    aktualis_elem
```

## 302. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
## 3.14
## 2.67
## 88.0
## 1234.0
```

Egy `range` keresztségű függvénnyel bármilyen számsorozatot is ki tudunk íratni egy `for` ciklusban. Csak arra figyeljünk, hogy ez a függvény is 0-tól kezd a léptetést, mint a listák és a tömbök! :).

Írjuk ki az egész számokat 0-tól 5-ig...ehhez a `range` függvénybe 6-ot kell írni paraméternek!

```
for aktualis_elem in range(6):
    aktualis_elem
```

```
## 0
## 1
## 2
## 3
## 4
## 5
```

Minden egész szám kiírása 2-től 8-ig az alábbi módon lehetséges. Itt a `range`-ben is a felső határ egy nyílt intervallumként megadható, mint a listaelemek `:-os` kiolvasása esetén (5.2.1. fejezet).

```
for aktualis_elem in range(2,9):
    aktualis_elem
```

```
## 2
## 3
## 4
## 5
## 6
## 7
## 8
```

Tömb elemeit ezzel a `range` függvénnyel kiolvashatjuk a cikluson belül a sorszámuk (indexszük) segítségével is akár.

```
elemszám = len(sokszam)

for aktualis_index in range(elemszám):
    sokszam[aktualis_index]
```

```
## 3.14
## 2.67
## 88
## 1234
```

De remélem érezzük, hogy ez kellően körülményes megoldás ahhoz képest, mintha közvetlenül a tömböt járnánk be a ciklussal. :)

## 2.7. A Pandas data frame objektum

Adattáblákat Pythonban **pandas** csomag **data frame** struktúrájában kezeljük!

Ezt, mint külön csomagot egyszer telepíteni kell.

```
pip install pandas
```

Aztán minden kódunk elején behivatkozni, ha használni akarjuk.

```
import pandas as pd
```

Adatvizualizációhoz a **pandas** csomaggal együttműködni képes **matplotlib** csomagra lesz szükség.

Itt is egyszer telepíteni kell.

```
pip install matplotlib
```

Aztán minden kódunk elején behivatkozzuk, ahol használni akarjuk.

```
import matplotlib.pyplot as plt
```

Itt is figyeljünk a **névterekre**, amikbe elraktuk a csomagok függvényeit!

Ezen a ponton jegyezném meg, hogy a **pandas** csomag olyan hatalmas, hogy függvényeinek és metódusainak külön dokumentációja érhető el. **Ha egy függvény vagy metódus használatakor elakad az ember, érdemes először ebben a dokumentációban utána nézni a problémás cucc működésének!**

Olvassuk be a `covid_19_clean_complete.csv` fájlt, és tároljuk le az adatait egy **corona** nevű Pandas data frame-ben!

Az adatfájl egy WHO által készített historikus kimutatás a COVID-19 vírus esetszámairól a Föld országaiban 2020. április 30-cal bezárólag.

## 322. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

A **pandas** data frame-ekbe a legkönnyebben talán **csv** kiterjesztésű állományként tárolt adattáblákat lehet beolvasni a **read\_csv** függvény segítségével. A **csv** állományok valójában olyan **txt** fájlok, amikben egy táblázat szerepel úgy, hogy az oszlophatárokat **vesszők** jelzik! Innen is a név: *comma separated values = csv*

**Figyelem!** Az alábbi beolvasó kód csak akkor működik, ha a *csv* fájlt az aktuálisan beállított **Working Directory**-ba másoltuk be!!

```
corona = pd.read_csv('covid_19_clean_complete.csv')
```

A **data frame** logikailag úgy kezelhető, mint egy **numpy tömbökből álló lista**, de a listaelemeket névvel is tudjuk azonosítani, mint egy **Dictionary-ben!!** Ezek a listaelem nevek az oszlopok = változók = ismérvek nevei! Gondoljunk bele, hogy ez mennyire logikus, hiszen a **numpy** tömbök elemeire vonatkozó azonos adattípus követelmény megfeleltethető a statisztikai ismérvek mérési skáláinak fogalmával!

Az előző bekezdésben írtakból kifolyólag a betöltendő *csv* fájllal szemben vannak a **pandas** csomagnak fontos előkövetelményei:

- az oszlopok vesszővel elválasztottak
- a tört számok tizedes pontot használnak
- a szöveges adatok idézőjelek között vannak

Ha angol nyelvű oldalról töltünk le adatokat *csv*-ben (pl. Kaggle), akkor a fenti követelményeknek szinte biztosan meg fognak felelni. Rosszul viselkedő *csv*-k esetén pedig a **read\_csv** függvény különböző paramétereivel kezelhetők a problémák (tizedespont vs tizedes vessző pl.). Részletek a függvény dokumentációjában.

A beolvasandó adattáblától a **pandas** elvárja azt a logikai felépítést, hogy a tábla soraiban legyenek a statisztikai megfigyelési egységeink (emberek, országok, lakások, autók stb.) és az oszlopokban pedig a megfigyeléseket leíró tulajdonságok, azaz változók (ember kora, ország GDP-je, autó márkája stb.).

Nézzük meg ezt az adatstruktúrát a **data frame info** módszerének segítségével!

```
corona.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 26400 entries, 0 to 26399
## Data columns (total 8 columns):
## #   Column              Non-Null Count  Dtype
## ---  ---
##
```



```
## 0 Province/State 8000 non-null object
## 1 Country/Region 26400 non-null object
## 2 Lat 26400 non-null float64
## 3 Long 26400 non-null float64
## 4 Date 26400 non-null object
## 5 Confirmed 26400 non-null int64
## 6 Deaths 26400 non-null int64
## 7 Recovered 26400 non-null int64
## dtypes: float64(2), int64(3), object(3)
## memory usage: 1.6+ MB
```

Itt tehát úgy néz ki, hogy *egy sor = egy földrajzi alrégió (mivel a Province kisebb egység, mint a Country) egy adott napon mért koronavírus adatai*. Ezek az adatok az adott napig *kumulált* esetszám, elhunytak száma és gyógyultak száma. Ez a fenti adatokból már nem derül ki, ez a WHO dokumentációjából jön az adattáblához. :) Amint látszik összesen 26400 sorunk és 8 oszlopunk, azaz ismervünk/változónk van.

Az `info` metódus eredményéből viszont látszik az is, hogy a **Province/State** oszlopban csak 8000 nem *null* (hiányzó érték) bejegyzés (azaz sor) van! Ez amiatt lehet, mert a kisebb országokat nem bontották szét az adatgyűjtők a WHO-nál alrégiókra, és így ezeknél az országoknál a **Province/State** oszlopot üresen hagyták.

Ami az `info` metódus eredményéből még érdekes, hogy a **string adattípust a pandas data frame object-nek hívja!** Ehhez hozzá kell szokni. :) Onnan jön az elnevezés, hogy általános programnyelvekben az **object** a legáltalánosabb adattípus, adatelemzésben pedig a legáltalánosabb mérési skála, amire mindent át lehet konvertálni az a szöveges adatok (*stringek*) nominális mérési skálája.

Nézzük meg a betöltött *adattábla* = *data frame* első pár rekordját A `head` metódusával. Alapból az első 5 sort írja ki.

```
corona.head()
```

```
## Province/State Country/Region Lat ... Confirmed Deaths Recovered
## 0 NaN Afghanistan 33.0000 ... 0 0 0
## 1 NaN Albania 41.1533 ... 0 0 0
## 2 NaN Algeria 28.0339 ... 0 0 0
## 3 NaN Andorra 42.5063 ... 0 0 0
## 4 NaN Angola -11.2027 ... 0 0 0
##
## [5 rows x 8 columns]
```

Itt az elején még valószínűleg nagyon 2020 elején vagyunk, így ne meglepő, hogy Afganisztánban és ilyen A betűs afrikai országokban még 0 eset (és így 0 halott, 0 gyógyult) van.

Ami érdekes még, hogy a **Province/State** oszlopban ilyen NaN kódok vannak, amik a hiányzó értékeket jelölik. Ugye az **info** metódusból tudjuk ugyebár, hogy ebben az oszlopban jó sok,  $26400 - 8000 = 18400$  érték szerepel, így nem meglepő, amit itt a **head**-ben látunk. :)

A data frame-nek nem csak metódusai vannak, hanem tulajdonságai, **property**-jei is! Ezeket is ponttal tudjuk lekérni, csak nem kell a végére zárójel.

Pl. egy tulajdonság az oszlopnevek listája. Ezt egy numpy tömbben adja majd vissza a gépszellem.

```
corona.columns
```

```
## Index(['Province/State', 'Country/Region', 'Lat', 'Long', 'Date', 'Confirmed',
##        'Deaths', 'Recovered'],
##        dtype='object')
```

Kérjük le, hogy mi az első és a hatodik oszlop neve!

```
corona.columns[0]
```

```
## 'Province/State'
```

```
corona.columns[5]
```

```
## 'Confirmed'
```

### 2.7.1. Hivatkozási lehetőségek data frame-ben

Egy-egy konkrét oszlopot, mint *property* is ki tudunk választani a nevének keresztül.

```
corona.Confirmed
```

```
## 0      0
## 1      0
## 2      0
## 3      0
## 4      0
##      ..
## 26395   6
## 26396  14
## 26397   6
## 26398   1
## 26399  15
## Name: Confirmed, Length: 26400, dtype: int64
```

De az is működik, ha azt mondjuk, hogy a data frame nem más, mint egy Dictionary, aminek az elemei `numpy` tömbök, és kiválasztjuk az elemet a listából a nevén (oszlopnév) keresztül.

```
corona["Confirmed"]
```

```
## 0      0
## 1      0
## 2      0
## 3      0
## 4      0
##      ..
## 26395   6
## 26396  14
## 26397   6
## 26398   1
## 26399  15
## Name: Confirmed, Length: 26400, dtype: int64
```

Itt jegyzem meg, hogy a `pandas` egy-egy oszlop adattípusát nem `numpy` tömbnek, hanem `Series`-nek hívja, de logikailag és technikailag is ez a `Series` ugyan úgy működik, mint a `numpy` tömbök.

```
type(corona.Confirmed)
```

```
## <class 'pandas.core.series.Series'>
```

Ha egy konkrét elem, pl. a 19000. sor értéke érdekel minket, akkor azt a megfelelő oszlop kiválasztása után szintén `[]`-vel tudjuk kikeresni, hiszen a kiválasztott oszlop maga egy `numpy` tömbként kezelhető `Series`, mint láttuk korábban.

```
corona.Confirmed[19000-1]
```

```
## 3
```

```
corona["Confirmed"][19000-1]
```

```
## 3
```

De a data frame-et `[kiválasztott sor, kiválasztott oszlop]` módon is tudjuk hivatkozni a `loc` és `iloc` metódusokkal. A különbség a kettő között,

## 362. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

hogy az `iloc` esetben az oszlopot a sorszámával, míg a `loc` esetben a nevével tudjuk kicsalogatni a jégre.

Szóval, a következő két kód *ugyan azt az elemet olvassa ki* a data frame-ből.

```
corona.iloc[19000-1, 5]
```

```
## 3
```

```
corona.loc[19000-1, "Confirmed"]
```

```
## 3
```

A `loc` és `iloc` hivatkozási módokban a `:` szimbólummal ki tudunk választani egész sorokat és oszlopokat is.

```
corona.iloc[:, 5]
```

```
## 0      0
## 1      0
## 2      0
## 3      0
## 4      0
##      ..
## 26395   6
## 26396  14
## 26397   6
## 26398   1
## 26399  15
## Name: Confirmed, Length: 26400, dtype: int64
```

```
corona.loc[19000-1, :]
```

```
## Province/State      NaN
## Country/Region      Malawi
## Lat                -13.254308
## Long                34.301525
## Date                4/2/20
## Confirmed           3
## Deaths              0
## Recovered           0
## Name: 18999, dtype: object
```

A `loc` és `iloc` segítségével egyszerre több oszlopot is ki tudunk választani pl.

Készítsünk egy dataframe-t a corona-ból, ami csak az országok nevét, a dátumot és a COVID-19 megerősített eseteinek, halottainak és gyógyultjainak számát tartalmazza.

```
corona.loc[:,['Country/Region', 'Date', 'Confirmed', 'Deaths', 'Recovered']]
```

```
##          Country/Region    Date  Confirmed  Deaths  Recovered
## 0          Afghanistan  1/22/20         0        0         0
## 1             Albania  1/22/20         0        0         0
## 2             Algeria  1/22/20         0        0         0
## 3             Andorra  1/22/20         0        0         0
## 4             Angola  1/22/20         0        0         0
## ...          ...          ...        ...        ...        ...
## 26395  Western Sahara  4/30/20         6         0         5
## 26396  Sao Tome and Principe  4/30/20        14         0         4
## 26397             Yemen  4/30/20         6         2         0
## 26398             Comoros  4/30/20         1         0         0
## 26399          Tajikistan  4/30/20        15         0         0
##
## [26400 rows x 5 columns]
```

```
corona.iloc[:,[1, 4, 5, 6, 7]]
```

```
##          Country/Region    Date  Confirmed  Deaths  Recovered
## 0          Afghanistan  1/22/20         0         0         0
## 1             Albania  1/22/20         0         0         0
## 2             Algeria  1/22/20         0         0         0
## 3             Andorra  1/22/20         0         0         0
## 4             Angola  1/22/20         0         0         0
## ...          ...          ...        ...        ...        ...
## 26395  Western Sahara  4/30/20         6         0         5
## 26396  Sao Tome and Principe  4/30/20        14         0         4
## 26397             Yemen  4/30/20         6         2         0
## 26398             Comoros  4/30/20         1         0         0
## 26399          Tajikistan  4/30/20        15         0         0
##
## [26400 rows x 5 columns]
```

Kérjük le a Province/State változó lehetséges értékeinek listáját az oszlopok `unique` módszerével!

```
corona['Province/State'].unique()
```

```
## array([nan, 'Australian Capital Territory', 'New South Wales',
##       'Northern Territory', 'Queensland', 'South Australia', 'Tasmania',
##       'Victoria', 'Western Australia', 'Alberta', 'British Columbia',
##       'Grand Princess', 'Manitoba', 'New Brunswick',
##       'Newfoundland and Labrador', 'Nova Scotia', 'Ontario',
##       'Prince Edward Island', 'Quebec', 'Saskatchewan', 'Anhui',
##       'Beijing', 'Chongqing', 'Fujian', 'Gansu', 'Guangdong', 'Guangxi',
##       'Guizhou', 'Hainan', 'Hebei', 'Heilongjiang', 'Henan', 'Hong Kong',
##       'Hubei', 'Hunan', 'Inner Mongolia', 'Jiangsu', 'Jiangxi', 'Jilin',
##       'Liaoning', 'Macau', 'Ningxia', 'Qinghai', 'Shaanxi', 'Shandong',
##       'Shanghai', 'Shanxi', 'Sichuan', 'Tianjin', 'Tibet', 'Xinjiang',
##       'Yunnan', 'Zhejiang', 'Faroe Islands', 'Greenland',
##       'French Guiana', 'French Polynesia', 'Guadeloupe', 'Mayotte',
##       'New Caledonia', 'Reunion', 'Saint Barthelemy', 'St Martin',
##       'Martinique', 'Aruba', 'Curacao', 'Sint Maarten', 'Bermuda',
##       'Cayman Islands', 'Channel Islands', 'Gibraltar', 'Isle of Man',
##       'Montserrat', 'Diamond Princess', 'Northwest Territories', 'Yukon',
##       'Anguilla', 'British Virgin Islands', 'Turks and Caicos Islands',
##       'Falkland Islands (Malvinas)', 'Saint Pierre and Miquelon'],
##      dtype=object)
```

### 2.7.2. Data frame-k módosítása

Nézzük meg újra a corona dataframe változóinak az adattípusát!

```
corona.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 26400 entries, 0 to 26399
## Data columns (total 8 columns):
##  #   Column          Non-Null Count  Dtype
## ---  ---
##  0   Province/State  8000 non-null   object
##  1   Country/Region  26400 non-null  object
##  2   Lat             26400 non-null  float64
##  3   Long            26400 non-null  float64
##  4   Date            26400 non-null  object
##  5   Confirmed       26400 non-null  int64
##  6   Deaths         26400 non-null  int64
##  7   Recovered       26400 non-null  int64
## dtypes: float64(2), int64(3), object(3)
## memory usage: 1.6+ MB
```

Mindenképp érdemes a Dátum oszlopot object (kvázi string) típusról ténylegesen dátum típusúvá alakítani! Így az időbeli kimutatásokat könnyebben lehet aggregálni év, negyedév, hónap, hét, nap szintekre.

Ezzel láthatjuk, hogy tudunk egy teljes oszlopot módosítani. A kulcs, hogy az oszlop korábbi önmagát felül kell írni a módosított (esetünkben a `to_datetime` csomag `to_datetime` függvénye jóvoltából egy dátummá konvertáláson átesett) verziójával.

```
corona.Date = pd.to_datetime(corona.Date)
```

```
## <string>:1: UserWarning: Could not infer format, so each element will be parsed individually,
```

```
corona.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 26400 entries, 0 to 26399
## Data columns (total 8 columns):
## #   Column          Non-Null Count  Dtype
## ---  ---
## 0   Province/State   8000 non-null   object
## 1   Country/Region   26400 non-null  object
## 2   Lat              26400 non-null  float64
## 3   Long             26400 non-null  float64
## 4   Date             26400 non-null  datetime64[ns]
## 5   Confirmed        26400 non-null  int64
## 6   Deaths           26400 non-null  int64
## 7   Recovered        26400 non-null  int64
## dtypes: datetime64[ns](1), float64(2), int64(3), object(2)
## memory usage: 1.6+ MB
```

Hozzunk létre egy teljesen új változót (leánykori nevén oszlopot :) a corona dataframe-ben, ami az adott dátumon továbbra is aktív koronavírusos esetek számát tartalmazza!

```
corona['Active'] = corona.Confirmed - corona.Deaths - corona.Recovered
```

```
corona.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 26400 entries, 0 to 26399
## Data columns (total 9 columns):
## #   Column          Non-Null Count  Dtype
## ---  ---
## 0   Province/State   8000 non-null   object
## 1   Country/Region   26400 non-null  object
## 2   Lat              26400 non-null  float64
## 3   Long             26400 non-null  float64
## 4   Date             26400 non-null  datetime64[ns]
## 5   Confirmed        26400 non-null  int64
## 6   Deaths           26400 non-null  int64
## 7   Recovered        26400 non-null  int64
## 8   Active           26400 non-null  int64
## dtypes: datetime64[ns](1), float64(2), int64(5), object(2)
## memory usage: 1.6+ MB
```

```
## 0 Province/State 8000 non-null object
## 1 Country/Region 26400 non-null object
## 2 Lat 26400 non-null float64
## 3 Long 26400 non-null float64
## 4 Date 26400 non-null datetime64[ns]
## 5 Confirmed 26400 non-null int64
## 6 Deaths 26400 non-null int64
## 7 Recovered 26400 non-null int64
## 8 Active 26400 non-null int64
## dtypes: datetime64[ns](1), float64(2), int64(4), object(2)
## memory usage: 1.8+ MB
```

```
corona.head()
```

```
## Province/State Country/Region Lat ... Deaths Recovered Active
## 0 NaN Afghanistan 33.0000 ... 0 0 0
## 1 NaN Albania 41.1533 ... 0 0 0
## 2 NaN Algeria 28.0339 ... 0 0 0
## 3 NaN Andorra 42.5063 ... 0 0 0
## 4 NaN Angola -11.2027 ... 0 0 0
##
## [5 rows x 9 columns]
```

### 2.7.3. Szűrés data frame-ben: logikai indexszálás

Ha valami logikai feltételt írunk egy data frame után [] jelek közé, akkor a logikai feltételnek megfelelő sorokat fogja nekünk kiválasztani a gép! Ez a **logikai indexszálás** c. művelet!

Pl. kérjük le azokat a rekordokat, ahol a halálozás 10000 feletti.

```
corona[corona.Deaths > 10000]
```

```
## Province/State Country/Region Lat ... Deaths Recovered Active
## 17561 NaN Italy 43.0000 ... 10023 12384 70065
## 17825 NaN Italy 43.0000 ... 10779 13030 73880
## 18089 NaN Italy 43.0000 ... 11591 14620 75528
## 18353 NaN Italy 43.0000 ... 12428 15729 77635
## 18617 NaN Italy 43.0000 ... 13155 16847 80572
## ... ... ... ...
## 26252 NaN France 46.2276 ... 24376 49476 91912
## 26273 NaN Italy 43.0000 ... 27967 75945 101551
## 26337 NaN Spain 40.0000 ... 24543 112050 76842
## 26359 NaN United Kingdom 55.3781 ... 26771 0 144482
```



```
## 26361          NaN          US  37.0902  ...   62996   153947  852481
##
## [135 rows x 9 columns]
```

Ha csak az ország és a dátum oszlopok kellenek az eredményből, akkor be lehet vetni a `loc` és `iloc`-ot.

```
corona.loc[corona.Deaths > 10000, ["Country/Region", "Date"]]
```

```
##      Country/Region      Date
## 17561          Italy 2020-03-28
## 17825          Italy 2020-03-29
## 18089          Italy 2020-03-30
## 18353          Italy 2020-03-31
## 18617          Italy 2020-04-01
## ...             ...      ...
## 26252          France 2020-04-30
## 26273          Italy 2020-04-30
## 26337          Spain 2020-04-30
## 26359  United Kingdom 2020-04-30
## 26361           US   2020-04-30
##
## [135 rows x 2 columns]
```

Ugyan azok a logikai műveletek és szimbólumok érvényesek itt is, mint az `if` elágazásoknál is.

Az eredmény menthető külön új data frame-be is:

```
szűrttészta = corona.loc[corona.Deaths > 10000, ["Country/Region", "Date"]]
szűrttészta.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 135 entries, 17561 to 26361
## Data columns (total 2 columns):
## #   Column      Non-Null Count  Dtype
## ---  ---
## 0   Country/Region  135 non-null   object
## 1   Date           135 non-null   datetime64[ns]
## dtypes: datetime64[ns](1), object(1)
## memory usage: 3.2+ KB
```

Vigyázzunk a sorindexek, még az eredeti data frame-ből jönnek. Pl. az első sor az az eredetiben a 17561-ik, így ezzel tudom kiválasztani, ha a `loc`-ot használom, mert ez a sorokat is a **nevükkel** azonosítja, mint az oszlopokat. A sor „neve”, pedig az eredeti data frame-ből örökölt index az ő kifacsart géplogikájában:

## 422. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
szűrttészta.loc[0,:]
```

```
## KeyError: 0
```

```
szűrttészta.loc[17561,:]
```

```
## Country/Region      Italy
## Date                2020-03-28 00:00:00
## Name: 17561, dtype: object
```

Viszont az `iloc` az mindent folytonosan sorszámmal azonosít, sort és oszlopot is, így az érteni fogja a 0-t.

```
szűrttészta.iloc[0,:]
```

```
## Country/Region      Italy
## Date                2020-03-28 00:00:00
## Name: 17561, dtype: object
```

Ha erre a `sima loc`-ot is rá akarjuk venni, akkor a `reset.index` metódust kell elsütöni. Figyeljük, hogy az eredménnyel felül kell írni az eredeti **szűrttészta** data frame-t!

```
szűrttészta = szűrttészta.reset_index()
szűrttészta.loc[0,:]
```

```
## index                17561
## Country/Region      Italy
## Date                2020-03-28 00:00:00
## Name: 0, dtype: object
```

Viszont figyeljük meg, hogy a régi sorindexek beköltöztek egy új `index` nevű oszlopba.

```
szűrttészta.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 135 entries, 0 to 134
## Data columns (total 3 columns):
## #   Column      Non-Null Count  Dtype
## ---  ---
## 0   index       135 non-null   int64
```

```
## 1 Country/Region 135 non-null object
## 2 Date           135 non-null datetime64[ns]
## dtypes: datetime64[ns](1), int64(1), object(1)
## memory usage: 3.3+ KB
```

Ha nem kellenek ezek az index adatok, törölhetjük is az oszlopot a `drop` metódus segítségével. A metódus első paraméterében megadjuk, hogy mely oszlopot akarjuk törölni a data frame-ből (ha listát adunk meg ide, akkor egyszerre több oszlopot is tudunk törölni), míg a második paraméterben megadjuk, hogy oszlopokat akarunk törölni, nem pedig sorokat.

```
szűrttészta = szűrttészta.drop("index", axis = "columns")
szűrttészta.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 135 entries, 0 to 134
## Data columns (total 2 columns):
## # Column          Non-Null Count  Dtype
## ---  ---
## 0 Country/Region  135 non-null   object
## 1 Date            135 non-null   datetime64[ns]
## dtypes: datetime64[ns](1), object(1)
## memory usage: 2.2+ KB
```

Az `axis = "index"` beállítással sorszám alapján sorokat lehet törölni a data frame-ből.

Nézzünk még pár szűrést logikai indexszálás segítségével végrehajtva!

Szűrjük le a corona dataframe-ből csak azokat a rekordokat, amik az USA, Olaszország és Irán adatait tartalmazzák! Egy `numpy` tömb elemeinek listába való tartozását a tömb (tehát data frame-ben az oszlop) `isin` metódusával tudunk vizsgálni.

```
corona[corona['Country/Region'].isin(['US', 'Italy', 'Iran'])]
```

```
##      Province/State Country/Region    Lat  ... Deaths Recovered  Active
## 133      NaN          Iran  32.0000  ...      0           0         0
## 137      NaN          Italy  43.0000  ...      0           0         0
## 225      NaN           US   37.0902  ...      0           0         1
## 397      NaN          Iran  32.0000  ...      0           0         0
## 401      NaN          Italy  43.0000  ...      0           0         0
## ...      ...          ...      ...  ...      ...           ...      ...
## 26009     NaN          Italy  43.0000  ...  27682       71252  104657
## 26097     NaN           US   37.0902  ...  60967      120720  858222
```

## 442. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
## 26269      NaN      Iran 32.0000 ... 6028 75103 13509
## 26273      NaN      Italy 43.0000 ... 27967 75945 101551
## 26361      NaN      US 37.0902 ... 62996 153947 852481
##
## [300 rows x 9 columns]
```

Ha az egész elé teszünk egy `~` jelet, akkor pedig tagadást végzünk, tehát megkapunk minden sort, ami NEM USA, Olaszország és Irán adata.

```
corona[~corona['Country/Region'].isin(['US', 'Italy', 'Iran'])]
```

```
##      Province/State      Country/Region ... Recovered Active
## 0      NaN      Afghanistan ...      0      0
## 1      NaN      Albania ...      0      0
## 2      NaN      Algeria ...      0      0
## 3      NaN      Andorra ...      0      0
## 4      NaN      Angola ...      0      0
## ...      ...      ...      ...      ...
## 26395      NaN      Western Sahara ...      5      1
## 26396      NaN      Sao Tome and Principe ...      4      10
## 26397      NaN      Yemen ...      0      4
## 26398      NaN      Comoros ...      0      1
## 26399      NaN      Tajikistan ...      0      15
##
## [26100 rows x 9 columns]
```

### 2.7.4. Hiányzó értékek kezelése data frame-ben

Az oszlopok `isnull` módszerével `True/False` módon megjelölhetők az oszlopon belüli hiányzó értékek.

```
corona['Province/State'].isnull()
```

```
## 0      True
## 1      True
## 2      True
## 3      True
## 4      True
##      ...
## 26395      True
## 26396      True
## 26397      True
## 26398      True
## 26399      True
## Name: Province/State, Length: 26400, dtype: bool
```

Aminek felhasználásával le is lehet őket kérdezni.

```
corona[corona['Province/State'].isnull()==True]
```

```
##      Province/State      Country/Region  ...  Recovered  Active
## 0              NaN      Afghanistan  ...         0         0
## 1              NaN        Albania  ...         0         0
## 2              NaN        Algeria  ...         0         0
## 3              NaN        Andorra  ...         0         0
## 4              NaN         Angola  ...         0         0
## ...            ...            ...  ...         ...         ...
## 26395           NaN  Western Sahara  ...          5          1
## 26396           NaN  Sao Tome and Principe  ...          4         10
## 26397           NaN            Yemen  ...          0          4
## 26398           NaN        Comoros  ...          0          1
## 26399           NaN      Tajikistan  ...          0         15
##
## [18400 rows x 9 columns]
```

Az oszlopoknak van egy `fillna` metódusa, amivel tetszőleges értékre le tudjuk cserélni a hiányzó értékeket. Most ez marha kreatív módon egy üres `str` lesz. :) Figyeljünk, hogy itt is felül kell írni az eredménnyel az eredeti oszlopot!

```
corona['Province/State'] = corona['Province/State'].fillna('')
corona.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 26400 entries, 0 to 26399
## Data columns (total 9 columns):
## #   Column      Non-Null Count  Dtype
## ---  ---
## 0   Province/State  26400 non-null  object
## 1   Country/Region  26400 non-null  object
## 2   Lat            26400 non-null  float64
## 3   Long           26400 non-null  float64
## 4   Date           26400 non-null  datetime64[ns]
## 5   Confirmed      26400 non-null  int64
## 6   Deaths         26400 non-null  int64
## 7   Recovered      26400 non-null  int64
## 8   Active         26400 non-null  int64
## dtypes: datetime64[ns](1), float64(2), int64(4), object(2)
## memory usage: 1.8+ MB
```

```
corona.head()
```

```
## Province/State Country/Region      Lat  ...  Deaths Recovered  Active
## 0                Afghanistan  33.0000  ...      0           0         0
## 1                Albania     41.1533  ...      0           0         0
## 2                Algeria     28.0339  ...      0           0         0
## 3                Andorra     42.5063  ...      0           0         0
## 4                Angola     -11.2027  ...      0           0         0
##
## [5 rows x 9 columns]
```

### 2.7.5. Adatvizualizáció data frame-en keresztül

Mentsük el a magyar adatokat egy új, **Hungary** nevű data frame-be! Ismét használjuk ki a dataframe logikai indexszálásának lehetőségét!

```
Hungary = corona[corona['Country/Region']=="Hungary"]
Hungary.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 100 entries, 129 to 26265
## Data columns (total 9 columns):
##  #   Column                Non-Null Count  Dtype
## ---  ---
## 0   Province/State        100 non-null   object
## 1   Country/Region        100 non-null   object
## 2   Lat                   100 non-null   float64
## 3   Long                  100 non-null   float64
## 4   Date                  100 non-null   datetime64[ns]
## 5   Confirmed             100 non-null   int64
## 6   Deaths               100 non-null   int64
## 7   Recovered             100 non-null   int64
## 8   Active                100 non-null   int64
## dtypes: datetime64[ns](1), float64(2), int64(4), object(2)
## memory usage: 7.8+ KB
```

Szűrjük le az újonnan létrehozott dataframe-ből a március előtti napokat! A jó hír, hogy `datetime` adattípusú oszlopokra ugyan úgy működnek a relációs jelek, mint számokra.

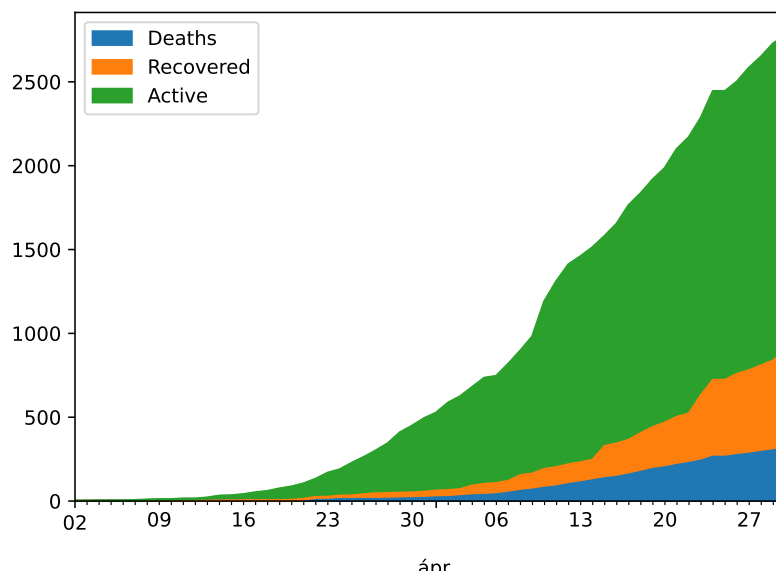
```
Hungary = Hungary[Hungary['Date'] > '2020-03-01']
Hungary.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 60 entries, 10689 to 26265
## Data columns (total 9 columns):
## #   Column          Non-Null Count  Dtype
## ---  ---
## 0   Province/State    60 non-null    object
## 1   Country/Region    60 non-null    object
## 2   Lat               60 non-null    float64
## 3   Long              60 non-null    float64
## 4   Date              60 non-null    datetime64[ns]
## 5   Confirmed         60 non-null    int64
## 6   Deaths            60 non-null    int64
## 7   Recovered         60 non-null    int64
## 8   Active            60 non-null    int64
## dtypes: datetime64[ns](1), float64(2), int64(4), object(2)
## memory usage: 4.7+ KB
```

Ábrázoljuk a COVID-19 magyar halottainak, gyógyultjainak és aktív eseteinek számát idő függvényében, halmozott területdiagramon a `matplotlib` segítségével!

Mivel a `matplotlib` együttműködik a `pandas`-al, így minden data frame-nek van külön metódusa a különböző diagramtípusokra. Pl. területdiagramra nem meglepő módon a `plot.area`. Ezek után csak a metódusban paraméterként meg kell adni, hogy mely oszlopok kerüljenek a diagram *x* és *y* tengelyeire. Illetve még egy extra paraméterben megadjuk, hogy *halmozott* diagramot szeretnénk készíteni: ez lesz a `stacked=True` paraméterbeállítás.

```
Hungary.plot.area(x="Date", y=["Deaths", "Recovered", "Active"], stacked=True)
```



## 2.8. Aggregálás data frame-ben

Szűrjük le a corona dataframe-ből a legfrissebb adatokat minden országra egy új, corona\_latest dataframe-be! Maximum függvényünk a `numpy` csomagból, tehát az `np` névtérből van.

```
corona_latest = corona[corona['Date']==np.max(corona['Date'])]
corona_latest.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 264 entries, 26136 to 26399
## Data columns (total 9 columns):
## #   Column      Non-Null Count  Dtype
## ---  ---
## 0   Province/State 264 non-null   object
## 1   Country/Region 264 non-null   object
## 2   Lat            264 non-null   float64
## 3   Long           264 non-null   float64
## 4   Date           264 non-null   datetime64[ns]
## 5   Confirmed      264 non-null   int64
## 6   Deaths        264 non-null   int64
## 7   Recovered      264 non-null   int64
## 8   Active         264 non-null   int64
```



```
## dtypes: datetime64[ns](1), float64(2), int64(4), object(2)
## memory usage: 20.6+ KB
```

Itt mivel egy ország akár lehet több régióval is jelen a sorok között országnév szerint össze tudjuk adni az összes megerősített koronavírusos esetet (*Confirmed* oszlop elemei). Magyarul **ország szintre szeretnénk összeg segítségével aggregálni** a megerősített koronavírus esetek számát.

Ehhez először a data frame `groupby` metódusával csoportosítani kell a sorokat ország szintre, majd megadni az összegzendő oszlopot és elsütni ezen oszlop `sum` metódusát az összegzéshez. Ha a `sum`-ot `avg`-re vagy `median`-ra cseréljük, akkor a kiválasztott oszlopnak nem az összegét, hanem az átlagát, illetve mediánját tudjuk nézni országok szerint. Azaz **átlaggal/mediánnal is aggregálhatunk az országok szintjére**.

```
corona_country = corona_latest.groupby('Country/Region')['Confirmed'].sum()

corona_country.info()
```

```
## <class 'pandas.core.series.Series'>
## Index: 187 entries, Afghanistan to Zimbabwe
## Series name: Confirmed
## Non-Null Count  Dtype
## -----
## 187 non-null    int64
## dtypes: int64(1)
## memory usage: 2.9+ KB
```

```
corona_country.head()
```

```
## Country/Region
## Afghanistan    2171
## Albania         773
## Algeria         4006
## Andorra         745
## Angola          27
## Name: Confirmed, dtype: int64
```

Az eredmény nem egy data frame, hanem mint láthatjuk egy `Series` lett, ami ugyebár logikailag egy `numpy` tömb! Mivel a `groupby` során az országok nevéből lett az új sorindex, így csak 1 oszlopunk maradt, az összegzett megerősített esetszám.

Ha azt szeretnénk, hogy a `corona_country`-ben az országok neve ne indexként, hanem külön oszlopként szerepeljen, akkor használjuk a `reset_index()` metódust!

```
corona_country = corona_country.reset_index()
```

```
corona_country.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 187 entries, 0 to 186
## Data columns (total 2 columns):
## #   Column          Non-Null Count  Dtype
## ---  ---
## 0   Country/Region  187 non-null   object
## 1   Confirmed       187 non-null   int64
## dtypes: int64(1), object(1)
## memory usage: 3.1+ KB
```

```
corona_country.head()
```

```
##   Country/Region  Confirmed
## 0   Afghanistan      2171
## 1     Albania         773
## 2     Algeria        4006
## 3     Andorra         745
## 4      Angola         27
```

Amennyiben a `groupby` metódus után egy általánosabb `agg` metódust használunk, akkor a `groupby`-ban megadott csoportosítás szerint egyszerre több művelet segítségével is összesíthetjük, azaz aggregálhatjuk a számértékű oszlop értékeit (pl. egyszerre nézünk átlagos és medián esetszámokat), Vagy akár több számértékű oszlopot is aggregálhatunk a `groupby` paraméterei szerint (pl. egyszerre nézünk átlagos esetszámot és halálozást is). Ráadásul, el is tudjuk nevezni az `agg` függvényen belül az újonnan létrehozott összesítő, azaz aggregált oszlopokat! Annyi trükk van a dologban, hogy az aggregáláshoz használt függvényeket (medián, átlag, szórás, stb.) a `numpy` csomagból szedjük ki az `np.` előtaggal!

Szóval a következő kódrészletben az `AtlagAktiv = ("Active", np.mean)` rész azt jelenti majd pl, hogy az `AtlagAktiv` oszlop a létrehozandó kimutatástáblában az eredeti data frame `Active` oszlopának átlaggal, azaz `np.mean` függvényével országos szintre összesített („*groupby*”-olt) értékeit tartalmazza.

Na, akkor mostmár tényleg készítsünk el egy ország szintű kimutatást az átlagos és medián aktív koronavírus eseteiről, illetve átlagos és medián halálozási számairól a legfrisebb dátumra! Azt is megtehetjük, hogy már az aggregáló kód végére rögtön odatoljuk a `reset_index`-et.

```
corona_kimutatas = corona_latest.groupby('Country/Region').agg(
    AtlagAktiv = ("Active", np.mean),
    MedianAktiv = ("Active", np.median),
    AtlagHalal = ("Deaths", np.mean),
    MedianHalal = ("Deaths", np.median)
).reset_index()
```

```
## <string>:1: FutureWarning: The provided callable <function mean at 0x00000167D93F13A0> is current
## <string>:1: FutureWarning: The provided callable <function median at 0x00000167D95B8720> is current
## <string>:1: FutureWarning: The provided callable <function mean at 0x00000167D93F13A0> is current
```

```
corona_kimutatas
```

```
##      Country/Region  AtlagAktiv  MedianAktiv  AtlagHalal  MedianHalal
## 0      Afghanistan    1847.0      1847.0      64.0      64.0
## 1      Albania        272.0      272.0      31.0      31.0
## 2      Algeria       1777.0     1777.0     450.0     450.0
## 3      Andorra        235.0      235.0      42.0      42.0
## 4      Angola         18.0       18.0       2.0       2.0
## ..      ...          ...         ...         ...         ...
## 182  West Bank and Gaza  266.0      266.0       2.0       2.0
## 183  Western Sahara     1.0       1.0       0.0       0.0
## 184      Yemen          4.0       4.0       2.0       2.0
## 185      Zambia        48.0      48.0       3.0       3.0
## 186      Zimbabwe      31.0      31.0       4.0       4.0
##
## [187 rows x 5 columns]
```

Azt látjuk, hogy az átlag és medián értékek mind az aktív esetek számára, mind a halálozási számokra megegyeznek. Ez azért van, mert a legtöbb ország ugyebár nem volt lebontva államokra és provinciákra, szóval egy nap csak egy érték érkezett a táblába rájuk mindenből. Egy értéknek pedig nyilván ugyan az az átlaga és a mediánja is! :)

Na, de lessünk meg pár olyan országot, ahol az adatok belső régiókra, provinciákra is le voltak bontva. Pl. Franciaország és az Egyesült Királyság ilyen országok voltak:

```
corona_kimutatas[corona_kimutatas["Country/Region"].isin(["France", "United Kingdom"])]
```

```
##      Country/Region  AtlagAktiv  MedianAktiv  AtlagHalal  MedianHalal
## 62      France    8409.909091      31.0    2219.090909      1.0
## 177  United Kingdom 13161.818182      13.0    2440.181818      1.0
```

Na itt már látszik az eltérés! Pl. a francia tartományok felében legfeljebb 31 aktív koronavírusos eset volt csak 2020.04.30-án, de a tartományok átlagában az érték már kerekítve 8410 eset! Ezt valószínűleg egy vagy kettő kiugróan sok esettel rendelkező tartomány okozza csak!

## 2.9. Egyszerű leíró statisztika data frame-ben

No, de most térjünk vissza a `corona_country` data frame-hez! Egyszerűsítsük az oszlopneveket! A data frame-k `columns` tulajdonságának felülírásával az oszlopnevek könnyen módosíthatók. Mivel ugye a `columns` tulajdonságban az összes oszlopnév szerepel listaként, így az új oszlopneveket listaként felsorolva [] jellel kell megadni.

```
corona_country.columns=['Country', 'COVID_Cases']

corona_country.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 187 entries, 0 to 186
## Data columns (total 2 columns):
## #   Column      Non-Null Count  Dtype
## ---  ---
## 0   Country     187 non-null   object
## 1   COVID_Cases 187 non-null   int64
## dtypes: int64(1), object(1)
## memory usage: 3.1+ KB
```

Nézzünk egy komplett leíró statisztikát a `COVID_Cases` változóra/ismérvre/oszlopra a `describe` metódus segítségével. Kerekítsük az eredményeket 2 tizedesjegyre (`round` függvény).

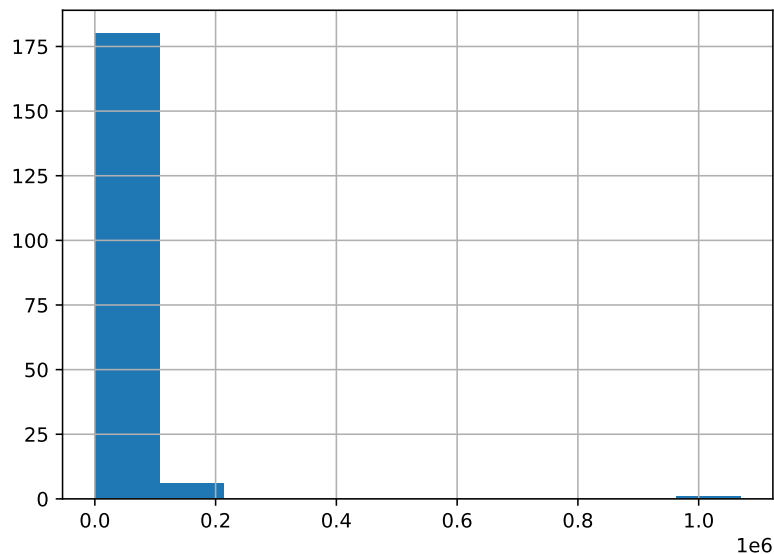
```
round(corona_country.COVID_Cases.describe(),2)
```

```
## count      187.00
## mean      17416.26
## std       84414.11
## min         1.00
## 25%        97.50
## 50%       746.00
## 75%      6254.50
## max     1069424.00
## Name: COVID_Cases, dtype: float64
```

Nézzük meg ezt az ordenáre módon jobbra elnyúló eloszlást hisztogramon és doboz ábrán is!

A hisztogramot simán a vizsgált oszlop `hist` módszerével le lehet kérni.

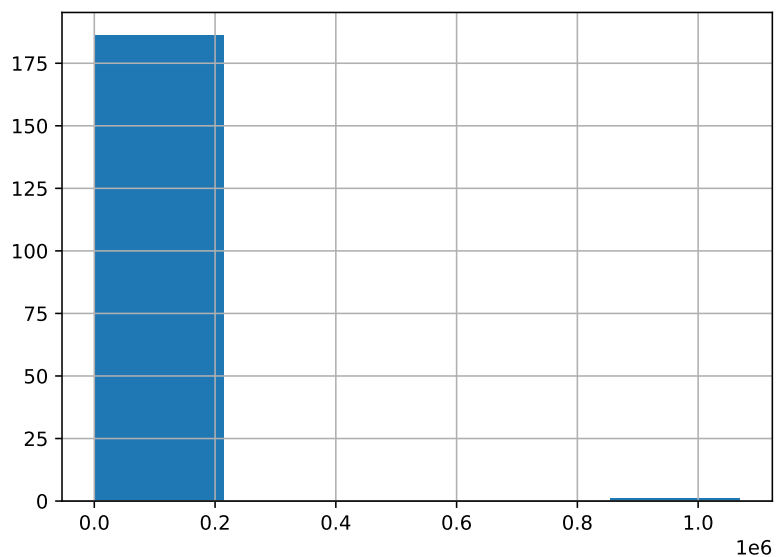
```
corona_country.COVID_Cases.hist()
```



Alapból egyenlő hosszúságú osztályközöket képez a python által a hisztogramokhoz, aminek a számát a `hist` függvényben a `bins` paraméteren keresztül tudjuk szabályozni.

Vigyünk le pl. az osztályközök számát 5-re.

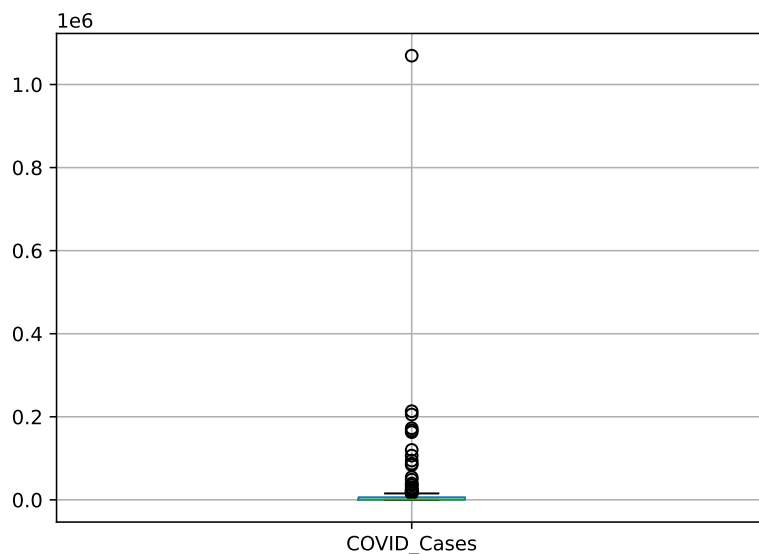
```
corona_country.COVID_Cases.hist(bins=5)
```



A `boxplot` metódus már alapvetően data frame, és nem oszlop szinten működik, és a metódus paraméterében kell megadni, hogy mely oszlopra vagy oszlopokra (neveket listaként felsorolva `[]` jellel) akarjuk az ábrát. Tehát, a doboz ábrát egyszerre több oszlopra is lekérhetjük egy ábrán belülre akár. Majd mindjárt nézünk ilyet is. Ennek a doboz ábránál van értelme, hiszen doboz ábránál nincsenek osztályközök, amiknek a számát esetlegesen az ismervünk (oszlopunk) eloszlására kell szabni.

```
corona_country.boxplot(column="COVID_Cases")
```

## 2.10. ADATMINŐSÉGI PROBLÉMÁK FELISMERÉSE ÉS KEZELÉSE LEÍRÓ STATISZTIKA SEGÍTSÉGÉVEL



## 2.10. Adatminőségi problémák felismerése és kezelése leíró statisztika segítségével

Olvassuk be a `population_by_country_2020.csv` nevű fájlt, és mentjük el a beolvasott adatokat egy **population** nevű data frame-be!

```
population = pd.read_csv('population_by_country_2020.csv')
population.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 235 entries, 0 to 234
## Data columns (total 11 columns):
##  #   Column                                Non-Null Count  Dtype
## ---  ---
##  0   Country (or dependency)              235 non-null   object
##  1   Population (2020)                    235 non-null   int64
##  2   Yearly Change                        235 non-null   object
##  3   Net Change                           235 non-null   int64
##  4   Density (P/Km²)                     235 non-null   int64
##  5   Land Area (Km²)                     235 non-null   int64
##  6   Migrants (net)                       201 non-null   float64
##  7   Fert. Rate                           235 non-null   object
```

```
## 8    Med. Age                235 non-null    object
## 9    Urban Pop %             235 non-null    object
## 10   World Share             235 non-null    object
## dtypes: float64(1), int64(4), object(6)
## memory usage: 20.3+ KB
```

A **population** dataframe-ből csak az országnév, népesség, népsűrűség és városi népesség aránya változókra lesz szükségünk. A többit törölhetjük is a data frame-ből! Mivel az oszlopnevekben mint fentebb láthatjuk elég sok a hányadékos módosító speciális karakter, így biztonságosabb most az oszlopokra a sorszámukkal hivatkozni. Láthatjuk az **info** metódus eredményéből, hogy a szükséges országnév, népesség, népsűrűség és városi népesség aránya oszlopok rendre a 0, 1, 4, 9 indexekkel bírnak.

```
population = population.iloc[:, [0, 1, 4, 9]]

population.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 235 entries, 0 to 234
## Data columns (total 4 columns):
## #   Column                Non-Null Count  Dtype
## ---  ---
## 0    Country (or dependency) 235 non-null    object
## 1    Population (2020)       235 non-null    int64
## 2    Density (P/Km²)         235 non-null    int64
## 3    Urban Pop %             235 non-null    object
## dtypes: int64(2), object(2)
## memory usage: 7.5+ KB
```

Egyszerűsítsük az oszlopneveket a population dataframe-ben!

```
population.columns = ['Country', 'Pop', 'PopDensity', 'UrbanPop']

population.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 235 entries, 0 to 234
## Data columns (total 4 columns):
## #   Column      Non-Null Count  Dtype
## ---  ---
## 0    Country     235 non-null    object
## 1    Pop         235 non-null    int64
## 2    PopDensity  235 non-null    int64
```



## 2.10. ADATMINŐSÉGI PROBLÉMÁK FELISMERÉSE ÉS KEZELÉSE LEÍRÓ STATISZTIKA SEGÍTSÉGÉVEL

```
## 3    UrbanPop    235 non-null    object
## dtypes: int64(2), object(2)
## memory usage: 7.5+ KB
```

Nézzük meg a population dataframe egyszerű leíró statisztikai mutatóit! Ha a `describe` metódust az egész data frame-n engedjük el, akkor minden numerikus (int vagy float) oszlopra megadja az alap leíró mutatókat.

```
round(population.describe(), 2)
```

```
##              Pop  PopDensity
## count  2.350000e+02      235.00
## mean   3.316936e+07      475.77
## std    1.351374e+08     2331.29
## min    8.010000e+02        0.00
## 25%    3.988760e+05        37.00
## 50%    5.459642e+06       95.00
## 75%    2.057705e+07     239.50
## max    1.439324e+09    26337.00
```

Az *UrbanPop* változónak mi baja? Elviekben az egy arányszám, annak is számnak kéne lennie, és meg kéne jelennie a `describe` metódus eredményében!

Kukkantsunk csak bele a data frame első 5 sorába!

```
population.head()
```

```
##      Country      Pop  PopDensity  UrbanPop
## 0      China  1439323776        153      61%
## 1      India  1380004385        464      35%
## 2  United States  331002651         36      83%
## 3   Indonesia  273523615        151      56%
## 4   Pakistan  220892340        287      35%
```

Áhhá! Százalékjel van benne! Ezért veszi szöveges adatnak a pythonállat!

Szedjük le ezt a százalékjelet! Erre szerencsére az egyes data frame oszlopoknak van egy `str.replace` metódusa, amiben megadhatjuk paraméterekkel, hogy az oszlopban milyen szövegrészeket mire akarunk cserélni. Itt most ugyebár százalékjelet fogunk üres stringre cserélni.

```
population['UrbanPop'] = population['UrbanPop'].str.replace('%', '')
population.head()
```

## 582. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
##          Country      Pop  PopDensity  UrbanPop
## 0          China  1439323776         153         61
## 1          India  1380004385         464         35
## 2  United States   331002651          36         83
## 3    Indonesia   273523615         151         56
## 4    Pakistan   220892340         287         35
```

Ez jó, de sajnos vannak benne hiányzó értékek, amik nem a szabványos Python NaN kóddal vannak jelölve, hanem ilyen spéci „N.A.” stringgel, amit a gépállat nem ismer fel, így az egész oszlopot **str**-nek (**object**) veszi a **numpy** tömbök egységes adattípus logikája alapján.

```
population.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 235 entries, 0 to 234
## Data columns (total 4 columns):
## #   Column      Non-Null Count  Dtype
## ---  ---
## 0   Country     235 non-null   object
## 1   Pop         235 non-null   int64
## 2   PopDensity  235 non-null   int64
## 3   UrbanPop    235 non-null   object
## dtypes: int64(2), object(2)
## memory usage: 7.5+ KB
```

Azt, hogy az **object** adattípus turpisságát az „N.A.”-k okozzák az **UrbanPop** oszlopban, arra leginkább az oszlop gyakorisági táblájából lehet felismerni. Ezt a gyakorisági táblát az oszlop **value\_counts** módszerével tudjuk lekérni.

```
population.UrbanPop.value_counts()
```

```
## UrbanPop
## N.A.      13
## 57        7
## 88        7
## 63        6
## 87        6
##          ..
## 50        1
## 81        1
## 28        1
## 37        1
## 10        1
## Name: count, Length: 81, dtype: int64
```

Láthatjuk, hogy a sok számérték mellett, 13 ország esetén hiányzó értékünk van ezzel a csúnya „N.A.” kóddal.

Na, akkor! Most csináljuk azt, hogy leszűrjük azt a 13 országot, ahol hiányzik a városi népesség arányára vonatkozó adat!

Ezek után próbáljuk meg a városi népesség arányára vonatkozó adatot `int` típusúvá konvertálni! Ha sikerült, nézzük meg a változó alap leíró statisztikai mutatóit is!

Először logikai indexszeléssel leszűrjük az „N.A.”-kat.

```
population = population[population['UrbanPop'] != "N.A."]
```

Majd az oszlop `astype` módszerével `int`-é konvertáljuk az egész oszlopot. A módszer paraméterében kell megadni, hogy milyen adattípusra akarjuk konvertálni kiszemelt kis oszlopunk! :) Végül jöhet a `describe`.

```
population['UrbanPop'] = population['UrbanPop'].astype(int)

population.describe()
```

##		Pop	PopDensity	UrbanPop
## count	2.220000e+02	222.000000	222.000000	222.000000
## mean	3.488611e+07	186.373874	59.234234	
## std	1.388508e+08	288.271695	24.230400	
## min	1.357000e+03	0.000000	0.000000	
## 25%	5.444048e+05	35.000000	43.000000	
## 50%	5.911701e+06	89.000000	60.500000	
## 75%	2.321589e+07	224.500000	79.000000	
## max	1.439324e+09	2239.000000	100.000000	

Úgy tűnik, helyreállt a világ rendje! Már nagyon szépen le tudjuk olvasni pl., hogy a Föld országainak legzsúfoltabb 25%-ban legalább 79 fő/Km<sup>2</sup> a népsűrűség. És azt is látni a `count`-ból, hogy már csak 222 országunk van a kezdeti 235 helyett, szóval nincsen itt a 13 „N.A.”.

## 2.11. Data frame-k összekapcsolása

Akkor most álmodjunk egy nagyot! Kössük össze a `population` data frame-ben található országokénti alapvető demográfiai ismérveket a `corona_country` data frame-ben lakó országokénti koronavírus esetszámokkal.

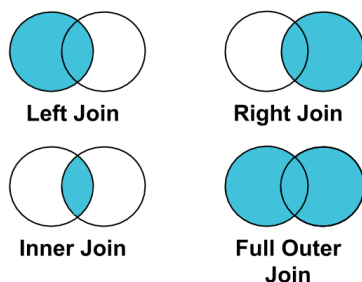
Nyilván ezt az összekötést az országok nevén keresztül lehet megtenni. Azaz pl. a magyar koronavírus esetszámokhoz a magyar demográfiai adatoknak kell kerülnie értelemszerűen. :)

A **pandas** csomagnak létezik egy **merge** névre hallgató függvénye, ami két data frame-et összeköt egy előre megadott közös oszlop alapján. Esetünkben ez a közös oszlop az országnév lesz. Ha egy kicsit „*adatbázisabbul*” szeretném kifejezni magam, akkor azt mondanám, hogy a **merge** függvény 2 tábla joinját oldja meg egy közös kulcs alapján.

Sőt, a **merge** függvény mindhárom alapvető táblakapcsolási módszert támogatja:

- **inner join:** Az összekötött táblában csak azok a sorok maradnak meg, amelyek mindkét data frame-ben szerepelnek.
- **left join:** Az összekötött táblában csak azok a sorok maradnak meg, amelyek az elsőre megnevezett data frame-ben szerepelnek (attól függetlenül, hogy a másodszorra megnevezett táblában van-e hozzájuk találat).
- **right join:** Az összekötött táblában csak azok a sorok maradnak meg, amelyek a másodszorra megnevezett data frame-ben szerepelnek (attól függetlenül, hogy az elsőre megnevezett táblában van-e hozzájuk találat).
- **full outer join:** Az összekötött táblában mindkét tábla minden sora megmarad.

A különböző típusú összekötési módokat remekül lehet halmazábrákkal szemléltetni:



Na, akkor mielőtt a tényleges **merge**-hez hozzálátunk, annyit ellenőrizzünk le, hogy ugyanaz-e a neve az országneveket tartalmazó oszlopnak mindkét data frame-ben, a **population**-ben és a **corona\_country**-ban is:

```
corona_country.columns
```

```
## Index(['Country', 'COVID_Cases'], dtype='object')
```

```
population.columns
```

```
## Index(['Country', 'Pop', 'PopDensity', 'UrbanPop'], dtype='object')
```

Szuper, mindkét táblában egységesen **Country** az összekötésre használandó oszlop neve! Nem meglepő, mert mindkét táblában átneveztük már korábban az oszlopokat, de azért jobb biztosra menni. :)

Akkor lássuk azt a **merge**-t! Most egy olyan összekötést csinálunk, hogy a **corona\_country** táblában lévő összes sorunk maradjon meg az összekötött táblában, mert alapvetően azok az országok érdekelnek, ahol megvan a COVID fertőzöttek száma. Ez az én data frame megadási sorrendben majd egy *left joint* fog jelenteni. :)

A **merge** függvényben a két data frame megadása után a **how** paraméter szabályozza a *join* jellegét, míg az **on** paraméterben adjuk meg az összekötésre használt oszlop nevét. A *join* tehát azért lesz *left*, mert a **corona\_country**-t adtam meg először, azaz „*balrabb*”. :)

```
corona_extended = pd.merge(corona_country, population, how='left', on='Country')
corona_extended.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 187 entries, 0 to 186
## Data columns (total 5 columns):
## #   Column      Non-Null Count  Dtype
## ---  -
## 0   Country     187 non-null   object
## 1   COVID_Cases 187 non-null   int64
## 2   Pop         168 non-null   float64
## 3   PopDensity  168 non-null   float64
## 4   UrbanPop    168 non-null   float64
## dtypes: float64(3), int64(1), object(1)
## memory usage: 7.4+ KB
```

Na, hát az új data frame **info** metódusa alapján van egy kis probléma. 187 – 168 = 19 országra a **corona\_country** data frame-ben nem volt találat a **population** data frame-ben.

### 2.11.1. A kapcsolási kulcsnak használt oszlop ellenőrzése és javítása

Lessük meg mik ezek az országok, ahol nem volt találat a **population** data frame-ben! Ezt pl. úgy tudjuk megtenni, hogy lekérdezzük a hiányzó értékek országát a **Pop** oszlopban.

```
corona_extended.Country[corona_extended.Pop.isnull()==True]
```

## 622. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
## 27                Burma
## 39            Congo (Brazzaville)
## 40            Congo (Kinshasa)
## 42            Cote d'Ivoire
## 46                Czechia
## 48            Diamond Princess
## 75                Holy See
## 91                Kosovo
## 92                Kuwait
## 102              MS Zaandam
## 113              Monaco
## 140      Saint Kitts and Nevis
## 142  Saint Vincent and the Grenadines
## 144      Sao Tome and Principe
## 150              Singapore
## 164              Taiwan*
## 173                US
## 180              Venezuela
## 182      West Bank and Gaza
## Name: Country, dtype: object
```

Elnézegetve az országneveket kialakulhat bennünk valami sejtés: valószínűleg ezeket az országokat máshogy hívják a **population** data frame-ben, mint a **corona\_country**-ban. Pl. *Taiwan* nevében valószínűleg nem lesz csillag, vagy *Czechia*-t inkább a hivatalosabb nevén jegyezheti a **population** tábla: *Czech Republic*. Esetleg a *US* is inkább *United States*-ként szerepelhet.

Teszteljük le ezeket az elméleteket egy egyszerű logikai indexes szűréssel az **isin** metódussal megtámogatva.

```
population.Country[population.Country.isin(["Czech Republic", "Taiwan", "United States"])]
```

```
## 2      United States
## 56           Taiwan
## Name: Country, dtype: object
```

Mintha bejönne az okoskodásunk, de a cseheket csak nem akarja megtalálni a cucc. Próbáljunk meg úgy szűrni, hogy ne pontosan keressük ezeket az országneveket, hanem azt nézzük meg, hogy mik azok a sorok a **population** data frame-ben, amik ezeket az országneveket *tartalmazzák* valahol a **Country** oszlopban. Ezt egyszerűen el tudjuk érni úgy, hogy az előző kódunkban az **isin** metódust **str.contains**-re cseréljük. Annyi van, hogy itt a keresett string mintázatokat egy stringben kell megadni (azaz NEM listaként) „|” jellel elválasztva őket. Ez így amúgy egy úgynevezett RegEx kifejezés, és ilyenekkel lehet komplexebben működtetni ezt a **str.contains** metódust. Az érdeklődőknek jó kiindulópont a link. :)

```
population.Country[population.Country.str.contains("Czech Republic|Taiwan|United States")]
```

```
## 2                United States
## 56                Taiwan
## 85    Czech Republic (Czechia)
## Name: Country, dtype: object
```

Ó, hogy a kedves felmenőiket a **population** data frame alkotóinak: hát nem ott van zárójelben a *Czech Republic* mögött, hogy *Czechia*?! „Ripsz!”

Hát valami hasonló módon be kéne lőni a maradék 16 nem egyező országnévet is, de most ezzel nem húzzuk a drága időnket, hanem javítjuk ezt a 3 esetet és újra összekötjük a tábláinkat. Aki a nem egyértelmű kapcsoló oszlop (kulcs) alapján történő data frame összekötés világában szeretne elmélyedni, neki érdemes lehet majd előbb-utóbb utána néznie a *fuzzy join* technikáknak, amiket a Pythonban pl. a *difflib* csomag támogat. De ezek a megoldások az itteni bevezető példának a kereteit bőven megugorják komplexitásában.

Szóval, akkor a 3 azonosított eltérő országnévet javítsuk a **population** data frame-ben. Azaz ott átírjuk ezeket az országnéveket arra a verzióra, ami a **corona\_country**-ban is szerepel. Ehhez megint az **str.replace**-t használjuk, mint anno az **UrbanPop** oszlop százalékjeleinek eltávolításakor. Ezt mindhárom esetben külön meg kell sajnos tenni.

```
population['Country'] = population['Country'].replace('United States', 'US')
population['Country'] = population['Country'].replace('Czech Republic (Czechia)', 'Czechia')
population['Country'] = population['Country'].replace('Taiwan', 'Taiwan*')
```

És akkor lássuk újra azt a **merge**-t! Most a többi nem kezelt esetet eldobjuk, szóval *inner joint* csinálunk.

```
corona_extended = pd.merge(corona_country, population, how='inner', on='Country')
corona_extended.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 171 entries, 0 to 170
## Data columns (total 5 columns):
##  #   Column      Non-Null Count  Dtype
## ---  ---
##  0   Country     171 non-null   object
##  1   COVID_Cases 171 non-null   int64
##  2   Pop         171 non-null   int64
##  3   PopDensity  171 non-null   int64
##  4   UrbanPop    171 non-null   int32
## dtypes: int32(1), int64(3), object(1)
## memory usage: 6.1+ KB
```

Szupszi! Már nem 168 sor van, amire van találat mindkét data frame-ben, mint az előbb, hanem 171, azaz pont a megjavított 3 országgal több! Na, erre már elő lehet venni az ünnepi laposüveget (leánykori nevén lapiüvit)! ;)

## 2.12. Kilógó értékek keresése és kezelése

A sikeres data frame összekötési művelet örömére, számoljuk ki a **corona\_extended** dataframe-ben az egymillió főre jutó COVID-19 esetek számát minden országra! Aztán nézzük is meg az oszlop leíró statisztikáit!

```
corona_extended['COVID_perMillion'] = corona_extended.COVID_Cases / corona_extended.Population
corona_extended['COVID_perMillion'].describe()
```

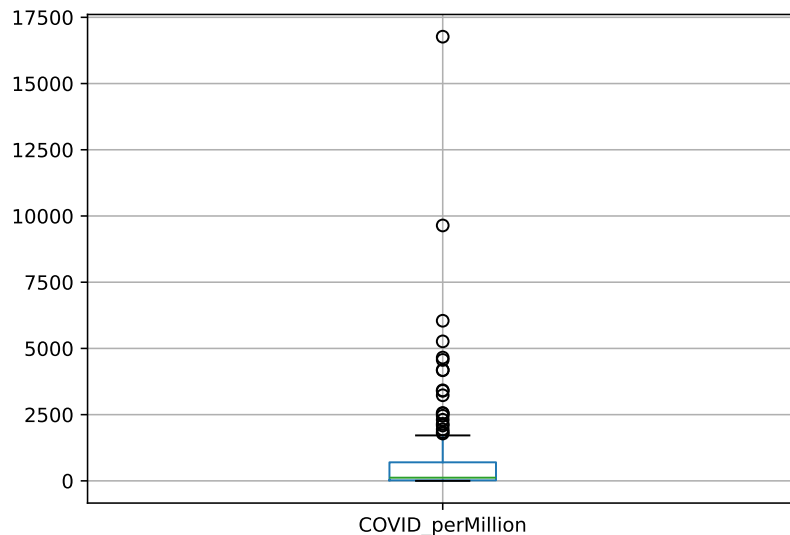
```
## count      171.000000
## mean       747.439425
## std        1775.534029
## min         0.201167
## 25%        19.420289
## 50%        119.830206
## 75%        700.197689
## max        16769.325985
## Name: COVID_perMillion, dtype: float64
```

Na, szuper, itt is látszik egy csodás jobbra elnyúló eloszlás, hiszen az országok  $\frac{3}{4}$ -ének az egymillió főre vetített COVID esetszáma nem haladja meg a 700-at, de ellenben a legnagyobb érték már majdnem 17 ezer fő! Ellenben az alsó 25% határa, a 19.4 egészen közel van a minimumhoz, a 0.2-höz. Szóval valószínűleg brutál felfelé kilógó elemeink vannak.

Ezt erősítsük is meg egy doboz ábrán.

```
corona_extended.boxplot(column="COVID_perMillion")
```





Az ábra alapján nagyjából olyan 3000 feletti értékek tűnnek extrém módon kilógónak (kb. 3000-nél van az első szakadás a dobozban a ponttal jelölt kilógó értékek körében; a szakadás alatti részek, még kb a normál adatok „természetes” folytatásának tekinthetők). Lássuk hát, hogy mik ezek!

Rendezzük a **COVID\_perMillion** szerint csökkenő sorrendbe a data frame-t, és kérjük le a sorbarendeztet verzióból azokat az értékeket, ahol a **COVID\_perMillion** nagyobb, mint 3000!

A data frame-t sorba rendezni a **sort\_values** metódussal lehet, amelynek paraméterében meg kell adni, hogy mely oszlop alapján rendezünk, és hogy a sorrend csökkenő vagy növekvő-e. Majd ezen a rendezett állapoton elsűthetünk pl. egy **iloc**-ot az első 10 sor kiválasztásához.

```
corona_extended.sort_values('COVID_perMillion', ascending=False).loc[corona_extended['COVID_perMi
```

##	Country	COVID_Cases	...	UrbanPop	COVID_perMillion
## 131	San Marino	569	...	97	16769.325985
## 3	Andorra	745	...	88	9642.140685
## 93	Luxembourg	3784	...	88	6044.940877
## 72	Iceland	1797	...	94	5266.042087
## 126	Qatar	13409	...	96	4654.201086
## 143	Spain	213435	...	80	4564.987989
## 16	Belgium	48519	...	98	4186.417453
## 77	Ireland	20612	...	63	4174.340484
## 148	Switzerland	29586	...	74	3418.520185

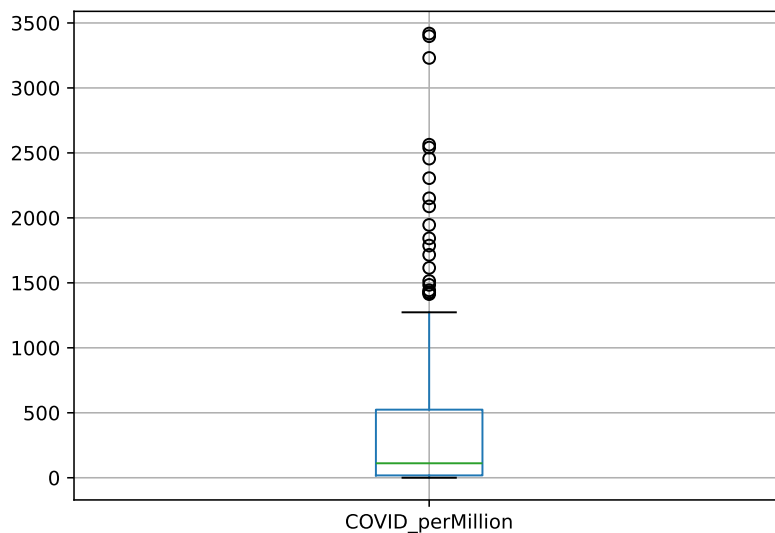
## 662. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
## 79          Italy      205463 ...      69      3398.226842
## 159          US        1069424 ...      83      3230.862341
##
## [11 rows x 6 columns]
```

Na, úgy néz ki, hogy az érintett országok többségében ilyen jó kicsi, zsúfolt államok. Persze vannak extrém kivételek, pl. ugye az olaszok a nagy repülő turistaforgalmuk miatt.

Na, ezeket az extrém módon kilógó értékeket kipucoljuk a data frame-ből, aztán ránézünk újra a **COVID\_perMillion** doboz ábrájára. Most a kilógó értékeket vegyük csak a 4000 feletti esetszámnak, mivel a sorrend alapján van egy nagyobb ugrás ott a svájci 3418-ról az ír 4174-re. Meg a doboz ábrán is látszik, hogy ez az utolsó 3 érték ebben a „toplistában” még azért közelebb van az adatok „természetes folytatásához”, és utána jön még egy ugrás az egymillió fölé vetített esetszámokban.

```
corona_extended = corona_extended[corona_extended['COVID_perMillion'] < 4000]
corona_extended.boxplot(column="COVID_perMillion")
```



Ez már egy kulturáltabb jobbra elnyúló eloszlás. Viszont, a medián még mindig túlságosan közel van az alsó kvartilishez, és a felső kvartilis eléggé elszakad.

Ennek szellemében még nézzünk rá arra, hogy mely országok esnek az egymillió fölé jutó COVID esetszám szerint az alsó kvintilisbe!

Egy data frame oszlop alsó kvintilisét az oszlop `quantile` metódusával számoljuk ki. A metódus alapvetően egy tetszőleges percentilist számol ki. Azt, hogy melyiket, azt a metódus paraméterében kell megadni 0 – 1 közötti számként. Szóval az alsó kvintilis alias 20. percentilis, ami alatt az adatok 20%-a található, egy 0.2 paraméterrel lesz megadható.

```
corona_extended['COVID_perMillion'].quantile(0.2)
```

```
## 10.058723607815136
```

Ezt a fenti kódot felhasználva egy logikai indexes szűrésben gyorsan meg is lesznek a népességarányos esetszám szerinti alsó kvintilisbe tartozó országok nevei is.

```
corona_extended[corona_extended['COVID_perMillion'] <
corona_extended['COVID_perMillion'].quantile(0.2)]
```

##	Country	COVID_Cases	...	UrbanPop	COVID_perMillion
## 4	Angola	27	...	67	0.821511
## 18	Benin	64	...	48	5.279134
## 19	Bhutan	7	...	46	9.071964
## 22	Botswana	23	...	73	9.780463
## 27	Burundi	11	...	14	0.925086
## 29	Cambodia	122	...	24	7.297102
## 33	Chad	73	...	23	4.444211
## 37	Comoros	1	...	29	1.149953
## 54	Ethiopia	131	...	21	1.139491
## 59	Gambia	11	...	59	4.551722
## 69	Haiti	81	...	57	7.103688
## 84	Kenya	396	...	28	7.364524
## 86	Laos	19	...	36	2.611483
## 90	Libya	61	...	78	8.877515
## 94	Madagascar	128	...	39	4.622437
## 95	Malawi	37	...	18	1.934140
## 100	Mauritania	8	...	57	1.720557
## 107	Mozambique	76	...	38	2.431577
## 108	Namibia	16	...	55	6.296969
## 109	Nepal	57	...	21	1.956288
## 112	Nicaragua	14	...	57	2.113350
## 114	Nigeria	1932	...	52	9.372290
## 120	Papua New Guinea	8	...	13	0.894152
## 142	South Sudan	35	...	25	3.126752
## 149	Syria	43	...	60	2.457050
## 151	Tajikistan	15	...	27	1.572715

## 682. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

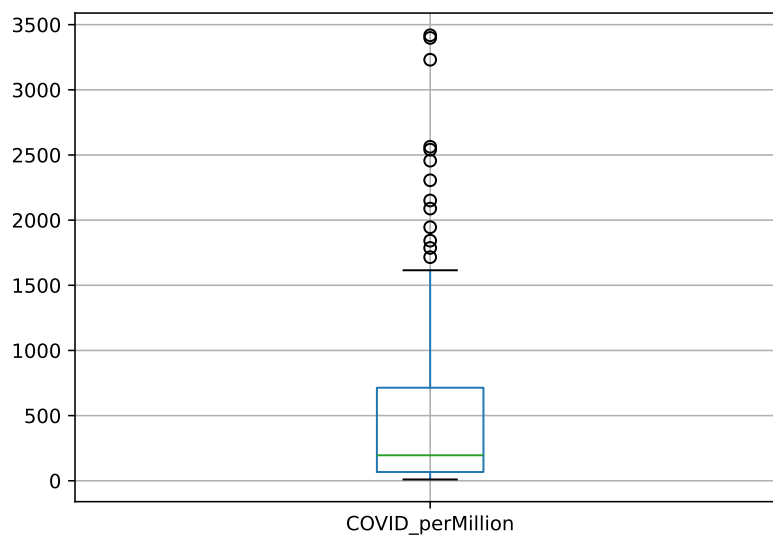
```
## 152      Tanzania      480 ...      37      8.035595
## 160      Uganda       83 ...      26      1.814564
## 166      Vietnam     270 ...      38      2.773823
## 167  Western Sahara    6 ...      87     10.044548
## 168      Yemen        6 ...      38      0.201167
## 169      Zambia     106 ...      45      5.765897
## 170      Zimbabwe     40 ...      38      2.691260
##
## [33 rows x 6 columns]
```

Az eredmények alapján úgy néz ki, hogy az egymillió főre jutó COVID esetszám szerinti alsó kvintilisbe elsősorban olyan afrikai országok esnek, ahol még 2020.04.30-án egyelőre nem tört ki tömeges járvány!

Ezeket az országokat távolítsuk el a `corona_country` dataframe-ből! Majd ezután tekintsük meg ismét az egymillió főre jutó COVID esetszám hisztogramját, és doboz ábráját!

```
corona_extended = corona_extended[corona_extended['COVID_perMillion'] >
                                   corona_extended['COVID_perMillion'].quantile(.2)]

corona_extended.boxplot(column="COVID_perMillion")
```



Na, ez már kb úgy néz ki, mint egy „egészségesen” jobbra elnyúló eloszlás doboz ábrája! :)

## 2.13. Korrelációs elemzések data frame-ben

Nézzünk rá a numerikus adattípusú oszlopok közti korrelációs mátrixra. Egyedül a **Pop** és **COVID\_Cases** oszlopokat hagyjuk ki a vizsgálatból, mert azok abszolút és nem népességarányos adatok, így csalóka lenne őket szerepeltetni a korrelációs vizsgálatokban, hiszen „triviálisan” korrelálnak: nagyobb népességű országban nyilván több az összes esetszám. :)

Azt, hogy csak két oszlopot ne válasszunk ki egy data frame-ben úgy tudjuk elérni, hogy a data frame oszlopnevei közül egy `isin` módszerrel kiválasztjuk a két kihagyandó oszlopot, majd az eredményt letagadjuk egy `~` jellel. Ezt a műveletet pedig beágyazzuk egy `loc` módszerbe, és meg is vagyunk! :)

```
corona_extended.loc[:,~corona_extended.columns.isin(['COVID_Cases', 'Pop'])]
```

```
##          Country  PopDensity  UrbanPop  COVID_perMillion
## 0      Afghanistan         60         25      55.769130
## 1          Albania        105         63      268.608244
## 2          Algeria         18         73       91.354724
## 5  Antigua and Barbuda        223         26      245.075514
## 6          Argentina         17         93       97.973762
## ..          ...          ...          ...          ...
## 161         Ukraine         75         69      237.939741
## 162  United Arab Emirates        118         86      1261.930506
## 163         United Kingdom        281         83      2540.744366
## 164          Uruguay         20         96       185.103621
## 165         Uzbekistan         79         50       60.921678
##
## [130 rows x 4 columns]
```

Erre az oszlopaiban megvágott data frame-re pedig egy `corr` nevű metódust tudunk alkalmazni, ami megadja a numerikus oszlopok közti korrelációk mátrixszát. Figyeljünk még arra, hogy a **Country** oszlopot is ki kell szedni a korrelációs számításban érintett oszlopok közül, hiszen nem numerikus adattípusú, így a korrelációs számítás nem értelmezett rajta! Ennyire azért nem okos ez a pitonka kígyócska! :)

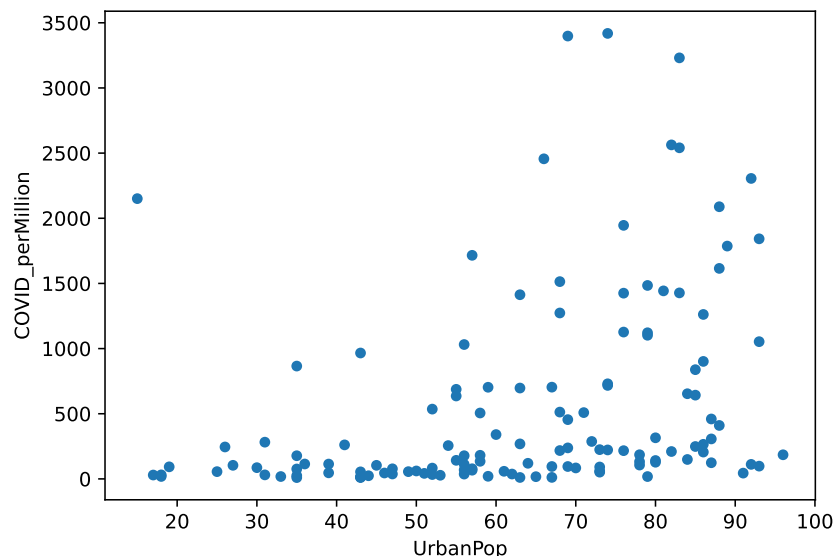
```
corona_extended.loc[:,~corona_extended.columns.isin(['Country', 'COVID_Cases', 'Pop'])].corr()
```

```
##          PopDensity  UrbanPop  COVID_perMillion
## PopDensity      1.000000 -0.051282      0.118036
## UrbanPop        -0.051282  1.000000      0.341160
## COVID_perMillion  0.118036  0.341160      1.000000
```

A korrelációs mátrixból látszik, hogy az egymillió főre jutó COVID esetszám leginkább a városi népesség arányával függ össze, teljesen logikus módon: egyirányú, közepes erősségű a kapcsolat. A zsúfolt városi közösségi tereken, tömegközlekedésen könnyebb megfertőződni. :)

Nézzük is meg a kapcsolatot pontdiagramon! Teljesen úgy működik a pontdiagram is, mint pl. a korábbiakban a magyar adatokon látott területdiagram, csak a metódus neve nem `plot.area`, hanem `plot.scatter`. :)

```
corona_extended.plot.scatter(x="UrbanPop", y="COVID_perMillion")
```

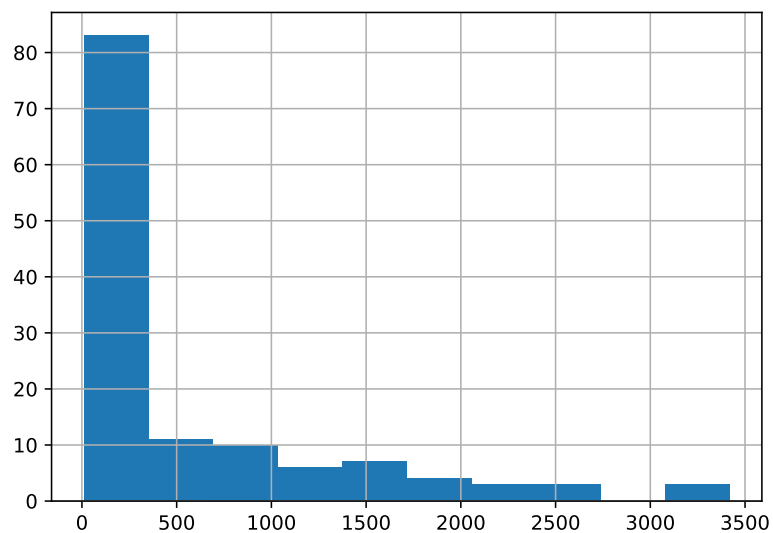


A pontdiagramon azt vehetjük észre, hogy az egymillió főre jutó COVID esetszám jobbra elnyúló eloszlása miatt jelenlévő felfelé kiugró értékek befolyásolják a két ismerv kapcsolatát. A kilógóan magas esetszámok miatt úgy tűnik, mintha az 500 alatti esetszámú országokban nem is lenne kapcsolat a két ismerv között. A városi népesség arányával nincsenek ilyen problémák, mivel annak eloszlása közel szimmetrikus.

A két ismerv/oszlop eloszlásáról írtakat a hisztogramokon is meg lehet lesni.

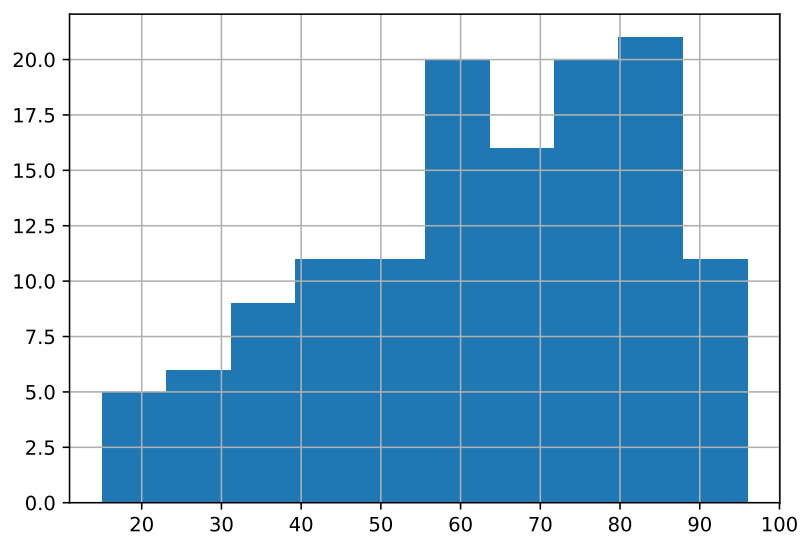
Az egymillió főre jutó esetszám hisztogramja, ami elég jobbra elnyúló.

```
corona_extended.COVID_perMillion.hist()
```



És a városi népesség arányáé, ami szimmetrikusabb egy fokkal, de némileg inkább balra elnyúló. A lényeg, hogy ezen nem segít a logaritmus. :)

```
corona_extended.UrbanPop.hist()
```



## 722. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

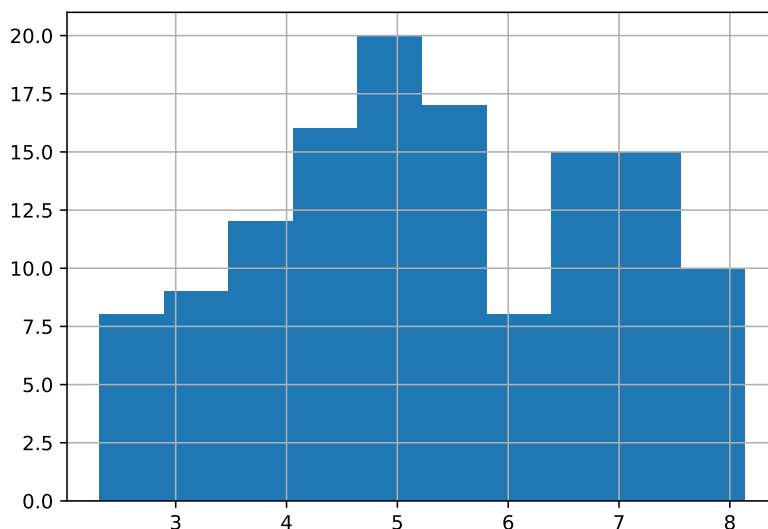
A kiugró értékek hatását, és az **eloszlás jobbra elnyúlóból közel szimmetrikussá alakítását** **logaritmussal** lehet elérni.

Készítsük is el a **COVID\_perMillion** oszlop természetes alapú logaritmusát egy új oszlopban a data frame-n belül.

```
corona_extended['log_COVID_perMillion'] = np.log(corona_extended['COVID_perMillion'])
```

Ezek után lessünk rá az új oszlop hisztogramjára:

```
corona_extended.log_COVID_perMillion.hist()
```



Sokkal szebb! :) Legalábbis szimmetria szempontjából biztos. Viszont van benne azért egy kétmódusú jelleg. Ez azt jelenti, hogy van az országoknak egy jelentősebb csoportja, ahol emelkedettebb a népességarányos COVID esetszám, mint az országok többségében, akik az alacsonyabb értéktartományban lévő „első móduzt” adják.

A korreláció az esetszám logaritmusa és a városi népesség aránya között pedig feljavul. Abszolút értékben több, mint 0.1 egységet emelkedik a korreláció, ami nem elhanyagolható mértékű javulás. :) Most a korrelációs mátrixból kivesszük a **PopDensity**-t is, hogy áttekinthetőbb legyen.

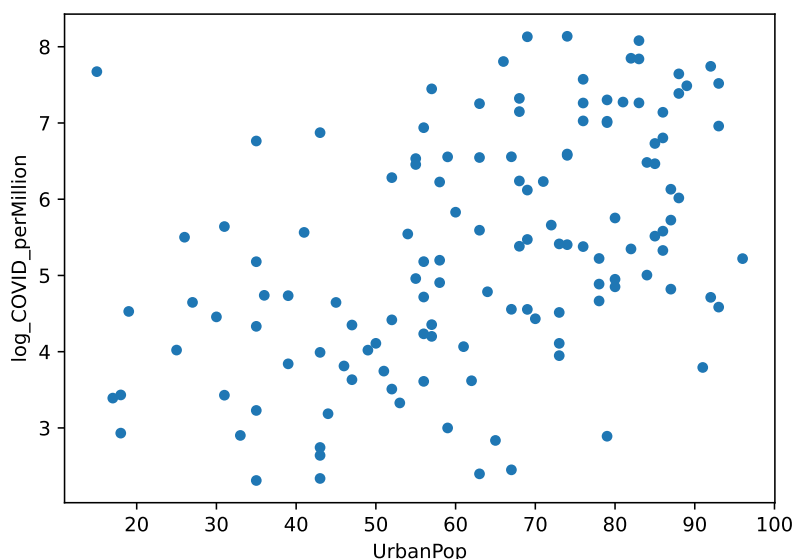
```
corona_extended.loc[:,~corona_extended.columns.isin(['Country', 'COVID_Cases', 'Pop',
```



```
##               UrbanPop  COVID_perMillion  log_COVID_perMillion
## UrbanPop          1.000000          0.341160          0.464379
## COVID_perMillion  0.341160          1.000000          0.826654
## log_COVID_perMillion 0.464379          0.826654          1.000000
```

A korreláció abszolút értékében bekövetkezett javulás oka szépen látható a pontdiagramon: nincsenek már olyan durva outlierek a pontdiagramon. A pontokra nagyobb pontossággal illeszthető egy képzeletbeli egyenes a teljes tartományon nem csak az 500 feletti egymilliófőre vetített esetszámmal bíró országokban.

```
corona_extended.plot.scatter(x="UrbanPop", y="log_COVID_perMillion")
```



Annyit lehet látni, hogy van egy ország, aminek hatalmas az egymillió főre jutó COVID esetszáma az elég alacsony, 20% alatti városi népesség arányához képest. Jó lenne rájönni mi ez az ország!

Ehhez csináljunk egy olyan verziót az előző pontdiagramból, amin minden ponton szerepel, hogy az melyik országot jelöli.

Ennek elkészítéséhez felhasználunk egy `enumerate` névre hallgató függvényt. Ha ezt a függvényt ráeresztjük a **Country** oszlopra a data frame-ben, és az eredményt egy `for` ciklussal bejárjuk, akkor igazából két listát is bejárunk prhuzamosan:

- Egyet, ami az ország sorszámát mutatja a data frame-ben 0-tól indexszelve. Ezt hívom én **sorszám**-nak.

- A másik listában pedig az országnevek vannak. Ez a kódban `szöveg`-nek becézem.

Fontos, hogy a két listát bejáró változó neve teljesen tetszőleges, akár „kismacska” és „gumimaci” is lehetnének. :)

```
for sorszám, szöveg in enumerate(corona_extended.Country):
    print(sorszám)
    print(szöveg)
```

```
## 0
## Afghanistan
## 1
## Albania
## 2
## Algeria
## 3
## Antigua and Barbuda
## 4
## Argentina
## 5
## Armenia
## 6
## Australia
## 7
## Austria
## 8
## Azerbaidzsan
## 9
## Bahamas
```

És ez így folytatódik tovább a data frame összes sorára, csak most ide nem íratom ki a több mint 100 értéket. :)

Na, ezt az `enumerate`-t használó `for` ciklust úgy hasznosítjuk, hogy először egy külön `fig` című objektumba elmentjük az alap pontdiagramos ábrát, amit az előbb is megcsináltunk. Aztán elindítjuk ezt a `for` ciklust az `enumerate` alapján, és a cikluson belül használjuk a `fig` objektum `annotate` metódusát, ami a pontok feliratozását valósítja meg. A metódus paramétereiben megadom először, hogy az aktuális `szöveg`-et, azaz az országnevet rakja fel, mint felirat. A következő paraméter, ami zárójelben van az csak optikai tuning. Ott azt csinálom, hogy az  $x, y$  koordinátáknak megfelelő oszlopok konkrét, pontdiagramon lévő koordinátáit kérdezem le az oszlopok `iat` tulajdonságában. Ez két lista, így mindig a *sorszámadik* elemét nézem a cikluson belül. Ezen koordináták közül a diagram  $x$  tengelyét adó **UrbanPop**-ét eltolom 0.05-tel. Így a pont felirata nem a pont középpontjában kezdődik, hanem attól

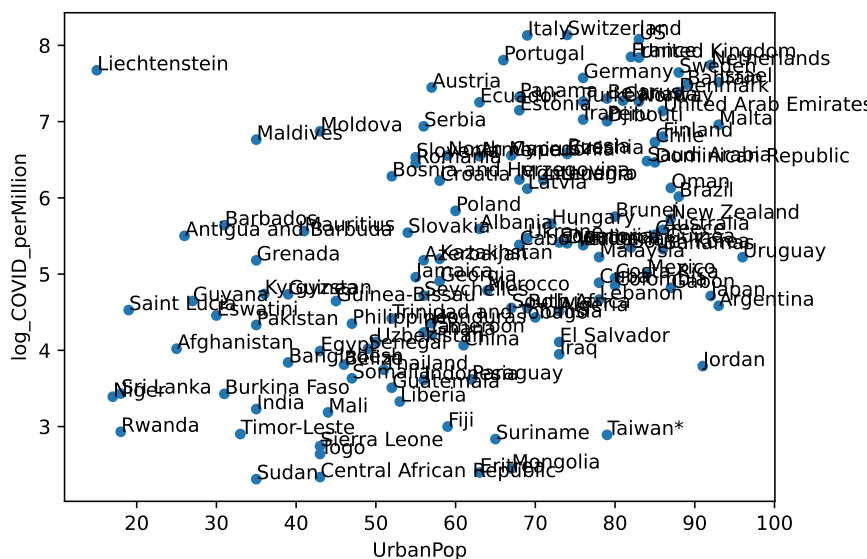
0.05 egységgel jobbra. Így olvashatóbb lesz a cucc. :) Nyilván a felirat  $y$  koordinátáját is tudnám itt szabályozni, és kedvem szerint fel-le rakni a felirat kezdőpontját, de erre itt nincs szükség, így az `annotate` paraméterben ezt a koordinátát csak változatlanul átadom.

Na, és lássuk is ezt a csodát működés közben! A kód végén egy `plt.show()` utasítással lehet a diagramot láthatóvá is tenni.

```
fig = corona_extended.plot.scatter(x="UrbanPop", y="log_COVID_perMillion")

for sorszám, szöveg in enumerate(corona_extended.Country):
    fig.annotate(szöveg, (corona_extended.UrbanPop.iat[sorszám]+0.05, corona_extended.log_COVID_perMillion.iat[sorszám]))

plt.show()
```



E voilá: a gyanús kis államunk a magas esetszámával a kis városi népesség arány ellenére *Lichtenstein*! :) Érdekes észrevenni még, hogy pl. Taiwan elég jól áll: a városi népesség arányához képest elég alacsony az esetszáma! Magyarországot, ha jól szemmelverjük, akkor látható, hogy gyakorlatilag pont a fő csapásirány közepén van kb: pont annyi nagyjából az esetszáma, amennyi a városi népesség aránya alapján „lennie kéne”. :)

## Gyakorló feladatok

1. Olvassuk be az `index_2019_-_pour_import_1_1.csv` nevű fájlt, és mentjük el a beolvasott adatokat egy **PressLiberty** nevű data frame-be!
  - Vigyázat! A fájlban tizedesvesszők vannak tizedes pont helyett! Használni kell a `read_csv` függvény `decimal` paraméterét! Meg kell a paraméterben adni, hogy a tizedeshelyeket a `,` karakter jelöli!
2. A **PressLiberty** data frame-ből csak az angol országnév (**EN\_country**) és a 2019-es sajtószabadsági index (**Score 2019**) oszlopokra lesz szükségünk. A sajtószabadsági indexben az alacsonyabb érték jelent szabadabb sajtót egy országban. A többi változót töröljük ki a data frame-ből!
3. A szűkített **PressLiberty** data frame oszlopainak neve legyen **Country** és **PressLiberty**!
4. Változtassuk meg az Egyesült Államok nevét „*United States*”-ről „*US*”-re a **PressLiberty** data frame-ben, hogy az összeköthető legyen a **corona\_extended** data frame-el az országneveken keresztül!
5. Inner Join művelet segítségével vezessük át a sajtószabadsági index vonatkozó adatokat a **corona\_extended** data frame-be!
6. Ábrázoljuk a **corona\_extended** data frame-ben a kapcsolatot a **log\_COVID\_perMillion** és **PressLiberty** ismérvek között pontdiagramon! Értelmezze röviden szövegesen is a kapcsolatot! Logikus-e a kapcsolat iránya?
7. Vizsgáljuk meg a **PressLiberty** eloszlását hisztogramon!
8. Adjuk hozzá a **corona\_extended** data frame-hez a **PressLiberty** logaritmusát **log\_PressLiberty** néven!
9. Nézzük meg a korrelációs mátrixot a **log\_COVID\_perMillion**, **PressLiberty** és **log\_PressLiberty** ismérvek között! Volt-e értelme a logaritmus alkalmazásának? Válaszát röviden indokolja!
10. Ábrázoljuk a **corona\_extended** data frame-ben a kapcsolatot a **COVID\_perMillion** logaritmusa és a **PressLiberty** logaritmusa között pontdiagramon! Az egyes pontokon szerepeljen az országok neve is!
  - Van-e olyan ország, amelyik a két ismerv kapcsolatát leíró általános tendenciához képest eltérően viselkedik? Válaszát röviden indokolja!

## Gyakorló feladatok megoldása

### 1. feladat

```
PressLiberty = pd.read_csv('index_2019_-_pour_import_1_1.csv', decimal=',')
PressLiberty.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 180 entries, 0 to 179
## Data columns (total 14 columns):
## #   Column                Non-Null Count  Dtype
## ---  ---
## 0   ISO                    180 non-null   object
## 1   Rank2019               180 non-null   int64
## 2   FR_Country             180 non-null   object
## 3   EN_country             180 non-null   object
## 4   ES_country             180 non-null   object
## 5   Score A                180 non-null   float64
## 6   Sco Exa                180 non-null   float64
## 7   Score 2019             180 non-null   float64
## 8   Progression RANK       180 non-null   int64
## 9   Rank 2018              180 non-null   int64
## 10  Score 2018             180 non-null   float64
## 11  Zone                   180 non-null   object
## 12  AR_country             180 non-null   object
## 13  FA_country             180 non-null   object
## dtypes: float64(4), int64(3), object(7)
## memory usage: 19.8+ KB
```

## 2. feladat

```
PressLiberty = PressLiberty.loc[:, ['EN_country', 'Score 2019']]
PressLiberty.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 180 entries, 0 to 179
## Data columns (total 2 columns):
## #   Column                Non-Null Count  Dtype
## ---  ---
## 0   EN_country            180 non-null   object
## 1   Score 2019            180 non-null   float64
## dtypes: float64(1), object(1)
## memory usage: 2.9+ KB
```

### 3. feladat

```
PressLiberty.columns = ['Country', 'PressLiberty']
PressLiberty.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 180 entries, 0 to 179
## Data columns (total 2 columns):
## #   Column          Non-Null Count  Dtype
## ---  ---
## 0   Country          180 non-null   object
## 1   PressLiberty      180 non-null   float64
## dtypes: float64(1), object(1)
## memory usage: 2.9+ KB
```

### 4. feladat

```
PressLiberty['Country'] = PressLiberty['Country'].replace('United States', 'US')
```

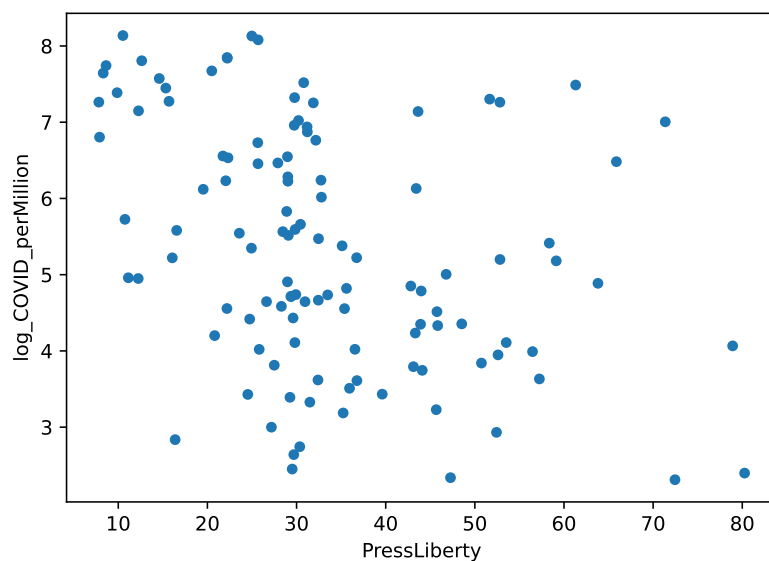
### 5. feladat

```
corona_extended = pd.merge(corona_extended, PressLiberty, how='inner', on='Country')
corona_extended.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 115 entries, 0 to 114
## Data columns (total 8 columns):
## #   Column          Non-Null Count  Dtype
## ---  ---
## 0   Country          115 non-null   object
## 1   COVID_Cases      115 non-null   int64
## 2   Pop              115 non-null   int64
## 3   PopDensity       115 non-null   int64
## 4   UrbanPop         115 non-null   int32
## 5   COVID_perMillion 115 non-null   float64
## 6   log_COVID_perMillion 115 non-null   float64
## 7   PressLiberty      115 non-null   float64
## dtypes: float64(3), int32(1), int64(3), object(1)
## memory usage: 6.9+ KB
```

## 6. feladat

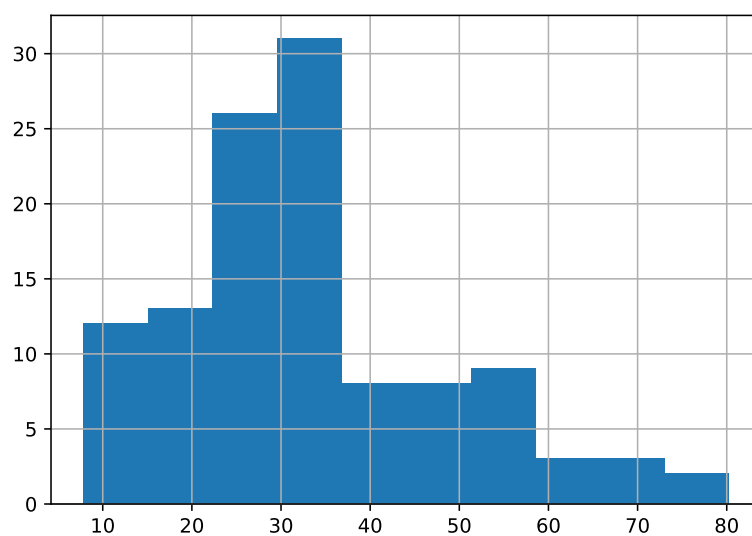
```
corona_extended.plot.scatter(x="PressLiberty", y="log_COVID_perMillion")
```



Úgy látszik, hogy a kapcsolat ellentétes irányú: a sajtószabadsági index növekedésével jellemzően csökken az esetszám. Mivel a magasabb index jelenti a kevésbé szabad sajtót, így első olvasatra nem logikus a kapcsolat iránya: kevésbé szabad sajtóval rendelkező országokban kevesebb az esetszám egymillió főre nézve. De egy picit belegondolva lehet logikus a dolog: a szabadabb sajtóval rendelkező országok jellemzően gazdagabb országok is. Feltehetően ilyen országokban a COVID tesztelésre is több erőforrás jut.

## 7. feladat

```
corona_extended.PressLiberty.hist()
```



Az eloszlás némileg jobbra elnyúló, vannak felső irányban kiugró értékek. Érdemes lehet logaritmust alkalmazni az oszlopon.

## 8. feladat

```
corona_extended['log_PressLiberty'] = np.log(corona_extended['PressLiberty'])
corona_extended.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 115 entries, 0 to 114
## Data columns (total 9 columns):
## #   Column              Non-Null Count  Dtype
## ---  -
## 0   Country             115 non-null   object
## 1   COVID_Cases         115 non-null   int64
## 2   Pop                 115 non-null   int64
## 3   PopDensity          115 non-null   int64
## 4   UrbanPop            115 non-null   int32
## 5   COVID_perMillion    115 non-null   float64
```



```
## 6  log_COVID_perMillion  115 non-null    float64
## 7  PressLiberty          115 non-null    float64
## 8  log_PressLiberty      115 non-null    float64
## dtypes: float64(4), int32(1), int64(3), object(1)
## memory usage: 7.8+ KB
```

## 9. feladat

```
corona_extended.loc[:, ['log_COVID_perMillion', 'PressLiberty', 'log_PressLiberty']].corr()
```

```
##                log_COVID_perMillion  PressLiberty  log_PressLiberty
## log_COVID_perMillion             1.000000      -0.381829      -0.430291
## PressLiberty                    -0.381829       1.000000       0.944278
## log_PressLiberty                 -0.430291       0.944278       1.000000
```

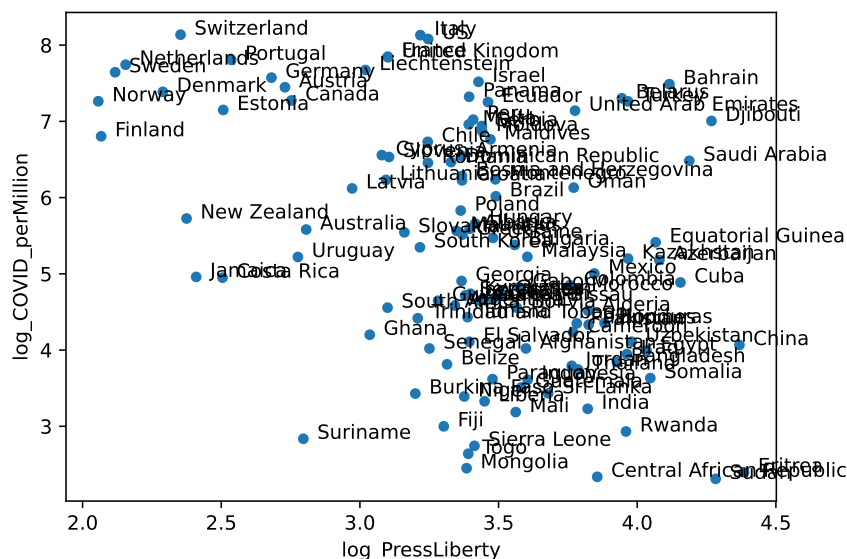
Volt értelme, a sajtószabadsági indexnek jobbra elnyúló az eloszlása, így a kilógó értékek hatását az egymillió főre vetített esetszámmal vett kapcsolatára tudta mérsékelni a logaritmus. Ez onnan látszódik, hogy a korreláció abszolút értékben 0.05 egységgel nőtt. Nem akkora a javulás, mint a városi népesség arányával vett korrelációnál tapasztaltuk, de azért észrevehető.

## 10. feladat

```
fig = corona_extended.plot.scatter(x="log_PressLiberty", y="log_COVID_perMillion")

for sorszám, szöveg in enumerate(corona_extended.Country):
    fig.annotate(szöveg, (corona_extended.log_PressLiberty.iat[sorszám]+0.05, corona_extended.log_

plt.show()
```



Úgy látszik, hogy pl. a dél-amerikai *Suriname*-ben még a viszonylag rossz sajtószabadsági indexhez képest is kevés esettalálható egymillió főre. *Bahrain*-ben és *Szaúd-Arábiában* viszont épp, hogy magas az esetszám a kevésbé szabad sajtó ellenére is. Itt lehet az olajvagyonból futja tesztelésre is *úgy*mond.

:-)

## 3. fejezet

# Leíró Statisztika ismételés és Valószínűségszámítás alapok

### 3.1. Leíró statisztikai mutatók

A Stat. I. PTSD roham kiváltását kezdjük egy nagyon egyszerű kis „Móricka példán”. Vizsgáljuk meg a `TSLA.xlsx` című táblában található adatokat, amik a **TESLA részvények napi záróárfolyam-változásait** mutatják ki **dollárban** (\$) 2019 májusától 2020 májusáig.

Az Excel táblákat, amennyiben az adattáblánk első értéke az *A1* cellában kezdődik és csak egy darab munkalapunk van, gond nélkül be lehet olvasni a `pandas` csomag `read_excel` függvényével egy data frame-be. Több munkalap esetén a `read_excel` függvény `sheet_name` paraméterével tudjuk megadni a beolvasandó munkalap nevét stringként. Viszont ahhoz, hogy ez működőképes legyen fel kell még telepítenünk egy `openpyxl` című kiegészítő csomagot. A biztonság kedvéért ezzel a művelettel együtt rögtön importáljuk a statisztikai számításokhoz szükséges `numpy` és az ábrák megjelenítéséhez szükséges `matplotlib` csomagokat is.

```
pip install openpyxl
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

Ezek után pedig akkor jöhet az Excel beolvasás data frame-be!

## 843. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

```
Tesla = pd.read_excel("TESLA.xlsx")
```

```
Tesla.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 250 entries, 0 to 249
## Data columns (total 2 columns):
## #   Column  Non-Null Count  Dtype
## ---  ---
## 0   Dátum    250 non-null    datetime64[ns]
## 1   TESLA    250 non-null    float64
## dtypes: datetime64[ns](1), float64(1)
## memory usage: 4.0 KB
```

```
Tesla.head()
```

```
##           Dátum      TESLA
## 0 2019-05-07  -8.279998
## 1 2019-05-08  -2.220002
## 2 2019-05-09  -2.860000
## 3 2019-05-10  -2.459992
## 4 2019-05-13 -12.510009
```

Láthatjuk, hogy két oszlopunk van: a dátum és a részvény árváltozása az adott napon az előző napi záróárfolyamhoz képest a **TESLA** oszlopban. Tehát 2019.05.07-én egy Tesla részvény kb. 8.3 dollárral ért kevesebbet a nap végére, mint amennyit 2019.05.06-án ért nap végén. Ellenben 05.13-án már 12.5 dollárral ér kevesebbet, mint előző nap, 05.12-én. Az `info` metódus alapján  $N = 250$  napnyi ilyen adatunk van, ami nagyjából meg is egyezik egy évben a tőzsdei kereskedési napok számával.

Nos, az első öt vizsgált nap alapján nem lennék Elon Musk helyében, elég szép mínuszokat produkál a részvénye. De lássuk, hogy a leíró statisztikai mutatók mit árulnak el, hogyan is teljesített ez a csodacég a teljes vizsgált 1 éves időtartamban! Vessük át be a data frame `describe` metódusát!

```
Tesla.describe()
```

```
##           Dátum      TESLA
## count                250  250.000000
## mean  2019-11-02 15:56:09.600000    1.783920
## min           2019-05-07 00:00:00 -152.359986
## 25%           2019-08-05 06:00:00   -3.770005
## 50%           2019-10-31 12:00:00    1.420006
```

## 75%	2020-02-02 06:00:00	6.957486
## max	2020-05-01 00:00:00	129.429993
## std	NaN	27.192611

Lássuk hát milyen sztorit mesélnek a kiszámított mutatóink!

- Úgy látszik, hogy abszolút értékben a legnagyobb veszteség ( $-152\$$ ) némileg nagyobb, mint a legnagyobb nyereség ( $+129\$$ ).
- Ugyanakkor, egy átlagos napon nagyjából jól járunk egy Tesla részvénnyel, mert kb.  $\mu = Y = 1.8\$$ -al növeli értékét.
- Viszont, marha nagy a rizikó a rendszerben, mert a szórás (angolul *standard deviation*, rövidítve **std**) alapján egy véletlenszerűen kiválasztott napon az árváltozás az  $\sigma = 1.8\$$ -os átlagtól várhatóan  $\pm 27.2\$$ -al térhet el. Azaz az árváltozások várható ingadozása az átlagos nyereségnek mintegy  $V = \frac{\sigma}{\mu} = \frac{27.2}{1.8} = 15.1$ -szerese! (*relatív szórás*)
- A medián árváltozás alapján azt mondhatjuk el, hogy a vizsgált időszakban a napok felében az elérhető maximális nyereség  $Me = 1.42\$$ , míg a napok másik felében a nyereség pedig legalább ennyi.
- Az alsó kvartilis alapján a kereskedési napok legrosszabb  $\frac{1}{4}$ -ében a veszteség nagyobb, mint  $3.77\$$ . Másképp: az árváltozás a napok negyedében kisebb, mint  $Q_1 = -3.77\$$ .
- A napjaink legjobb 25%-ban pedig a nyereség legalább  $Q_3 = 6.96\$$

Ha a fenti megállapításokat összenézzük, akkor arra juthatunk, hogy az **árváltozások eloszlása kb. szimmetrikus** lehet, egy **enyhe jobbra elnyúlással**. A **szimmetria mellett szól**, hogy az átlag nem nagyon tér el a mediántól (tehát a kilógó értékekre érzékeny átlag nem nagyon mozog el a kilógó értékekre robusztus felezőponttól), és a medián nagyjából egyenlő távolságra van az alsó és felső kvartilisektől (szóval az 50%-os pont a gyakoriságok szerint, értékekben is kb. a 25% és 75%-os pontok közepén helyezkedik el). Ellenben az **enyhe jobbra elnyúlás mellett érvel**, hogy azért mégis az átlag enyhén nagyobb mediánnál (tehát a felfelé kilógó értékek kicsit felfelé húzzák az átlagértéket a felezőponthoz képest), és a medián enyhén közelebb van az alsó kvartilishoz, mint a felsőhöz (tehát az adatok többsége egy kicsit inkább az értéktartomány aljára koncentrálódik  $\rightarrow$  a kisebb értékből egy kicsit több van). De ezek tényleg nagyon enyhe eltérések. Sőt, **akár még a balra elnyúló eloszlás felé is lehet érvelni** azzal, hogy  $Q_3$  közelebb van a maximumhoz, mint  $Q_1$  a minimumhoz, tehát a *felső* 25%-ban lévő értékek „kevésbé húznak szét”, nem annyira kilógóak, mint az *alsó* 25%-ban lévők.

### 3.1.1. Hisztogram és Alakmutatók

Az előző bekezdés dilemmáit leginkább **egy hisztogram segítségével tudnánk tisztázni**. A **hisztogramhoz** viszont szükségünk van egy

**osztályközös gyakorisági táblára**, hiszen az árfolyamváltozások értékkészlete elég tág. Az osztályközöket vegyük egyenlő hosszúra. Ezek után már csak azt kell eldöntenünk, hogy **hány osztályközt** hozzunk létre! Ezt ugye Stat. I-ben legtöbbször a „ $2^k$  szabállyal” adtuk meg. Tehát az osztályközök száma a legkisebb olyan  $k$  szám, amire igaz az, hogy  $2^k \geq N$ , ahol  $N$  az adataink elemszáma. Azt láttuk a `describe` eredményéből is pl., hogy  $N = 250$ . Ez alapján pedig a keresett  $k$  az 8 lesz, mert  $k = 8 : 2^8 = 256 > 250$ , ám  $k = 7 : 2^7 = 128 < 250$ . Ha valaki ezt pironul szeretné kiszámolni arra figyeljen, hogy ott a hatványozás jele a **\*\***.

```
2**7
```

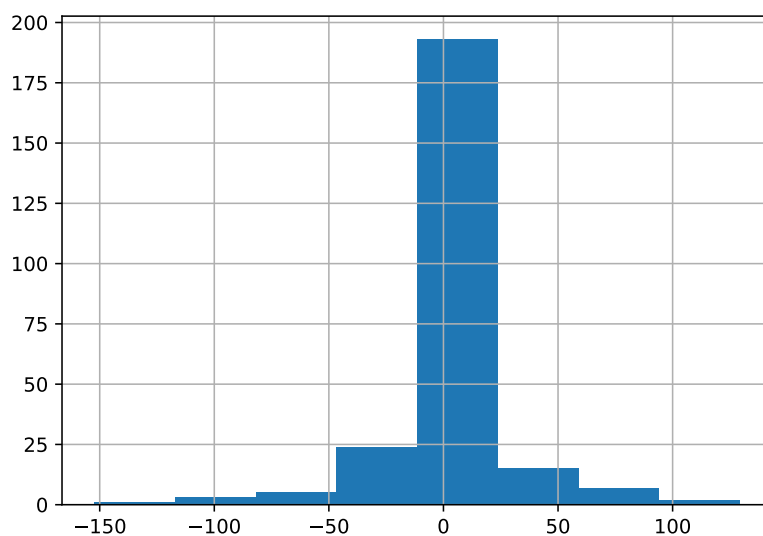
```
## 128
```

```
2**8
```

```
## 256
```

Akkor hát nézzük meg a 8 egyenlő hosszú osztályközzel bíró gyakorisági tábla alapján készített hisztogramot.

```
Tesla.TESLA.hist(bins = 8)
```



Nagyon szép, tényleg sac/kb szimmetrikus az eloszlás: középen, az 1.8\$-os átlag körül csoportosul a legtöbb elem, és az ennél kisebb és nagyobb értékekből arányosan kevesebb van. Bár némileg kicsit csúcsosnak néz ki az eloszlás: a középső, átlag körüli, leggyakoribb értéktartományra koncentrálódik az adatok legnagyobb része, kb. 190 nap értéke az  $N = 250$ -ból. Lássuk mi mondanak erről az  $\alpha_3, \alpha_4$  **alakmutatók!**

Az  $\alpha_3$  **aszimmetria mutató** Pythonban egy data frame oszlop `skew` metódusával, míg az  $\alpha_4$  **csúcsossági mutató** az oszlop `skew` metódusával számítható.

```
Tesla.TESLA.skew()
```

```
## -0.5253045816998407
```

```
Tesla.TESLA.kurt()
```

```
## 9.086855960563858
```

Az  $\alpha_3$  értéke nagyon picit negatív, így enyhe balra elnyúlást jelez, de a hisztogram alapján látszik, hogy ez tényleg nagyon gyenge tendencia. Ez ugyebár abból adódik hogy  $Q_3$  közelebb van a maximumhoz, mint  $Q_1$  a minimumhoz. Tehát ezt a nagyon enyhe „balra elnyúlást” csak a maximum némileg kilógó viselkedése okozza, amire az  $\alpha_3$  ugyebár érzékeny, hiszen az átlag ( $\mu$ ) alapján számoljuk őt ki (átlag körüli harmadik momentum). Ellenben az  $\alpha_4 = +9.09$ -es nagyon erősen pozitív értéke egyértelműen csúcsos eloszlás mutat, amit gyönyörűen látunk is a hisztogramon.

### 3.1.2. Gyakorisági tábla lekérése

Ha szeretnénk **megtekinteni a hisztogram mögött lakó osztályközös gyakorisági táblát**, akkor a dolgunk annyi, hogy a `hist` metódus helyett a `numpy` csomag `histogram` függvényével készítsük el a hisztogramot, és az eredményt mentjük el egy külön objektumba. A `histogram` függvény első paramétere az a data frame oszlop, amiből hisztogramot készítenénk, míg a második paramétere a `bins`, ami ugyan az, mint a data frame `hist` metódusában. Az elmentett eredményt data frame-é konvertáljuk a `pandas` csomag `DataFrame` függvénye segítségével, majd az eredményt transzponáljuk (alias 90 fokkal elforgatjuk) a data frame-k `transpose` metódusa segítségével.

```
gyaktábla = np.histogram(Tesla.TESLA, bins = 8)
gyaktábla = pd.DataFrame(gyaktábla).transpose()
gyaktábla
```

## 883. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

```
##          0          1
## 0    1.0 -152.359986
## 1    3.0 -117.136239
## 2    5.0 -81.912491
## 3   24.0 -46.688744
## 4  193.0 -11.464996
## 5   15.0  23.758751
## 6    7.0  58.982498
## 7    2.0  94.206246
## 8   NaN 129.429993
```

Amit kaptunk az egy olyan tábla, aminek az **első oszlopa a gyakoriságok** értéke, míg a **második az adott osztályköz alsó határa**. Ezért van az utolsó sorban üres (NaN) érték, mert az ottani alsó határ az a vizsgált ismervünk (árváltozásunk) maximuma, ami fölött természetesen már nincs érték.

Nevezzük át tartalmuknak megfelelően az oszlopokat, majd a **shift** módszer segítségével rakjuk be a táblába az adott osztályközök felső határait is. Itt a **shift** módszerrel az alapelv, hogy az adott osztályköz felső határa nem más, mint a következő sor alsó határa.

```
gyaktábla.columns = ['Gyakoriság', 'AlsóHatár']
gyaktábla.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 9 entries, 0 to 8
## Data columns (total 2 columns):
## #   Column      Non-Null Count  Dtype
## ---  ---
## 0   Gyakoriság    8 non-null      float64
## 1   AlsóHatár     9 non-null      float64
## dtypes: float64(2)
## memory usage: 276.0 bytes
```

```
gyaktábla['FelsőHatár'] = gyaktábla.AlsóHatár.shift(-1)
gyaktábla
```

```
##   Gyakoriság  AlsóHatár  FelsőHatár
## 0         1.0 -152.359986 -117.136239
## 1         3.0 -117.136239  -81.912491
## 2         5.0  -81.912491  -46.688744
## 3        24.0 -46.688744  -11.464996
## 4       193.0 -11.464996   23.758751
## 5        15.0  23.758751   58.982498
## 6         7.0  58.982498   94.206246
```



```
## 7          2.0    94.206246  129.429993
## 8          NaN   129.429993          NaN
```

Na, ez egész pofás! Már csak annyi van, hogy rendezzük logikus sorrendbe az oszlopokat, és töröljük azt a nyomorult utolsó sort a NaN-nal.

```
gyaktábla = gyaktábla[['AlsóHatár', 'FelsőHatár', 'Gyakoriság']]
gyaktábla = gyaktábla.drop(8, axis = "index")
gyaktábla
```

```
##      AlsóHatár  FelsőHatár  Gyakoriság
## 0 -152.359986 -117.136239          1.0
## 1 -117.136239 -81.912491          3.0
## 2  -81.912491 -46.688744          5.0
## 3  -46.688744 -11.464996         24.0
## 4  -11.464996  23.758751        193.0
## 5   23.758751  58.982498         15.0
## 6   58.982498  94.206246          7.0
## 7   94.206246 129.429993          2.0
```

Na, ez végre tök szépen olvasható! :) Láthatjuk például, hogy 5 olyan kereskedési napunk volt a vizsgált időszakban, amikor az árfolyamváltozás  $-81\$$  és  $-46\$$  között volt, azaz a Tesla 46 és 81 dollár közti *veszteséget* produkált ezen az 5 napon. Ellenben 15 napon  $23\$$  és  $58\$$  dollárt lehetett kaszálni egy Tesla részvényen. De amint a hisztogramon is látszott: a legtöbb, 193 napon az árfolyamváltozások a  $11\$$  *veszteség* és a  $23\$$  *nyereség* között (tehát úgy kb az  $1.8\$$  átlag környékén) ingadoztak.

### 3.1.3. Gyakorisági tábla bővítése

Ha szeretnénk, akkor a **Relatív Gyakoriságokat** is ki tudjuk számítani a táblába: ugyebár minden gyakoriságot leosztunk a teljes elemszámmal ( $N$ ), ami a gyakoriságok összege. Ez Pythonban úgy néz ki, hogy a data frame gyakoriság oszlopát elosztjuk annak összegzett verziójával. Az összeg alias `sum` függvényt itt a `numpy` csomagból raboljuk el.

```
gyaktábla['RelatívGyak'] = gyaktábla.Gyakoriság / np.sum(gyaktábla.Gyakoriság)
gyaktábla
```

```
##      AlsóHatár  FelsőHatár  Gyakoriság  RelatívGyak
## 0 -152.359986 -117.136239          1.0         0.004
## 1 -117.136239 -81.912491          3.0         0.012
## 2  -81.912491 -46.688744          5.0         0.020
```

## 3	-46.688744	-11.464996	24.0	0.096
## 4	-11.464996	23.758751	193.0	0.772
## 5	23.758751	58.982498	15.0	0.060
## 6	58.982498	94.206246	7.0	0.028
## 7	94.206246	129.429993	2.0	0.008

Szuper, így már láthatjuk, hogy az az 5 nap, amikor 46\$ és 81\$ közti *veszteségünk* volt az az összes vizsgált napnak 2%-át jelenti.

Lehet **kumulálni** is a `cumsum` metódus segítségével. Számítsuk is ki így a *kumulált relatív gyakoriságot*!

```
gyaktábla['KumRelatívGyak'] = gyaktábla.RelatívGyak.cumsum()
gyaktábla
```

##	AlsóHatár	FelsőHatár	Gyakoriság	RelatívGyak	KumRelatívGyak
## 0	-152.359986	-117.136239	1.0	0.004	0.004
## 1	-117.136239	-81.912491	3.0	0.012	0.016
## 2	-81.912491	-46.688744	5.0	0.020	0.036
## 3	-46.688744	-11.464996	24.0	0.096	0.132
## 4	-11.464996	23.758751	193.0	0.772	0.904
## 5	23.758751	58.982498	15.0	0.060	0.964
## 6	58.982498	94.206246	7.0	0.028	0.992
## 7	94.206246	129.429993	2.0	0.008	1.000

Remek, az utolsó sorban ott a 100%-os kumulált relatív gyakoriság, ahogy kell, és azt is látjuk, hogy a vizsgált napjaink 3.6%-ban volt a *veszteség* nagyobb, mint 46\$ (azaz az árváltozás kisebb, mint -46\$).

### 3.1.4. Súlyozott átlag és szórás Pythonban

Ha szeretnénk pl. **súlyozott átlagot számítani** a gyakorisági táblából azt is minden további nélkül megtehetjük! Kb. **pont ugyan úgy, mint Excelben!** Itt most a *SZORZATÖSSZEG* függvény szerepét a numpy-féle `sum` függvény tölti be! Ezzel ugyan úgy **le tudjuk tükrözni a szummás statos képleteinket Pythonban, mint ahogy azt Excelben megtettük.**

Ugyebár a súlyozott átlaghoz két dolog kellene, a **gyakoriságok**, alias  $f_i$ -k, és az **osztályközepek**, az  $Y_i$ -k. Előbbiek megvannak csak hozzáadom az **f\_i** jelölést az oszlop nevéhez, mígy az  $Y_i$ -ket kiszámolom, mint az osztály alsó és felső határának átlaga egy új **Y\_\_i** nevű oszlopba! Majd a data frame oszlopnevekben mindig `'_'` szimbólummal jelölöm az alsó indexet!

```
gyaktábla = gyaktábla.rename(columns = {"Gyakoriság": "f_i"})
gyaktábla['Y_i'] = (gyaktábla.AlsóHatár + gyaktábla.FelsőHatár) / 2
gyaktábla
```

	AlsóHatár	FelsőHatár	f_i	RelatívGyak	KumRelatívGyak	Y_i
## 0	-152.359986	-117.136239	1.0	0.004	0.004	-134.748112
## 1	-117.136239	-81.912491	3.0	0.012	0.016	-99.524365
## 2	-81.912491	-46.688744	5.0	0.020	0.036	-64.300618
## 3	-46.688744	-11.464996	24.0	0.096	0.132	-29.076870
## 4	-11.464996	23.758751	193.0	0.772	0.904	6.146877
## 5	23.758751	58.982498	15.0	0.060	0.964	41.370625
## 6	58.982498	94.206246	7.0	0.028	0.992	76.594372
## 7	94.206246	129.429993	2.0	0.008	1.000	111.818119

Nagyon jó, így már tudom is alkalmazni a súlyozott átlag képletét:

$$\mu = \bar{Y} = \frac{\sum_i f_i Y_i}{N}$$

```
átlag = np.sum(gyaktábla.f_i * gyaktábla.Y_i) / len(Tesla.TESLA)
átlag
```

```
## 4.456137313500011
```

Remek, hát az osztályközepek használatával kicsit felélőttem a valóságnak (ami kb. 1.8\$ volt ugyebár), de hát ez ugye csak egy becslés. :)

A súlyozott szórást ugyan ezzel az elvvel ki lehet számolni pythonkával a képlete alapján:

$$\sigma = \sqrt{\frac{\sum_i f_i (Y_i - \bar{Y})^2}{N}}$$

```
gyaktábla
```

	AlsóHatár	FelsőHatár	f_i	RelatívGyak	KumRelatívGyak	Y_i
## 0	-152.359986	-117.136239	1.0	0.004	0.004	-134.748112
## 1	-117.136239	-81.912491	3.0	0.012	0.016	-99.524365
## 2	-81.912491	-46.688744	5.0	0.020	0.036	-64.300618
## 3	-46.688744	-11.464996	24.0	0.096	0.132	-29.076870
## 4	-11.464996	23.758751	193.0	0.772	0.904	6.146877
## 5	23.758751	58.982498	15.0	0.060	0.964	41.370625
## 6	58.982498	94.206246	7.0	0.028	0.992	76.594372
## 7	94.206246	129.429993	2.0	0.008	1.000	111.818119

```
szórás = np.sqrt(np.sum(gyaktábla.f_i * (gyaktábla.Y_i-átlag)**2) / len(Tesla.TESLA))
szórás
```

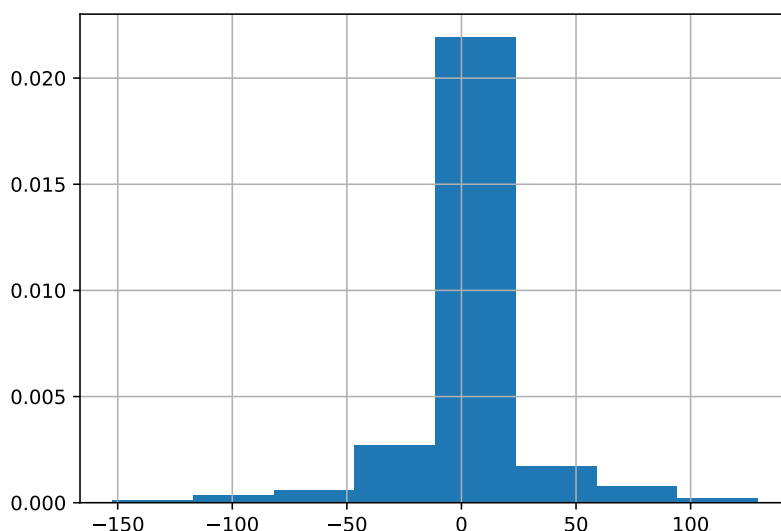
```
## 27.048902512450574
```

Na, ezt már nem löttük annyira mellé a valós 27.19\$-hez képest. Ezzel meg is vagyunk, juppí! :)

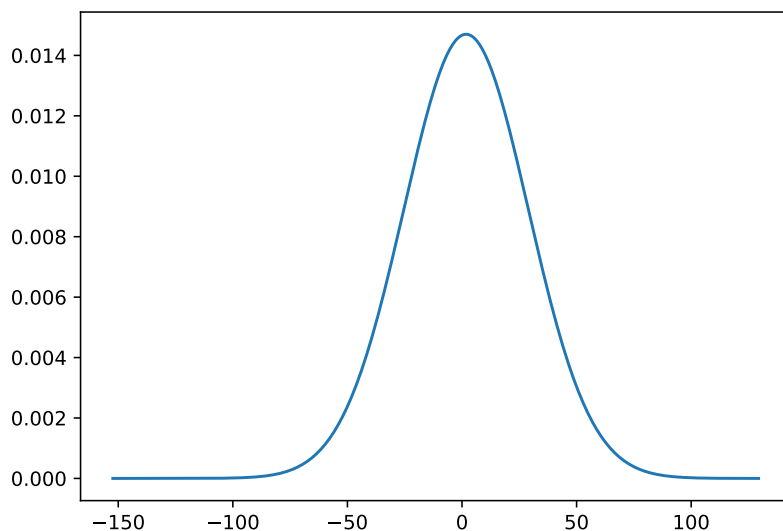
### 3.2. A normális eloszlás és sűrűségfüggvénye

Térjünk vissza a Tesla részvényárfolyamok hisztogramjához. Most rajzoljuk ki úgy a cuccot, hogy a `hist` metóduson bekapcsolunk egy `density = True` beállítást. Ez úgy rajzolja ki a hisztogramot, hogy az  $y$  tengelyen nem a gyakoriságok jelennek meg, hanem azoknak egy úgy skálázott verziója, hogy a maximum érték az adott osztály osztályközepének és a körülötte lévő  $\pm 2$  érték együttes relatív gyakoriságával arányos. Mivel egy konkrét érték gyakorisága itt most  $\frac{1}{250}$ , így a maximum érték egy jó alacsony szám lesz az  $y$  tengelyen, egész konkrétan kb.  $\frac{5}{250} = 0.02$ . A többi oszlop magassága az eredeti gyakoriságok szerint legyártott hisztogram alapján van belőve ehhez a maximum értékhez. Szóval, a hisztogram alakja nem változik, csak az  $y$  tengely van máshogy beskálázva.

```
Tesla.TESLA.hist(bins = 8, density = True)
```



Meg is van a csodaszépen szimmetrikus eloszlást mutató hisztogramunk. Ezen a „*relatív gyakoriságos*” hisztogramon azt a gondolatot kell most elképzelni, hogy milyen alakzatot kapunk, ha **az oszlopokat összekötjük egy folytonos vonallal**. Nos, nem kell sokat fantáziálni, a vonnallal összekötés az **alábbi alakzathoz hasonló függvényt eredményez**:



Amit itt látunk az nem más, mint **egy normális eloszlás sűrűségfüggvénye**. Sőt, pontosítok is, ez az **alakzat egy  $\mu = 1.8$  átlagú és  $\sigma = 27.19$  szórású normális eloszlás sűrűségfüggvénye**, hiszen ennyi volt a **Tesla árfolyamváltozások** adatsorának átlaga és szórása, aminek a **hisztogramja alapján gondolatban kirajzoltuk ezt a sűrűségfüggvényt**. Ezt ilyenkor úgy szoktuk szakszerűen mondani, hogy amit látunk az egy  $N(1.8, 27.19)$  eloszlás sűrűségfüggvénye. Általánosságban egy normális eloszlásra pedig  $N(\mu, \sigma)$  jelöléssel hivatkozunk.

Miért is van ez így? Mivel **azt, hogy konkrétan milyen egy normális eloszlású sűrűségfüggvény alakja, meghatározza, hogy mennyi az adatsor átlaga és szórása, amire ezt a sűrűségfüggvényt úgymond illeszteni szeretnénk**. Azt, hogy hogyan az alábbi kis interaktív ábra szemlélteti:

Amúgy a normális eloszlás sűrűségfüggvényének hozzárendelési  $f(x)$  utasítása  $\mu$  átlag és  $\sigma$  szórás függvényében az alább alakot ölti.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Mielőtt szörnyet halunk az Analízis emlékek által kiváltott PTSD-ben,

megnyugtatók mindenkit: erre a épletre igazából nekünk nem lesz szükségünk, de egyszer nem árt ha látjuk a függvény mögötti képletet is. :)

### 3.2.1. A sűrűségfüggvény használata

No, de „*Mit adtak nekünk a rómaiak?*” Azaz jogosan kérdezhetjük, hogy mire tudjuk használni ezt a sűrűségfüggvényt? Első és legfontosabb funkciója, hogy ha a függvény  $f(x)$  formulájába behelyettesítünk egy  $x$  értéket, akkor megkapom, hogy **mi a valószínűsége, hogy az  $Y$  adatsoromból egy véletlenszerűen kihúzott  $Y_i$  érték az  $x$ -et vesz fel.** Magyarul  $f(x) = P(Y_i = x)$ . Most itt egy picit **pontatlan voltam**. Ugyanis nem egész pontosan a  $Y_i = x$  esemény valószínűségét kapjuk meg, mert nagy értékkészlet esetén az gyakorlatilag 0 lenne. Gondoljunk bele: annak a valószínűsége, hogy a Tesla napi árváltozása éppen pont 2.760009\$ az tényleg gyakorlatilag 0, de a valóságban pont ennyi volt a cucc 2019. május 23-án. Szóval, abszolút nem lehetetlen... Azaz, egész pontosan az  $f(x)$  sűrűségfüggvény érték *arányos* a  $P(x < Y_i < x + \epsilon)$  esemény valószínűségével, ahol az  $\epsilon$  egy *nagyon kicsi* szám. Konkrétan, azt mondhatjuk, hogy  $P(x < Y_i < x + \epsilon) = f(x) \times \epsilon$ . Tehát, annak a valószínűsége, hogy a random módon kihúzott  $Y_i$  értékünk az  $x$ -nek egy *nagyon kicsi*,  $\epsilon$ -nyi környezetébe esik, arányos az  $f(x)$  sűrűségfüggvény értékével. Ha nagyobb ez a valószínűség, akkor nagyobb a sűrűségfüggvény érték is, és fordítva. Szóval, annyi biztosan elmondható, hogy amelyik  $x$  pontnak nagyobb az  $f(x)$  sűrűségfüggvény értéke, annak nagyobb is a bekövetkezési valószínűsége, ha véletlenszerűen kiválasztok egy tetszőleges  $Y_i$  értéket. Csak a két dolog (sűrűségfüggvény és valószínűség) nem ugyan az, az eltérésük egy  $\epsilon$  szorzó. Emiatt van az is, hogy a Pythonban a `hist` metódus `density = True` beállítással a relatív gyakoriságokat a legnagyobb gyakoriságú  $Y_i$  osztályközép  $\pm 2$  környezet relatív gyakorisága alapján mutatja meg a hisztogram  $y$  tengelyén. De most nekünk **szemléletes szempontból teljesen jó lesz, ha úgy gondolunk a helyettesítési értékre  $x$  helyen, mint az  $x$  érték bekövetkezési valószínűsége.** Azaz, mi a  $f(x) = P(Y_i = x)$  értelmezéssel megyünk tovább.

Ez alapján, ha meg akarjuk tudni, hogy mennyi a valószínűsége, hogy a Tesla egy random napi árfolyamváltozása épp a 2019. május 23-i kb. 2.76\$-al lesz egyenlő, akkor ahhoz az alábbi csodát kell kiszámolni.

$$P(Y_i = 2.76) = f(2.76) = \frac{1}{27.19\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{2.76-1.8}{27.19}\right)^2}$$

Hiszen azt tudjuk, hogy a megfigyelt kereskedési napok alapján az árváltozások átlaga  $\mu = 1.8\$$  és szórása  $\sigma = 27.19\$$ .

Na, ezt számolja ki kézzel az, aki papíron tanulja a Stat. II-t. :) Mi Pythonban be tudjuk vetni a `scipy` csomag `stats` névterében található függvényeket egy ilyen normális eloszlás sűrűségfüggvény-érték kiszámítására!

Telepítsuk a csomagot és importáljuk a szükséges függvényeket egy `stats` névtérbe. A **csomagot nagyon sokat fogjuk használni a félév során**, és egy szép alapos dokumentációja van. Érdemes olvasgatni! :)

```
pip install scipy
import scipy.stats as stats
```

Majd a névtér `norm.pdf` függvénye segítségével számoljuk ki a keresett valószínűséget! A függvény **3 paraméterrel operál, ebben a sorrendben:  $x, \mu, \sigma$** . Annyit érdemes megjegyezni, hogy a függvény a  $\mu$  átlagot location-nek, azaz `loc`-nak, míg a  $\sigma$  szórást `scale`-nek nevezi a saját kis nyelvjárásában. A függvény neve pedig az angol *probability density function*-ból (valószínűségi sűrűségfüggvény) rövidül *pdf*-nek.

```
mú = 1.8
szigma = 27.19
stats.norm.pdf(x = 2.76, loc = mú, scale = szigma)
```

```
## 0.014663247477553697
```

Szuper, ez azt jelenti, hogy kb. 1.466% a valószínűsége annak, hogy egy véletlenszerű napon egy Tesla részvényt 2.76\$-t lehet kaszálni.

Ezen a ponton **érdemes belegondolni, hogyan is reagált a sűrűségfüggvény a szórás növekedésére...ellaposodott!** Na, ez most teljesen érthetővé válik, hiszen **ha a szórás nő**, az azt jelenti, hogy a **szélsőségesen magas vagy alacsony  $Y_i$  értékek bekövetkezési valószínűsége is megnő**...és ha a **függvény  $f(x)$  értéke épp ezekkel a  $P(Y_i = x)$  valószínűségekkel egyenlő**, akkor épp a két szélén fog „meghízni” a függvény képe, azaz ellaposodik!

### 3.2.2. A sűrűségfüggvény integrálja

Azért láthatjuk, hogy egy konkrét érték bekövetkezési valószínűsége, alapból nem valami nagy, épp azért, amit fejtegettünk korábban is: az árváltozások értékészlete elég nagy, egy konkrét érték (vagy annak kis környezetének) bekövetkezési valószínűsége elég kicsi. Emiatt nem is ezt a kérdést szoktuk általában feltenni a sűrűségfüggvénynek, hanem pl. azt, hogy **mi a valószínűsége annak, hogy egy véletlenszerűen kihúzott  $Y_i$  érték egy előre megadott  $x$  érték alatt helyezkedik el?** Tehát a  $P(Y_i < x)$  valószínűséget keressük általában. Ez pedig nem más, mint a **sűrűségfüggvény  $x$  alatti részének területe**, vagyis a  $\int_{-\infty}^x f(x)dx$  improprius integrál.

Na, ha a sűrűségfüggvény helyettesítési értékét nem akartuk kézzel-lábbal kiszámolni, akkor ezt az improprius integrált meg pláne nem! Szerencsére, **van**

**erre is beépített függvényünk** a `scipy` csomagban `norm.cdf` néven. Ugyan úgy működik, mint a `norm.pdf`, 3 paramétere van, ugyan abban a sorrendben: `x`, `loc`, `scale`, csak a  $P(Y_i < x)$ -et számítja ki, nem a  $P(Y_i = x)$ -t a megadott átlagú és szórású normális eloszlás sűrűségfüggvény alapján. A függvény neve az angol *cumulative density function*-ból (kumulált sűrűségfüggvény) rövidül *cdf*-nek. Ha belegondolunk ez logikus, hiszen felösszegezzük (azaz felkumuláljuk) az egyes  $Y_i$  elemek bekövetkezési valószínűségét  $-\infty$ -tól  $x$ -ig.

Lássuk akkor hát pl., hogy mennyi a valószínűsége, hogy egy véletlenszerűen kiszúrt kereskedési napon a Teslával 82\$-nál nagyobb *veszteségünk* lesz! Tehát, a  $P(Y_i < -82)$  valószínűséget keressük.

```
stats.norm.cdf(x = -82, loc = mű, scale = szigma)
```

```
## 0.0010280208392538434
```

Ez pedig a korábban megadott  $\mu$  átlaggal és  $\sigma$  szórással nem más, mint kb. 0.1%. Szóval, szerencsére egy jó kicsi érték! :)

Természetesen ha egy  $x$  érték alá esési valószínűségét ki tudjuk számolni, akkor az  $x$  felé esés valószínűségét már gyerekjáték kiszámolni, hiszen a „**felé esés**” az „**alá esés**” **komplementer eseménye**, azaz  $P(Y_i > x) = 1 - P(Y_i < x)$ . Szemléletesen pedig itt a **sűrűségfüggvény  $x$  feletti részének területét** számoljuk ki.

Eszerint gyorsan meg tudjuk adni, hogy mi annak a valószínűsége, hogy egy random napon a Tesla részvényen 20\$-nál többet kaszálunk, hiszen  $P(Y_i > 20) = 1 - P(Y_i < 20)$ . Tehát, az egész dolog ismét megoldható a `norm.cdf` függvény segítségével.

```
1 - stats.norm.cdf(x = 20, loc = mű, scale = szigma)
```

```
## 0.25163173906817715
```

Na, ez nem is rossz, a 20\$ feletti nyereség valószínűsége egy napon egy kicsit több, mint 25%!

Ha pedig azt szeretnénk megtudni, hogy mi a valószínűsége, hogy a véletlenszerűen kihúzott  $Y_i$  értékünk épp két előre megadott  $x$  és  $y$  érték közé esik, akkor egyszerűen a **nagyobb érték alá esés valószínűségéből kivonjuk a kisebb érték alá esés valószínűségét**. Azaz, ha  $x > y$ , akkor  $P(y < Y_i < x) = P(Y_i < x) - P(Y_i < y)$ , de ha  $x < y$ , akkor  $P(x < Y_i < y) = P(Y_i < y) - P(Y_i < x)$  a számítás menete. Tehát, ekkor is az egész sztori megoldható pythonban a `norm.cdf` függvénnyel. Grafikusan pedig úgy képzeljük el a dolgot, mint a **sűrűségfüggvény  $x$  és  $y$  közötti részének területe**.



Szóval, ha azt szeretném megtudni, hogy mi a valószínűsége annak, hogy egy véletlenszerűen kiválasztott napon egy Tesla részvénnel 47\$ és 82\$ közti veszteséget produkálunk, akkor megnézem a  $-47$  alá esés valószínűségét, és kivonom belőle a  $-82$  alá esés valószínűségét (tartom a nagyobból vonom a kisebbet elvet ugyebár).

```
stats.norm.cdf(x = -47, loc = mű, scale = szigma) - stats.norm.cdf(x = -82, loc = mű, scale = szigma)
```

```
## 0.035316558328262415
```

Nagyon jó, akkor már azt is tudjuk, hogy kb. 3.5% a valószínűsége, hogy a Tesla egy napon 47\$ és 82\$ közti veszteséget produkál.

**Összefoglalva** tehát a sűrűségfüggvény,  $f(x)$  segítségével a következő események bekövetkezési valószínűsége számítható ki, ahol  $Y_i$  a vizsgált adatsornak egy véletlenszerűen kihúzott  $i$ -edik eleme,  $x$  és  $y$  pedig előre adott számok:

- $x$  bekövetkezése:  $P(Y_i = x) = f(x)$
- $x$  alá esés:  $P(Y_i < x) = \int_{-\infty}^x f(x)dx$
- $x$  felé esés:  $P(Y_i > x) = 1 - P(Y_i < x)$
- $x$  és  $y$  közé esés:  $P(y < Y_i < x) = P(Y_i < x) - P(Y_i < y)$

Mindezen valószínűségek számolása a hozzájuk tartozó sűrűségfüggvény interaktív ábrájával alább tekinthetők át. Egy *apró megjegyzés*: az alá-felé-közé esési valószínűségeknél azért hagytam el mindenhol a  $=$  jelet, mert a nagy értékkészlet miatt ugyebár egy konkrét érték bekövetkezési valószínűsége nagyon kicsi, így a tartományba esésnél elhanyagolható. *Nem oszt, nem szoroz úgymond.*

### 3.2.3. Valószínűség vs Relatív Gyakoriság

Na jó, már tudjuk akkor használni a normális eloszlás sűrűségfüggvényét. Jó-jó, de **ennek mi értelme?** Oké, akkor az előbb a sűrűségfüggvénnyel kiszámoltuk, hogy a Tesla részvényekkel a 47\$ és 82\$ közti veszteség valószínűsége 3.5%.

```
stats.norm.cdf(x = -47, loc = mű, scale = szigma) - stats.norm.cdf(x = -82, loc = mű, scale = szigma)
```

```
## 0.035316558328262415
```

De ez egy olyan dolog, amit az osztályközös **gyakorisági tábla relatív gyakoriságaiból is tudtunk már, nem?** Hiszen ott megnéztük a  $-82$  és  $-47$  értékek közötti napok számát, mint kedvező esetek, és elosztottuk a teljes  $N = 250$  elemszámmal, mint összes eset. Tehát ilyen elven az is, a 47\$ és 82\$ közti veszteség valószínűsége, nem?

## gyaktábla

##	AlsóHatár	FelsőHatár	f_i	RelatívGyak	KumRelatívGyak	Y_i
## 0	-152.359986	-117.136239	1.0	0.004	0.004	-134.748112
## 1	-117.136239	-81.912491	3.0	0.012	0.016	-99.524365
## 2	-81.912491	-46.688744	5.0	0.020	0.036	-64.300618
## 3	-46.688744	-11.464996	24.0	0.096	0.132	-29.076870
## 4	-11.464996	23.758751	193.0	0.772	0.904	6.146877
## 5	23.758751	58.982498	15.0	0.060	0.964	41.370625
## 6	58.982498	94.206246	7.0	0.028	0.992	76.594372
## 7	94.206246	129.429993	2.0	0.008	1.000	111.818119

Na igen ám, de itt ez a relatív gyakoriság 2.0%!! Na akkor most kinek higgyek? Mi ez a keresett valószínűség? 3.5% ahogy a sűrűségfüggvény mondja vagy 2.0%, ahogy a relatív gyakorisággal kiszámoltam? Mi a kettő válasz közötti különbség?

Nos, azt kell észrevenni, hogy a **relatív gyakoriságos 2.0% kiszámításánál csak a megfigyelt adatokat, azaz a megfigyelst statisztikai MINTÁT vettem csak figyelembe!!** Tehát a 2.0% esetén a **pontos értelmezés az, hogy a megfigyelt napjainknak 2%-a volt olyan, hogy a részvénnnyel 47-82 dollárt veszítettünk!!**

Ezzel szemben a 3.6%, amit az adatokra illeszkedő átlagú és szórású normális eloszlás sűrűségfüggvénye (Gauss görbéje) alapján számoltunk már egy *elvi valószínűség!* Konkréten, **annak az ELVI valószínűsége, hogy a részvénnnyel 47-82 dollár közti összeget veszíték 3.6%!!** Ez azért lehet egy elvi érték, hiszen mivel az  $x$  tengely felett egy folytonos vonallal összekötött  $f(x)$  függvényről beszélünk, ami így **pozitív bekövetkezési valószínűséget rendel olyan  $x$  értékekhez is, amik a megfigyelt adatok között még nem szerepelnek!!** Tehát, **a sűrűségfüggvény megfigyelt adataimon kívüli világot is figyelembe veszi!** Emiatt mondhatom a sűrűségfüggvényből származó értékeket *valódi VALÓSZÍNŰSÉG*nek, és nem csak relatív gyakoriságnak!

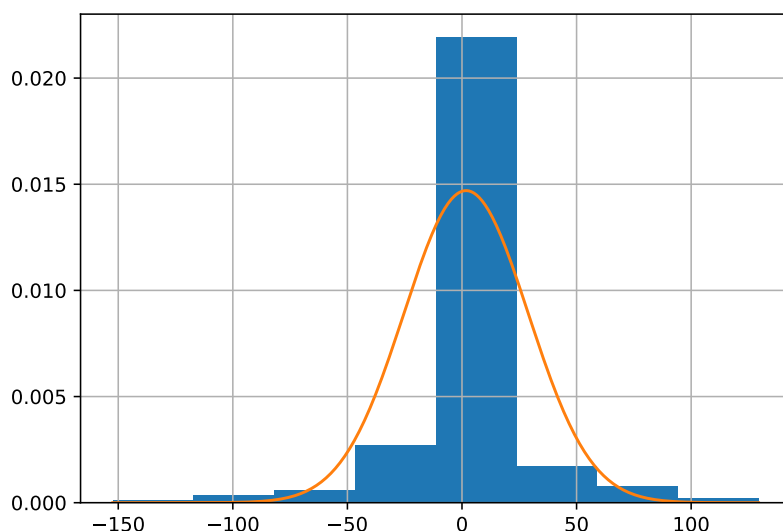
*Nota bene:* ehhez azért az is kell, hogy az eloszlás, aminek a sűrűségfüggvényét használok tényleg illeszkedjen az adatokra! Itt azért most a normális eloszlással lehetnek gondok, hiszen amint láttuk pl. az  $\alpha_4$  a Tesla részvények eloszlása kicsit csúcsosabb az  $N(1.8, 27.19)$  eloszlás sűrűségfüggvényénél.

Azt, hogy egy eloszlás sűrűségfüggvény mennyire illeszkedik a megfigyelt adatok hisztogramjára a következőképpen tudjuk grafikusán megvizsgálni.

- Először elkészítjük a hisztogramot a `hist` metódussal, `density = True` paraméterrel, hogy az  $y$  tengely skálázása összemérhető legyen a sűrűségfüggvény  $y$  tengelyével, ami ugyebár valószínűségeket mutat ki.

- Utána megadjuk egy külön objektumban a sűrűségfüggvény  $x$  tengelyének tartományát a `np.arange` függvénnyel. A függvény paraméterezése azt mondja nekünk el, hogy a létrehozott  $x$  tengely a megfigyelt Tesla árváltozások minimuma és maximuma között fog terjedni 0.01-es lépésközzel.
- Kövi lépésben megadjuk a sűrűségfüggvény  $y$  tengelyét egy külön objektumban a `scipy` csomag `norm.pdf` függvényével. Ha ennek a függvénynek az `x` paraméterében több értéket adunk át, akkor mindegyikhez szépen kiszámolja a sűrűségfüggvény  $f(x)$  helyettesítési értékét. Az átlagot (`loc` paraméter) és szórást (`scale` paraméter) most közvetlenül a data frame-ből számolom ki a függvényen belül.
- Egy sima `matplotlib` csomag `plt` névteréből származó `plot` függvénnyel felrajzolunk egy olyan vonaldiagramot a hisztogramra, aminek az  $x$  és  $y$  tengelye az előző két pontban létrehozott értékekből áll.
- Végül a `plt.show()` paranccsal kikényszerítjük, hogy ezt a többretegű ábrát így egyben mutassa meg nekünk a gépállat.
- **FONTOS!** Az alábbi kódsort mindig egyben futtassuk le, mert csak így fogja szépen egyben összerakni a kívánt ábrát! Ha soronként futtatjuk, akkor két külön ábránk lesz belőle!

```
Tesla.TESLA.hist(bins = 8, density = True)
x_tengely = np.arange(np.min(Tesla.TESLA), np.max(Tesla.TESLA), 0.01)
y_tengely = stats.norm.pdf(x = x_tengely, loc = np.mean(Tesla.TESLA), scale = np.std(Tesla.TESLA))
plt.plot(x_tengely, y_tengely)
plt.show()
```



Egyszerűen *utsukushii*, nemigaz? :) Szépen látszik, hogy a valódi árváltozás eloszlás kissé csúcsosabb, mint amit az adatokra illeszkedő normális eloszlás sűrűségfüggvénye sugall.

Egyébként majd ilyen elvi sűrűségfüggvények illeszkedési jóságát valós hisztogramokhoz egzaktabban is megtanuljuk majd mérni a félév során, mint a szemmelverés. :)

### 3.2.4. Centrális Határeloszlás Tétel (CHT)

A normális eloszlás esetében viszont van egy **valószínűségi számítási tétel**, ami megadja, hogy a normális eloszlás milyen tulajdonságú adatsorok esetén lesz egy jól illeszkedő eloszlás a megfigyelt adatok hisztogramjára. Ez a tétel pedig a **Centrális Határeloszlás Tétel**, leánykori nevén **CHT**. Maga a tétel azt mondja, hogy ha az adatsor egy  $Y_i$  értéke véletlen hatások összegződéséként áll elő, akkor az adatok hisztogramja normális eloszlású sűrűségfüggvényt követ.

A tétel tehát ilyen klasszikus ha  $\rightarrow$  akkor típusú matematikai tétel. És az „akkor” utáni rész az érthetőbb. Ha valami felétel teljesül, akkor az adatsorunk normális eloszlású. Ezt oké, értjük. De mit jelent az a rész, ami a tétel feltételében van? Hogy ha az  $Y_i$  értékek véletlen hatások összegeként állnak elő.

Nos, ez utóbbi rész megértéséhez nézzünk rá ismét a Tesla árváltozások adatsorának első 5 elemére a `data frame head` módszerével.

```
Tesla.head()
```

```
##          Dátum      TESLA
## 0 2019-05-07  -8.279998
## 1 2019-05-08  -2.220002
## 2 2019-05-09  -2.860000
## 3 2019-05-10  -2.459992
## 4 2019-05-13 -12.510009
```

Vegyük például a 2019 május 13-i  $Y_5 = -12.510009$ -os veszteséget. Nos ez az érték úgy jött ki, hogy az adott nap (2019. 05. 13.) véletlenszerű gazdasági eseményeinek hatása összegződött, és így kötöttünk ki ott, hogy a Tesla részvény a nap végére kb. 12 és fél dollárral kevesebbet ér. Tehát, reggel mondjuk bejelentik a kínaiak, hogy vizsgálatot indítanak az egyik Tesla gyár munkakörülményei ellen, aminek hatására elkezd esni a részvény értéke, de aztán délben Musk tweetel egyet, hogy „no para, átviszem a gyárat Mexikóba”, aminek hatására nyugi lesz és elindul felfelé a részvény értéke, de aztán nap végére beesik egy hír, hogy a Mexikóban máris tüntikéznak a tervezett Tesla gyár ellen, ami megint elkezd levinni a részvény értékét és a

nap végére uda jutunk, hogy a részvény  $-12.510009\%$ -al zár... Szóval az **adott nap véletlenszerű gazdasági eseményeinek összegződése**ként áll elő a nap végi  $Y_i$  Tesla árváltozás. Ezt jelenti az, hogy az  $Y_i$  értékek véletlen hatások összegeként állnak elő. És ilyen esetekben az  $Y_i$  adatsorhoz tartozó hisztogram normális eloszlású sűrűségfüggvényt követ a CHT szerint!

Láthatjuk a Tesla részvény vizsgálatának korábbi tapasztalataink alapján, hogy a csúcsosság miatt azért a pénzügyi piacokon ez a tétel nem érvényesül annyira pontosan, de közelítőleg igen. De több egyéb esetben elég szépen érvényesül: Pl. egy termelőgép által a nap végén gyártott selejtes termékek száma esetén. Az adott napi selejtszám értéke (ha nincs szabotőr a gyárban) az adott napi véletlen hatások összegződése állítja elő. Így, ha több nap nap végi selejtszámait vizsgáljuk, akkor azok hisztogramja csudiszép normális eloszlást kell, hogy kirajzoljon.

### 3.2.5. Inverz Értékek

A sűrűségfüggvényektől lehet „visszafelé is kérdezni”. Tehát, nem csak arra képesek, hogy mondok egy eseményt (pl. mi a valószínűsége, hogy 80%-nál nagyobb veszteségem lesz a Teslán) és adnak hozzá valószínűséget, hanem arra is, hogy mondok nekik valószínűséget, és az **inverz értékeik** segítségével adnak hozzá értéket. Szóval, tudok tőlük olyat kérdezni, hogy pl. *Mi az az érték, aminél csak 5% valószínűséggel veszíték többet a Teslán?* Magyarul, a  $P(Y_i < x) = 0.05$  kifejezésben megadja nekem a sűrűségfüggvény inverz értéke az  $x$ -et. Ilyenkor az történik a háttérben, hogy a sűrűségfüggvény primitívfüggvényéből (amit hívnak eloszlásfüggvénynek is, de a mi szempontunkból ez az elnevezés nem fontos) „kifejezzük az  $x$ -et”.

Természetesen, a `scipy`-nak erre is van beépített függvénye `norm.ppf` néven, ami 3 paramétert kíván: a  $P(Y_i < x)$  alá esési valószínűséget (ezt a függvény `q`-nak hívja az angol kvantilis=quantile szóból), a  $\mu$  átlagot (`loc`) és a  $\sigma$  szórását (`scale`). Akkor hát lássuk mi is az az érték, aminél csak 5% valószínűséggel veszíték többet a Teslán?

```
stats.norm.ppf(q = 0.05, loc = np.mean(Tesla.TESLA), scale = np.std(Tesla.TESLA))
```

```
## -42.85439941594264
```

Oké, tehát csak 5% valószínűséggel veszítünk kb. 42.85%-nál többet. Vagy másképp: 5% a valószínűsége, hogy egy random napon a Tesla árváltozás kisebb lesz, mint  $-42.85\%$ . Jó tudni. :) Amúgy pénzügyekben ezeket az értékeket 5%-os *Value at Risk*-nek szokás becézni, mint 5%-os kockáztatott érték. Általában úgy szól a törvényi szabályozás ezeket az értékeket a befektetési bankoknak be kell raknia biztonsági tartalékba egy-egy pénzügyi befektetési portfólió után.

Ha valaki mélyebben belegondol a Stat. I-es rémképeibe, az **eloszlások inverz értékeihez is találhat analógiát, mégpedig a percentiliseket!** Hiszen a megfigyelt árváltozások **5. percentilise** megadja, hogy mi az az érték, aminél az adatok 5%-a kisebb csak. Tehát ez az értelmezés lehet a „*Mi az az érték, aminél csak 5% a valószínűsége, hogy egy random napon a Tesla árváltozás kisebb lesz?*” c. kérdés analógiája.

Lássuk, akkor mi a megfigyelt érváltozások 5. percentilise! Itt most a data frame `quantile` metódusát vetjük be, aminek a paraméterében tizedestörtként kell megadni a keresett percentilis sorszámát. Tehát az 5. percentilisének 0.05-öt adunk meg.

```
Tesla.TESLA.quantile(0.05)
```

```
## -28.48700139999998
```

Ahha, ez csak kb.  $-28.5\%$ ! Tehát a megfigyelt napjaink 5%-ban volt nagyobb veszteségünk, mint  $28.5\%$ ! Ez azért **lényegesen kisebb érték, mint a sűrűségfüggvényből származó  $42.85\%$ -os veszteség!** És valószínűleg a **sűrűségfüggvényből származó érték a realisabb**, hiszen az a számolás során ugyebár **olyan értékeket is figyelembe vett** valami pozitív bekövetkezési valószínűséggel, **amiket a megfigyelt adatsor még egyáltalán „nem is látott”**, mert kisebbek pl. mint a minimum értéke. Tehát, megint elmondhatjuk, hogy **ha az elvi eloszlásból „keresek percentilist”, akkor a megfigyelt adatokon kívüli világot is figyelembe veszem!** Azaz, **általánosítok.**

Az tehát, hogy mondjuk egy befektetési bank a befektetéseinek *Value at Risk* értékét a megfigyelt korábbi adatokból, vagy egy azokra jól illeszkedő elvi eloszlásból számolja egyáltalán nem mindegy! Persze itt a jól illeszkedő eloszlás nem feltétlenül a normális eloszlás, de rengeteg egyéb, kellően egzotikus sűrűségfüggvénnyel rendelkező eloszlás van a palettán, lehet válogatni. :) Persze a válogatáshoz dolgozni is kéne, és nagy a csábítás, hogy egyszerűen inkább a megfigyelt múltbeli adatok alapján mondjon az ember egy percentilist...a nagy befektetési bankok többsége 2008 előtt ezt is csinálta, mert megtehetette. Aztán a 2008-as pénzügyi válság lett belőle. Erről is szól részben a *The Black Swan: The Impact of the Highly Improbable* c. könyv. Ajánlom minden érdeklődőnek, tartalmas és közérthető olvasmány. :) 2008 óta a törvényi szabályozás (Európában a Bázeli III., 2023-tól Bázeli IV.) kötelezi a bankokat, hogy a befektetéseikhez *Value at Risk*-et az adataikra megfelelően illeszkedő elvi eloszlásból számoljanak.

Természetesen ilyen inverz érték formájában „pozitív” dolgot is kérdezhetek: **Mi az az érték, aminél csak 1% a valószínűsége, hogy többet nyerünk egy Tesla részvényen?** Azaz, mi a 99%-os valószínűséggel elérhető legnagyobb nyereség?

Ekkor a kérdés ugyebár úgy szól matematikai formájában, hogy mi az az  $x$ , aminél  $P(Y_i > x) = 0.01$ -et kapunk. De mivel a `scipy` csomag `norm.ppf` függvénye **csak alá esési valószínűséghez tud visszakeresni értékeket**, így inkább a kérdés átfogalmazott verzióját kérdezzük meg a gépszellemtől: **Mi az az  $x$ , aminél  $P(Y_i < x) = 0.99$ -et kapunk?**

```
stats.norm.ppf(q = 0.99, loc = np.mean(Tesla.TESLA), scale = np.std(Tesla.TESLA))
```

```
## 64.91674710828752
```

Tehát, csak 1% eséllyel tudok többet nyerni egy nap a Teslán, mint kb. 65\$.

### 3.2.6. A Standard Normális Eloszlás

Még egy fontos dologról kell megemlékeznünk a normális eloszlás kapcsán, a  $\mu = 0$  átlagú és  $\sigma = 1$  szórású  $N(0,1)$  eloszlásról, ami **standard normális eloszlás** néven külön helyet kapott a pokolban.

Ami miatt külön kiemelt helye van ennek a standard normális eloszlásnak az az, hogy minden  $N(\mu, \sigma)$  normális eloszlás áttranszformálható standard normális  $N(0, 1)$  eloszlássá. Mégpedig a következő formulával.

$$z_i = \frac{Y_i - \mu}{\sigma}$$

Tehát, ha egy **normális eloszlású  $Y_i$  adatsor minden eleméből kivonom az átlagot és az eredményt elosztom a szórással, akkor az így előálló  $z_i$  adatsor már standard normális eloszlású lesz.** Ez a művelet a **standardizálás/noralizálás művelete**.

Amúgy azt, hogy egy adatsor/sokaság valamilyen eloszlást követ, azt  $\sim$  jellel szokás jelölni. Tehát azt mondhatom, hogy  $Y_i \sim N(\mu, \sigma)$ , de  $z_i \sim N(0, 1)$ .

Lássuk is akkor a standardizálást a gyakorlatban a Tesla részvények árváltozásain, és állítsuk elő ezt a  $z_i$  oszlopot.

```
Tesla['z_i'] = (Tesla.TESLA - np.mean(Tesla.TESLA))/np.std(Tesla.TESLA)
round(Tesla.describe(), 2) # 2 tizedesre kerekítés az átláthatóság miatt
```

```
##                                Dátum    TESLA    z_i
## count                        250    250.00    250.00
## mean    2019-11-02 15:56:09.600000    1.78    0.00
## min      2019-05-07 00:00:00 -152.36    -5.68
## 25%      2019-08-05 06:00:00  -3.77    -0.20
## 50%      2019-10-31 12:00:00   1.42    -0.01
```

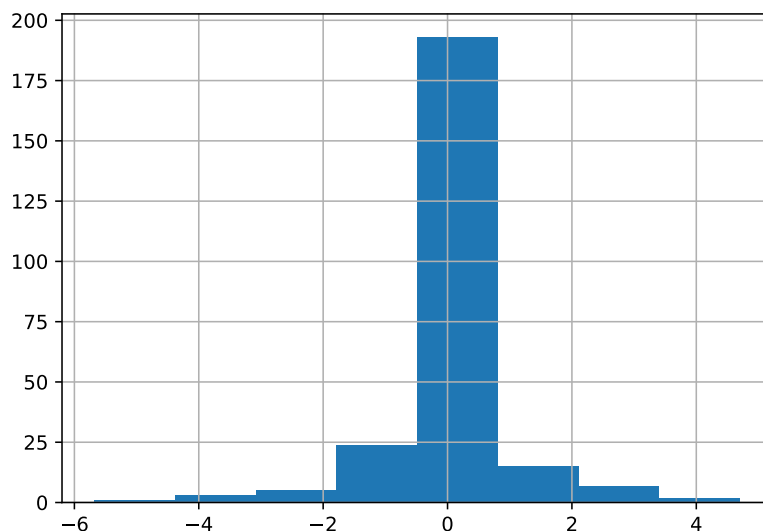
## 1043. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰÉGSZÁMÍTÁS ALAPOK

```
## 75%          2020-02-02 06:00:00    6.96    0.19
## max          2020-05-01 00:00:00   129.43    4.70
## std                               NaN    27.19    1.00
```

Láthatjuk a leíró statisztikákból, hogy a  $z_i$  adatsornak már kb. 0 az átlaga és kb. 1 a szórása 2 tizedesre kerekítve.

De a hisztogram alapján az eloszlás továbbra is normális maradt.

```
Tesla.z_i.hist(bins=8)
```



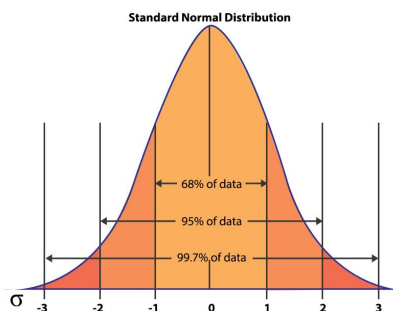
Ami miatt szeretni szokás a standard normális eloszlást az az a jellemzője, hogy

- Az adatok kb. középső 68.2%-a  $-1$  és  $+1$  között
- Az adatok kb. középső 95.4%-a  $-2$  és  $+2$  között
- Az adatok kb. középső 99.7%-a  $-3$  és  $+3$  között

helyezkedik el.

Ezt szemlélteti az alábbi ábra is.





De ezt ellenőrizhetjük is könnyen Pythonban is, pl. a  $\pm 2$ -re. A `norm.cdf` függvény ugyanis `loc=0` és `scale=1` beállításokkal fut, ha nem adunk meg neki mást. Tehát számoljuk ki  $z_i \sim N(0, 1)$  esetén a  $P(-2 < z_i < +2)$  valószínűséget!

```
stats.norm.cdf(2)-stats.norm.cdf(-2)
```

```
## 0.9544997361036416
```

Jé, tényleg kb. 95.4%! :) **Ezt a tulajdonságát majd ki fogjuk a későbbiekben használni a standard normális eloszlásnak, szóval jól jegyezzétek meg! :)**

A fenti tulajdonságok miatt sokan szokták úgy keresni a kilógó értékeket egy adatsorban, hogy standardizálják őket, és megnézik melyek azok az értékek, amik kívül esnek a  $\pm 2$  intervallumon, mondván az ilyen értékek vagy az adatok alsó vagy a felső 2.5%-ba tartoznak (a sűrűségfüggvényből látszik, hogy a 95%-on kívüli 5% egyenletesen oszlik meg a függvény két széle között...szimmetriksu az eloszlás ugyebár :)).

Ezt az elvet mi is könnyen tudjuk alkalmazni:

```
Tesla[(Tesla.z_i < -2) | (Tesla.z_i > 2)]
```

```
##          Dátum          TESLA          z_i
## 185 2020-01-30    59.820008  2.138541
## 187 2020-02-03   129.429993  4.703562
## 188 2020-02-04   107.059998  3.879262
## 189 2020-02-05  -152.359986 -5.679967
## 197 2020-02-18    58.369995  2.085110
## 198 2020-02-19    59.019959  2.109060
## 201 2020-02-24   -67.210022 -2.542321
## 204 2020-02-27  -99.799988 -3.743211
## 206 2020-03-02    75.630005  2.721115
## 211 2020-03-09   -95.479980 -3.584026
## 214 2020-03-12   -73.679992 -2.780729
```

```
## 216 2020-03-16 -101.549988 -3.807696
## 218 2020-03-18 -68.980011 -2.607542
## 219 2020-03-19 66.420014 2.381741
## 222 2020-03-24 70.709991 2.539820
## 235 2020-04-13 77.950012 2.806604
## 236 2020-04-14 58.940003 2.106114
## 241 2020-04-21 -59.640014 -2.263378
## 245 2020-04-27 73.599976 2.646312
## 249 2020-05-01 -80.559998 -3.034247
```

Meg is vannak a kiugróan nagy veszteséget vagy nyereséget szolgáltató napjaink. :)

De **ezzel a módszerrel vigyázzunk!** A standardizált  $z_i$  értékek alapján történő kilógó érték keresés **csak akkor működik, ha az eredeti (standardizálás előtti) adatsorunk is már eleve normális eloszlású volt!** Hiszen csak ekkor lesz a transzformált adatsor is szimmetrikus normális eloszlású és lesz igaz rá a  $P(-2 < z_i < +2) = 0.954$  összefüggés! Szóval **kilógó érték kereséshez inkább használjuk a tetszőleges eloszlásokon is működőképes doboz ábrás módszert!** :)

Még egy utolsó gondolat. A standardizált  $z_i$  értékeknek van egy olyan értelmezése is, hogy megadják, az adott érték a  $\sigma$  szórás hányszorosával tér el a  $\mu$  átlagtól. Tehát, pl. a fenti szűrésben szereplő 2020 január 30-i 59.82\$-os árváltozás a 27.19-es szórás kb. 2.14-szeresével tér el az 1.8\$-os átlagtól.

### 3.3. Az Exponenciális eloszlás

Na, hát akkor most engedjük el egy kicsit a Tesla részvények árváltozásait, és vizsgáljunk meg egy másik adatsort, ami a CancerSurvival.xlsx fájlban lakik. Ebben az adattáblában 58 súlyos fej- és nyakrák páciensről rögzítették, hogy *hány hónapig* maradtak életben kemoterápia után. Az adatok valóságok, 1988-ból a forrás ez a tanulmány.

Töltsük is be az adatokat egy pandas data frame-be!

```
Surv = pd.read_excel("CancerSurvival.xlsx")
```

```
Surv.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 58 entries, 0 to 57
## Data columns (total 2 columns):
## #      Column      Non-Null Count  Dtype
## ---  -
##      Column      Non-Null Count  Dtype
```

```
## 0   Sorszám    58 non-null    float64
## 1   SurvMonth  58 non-null    float64
## dtypes: float64(2)
## memory usage: 1.0 KB
```

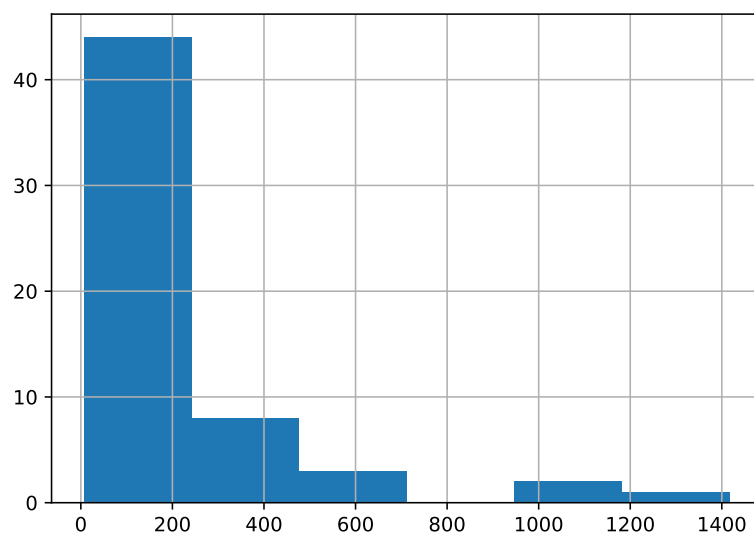
```
Surv.head()
```

```
##   Sorszám  SurvMonth
## 0      1.0      6.53
## 1      2.0      7.00
## 2      3.0     10.42
## 3      4.0     14.48
## 4      5.0     16.10
```

Mint láthatjuk, ebben a data frame-ben is csak két oszlopunk van. Az első a páciens sorszáma, a második pedig a kemoterápiától számított túlélési idő hónapokban megadva (*SurvMonth*).

Nézzünk rá egy hisztogrammal a túlélési idők eloszlására. Mivel most  $N = 58$ , így a legkisebb olyan  $k$ , amire  $2^k$  már épp nagyobb  $N$ -nél az a 6 lesz, hiszen  $2^6 = 64$ . Tehát 6 osztályközt hozunk létre a hisztogramon.

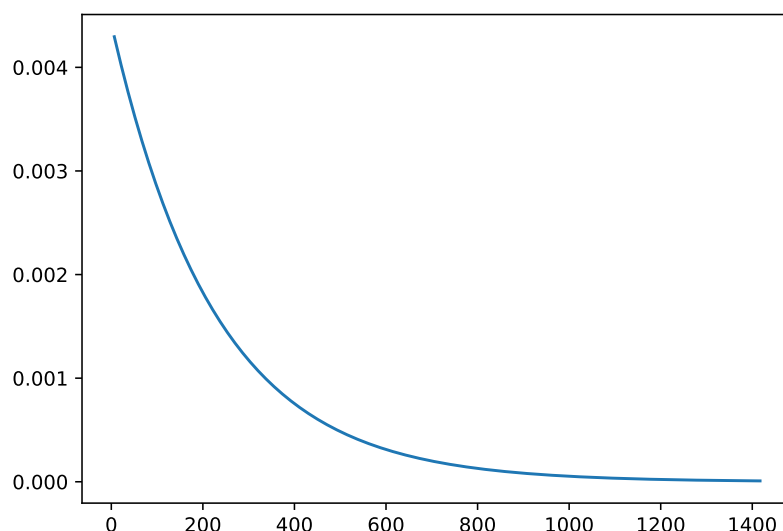
```
Surv.SurvMonth.hist(bins=6)
```



## 1083. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

Nos, hát itt az látszódik, hogy az eloszlásunk jobbra elnyúló: a túlélési idők nagy többsége (45 az 58-ból konkrétan) 256 hónapon belüli, de a maradék 13 meghaladja ezt, sőt 3 páciens 1000 hónapnál is hosszabb ideig élt túl a kemoterápia után.

A jobbra elnyúlás miatt, ha folytonos vonallal összekötjük a hisztogram oszlopait, akkor valami ilyesmi függvényábrát kapunk, mint alább.



Ez az alakzat pedig az **exponenciális eloszlás sűrűségfüggvénye**. Ennek a sűrűségfüggvénynek a konkrét alakját egy  $\lambda$  paraméter határozza meg. Minél nagyobb  $\lambda$ , annál meredekebben jobbra elnyúló a sűrűségfüggvény. Ezt alább lehet kipróbálni.

Természetesen a  $\lambda$ -nak van köze a valós adatok átlagához és szórásához, egész konkrétan mindkét érték  $\mu = \sigma = \frac{1}{\lambda}$ . Tehát, az exponenciális eloszlásban azonos átlagot és szórást tételezünk fel az adatokra, és ennek a közös értéknek a reciproka (a  $\lambda$ ) határozza meg, hogy mennyire meredeken nyúlik jobbra az eloszlás sűrűségfüggvénye. Emiatt az exponenciális eloszlásokat  $Exp(\lambda)$  módon szokták jelölni.

Persze valós adatokon gyakorlatilag sosem fog teljesülni, hogy  $\mu = \sigma$ , de láthatjuk egy **describe** metódusból, hogy a túlélési adatok esetén a két mutató értéke aránylag közel esik egymáshoz:  $\mu = 226.17 \approx \sigma = 273.94$

```
## count      58.000000
## mean       226.173793
## std        273.943381
```

```
## min          6.530000
## 25%          83.250000
## 50%         151.500000
## 75%         237.000000
## max         1417.000000
## Name: SurvMonth, dtype: float64
```

A `scipy` csomagban a `norm.pdf`, `norm.cdf` és `norm.ppf` függvények mintájára léteznek `expon.pdf`, `expon.cdf` és `expon.ppf` függvények is. Használatuk és jelentésük teljesen megegyezik a normális eloszlásnál látott függvényekkel. Egyetlen különbség ugyebár, hogy exponenciális eloszlásnál csak az egységes  $\lambda$ -t kell megadni a külön  $\mu$  és  $\sigma$  helyett, mint ahogy a normális eloszlásnál működött a dolog. A `scipy` csomag ezt úgy oldja meg, hogy a szórásból számolja vissza a  $\lambda$ -t, tehát a függvényeknek a `scale` paraméterében kell átadni az adatok szórását, amire az exponenciális eloszlást illeszteni akarjuk. Ez alapján akkor most a túlélési idők esetében  $\lambda = \frac{1}{\sigma} = \frac{1}{273.94} = 0.00365$ . Tehát az egyes  $Y_i$  túlélési idők  $Exp(0.00365)$  eloszlástkövetnek:  $Y_i \sim Exp(0.00365)$

Ezek alapján számoljunk ki pár valószínűséget a túlélési időkre vonatkozóan:

1. Mi a valószínűsége, hogy kemoterápia után pont egy évet, azaz 12 hónapot fogunk élni?

```
stats.expon.pdf(12, scale = np.std(Surv.SurvMonth))
```

```
## 0.003523104116060953
```

Ez egy jó alacsony, kb. 0.3%-os valószínűség. Nem lepődünk meg, hiszen egy konkrét pont bekövetkezése a nagy túlélési idő-értékkészlet miatt itt is kicsi.

2. Mi a valószínűsége, hogy kemoterápia után több, mint öt évet, azaz 60 hónapot fogunk élni?

```
1 - stats.expon.cdf(60, scale = np.std(Surv.SurvMonth))
```

```
## 0.8017677791228195
```

Az eredmény kb. 80%, egész jó kilátások!

3. Mi a valószínűsége, hogy a kemoterápia utáni harmadik év során, azaz 24 és 36 hónap között fogunk elpatkolni?

```
stats.expon.cdf(36, scale = np.std(Surv.SurvMonth)) - stats.expon.cdf(24, scale = np.s
```

```
## 0.03956914214644276
```

A számítások alapja itt is az, hogy  $f(x) = P(Y_i = x)$ , tehát **a sűrűségfüggvény helyettesítési értékre  $x$  helyen megegyezik az  $x$  érték bekövetkezési valószínűségével** egy véletlen húzás esetén az adatsorból. A  $P(Y_i < x)$  valószínűség pedig exponenciális sűrűségfüggvény esetén is az  $\int_{-\infty}^x f(x)dx$  improprius integrállal számítható, azaz a sűrűségfüggvény  $x$  alatti területével egyezik meg.

Ezeket a vizuális jelentéstartalmakat az alábbi interaktív ábrán meg lehet nézni és ki lehet próbálni úgy, ahogy a normális eloszlásnál lehetett.

Természetesen *inverz értéket* is tudunk számolni az exponenciális eloszlásban is. Nézzük meg pl, hogy Mi az az idő, aminél csak 1% a valószínűsége, hogy egy kemoterápiával kezelt fej- és nyakrák páciens tovább él. Ugyebár a számításhoz úgy kell átfogalmazni a kérdést, hogy mi az az idő, ami esetén csak 99% a valószínűsége, hogy egy kemoterápiával kezelt fej- és nyakrák páciens már *NEM* él tovább. Hiszen az `expon.ppf` függvény is *alá esési* valószínűségekből dolgozik, mint a `norm.ppf`.

```
stats.expon.ppf(0.99, scale = np.std(Surv.SurvMonth))
```

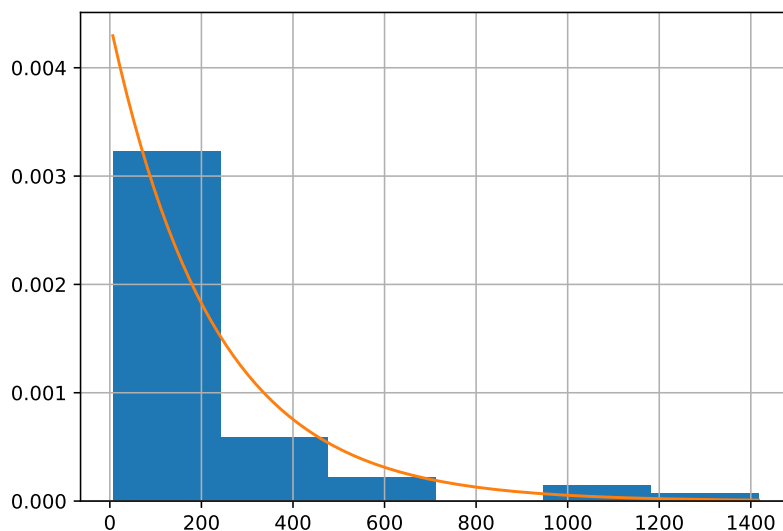
```
## 1250.6331218835987
```

A megfejtés kb. 1250 hónap, azaz 104 év! De hát ugye ez a nagy érték alapvetően a jobbra elnyúlás miatt van, hiszen a jobbra elnyúló eloszlásokra jellemzőek a felfelé kilógó értékek, így a jobbra elnyúló sűrűségfüggvényeknek is számolnia kell ezekkel az outlier elemekkel.

Végül pedig nézzük meg szépen, hogy ez az exponenciális sűrűségfüggvény mennyire illeszkedik a túlélési idők hisztogramjára, ahogy a normális eloszlás esetén is megtettük egy hisztogramra illesztett `matplotlib`-es vonaldiagrammal.

```
Surv.SurvMonth.hist(bins = 6, density = True)
x_tengely = np.arange(np.min(Surv.SurvMonth), np.max(Surv.SurvMonth), 0.01)
y_tengely = stats.expon.pdf(x_tengely, scale = np.mean(Surv.SurvMonth))
plt.plot(x_tengely, y_tengely)
plt.show()
```

### 3.4. A VARIANCIAHÁNYADOS PYTHONBAN - KOKAIN A BALATONBAN<sup>111</sup>



Itt egész pofásnak tűnik az illeszkedés, így szemmelverésre jobban is illeszkedik ez az exponenciális eloszlás a túlélési időkre, mint a normális eloszlás illeszkedett a Tesla árváltozásokra. :)

### 3.4. A Varianciahányados Pythonban - Kokain a Balatonban

Egy dolgot kellene még átismételnünk a Stat. I-es rémképeink közül, ami többször is elő fog jönni a Stat. II-es tanulmányinkban: a **Varianciahányados** fogalmát.

A varianciahányados ugyebár arra szolgál, hogy **két ismérv, egy minőségi („szöveges”) és egy mennyiségi („számértékű”) ismérv kapcsolatának szorosságát adja meg, százalékos formában**. Tehát olyan kérdéseket lehet vele megválaszolni, mint...

- Hány százalékban befolyásolja a nem (minőségi ismérv) a fizetést (mennyiségi ismérv)?
- Hány százalékban befolyásolja a kerület (minőségi ismérv) a budapesti lakások árát (mennyiségi ismérv)?
- Hány százalékban befolyásolja a Balaton Sound jelenléte (minőségi ismérv) a Balatonban található kokain mennyiségét (mennyiségi ismérv)?

A legutolsó kérdés a listán elsöre meredeknek tűnik, de ez a tanulmány épp egy ilyen kérdésekkel is foglalkozik. Az általuk használt adatok egy részét találjuk a BalatonSoundCocaine.xlsx című fájlban.

A fájlban **3 balatoni vízminőséget ellenőrző állomás összesen  $N = 540$  mérését látjuk**. Egy állomás egy hónapban 20 mérést végez, és mindhárom állomás esetén a nyári hónapok (június, július, augusztus) mérései vannak a fájlban 3 évre (2017, 2018, 2019). Így egy állomás esetében  $20 \times 3 \times 3 = 180$  mérésünk van, azaz a három állomásra összesen  $3 \times 180 = 540$  mérési adatunk van. Az **Excel fájlunk mindegyik mérés esetében tartalmazza a mérőállomás sorszámát (1., 2., 3.), a mérés évét és hónapját, valamint a vízben mért kokain mennyiségét nanogram/literben**.

Olvassuk is be az Excelt egy data frame-be és lessük meg, hogy ez tényleg így van-e!

```
Balcsi = pd.read_excel("BalatonSoundCocaine.xlsx")
```

```
Balcsi.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 540 entries, 0 to 539
## Data columns (total 4 columns):
## #   Column   Non-Null Count  Dtype
## ---  ---
## 0    Ev       540 non-null    int64
## 1    Honap    540 non-null    object
## 2    Allomas  540 non-null    object
## 3    Kokain   540 non-null    float64
## dtypes: float64(1), int64(1), object(2)
## memory usage: 17.0+ KB
```

```
Balcsi.head()
```

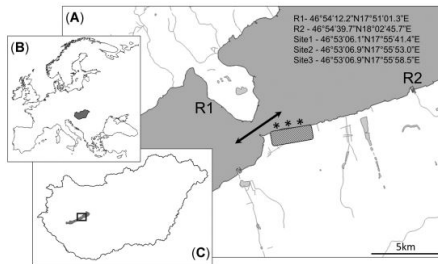
```
##      Ev   Honap   Allomas  Kokain
## 0  2017  június  1. állomás  0.03891
## 1  2017  június  1. állomás  0.01879
## 2  2017  június  1. állomás  0.03193
## 3  2017  június  1. állomás  0.03510
## 4  2017  június  1. állomás  0.01107
```

Igen, az oszlopok (ismérvek) neve és adattípusa és az első öt sor tartalma alapján úgy néz ki, hogy rendben van a tábla, azok az oszlopok szerepelnek benne, amit a leírás alapján vártunk is.



### 3.4. A VARIANCIAHÁNYADOS PYTHONBAN - KOKAIN A BALATONBAN<sup>113</sup>

Oké, akkor itt mérési adatokat látunk. Hogy a túróba jön az egészhez a Balaton Sound. Egyrészt úgy, hogy a három mérőállomás épp a Sound helyszíne környékén található Siófokon. Konkrét koordináták az alábbi ábrán.



És hát a mérések pont a fesztivál előtt (június), közben (július) és után (augusztus) készültek. Tehát, ha **a Sound jelenlétének van hatása a víz kokain tartalmára, akkor a mérés hónapja aránylag nagy százalékban kell, hogy meghatározza a kokaintartalmat.** Szóval a vizsgált **minőségi ismervünk a mérés hónapja, mennyiségi ismervünk a kokaintartalom** lesz. A két ismerv kapcsolatának szorosságát pedig akkor a varianciahányados adja meg.

A varianciahányados kiszámításának első lépése egy olyan segédtáblázat összeállítása, amely sorait a minőségi ismerv lehetséges értékei adják, és 3 oszlopa van, ami a minőségi ismerv  $j$  indexszel jelölt csoportjai szerint bontva tartalmazza az elemszámokat ( $N_j$ ), a mennyiségi ismerv részatlagaikat ( $\bar{Y}_j$ ) és szórásait ( $\sigma_j$ ). Ezt a segédtáblát Pythonban a data frame-k `groupby` és `agg` módszereivel hozhatjuk létre. Persze az `agg`-n belül használjuk a `numpy` csomag `mean` és `std` függvényeit is a `count` mellett (amely utóbbi függvényt stringként kell beadni az `agg`-ba, mint az oszlopneveket). Illetve, ne felejtsük el a végén a `'reset_index'`, módszer használatát, különben a minőségi ismervünk értékei a data frame sorindexeiként végzik, és nem lesz külön oszlopuk!

```
Segéd = Balcsi.groupby('Honap').agg(  
    Elemszam = ('Kokain', 'count'),  
    Reszatlagok = ('Kokain', np.mean),  
    Szorasok = ('Kokain', np.std)  
) .reset_index()
```

```
## <string>:1: FutureWarning: The provided callable <function mean at 0x00000167D93F13A0> is current  
## <string>:1: FutureWarning: The provided callable <function std at 0x00000167D93F14E0> is current
```

Segéd

```
##          Honap  Elemszam  Reszatlagok  Szorasok  
## 0  augusztus        180      0.029198  0.011618
```

## 1	július	180	64.859463	95.853905
## 2	június	180	0.029752	0.011614

Meg is vagyunk! Látszik, hogy **a Sound hónapjának van hatása**: júliusban nagyságrendekkel több az átlagos kokain mennyisége a Balaton vizének, mint a másik két nyári hónapban. De a **hatás nagyságát nehéz megfogni már szemmelvezéssel**, mivel a **kokain mennyiségek szórása is ebben a hónapban a legnagyobb**. Sőt, ez az egyetlen hónap, amikor a konkrét mérések kokain mennyiségének szórása *nagyobb* az átlagos kokain mennyiségnél! Szóval, kell azért egy check arra a varianciahányadosra.

A varianciahányados, a  $H^2$  mutató értékéhez úgy jutunk el, hogy **elkezdjük az előbb felépített segédtablázatunk alapján kiszámolni a mennyiségi ismérvt (azaz most a kokain mennyiségt) teljes, hónapoktól független teljes átlagát és teljes szórását**. Ugyebár a **teljes átlagos kokainmennyiség (főátlag,  $\bar{Y}$ )**, nem más, mint a **részátlagok ( $\bar{Y}_j$ ) részelemszámokkal ( $N_j$ ) súlyozott átlaga**:

$$\bar{Y} = \frac{\sum_j N_j \bar{Y}_j}{\sum_j N_j}$$

Ilyen stílusú súlyozott átlagokat számolgattunk már az 1.4. fejezetben, csak gyakorisági táblából. Ez ugyan az a szitu, és itt is a `np.sum` függvényt be tudjuk vetni. Ellenőrzéshez ki tudjuk számolni ezt a főátlagot úgy is, hogy az `np.mean` függvényt ráeresszük a data frame teljes Kokain oszlopára.

```
főátlag = np.sum(Segéd.Elemszam * Segéd.Reszatlagok) / (np.sum(Segéd.Elemszam))
főátlag
```

```
## 21.63947090740741
```

```
np.mean(Balcsi.Kokain)
```

```
## 21.639470907407407
```

Stimm egészen az utolsó jó sokadik tizedesjegyig. Ezzel megvagyunk. :)

A variancia-hányados lelke viszont abban leledzik, hogy a kokainmennyiség (mint mennyiségi ismérvt) teljes szórása ( $\sigma$ ) csak úgy kapható meg a segédábla alapján, ha előbb kiszámoljuk a belső szórást ( $\sigma_B$ ) és a külső szórást ( $\sigma_K$ ).

A belső szórást a **belső variancián, azaz belső szórásnégyzeten keresztül kapjuk meg**. Ez pedig nem más, mint a **részszórások  $\sigma_j^2$  négyzeteinek elemszámokkal ( $N_j$ ) súlyozott átlaga**.

$$\sigma_B^2 = \frac{\sum_j N_j \sigma_j^2}{\sum_j N_j}$$

### 3.4. A VARIANCIAHÁNYADOS PYTHONBAN - KOKAIN A BALATONBAN<sup>115</sup>

Az előző számítás alapján ez is egész könnyen tud menni Pythonban, csak a részsórások négyzetre emelésére kell figyelni.

```
belső_var = np.sum(Segéd.Elemszam * Segéd.Szorasok**2)/(np.sum(Segéd.Elemszam))
belső_var
```

```
## 3062.6571029636407
```

A belső szórás pedig egyszerűen a belső variancia gyöke.  $\sigma_B = \sqrt{\sigma_B^2}$   
Általánosságban  $\sigma_B$  azt jelenti, hogy **egy véletlenszerűen kiválasztott egyed konkrét mennyiségi ismerv értéke várhatóan mennyivel tér el saját csoportjának átlagától.**

Konkrét esetünkben ez az alábbi módon néz ki.

```
belső_szórás = np.sqrt(belső_var)
belső_szórás
```

```
## 55.34127847243539
```

Tehát, **egy mérés kokainmennyisége várhatóan 55.3 nanogram/literrel tér el saját hónapjának átlagos kokainmennyiségtől.** Ami azért nem egy elhanyagolható mennyiségű szóródás a mérési hónapokon *belül*.

A másik vége a dolognak a külső szórás, ami szintén a négyzetén, a külső variancián keresztül számítható. A külső variancia pedig a **részátlagok  $\bar{Y}_j$  elemszámokkal ( $N_j$ ) súlyozott szórása a mennyiségi ismerv főátlaga  $\bar{Y}$  körül.**

$$\sigma_K^2 = \frac{\sum_j N_j (\bar{Y}_j - \bar{Y})^2}{\sum_j N_j}$$

Az 1.4. fejezet alapján ezt is meg tudjuk azért alkotni `np.sum` bevetésével.

```
külső_var = np.sum(Segéd.Elemszam * (Segéd.Reszatlagok - főátlag)**2)/(np.sum(Segéd.Elemszam))
külső_var
```

```
## 933.983870298589
```

A külső szórás pedig egyszerűen a külső variancia gyöke.  $\sigma_K = \sqrt{\sigma_K^2}$   
Általánosságban  $\sigma_K$  azt jelenti, hogy **egy csoport átlaga várhatóan mennyivel tér el a mennyiségi ismerv főátlagától.**

Konkrét esetünkben ez az alábbi módon néz ki.

```
külső_szórás = np.sqrt(külső_var)
külső_szórás
```

```
## 30.561149688756622
```

Tehát, egy hónap átlagos kokainmennyisége várhatóan 30.5 nanogram/literrel tér el az átlagosan mért kokainmennyiségtől. Tehát a hónapok között is van egy jelentős szóródásunk, viszont ez kicsit kisebb, mint a csoporton belüli szóródás.

Ebből a két tényezőből pedig összeadható a **teljes variancia**:  $\sigma^2 = \sigma_B^2 + \sigma_K^2$ . A teljes szórás pedig ennek a mennyiségnek a gyöke:  $\sigma = \sqrt{\sigma^2} = \sqrt{\sigma_B^2 + \sigma_K^2}$ . Mivel tagonként nem vonhatunk gyököt, így ez az összefüggés ugyebár a szórásokra NEM lesz igaz!!

A teljes szórás pedig általánosan ugye azt mutatja meg, hogy egy véletlenszerűen kiválasztott egyed konkrét mennyiségi ismérték értéke várhatóan mennyivel tér el a mennyiségi ismérték csoportoktól független, teljes főátlagától.

Lássuk, hogy ez a mi esetünkben hogy fest! Ellenőrzésnek számoljuk ki a teljes szórást úgy is, hogy az `np.std` függvényt ráeresztjük a data frame teljes Kokain oszlopára.

```
teljes_var = belső_var + külső_var
teljes_szórás = np.sqrt(teljes_var)
teljes_szórás
```

```
## 63.218992187966975
```

```
np.std(Balcsi.Kokain)
```

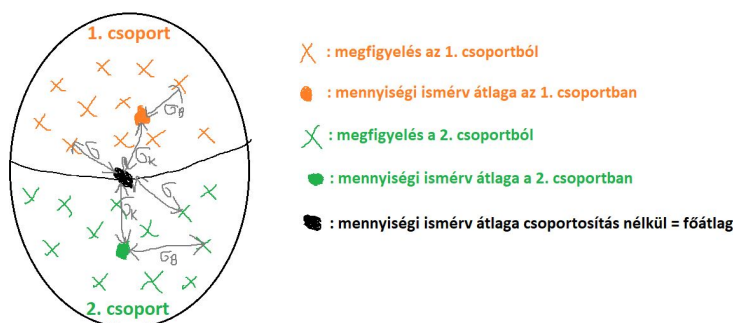
```
## 63.08427864039577
```

Hát ez csak majdnem stimmel. Ennek az oka az, hogy a Segéd data frame-ben a részátlagok és részszerzők 6 tizedesjegyre le lettek kerekítve. De nagyságrendileg stimmelünk! :)

Mindez pedig azt jelenti, hogy egy mérés kokainmennyisége várhatóan 63 nanogram/literrel tér el az átlagosan mért kokainmennyiségtől.

A varianciarányados logikája úgy bukik ki ebből a  $\sigma^2 = \sigma_B^2 + \sigma_K^2$  felbontásból, hogy elképzeljük ezeket a különböző szórásokat vizuálisan, mint távolságokat. Az alábbi ábra egy egyszerűbb rendszert mutat, ahol a minőségi ismérték csak két csoportot alkot (narancsok és zöldek), nem pedig hármat, mint amennyit a mi három hónapunk generál.

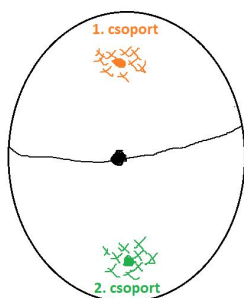
### 3.4. A VARIANCIAHÁNYADOS PYTHONBAN - KOKAIN A BALATONBAN<sup>117</sup>



Tehát vizuálisan a következőképp érdemes gondolni a különböző  $\sigma$ -kra:

- $\sigma_B$ : Megfigyelések távolsága saját csoportjuk átlagától
- $\sigma_K$ : Csoportátlag távolsága a főátlagtól
- $\sigma$ : Megfigyelések távolsága a főátlagától

Ezek alapján nekünk az a jó a csoportosítás, azaz a minőségi ismerv magyarázóereje szempontjából, ha **fix**  $\sigma$  mellett  $\sigma_K$  **nagy** és  $\sigma_B$  **kicsi**. Mert ekkor a **csoportátlagok messze vannak** a főátlagtól és így **impliciten egymástól** is, míg a csoportátlagtól az egyes **megfigyelések nagyon kis mértékben térnek csak el saját csoportátlaguktól**:



Ebben az esetben, ahogy az ábráról is látszik a csoportosításunk, azaz a minőségi ismervünk magyarázóereje nagy! Tehát az kell nekünk, hogy az  $\sigma$  minél nagyobb részét tegye ki  $\sigma_K$ . Viszont, mivel csak a teljes **varianciára** igaz az, hogy **egyenlő a külső és belső VARIANCIA összegével**, így azt mondjuk, hogy **azt szeretnénk látni, hogy a  $\frac{\sigma_K^2}{\sigma^2}$  hányados nagy legyen!** Ez a mutató lesz tehát a **varianciahányados**, és a most elvégzett módszer neve a **variancia-analízis**, azaz **ANOVA = ANalysis Of VAriances**.

**Azért a varianciákra néztük végül a dolgokat**, mert  $\sigma^2 = \sigma_B^2 + \sigma_K^2$ , így a  $\frac{\sigma_K^2}{\sigma^2}$  variancia-hányados biztos, hogy 0 – 1 közötti, és **százalékosan** is értelmezhető, hiszen  $\sigma_K^2$  része  $\sigma^2$ -nek. Ez alapján nekünk most:

$\frac{\sigma_K^2}{\sigma^2} = \frac{933.98387}{3996.64097} = 0.23369 = 23.369\% \rightarrow$  **A hónap (tehát a Sound jelenléte) a Balatonban mért kokainmennyiség alakulásának (varianciájának) kb. 23%-át magyarázza a megfigyelt adatok körében!** Ez egy **közepes magyarázóerő**nek tekinthető, mivel a variancia-hányados a következőképpen „*korszakoljuk*”:

- variancia-hányados  $< 10\%$   $\rightarrow$  gyenge kapcsolat
- $10\% \leq$  variancia-hányados  $\leq 50\%$   $\rightarrow$  közepes kapcsolat
- variancia-hányados  $> 50\%$   $\rightarrow$  erős/szoros kapcsolat

### 3.4.1. További minőségi ismérvek és a kokainmennyiség

Az eredmény tehát azt mondja, hogy a kokainmennyiség alakulásának csak kb. 23%-át tudjuk csak lefedni azzal, hogy a mérés melyik hónapban készült. Tehát hónapokon *belül* is jelentős mértékű szóródás maradt a mennyiségi ismérvünkben, jelesül a kokainmennyiségben. Mi okozhatja még a kokainmennyiség szóródását? Hát, az elérhető adatok tekintetében két dolgot tudunk még megvizsgálni: azt, hogy **melyik mérőállomáson történt a mérés**, illetve azt, hogy **melyik évben**. Reméljük, hogy *inkább az év magyarázza még relatíve nagyobb mértékben a kokainmennyiség alakulását* (pl. emelkedő trend tapasztalható a Sound népszerűségének növekedésével), mert *ha a kokainmennyiség szóródása inkább a mérőállomástól függ, az aggasztó lenne a mérés megbízhatóságára nézve*.

Mivel mind a mérőállomás azonosítója, mind az évszám jelen szituációban minőségi ismérvként kezelhető, így e két ismérvnek a kokainmennyiséggel, mint mennyiségi ismérvvel vett varianciahányadosát kell megvizsgálnunk.

Először nézzük a **mérőállomások** esetét.

Itt is kell ugyebár egy kiinduló segédtablázat.

```
AllomasTabla = Balcsi.groupby('Allomas').agg(
    Elemszam = ('Kokain', 'count'),
    Reszatlagok = ('Kokain', np.mean),
    Szorasok = ('Kokain', np.std)
).reset_index()
```

```
## <string>:1: FutureWarning: The provided callable <function mean at 0x00000167D93F13...
## <string>:1: FutureWarning: The provided callable <function std at 0x00000167D93F14E...
```

```
AllomasTabla
```

##	Allomas	Elemszam	Reszatlagok	Szorasok
## 0	1. állomás	180	30.101843	84.544224
## 1	2. állomás	180	22.314141	61.220808
## 2	3. állomás	180	12.502428	30.877849

### 3.4. A VARIANCIAHÁNYADOS PYTHONBAN - KOKAIN A BALATONBAN<sup>119</sup>

Olybá tűnik, hogy az 1. állomás átlagban némileg kicsit magasabb kokainmennyiséget mér, mint a többi, de az állomás méréseinek szórása is magas, az átlagos kokainmennyiség kb.  $\frac{84.5}{30.1} = 2.8$ -szorosa. Szóval, vágjunk rendet a variancia-hányados segítségével, és lássuk mekkora hatást gyakorol a mérőállomás a kokainmennyiségre!

A számításokban annyi **egyszerűsítést** teszek, hogy

- A **teljes varianciát örökítem** az előző számolásokból, hiszen az a csoportosítás alapját képező minőségi ismerv megváltozásával **nem változik**.
- Az előző pont és a belső variancia kiszámolása után pedig a varianciahányadost a  $\sigma^2 = \sigma_B^2 + \sigma_K^2$  összefüggés átrendezésével  $H^2 = \frac{\sigma^2 - \sigma_B^2}{\sigma^2} = 1 - \frac{\sigma_B^2}{\sigma^2}$  módon számolom ki

Ezzel a két módosítással **megúszom a külső szórásnégyzet** kissé macerás **kiszámítását**.

```
belső_var_Állomás = np.sum(AllomasTabla.Elemszam * AllomasTabla.Szorasok**2)/np.sum(AllomasTabla.Elemszam)

VarHányadÁllomás = 1 - belső_var_Állomás / teljes_var
VarHányadÁllomás
```

```
## 0.01174053573946432
```

Az eredmény mindössze 1.17%. Tehát megnyugodhatunk, az **állomás a mért kokainmennyiség alakulását csak alig több, mint 1%-ban határozza meg!** A mérés **nem függ lényegében az állomástól**, így ilyen szempontból megbízhatónak tekinthetők!

Lássuk az **évszámokat**! Most is kezdjük a segédtáblával.

```
EvTabla = Balcsi.groupby('Ev').agg(
    Elemszam = ('Kokain', 'count'),
    Reszatlagok = ('Kokain', np.mean),
    Szorasok = ('Kokain', np.std)
).reset_index()
```

```
## <string>:1: FutureWarning: The provided callable <function mean at 0x00000167D93F13A0> is current
## <string>:1: FutureWarning: The provided callable <function std at 0x00000167D93F14E0> is current
```

```
EvTabla
```

## 1203. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

##	Ev	Elemszam	Reszatlagok	Szorasok
## 0	2017	180	0.536273	0.734331
## 1	2018	180	1.964270	3.933046
## 2	2019	180	62.417870	97.366788

Itt azért elég látványos eltéréseket találunk! A 2019-es évben hatalmasat ugrott a Balatonban mérhető kokainmennyiség. A szórása is magas, de pl. relatíve nézve 2018-ban magasabb volt: 2018-ban a szórás durván kétszerese volt az átlagnak  $\frac{3.93}{1.96}$ , míg 2019-ben csak durván másfélszerese  $\frac{97.37}{62.42} = 1.56$ . Szóval itt lehet még komolyabb magyarázóerő! De ne találgassunk, hanem lássuk a varianciahányadost! A számolásnál megint alkalmazzuk az állomásoknál bevezetett két **egyszerűsítést**!

```
belső_var_Év = np.sum(EvTabla.Elemszam * EvTabla.Szorasok**2)/np.sum(EvTabla.Elemszam)

VarHányadÉv = 1 - belső_var_Év / teljes_var
VarHányadÉv
```

```
## 0.20797660221119585
```

Na, itt is egy közepes magyarázóerőnk van, kb. 21%.

Tehát alapvetően a Balatonban mérhető kokainmennyiséget két *közepes magyarázóerejű* dolog mozgatja nyáron:

- **Hónap:** A Balaton Sound hónapjában (július) átlagban valamivel több a kokain
- **Év:** 2019-ben sokkal több az átlagos kokainmennyiség, feltehetően a Sound megnövekedett haza és külföldi népszerűsége miatt

Szerencsére a mérőállomás maga nem befolyásolja érdemben a mért kokain szóródását, így a mérések ilyen szempontból megbízhatóan tekinthetők!