

**Következtető Statisztika
Python Jegyzet
2025**

Következtető Statisztika

Python Jegyzet

2025

Budapesti Corvinus Egyetem

Budapest, 2025

Következtető Statisztika Python Jegyzet 2025

Szerző: Kovács László

Tördelés, grafikai kivitelezés: Kovács László

ISBN 978-963-503-974-6

Kiadó: Budapesti Corvinus Egyetem

Budapest, 2025

Következtető Statisztika Python Jegyzet

Kovács László

2025-06-30

Tartalomjegyzék

1. Előhang	7
2. Statisztikához szükséges Python nyelvi alapok	11
2.1. Programozási alapelvek	11
2.2. A Pythonról általában	12
2.3. A Spyder felülete	13
2.4. Working Directory	15
2.5. Alapvető Python adattípusok és adatszerkezetek	15
2.6. Vezérlési szerkezetek	31
2.7. A Pandas data frame objektum	35
2.8. Aggregálás data frame-ben	52
2.9. Egyszerű leíró statisztika data frame-ben	56
2.10. Adatminőségi problémák felismerése és kezelése leíró statisztika segítségével	59
2.11. Data frame-k összekapcsolása	63
2.12. Kilogó értékek keresése és kezelése	68
2.13. Korrelációs elemzések data frame-ben	73
Gyakorló feladatok	80
Gyakorló feladatok megoldása	80
3. Leíró Statisztika ismétlés és Valószínűségszámítás alapok	87
3.1. Leíró statisztikai mutatók	87
3.2. A normális eloszlás és sűrűségfüggvénye	96

3.3. Az Exponenciális eloszlás	110
3.4. A Varianciahányados Pythonban - Kokain a Balatonban	115
4. Eloszlások és Mintavételezés	125
4.1. Véletlenszám generálás és Mintavételezés	125
4.2. Elvi Eloszlás vs Megfigyelt Minta	138
4.3. Statisztikai sokaságok FAE mintavételezése	146
5. A becsléselmélet alapjai	155
5.1. Ismétlés: Balaton átúszás eredmények és ezek FAE mintái	155
5.2. A torzítatlanság fogalma	159
5.3. A korrigált mintavariancia	164
5.4. A medián torzítatlansága	168
5.5. A Standard Hiba (<i>SH</i>) fogalma	169
5.6. Az átlagos négyzetes hiba (<i>MSE</i>) fogalma	175
6. A sokasági átlag intervallumbecslése	179
6.1. Ismétlés: Az átlag standard hibája a Balaton átúszás eredményeken	179
6.2. A mintaátlagok eloszlása	181
6.3. A sokasági átlag konfidencia-intervalluma	185
6.4. Intervallumbecslés a gyakorlatban	192
6.5. Összefoglalás az átlag konfidencia-intervallumairól	203
6.6. Esettanulmány	203
7. További FAE becslések és a Bootstrap módszer	209
7.1. A Konfidencia-intervallumok két általános tulajdonsága	209
7.2. Arányok konfidencia-intervalluma	210
7.3. A Bootstrap becslések általános elve	221
7.4. A Medián Bootstrap intervallumbecslése	226
7.5. A Szórás Bootstrap intervallumbecslése	230

8. Becslések EV és R mintából	235
8.1. Intervallumbecslés visszatevés nélküli egyszerű véletlen (EV) mintákból	235
8.2. Átlag becslése Arányosan Rétegzett (AR) mintákból	240
9. Hipotézisvizsgálat alapjai	247
9.1. A hipotézisvizsgálat alapgondolata	247
9.2. A sokasági átlagra vonatkozó <i>t-próba</i>	253
9.3. A sokasági átlagra vonatkozó <i>z-próba</i>	262
9.4. A sokasági arányra vonatkozó <i>z-próba</i>	264
9.5. A hipotézisvizsgálat menete madártávlatból	266
10. Egymintás és Kétmintás próbák	269
10.1. A szórásra vonatkozó χ^2 -próba	269
10.2. A kétmintás hipotézisvizsgálatok elve	273
10.3. Az átlagokra vonatkozó kétmintás <i>z-próba</i>	275
10.4. Az arányokra vonatkozó kétmintás <i>z-próba</i>	278
11. Nemparaméteres próbák	283
11.1. Nemparaméteres próbák elve	283
11.2. Illeszkedésvizsgálatok	285
11.3. Függetlenségvizsgálatok (homogenitásvizsgálatok)	289
12. Kétváltozós lineáris regresszió	293
12.1. Budapesti lakások vizsgálata	293
12.2. A Kétváltozós Lineáris Regresszió OLS elvű becslése	297
12.3. Az OLS Regresszió magyarázóerejének mérése	304
12.4. A Regresszió magyarázóereje nem megfigyelt adatok körében	305
12.5. A modell együtthatóinak értelemezése	308
12.6. Egy gyakorló példa COVID adatokon	308

13. Többváltozós OLS Regresszió alapjai	313
13.1. Magyar járások COVID-19 halálozási arányai	313
13.2. A Többváltozós Lineáris Regresszió OLS elvű előállítása	316
13.3. A magyarázóváltozók marginális hatása	319
13.4. Magyarázóváltozók fontossági sorrendjének megállapítása t-próba alapján	326
13.5. 5. Konfidencia-intervallumok és a t-próba kapcsolata	329
13.6. Nominális magyarázóváltozók - Használtautó adatok	331
13.7. Dummy változók és működésük Pythonban	332
13.8. A korrigált R-négyzet mutató	337

1. fejezet

Előhang

Ez a jegyzet hivatalosan a Budapesti Corvinus Egyetem gazdaságinformatikus hallgatóinak készült abból a célból, hogy a Statisztika II. tárgy sikeres abszolválásához szükséges elméleti fogalmakat és Python programnyelvi eszközöket egy egységes online tananyagban mutassa be. A jegyzet legfrissebb, naprakész változata Githubon folyamatosan elérhető.

Viszont, a jegyzet elkészítése során arra jutottam, hogy egy kicsit általánosabb célú anyagot szeretnék készíteni: egy olyan átfogó jegyzetet, amely a Következtető Statisztika fogalmait és modern eszközeit olyan szinten mutassa be, amelyek szükségesek ahhoz, hogy a komplex prediktív statisztikai modellek és a gépi tanulás fogalmait, illetve azok matematikai és technológiai hátterét az olvasó önállóan is képes legyen feldolgozni. A motiváció ehhez valahonnan onnan jött, hogy azt tapasztaltam a modern statisztikai tankönyvek és kurzusok világában rengeteg olyan található már, amely gépi tanulás, azon belül is a mélytanulás és nyelvi modellek területével foglalkozik, de ezek jellemzően nem igen foglalkoznak a téma statisztikai alapfogalmaival. Vagy ismertnek veszik őket, és nem igazán részletezik a szemléletes jelentésüket sem, vagy úgy próbálják meg a komplex prediktív modelleket bemutatni, hogy a statisztikai alapokat elfedik/kihagyják. Ez utóbbi eset általában oda torkollik, hogy a prediktív és gépi tanuló modellek kapcsán nem esik szó olyan témakról, amelyek hiányos ismerete a legtöbb gyakorlati alkalmazás kudarcához vezet:

1. **Minden predikciónak van hibája** (lásd a konfidencia-intervallum fogalmát a jegyzetben)
2. **Minden modell** (legyen az bármilyen kifinomult mélytanuló modell is) valamilyen szinten **feltételezéseket és megkötéseket tesz az adatok viselkedésére és/vagy struktúrájára**

A fenti két pont megértéséhez (és így a prediktív modellek sikeres gyakorlati alkalmazásához) pedig elengedhetetlen a következtető

statisztika fogalmainak alapos ismerete. Szóval, a **jegyzet végső célja**, hogy

1. A predikítv modellezéshez és gépi tanuláshoz szükséges **következtető statisztikai alapokat megadja csak a legszükségesebb matematikai formalizmus alkalmazásával**
2. Ahol lehetséges a **matematikai összefüggéseket nem formálisan bizonyítva, hanem azok jelentéstartalmát szimulációkkal szemléltetve**
3. A **következtetős statisztika modern, szimulációs alapú megoldásait is ismertesse** (pl. Bootstrap becslések) a klasszikus eszközök mellett
4. Megmutassa, hogy a **következtetős statisztika eszközei hogyan alkalmazhatók egy egyszerű gépi tanuló modell, a többváltozós regresszió esetében**
5. Mindehhez **korszerű Python nyelvi ismereteket is adjon az adatelemzés területén**

Ennek kapcsán **személyes megjegyzés, hogy a gépi tanulás és prediktív modellezés területén szerintem jobb tankönyv nincs a világban**, mint az elsősorban a Stanford egyetem oktatói által írt és a Springer gondozásában megjelent **Introduction to Statistical Learning (ISL)**. Viszont, ez a tankönyv abba a csoportba tartozik a korábban említettek közül, amely a Következtető Statisztika fogalmait ismertnek veszi, és azok bemutatásával érdemben nem foglalkozik. A jegyzet lényege tehát nagyjából abban foglalható össze, hogy ennek a könyvnek az értelmezéséhez szükséges statisztikai alapfogalmakat és Python programnyelvi ismereteket foglalja össze a lehető legkevésebb előismeret felételezve.

Python oldalról konkrétan semmilyen programozási előismeret nem szükséges a jegyzet olvasásához, a **2. fejezet** a programozás és a Python nyelv alapjaival kezd, teljesen a nulláról. **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, akkor a 2. fejezet neki számára mértékben kihagyható. **Nyugodtan kezdhető tehát a jegyzet a 3. fejezettől a Pythonban** és azon belül **pandasban járatos olvasóknak**. Esetleg a 2.8-2.13. fejezetek gyors árpörgetése ajánlott, hogy a statisztikai számítások elvégzéséhez és a statisztikai adattáblák kezeléséhez szükséges Python megoldásokat felelevenítse. Ugyanakkor fontos megjegyezenem, hogy a **Statisztika II.** tárgy tanóráin a **2. fejezetben szereplő Python 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 a **2. fejezetet is**, és ha bármilyen kérdés van, **KÉRDEZZEN!!** Viszont, statisztikai oldalról néhány előismeretet azért feltételezem a jegyzet megírása során. Konkrétan a következő leíró statisztikai fogalmak ismeretét veszem adottnak:

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

A jegyzet céljának és alapelveinek tisztázása után, szeretném tételesen is 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 és a Következtető Statisztikán 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 adatszerkezeteit (`numpy` és `pandas` csomagok).
- A Python nyelv megoldásait a valószínűségszámítás és következtető statisztika területén (`scipy` csomag `stats` modul)
- A többváltozós lineáris regressziós modellek egyszerű kezelését és használatát Python nyelven (`statsmodels` csomag)
- A Python nyelv legalapvetőbb adatvizualizációs képességeit a `matplotlib` csomagon keresztül.
- Egy *egyszerűbb* adatelemzési folyamatban felmerülő leggyakoribb adatminőségi problémák azonosítási és megoldási módjait Python nyelven
- A *Spyder* fejlesztőkörnyezet működését és lehetőségeit adatelemzési feladatok megoldása során.
- A becsléselmélet és hipotézisvizsgálat alapfogalmainak és a szükséges valószínűségszámítási háttérismereket bemutatása. Levezetések nélkül, szimulációs példákkal szemléltetve a fogalmakat.
- A konfidencia-intervallumok számításának klasszikus, nevezetes eloszlásokon alapuló módszereinek és modern, szimulációkon alapuló megoldásainak (Bootstrap) bemutatása valós adatakon, esettanulmányokon keresztül.
- A gyakorlatban leginkább alkalmazott egy- és kétmintás paraméteres valamint nagymintás nemparaméteres statisztikai próbák bemutatása valós adatakon, esettanulmányokon keresztül.
- A többváltozós lineáris regresszió alapjainak bemutatása és a következtető statisztika eszközeinek alkalmazása a regressziós elemzésekben valós adatakon, esettanulmányokon keresztül.

A jegyzetnek **nem célja**:

- Teljes körű áttekintést adni a Python nyelv elemeiről és adatszerkezeteiről. A Pythont végig csak *szkriptnyelvként* használjuk, nem pedig általános célú programnyelvként.

- A Python minden lehetséges fejlesztőkörnyeztetét (Jupyter Notebook, Visual Studio Code, repl.it.com, stb.) bemutatni.
- A `numpy`, `pandas`, `scipy`, `statsmodels` és `matplotlib` csomagok teljeskörű működéséről. Csak az alapvető leíró statisztikai és adatkezelési funkciókat tekintjük át a következtető statisztikai funkciók részletes bemutatása mellett.
- 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 teljeskörű áttekintése a statisztikai modellezés vagy éppen gépi tanulás területén (pl. `sklearn`, `tensorflow`, stb. csomagok)
- A többváltozós lineáris regressziót is csak az alapok szintjén mutatja be a jegyzet. Nem kerülnek tárgyalásra részletesen a standard (Gauss-Markov) modellfeltételek, és az Ökonometria területét érdemben nem érintjük.

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.

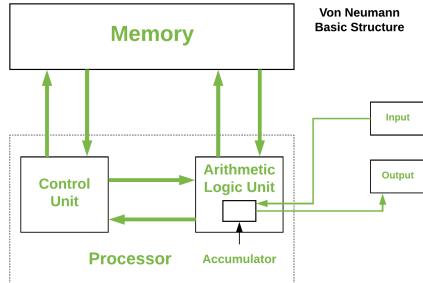
Talán az már kijelenthető, hogy közismert a tény, mi szerint a mai számítógépek alapvetően a Neumann-elvek szerint működnek.

A mi szempontunkból ez csak annyit jelent, hogy a számítógépet alapvetően *utasítások* végrehajtására használjuk programozás során: pl. számold ki ezt, vagy rajzold ki amazt. 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...annyi! Alapvetően a modern számítógépek ennyi alkotóelemből állnak (a háttértár programozás szempontjából irreleváns). Mindez egy cuki ábrán (a processzor belső felépítése minket jelenleg nem érdekel):

122. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK



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 rekroddal és minimum 30-40 oszloppal = változóval rendelkezik).

2.2. A Pythonról általában

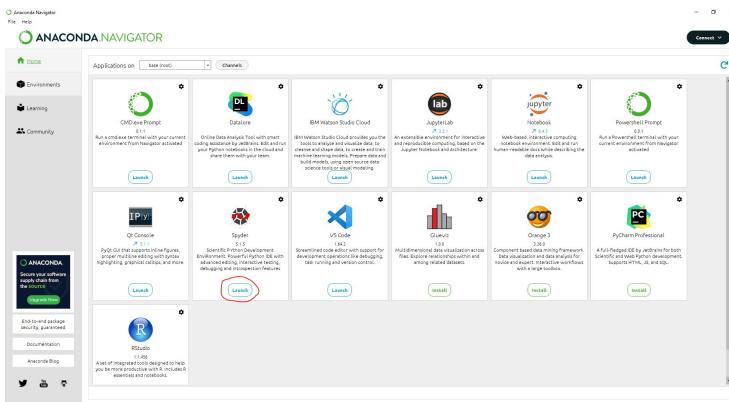
Az Python a jegyzet írásakor a legnépszerűbb általános célú programozási nyelv, 2024 januárjában a TIOBE index alapján a legtöbb sor programkódot Python nyelven írják a fejlesztők.

A mi szempontunkból a Python olyan szempontból vonzó, hogy a külső kiegészítő csomagjai segítségével a valószínűsgeszámítás, 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 Pythonat 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 körülmenyesebb Pythonban, mint R-ben vagy Matlabban. De hát *valamit valamiért*. :)

A Pythonot, mi az **Anaconda keretrendszerből** 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.

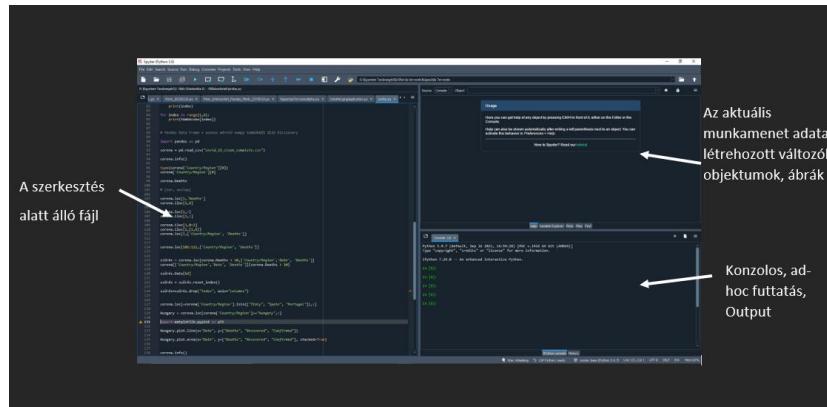
Miután telepítettük és elindítottuk az Anaconda keretrendszert, a kezdőképernyőről rögtön indíthatjuk is a Spyder-t:



2.3. A Spyder felülete

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

142. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK



A Python kódokat a Spyder-ben `.py` kiterjesztésű szkriptfájlokban fogjuk írni. Egy ilyet 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 minden 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 hajtatni 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 minden 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 3×2 :

Ezek után a jegyezetek további részében a feladatok elvégzéséhez szükséges Python kód részleteket és azok eredményét az alábbi módon jelölöm:

3+2

5

3*2

6

Ha az összes *.py* fájlukban lévő kódot le szeretnénk futtatni abban a sorrendben,



ahogy a fájlból 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 rejtelmeibe van még egy fontos dolog, amiről még szót kell ejteni: a *Working Directory* kérdéséről. A *Working Directory* az a mappa, ahonnan a *pitonállat* 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 alapból 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 hajtattunk végre, de a memóriában (RAM-ban) nem tároltattunk el még vele semmit. Most itt az idő! Az utasítások eredményét a = szimbólummal tudjuk a memóriába valamelyen 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 elnevezéseknel. Öreg vagyok már, na! :)). A Spyder képernyőnek ezen a *Variable Explorer* részén látjuk mindenkor 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 mindenkor egy statisztikai megfigyelést leíró tulajdonságot/ismérvet jelent (pl. munkavállaló jövdeleme). Ennek ellenére én is gyakran használom a Python memória objektumokra a változó elnevezést. :)

A Python objektumoknak mindenkor van **adattípusa** is, ami **leírja, hogy az adott objektumban számértékű, szöveges, dátum vagy valami egyéb jellegű adatot tárolunk-e**. Ez azért marha fontos, mert mindenkor az adattípustól függ, hogy minden 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 függvény**vel 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 minden műveletet végeztet el a gépállattal, és a zárójelek között pedig megadjuk, hogy minden 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 mindenkor Python objektum `int` adattípusú, akkor az azt jelenti, hogy ő bezony csak egész számokat tud elközelni a világban, tört számokat nem tud tárolni.

Törtszámok tárolására vannak a `float` adattípusú objektumok.

```
tört = 3/2
type(tört)
```

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

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

Néhány egyéb fontosabb adattípus és megadási módjuk:

```
szöveg = "Hello There!" # Figyeljünk rá, hogy szöveget a kódban csak idézőjelek közé rakunk! Mir
type(szöveg)
```

```
## <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'>
```

182. 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'>
```

Tizedestörtekkel 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` függvényel, akkor annak az egészrészét veszi.

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

```
## 3
```

```
type(fura)
```

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

Pythonban a rosszul beállított adattípusokból születhat 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
```

202. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

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

```
szoveg*szam
```

```
## '3333'
```

Érdemes 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. Ezttől onnan látni a type függvény nélkül, hogy pl. a `float(szoveg)+szam` eredménye 7.0 lesz a `int(szoveg)+szam`-félé 7 helyett.

Viszont, ha a stringben egy tizedes törteket tárolok el (rendesen tizedes ponttal), akkor az már összeveszik az int függvényvel, és ezekben a csillagálásokban hibát dob a pitonállat.

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

```
szoveg+szam
```

```

## 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

```

222. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
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

Egyeszerre több értéket tartalmazó objektumot is fel tudunk venni a Python memoriájába, 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 egységeség 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
```

242. 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 hosszas `__` részek. Illetve, a metódusok zárójelei közé lehet majd egyéb paramétereket is írni, amik szabályozzák a metódus működését. Ilyen például a lista elemeinek sorbarendezési művelete.

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

```
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 minden 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 lehagyom 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']
```

262. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

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

```
## ['Mócsai', 'Musa', 'László', 'Kovács', 'Balázsne', '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 rendezősít 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 megenni, hogy a lista neve után [] zárójelek között :-tal elválasztva megadjuk a kiválasztás kezdeti és végső sorszámát: **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 egyszerűsséggel 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 ismérvt: vezeték- 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'>
```

282. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
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 kódöt csak egyszer kell lefutatni, utána a Python minden emlékezni fog van már neki egy **numpy** névre hallgató kiegészítő csomagja.

Viszont, a következő kódöt minden futtatás előtt 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)
```

```
## <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` függvényel.

```
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!

302. 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 minden azonos adattípusúnak kell lenniük! Ha alapból nem azok, akkor a gépállat átkonvertál minden 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(szöveges_tömb[0])
```

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

```
type(szöveges_tömb[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 Pitonul 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")
```

322. 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 pitonka 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 pitonállat, 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 kód részletet 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
```

342. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

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

Egy `range` keresztségű függvényel bármilyen számsorozatot is ki tudunk íratni egy `for` ciklusban. Csak arra figyeljünk, hogy ez a függvény is 0-tól kezdi 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 kiíratása 2-től 8-ig az alábbi módon lehetséges. Itt a `range`-ben is a felső határ egy nyílt intervallumként megadható, mint a listaelemek :-os kiolasása 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` függvényel kiolvashatjuk a cikluson belül a sorszámuk (indexük) 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 proglémás 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.

362. 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! Gondolunk 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* fájjal 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` metódusának 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 ismérünk/változónk van.

Az **info** metódus eredményéből viszont látszik az is, hogy a **Province/State** oszloban 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 adattípus a pandas data frame object-nek hívja!** Ehhez hozzá kell szokni. :) Onnan jön az elnevezés, hogy általános programnyelvekben az **object** a legáltalánosabb adattípus, adatalemzésben pedig a legáltalánosabb mérési skála, amire minden 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.

382. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

Ami érdekes még, hogy a **Province/State** oszlopban ilyen `NaN` kódok vannak, amik a hiányzó értékeket jelölik. Ugye az `info` metódusból tudjuk ulyebár, 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 nevén 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 műközik, 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,

402. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

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

Szóval, a következő két kód *uggyan 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.

Készítünk 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` metódusával!

422. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
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 változónak 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 kímutatásokat 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'] = coronaConfirmed - coronaDeaths - coronaRecovered
corona.info()
```

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

442. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
## 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 venni 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:

```
szűröttészta = corona.loc[corona.Deaths > 10000, ["Country/Region", "Date"]]
szűröttészta.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-ból 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-ból örökolt index az ő kifacsart géplogikájában:

462. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
szűröttészta.loc[0,:]
```

```
## KeyError: 0
```

```
szűröttészta.loc[17561,:]
```

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

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

```
szűröttészta.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 elszütni. Figyeljük, hogy az eredménnyel felül kell írni az eredeti `szűröttészta` data frame-t!

```
szűröttészta = szűröttészta.reset_index()
szűröttészta.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 oszloposba.

```
szűröttészta.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-ból (ha listát adunk meg ide, akkor egyszerre több oszlopot is tudunk törölni), míg a második paraméterben megadjuk, hogy oszlopokat akarunk törölni, nem pedig sorokat.

```
szűröttészta = szűröttészta.drop("index", axis = "columns")
szűröttészta.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-ból.

Nézzünk még pár szűrést logikai indexszálás segítségével végrehajtva!

Szűrjük le a corona dataframe-ból 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

482. 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 előtérünk egy ~ jelet, akkor pedig tagadást végezünk, 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` metódusával `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 ...
## 1             NaN            Albania ...
## 2             NaN           Algeria ...
## 3             NaN          Andorra ...
## 4             NaN           Angola ...
## ...
## 26395         ...        Western Sahara ...
## 26396         NaN  Sao Tome and Principe ...
## 26397         NaN            Yemen ...
## 26398         NaN          Comoros ...
## 26399         NaN        Tajikistan ...
##
## [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
```

502. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
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()
```

```

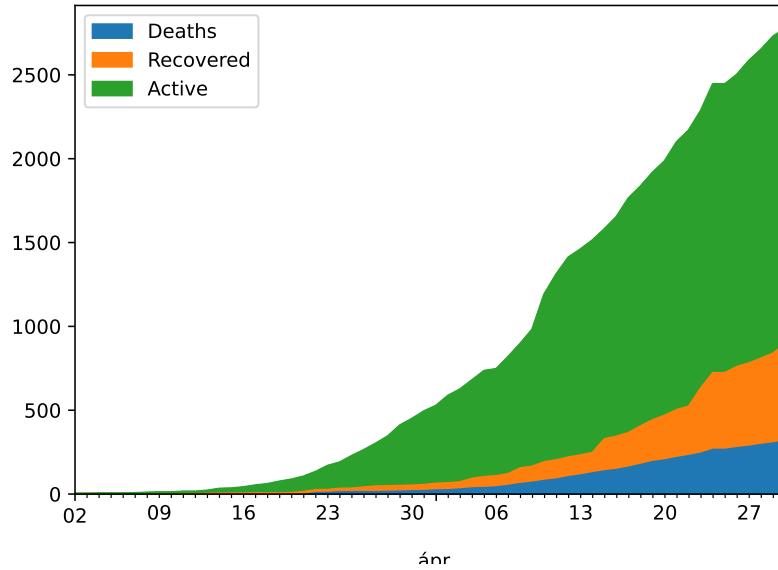
## <class 'pandas.core.frame.DataFrame'>
## Index: 60 entries, 10689 to 26265
## Data columns (total 9 columns):
## #   Column      Non-Null Count  Dtype  
## ---  --          -----          ----- 
## 0   Province/State 60 non-null    object 
## 1   Country/Region 60 non-null    object 
## 2   Lat            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 névterből 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!

542. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
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 legfrissebb 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 0x000002140ED913A0> is curr
## <string>:1: FutureWarning: The provided callable <function median at 0x0000021412168720> is cu
## <string>:1: FutureWarning: The provided callable <function mean at 0x000002140ED913A0> is curr

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

562. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

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 kerektíve 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 kompletter leíró statisztikát a *COVID_Cases* változóra/ismérvre/oszlopra a `describe` metódus segítségével. Kerekítük az eredményeket 2 tizedesjegyre (`round` függvény).

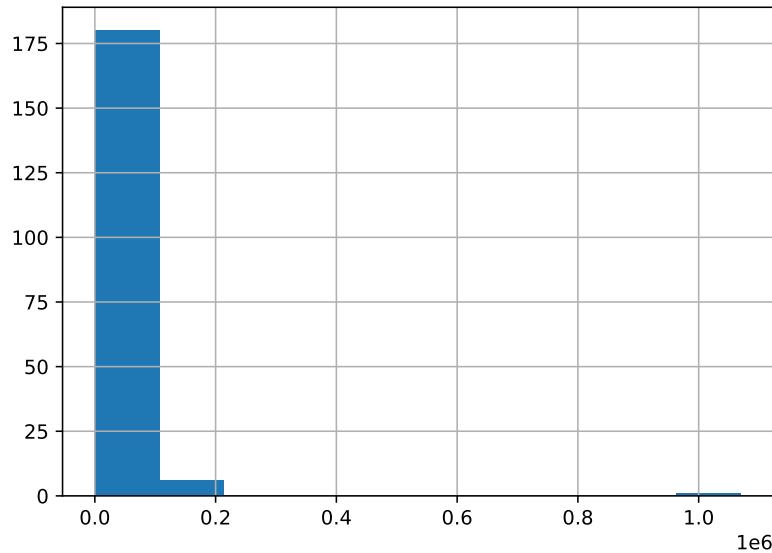
```
round(corona_country.COVID_Cases.describe(),2)
```

```
## 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 ordenáré módon jobbra elnyúló eloszlást hisztogramon és doboz ábrán is!

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

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

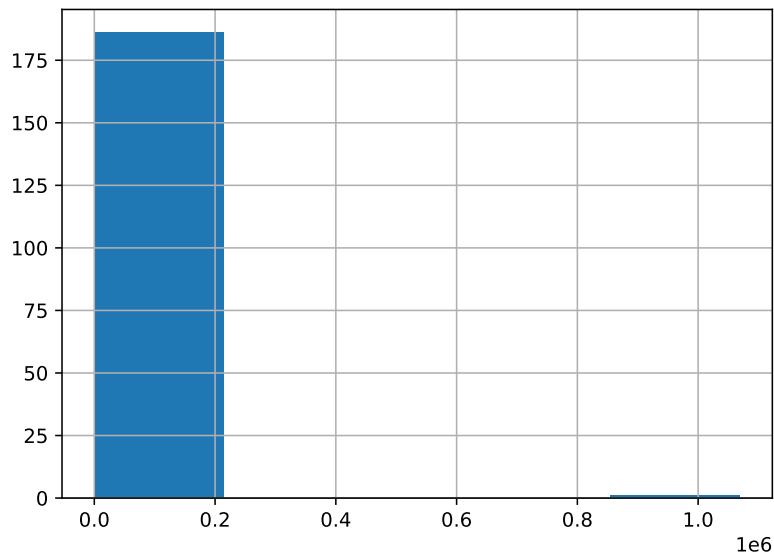


Alapból egyenlő hosszúságú osztályközötket képez a pitonállat a hisztogramokhoz, aminek a számát a `hist` függvényben a `bins` paraméteren keresztül tudjuk szabályozni.

Vigyük le pl. az osztályközök számát 5-re.

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

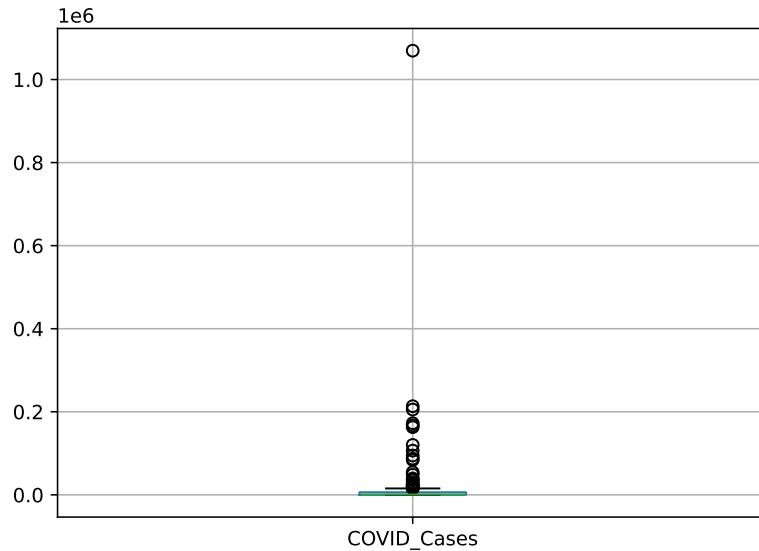
582. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK



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 [] jelleggel) akarjuk az ábrát. Tehát, a doboz ábrát egyszerre több oszlopra is lekérhetjük egy ábrán belülre akár. Majd minden ábrán 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 mentsük 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 
```

602. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
##   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-ból csak az országnév, népesség, népsűrűség és városi népesség aránya változókra lesz szükségünk. A többöt törölhetjük is a data frame-ból! Mivel az oszlopnevekben mint fentebb láthatjuk elég sok a hárnyadék módon speciális karakter, így biztonságosabb most az oszlopokra a sorszámmal 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
```

Egyszerűsítük 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 pitonállat!

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

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

622. 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 kódossal vannak jelölve, hanem ilyen speci „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` oszlopból, arra leginkább az oszlop gyakorisági táblájából lehet felismerni. Ezt a gyakorisági táblát az oszlop `value_counts` metódusával 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 indexseléssel leszűrjük az „N.A.”-kat.

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

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

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

	Pop	PopDensity	UrbanPop
## count	2.220000e+02	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/Km² 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 országoknál alapvető demográfiai ismérveket a `corona_country` data frame-ben lakó országoknál koronavírus esetszámokkal.

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

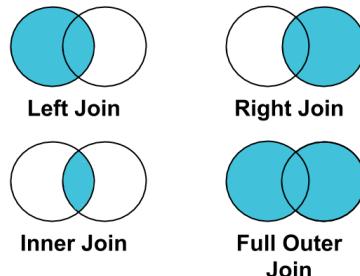
642. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

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 „*adatbázisabban*” szeretném kifejezni magam, akkor azt mondanám, hogy a `merge` függvény 2 tábla joinját oldja meg egy közös kulcs alapján.

Sőt, a `merge` függvény mindenkor alapvető táblakapcsolási módszert támogatja:

- **inner join:** Az összekötött táblában csak azok a sorok maradnak meg, amelyek minden 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 minden sor megmarad.

A különböző típusú összekötési módokat remekül lehet halmagábrákkal szemléltetni:



Na, akkor mielőtt a tényleges `merge`-hez hozzájárunk, annyit ellenőrizzünk le, hogy ugyanaz-e a neve az országneveket tartalmazó oszlopnak minden 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, minden két táblában egységesen **Country**' az összekötésre használandó oszlop neve! Nem meglepő, mert minden két 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 „balról”. :)

```
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** oszlopból.

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

662. 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 más hogy 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 megtámadatva.

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

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

Minthá bejönne az okoskodásunk, de a csehekkel csak nem akarja megtalálni a cucc. Próbálunk 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 lóni a maradék 16 nem egyező országnevet is, de most ezzel nem húzzuk a drága időket, 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 komplexitásban.

Szóval, akkor a 3 azonosított eltérő országnevet javítsuk a **population** data frame-ben. Azaz ott átírjuk ezeket az országneveket arra a verzióra, ami a **corona_country**-ban is szerepel. Ehhez megint az **str.replace**-t használjuk, mint anno az **UrbanPop** oszlop százaléklejelének eltávolításakor. Ezt mindenkor 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
```

682. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

Szupszi! Már nem 168 sor van, amire van találat minden két 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!

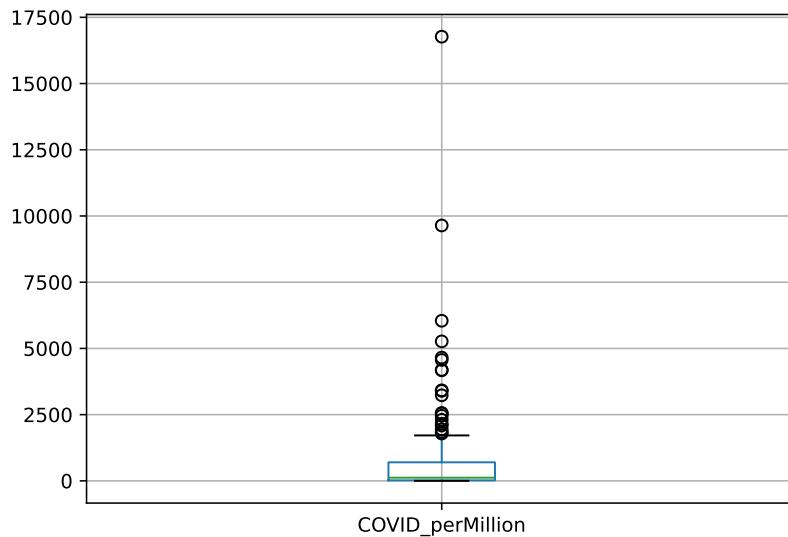
```
corona_extended['COVID_perMillion'] = corona_extended.COVID_Cases / corona_extended.Population  
corona_extended['COVID_perMillion'].describe()
```

```
## 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 egészben 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 sorbarendezett 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 elsüthetünk pl. egy **iloc**-ot az első 10 sor kiválasztásához.

```
corona_extended.sort_values('COVID_perMillion', ascending=False).loc[corona_extended['COVID_perMillion'] > 3000]
```

	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

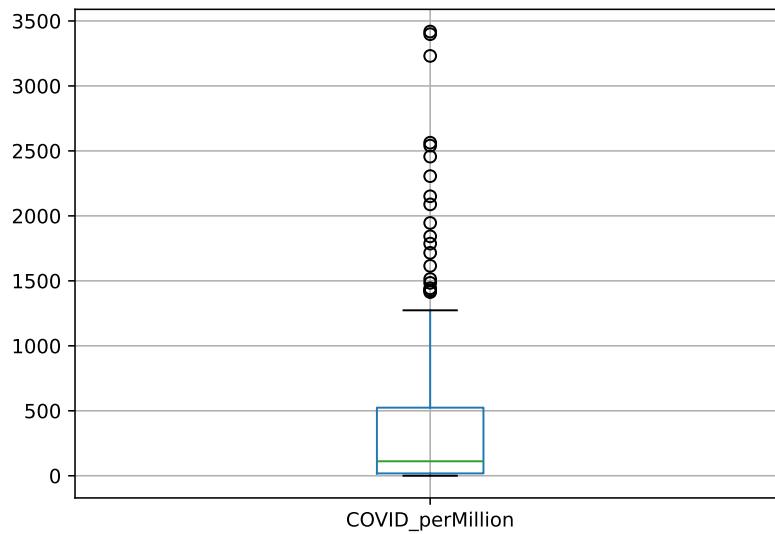
702. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
## 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 töbsegében ilyen jó kicsi, zsúfolt államok. Persze vannak extrém kivételek, pl. ugye az olaszok a nagy repülős turistaforgalmuk miatt.

Na, ezeket az extrém módon kilógó értékeket kipucsoljuk a data frame-ból, aztán ránézünk újra a **COVID_perMillion** doboz ábrájára. Most a kilógó értékeket vegyük csak a 4000 feletti esetszámoknak, mivel a sorrend alapján van egy nagyobb uggrás 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 uggrás az egymillió főre vetített esetszámokban.

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



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

Ennek szellemében még nézzünk rá arra, hogy mely országok esnek az egymillió főre 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

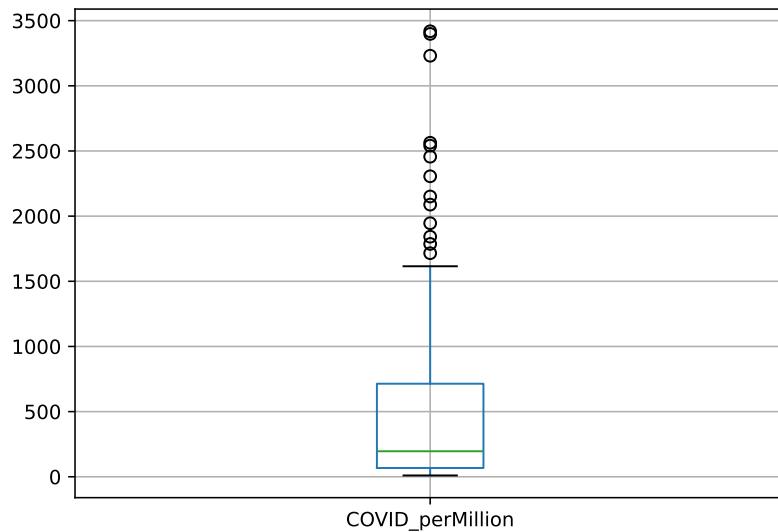
722. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

```
## 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** metódussal kiválasztjuk a két kihagyandó oszlopot, majd az eredményt letagadjuk egy **~** jelrel. Ezt a műveletet pedig beágyazzuk egy **loc** metódusba, 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()
```

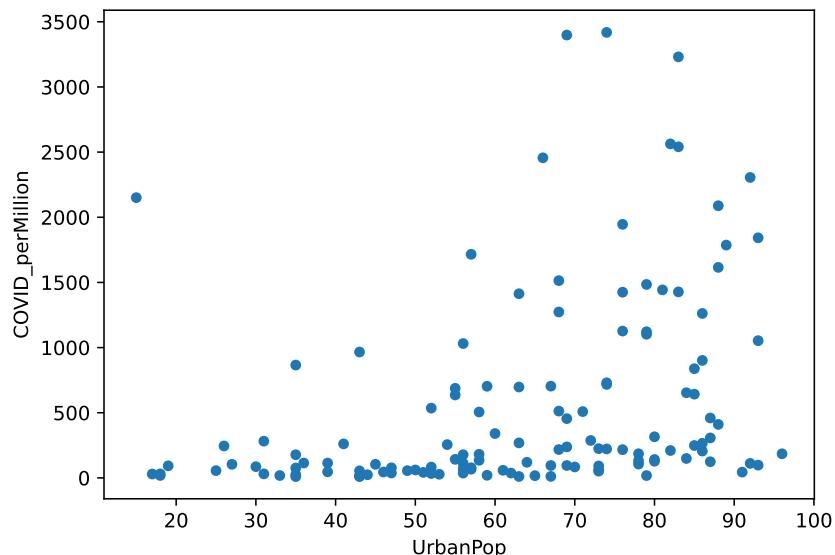
```
##          PopDensity UrbanPop COVID_perMillion
## PopDensity      1.000000 -0.051282      0.118036
## UrbanPop       -0.051282  1.000000      0.341160
## COVID_perMillion  0.118036  0.341160      1.000000
```

742. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

A korrelációs mátrixból látszik, hogy az egymillió főre jutó COVID esetszám leginkább a városi népesség arányával függ össze, teljesen logikus módon: egyirányú, közepes erősséggű 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")
```

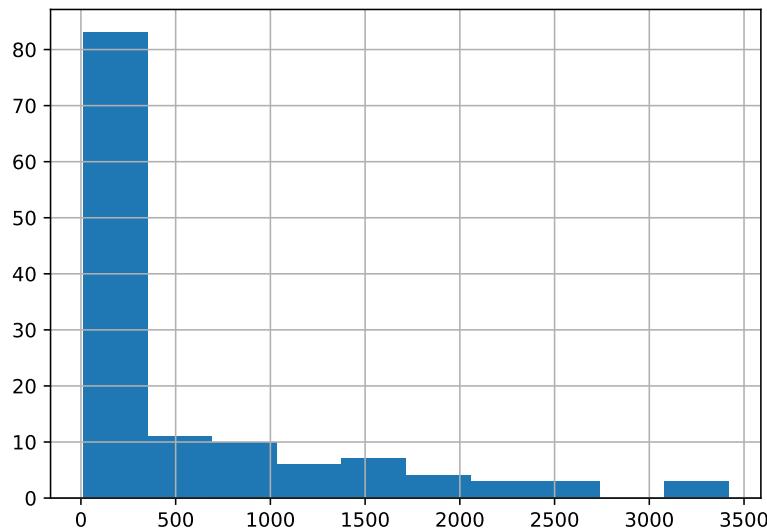


A pontdiagramon azt vehetjük észre, hogy az egymillió főre jutó COVID esetszám jobbra elnyúló eloszlása miatt jelenlévő felfelé kiugró értékek befolyásolják a két ismerv kapcsolatát. A kilógóan magas esetszámok miatt úgy tűnik, mintha az 500 alatti esetszámú országokban nem is lenne kapcsolat a két ismerv 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.

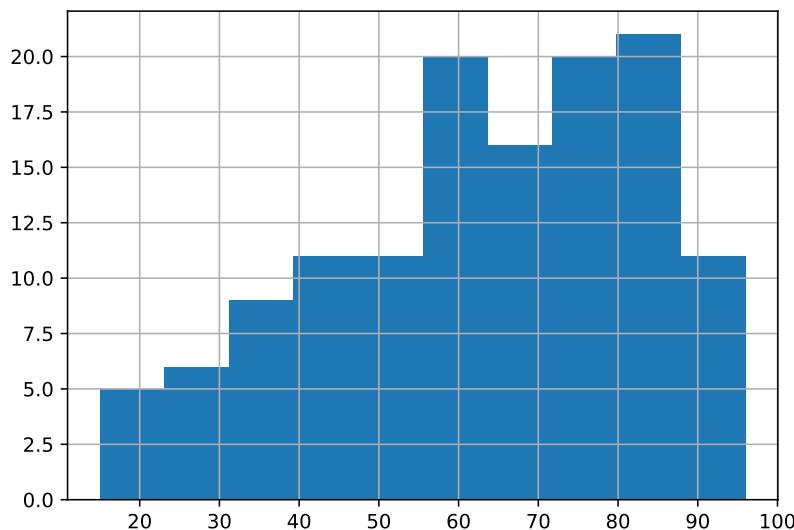
Az egymillió főre jutó esetszám hisztogramja, ami elég jobbra elnyúló.

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



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

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



762. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

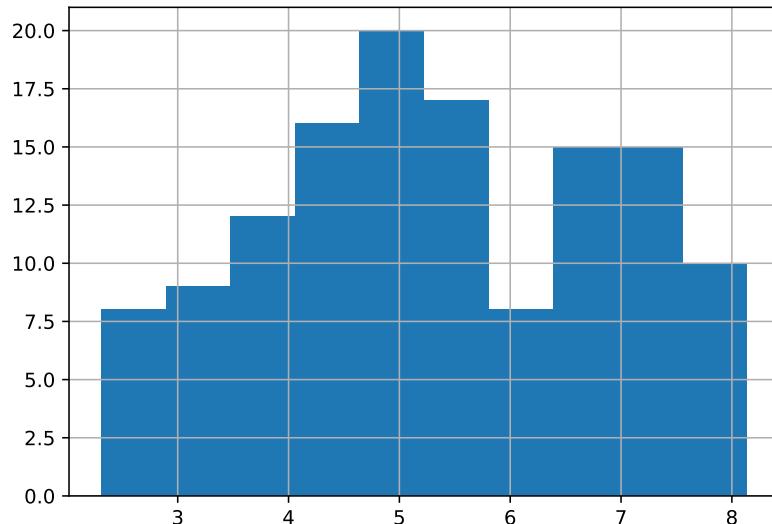
A kiugró értékek hatását, és az **eloszlás jobbra elnyúlóból közel szimmetrikussá** alakítását logaritmussal lehet elérni.

Készítsük is el a **COVID_perMillion** oszlop természetes alapú logaritmusát egy új oszloppban 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 kétmóduszú 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 móduszt” adják.

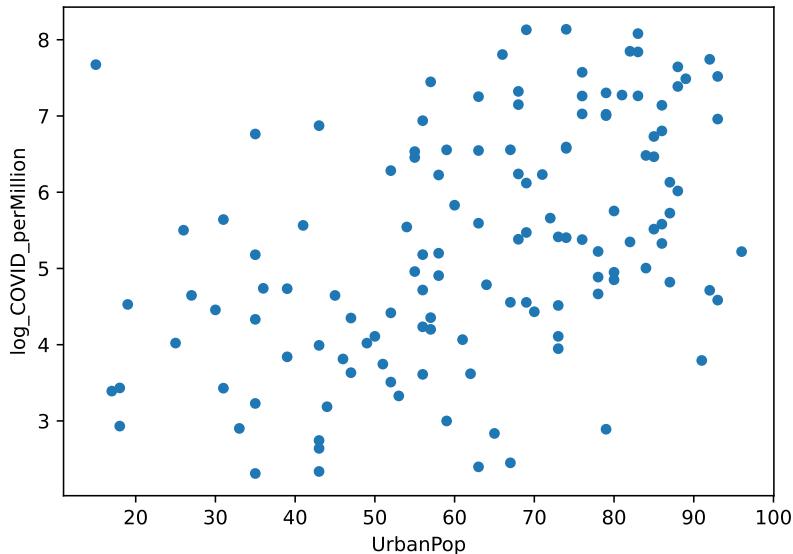
A korreláció az esetszám logaritmusa és a városi népesség aránya között pedig feljavul. Abszolút értékben több, mint 0.1 egységet emelkedik a korreláció, ami nem elhanyagolható mértéjű javulás. :) Most a korrelációs mátrixból kivesszük a **PopDensity-t** is, hogy áttekinthetőbb legyen.

```
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 képeletbeli 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 csinálunk 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 pruhuzamosan:

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

782. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

- 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  
## Azerbaijan  
## 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 minden a *sorszám* 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

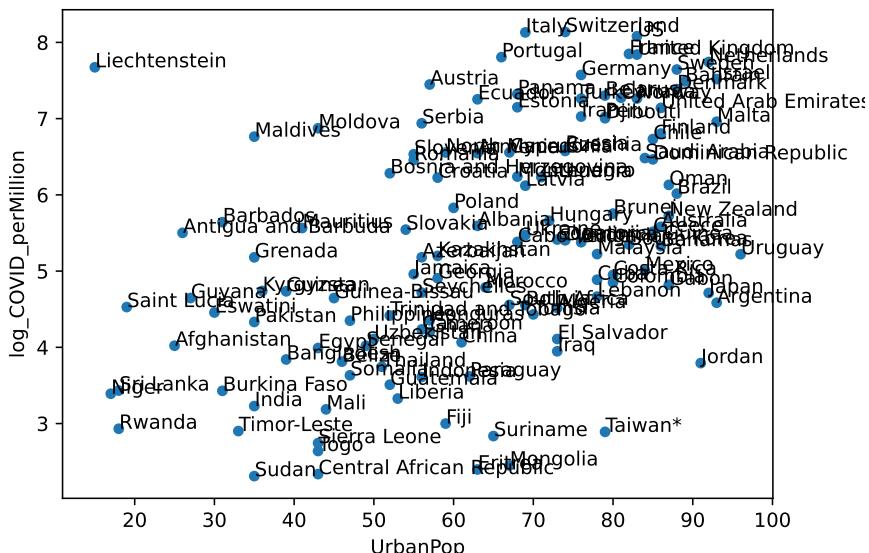
0.05 egységgel jobbra. Így olvashatóbb lesz a cucc. :) Nyilván a felirat *y* koordinátáját is tudnám itt szabályozni, és kedvem szerint fel-le rakni a felirat kezdőpontját, de erre itt nincs szükség, így az `annotate` paraméterben ezt a koordinátát csak csak változatlanul átadom.

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_pe

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! Magyarország, 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 „leennie kéne”. :)

802. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

Gyakorló feladatok

1. Olvassuk be az index_2019_-_pour_import_1_1.csv nevű fájlt, és mentsük 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**) oszlopakra lesz szükségünk. A sajtószabadsági indexben az alacsonyabb érték jelent szabadabb sajtót egy országban. A többi változót töröljük ki a data frame-ból!
3. A szüksített **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 irányá?
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 ismérvek 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

```

822. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK

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')
```

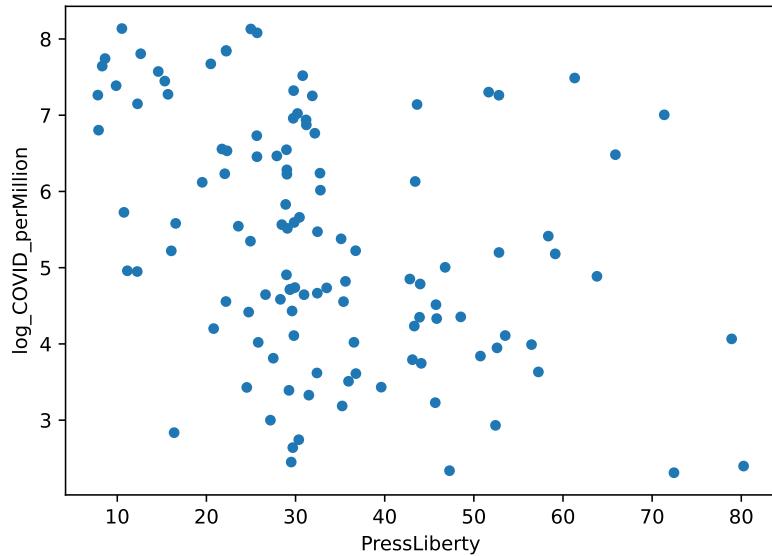
5. feladat

```
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

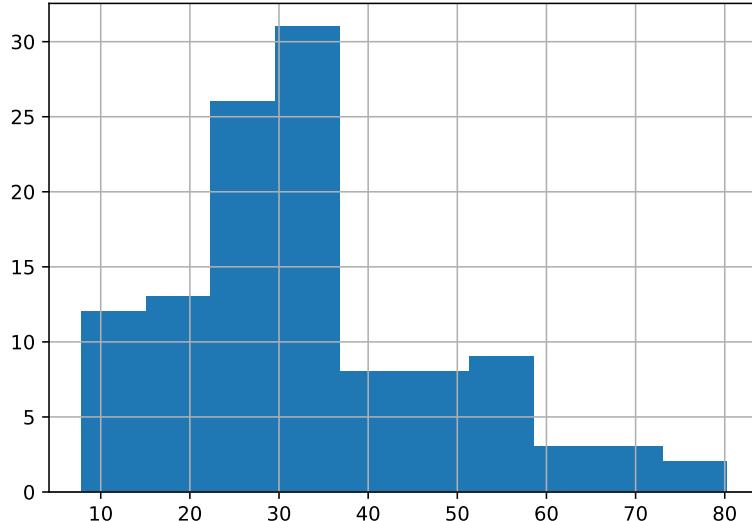
```
corona_extended.plot.scatter(x="PressLiberty", y="log_COVID_perMillion")
```



Úgy látszik, hogy a 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 egymilliós főre vetített esetszámmal vett kapcsolatára tudta mérsékelní 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.

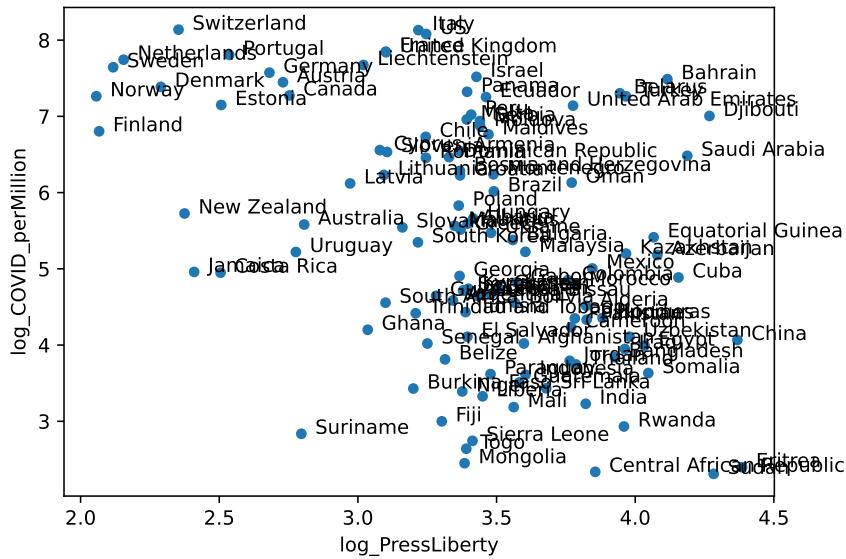
10. feladat

```
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_COVID_perMillion.iat[sorszám]+0.05))

plt.show()
```

862. FEJEZET. STATISZTIKÁHOZ SZÜKSÉGES PYTHON NYELVI ALAPOK



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

3. fejezet

Leíró Statisztika ismétlés 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!

883. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

```
Tesla = pd.read_excel("TSLA.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** oszlopból. 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 minuszokat 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 ($-152\$$) némi leg nagyobb, mint a legnagyobb nyereség ($+129\$$).
- Ugyanakkor, egy átlagos napon nagyjából jól járunk egy Tesla részvénnyel, mert kb. $\mu = \bar{Y} = 1.8\$$ -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 nyereségnél 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 **ennyhe 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 kvartilektől (szóval az 50%-os pont a gyakoriságok szerint, értékekben is kb. a 25% és 75%-os pontok közepén helyezkedik el). Ellenben az **ennyhe 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 kvartilishez, mint a felsőhöz (tehát az adatok többsége egy kicsit inkább az értéktartomány aljára koncentrálódik → 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

903. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

osztályközös gyakorisági táblára, hiszen az árfolyamváltozások értékkészlete elég tág. Az osztályközöket vegyük egyenlő hosszúra. Ezek után már csak azt kell eldöntenünk, hogy **hány osztályközöt** 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 pitonul szeretné kiszámolni arra figyeljen, hogy ott a hatványozás jele a **.

27**

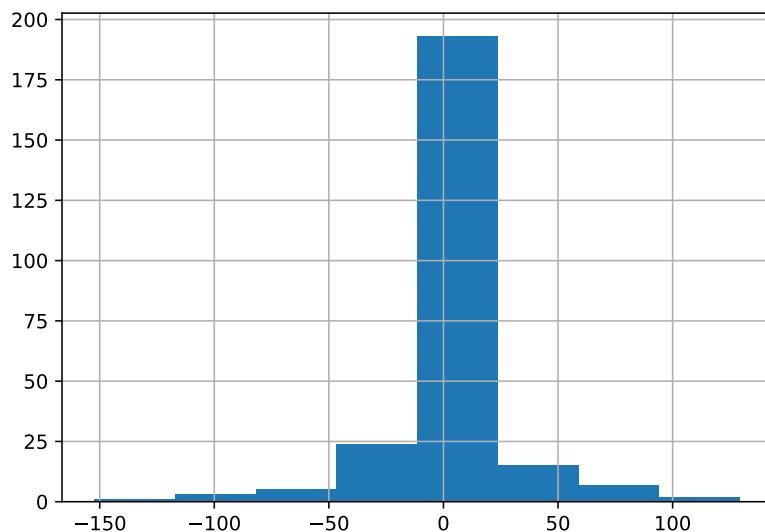
`## 128`

28**

`## 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 értékekkel 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 $N = 250$ -ből. Lássuk mi mondanak erről az α_3, α_4 alakmutatók!

Az α_3 **aszimmetria mutató** Pythonban egy data frame oszlop `skew` metódusával, míg az α_4 **csúcsossági mutató** az oszlop `skew` metódusával számítható.

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

```
## -0.5253045816998407
```

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

```
## 9.086855960563858
```

Az α_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 némi leg kilőgő viselkedése okozza, amire az α_3 ugyebár érzékeny, hiszen az átlag (μ) 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 mentsük 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 transponáljuk (alias 90 fokkal elforgatjuk) a data frame-k `transpose` metódusa segítségével.

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

923. 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 ismérünk (árváltozásunk) maximuma, ami fölött természetesen már nincs érték.

Nevezzük át tartalmuknak megfelelően az oszlopokat, majd a `shift` metódus segítségével rakjuk be a táblába az adott osztályközök felső határait is. Itt a a `shift` metódusnál az alapely, 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 részvénnyen. 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** iis 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
```

943. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

```
## 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 átlaghhoz két dolog kellenek, a **gyakoriságok**, alias f_i -k, és az **osztályközepék**, az Y_i -k. Előbbiekn 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 jelööm 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 osztályközepk 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 pitonkával 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

963. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

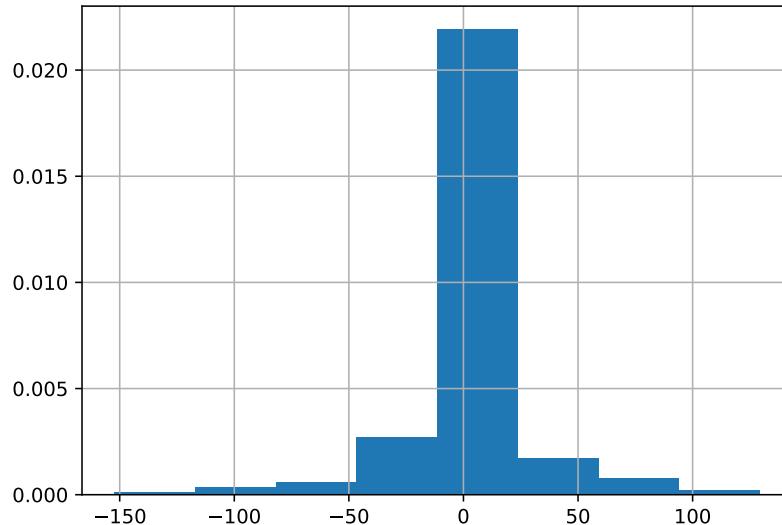
```
szórás = np.sqrt(np.sum(gyaktábla.f_i * (gyaktábla.Y_i-áttag)**2) / len(Tesla.TESLA))
szórás
## 27.048902512450574
```

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

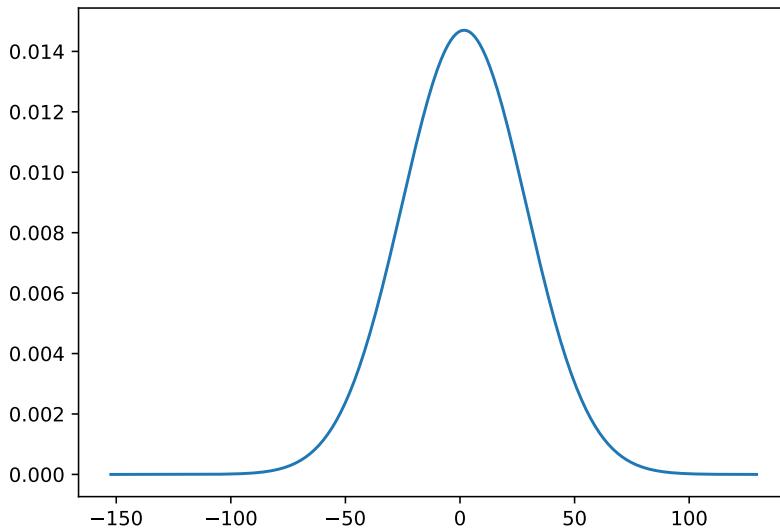
3.2. A normális eloszlás és sűrűségfüggvénye

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 ± 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 tenegelyen, 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 máshogyan 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 vonallal 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 μ átlag és σ szórás függvényében az alábbi alakot ölti.

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

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

983. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

megnyugtatok 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 behelyettesítek 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 pici **pontatlan voltam**. Ugyanis nem egész pontosan a $Y_i = x$ esemény valószínűségét kapjuk meg, mert nagy értékkészlet esetén az gyakorlatilag 0 lenne. Gondolunk 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 ϵ 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*, ϵ -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 ϵ szorzó. Emiatt van az is, hogy a Pythonban a **hist** metódus **density = True** beállítással a relatív gyakoriságokat a legnagyobb gyakoriságú Y_i osztályközép ± 2 környezet relativ 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.

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

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 venni a **scipy** csomag **stats** névterében található függvényeket egy ilyen normális eloszlás sűrűségfüggvény-érték kiszámítására!

Telepítsuk a csomagot és importáljuk a szükséges függvényeket egy `stats` névterbe. 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, μ, σ** . Annyit érdemes megjegyezni, hogy a függvény a μ átlagot location-nek, azaz `loc`-nak, míg a σ 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)
```

```
## 0.01466324747753697
```

Szuper, ez azt jelenti, hogy kb. 1.466% a valószínűsége annak, hogy egy véletlenszerű napon egy Tesla részvénnyel 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 szélen fog „meghízni” a függvény képe, azaz ellaposodik!

3.2.2. A sűrűségfüggvény integrálja

Azért láthatjuk, hogy egy konkrét érték bekövetkezési valószínűsége, alapján nem valami nagy, épp azért, amit fejtégettünk korábban is: az árváltozások értékkészlete elég nagy, egy konkrét érték (vagy annak környezetének) bekövetkezési valószínűsége elég kicsi. Emiatt nem is ezt a kérdést szoktuk általában felenni 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**

1003. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

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 $-\infty$ -től x -ig.

Lássuk akkor hát pl., hogy mennyi a valószínűsége, hogy egy véletlenszerűen kiszűrt 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 μ átlaggal és σ 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 részvényén 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 pitonban a `norm.cdf` függvénytel. 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 részvénnyel 47\$ és 82\$ közti veszteséget produkálunk, akkor megnézem a -47 alá esés valószínűségét, és kivonom belőle a -82 alá esés valószínűségét (tartom a nagyobb 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 értékképzől 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 sűrűségfüggvénytel 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?

1023. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

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% ahogyan a sűrűségszámítás mondja vagy 2.0%, ahogyan 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 vettek csakis figyelembe!!** Tehát a 2.0% esetén a pontos értelmezés az, hogy a megfigyelt napjainknak 2%-a volt olyan, hogy a részvénnyel **47-82 dollárt** veszítettük!

Ezzel szemben a 3.6%, amit az adatokra illeszkedő átlagú és szórású normális eloszlás sűrűségszámításának (Gauss görbüje) alapján számoltunk már egy *elvi valószínűség!* Konkréten, **annak az ELVI valószínűsége, hogy a részvénnyel 47-82 dollár közti összeget veszítek 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 sűrűségszámítás megfigyelt adataimmon **kívüli világot is figyelembe veszi!** Emiatt mondhatom a sűrűségszámításból származó értékeket *valódi VALÓSZÍNŰSÉG*nek, és nem csak relatív gyakoriságnak!

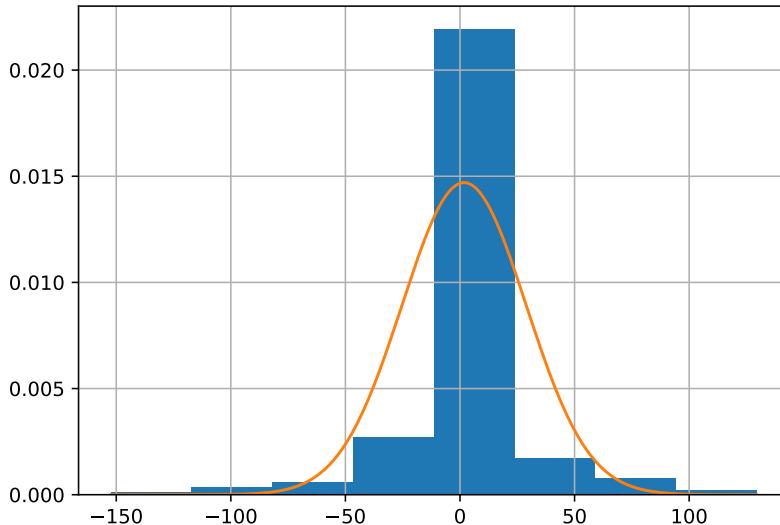
Nota bene: ehhez azért az is kell, hogy az eloszlás, aminek a sűrűségszámításának használom tényleg illeszkedjen az adatokra! Itt azért most a normális eloszlással lehetnek gondok, hiszen amint láttuk pl. az α_4 a Tesla részvénnyek eloszlása kicsit csúcsosabb az $N(1.8, 27.19)$ eloszlás sűrűségszámításánál.

Azt, hogy egy eloszlás sűrűségszámításának mennyire illeszkedik a megfigyelt adatok hisztogramjára a következőképpen tudjuk grafikusan megvizsgálni.

- Először elkészítjük a hisztogramot a `hist` metódussal, `density = True` paraméterrel, hogy az y tengely skálázása összemérhető legyen a sűrűségszámításának 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-ból 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()` parancccsal kiképítsük, hogy ezt a többrétegű árat így egyben mutassa meg nekünk a gépállat.
- **FONTOS!** Az alábbi kódsort mindenkor mindenkor 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()
```



1043. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

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 **valószínűségszmítási 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 összegződéseként áll elő, akkor az adatok hisztogramja normális eloszlású sűrűségfüggvényt követ.

A téTEL tehát ilyen klasszikus ha \rightarrow akkor típusú matematiaki 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 metódusával.

```
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 délután 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 már tümtikéznek a tervezett Tesla gyár ellen, ami megint elkezdi levinni a részvény értékét és a

nap végére uda jutunk, hogy a részvény $-12.510009\$$ -al zár... Szóval az adott nap véletlenszerű gazdasági eseményeinek összegződéseként 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 tételem 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 szabotór 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 veszítek 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 quantile=szóból), a μ átlagot (`loc`) és a σ szórást (`scale`). Akkor hát lássuk mi is az az érték, aminél csak 5% valószínűséggel veszítek 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 másnéven: 5% 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.

1063. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

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 árváltozások 5. percentilise! Itt most a data frame **quantile** metódusát vetjük be, aminek a paraméterében tizedestörteként kell megadni a keresett percentilis sorszámát. Tehát az 5. percentilisnél 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 súrűségfüggvényből származó 42.85\$-os veszteség!** És valószínűleg a súrűségfüggvényből származó éték a reálisabb, 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 megtehette. 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 befektetésekhez 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 valószínűsséggel 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 gépszellemről: **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 megemlékezniünk 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 strandard 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 valamelyen eloszlást követ, azt \sim jelrel 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 kereítés 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

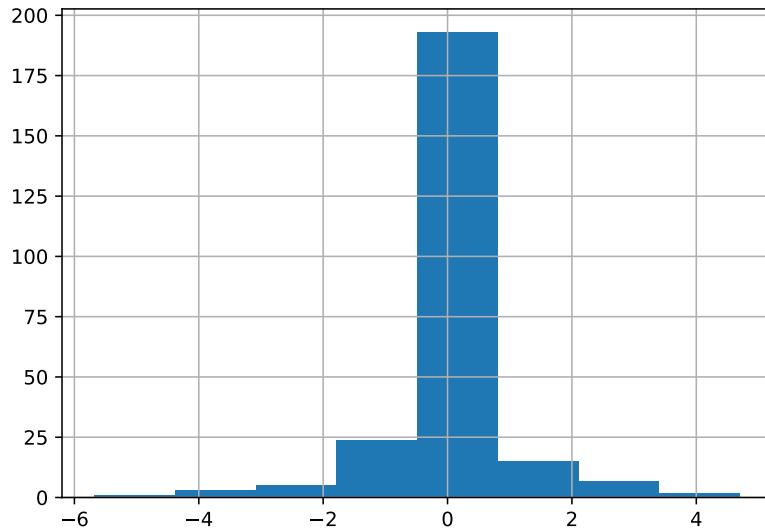
1083. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

```
## 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)
```

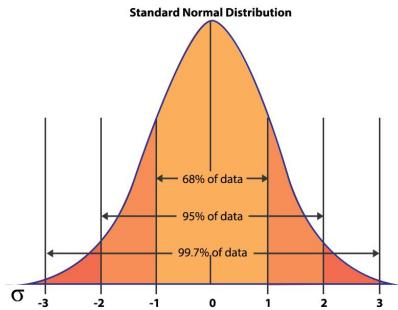


Ami miatt szeretni szokás a standard normális eloszlást az a jellemzője, hogy

- 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 ± 2 -re. A `norm.cdf` függvény ugyanis `loc=0` és `scale=1` beállításokkal fut, ha nem adunk meg neki mászt. Tehát szűmöljük 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 ± 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 széla 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

1103. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

```
## 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 σ szórás hányszorosával tér el a μ átlagtól. Tehát, pl. a fenti szürésben 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 vizsgálunk 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 valósak, 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  
## ---  -----      -----          -----
```

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

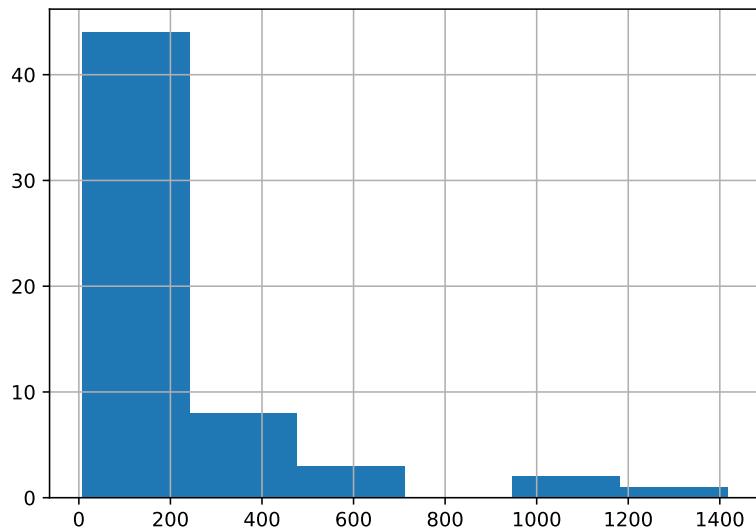
```
Surv.head()
```

```
##   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 legksiebb 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.

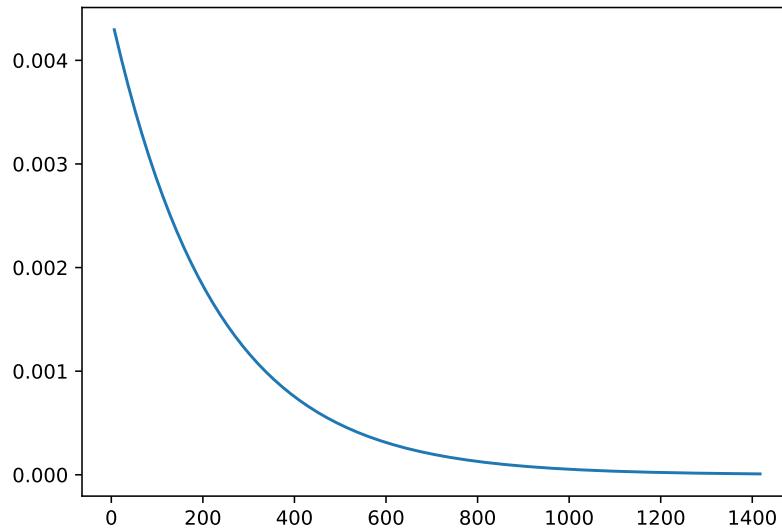
```
Surv.SurvMonth.hist(bins=6)
```



1123. 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 λ paraméter határozza meg. Minél nagyobb λ , annál meredekebben jobbra elnyúló a sűrűségfüggvény. Ezt alább lehet kipróbálni.

Természetesen a λ -nak van köze a valós adatok átlagához és szórásához, egész konkrétan minden 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 (λ) 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      58.00000
## 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 λ -t kell megadni a külön μ és σ helyett, mint ahogy a normális eloszlásnál működött a dolog. A **scipy** csomag ezt úgy oldja meg, hogya a szórásból számolja vissza a λ -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 $Exp(0.00365)$ eloszláskövetnek: $Y_i \sim Exp(0.00365)$

Ezek alapján számolunk ki pár valószínűséget a túlélési időkre vonatkozóan:

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

```
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 elni?

```
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 hónapközött fogunk elpatkolni?

1143. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

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

```
## 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`.

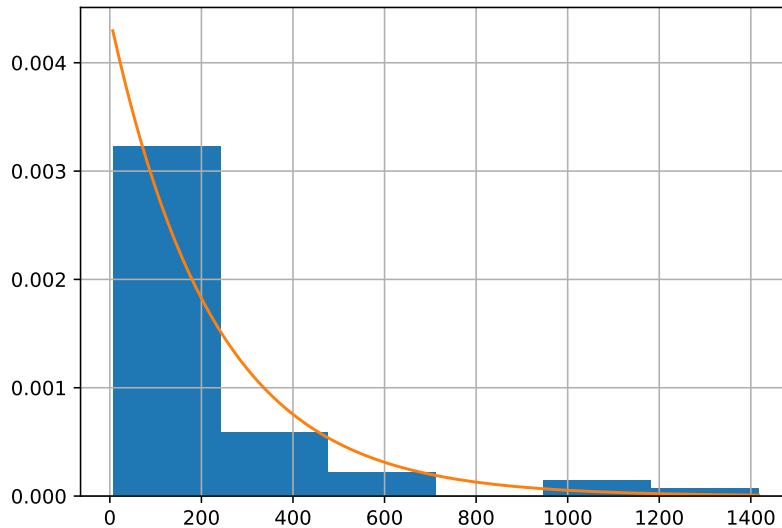
```
stats.expon.ppf(0.99, scale = np.std(Surv.SurvMonth))
```

```
## 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 megettük 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()
```



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

1163. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

A legutolsó kérdés a listán elsőre 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 minden hónapban állandóan 3 mérést vannak a nyári hónapokban (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 hónappját, 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
```

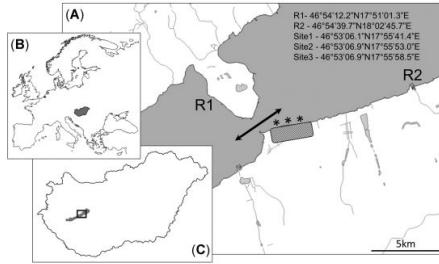
```
Balcsi.head()
```

```
##      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 BALATONBAN117

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 ismérvünk a mérés hónapja, mennyiségi ismérvünk a kokaintartalom lesz. A két ismérv kapcsolatának szorosságát pedig akkor a varianciahányados adja meg.

A varianciahányados kiszámításának első lépése egy olyan segétáblázat összeállítása, amely sorait a minőségi ismérv lehetséges értékei adják, és 3 oszlopa van, ami a minőségi ismérv j indexsel jelölt csoportjai szerint bontva tartalmazza az elemszámokat (N_j), a mennyiségi ismérv részátlagait (\bar{Y}_j) és szórásait (σ_j). Ezt a segédtáblát Pythonban a data frame-k groupby és agg metódusaival 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 oszlopnevket). Illetve, ne felejtsük el a végén a ‘reset_index’ metódus használatát, különben a minőségi ismérvünk értékei a data frame sorindexeként 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 0x000002140ED913A0> is currentl
## <string>:1: FutureWarning: The provided callable <function std at 0x000002140ED914E0> is currentl
```

Segéd

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

1183. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

```
## 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 szemmelveréssel**, mivel a **kokain mennyiségek szórása is ebben a hónapban a legnagyobb**. Sőt, ez az egyetlen hónap, amikor a konkrét mérések kokain mennyiségeinek szórása *nagyobb* az átlagos kokain mennyiségénél! Szóval, kell azért egy check arra a varianciahányadosra.

A varianciahányados, a H^2 mutató értékéhez úgy jutunk el, hogy elkezdjük az előbb felépített segédtáblázatunk alapján kiszámolni a mennyiségi ismérv (azaz most a kokain mennyiség) teljes, hónapoktól független teljes átlagát és teljes szórását. Ugyebár a teljes átlagos kokainmennyiség (főátlag, \bar{Y}), nem más, mint a rész átlagok (\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 ráeresztsük 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 ismérv) teljes szórása (σ) csak úgy kapható meg a segédtábla alapján, ha előbb kiszámoljuk a belső szórást (σ_B) és a külső szórást (σ_K).

A belső szórást a **belső variancián, azaz belső szórásnégyzeten keresztül kapjuk meg**. Ez pedig nem más, mint a **részszórások** σ_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 BALATONBAN119

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

```
belso_var = np.sum(Seged.Elemszam * Seged.Szorasok**2)/(np.sum(Seged.Elemszam))
belso_var
```

```
## 3062.6571029636407
```

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

Konkrét esetünkben ez az alábbi módon néz ki.

```
belso_szoras = np.sqrt(belso_var)
belso_szoras
```

```
## 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 ismérv 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.

```
kulszo_var = np.sum(Seged.Elemszam * (Seged.Reszatlagok - főátlag)**2)/(np.sum(Seged.Elemszam))
kulszo_var
```

```
## 933.983870298589
```

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

Konkrét esetünkben ez az alábbi módon néz ki.

1203. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

```
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 mennyiségnak 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 általában ugye azt mutatja meg, hogy egy véletlenszerűen kiválasztott egyed konkrét mennyiségi ismérv értéke várhatóan mennyivel tér el a mennyiségi ismérv csoportuktól 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)
```

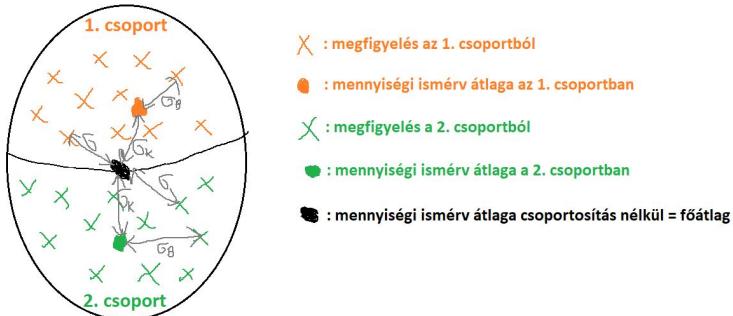
```
## 63.08427864039577
```

Hát ez csak majdnem stimmel. Ennek az oka az, hogy a `Segéd` data frame-ben a rész átlagok és részszórások 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 varianciahányados 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 ismérv csak két csoportot alkot (*narancsok* és *zöldek*), nem pedig hármat, mint amennyit a mi három hónapunk generál.

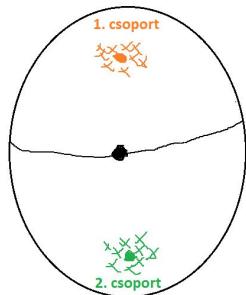
3.4. A VARIANCIAHÁNYADOS PYTHONBAN - KOKAIN A BALATONBAN121



Tehát vizuálisan a következőképp érdemes gondolni a különböző σ -kra:

- σ_B : Megfigyelések távolsága saját csoportjuk átlagától
- σ_K : Csoportátlag távolsága a főátlagtól
- σ : Megfigyelések távolsága a főátlagától

Ezek alapján nekünk az a jó a csoportosítás, azaz a minőségi magyarázóereje szempontjából, ha fix σ mellett σ_K nagy és σ_B kicsi. Mert ekkor a **csoportátlagok messze vannak** a főátlagtól és így implice **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 ismérviünk magyarázóereje nagy! Tehát az kell nekünk, hogy az σ minél nagyobb részét tegye ki σ_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 $\frac{\sigma^2}{\sigma^2}$ hárnyados 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 $\frac{\sigma^2}{\sigma^2}$ variancia-hányados biztos, hogy 0 – 1 közötti, és **százalékosan** is értelmezhető, hiszen σ_K^2 része σ^2 -nek. Ez alapján nekünk most:

1223. FEJEZET. LEÍRÓ STATISZTIKA ISMÉTLÉS ÉS VALÓSZÍNŰSÉGSZÁMÍTÁS ALAPOK

$\frac{\sigma_K^2}{\sigma^2} = \frac{933.98387}{3996.64097} = 0.23369 = 23.369\% \rightarrow$ A hónap (tehát a Sound jelenléte) a Balatonban mért kokainmennyiség alakulásának (varianciájának) kb. 23%-át magyarázza a megfigyelt adatok körében! Ez egy közepes magyarázóérőnek tekinthető, mivel a variancia-hányadost 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 hónapkon belül is jelentős mértékű szódósás 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 variancia-hányadósát kell megvizsgálnunk.

Először nézzük a mérőállomások esetét.

Itt is kell ugyebár egy kiinduló segédtáblázat.

```
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 0x000002140ED913>
## <string>:1: FutureWarning: The provided callable <function std at 0x000002140ED914E>
```

```
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 BALATONBAN123

Olybá túnik, hogy az 1. állomás átlagban némileg kicsit magasabb kokainmenyniséget 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 örököitem** az előző számolásokból, hiszen az a csoportosítás alapját képzőü minőségi ismérő meg változásával **nem változik**.
- Az előző pont és a belső variancia kiszámolása után pedig a varianciahányadost a $\sigma^2 = \sigma_B^2 + \sigma_K^2$ összefüggás 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**.

```
belso_var_Allomas = np.sum(AllomasTabla.Elemszam * AllomasTabla.Szorasok**2)/np.sum(AllomasTabla.Elemszam)
VarHanyadAllomas = 1 - belso_var_Allomas / teljes_var
VarHanyadAllomas
## 0.01174053573946432
```

Az eredmény minden 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 tekintetők!

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 0x000002140ED913A0> is currentl
## <string>:1: FutureWarning: The provided callable <function std at 0x000002140ED914E0> is currentl
```

```
EvTabla
```

1243. 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 másfélszerezre $\frac{97.37}{62.42} = 1.56$. Szóval itt lehet még komolyabb magyarázóerő! De ne találgassunk, hanem lássuk a varianciahányadost! A számolásnál megint alkalmazzuk az állomásoknál bevezetett két **egyszerűsítést!**

```
belső_var_Ev = np.sum(EvTabla.Elemszam * EvTabla.Szorasok**2) / np.sum(EvTabla.Elemszam)
```

```
VarHányadEv = 1 - belső_var_Ev / teljes_var  
VarHányadEv
```

```
## 0.20797660221119585
```

Na, itt is egy közepe 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 megbízhatóanek tekinthetők!

4. fejezet

Eloszlások és Mintavételezés

4.1. Véletlenszám generálás és Mintavételezés

Minden programnyelv alapelemének számít egy olyan függvény, ami $0 - 1$ értékek közötti véletlen számokat generál. Egész pontosan ezek a függvények **úgy generálnak számokat, hogy azok $0 - 1$ között minden értéket azonos valószínűséggel**, tehát **egyenletes eloszlással** vehetnek fel. Az ilyen $0 - 1$ között egyenletes eloszlású véletlenszám generátorok minden programnyelv alapvető eszközei, és a Számítástudományból tanult lineáris kongruenciarendszereken alapulnak.

A $0 - 1$ közötti teljesen véletlen számok generálásához Pythonban egy `random` c. csomagot kell importálni, és ennek a (nem meglepő módon) `random` névre keresztelt függvényével tudunk véletlen számokat dobálni.

```
import random

GondoltamEgySzámra = random.random()
GondoltamEgySzámra

## 0.09082371360654995
```

Fenti kódsort futtatva nyilván mindenkinél más eredménye lesz, hiszen *random* számot generálunk. :) A lényeg, hogy a számunk $0 - 1$ közötti!

Node, mit csinál ez a véletlenszám generátor statisztikus szemmel tekintve? Nos, ilyenkor ők egy olyan Y_i adatsorból húznak véletlenszerűen

egy x számértéket, aminek az eloszlása $0–1$ közötti egyenletes eloszlás. Ennek a matematikai jelölésrendszere: $Y_i \sim U(0, 1)$. Az U onnan jön, hogy az egyenletes eloszlás in English *uniform distribution*.

Húzzunk akkor most ebből a $Y_i \sim U(0, 1)$ adatsorból 50 db véletlen értéket, és tároljuk el az eredményeket egy Python list objektumban! Azaz, **vegyünk a $U(0, 1)$ eloszlásból egy $n = 50$ elemű mintát!**

Technikailag úgy kell eljárnunk, hogy létrehozunk egy üres listát Pythonban, és az `append` metódusa segítségével foltoljuk 50 db random $0–1$ közötti számmal.

```
SzerencsétlenVéletlenek = [] # üres lista létrehozása

for index in range(50):
    SzerencsétlenVéletlenek.append(random.random()) # véletlen szám hozzáadása a listához

SzerencsétlenVéletlenek
```

[0.545626169578343, 0.9167775390495727, 0.4249307778984388, 0.09563866723341008, 0.1...

Itt is van az 50 szép kicsi véletlen számom. Nyilván ezek megint mindenkinél más értékek. :)

Következő lépésben **ezt az 50 elemű mintát transzformáljuk át olyanná, hogy ne egy $U(0, 1)$ eloszlásból, hanem mondjuk egy $U(40, 160)$ eloszlásból származó minta!** Magyarul, az kéne, hogy ez az 50 véletlen szám, ne 50 db $0 – 1$ közötti véletlen szám, hanem 50 db $40 – 160$ közötti véletlen szám legyen!

Ezt a következőképpen tudjuk elérni:

- Ha a $0 – 1$ közti véletlenszámokat megsorozzuk $160 – 40 = 120$ -szal, akkor $0 – 120$ közti véletlen számok lesznek.
- Ha a $0 – 120$ közti véletlen számokhoz hozzáadunk 40-et, akkor $(0 + 40)$ és $(120 + 40)$ közti, azaz $40 – 160$ közti véletlen számok leszenek.

Ahhoz, hogy ezt technikailag kivitelezni tudjuk a `s` listánkat először numpy tömbbé kell konvertálni. De utána elég egyszerűen menni fog a dolog.

```
import numpy as np

SzerencsétlenVéletlenek = np.array(SzerencsétlenVéletlenek)
SzerencsétlenVéletlenek = (SzerencsétlenVéletlenek * (160-40)) + 40
SzerencsétlenVéletlenek
```

array([105.47514035, 150.01330469, 90.99169335, 51.47664007,

```

##      126.33338682, 122.68924064, 55.34564897, 105.71937447,
##      48.44943657, 152.70066955, 101.74558253, 150.52494645,
##      56.17891077, 59.70343748, 144.37588857, 68.98453815,
##      47.81720942, 110.08277665, 101.32958456, 48.44160652,
##      117.54828329, 54.20625513, 79.9338664 , 48.15935041,
##      99.02266132, 65.32341873, 53.85617975, 47.41269973,
##      66.84837181, 122.42388624, 59.72725619, 130.4206215 ,
##      155.6477233 , 118.43277676, 63.96821332, 78.89768941,
##      84.69169809, 155.93952376, 134.9262267 , 40.22954029,
##      148.52700969, 140.26872004, 132.62687294, 154.77985502,
##      117.50605762, 43.17645376, 82.40709135, 80.90017844,
##      155.86687063, 54.41856955])

```

Olybá tűnik a foglalkozásunk elérte célját: tényleg nincs itt 40-nél kisebb és 160-nál nagyobb szám! :)

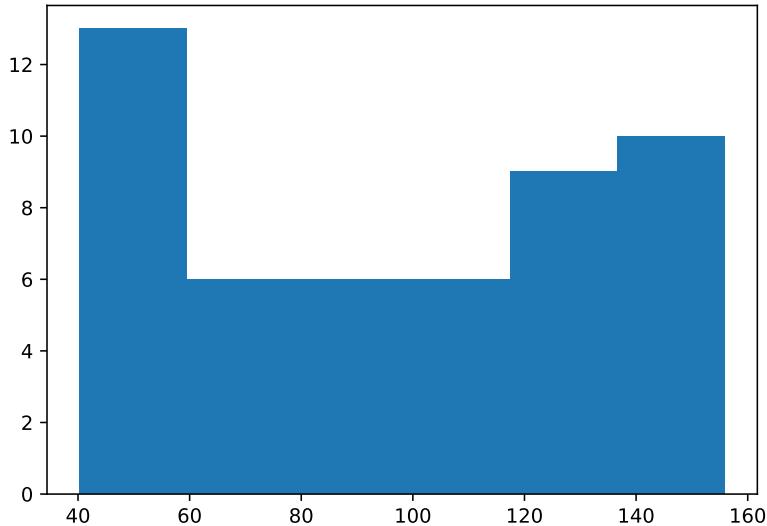
Nézzük meg, hogy mennyire egyenletes eloszlásúak a generált $U(40, 160)$ eloszlású számaink! Rajzoljuk ki a hisztogramot! Ehhez kelleni fog a `matplotlib` csomag `hist` függvénye is. Eza függvény lényegében a `numpy` csomag `histogram` függvénye, de listákra csak ezt lehet alkalmazni. Ugyebár még nem mentettük a dolgokat `pandas` data frame-be, ezért ilyen körülöményes kicsit a hisztogram rajzolás. Mivel 50 adatot generáltunk, így a hisztogramhoz $k = 6$ osztályköz kell a $2^k \geq N$ szabály alapján ($2^6 = 64$)

```

import matplotlib.pyplot as plt

Hisztogram = plt.hist(SzerencsétlenVéletlenek, bins = 6)
plt.show()

```



Mit is kéne látni ezen a hisztogramon? Hát általánossában az $U(a, b)$ eloszlás, azaz az (a, b) intervallumon egyenletes eloszlás sűrűségfüggvénye $f(x) = \frac{1}{b-a}$, hiszen a sűrűségfüggvény helyettesítési értéke az egy konkrét x érték bekövetkezési valószínűsége az $U(a, b)$ eloszlásban, $P(Y_i = x)$. Ami az egyenletes eloszlás esetén minden x -re $a - b$ között ugyanannyi, nevezetesen „1 per az (a, b) intervallum hossza!

Esetünkben tehát **az $U(40, 160)$ eloszlás sűrűségfüggvénye** az $f(x) = \frac{1}{160-40} = \frac{1}{120} = 0.0083$ magasságban futó vízszintes egyenes. Vagyis a hisztogramnak minden x értékre kb. ugyanakkora gyakoriságot kell mutatnia, ami azért szemmel láthatóan nem így van!

Rajzoljuk csak ki a hisztogramot `density = True` beállítással és véssük fel rá az $U(40, 160)$ eloszlás sűrűségfüggvényét! Ezt az ábrát hasonlóan lehet elkészíteni, mint ahogy a normális eloszlás sűrűségfüggvény illeszkedését vizsgáltuk garfikusan a Tesla árváltozások hisztogramjára az 3.2.3. fejezet végén. Sőt, természetesen a `scipy` csomag `stats` névterének `uniform.pdf` függvényével is ki tudjuk számolni az $U(40, 160)$ eloszlás konstans $f(x) = 0.0083$ -as sűrűségfüggvényét, ahogy az 1. heti anyagban kiszámoltuk a normális eloszlás sűrűségfüggvény $f(x)$ értékeit a `norm.pdf` függvényivel. Arra kell figyelni az `uniform.pdf` függvénynél, hogy a `loc` nevű paraméterben kell megadni az $U(a, b)$ eloszlás alsó határát, azaz a -t, míg a `scale` paraméterben a **terjedelemét**, azaz az $a - b = 160 - 40 = 120$ -at!

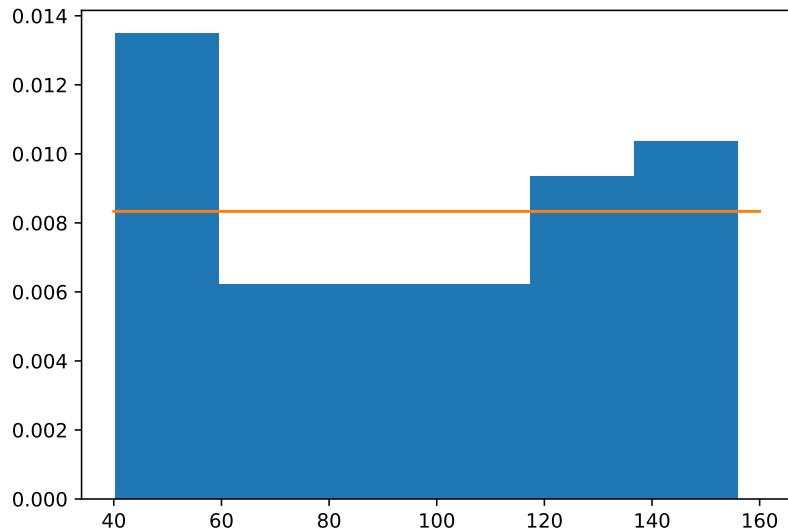
```
import scipy.stats as stats
```

```

eloszlás_alsó_határ = 40
eloszlás_felső_határ = 160
eloszlás_terjedelem = eloszlás_felső_határ - eloszlás_alsó_határ

Hisztogram = plt.hist(SzerencsétlenVéletlenek, bins = 6, density = True)
x_tengely = np.arange(eloszlás_alsó_határ, eloszlás_felső_határ, 0.01)
y_tengely = stats.uniform.pdf(x = x_tengely, loc = eloszlás_alsó_határ,scale = eloszlás_terjedelem)
plt.plot(x_tengely, y_tengely)
plt.show()

```



Láthatjuk azért, hogy az 50 elemű **minta tapasztalt gyakoriságai nem annyira tükrözik az egyenletes eloszlás vízszintes $f(x) = 0.0083$ magas sűrűségfüggvényét, de azért többnyire a függvény vonala körül mozognak.** Azt pedig már lecsekkoltuk, hogy a vgenerált minta minden értéke tényleg a $(40, 160)$ intervallumban található, de ez most szépen látszik a hisztogram x tengelyén is.

Tehát, sikeresen **eltranszformáltuk az $U(0,1)$ eloszlású 50 elemű mintánkat $U(40,160)$ eloszlásúvá.** Mindjárt látjuk, hogy az $U(0,1)$ eloszlású véletlen számokat nem csak egy másik egyenletes $U(a,b)$ eloszlásúvá, hanem tetszőleges eloszlásúvá is tudjuk tárnszformálni!!

4.1.1. A mintagenerálás általános elve

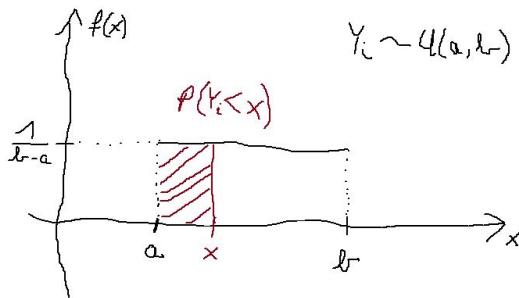
Gondolkozzunk el azon, hogyan is lehet formalizálni azt a képletet, amivel az $U(0, 1)$ eloszlású véletlen számokból $U(a, b)$ eloszlású véletlen számokat csináltunk! Ugyebár a generált számot megszoroztuk az (a, b) intervallum hosszával, és hozzáadtuk az alsó határt, a -t. Azaz formálisan azt mondhatjuk, hogy ha $y \sim U(0, 1)$, akkor $z = y \times (b - a) + a \sim U(a, b)$.

De honnan is jön ez a képlet? **Gondoljunk bele, hogy $U(a, b)$ eloszlás esetén hogyan is számolunk ki egy $P(Y_i < x)$ „alá esési” valószínűséget?** Azaz, mi a $\int_{-\infty}^x \frac{1}{b-a} dx$ improprius integrál eredménye?

Ez azért nem olyan bonyi vállalkozás, mint mondjuk egy normális eloszlás esetén kiszámolni a cuccot, hiszen mivel a alatt az $a - b$ közötti egyenletes eloszlásban 0 valószínűséggel lehetnek értékek, így az improprius integrálunk rögtön határozott integrállá válik: $\int_a^x \frac{1}{b-a} dx$. Ezzel pedig már könnyen elbánunk, hiszen az integrálandó függvényben nincs is benne a függvény változója, az x , tehát konstans függvényről van szó:

$$\int_a^x \frac{1}{b-a} dx = \left[\frac{x}{b-a} \right]_a^x = \frac{x}{b-a} - \frac{a}{b-a} = \frac{x-a}{b-a}$$

Az eredmény szemléletesen persze az alábbi téglalap területe, hiszen a két oldal hossza $oldal_1 = x - a$, illetve a sűrűségfüggvény maga $oldal_2 = f(x) = \frac{1}{b-a}$ és a téglalap területe $T = oldal_1 \times oldal_2 = (x - a) \times \frac{1}{b-a} = \frac{x-a}{b-a}$



Szóval, ha véletlenszerűen húzok egy Y_i számot az $U(a, b)$ eloszlásból, akkor annak a valószínűsége, hogy x -nél kisebb értéket kapok nem más, mint $P(Y_i < x) = \frac{x-a}{b-a}$.

Nagyon jó. Akkor, nézzük meg mi most az inverz függvény az egyenletes eloszlásban! Azaz hogyan válaszolom meg azt a kérdést mi szerint: **Mi az az érték, aminél csak 5% valószínűséggel kapok kisebb értéket egy $U(a, b)$ eloszlásban?**

Ugye ekkor az $P(Y_i < x) = \frac{x-a}{b-a}$ összefüggésből kéne kifejeznem x -et, ami aránylag könnyű művelet:

$$x = P(Y_i < x) \times (b - a) + a$$

Hoppácska! Amit kaptunk az gyakorlatilag **ugyan az a formula, amivel a** $y \sim U(0, 1)$ eloszlásból $z \sim U(a, b)$ eloszlást **csináltunk**: $z = y \times (b - a) + a$

Tehát, általánosságban azt az eredményt kapjuk, hogy ha egy $U(0, 1)$ eloszlású véletlen számra $P(Y_i < x)$ valószínűségeként tekintek, és berakom azt egy tetszőleges eloszlás inverz függvényébe, és megkeresem a hozzá tartozó x értéket, akkor amit kapok eredményül az olyan eloszlású véletlen szám lesz, mint amilyen eloszlás inverz függvényébe beírtam az $U(0, 1)$ eloszlású véletlen számot!! Ugyebár simán tekinthetek egy 0 – 1 közötti véletlenszámra $P(Y_i < x)$ valószínűségeként, hiszen minden valószínűség egy 0 – 1 közötti szám. :)

Nézzük is meg az elvet a gyakorlatban! Ugye a `scipy` csomagban minden eloszlás inverz függvényét a `ppf` rövidítésű függvényel számoltuk ki (lásd 3.2.5. fejezetben).

```

eloszlás_alsó_határ = 40
eloszlás_felső_határ = 160
eloszlás_terjedelem = eloszlás_felső_határ - eloszlás_alsó_határ

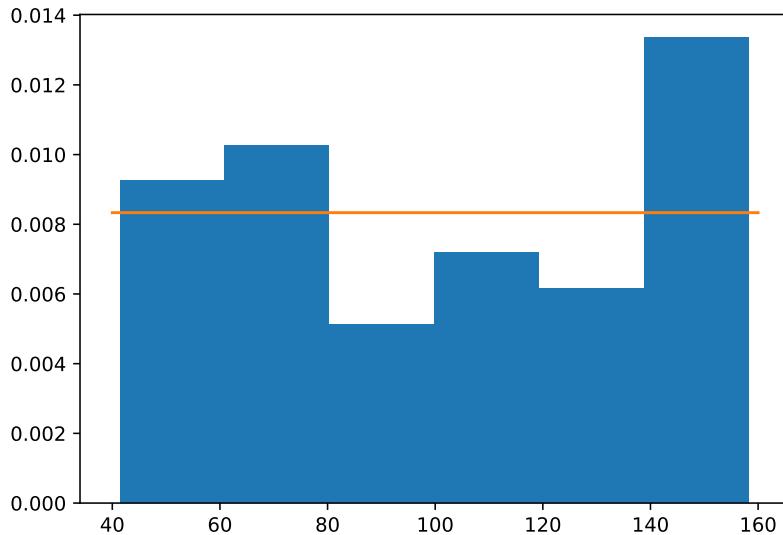
# Leürítjük a korábbi véletlenszámaink listáját és új üres listát hozunk létre
SzerencsétlenVéletlenek = []

# 50 db U(0,1) eloszlású szám generálása
for index in range(50):
    SzerencsétlenVéletlenek.append(random.random()) # véletlen szám hozzáadása a listához

# U(40,160)-á transzformálás inverz függvénnyel
SzerencsétlenVéletlenek = stats.uniform.ppf(
    SzerencsétlenVéletlenek,
    loc = eloszlás_alsó_határ,
    scale = eloszlás_terjedelem)

# Eredmény ellenőrzése hisztogramon
Hisztogram = plt.hist(SzerencsétlenVéletlenek, bins = 6, density = True)
x_tengely = np.arange(eloszlás_alsó_határ, eloszlás_felső_határ, 0.01)
y_tengely = stats.uniform.pdf(x = x_tengely, loc = eloszlás_alsó_határ, scale = eloszlás_terjedelem)
plt.plot(x_tengely, y_tengely)
plt.show()

```



E voilá: siker! Az értékek 40 – 160 között mozognak, és a generált minta gyakoriságai kb. az $U(40, 160)$ eloszlás $f(x)$ sűrűségfüggvénye körül ingadoznak!

4.1.2. Nem egyenletes eloszlású minták generálása

Na, akkor abban bízunk, hogy az egyenletes eloszlás esetében megfigyelt trükk működni fog más eloszlásokra is.

Tehát, ha egy $U(0, 1)$ eloszlású véletlen számot berakom egy tetszőleges eloszlás inverz függvényébe, akkor amit eredményül kapok az olyan eloszlású véletlen szám, mint amilyen eloszlás inverz függvényébe beírtam az kezdeti $U(0, 1)$ eloszlású véletlen számot!!

Generálunk akkor mondjuk először egy $N(80, 20)$ eloszlású 50 elemű mintát! Tehát, a normális eloszlásunk átlaga $\mu = 80$ és szórása $\sigma = 20$. A kezdeti $U(0, 1)$ számok transzformálására pedig használhatjuk akkor a **scipy** csomag **norm.ppf** függvényét. Figyeljünk, hogy a sűrűségfüggvény rajzolásakor az x tengely alsó-felső határait már a generált adatokból szedjük, mert az elvi eloszlás alapján nem lehet itt megmondani! \rightarrow Nem $U(a, b)$ egyenletes eloszlásunk van már úgymond. :)

```
átlag = 80
szórás = 20
```

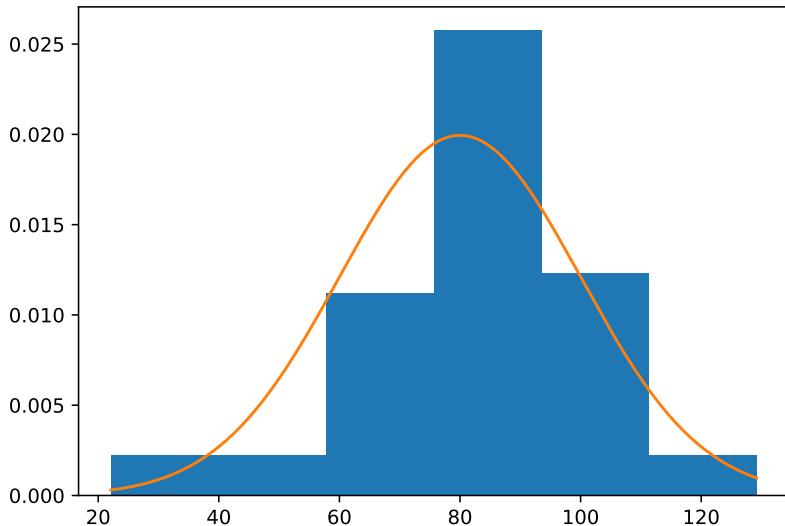
```
# Leürítjük a korábbi véletlenszámaink listáját és új üres listát hozunk létre
```

```
SzerencsétlenVéletlenek = []

# 50 db  $U(0,1)$  eloszlású szám generálása
for index in range(50):
    SzerencsétlenVéletlenek.append(random.random()) # véletlen szám hozzáadása a listához

#  $N(80,20)$ -á transzformálás inverz függvényel
Normika = stats.norm.ppf(
    SzerencsétlenVéletlenek,
    loc = átlag,
    scale = szórás)

# Eredmény ellenőrzése hisztogramon
Hisztogram = plt.hist(Normika, bins = 6, density = True)
x_tengely = np.arange(np.min(Normika), np.max(Normika), 0.01)
y_tengely = stats.norm.pdf(x = x_tengely, loc = átlag, scale = szórás)
plt.plot(x_tengely, y_tengely)
plt.show()
```



Na, egészen jól illeszkednek a generált adatok gyakoriságai a normális eloszlás sűrűségfüggvényéhez! :)

Természetesen, expoenciális eloszlásra is hasonlóan működik a trükk, de azt már nem nagy *copy-paste mágia* lenne lekódolni. :)

A lényeg viszont, hogy ez a megoldás tényleg **minden létező eloszlásra**

működik! Olyanokra is, amik nincsenek benne a `scipy`-ban!! Csak **tudni** kell az eloszlás **inverz** függvényének képletét, és abba beírogatva $U(0, 1)$ eloszlású számokat, le is generáltunk egy mintát az eloszlásból!

A jó hír viszont, hogy a `scipy`-ban szereplő eloszlásoknak van egy `rvs` általánosított függvénye, ami **automatikusan eljátsza a fenti trükköt, így nem kell** az eddig használt hosszabb kód részletet **mindig copy-pastálni**. A függvényben az aktuális eloszlásnak (egyenletes, normális, exponenciális vagy más) megfelelő módon kell megadni a `loc` és `scale` paramétereket, míg a generálandó számok mennyiségét (a minta elemszámát) a `size` paraméterben lehet beállítani.

Tehát, egyszerűsítve az alábbi módon is lehet pl. $U(40, 160)$ eloszlású adatokat (mintát) generálni.

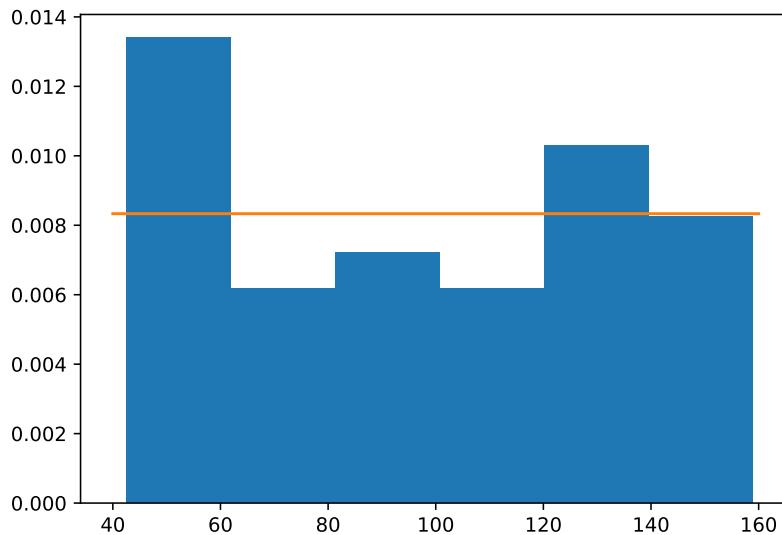
```

eloszlás_alsó_határ = 40
eloszlás_felső_határ = 160
eloszlás_terjedelem = eloszlás_felső_határ - eloszlás_alsó_határ

# U(40, 160) minta generálása
SzerencsésebbVéletlenek = stats.uniform.rvs(
    loc = eloszlás_alsó_határ,
    scale = eloszlás_terjedelem,
    size = 50)

# Eredmény ellenőrzése hisztogramon
Hisztogram = plt.hist(SzerencsésebbVéletlenek, bins = 6, density = True)
x_tengely = np.arange(eloszlás_alsó_határ, eloszlás_felső_határ, 0.01)
y_tengely = stats.uniform.pdf(x = x_tengely, loc = eloszlás_alsó_határ, scale = eloszlás_alsó_határ)
plt.plot(x_tengely, y_tengely)
plt.show()

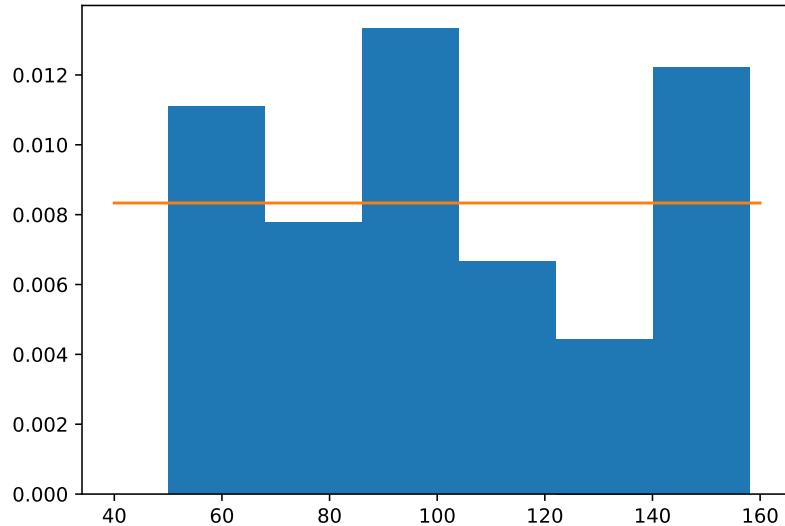
```



Ami az `rvs` függvények további nagy előnye, hogy ha kihasználjuk a függvény `random_state` paraméterét, és mindenki beírjuk oda ugyan azt a számot, pl. mondjuk az én szülinapom évét, ami 1992, akkor mindenki ugyan azt az 50 db véletlen számot generálta le! És ugyan azt a hisztogramot fogjuk már bambulni!):

```
# U(40,160) minta generálása
SzerencsésebbVéletlenek = stats.uniform.rvs(
    loc = eloszlás_alsó_határ,
    scale = eloszlás_terjedelem,
    size = 50,
    random_state = 1992)

# Eredmény ellenőrzése hisztogramon
Hisztogram = plt.hist(SzerencsésebbVéletlenek, bins = 6, density = True)
x_tengely = np.arange(eloszlás_alsó_határ, eloszlás_felső_határ, 0.01)
y_tengely = stats.uniform.pdf(x = x_tengely, loc = eloszlás_alsó_határ, scale = eloszlás_terjedelem)
plt.plot(x_tengely, y_tengely)
plt.show()
```



Pazar! :) A `random_state` paraméterben megadott 1992 értéket szokás a véletlenszám generálás **véletlen magjának** nevezni. In English *random seed*.

Generáljunk még az $U(40, 160)$ eloszlás 50 elemű mintája mellé egy 50 elemű $N(80, 20)$ és egy 50 elemű $Exp(0.0125)$ eloszlású mintát is, szintén 1992-es véletlen maggal. Majd fűzzük össze az eredményeket egy pandas data frame-be úgy, hogy a különböző eloszlású adatok adják a tábla oszlopait (sorok száma akkor így 50 lesz ugyebár). Az Exponenciális eloszlás λ paramétert a `scipy` függvények `scale` paraméterében lehet megadni a reciprokával. Hiszen az Exponenciális eloszlás szórása $\frac{1}{\lambda}$. Emlékezetetőnek lásd 3.3. fejezet!

```
# N(80, 20) minta generálása
átlag = 80
szórás = 20

NormalisAdatok = stats.norm.rvs(
    loc = átlag,
    scale = szórás,
    size = 50,
    random_state = 1992)

# Exp(0.0125) minta generálása
lam = 0.0125

ExpiAdatok = stats.expon.rvs(
    scale = (1/lam),
```

```

size = 50,
random_state = 1992)

# Eredmények összefűzése data frame-be
import pandas as pd

AdatokEgyben = pd.DataFrame(
    list(zip(SzerencsésebbVéletlenek, NormalisAdatok, ExpiAdatok)),
    columns=['Egyenletes', 'Normal', 'Expon'])
AdatokEgyben.info()

```

```

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 50 entries, 0 to 49
## Data columns (total 3 columns):
## #   Column      Non-Null Count  Dtype  
## ---  -----      -----          Dtype  
## 0   Egyenletes  50 non-null    float64
## 1   Normal      50 non-null    float64
## 2   Expon       50 non-null    float64
## dtypes: float64(3)
## memory usage: 1.3 KB

```

Szuper, az `info` metódus alapján megvan egyben a 3 db 50 elemű mintánk! Egy-egy $k = 6$ osztályközzel operáló hisztogramon meg is tudjuk nézni az eredményt, hála a data frame `hist` metódusának. :)

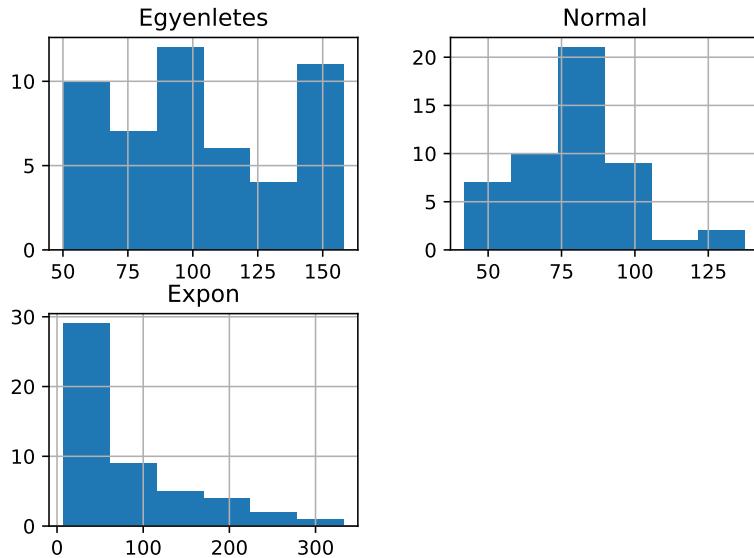
```
AdatokEgyben.hist(bins = 6)
```

```

## array([[<Axes: title={'center': 'Egyenletes'}>,
##        <Axes: title={'center': 'Normal'}>],
##        [<Axes: title={'center': 'Expon'}>, <Axes: >]], dtype=object)

plt.show()

```



Na, meg is vagyunk, nagyából mindegyik hisztogram követi a neki megfelelő Egyenletes, Normális és Exponenciális sűrűségfüggvény alakot.

4.2. Elvi Eloszlás vs Megfigyelt Minta

Azt már több soron megállapítottuk, hogy a **megfigyelt adatokból készített hisztogramok NEM követik hajszálponosan a hozzájuk tartozó eloszlások sűrűségfüggvényeit**. Valamennyire eltérnek tőle, ezt az előző fejezet végén álló 3 hisztogram is szemlélteti. Ez amúg egy **teljesen logikus és természetes jelenség**, hiszen az általunk vizsgált eloszlások $f(x)$ sűrűségfüggvényei rengeteg számhoz rendelnek pozitív bekövetkezési valószínűséget egy véletlen húzás esetén:

- Az $N(80, 20)$ eloszlás konkrétan *minden valós számnak pozitív bekövetkezési esélyt tulajdonít*
- Az $Exp(0.0125)$ eloszlás *minden pozitív valós számnak*
- Az $U(40, 160)$ pedig *minden 40 és 160 közötti valós számnak*

Mindegyik számhalmaz elemszáma **végtelen**. Tehát, **akármelyik eloszlást is nézzük**, ahhoz hogy a sűrűségfüggvényt teljes egészében meg tudjuk figyelni, végtelen sok elemű mintákat kéne generálni. Nyilván, ez nem nagyon fog sosem összejönni. Ezért vannak a **némileg hektikus hisztogram alakok**. :)

Ami csak azért zavaró, mert a **valóságban** nem vagyunk istenségek, így a megfigyelt mintánk mögött meghúzódó **valódi sűrűségfüggvényt SOSEM tudjuk megfigyelni**, csak azokat az **adatokat**, amiket a mintánk **tartalmaz**. A minta elemszáma pedig sosem végtelen. Legfeljebb csak reménykedni tudunk abban, hogy a minta hisztogram alapján sac/kb be tudjuk lőni, hogy az adataink mögött milyen eloszlás lappang a háttérben. Ugyan ez igaz a különféle statisztikai mutatóra is! Értelemszerűen más értékeket kapunk az átlagra, szórásra, mediánra vagy az x -nél kisebb elemek arányára, ha azokat a megfigyelt mintaadatokból számítjuk, és nem az adatok mögöt rejző sűrűségfüggvény alapján!

4.2.1. Az elvi eloszlás alapján számolt statisztikai mutatók

Vegyük csak végig, hogy a **3 vizsgált eloszlásunkban** hogyan is kapjuk meg a **sűrűségfüggvény ismeretében az alap statisztikai mutatókat**.

- $N(80, 20)$ eloszlás
 - Átlag: $\mu = 80$
 - Szórás: $\sigma = 20$
 - Medián: inverz függvény (ppf) értéke 0.5-nél
 - 100-nál kisebb értékek aránya: $P(Y_i < 100) = \int_{-\infty}^{100} f(x)dx$
- $Exp(0.0125)$ eloszlás
 - Átlag: $\mu = \frac{1}{\lambda} = \frac{1}{0.0125} = 80$
 - Szórás: $\sigma = \sqrt{\frac{1}{\lambda}} = \sqrt{\frac{1}{0.0125}} = 80$
 - Medián: inverz függvény (ppf) értéke 0.5-nél
 - 100-nál kisebb értékek aránya: $P(Y_i < 100) = \int_{-\infty}^{100} f(x)dx$
- $U(40, 160)$ eloszlás
 - Átlag: $\mu = \frac{a+b}{2} = \frac{40+160}{2} = 100$
 - Szórás: $\sigma = \sqrt{\frac{(b-a)^2}{12}} = \sqrt{\frac{(160-40)^2}{12}} = 34.64$
 - Medián: inverz függvény (ppf) értéke 0.5-nél
 - 100-nál kisebb értékek aránya: $P(Y_i < 100) = \int_{-\infty}^{100} f(x)dx$

Ezeket **számoljuk is ki minden eloszlásra** egy-egy külön list-be, majd fűzzük őket össze egy data frame-be! A data frame sorindexeit pedig nevezzük el a számított mutatók neve alapján!

```
# N(80,20) paraméterek
átlag = 80
szórás = 20
```

```

# Exp(0.0125) paraméterek
lam = 0.0125

# U(40,160) paraméterek
eloszlás_alsó_határ = 40
eloszlás_felső_határ = 160
eloszlás_terjedelem = eloszlás_felső_határ - eloszlás_alsó_határ

NormalisStat = [
    átlag, # Átlag
    szórás, # Szórás
    stats.norm.ppf(0.5, loc=átlag, scale=szórás), # Medián
    stats.norm.cdf(100, loc=átlag, scale=szórás)] # P(Y_i<100)

ExponStat = [
    (1/lam), # Átlag
    (1/lam), # Szórás
    stats.expon.ppf(0.5, scale=(1/lam)), # Medián
    stats.expon.cdf(100, scale=(1/lam))] # P(Y_i<100)

EgyenletesStat = [
    ((eloszlás_alsó_határ + eloszlás_felső_határ)/2), # Átlag
    ((eloszlás_felső_határ-eloszlás_alsó_határ)/np.sqrt(12)), # Szórás
    stats.uniform.ppf(0.5, loc=eloszlás_alsó_határ, scale=eloszlás_terjedelem), # Medián
    stats.uniform.cdf(100, loc=eloszlás_alsó_határ, scale=eloszlás_terjedelem)] # P(Y_i<100)

ElviStatMutatok = pd.DataFrame(
    list(zip(EgyenletesStat, NormalisStat, ExponStat)),
    columns=['Egyenletes', 'Normal', 'Expon'],
    index = ['Atlag', 'Szoras', 'Median', 'P(Y_i<100)']
)

```

ElviStatMutatok

	Egyenletes	Normal	Expon
## Atlag	100.000000	80.000000	80.000000
## Szoras	34.641016	20.000000	80.000000
## Median	100.000000	80.000000	55.451774
## P(Y_i<100)	0.500000	0.841345	0.713495

Meg is vagyunk!

4.2.2. A megfigyelt minta alapján számolt statisztikai mutatók

Most pedig tegyük az elvi statisztikai mutatók értéke mellé az 50 elemű mintákból számolt verziókat! Hasonlóan összerakhatjuk őket egy data frame-be, mint fentebb csináltuk az elvi értékekkel.

A 100-nál kisebb értékek arányának (relatív gyakoriságának) számolásához két technikai megjegyzés:

- A számolásnál azt a trükköt süjtük el, hogy pl. az `AdatokEgyben['Normal'] < 100` parancs egy `bool` tömböt ad vissza aszerint, hogy az `AdatokEgyben['Normal'] < 100` logikai állítás kire `True` és `False`.
- A `numpy` csomag `sum` függvénye pedig egy ilyen tömbre `True = 1` és `False = 0` kódolásokkal végzi el az összegzést: azaz megadja a *kedvező esetek*, a 100-nál kisebb értékek *darabszámát*.

```
NormalisMintaStat = [
    np.mean(AdatokEgyben['Normal']), # Átlag
    np.std(AdatokEgyben['Normal']), # Szórás
    np.median(AdatokEgyben['Normal']), # Medián
    np.sum(AdatokEgyben['Normal'] < 100)/len(AdatokEgyben)] # P(Y_i<100), most relatív gyak.

ExponMintaStat = [
    np.mean(AdatokEgyben['Expon']), # Átlag
    np.std(AdatokEgyben['Expon']), # Szórás
    np.median(AdatokEgyben['Expon']), # Medián
    np.sum(AdatokEgyben['Expon'] < 100)/len(AdatokEgyben)] # P(Y_i<100), most relatív gyak.

EgyenletesMintaStat = [
    np.mean(AdatokEgyben['Egyenletes']), # Átlag
    np.std(AdatokEgyben['Egyenletes']), # Szórás
    np.median(AdatokEgyben['Egyenletes']), # Medián
    np.sum(AdatokEgyben['Egyenletes'] < 100)/len(AdatokEgyben)] # P(Y_i<100), most relatív gyak.

MintaStatMutatok = pd.DataFrame(
    list(zip(EgyenletesMintaStat, NormalisMintaStat, ExponMintaStat)),
    columns=['Egyenletes', 'Normal', 'Expon'],
    index = ['Atlag', 'Szoras', 'Median', 'P(Y_i<100)']
)

MintaStatMutatok
```

	Egyenletes	Normal	Expon
## Atlag	101.865225	78.951917	82.311381

```
## Szoras      33.462534 18.922748 76.278763
## Median     97.446914 76.384652 52.148485
## P(Y_i<100)  0.540000  0.880000  0.700000
```

Meg is vagyunk!

4.2.3. A mintavételi hiba (MVH) koncepciója

Nézzük össze akkor kétféle stat. mutatókat tartalmazó data frame-et egymással!

ElviStatMutatok

```
##           Egyenletes   Normal   Expon
## Atlag      100.000000 80.000000 80.000000
## Szoras    34.641016 20.000000 80.000000
## Median     100.000000 80.000000 55.451774
## P(Y_i<100)  0.500000  0.841345  0.713495
```

MintaStatMutatok

```
##           Egyenletes   Normal   Expon
## Atlag      101.865225 78.951917 82.311381
## Szoras    33.462534 18.922748 76.278763
## Median     97.446914 76.384652 52.148485
## P(Y_i<100)  0.540000  0.880000  0.700000
```

Láthatjuk, hogy a statisztikai mutatók terén is azt tapasztaljuk, amit a hisztogramok és a sűrűségfüggvény esetén. Az elvi értékekhez az 50 elemű mintából számolt mutatók közel vannak, de nem egyeznek velük.

A kétféle értékek (elvi és mintából számolt) közti eltérést hívjuk **MINTAVÉTELI HIBÁNAK** (MVH). Mivel a gyakorlatban csak egy adott elemszámú mintát tudunk megfigyelni, és abból kellene rájönnünk a vizsgált mutatók valódi értékére, a hisztogramból meg a valódi eloszlásra, nagyon jó lenne ezt az MVH-t megmérni/kiszámolni/megbecsülni! Na, ez a statisztikai becsléselmélet alapfeladata!

Az MVH kiszámolásához hosszú az út, de az MVH viselekdését megérthetjük, ha generálunk még pár extra mintát az eddig vizsgált eloszlásokból, különböző elemszámokkal.

Ehhez írunk egy Python függvényt, ami az eddig elvégzett műveleteinket egy függvényben valósítja meg tetszőleges mintaelemszám és véletlenmag mellett:

- $N(80, 20)$, $Exp(0.0125)$ és $U(40, 160)$ eloszlású minták generálása
- A mintákból a vizsgált 4 statisztikai mutató számítása és azok összefűzése data frame-be

Ez a függvény némi copy-paste után az alábbi formát ölti.

```
def MintaGeneralas(Elemszam, VeletlenMag):
    # U(40, 160) minta generálása
    eloszlás_alsó_határ = 40
    eloszlás_terjedelem = 160-40

    SzerencsésebbVéletlenek = stats.uniform.rvs(
        loc = eloszlás_alsó_határ,
        scale = eloszlás_terjedelem,
        size = Elemszam,
        random_state = VeletlenMag)

    # N(80,20) minta generálása
    átlag = 80
    szórás = 20

    NormalisAdatok = stats.norm.rvs(
        loc = átlag,
        scale = szórás,
        size = Elemszam,
        random_state = VeletlenMag)

    # Exp(0.0125) minta generálása
    lam = 0.0125

    ExpiAdatok = stats.expon.rvs(
        scale = (1/lam),
        size = Elemszam,
        random_state = VeletlenMag)

    # Eredmények összefűzése data frame-be
    AdatokEgyben = pd.DataFrame(
        list(zip(SzerencsésebbVéletlenek, NormalisAdatok, ExpiAdatok)),
        columns=['Egyenletes', 'Normal', 'Expon'])

    NormalisMintaStat = [
        np.mean(AdatokEgyben['Normal']), # Átlag
        np.std(AdatokEgyben['Normal']), # Szórás
        np.median(AdatokEgyben['Normal']), # Medián
        np.sum(AdatokEgyben['Normal'] < 100)/len(AdatokEgyben)] # P(Y_i<100), most relatív gyak.
```

```

ExponMintaStat = [
    np.mean(AdatokEgyben['Expon']), # Átlag
    np.std(AdatokEgyben['Expon']), # Szórás
    np.median(AdatokEgyben['Expon']), # Medián
    np.sum(AdatokEgyben['Expon'] < 100)/len(AdatokEgyben)] # P(Y_i<100), most relatívi százalék

EgyenletesMintaStat = [
    np.mean(AdatokEgyben['Egyenletes']), # Átlag
    np.std(AdatokEgyben['Egyenletes']), # Szórás
    np.median(AdatokEgyben['Egyenletes']), # Medián
    np.sum(AdatokEgyben['Egyenletes'] < 100)/len(AdatokEgyben)] # P(Y_i<100), most relatívi százalék

MintaStatMutatok = pd.DataFrame(
    list(zip(EgyenletesMintaStat, NormalisMintaStat, ExponMintaStat)),
    columns=['Egyenletes', 'Normal', 'Expon'],
    index = ['Atlag', 'Szoras', 'Median', 'P(Y_i<100)']
)

return MintaStatMutatok

```

Ha a `MintaGeneralas` függvény definícióját megadó fenti kód szépen lefutott, akkor a következőképpen tudjuk használni a függvényt egy 50 elemű minta generálására 1994-es véletlenmag mellett.

```
MintaGeneralas(50, 1994)
```

	Egyenletes	Normal	Expon
## Atlag	104.663427	81.542000	86.208355
## Szoras	33.384088	23.684983	74.027139
## Median	107.564478	83.357606	66.232602
## P(Y_i<100)	0.400000	0.800000	0.720000

Na, úgy néz ki működik! :)

Akkor most lessük meg **mi** történik a mintából számolt és az elvi statisztikai mutatók közti különbségekkel (alias az MVH-val), ha nem 50, hanem mondjuk 1000 elemű mintákat generálok az eloszlásokból.

```
ElviStatMutatok
```

	Egyenletes	Normal	Expon
## Atlag	100.000000	80.000000	80.000000
## Szoras	34.641016	20.000000	80.000000
## Median	100.000000	80.000000	55.451774
## P(Y_i<100)	0.500000	0.841345	0.713495

```
MintaGeneralas(50, 1992)
```

```
##              Egyenletes      Normal      Expon
## Atlag       101.865225   78.951917   82.311381
## Szoras     33.462534   18.922748   76.278763
## Median     97.446914   76.384652   52.148485
## P(Y_i<100)  0.540000   0.880000   0.700000
```

```
MintaGeneralas(1000, 1992)
```

```
##              Egyenletes      Normal      Expon
## Atlag       100.646326   80.106832   81.413684
## Szoras     34.791814   20.003538   80.432945
## Median     100.488619   79.798321   56.105934
## P(Y_i<100)  0.499000   0.838000   0.706000
```

Na, hát ha 1000 elemet tudtam megfigyelni az 50 helyett, akkor minden statisztikai mutatóm mintából számolt értéke lényegesen közelebb van a valós, elvi értékhez! Tehát, olybá tűnik, hogy **mintaelemszám növelésével az MVH lecsökken!** Ez egy nagyon fontos tulajdonság, ami **minden statisztikai mutató esetén általánosságban igaz marad!** Mivel ez egy marha fontos dolog, így kaptok róla egy mémet is! **Égjen csak ez be az agyakba! ;)**



További kísérletezgetésre a mintavétel hibával kapcsolatban, nyugodtan lehet használni az alábbi kis interaktív felületet.

A felületet nyomogatva érdemes megfigyelni pár dolgot.

- A mintaelemszám növelésével a hiszogram alakja is közelebb kerül az adatok mögött álló eloszlás sűrűségfüggvényéhez. Tehát **elemszám növelésével a sűrűségfüggvény mintavételi hibája is csökken** úgymond.
- Az **átlag és a 100-nál kisebb értékek aránya általában mindhárom eloszlásnál elég közel van a valós, elméleti értékhez.**

- Ellenben a **medián** és **szórás** értékek az **exponenciális eloszlás** esetében aránylag többször is durván eltérnek a valós értéktől még nagyobb (pl. 500 körüli) **elemszám** esetén is.

4.3. Statisztikai sokaságok FAE mintavételezése

Amit az első két fejezetben műveltünk az az **elméleti eloszlások Független Azonos Eloszlású**, leánykori nevén **FAE** mintavételezése.

A FAE mintának ezek szerint két jellemzője van:

- **Független:** a **mintaelemek kihúzása véletlenszerű**, tehát az, hogy mit húztam mondjuk 3-nak **nem függ** attól mit húztam 1-nek és 2-nak.
- **Azonos Eloszlású:** A minta minden eleme **ugyan abból az eloszlásból** (pl. $N(80, 20)$ vagy $Exp(0.0125)$) származik, nincs közben váltás arra nézve, hogy épp milyen eloszlásból húzzuk ki véletlenszerűen az értékeket.

Ezt a FAE mintavételt **valós statisztikai adatsorok, azaz sokaságok esetén is el lehet végezni**, nem csak elméleti eloszlások esetén. Mondjuk meg akarom ismerni a kb. 4,5 millió magyar munkavállaló havi bruttó átlagkeresetét. Nyilván a NAV-nál megvan mind a 4,5 millió dolgos magyar *en_ber* kereseti adata tételesen (tehát ezek az értékek nem egy exponenciális eloszlással vannak megadva pl., hanem tételesen fel vannak sorolva egy táblában valahol), csak hát tudjuk mikor adják ki ezeket az adatokat bárkinek is. Ezért inkább azt csináljuk, hogy mondjuk a népességnyilvántartótól elkérjük a 4,5 millió magyar munkavállaló lakcímét, beöntjük a címeket tartalmazó céltíket egy kalapba, és becsukott szemmel, random húzunk 100 címet *visszatevéssel*, azaz egy kihúzott címet visszateszünk minden a kalapba, és nem rakjuk félre. Majd minden a 100 címre elmegyünk és megkérdezzük az emberek havi bruttó keresetét. Ezzel a **véletlen, visszatevéses mintavételel a magyar munkavállalók sokaságából** valójában **FAE** mintavételezést végeztünk. Miért is?

1. A **mintaelemek függetlenek**, hiszen „*becsukott szemmel*”, véletlenszerűen húztuk ki a minta elemeket.
2. Mivel **visszatevéses volt a mintavétel, így a keresetek eloszlása nem változott meg a mintavétel során**. Tehát végig ugyan azt az elméleti kereseteloszlást mintavételeztünk. Pont úgy, mint amikor minden ugyan abból az elvi sűrűségfüggvényből generáltuk a mintaelemeket az eddig elvégzett mintavételeink során.

Tehát, bízunk abban, hogy ami mondjuk a **mintából számolt átlagkereset és a teljes sokaság átlagekeresete között áll az ugyan az az MVH, mint amivel találkoztunk az eloszlások elvi átlaga és a belőlük**

generált minták tapasztalati átlaga esetében is!! Persze a gyakorlatban nem minden mintavétel végezni, mert

1. Nem tudunk visszatenni elemeket. Ez a **kisebbik gond**. Hiszen, ha a minta mérete a sokaság méretéhez képest nagyon kicsi (pl. 100 főt választunk kis a 4,5 millióból és nem 2 milliót :)), akkor a **visszatevéses és visszatevés nélküli mintavétel nagyjából ugyan az**, mert nagyon kicsi lesz az esélye annak, hogy a visszatevéses esetben valakit tényleg kétszer beválasztunk a mintába.
2. Nem tudunk ténylegesen véletlenszerűen kiválasztani elemeket. Ez a **nagyobb baj**. És sajnos gyakori is. Pl. *nem kapunk címjegyzéket, amiből sorsolni lehet a mintánkba embereket*. Mert ekkor a mintavételi hiba mellé bejön az úgynevezett **nem mintavételi hiba**. Ezek olyan tényezők, amik „*emberi gyarlóság*” miatt kerülnek a rendszerbe. Pl. nagyon tetszenek a vörös hajú lányok, így aránytalanul sokat kérdezzek meg belőlük ahhoz képest ahányan a sokaságban vannak. Vagy, lusta vagyok lemníni 500 fő alatti településre, így az alacsonyabb keresetű társadalmi rétegből is kevesebb embert kérdezzek meg, mint ahányat kéne a sokasági elemszámuk alapján. Tehát, szakszóval azt mondom, hogy a **minta nem lesz reprezentatív hajszínre és település méretre**. Ezekkel a **nem mintavételi hibákkal most nem foglalkozunk, feltesszük, hogy nincsenek**. Mert **kiszámolásuk és korrigálásuk elég bonyolult**, komplett mesterszakok foglalkoznak a kérdéssel.

Szóval, első körben nézzük meg **hogyan tudunk Pythonban egy ismert sokaságból FAE mintát venni**. A LIDLBalaton2022.xlsx fájlban a 2022-es LIDL Balaton átúszás résztvevőinek neve, neme és percben mért időeredménye található. Olvassuk be a táblát pandas data frame-be!

```
Balcsi = pd.read_excel("LIDLBalaton2022.xlsx")
Balcsi.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 9751 entries, 0 to 9750
## Data columns (total 3 columns):
## #   Column  Non-Null Count  Dtype  
## #   --   --   --   --   --   --   --   -- 
## 0   Nev     9751 non-null   object 
## 1   Nem     9751 non-null   object 
## 2   PERC    9751 non-null   float64
## dtypes: float64(1), object(2)
## memory usage: 228.7+ KB
```

```
Balcsi.head()
```

```
##                                     Nev Nem      PERC
## 0          Aba Attila    F 142.416667
## 1        Abaffy Károly  F 197.883333
## 2        Abaffy Kornél  F 197.983333
## 3  Abelovszki Hajnalka N 182.000000
## 4     Abért Valentin ifj  F 222.516667
```

Szuper, úgy látszik megvan mind az $N = 9751$ versenyző időeredménye hiánytalanul. **Ez lesz most akkor a sokaságunk.**

Számolunk ki az időeredményekre három statisztikai mutatót:

- Az **átlagos** időt Jele: \bar{Y} vagy μ
- Az egyéni idők **szórását** Jele: σ
- A 3 óra (180 perc) felett teljesítők **arányát** Jele: P
 - Itt a számolásnál azt a trükköt süttük el, hogy a `Balcsi.PERC > 180` parancs egy `bool` tömböt ad vissza aszerint, hogy a `Balcsi.PERC > 180` logikai állítás kire `True` és `False`.
 - A `numpy` csomag `sum` függvénye pedig egy ilyen tömbre `True = 1` és `False = 0` kódolásokkal végzi el az összegzést: azaz megadja a *kedvező esetek*, a Balatont 180-an percnél hosszabb idő alatt átúszók *darabszámát*.

```
SokasagiAtlag = np.mean(Balcsi.PERC)
SokasagiSzoras = np.std(Balcsi.PERC)
SokasagiArany = np.sum(Balcsi.PERC > 180)/len(Balcsi)

SokasagiAtlag
```

```
## 167.52914060096398
```

```
SokasagiSzoras
```

```
## 44.08833385551663
```

```
SokasagiArany
```

```
## 0.3295046661880833
```

Meg is vagyunk, **ezek akkor a statisztikai mutatóink valós, sokasági értékei**:

- $\bar{Y} = \mu = 167.5$ perc
- $\sigma = 44.1$ perc
- $P = 0.3295 = 32.95\%$

Ezek alapján a **sokaság**, azaz a teljes adatsor esetén a következő jelölési konvenciókkal élünk:

- Elemszám N
- Sokaságból számított statisztikai mutatók együttes jelölése: θ
 - Tehát θ egy általános jelölés egy sokasági mutatószámra.
 - Állhat mögötte átlag ($\bar{Y} = \mu$), szórás (σ), arány (P), de akár medián (Me), módusz (Mo) stb. is!

Általános szabály, hogy a „*sokasági dolgokat*” vagy **nagy** vagy **görög betűvel** jelöljük.

Akkor most **vegyünk a Balaton 2022-es átúszóinak a sokaságából** egy $n = 100$ elemű **mintát!** Szerencsére Pythonban ezt könnyű végrehajtani, mivel a pandas data frame-nek létezik `sample` metódusa, aminek az `n` paraméterében meg lehet mondani, hogy **hány elemű mintánk** legyen. A `replace = True` paraméter beállítással pedig azt mondjuk meg gépállatnak, hogy **visszatevéses véletlen, azaz FAE mintát** szeretnénk. Itt is van `random_state` paraméter, amit azonos számla (pl. továbbra is 1992-re) beállítva *ugyan azt a véletlen mintát fogjuk kapni*.

```
BalcsiMinta = Balcsi.sample(n = 100, replace = True, random_state = 1992)
```

```
BalcsiMinta.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 100 entries, 7991 to 5727
## Data columns (total 3 columns):
## #   Column  Non-Null Count  Dtype  
## ---  --     --     --    
## #   0   Nev      100 non-null    object 
## #   1   Nem      100 non-null    object 
## #   2   PERC     100 non-null    float64
## dtypes: float64(1), object(2)
## memory usage: 3.1+ KB
```

Olybá néz ki megvan az $n = 100$ elemű FAE mintánk! :)

Mielőtt nekiugrunk, és kiszámoljuk a minta alapján az időeredmények átlagát, szórását és a 3 órán felül úszók arányát, itt is szögezzünk le **jelölési konvenciót a mintára vonatkozóan**:

- Elemszám n
- A mintából számított statisztikai mutatók együttes jelölése: $\hat{\theta}$
 - Tehát $\hat{\theta}$ egy általános jelölés egy sokasági mutatószámra.
 - Általánosságban a „*kalap*”, a $\hat{\theta}$ minden a mintából **becsült** érték jele.
 - Állhat $\hat{\theta}$ mögött átlag (\bar{y}), szórás (s^*), arány (p), de akár medián (me), módusz (mo) stb. is!

Általános szabály, hogy a „*mintából számolt dolgokat*” vagy **kis betűvel** vagy **kalappal** jelöljük.

Na, akkor nézzük meg a **statisztikai mutatóink** $n = 100$ elemű **mintából számítva/becsülve!**

```
MintaAtlag = np.mean(BalcsiMinta.PERC)
MintaSzoras = np.std(BalcsiMinta.PERC)
MintaArany = np.sum(BalcsiMinta.PERC > 180)/len(BalcsiMinta)
```

```
MintaAtlag
```

```
## 164.44033333333334
```

```
MintaSzoras
```

```
## 38.62827175499542
```

```
MintaArany
```

```
## 0.3
```

Meg is vagyunk, **ezek akkor a statisztikai mutatóink mintából becsült értékei:**

- $\bar{y} = 164.4$ perc
- $s^* = 38.6$ perc
- $p = 0.3 = 30\%$

Ezek sem egyeznek pontosan a mutatók valós, sokasági értékével, de *nem térnek el nagyon durván* a valóságtól. Most is abban bízunk, hogy a **becsült és valós értékek közti eltérés a MVH miatt van, és akkor most ráfordulunk ennek az MVH-nak a meghatározására!**

Ehhez a **0. lépés, hogy nagyon-nagyon sok, pl. 10000 db 100 elemű FAE mintát veszünk** a sokaságunkból, és mindegyik mintában kiszámoljuk az átlagot, szórást és arányt.

4.3.1. Ismételt Mintavételezés (*Resampling*)

Amit még ebben az anyagban megnézünk az az volna, **hogyan generálunk le 10000 db 100 elemű FAE mintát** a 2022-es Balaton átúszók sokaságából.

Természetesen a kódunk alapja egy 10000 iteráció végiszaladó `for` ciklus lesz. Ami némileg érdekes, hogy miképpen tároljuk el a mintavételezett adatokat. A számításokhoz igazából csak a PERC oszlopra van szükség, így egy olyan data frame-t generálunk, aminek 10000 sorában lesznek a különböző mintavételek, míg 100 oszlopában a 100 db mintaelem minden egyes mintavételezési körben.

Ehhez először létrehozunk egy üres 100 oszlopos data frame-t. Ennek a lépései:

- Létrehozzuk az oszlopneveket
 - Létrehozunk egy `numpy` tömböt, amiben 1 – 100-ig vannak az egész számok
 - Ezt az `int` tömböt `string`gé konvertáljuk
 - minden számhoz hozzáfüzzük az „`Elem`” előtagot a `numpy` csomag `core.defchararray` névterében lakó `add` függvényel
- A `pandas` csomag `DataFrame` függvényével létrehozzuk az üres 100 oszlopos data frame-t, az előző pontban megadott oszlopnevekkel.

```
oszlopnevek = (np.array(range(100))+1)
oszlopnevek = oszlopnevek.astype(str)
oszlopnevek = np.core.defchararray.add("Elem", oszlopnevek)
oszlopnevek
```

```
## array(['Elem1', 'Elem2', 'Elem3', 'Elem4', 'Elem5', 'Elem6', 'Elem7',
##        'Elem8', 'Elem9', 'Elem10', 'Elem11', 'Elem12', 'Elem13', 'Elem14',
##        'Elem15', 'Elem16', 'Elem17', 'Elem18', 'Elem19', 'Elem20',
##        'Elem21', 'Elem22', 'Elem23', 'Elem24', 'Elem25', 'Elem26',
##        'Elem27', 'Elem28', 'Elem29', 'Elem30', 'Elem31', 'Elem32',
##        'Elem33', 'Elem34', 'Elem35', 'Elem36', 'Elem37', 'Elem38',
##        'Elem39', 'Elem40', 'Elem41', 'Elem42', 'Elem43', 'Elem44',
##        'Elem45', 'Elem46', 'Elem47', 'Elem48', 'Elem49', 'Elem50',
##        'Elem51', 'Elem52', 'Elem53', 'Elem54', 'Elem55', 'Elem56',
##        'Elem57', 'Elem58', 'Elem59', 'Elem60', 'Elem61', 'Elem62',
##        'Elem63', 'Elem64', 'Elem65', 'Elem66', 'Elem67', 'Elem68',
##        'Elem69', 'Elem70', 'Elem71', 'Elem72', 'Elem73', 'Elem74',
##        'Elem75', 'Elem76', 'Elem77', 'Elem78', 'Elem79', 'Elem80',
##        'Elem81', 'Elem82', 'Elem83', 'Elem84', 'Elem85', 'Elem86',
##        'Elem87', 'Elem88', 'Elem89', 'Elem90', 'Elem91', 'Elem92',
##        'Elem93', 'Elem94', 'Elem95', 'Elem96', 'Elem97', 'Elem98',
##        'Elem99', 'Elem100'], dtype='<U15')
```

```
MintaDF = pd.DataFrame(columns = oszlopnevek)
MintaDF
```

```
## Empty DataFrame
## Columns: [Elem1, Elem2, Elem3, Elem4, Elem5, Elem6, Elem7, Elem8, Elem9, Elem10, Ele
## Index: []
##
## [0 rows x 100 columns]
```

Ha ezzel megvagyunk, akkor egyszerűen egy `for` ciklusban feltölthetjük a data frame-t sorokkal. Ugye 1 sor = 1 db 100 elemű minta. Ehhez pedig az 1. fejezet elején már használt `append` (egy új sor hozzáadása) metódusát vetjük be az alábbi kódban látható módon. Arra kell figyelni, hogy az `append` előtt a kiválasztott minta időadatait tartalmazó tömb `index`-eit a data frame-ünk *oszlopneveire* kell átrakni, különben az `append` nem tudja összekötni a mintaelemeket a cél data frame oszlopaival! Természetesen most véletlenmagot nem rögzítünk a `sample` metódusban, különben a műveletnek nem lenne értelme (10000-szer venné ki ugyan azt a mintát ugyebár).

Futtatás előtt **arra készüljünk lélekben, hogy az alábbi kódrészlet futásideje elég hosszadalmas** (több perc) is lehet, mivel marhára nem optimalizáltuk a 10000 iterációjú `for` ciklusunkat... Ejjnye! :)

```
for index in range(10000):
    AktualisMinta = Balcsi['PERC'].sample(n = 100, replace = True)
    AktualisMinta.index = oszlopnevek
    MintaDF = MintaDF.append(AktualisMinta, ignore_index = True)

MintaDF
```

```
##           Elem1      Elem2      Elem3     ...     Elem98      Elem99      Elem100
## 0    164.800000  129.066667  156.166667  ...  207.350000  159.666667  165.883333
## 1    152.516667  212.483333  152.900000  ...  307.266667  119.783333  128.216667
## 2    145.666667  185.266667  169.516667  ...  167.733333  228.366667  215.633333
## 3    185.683333  120.333333  201.250000  ...  182.766667  177.666667  112.450000
## 4    117.483333  142.350000  320.266667  ...  188.566667  189.166667   99.916667
## ...
## 9995  162.333333   83.350000  146.750000  ...  164.250000  131.933333  128.183333
## 9996  100.416667  161.433333  187.366667  ...  160.483333  168.416667  209.300000
## 9997  146.450000  160.783333  165.483333  ...  158.816667  167.733333  183.400000
## 9998  139.250000  140.466667  130.933333  ...  153.616667  112.366667  196.050000
## 9999  147.316667  173.450000  106.100000  ...  185.616667  171.800000  135.166667
##
## [10000 rows x 100 columns]
```

Ezzel meg is vagyunk, van az időeredményekból 10000 db $n = 100$ elemű mintánk! :) Csak, hogy ne kelljen ezt a hosszú mintagenerálási időt végigvárni, utolsó lélegzetünkkel **mentsük el az eredményünket tartalmazó data frame-t egy Excel fájlba**. A data frame `to_excel` metódusának `index = False` beállításával azt érjük el, hogy a sorindexeket ne írja ki egy külön oszlopba az Excelbe. Ezek most egyszerű sorszámok csak, semmi jelentőségük, felesleges az Excelben erre elpazarolni egy oszlopot. :)

```
MintaDF.to_excel('MintaDataFrame.xlsx', index = False)
```


5. fejezet

A becsléselmélet alapjai

5.1. Ismétlés: Balaton átúszás eredmények és ezek FAE mintái

Folytassuk ott a dolgainkat, ahol a 2. heti anyagban abbahagyottuk. Töltsük be egy `pandas` data frame-be a LIDLBalaton2022.xlsx fájl adatait. Ebben az Excelben a 2022-es LIDL Balaton átúszás résztvevőinek *neve*, *neme* és *percben* mért *időeredménye* található. Ez az adatsor lesz most nekünk a **sokaságunk**.

```
# Elemzéshez és ábrázoláshoz szükséges csomagok betöltése
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats

# Adatbeolvasás data frame-be
Balcsi = pd.read_excel("LIDLBalaton2022.xlsx")

Balcsi.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 9751 entries, 0 to 9750
## Data columns (total 3 columns):
## #   Column  Non-Null Count  Dtype  
## ---  --    --    --    
## #   0   Nev      9751 non-null   object 
## #   1   Nem      9751 non-null   object 
## #   2   PERC     9751 non-null   float64
## dtypes: float64(1), object(2)
```

```
## memory usage: 228.7+ KB
```

```
Balcsi.head()
```

```
##           Nev Nem      PERC
## 0        Aba Attila   F 142.416667
## 1     Abaffy Károly   F 197.883333
## 2     Abaffy Kornél   F 197.983333
## 3 Abelovszki Hajnalka   N 182.000000
## 4    Abért Valentin ifj   F 222.516667
```

Oké, meg is vagyunk! A sokságnak $N = 9751$ eleme van, tehát ennyi versenyző úszta át 2022-ben a Balatont. A sokságnak alapvetően egy ismérővel, **az időeredménnyel** (PERC oszlop) **fogunk foglalkozni**, és ennek három statisztikai mutatóját vagy szébb fogalmazva statisztikai paraméterét fogjuk megvizsgálni:

- Az **átlagos** időt Jele: \bar{Y} vagy μ
- Az egyéni idők **szórását** Jele: σ
- A 3 óra (180 perc) felett teljesítők **arányát** Jele: P

Ahogy a 4.3. fejezetben ezt tisztáztuk, egy **teljes statisztikai** adatsor, azaz **sokaság statisztikai mutatóit/paramétereit** együttesen θ -val jelöljük.

Akkor hát **számoljuk is ki ezeket** a θ -kat! Még kiszámoljuk az időeredmények varianciáját (szórásnégyzetét) is, mert erre is szükségünk lesz a későbbiekbén.

```
SokasagiAtlag = np.mean(Balcsi.PERC)
SokasagiSzoras = np.std(Balcsi.PERC)
SokasagiVariancia = SokasagiSzoras**2
SokasagiArany = np.sum(Balcsi.PERC > 180)/len(Balcsi)
```

```
SokasagiAtlag
```

```
## 167.52914060096398
```

```
SokasagiSzoras
```

```
## 44.08833385551663
```

```
SokasagiArany
```

```
## 0.3295046661880833
```

5.1. ISMÉTLÉS: BALATON ÁTÚSZÁS EREDMÉNYEK ÉS EZEK FAE MINTÁI

Meg is vagyunk! Akkor következő lépésekben vegyünk is egy $n = 100$ elemű **FAE** (tehát visszatevéses véletlen) mintát ebből a sokaságból egy 1992-es véletlen mag mellett. Majd számoljuk is ki a három vizsgált statisztikai paraméter értékét a mintában.

```
BalcsiMinta = Balcsi.sample(n = 100, replace = True, random_state = 1992)

BalcsiMinta.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 100 entries, 7991 to 5727
## Data columns (total 3 columns):
## #   Column Non-Null Count Dtype  
## ---  -----  --------------  object 
## 0   Nev     100 non-null    object 
## 1   Nem     100 non-null    object 
## 2   PERC    100 non-null    float64
## dtypes: float64(1), object(2)
## memory usage: 3.1+ KB
```

```
MintaAtlag = np.mean(BalcsiMinta.PERC)
MintaSzoras = np.std(BalcsiMinta.PERC)
MintaArany = np.sum(BalcsiMinta.PERC > 180)/len(BalcsiMinta)
```

```
MintaAtlag
```

```
## 164.4403333333334
```

```
MintaSzoras
```

```
## 38.62827175499542
```

```
MintaArany
```

```
## 0.3
```

Ezek szerint összefoglalva statisztikai **paramétereink** mintából becsült értékei:

- $\bar{y} = 164.4$ perc
- $s^* = 38.6$ perc
- $p = 0.3 = 30\%$

Szintén a 4.3. fejezetben szerepelt, hogy a **statisztikai paraméterek mintából becsült értékeit együttesen $\hat{\theta}$ -val jelöljük**. A $\hat{\theta}$ -ok speciális elnevezése: **becslőfüggvény** a *valódi, sokasági θ* -hoz. Szóval, a mintaátlag (\bar{y}) a sokasági átlag $\bar{Y} = \mu$ **becslőfüggvénye**, a mintából számolt szórás (s^*) a valós, sokasági szórás (σ) **becslőfüggvénye** és a mintából számított 3 órán túli úszók aránya (p), a sokasági arány (P) **becslőfüggvénye**.

A feladatunk pedig az lenne, hogy rájöjjünk: **Hogyan tudunk egyetlen egy db n elemű minta $\hat{\theta}$ becslőfüggvény értékeiből következtetni a teljes N elemű sokaság valós θ paraméter értékeire?** Ez a **statisztikai becsléselmélet alapfeladata**. Azt is körbejártuk a 4.3. fejezetben, hogy ha a mintavételünk tényleg rendes FAE mintavétel, akkor ami $\hat{\theta}$ és a valós θ között áll az nem más, mint a **mintavételi hiba (MVH)**. A feladatunk tehát konkrétabban nézve az **MVH kiszámítása vagy legalábbis valamilyen közelítése**.

Ahhoz, hogy **elinduljunk az MVH számítás rögös útján** egy olyan trükkkel élünk, ami kihasználja, hogy most éppen ismerjük a teljes Balatont átúszó sokaságot. Nyilván a gyakorlatban azért kell becsléselmélettel meg MVH számítással foglalkozni, mert a sokaságot nem tudjuk megismerni. :) De most mivel megvannak a sokasági adatok, így **kivehetünk a sokasági időeredményekből nagyon-nagyon sok, mondjuk 10000 db $n = 100$ elemű mintát**. Ezt meg is tettük a 4.3. fejezet legvégén. Most csak visszatöljtük az eredményt tartalmazó Excel táblát **egy data frame-be**. Az Excel fájl, amit ebben a jegyzetben használok innen elérhető.

```
MintaVetelek100Elem = pd.read_excel("MintaDataFrame.xlsx")
MintaVetelek100Elem
```

```
##           Elem1      Elem2      Elem3     ...      Elem98      Elem99      Elem100
## 0    164.800000  129.066667  156.166667  ...  207.350000  159.666667  165.883333
## 1    152.516667  212.483333  152.900000  ...  307.266667  119.783333  128.216667
## 2    145.666667  185.266667  169.516667  ...  167.733333  228.366667  215.633333
## 3    185.683333  120.333333  201.250000  ...  182.766667  177.666667  112.450000
## 4    117.483333  142.350000  320.266667  ...  188.566667  189.166667  99.916667
## ...
## 9995  162.333333  83.350000  146.750000  ...  164.250000  131.933333  128.183333
## 9996  100.416667  161.433333  187.366667  ...  160.483333  168.416667  209.300000
## 9997  146.450000  160.783333  165.483333  ...  158.816667  167.733333  183.400000
## 9998  139.250000  140.466667  130.933333  ...  153.616667  112.366667  196.050000
## 9999  147.316667  173.450000  106.100000  ...  185.616667  171.800000  135.166667
##
## [10000 rows x 100 columns]
```

Oké, az eredményből látjuk is, hogy úgy néz ki a data frame, hogy **1 sor**

tartalmaz 1 db 100 elemű mintát és a mintaelémeket (tehát a mintába besorolt versenyző percben mért időeredményét) **az oszlopban tároljuk.**

Ez a tárolási forma azért is kényelmes, mert a data frame „*szeletelésével*” bármikor tudunk $n < 100$ elemű mintákat is előállítani. Hiszen a mintavétel a pitonkának köszönhetően teljesen véletlenszerű volt, így **ha kiválasztom a tábla első 20 oszlopát az olyan, mintha lenne 10000 db $n = 20$ elemű mintám is!** Ezt most tegyük is akkor meg!

```
MintaVetelek20Elem = MintaVetelek100Elem.iloc[:, 0:20]
MintaVetelek20Elem
```

```
##           Elem1        Elem2        Elem3      ...       Elem18       Elem19       Elem20
## 0    164.800000  129.066667  156.166667  ...  136.066667  115.616667  174.966667
## 1    152.516667  212.483333  152.900000  ...  150.933333  172.316667  218.083333
## 2    145.666667  185.266667  169.516667  ...  140.300000  121.900000  195.650000
## 3    185.683333  120.333333  201.250000  ...  147.950000  264.466667  137.033333
## 4    117.483333  142.350000  320.266667  ...  150.500000  245.600000  178.166667
## ...
## 9995   162.333333   83.350000  146.750000  ...  178.166667  143.166667  146.433333
## 9996   100.416667  161.433333  187.366667  ...  139.366667  166.483333  129.966667
## 9997   146.450000  160.783333  165.483333  ...  113.033333  195.250000  167.200000
## 9998   139.250000  140.466667  130.933333  ...  182.200000  148.633333  121.183333
## 9999   147.316667  173.450000  106.100000  ...  217.333333  198.366667  166.133333
##
## [10000 rows x 20 columns]
```

Szuper, olybá tűnik akkor jók vagyunk! :)

5.2. A torzítatlanság fogalma

Na hát akkor vizsgáljuk meg először a 10000 db $n = 100$ elemű mintát alaposabban ahhoz, hogy megértsük: **hogyan is viselkednek a $\hat{\theta}$ becslőfüggvények a valós θ paraméterekhez képest.**

Először **számoljuk ki mindegyik 100 elemű mintában a három statisztikai mutatónk, azaz becslőfüggvényünk értékét:** átlag, szórásnyagyezet (variancia) és a 180 percen felül teljesítők aránya. A szórás kapcsán kényelmesebb lesz a gyökjel nélkül vizsgálni a dolgokat, ezért veszünk varianciát. Szerencsénkre, a numpy csomag statisztikai függvényei `axis = 1` paraméter beállítással soronként és NEM oszloponként számolják ki az átlagot, varianciát, összegeket, így **minden mintára ki tudjuk számolni a becslőfüggvények értékét 1-1 új oszlopból.** Figyeljünk arra, hogy mivel a data frame oszlopai folyamatosan bővülnek, így manuálisan le kell szorítani

a numpy statisztikai függvények alkalmazását minden az első 100 oszlopra az iloc metódussal!

```
MintaVetelek100Elem['Atlagok'] = np.mean(MintaVetelek100Elem.iloc[:, 0:100], axis=1)
MintaVetelek100Elem['Varianciak'] = np.std(MintaVetelek100Elem.iloc[:, 0:100], axis=1)*
MintaVetelek100Elem['Aranyok'] = np.sum(MintaVetelek100Elem.iloc[:, 0:100] > 180, axis=1)

MintaVetelek100Elem

##           Elem1      Elem2      Elem3 ...     Atlagok  Varianciak Aranyok
## 0    164.800000  129.066667  156.166667 ...  171.261667  1779.537092   0.34
## 1    152.516667  212.483333  152.900000 ...  158.026500  1941.843989   0.29
## 2    145.666667  185.266667  169.516667 ...  172.278833  1987.792994   0.34
## 3    185.683333  120.333333  201.250000 ...  163.434167  2045.349935   0.33
## 4    117.483333  142.350000  320.266667 ...  166.552833  2077.951717   0.33
## ...
## 9995  162.333333  83.350000  146.750000 ...  163.682333  1687.185827   0.28
## 9996  100.416667  161.433333  187.366667 ...  164.332833  1748.105975   0.32
## 9997  146.450000  160.783333  165.483333 ...  163.702333  1425.601189   0.30
## 9998  139.250000  140.466667  130.933333 ...  166.423833  1564.257813   0.37
## 9999  147.316667  173.450000  106.100000 ...  173.050333  2302.607722   0.44
##
## [10000 rows x 103 columns]
```

Oké, olybá tűnik, hogy minden a 10000 db mintára megvan minden három becslőfüggvény a MintaVetelek100Elem data frame utolsó három oszlopában.

Következő lépésként vegyük a mintaátlagos és a mintaarányok átlagát, és vessük össze az eredményeke a valós sokasági átlaggal és aránnyal!

```
AtlagokAtlaga = np.mean(MintaVetelek100Elem['Atlagok'])
AranyokAtlaga = np.mean(MintaVetelek100Elem['Aranyok'])

[AtlagokAtlaga, SokasagiAtlag]
```

```
## [167.4932536, 167.52914060096398]
```

```
[AranyokAtlaga, SokasagiArany]
```

```
## [0.329556, 0.3295046661880833]
```

Hoppá, a kétféle értékek egészen közel vannak egymáshoz! Sőt, némi kerekítéssel a becslőfüggvények átlaga megegyezik az adott paraméter valós sokasági értékével!

```
[round(AtlagokAtlaga,1), round(SokasagiAtlag,1)]  
  
## [167.5, 167.5]  
  
[round(AranyokAtlaga*100, 1), round(SokasagiArany*100, 1)]  
  
## [33.0, 33.0]
```

Amit itt tapasztalunk az a **TORZÍTATLANSÁG** jelensége. Eszerint, ha az összes lehetséges mintából számolt $\hat{\theta}$ -ok átlaga (vagy más szóval várható értéke) megegyezik a valós, sokasági θ értékével, akkor a $\hat{\theta}$ becslőfüggvény torzítatlan. A torzítatlanság esetünkben azért teljesül csak kerekítésekkel, mert mi csak 10000 db minta alapján vizsgálódunk, és nem vettük ki az összes lehetséges mintát, mivel azt valószínűleg nem bírta volna el a RAM-unk az $N = 9751$ elemű sokaság esetén. :)

A fenti fogalom matematikai formalizmussal az alábbi formát ölti. A képletben az $E()$ függvény az átlagolás, azaz várható érték jele, ami az angolban ugyebár *expected value* álnéven fut ... innen az E . :)

$$E(\hat{\theta}) = \theta$$

Ha a fenti egyenlőség teljesül az összes lehetséges tetszőleges n elemű mintában, akkor $\hat{\theta}$ torzítatlan becslése θ -nak.

Ebből kiindulva pedig megadhatjuk a **torzítottság** fokának (angolul **Bias**, rövidítve Bs) definícióját is, ami nem más, mint a becslőfüggvények ($\hat{\theta}$ -ok) várható értékének különbsége a valós, sokasági θ értéktől:

$$Bs = E(\hat{\theta}) - \theta$$

Láthatjuk, hogy 10000 db mintát vizsgálva az **átlag** és **arány**, mint θ paraméterek esetében elég kicsi ez a Bs . Mindkét esetben **1 tizedesre** kerekítve **0** a **torzítás**.

```
# Bs(Átlag)  
round(AtlagokAtlaga - SokasagiAtlag, 1)  
  
## -0.0
```

```
# Bs(Arány)
round(AranyokAtlaga - SokasagiArany, 1)
```

```
## 0.0
```

Oké, a kis 10000 db $n = 100$ mintás kísérletünk alapján azt mondhatjuk, hogy a **mintaátlag** és **mintaárány** torzítatlan becslőfüggvényei a valós sokasági átlagnak és sokasági aránynak.

De mi a helyzet a szórás frontján? Konkrétan, **első körben vizsgáljuk meg**, hogy a mintákból számolt szórásnégyzetek torzítatlan becslései-e a sokasági szórásnégyzetnek, azaz **varianciának**!

```
VarianciakAtlaga = np.mean(MintaVetelek100Elem['Varianciak'])
```

```
[VarianciakAtlaga, SokasagiVariancia]
```

```
## [1925.3877225365777, 1943.781182155494]
```

```
# Bs(Variancia)
round(VarianciakAtlaga - SokasagiVariancia, 1)
```

```
## -18.4
```

Jaj! Olybá tűnik, hogy a **válasz NEM!** A mintából számolt varianciák, azaz $(s^*)^2$ -ek átlaga a valós, sokasági varianciához képest esetünkben 18.4-gyel alacsonyabb érték! Tehát, a mintavariancia, mint becslőfüggvény lefelé torzít a valós sokasági szóráshoz képest! Kellemetlen. Hiszen, ez azt jelenti, hogy egy mintából számolt variancia a valós sokasági értékhez képest jó eséllyel kisebb lesz. Ez azért szerencsétlen, mert eszerint a vizsgált ismérvünk szóródásáról, indagozásáról egy mintából nézve a valósághoz képest jellemzően kisebb értéket látunk. Tehát, pl. egy részvény árfolyamának szóródását (kockázat) az árfolyamadatok egy mintájából nézve a valósághoz képest jellemzően kisebbnek látjuk. A „kockázat” alulbecslése pedig egy olyan probléma, amivel kezdeni kell valamit!

A jelenség matematikailag az alábbi módon írható le:

$$E((s^*)^2) < \sigma^2$$

Azaz:

$$Bs((s^*)^2) < 0$$

5.2.1. Az aszimptotikus torzítatlanság fogalma

Ami valamilyen szinten menti a helyzetet az az a tény, hogy bár a **sokasági variancia** alapból torzítottan becsülhető a mintavarianciákkal, de a becslés viszont **aszimptotikusan torzítatlan**. Ez azt jelenti, hogy mintaelemszám növelésével a torzítás mértéke ($|Bs|$) csökken, konkrétan 0-ra tart. Azaz:

$$\lim_{n \rightarrow \infty} Bs((s^*)^2) = 0$$

Próbáljuk szemléltetni a jelenséget! A data frame-k 1. fejezet végén bemutatott **oszlopkiválasztásával** 10000 db $n = \{10, 20, 30, \dots, 90, 100\}$ elemű minta esetén kiszámoljuk a mintavarianciák $Bs((s^*)^2)$ értékét a valós sokasági varianciához (σ^2) képest. Természetesen, technikailag ezt egy **for** ciklus segítségével tudjuk megoldani:

- A ciklus minden iterációjában kiválasztjuk a megfelelő elemszámú mintákat a **MintaVetelek100Elem** data frame-ból
- Kiszámoljuk minden elemszám esetén $(s^*)^2$ -et mind a 10000 db mintára
- Kiszámoljuk és egy **list**-ben eltároljuk $Bs((s^*)^2)$, mint $E((s^*)^2)$ és a valós, sokasági variancia, σ^2 különbsége

```
# Üres lista létrehozása Bs-ek tárolására
Bs_Lista = []
# Vizsgált elemszámok listájának létrehozása
# 10 és 100 közötti egész számok felsoroltatása a 'range' függvényben 10-es lépésközzel
# Felső határ 101 a nyílt intervallum miatt
Elemszam_Lista = range(10, 101, 10)

# Ciklus indítása
for AktualisElemszam in Elemszam_Lista:
    AktualisMintaVetelek = MintaVetelek100Elem.iloc[:, 0:AktualisElemszam].copy()
    AktualisMintaVetelek['Varianciak'] = np.std(AktualisMintaVetelek, axis = 1)**2
    AktualisVarianciakAtlaga = np.mean(AktualisMintaVetelek['Varianciak'])
    AktualisBs = AktualisVarianciakAtlaga - SokasagiVariancia
    Bs_Lista.append(round(AktualisBs, 1))

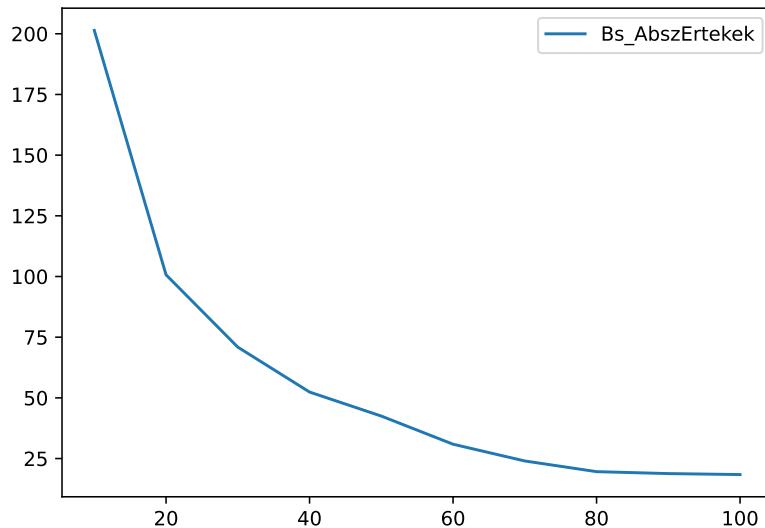
# Eredmény megtekintése
Bs_Lista

## [-201.4, -100.7, -70.9, -52.4, -42.5, -30.9, -24.0, -19.6, -18.8, -18.4]
```

Szépen láthatjuk, hogy a Bs értékek abszolút értéke az elég hatalmas 201.4-től indulva **szépen lefut** az $n = 100$ esetben **korábban is** mért 18.4-be. Az eredmények még látványosabbak egy vonaldiagramon ábrázolva.

```
# Vizsgált elemszámok és a mért Bs-ek data frame-be rendezése
# Ahol az elemszámok a sorindexek
BsData = pd.DataFrame(np.abs(Bs_Lista), columns=['Bs_AbszErtekek'], index = range(10, 100))

# Ábrázolás a 'plot' metódussal: nem kell paraméterezni, mert csak egy oszlopunk van
BsData.plot()
plt.show()
```



Szépen, gyakorlatilag exponenciális ütemben csökken a Bs abszolút érték, bár a csökkenés nagysága $n = 90$ -ről $n = 100$ -ra már nem túl jelentős! Ez azt jelenti, hogy varianciák esetén a torzítás mértéke függ az n elemszámktól! Minél nagyobb az elemszám, annál kisebb a torzítás mértéke, tehát az $|Bs|$.

5.3. A korrigált mintavariancia

A 2.1. fejezetben tapasztalt tényt, miszerint a mintavariancia $(s^*)^2$ a valós, sokasági σ^2 -nek **aszimptotikusan torzítatlan becslése** fel lehet használni a variancia torzítási problémára megoldására.

Ugyebár azt tudjuk az szimptomikusan torzítatlanságból, hogy minél nagyobb az elemszám, annál kisebb a torzítás mértéke. Sőt, azt is meg lehet mondani, hogy a mintavariánciák várható értéke, $E((s^*)^2)$ arányaiban $\frac{n-1}{n}$ -nel tér el

a sokasági varianciától, σ^2 -től. Azaz igaz a következő egyenlőség:

$$\frac{E((s^*)^2)}{\sigma^2} = \frac{n-1}{n}$$

Újrahasznosítva a *Bs*-ek meghatározására alkalmazott **for** ciklusos megoldásunkat, a fenti összefüggés helyessége is ellenőrizhető $n = \{10, 20, 30, \dots, 90, 100\}$ elemszámok mesetén.

```
# Üres lista létrehozása a (várható érték) / (sokasági variancia) hánnyadosok tárolására
Hanyados_Lista = []
# Üres lista létrehozása az (n-1)/n hánnyadosok tárolására
ElemszamHanyados_Lista = []
# Vizsgált elemszámok listájának létrehozása
# 10 és 100 közötti egész számok felsoroltatása a 'range' függvényben 10-es lépésközzel
# Felső határ 101 a nyílt intervallum miatt
Elemszam_Lista = range(10, 101, 10)

# Ciklus indítása
for AktualisElemszam in Elemszam_Lista:
    AktualisMintaVetelek = MintaVetelek100Elem.iloc[:, 0:AktualisElemszam].copy()
    AktualisMintaVetelek['Varianciak'] = np.std(AktualisMintaVetelek, axis = 1)**2
    AktualisVarianciakAtлага = np.mean(AktualisMintaVetelek['Varianciak'])
    Hanyados_Lista.append(round(AktualisVarianciakAtлага/SokasagiVariancia, 3))
    ElemszamHanyados_Lista.append(round((AktualisElemszam - 1)/AktualisElemszam, 3))

# Eredmények összefűzése data frame-be
Hanyados_df = pd.DataFrame(
    list(zip(ElemszamHanyados_Lista, Hanyados_Lista)),
    columns=['(n-1)/n', 'VarhatoErtek/SokasagiVar'])
Hanyados_df

##      (n-1)/n  VarhatoErtek/SokasagiVar
## 0      0.900          0.896
## 1      0.950          0.948
## 2      0.967          0.964
## 3      0.975          0.973
## 4      0.980          0.978
## 5      0.983          0.984
## 6      0.986          0.988
## 7      0.988          0.990
## 8      0.989          0.990
## 9      0.990          0.991
```

Szuper, aránylag szépen kijön a kétféle hánnyadosok közötti egyezőség! :) Persze itt is van némi eltérés, mivel csak 10000 db mintát vizsgálunk

és nem az összes lehetségeset, de ez még így is látványos egyezés! Így már érthető, hogy a B_s abszolút értéke miért nem csökkent már látványosan $n = 90$ -ről $n = 100$ -ra: az $\frac{n-1}{n}$ hányados minden esetben már elég kicsit volt, így a torzítás mértéke is!

Viszont, ha a $\frac{E((s^*)^2)}{\sigma^2} = \frac{n-1}{n}$ egyenlőség igaz, akkor azt átrendezve a következő összefüggésre jutunk:

$$\sigma^2 = \frac{n}{n-1} \times E((s^*)^2)$$

Konstans szorzót egy átlagolás ($E(\dots)$) eredményén alkalmazni ugyan az, mintha minden kiátlagolandó elemet felsoroztam volna azzal a szorzóval. Tehát az $\frac{n}{n-1}$ bevhető a várható érték függvényen belülre:

$$\sigma^2 = E\left(\frac{n}{n-1} \times (s^*)^2\right)$$

Mindezek alapján pedig azt mondhatjuk, hogy **az $s^2 = \frac{n}{n-1} \times (s^*)^2$ módon KORRIGÁLT MINTAVARIANCIA már TORZÍTATLANUL becsli a valós sokasági varianciát, azaz σ^2 -t!** Hiszen $\sigma^2 = E(s^2)$.

Tyűha, ez nagyon szépen hangzik! :) Próbáljuk ki! Számoljuk ki a 10000 db $n = 100$ elemű mintában a korrigált mintavarianciákat, és nézzük meg azok átlagát (várható értékét)!

```
# Elemszám megadása külön változóban
n = 100

# Korrigált varienciák
MintaVetelek100Elem['KorrigaltVar'] = (n/(n-1)) * MintaVetelek100Elem['Varianciak']

# Torzítatlanság ellenőrzése
KorrVarAtlaga = np.mean(MintaVetelek100Elem['KorrigaltVar'])

[KorrVarAtlaga, SokasagiVariancia]
```

```
## [1944.8360833702807, 1943.781182155494]
```

```
round(KorrVarAtlaga - SokasagiVariancia, 1)
```

```
## 1.1
```

Győzelem! :) Ha nem is szünt meg teljesen a dolog, de láthatóan nagyon alacsony, majdnem elhanyagolható lett a B_s mértéke! Sőt, már nem lefele torzítunk azzal a minimális 1.1-gyel, hanem felfelé, ami egy szóródás

becslésnél még a „*jobbak eset*”. Lásd a korábbi pénzügyi kockázat becslése példát.
:) Ha lenne több mintánk, akkor a korrekció ki is nullázná a *Bs-t*.

Ezek alapján akkor jó lenne, ha lenne valami beépített függvényünk az std helyett, ami mintaadatok esetén alapból a **KORRIGÁLT SZÓRÁS**
 $s = \sqrt{s^2} = \sqrt{\frac{n}{n-1} \times (s^*)^2}$ értékét számolja!

Nos, valójában az std tud korrigált szórást számolni egy extra paraméter segítségével. Ahhoz, hogy megértsük a paraméter működését egy picit végig kell gondolni a korrigált szórás képletének a működését.

Alapból a mintaadatok s^* szórását az alábbi képlettel számoljuk:

$$s^* = \sqrt{\frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n}}$$

Azaz, megnézzük, hogy minden y_i mintaelem mennyivel tér el a minta \bar{y} átlagától, majd ezen eltéréseket négyzetre emelve összeadjuk és az összeget leosztjuk a minta n elemszámával, végül gyököt vonunk az egész hánnyadosból. Ennek az értéknek a négyzete a sima, *nem korrigált* variancia:

$$(s^*)^2 = \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n}$$

Ha a fenrti variancia képletet beszorozzuk $\frac{n}{n-1}$ -gyel akkor a következő egyszerűsítéseket tehetjük:

$$s^2 = \frac{n}{n-1} \times \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n} = \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n-1}$$

Tehát, a minta korrigált szórását úgy számoljuk ki mint a nem korrigáltat, csak a NEVEZŐBEN $n - 1$ -gyel osztunk, nem pedig n -nel:

$$s = \sqrt{\frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n-1}}$$

Ezt az eltérést a nevezőben a ddof = 1 paraméter beállítással jelezzük a numpy csomag std függvényében. Könnyen kitalálható, hogy alapértelmezésben ddof = 0 beállítással fut az std függvény. :)

Lássuk is a dolgot akcióban!

```
# Korrigált varianciák 'std'-vel
MintaVetelek100Elem['KorrigaltVar_std'] = np.std(MintaVetelek100Elem.iloc[:,0:100], axis=1, ddof=1)

# Torzítatlanság ellenőrzése
```

```
KorrVarAtlaga_std = np.mean(MintaVetelek100Elem['KorrigaltVar_std'])

[KorrVarAtlaga_std, SokasagiVariancia]

## [1944.8360833702802, 1943.781182155494]

round(KorrVarAtlaga_std - SokasagiVariancia, 1)

## 1.1
```

Királyság! Tökéletesen ugyan ott vagyunk, mint az előbb a manuális számolással! :)

Szépen szakszavakkal összefoglalva tehát az a fő tanulságunk, hogy

1. A sima mintavariancia $((s^*)^2)$ a sokasági variancia σ^2 TORZÍTOTT becslőfüggvénye
2. A korrigált mintavariancia (s^2) viszont a sokasági variancia σ^2 TORZÍTATLAN becslőfüggvénye

5.4. A medián torzítatlansága

Stat. 1-en nagyon fontos mutatóink volt a medián, mint a vizsgált ismérv felezőpontja, hiszen nem volt érzékeny a kilógó értékekre az adatsorban, mint az átlag. **Nézzük meg** itt a Balaton átúszás 100 elemű mintáinak példáján, hogy ez a statisztikai paraméter **torzítatlanul becsülhető-e!**

```
# Sokasági medián átúszási idő
SokasagiMedian = np.median(Balcsi.PERC)

# Mintabeli mediánok kiszámítása
MintaVetelek100Elem['Medianok'] = np.median(MintaVetelek100Elem.iloc[:, 0:100], axis=1)

# Mintabeli mediánok átlaga
MedianokAtlaga = np.mean(MintaVetelek100Elem['Medianok'])

# Torzítatlanság ellenőrzése
[MedianokAtlaga, SokasagiMedian]

## [162.3393025, 162.26666666666668]
```

```
round(MedianokAtlaga - SokasagiMedian, 1)
```

```
## 0.1
```

Olybá tűnik, hogy a medián a mintabeli mediánokkal **torzítatlanul becsülhető**. A $Bs(me) = E(me) - Me$ eltérés olyan minimális, hogy simán elhíhető, hogy megszűnik, ha az összes lehetséges $n = 100$ lemeű mintát vizsgálnánk és nem csak 10000-et.

5.5. A Standard Hiba (*SH*) fogalma

Szép és jó, hogy megállapítottuk, hogy a mintaátlag, mintaarány, korrigált mintavariancia és mintamedián **torzítatlan becslőfüggvényei** a nekik megfelelő sokasági θ paramézereknek, de **mire jó ez nekünk a gyakorlatban, amikor csak egyetlen egy darab mintavételünk van?**

Hiszen, mint a 3-4. fejezetekben tapasztaltuk, a torzítatlanság csak annyit mond, hogy ha van **nagyon-nagyon sok mintavételünk**, akkor a vizsgált statisztikai mutatóink/paraméterünk mintából számított értékei (becslőfüggvények) átlagosan eltalálják a mutató valós, sokasági értékét. De sajnos ebbe a definícióba nagyon sok minden belefér, és igazából **önmagában a mintavételi hibáról (MVH)** nem mond semmit. Ezt a problémát nagyon jól érzékelteti a következő favicc.

„Egy mérnök, egy fizikus és egy statisztikus együtt mennek vaddisznóra vadászni. Alig tesznek meg néhány lépést az erdőben, márás észrevesznek 150 méterre egy hatalmas példányt. A mérnök felemeli a puskáját, céloz és lő, de három méterrel melléitalál jobbra. A fizikus így okoskodik:”Egy kis szellő fúj balról, ha kicsit balra célezök, akkor eltalálom.” Ó is célbaveszi a szarvast, lő és három méterrel balra elvéti. A statisztikus felugrik, és örvendezni kezd: „Megvan! Megvan! Eltaláltuk!”

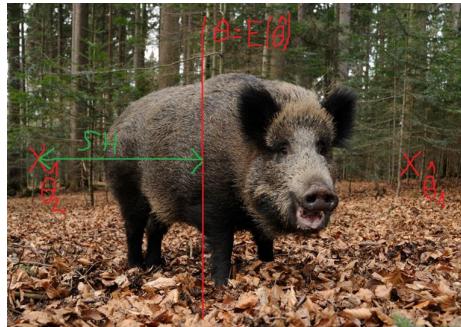
– Méltán Ismeretlen Szerző

Ugyebár kedvenc viccbéli statisztikus azért örvendez, mivel a három méterrel jobbra és balra hibázó két lövés átlagban pont telibe kapta szegény vaddisznónkat! Na, hát erről szól a **torzítatlanság** is:

- Le akarunk vadászni *lövésekkel*, azaz *mintavételekkel* egy statisztikai paraméter sokasági értékét, θ -t.
- Az első lövés, az 1. mintavételből származó becslőfüggvényünk értéke $\hat{\theta}_1$
- A második lövés, a 2. mintavételből származó becslőfüggvényünk értéke $\hat{\theta}_2$

- Ezek átlagban, azaz várható értékben eltallálják a keresett valós értéket, θ -t: $E(\hat{\theta}) = \theta$
- Íme: ez a *torzítatlan becslés* definíciója :)

Az statisztikai „ $\hat{\theta}$ ”-os vadászat és a vaddisznó vadász közti **analógia az alábbi ábrán szemléltethető**. **FONTOS** :)



A fenti ábrán bejelöltem zölddel egy tetszőleges $\hat{\theta}_i$ és a valós, sokasági θ közti távolságot. Valójában ez az a távolság, amire kíváncsiak vagyunk egy tetszőleges FAE mintából számolt becslés és a keresett mutató sokasági értéke közötti távolság! Hiszen a gyakorlatban csak egy db mintavételünk van, és az egyetlen egy megfigyelt mintából kimókolt $\hat{\theta}$ és θ közti távolságot kéne kiszámolni! Ez lenne ugyebár a MVH, amit kersünk! Nos, a torzítatlanságnak hála van egy módszer, amivel ezt a távolságot ki lehet számolni! Hiszen, ha a torzítatlanság miatt $E(\hat{\theta}) = \theta$, akkor ez azt jelenti, hogy a sok-sok mintából számolt $\hat{\theta}_i$ értékek szórása épp a keresett zöld távolság. Hiszen mi is a szórás általános értelmezése? Egy véletlenszerűen kiválasztott elem az adatsorból várhatóan mennyivel tér el az átlagtól. Hogyan fordul ez le a $\hat{\theta}$ -ok adatsorára? Ha a sok-sok lehetséges mintavételből kiválasztok egyet, akkor a kiválasztott mintából számolt $\hat{\theta}$ várhatóan szórásnyival tér el a $\hat{\theta}$ -ok átlagától, azaz a torzítatlanság miatt épp a valós, sokasági θ értékétől. Ebből az okfejtésből kiindulva a $\hat{\theta}$ -ok szórását standard mintavételei hibának, röviden csak standard hibának, „SH”-nak nevezzük.

Akkor ezen felbuzdulva **számoltassuk ki** pitonkával az átlag és arány **standard hibáit**, mint a $\hat{\theta}$ -ként funkcionáló **mintaátlagok és minatarányok szórása!** Itt sima szórást kell stámolni, semmi korrekció nem kell az std függvényben.

```
SH_Atlag = np.std(MintaVetelek100Elem.Atlagok)
SH_Arany = np.std(MintaVetelek100Elem.Aranyok)
```

```
SH_Atlag
```

```
## 4.402018186841923
```

```
SH_Arany
```

```
## 0.04667871960540553
```

Mivel az átlag és arány torzítatlan becslések, így a szórásként kiszámolt standard hibát a következőképpen lehet értelmezni:

- 100 elemű minták esetén egy konkrét mintaátlag várhatóan 4.4 perccel tér el az átlagok átlagától, azaz a valós, sokasági átlagos átúszási időtől.
- Egy konkrét 100 elemű mintában a Balatont 3 órán túl átúszók aránya várhatóan 4.67 százalékponttal tér el a teljes sokaság hasonló arányától.

Hasonlóan működik a dolog ezen a szinten a varianciákra és mediánokra is. HF kiszámolni és értelmezni az eredményeket. :)

Viszont, **olybá tűnik, hogy nem vagyunk sokkal előrébb**. Mivel, bár a standard hiba megadja, hogy egy mintából számolt becslés várhatóan mennyivel tér el a valóságtól, de a standard hiba (*SH*) kiszámolásához sok-sok mintavételre van szükség, hiszen ezekből a mintákból számolt $\hat{\theta}$ -k szórásaként tudjuk az *SH* értéket megahtározni! Az kéne, hogy az *SH*-t egyetlen egy mintavételből is valahogy ki tudjuk számolni!

Erre az átlag és az arány esetében van megoldásunk. Ugyanis e két mutató esetében a ***SH* kifejezhető zárt formulával** is:

- $SH(\bar{y}) = \frac{\sigma}{\sqrt{n}}$
- $SH(p) = \sqrt{\frac{P(1-P)}{n}}$

Tehát, a két standard hiba a sokasági szórás (σ) és a keresett sokasági arány (P) és a mintaelemszám n ismeretében megahtározahtó. Próbáljuk is ki a dolgot.

```
n = 100
```

```
SH_Atlag_Formula = SokasagiSzoras / np.sqrt(n)
SH_Arany_Formula = np.sqrt((SokasagiArany * (1-SokasagiArany))/n)
```

```
[SH_Atlag, SH_Atlag_Formula]
```

```
## [4.402018186841923, 4.408833385551663]
```

```
[SH_Arany, SH_Arany_Formula]
```

```
## [0.04667871960540553, 0.04700333404646558]
```

Olybá tűnik, hogy **nagyjából egyezik a két érték.** :) A minimális eltérés megint abból fakad, hogy nem az összes lehetséges mintát vizsgáltuk, csak 10000 db-ot, amikor a SH -kat a $\hat{\theta}$ -ok szórásaként számoltuk ki.

Viszont, itt **megint az a probléma, hogy a SH kiszámításhoz olyan dolgokat kell ismerni, amiket egyetlen egy mintavétel esetén nem ismerünk:** sokasági szórás (σ) és a sokasági arány (P). **NODE!** Ezeket az ismeretlenek legalább tudjuk helyettesíteni az egy mintavételből számolt torzítatlan becslésükkel: a P -t helyettesítjük p -vel, a σ -t pedig a korrigált szórással, s -el (hiszen egy torzítatlan becslése kell).

Ennyi ismerettel pedig akkor pl. az 5. mintánk alapján a $SH(\bar{y}) \approx \frac{s}{\sqrt{n}}$ és $SH(p) \approx \sqrt{\frac{p(1-p)}{n}}$ közelítő képletekkel meg tudjuk már határozni a SH -kat.

```
n = 100
```

```
SH_Atlag_ÖtödikMinta = np.sqrt(MintaVetelek100Elem.Varianciak[4] / n)
```

```
SH_Arany_ÖtödikMinta = np.sqrt((MintaVetelek100Elem.Aranyok[4] * (1 - MintaVetelek100Elem.Aranyok[4])) / n)
```

```
SH_Atlag_ÖtödikMinta
```

```
## 4.558455568470775
```

```
SH_Arany_ÖtödikMinta
```

```
## 0.04702127178203499
```

Nem tűpontos a SH közelítése, de azért **nagyságrendileg látszik, hogy jó nyomon járunk már egy mintavétel alapján is!** :)

Sajnos hasonló közelítő képleteink a varienciák més mediánok esetén NINCSENEK. Ott majd más trükkökkel próbáljuk kiszámolni az SH -kat egy mintavétel alapján. De erről majd pár anyaggal később. :)

Most még egy fontos elnevezés: a SH^2 -et gyakran nevezi a szaknyelv a becslőfüggvény VARIANCIÁJÁNAK. Én nem szeretem ezt az elnevezést, mert könnyű összekeverni a minta vagy éppen a sokasági adatok varianciájával, de sok helyen használják ezt az elnevezést, így fontos tudni! Tehát, ha valahol olyat olvastok, hogy a mintaátlagok varianciája ennyi vagy a mintaarányok varianciája amennyi, akkor ott a költő az adott

mutatók SH^2 -re gonfolt. Jelölni pedig az elnevezés alapján logikus módon így szokás a dolgot: $SH^2(\hat{\theta}) = Var(\hat{\theta})$

Ennek kapcsán talán fontos megemlékezni összefoglalásként arról, hogy itt a becsléselméletben milyen különböző szórásokkal, illetve varianciákkal találkoztunk:

- **Sokasági szórás:** A sokaság elemeinek várható eltérése a sokaság átlagától. Jele: σ .
 - Négyzete: sokasági variancia, σ^2
- **Korrigálatlan mintaszórás:** Egy db minta elemeinek várható eltérése az egy db mintánk átlagától. Jele: σ^*
 - Négyzete: korrigálatlan mintavariancia, $(\sigma^*)^2$
- **Korrigált mintaszórás:** Ugyan az, mint a korrigálatlan mintaszórás, csak *torzítatlan* becslést ad a valós, sokasági szóródásra. Jele: s
 - Négyzete: korrigált mintavariancia, s^2
- **Standard hiba:** Sok-sok mintából számolt becslőfüggvény, azaz $\hat{\theta}$ szórása. *Torzítatlanság* esetén egy konkrét mintából számolt becslőfüggvény eltérése a vizsgált θ paraméter valós, sokasági értékétől. Jele: $SH(\hat{\theta})$
 - Négyzete: becslőfüggvény varianciája, $Var(\hat{\theta})$

5.5.1. A konzisztens becslés fogalma

A standard hibák formulájából az is kikövetkeztethető a **mintaátlag** és **mintaarány** becslőfüggvények esetében, hogy ezek **konzisztens** becslések **is a valós sokasági átlagra, arányra**.

Ugyanis, általánosságban **egy $\hat{\theta}$ becslőfüggvény akkor konzisztens, ha SH -ja, a mintaelemszám (n) növelésével 0-ba tart**:

$$\lim_{n \rightarrow \infty} SH(\hat{\theta}) = 0$$

Azaz, a egyre nagyobb a mintaméret, akkor a vizsgált statisztikai mutatóink egy mintából számított értéke ($\hat{\theta}$) egyre közelebb lesz a mutató valós, sokasági értékéhez (θ).

Könnyen látható, hogy **mintaelemszám (n) függvényében, mind a mintaátlagok, mind a mintaarányok standard hibái $f(n) \sim \frac{1}{n}$ stílusú hiperbola függvények, amik a végtelenbe tartó n nevező esetén 0-ba tartanak:**

$$\lim_{n \rightarrow \infty} SH(\bar{y}) = \lim_{n \rightarrow \infty} \frac{\sigma}{\sqrt{n}} = 0$$

és

$$\lim_{n \rightarrow \infty} SH(p) = \lim_{n \rightarrow \infty} \sqrt{\frac{P(1-P)}{n}} = 0$$

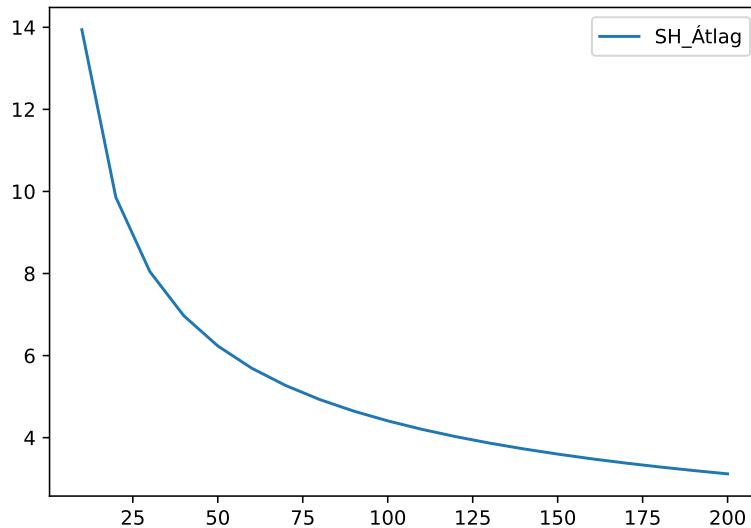
Rajzoljuk is ki a standard hiba függvényt mondjuk $SH(\bar{y})$ esetében
 $n = \{10, 20, \dots, 200\}$ elemszámok mellett, és látni fogjuk a hiperbola alakot.
Bár a dolog nem teljesen tiszta, mert a 0-ba tartás sebesség a képlet alapján „gyökös”. :) Az ábrázolást végző Python kód logikája teljesen egyezik azzal, amit a 2.1. fejezetben is használtunk.

```
# Üres lista létrehozása a különböző elemszámok mellett vett SH-k tárolására
SH_Lista = []
# Vizsgált elemszámok listájának létrehozása
# 10 és 200 közötti egész számok felsoroltatása a 'range' függvényben 10-es lépésközze
# Felső határ 201 a nyílt intervallum miatt
Elemszam_Lista = range(10, 201, 10)

# Ciklus indítása SH-k számításához
for AktualisElemszam in Elemszam_Lista:
    SH_Lista.append(round(SokasagiSzoras / np.sqrt(AktualisElemszam), 3))

# Vizsgált elemszámok és a mért SH-k data frame-be rendezése
# Ahol az elemszámok a sorindexek
SHData = pd.DataFrame(SH_Lista, columns=['SH_Atlag'], index = range(10, 201, 10))

# Ábrázolás a 'plot' metódussal: nem kell paraméterezni, mert csak egy oszlopunk van
SHData.plot()
plt.show()
```



5.6. Az átlagos négyzetes hiba (*MSE*) fogalma

Ugyebár arra jutottunk, hogy egy statisztikai paraméter becslőfüggvénynek a szórása, mint standard hiba, csak akkor adja meg egy konkrét mintából számolt $\hat{\theta}$ várható eltérését a valós, sokasági θ értéktől, ha a becslőfüggvény torzítatlan, mert ekkor egyezik meg θ a sok-sok mintából számolt $\hat{\theta}$ -ok átlagával, $E(\hat{\theta})$ -val.

Ha nem torzítatlan becslőfüggvényről beszélünk, akkor manuálisan ki tudjuk számolni a sok-sok mintából megadott $\hat{\theta}$ -ok várható eltérését a valós, sokasági θ -tól. Ez a mutató lesz a $\hat{\theta}$ átlagos négyzetes hibája, angolul **Mean Squared Error = MSE**. A számoláshoz simán a klasszikus variancia képletet kell alkalmazni a következő módon:

$$MSE(\hat{\theta}) = \frac{\sum_{i=1}^K (\hat{\theta}_i - \theta)^2}{K}$$

A képletben K a mintavételek száma (nekünk most 10000) $\hat{\theta}_i$ egyszerűen az i -edik mintából számolt $\hat{\theta}$ értéke.

Számoljuk is ki az *MSE*-t, az egyetlen torzított becslőfüggvényre, a korrigálatlan mintavarianciára! A képletnél muszáj Pythonban a `sum` függvényt alkalmazni, és „manuálisan” lekódolni a formulát, mivel a beépített

`std` függvényel θ helyett $E(\hat{\theta})$ -hoz viszonyítanánk, és a torzítottság miatt e két érték nem esik most egybe!

```
MSE_Varianciak = np.sum((MintaVetelek100Elem.Varianciak - SokasagiVariancia)**2)/10000

MSE_Varianciak

## 163460.24046057483
```

Királyság! Namármost. Ez a *MSE* érték valójában a **becslés kétféje hibájának négyzetes összege**. Konkrétan

$$MSE = SH^2 + Bs^2$$

Ugyebár itt egy $\hat{\theta}$ becslőfüggvény eltérését a valós θ -tól két lépcsőben lehet megközelíteni:

1. *SH*: Mennyivel tér el egy konkrét minta $\hat{\theta}$ -ja a becslések átlagától, $E(\hat{\theta})$ -től.
2. *Bs*: Mennyivel tér el a becslések átlaga a valós θ -tól: $Bs = E(\hat{\theta}) - \theta$

És ha kiszámoljuk a gyakorlatban, akkor látjuk, hogy az *EMSE* tényleg **ennek a fenti kétféle hibának az összege**.

```
Bs_Varianciak = np.mean(MintaVetelek100Elem.Varianciak) - SokasagiVariancia
SH_Varianciak = np.std(MintaVetelek100Elem.Varianciak)
```

```
MSE_Varianciak_Osszeggel = Bs_Varianciak**2 + SH_Varianciak**2

[MSE_Varianciak, MSE_Varianciak_Osszeggel]
```

```
## [163460.24046057483, 163460.2404605743]
```

Jé, tényleg jó az összes logikánk is! :)

Természetesen, ahol **torzítatlan a becslés, ott a $Bs = 0$ miatt az $MSE = SH^2$ azonosság áll**. Vegyük pl. a mintaarányok esetét.

```
MSE_Aranyok = np.sum((MintaVetelek100Elem.Aranyok - SokasagiArany)**2)/10000
SH_Aranyok = np.std(MintaVetelek100Elem.Aranyok)
```

```
[MSE_Aranyok, SH_Aranyok**2]
```

```
## [0.0021789054991602453, 0.0021789028640000706]
```

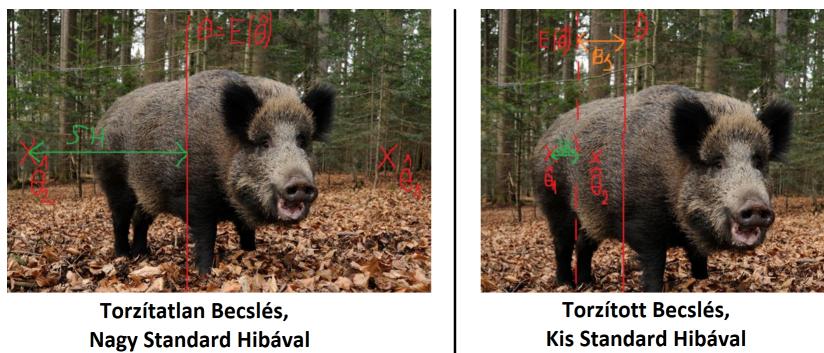
ExcellenT! :)

5.6.1. Különböző becslőfüggvények összehasonlítása

Az *MSE*-t kiválóan lehet használni, mint egy olyan mérőszámot, amivel választani tudunk egy θ sokasági paramétere terre adott több lehetséges $\hat{\theta}$ becslőfüggvény alternatíva közül. Kiváló példa erre az $(s^*)^2$ és s^2 , mint két alternatív becslőfüggény a sokasági variancia, σ^2 becslésére.

Mondhatnánk erre a kérdésre válaszként csípőből azt, hogy „de hát a torzítatlan becslőfüggvény biztos jobb”. Nos nem feltétlenül. Mert mi van ha egy torzított becslés standard hibája annyival kisebb a torzítatlannál, hogy az ellensúlyozza a torzítás mértékét, és a végén egy „lövés” (azaz $\hat{\theta}$ becslés) a torzított becslőfüggvényből közelebb esik a valós θ -hoz, mint a torzítatlan becslésből származó „lövés”.

Ezt nagyon jól lehet érzékelni az 5. fejezet vaddisznó vadászatos példáján egy másik nézőpontból: mi van ha a torzítatlan becslésnek akkor a standard hibája, hogy egy konkrét „lövés” (azaz $\hat{\theta}$ becslés) jóval messzebb lesz a valós θ -tól, mint egy torzított becslőfüggvényből származó becslés?



Az ábrán látható, hogy egy enyhén „balra” torzított becslőfüggvényből származó, de kis standard hibájú $\hat{\theta}_i$ becslések még eltalálják a vaddisznót, de a torzítatlan becslések, bár sok-sok mintavétel átlagában nagyon jól működnek, de egy konkrét $\hat{\theta}_i$ lövésnek akkora a standard hibája, hogy nagyon messzire elváti azt a vaddisznót!

Tehát a fenti jelenség miatt, ha egy adott θ -ra több becslőfüggvény közül kell választanunk, akkor azt az *MSE* alapján szabad csak megtennünk, mert az egyszerre veszi figyelembe a torzítás mértékét és a standard hibát is.

Lássuk a dolgot a gyakorlatban: Mi a jobb becslés a sokasági varianciára? A korrigált vagy a torzítatlan mintavariancia?

```
# Torzított becslés = Korrigálatlan mintavar.
Bs_Varianciak = np.mean(MintaVetelek100Elem.Varianciak) - SokasagiVariancia
SH_Varianciak = np.std(MintaVetelek100Elem.Varianciak)

MSE_Varianciak = Bs_Varianciak**2 + SH_Varianciak**2
```

```
# Torzítatlan becslés = Korrigált mintavar.
SH_KorrVarianciak = np.std(MintaVetelek100Elem.KorrigaltVar)

MSE_KorrVarianciak = 0 + SH_KorrVarianciak**2

[MSE_Varianciak, MSE_KorrVarianciak]

## [163460.2404605743, 166433.95684503784]
```

Hoppá, olybá tűnik, hogy $MSE((s^*)^2) < MSE(s^2)$, tehát a **nem korrigált** mintavariancia van közelebb a valós, sokasági σ^2 -hez egy „átlagos” mintavétel esetén, hanem a sima korrigálatlan verzió. Ugyanakkor az adatsor bizonytalanságának, szóródásának „rendszeres” alábecslése a legtöbb esetben nagyobb probléma, mint a némi magasabb standard hiba. Tehát, bár összességében, azaz MSE -ben az $(s^*)^2$ mintavételi hibája kisebb, mint s^2 -nek, de a kisebb hiba irányba „lefelé” van a torzítás miatt, és ezt nem szeretjük itt most. Inkább bevállalunk egy valamivel nagyobb, de „szimmetrikus” hibát. Így variancia esetében felülírjuk az MSE döntését, és a **korrigált mintavarianciát használjuk a legtöbb esetben**. Vagy másképpen fogalmazva azt mondhatjuk, hogy variancia esetén nagyobb súlyt helyezünk MSE -ben a Bs^2 -re, mint a SH^2 -re, és nem egyenlő mértékben preferáljuk a csökkenésüket. Pl. az **átlag standard hibájának $\frac{s}{\sqrt{n}}$ elvű közekítésénél ez azért is jogos**, mert a korrigálatlan mintaszórás használata esetén egy mintaátlag távolságát a valós, sokasági átlagtól az „átlagos mintavételben” alábecsülnénk, ami azért kellemetlen. :)

Ha két torzítatlan becslőfüggvény közül kell választanunk, akkor simán választhatjuk azt, aminek a standard hibája kisebb, hiszen ilyenkor ez egyben azt is jelenti, hogy az MSE -je is kisebb, mivel $Bs^2 = 0$. Ezt hívják úgy is a szakirodalomban, mint a **hatásosság kritériuma**: Két becslőfüggvény közül az a hatásosabb, amelynek SH -ja kisebb. :)

6. fejezet

A sokasági átlag intervallumbecslése

6.1. Ismétlés: Az átlag standard hibája a Balaton átúszás eredményeken

Ugyebár az 5. fejezetben a 2022-es Balaton átúszás résztvevőinek időeredményeit vizsgáltuk mintavételi és becslésemelleti szempontból elég alaposan. Töltsük is be egy `pandas` data frame-be ismét a LIDLBalaton2022.xlsx fájl adatait. Ebben az Excelben megvan az összes résztvevő időeredménye a PERC oszlopban. Ez az adatsor lesz most nekünk tehát a **sokaságunk**.

```
# Elemzéshez és ábrázoláshoz szükséges csomagok betöltése
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats

# Adatbeolvasás data frame-be
Balcsi = pd.read_excel("LIDLBalaton2022.xlsx")

Balcsi.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 9751 entries, 0 to 9750
## Data columns (total 3 columns):
## #   Column  Non-Null Count  Dtype
## ---  -----  -----  -----
```

```
## 0 Nev 9751 non-null object
## 1 Nem 9751 non-null object
## 2 PERC 9751 non-null float64
## dtypes: float64(1), object(2)
## memory usage: 228.7+ KB
```

```
Balcsi.head()
```

	Nev	Nem	PERC
## 0	Aba Attila	F	142.416667
## 1	Abaffy Károly	F	197.883333
## 2	Abaffy Kornél	F	197.983333
## 3	Abelovszki Hajnalka	N	182.000000
## 4	Abért Valentin ifj	F	222.516667

Meg is van akkor minden az $N = 9751$ részvevőnk. Ebben a tananyagban **most kizárolag az időeredmények átlagának becslésével** fogunk foglalkozni. Úgyhogy számoljuk ki, hogy az összes átúszó, azaz a **sokaság**, tekintetében mi az **időeredmények átlaga** ($\mu = \bar{Y}$). Ezen kívül majd jól jön még nekünk referenciaiként az időeredménynek sokasági **szórása** (σ) is.

```
SokasagiAtlag = np.mean(Balcsi.PERC)
SokasagiSzoras = np.std(Balcsi.PERC)
```

```
SokasagiAtlag
```

```
## 167.52914060096398
```

```
SokasagiSzoras
```

```
## 44.08833385551663
```

Ezeka alapján tudjuk tehát, hogy egy átlagos Balaton átúszó $\mu = 165.5$ perc alatt teljesítette a távot, amitől egy konkrét versenyző saját időeredménye várhatóan $\sigma = 44.1$ perccel tér el.

Az átlag becslése során az a feladatunk, hogy ezt a $\mu = 165.5$ perces átlagot valahogyan „megtippeljük” egy visszatevéses véletlen (azaz FAE) minta adatai alapján.

Tehát, vegyük is egy $n = 100$ elemű mintát a Balatonátúszók sokaságából a kedvenc 1992-es véletlen magunk mellett, és nézzük meg, hogy mennyi a mintaátlag, azaz \bar{y} értéke.

```
BalcsiMinta = Balcsi.sample(n = 100, replace = True, random_state = 1992)

MintaAtlag = np.mean(BalcsiMinta.PERC)

MintaAtlag

## 164.4403333333334
```

Tehát ebben az $n = 100$ elemű **mintában az átlagos átúszási idő** 164.4 perc. Az 5. fejezetben elvégzett okoskodásunk alapján a **mintaátlag** alapján úgy tudjuk lehatárolni a sokasági átlag (μ) értékét, hogy a **mintaátlag** értékre rámérem \pm annak **standard hibáját**. Hiszen a standard hiba megmutatja, hogy egy véletlenszerűen kiválasztott mintavétel átlaga várhatóan mennyivel tér el a valós sokasági átlagtól (mivel a \bar{y} mintaátlag alapból egy torzítatlan becslőfüggvénye a μ sokasági átlagnak). A gondolatmenet alapján tehát azt mondhatjuk, hogy a **sokasági átlag várhatóan mintaátlag \pm standard hiba által lehatárolt intervallumban nyugszik**.

Jó hír, hogy ugyebár az **átlag standard hibája sokasági szórás osztva gyök alatt mintaelemszám**, azaz $\frac{s}{\sqrt{n}}$ képlettel számolható, aminek az értékét egy mintavétel alapján is meg tudjuk közelíteni, ha a sokasági szórást, annak torzítatlan becslésével a **korrigált mintaszórással helyettesítjük**. Tehát, az **egy szem** $n = 100$ mintából a **standard hiba értéke az** $\frac{s}{\sqrt{n}}$ képlettel megközelíthető.

Ez alapján akkor az alábbi számolást tudjuk elkövetni a mintánkon.

```
n = 100
s = np.std(BalcsiMinta.PERC, ddof = 1) # figyeljünk a korrekcióra!
SH = s/np.sqrt(n)

[MintaAtlag - SH, MintaAtlag + SH]

## [160.55804594814538, 168.3226207185213]
```

Az eredményünk alapján a valós, sokasági átlag (μ) várhatóan 160.6 és 168.3 perc között, azaz a [160.6, 168.3] intervallumban helyezkedik el. Nos, az **intervallumos becslésünk helyes is, hiszen a valós sokasági átlag ugyebár $\mu = 167.5$ perc, ami tényleg benne van a mintánk alapján lehatárolt intervallumban**.

6.2. A mintaátlagok eloszlása

Na jó-jó, egy mintavétel esetén szerencsénk is lehetett. **Mennyire működik ez a standard hibás módszer jól sok-sok $n = 100$ mintavétel esetében?**

Töltsük csak be egy data frame-be azt a táblát, ami 10000 db $n = 100$ FAE mintavétel adatait tartalmazza! Az átlalam generált Excel, ami tartalmazza a 10000 minta adatait innen érhető el.

```
MintaVetelek100Elem = pd.read_excel("MintaDataFrame.xlsx")
MintaVetelek100Elem
```

```
##           Elem1        Elem2        Elem3      ...       Elem98       Ele99       Ele100
## 0    164.800000  129.066667  156.166667  ...  207.350000  159.666667  165.883333
## 1    152.516667  212.483333  152.900000  ...  307.266667  119.783333  128.216667
## 2    145.666667  185.266667  169.516667  ...  167.733333  228.366667  215.633333
## 3    185.683333  120.333333  201.250000  ...  182.766667  177.666667  112.450000
## 4    117.483333  142.350000  320.266667  ...  188.566667  189.166667  99.916667
## ...
## 9995   162.333333   83.350000  146.750000  ...  164.250000  131.933333  128.183333
## 9996   100.416667  161.433333  187.366667  ...  160.483333  168.416667  209.300000
## 9997   146.450000  160.783333  165.483333  ...  158.816667  167.733333  183.400000
## 9998   139.250000  140.466667  130.933333  ...  153.616667  112.366667  196.050000
## 9999   147.316667  173.450000  106.100000  ...  185.616667  171.800000  135.166667
##
## [10000 rows x 100 columns]
```

Oké, az eredményből látjuk is, hogy úgy néz ki a data frame, hogy **1 sor tartalmaz 1 db 100 elemű mintát és a mintaelemeket** (tehát a mintába besorolt versenyző percben mért időeredményét) **az oszlopokban tároljuk**.

Akkor most **minden minta esetében számoljuk ki a $\bar{y} \pm SH$ intervallumot**, és nézzük meg, hogy a valós sokasági átlag (μ) beleesik-e az intervallumba! Annyi előnyt is adjunk magunknak, hogy a standard hibát a sokasági szórás, azaz σ ismeretében számoljuk ki. Tehát a $SH = \frac{\sigma}{\sqrt{n}}$ képletet alkalmazzuk. Ugye ez annyiban előny, hogy σ -t egy db 100 elemű minta vizsgálata esetén NEM ismerjük! A számolás során figyeljünk arra, hogy a numpy függvényeket `axis = 1` paraméterrel használjuk, hiszen egy db minta elemei a sorokban vannak. Illetve, a számolást minden szorítsuk le a data frame első 100 oszlopára, hiszen a data frame oszlopait folyamatosan bővíteni fogjuk!

```
n = 100
SH = SokasagiSzoras / np.sqrt(n)

MintaVetelek100Elem['AtlagAlsoHatar'] = np.mean(MintaVetelek100Elem.iloc[:,0:100], axis=1)
MintaVetelek100Elem['AtlagFelsoHatar'] = np.mean(MintaVetelek100Elem.iloc[:,0:100], axis=1)
MintaVetelek100Elem

##           Elem1        Elem2      ... AtlagAlsoHatar AtlagFelsoHatar
## 0    164.800000  129.066667  ...      166.852833      175.670500
```

```

## 1 152.516667 212.483333 ... 153.617667 162.435333
## 2 145.666667 185.266667 ... 167.870000 176.687667
## 3 185.683333 120.333333 ... 159.025333 167.843000
## 4 117.483333 142.350000 ... 162.144000 170.961667
## ...
## 9995 162.333333 83.350000 ... 159.273500 168.091167
## 9996 100.416667 161.433333 ... 159.924000 168.741667
## 9997 146.450000 160.783333 ... 159.293500 168.111167
## 9998 139.250000 140.466667 ... 162.015000 170.832667
## 9999 147.316667 173.450000 ... 168.641500 177.459167
##
## [10000 rows x 102 columns]

```

Oké, akkor **meg is vannak** az átlag intervallumos becslésének **alsó-felső határai**. Számoljuk akkor **ki a találati arányt!** A számoláshoz azt a trükköt alkalmazzuk, amit a 4. fejezetben sütöttünk el: a `AdatokEgyben['Normal'] < 100` parancs egy `bool` tömböt ad vissza, amit összegezve megkapjuk a „*kedvező esetek*”, vagyis a μ -t helyesen eltaláló intervallumok darabszámát.

```

MintaVetelekSzama = len(MintaVetelek100Elem)

np.sum((MintaVetelek100Elem['AtlagAlsoHatar'] < SokasagiAtlag) & (MintaVetelek100Elem['AtlagFelsőHatar'] > SokasagiAtlag)) / MintaVetelekSzama
## 0.684

```

Nos, olybá tűnik, hogy a $\bar{y} \pm SH$ módszer csak a **mintavételek kb. 68%-ban találja el a valós, sokasági átlagot, azaz μ -t!** Gáz Géza! Azért ennél nagyobb találati arányt szeretnénk! Mondjuk legalább valami 90% környékét.

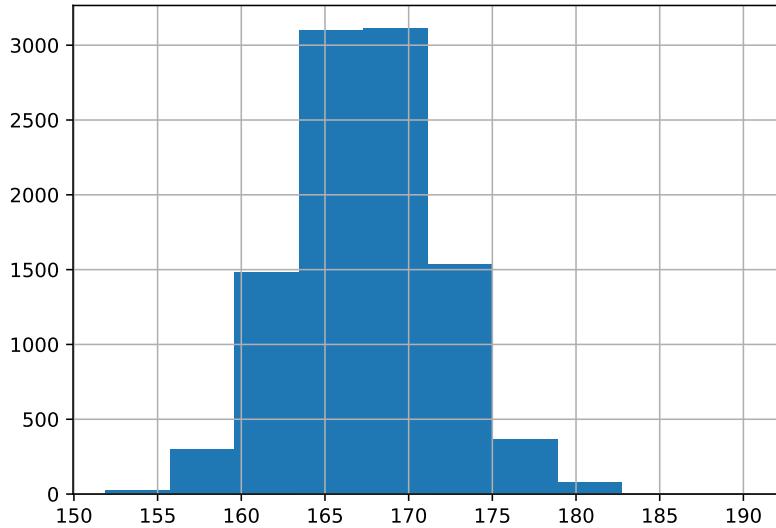
Ahhoz, hogy megértsük miért alakul gyéren ennek a módszernek találati aránya, **nézzünk csak rá az \bar{y} mintaátlagok hisztogramjára!** Most a hisztogramon nem optimalizálom az osztályközök számát, elfogadom a `numpy` alapbeállításait.

```

MintaVetelek100Elem['MintaAtlagok'] = np.mean(MintaVetelek100Elem.iloc[:,0:100], axis=1)

MintaVetelek100Elem.MintaAtlagok.hist()

```



Hoppácska! Dehát, ez itt a világ legszebb normális eloszlása!

Ami azért második elgondolásra teljesen logikus, mivel a Centrális Határeloszlás Tétel (**CHT**) dolgozik a háttérben. Ha nem ugrik be a CHT, akkor vissza a 3.2.4. fejezethez! :)

A **CHT** szerint ugyebár ha az adatsor elemi véletlen hatások összegződéseként állnak elő, akkor az adatsor normális eloszlást követ. A \bar{y} mintaátlagok adatsora pedig pont olyan adatsor, ami a **CHT** feltételnek megfelel! Hiszen a mintaátlag úgy jön ki, hogy a mintaelémeket összeadom és elosztom a minta elemszámával. Mivel a mintavétel módja **FAE**, így biztos lehetünk benne, hogy egy mintaelém, az egy véletlen húzás, egy véletlen hatás eredménye. Aztán meg ezeket adom össze. Végén osztok n -bel, de az minden ugyan annyi, így nem változtat a lényegen.

Ha pedig a sok-sok mintából számolt átlagok adatsora normális eloszlású, akkor azt is tudom, hogy milyen átlagú és milyen szórású normális eloszlást követ!

- A **torzítatlanság** miatt tudom, hogy a mintaátlagok átlaga a sokasági átlag, azaz μ .
- Mintaátlagok szórása pedig ugye nem más, mint a **standard hiba**, tehát $\frac{\sigma}{\sqrt{n}}$

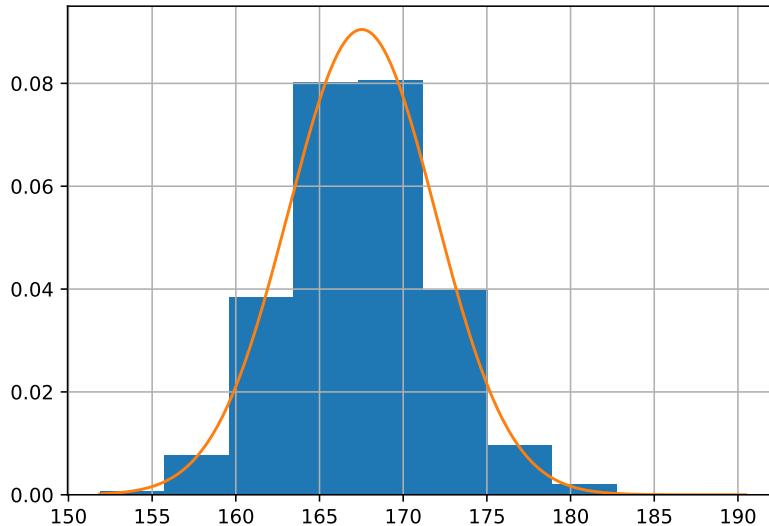
Szumma szummárom, akkor a **sok-sok mintából számolt mintaátlagok** \bar{y}

adatsora az alábbi eloszlást követi:

$$\bar{y} \sim N\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$$

Ezt az illeszkedést simán letesztelhetjük grafikusan is az 3.2.3. fejezetében látott módon. Figyeljük meg, hogy a `stats` csomag `norm.pdf` függvényében az átlagot a korábban kiszámolt `SokasagiAtlag`, a szórást pedig a szintén az előbb kiszámolt `SH` objektumok segítségével adom meg!

```
MintaVetelek100Elem.MintaAtlagok.hist(density = True)
x_tengely = np.arange(np.min(MintaVetelek100Elem.MintaAtlagok), np.max(MintaVetelek100Elem.MintaAtlagok))
y_tengely = stats.norm.pdf(x = x_tengely, loc = SokasagiAtlag, scale = SH)
plt.plot(x_tengely, y_tengely)
plt.show()
```



Nagyon szép, az illeszkedés, olybá tűnik a **CHT ismét működik! :)**

6.3. A sokasági átlag konfidencia-intervalluma

Nézzük akkor meg mi a valószínűsége, hogy egy véletlenszerűen kiválasztott érték egy $N(\mu, SH)$ eloszlásban a $\mu \pm SH$ intervallumba esik!

```
stats.norm.cdf(x = SokasagiAtlag + SH, loc = SokasagiAtlag, scale = SH) - stats.norm.cdf(x = SokasagiAtlag - SH, loc = SokasagiAtlag, scale = SH)
```

0.6826894921370872

Hoppáré! Ez is éppen kb. 68%! Tehát, az, hogy egy $\bar{y} \pm SH$ becslés csak a mintavételek 68%-ban pontos nagyjából egy valószínűségszámítási szükségszerűség.

Kérdés, hogy mit tehetünk ez ellen? Mihez kezdhetünk, ha mondjuk nem 68%-os, hanem valami jobb, mondjuk 95%-os megbízhatóságú intervallumbecslést szeretnénk adni a sokasági átlagra (vagy más néven sokasági várható értékre)?

Az okoskodáshoz a 3.2.6. fejezetre fogunk támaszkodni.

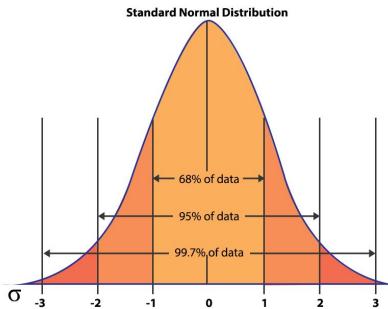
Ugyanis azt tudjuk, hogy ha a **mintaátlagok eloszlása** az alábbi:

$$\bar{y} \sim N\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$$

Akkor a **standardizált/normalizált mintaátlagok eloszlása** pedig **standard normális eloszlású** lesz:

$$\frac{\bar{y} - \mu}{\frac{\sigma}{\sqrt{n}}} = z \sim N(0, 1)$$

Standard normális eloszlás esetén pedig **mindig igaz**, hogy $P(-2 < z < +2) \approx 95\%$. Emlékeztetőként itt az ábra az $N(0, 1)$ standard normális eloszlás sűrűségszámítási görbéről az 1. heti tananyag 2.6. fejezetéből.



Most az egyszerűség miatt vegyük a \approx -ot $=$ -nek:

$$P(-2 < z < +2) = 95\%$$

Ebbe a fenti összefüggésbe beírjuk a képletet, amivel kiszámoltuk z -t:

$$P\left(-2 < \frac{\bar{y} - \mu}{\frac{\sigma}{\sqrt{n}}} < +2\right) = 95\%$$

Most egyelőre azzal a feltevéssel élünk, hogy ismerjük σ -t, azaz a sokasági szórást. Mondjuk pl. valami előzetes teljeskörű adatfelvételből. Célunk, hogy a valós, sokasági átlagot (μ -t) foglaljuk valamiféle határok közé egy darab mintaátlag (\bar{y}) ismeretében (hiszen majd nem akarjuk minden kivenni az összes lehetséges mintát). Tehát, az összefüggésből fejezzük a μ sokasági átlagot:

$$P\left(\bar{y} - 2 \times \frac{\sigma}{\sqrt{n}} < \mu < +2 \times \frac{\sigma}{\sqrt{n}}\right) = 95\%$$

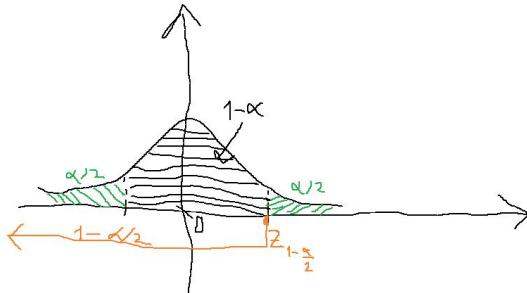
Tehát, ez a fenti összefüggés azt jelenti, hogy a **sokasági átlag az egy darab mintaátlag $\pm 2SH$ intervallumban van kb. 95%-os valószínűséggel**. Ezt hívjuk az **áttag 95%-os konfidencia-intervallumának**. A 2 pedig a 95%-os megbízhatósági szinthez tartozó k megbízhatósági szorzó. Mindezeket pedig csupán egy darab n elemű mintából ki is tudjuk számolni, ha σ -t helyettesítjük s -sel! Viszont fontos, hogy a mintánkat véletlenszerűen válasszuk ki, mert csak így kapunk a mintaátlagok eloszlására a látott normális eloszlást! Ugyebár a CHT-nak kellene a véletlen kiválasztású mintaelémek a „véletlen hatások összegződése” részhez.

6.3.1. Az átlag konfidencia-intervallumának általános alakja

Ha átalánosságban akarjuk felírni ezt a konfidencia-intervallumot, akkor úgy szoktunk fogalmazni, hogy $1 - \alpha$ megbízhatóságú intervallumot írunk fel, ahol α a **hibázásunk valószínűsége**, tehát, annak a valószínűsége hogy a sokasági átlag mégsem a konfidencia-intervallumban van:

$$P\left(\bar{y} - z_{1-\frac{\alpha}{2}} \times \frac{\sigma}{\sqrt{n}} < \mu < +z_{1-\frac{\alpha}{2}} \times \frac{\sigma}{\sqrt{n}}\right) = 1 - \alpha$$

Itt a z érték azt akarta jelenteni, hogy azt a **z értéket kell levadászní**, ami esetén az „alá esés” valószínűsége a standard normális eloszlásban épp $1 - \frac{\alpha}{2}$. Ennek okát szemlélteti az alábbi ábra.



Ugyebár a cél az, hogy a $z \sim N(0, 1)$ eloszlásban megtaláljuk azt a k értéket, ahol $P(-k < z < +k) = 1 - \alpha$. Ugyebár a 95% esetén is innen kaptuk a

2-t. Ezt pedig akkor úgy kapjuk meg a **fenti ábra alapján**, hogy ha tudjuk, hogy a $\pm k$ közé esés valószínűsége $1 - \alpha$, akkor a tartományon **kívülre esés valószínűsége α** , a hibázásunk megengedett valószínűsége. Ami a normális eloszlás sűrűségfüggvényének **szimmetriája** miatt **egyenlően oszlik el** $-k$ alá és $+k$ felé. Tehát a $+k$ felé esési valószínűsége $\alpha/2$. De mivel a pitonban a stats csomag `norm.ppf` függvénye csak „alá esési” valószínűségekből **dolgozik**, így a $+k$ alá esés valószínűséget kell tudnunk, ami a felé esés komplementere, azaz $1 - \frac{\alpha}{2}$, ami az **ábrán a narancssárga rész területe a sűrűségfüggvényben**.

Láthatjuk a logika szuperül működik a 95%-os megbízhatóság, azaz $\alpha = 5\%$ esetére.

```
alfa = 0.05
stats.norm.ppf(1-(alfa/2))

## 1.959963984540054
```

Az eredmény nem pontosan 2, hanem kb. 1.96. Ugyebár a **levezetésben kerekítettem**, de a megbízhatósági szorzó számítás lényege szerintem átjött. :)

Szumma-szummárum. Az átlag tetszőleges megbízhatóságú intervallumbecslése / konfidencia-intervalluma általános alakban a következő módon számítható:

$$\bar{y} \pm k \times SH$$

Tehát, a **mintaátlagra (\bar{y}) rámérjük \pm a standard hiba k -szorosát**, ahol a k , mint megbízhatósági szorzó állítja be a kívánt megbízhatósági szintet. Ez az általános formula azért nagyon fontos, mert a későbbiekkben az átlagra vonatkozó becslések során minden csak annyit változtatunk rajta, hogy az SH és a k mögött álló konkrét képlet fog csak változni, de ez a fenit alaplogika végig megmarad!!

6.3.2. A konfidencia-intervallum megbízhatóságának ellenőrzése

Utolsó lépésben ellenőrizzük le konfidencia-intervallumos formulánk működését, és **nézzük meg, hogy a σ -val számolt SH -t használva tudunk-e egy 98%-os megbízhatóságú intervallumbecslést készíteni a Balatont átúszók időeredményeinek átlagára a $k = z_{1-\frac{\alpha}{2}}$ megoldásunkkal**.

Tehát a 10000 db mintánk **mintaátlagára a mostani helyzetben a**

$$\Delta = k \times SH = z_{1-\frac{\alpha}{2}} \times \frac{\sigma}{\sqrt{n}}$$

távolságot kell \pm felmérni. Ezt a \triangle távolságot hívjuk a **konfidencia-intervallum hosszának**, vagy másnéven a **becslés teljes hibahatárának**.

Nézzük akkor meg, hogy egy ilyen becslés tényleg kb. 98%-os találati arányt eredményez-e?

```

n = 100
alfa = 1 - 0.98
k = stats.norm.ppf(1-alfa/2)
SH = SokasagiSzoras / np.sqrt(n)
delta = SH * k

MintaVetelek100Elem['AtlagAlsoHatar'] = MintaVetelek100Elem['MintaAtlagok'] - delta
MintaVetelek100Elem['AtlagFelsoHatar'] = MintaVetelek100Elem['MintaAtlagok'] + delta
MintaVetelek100Elem.iloc[:,99:103]

##           Elem100   AtlagAlsoHatar   AtlagFelsoHatar   MintaAtlagok
## 0      165.883333    161.005186    181.518147    171.261667
## 1      128.216667    147.770020    168.282980    158.026500
## 2      215.633333    162.022353    182.535314    172.278833
## 3      112.450000    153.177686    173.690647    163.434167
## 4      99.916667    156.296353    176.809314    166.552833
## ...
## 9995  128.183333    153.425853    173.938814    163.682333
## 9996  209.300000    154.076353    174.589314    164.332833
## 9997  183.400000    153.445853    173.958814    163.702333
## 9998  196.050000    156.167353    176.680314    166.423833
## 9999  135.166667    162.793853    183.306814    173.050333
##
## [10000 rows x 4 columns]

```

Oké, akkor **megvannak az új intervallumbecsléseink** mind a 10000 mintára. Lássuk a pontosságukat!

```

MintaVetelekSzama = len(MintaVetelek100Elem)

np.sum((MintaVetelek100Elem['AtlagAlsoHatar'] < SokasagiAtlag) & (MintaVetelek100Elem['AtlagFelsoHatar'] > SokasagiAtlag)) / MintaVetelekSzama
## 0.9782

```

És tényleg kb. 98% a találati arány, győzelem! :)

6.3.3. 3.3. A konfidencia-intervallum két fontos tulajdonsága

A konfidencia-intervallum **megbízhatósági szintjével** csinján kell bánna. Ha megfigyeljük a korábbi számításainkat, akkor láthatjuk, hogy

- 95%-os megbízhatósághoz $k = 1.96$
- 98%-os megbízhatósághoz viszont már $k = 2.3$

megbízhatósági szorzó tartozik.

A $\Delta = k \times SH$ összefüggés miatt pedig könnyű látni, hogy **megbízhatósági szint növelésével a becslési hibahatár nő, azaz a konfidencia-intervallum tágul**. Teljesen logikus: ha **nagyobb találati arányt akarok, akkor „növelni kell a hálót”, így nagyobb eséllyel akad fenn rajta a sokasági átlag**. 100%-os megbízhatóság pedig egy esetben van, ha az intervallumbecslésünk a $\pm\infty$ tartomány, ami ugyebár nem túl hazsnos becslési intervallum... :)

Érdemes kipróbálni a dolgot még az 1. fejezetben kivett 100 elemű mintán mondjuk $\alpha = \{0.2, 0.1, 0.05, 0.01, 0.001\}$ hibavalószínűségek mellett egy **for** ciklussal. A számoláshoz felhasználom a korábban kiszámolt **MintaAtlag** és **SH** objektumokat.

```
alfa_lista = [0.2, 0.1, 0.05, 0.01, 0.001]

for aktualis_alfa in alfa_lista:
    also_hatar = MintaAtlag - SH * stats.norm.ppf(1-aktualis_alfa/2)
    felso_hatar = MintaAtlag + SH * stats.norm.ppf(1-aktualis_alfa/2)
    print(
        "Megbízhatóság: "+str((1-aktualis_alfa)*100)+"% - Konf. Int.: ["+
        str(round(also_hatar,2))+", "+str(round(felso_hatar,2))+" ]")

## Megbízhatóság: 80.0% - Konf. Int.: [158.79, 170.09]
## Megbízhatóság: 90.0% - Konf. Int.: [157.19, 171.69]
## Megbízhatóság: 95.0% - Konf. Int.: [155.8, 173.08]
## Megbízhatóság: 99.0% - Konf. Int.: [153.08, 175.8]
## Megbízhatóság: 99.9% - Konf. Int.: [149.93, 178.95]
```

Szépen megfigyelhető a leírt jelenség: a **megbízhatóság növelésével a konfidencia-intervallum egyre csak tágul, azaz a becslési hibahatár folyamatosan nő**.

- 90% megbízhatóság esetén az átlagos időeredményt még valahova 157 és 172 perc közé tippeljük,
- 99% megbízhatósánál viszont már 153 és 176 perc közé!

A jelenséget mérsékelni a **mintaelemszám növelésével lehet! Nézzük meg az előző for ciklust egy $n = 20$ elemű mintán az $n = 100$ helyett!** Számoljuk ki az első 20 oszlop alapján a \bar{y} mintátlagot. Mivel a kiválasztás FAE volt, így olyan lesz a dolog, mintha csak 20 elemet választottunk volna ki

a mintavétel során, nem pedig 100-at. Az $SH = \frac{\sigma}{\sqrt{n}}$ is könnyen újraszámolható $n = 20$ mellett.

```
MintaAtlag20Elem = np.mean(BalcsiMinta.PERC[0:20])
n = 20
SH20Elem = SokasagiSzoras / np.sqrt(n)

alfa_lista = [0.2, 0.1, 0.05, 0.01, 0.001]

for aktualis_alfa in alfa_lista:
    also_hatar = MintaAtlag20Elem - SH20Elem * stats.norm.ppf(1-aktualis_alfa/2)
    felso_hatar = MintaAtlag20Elem + SH20Elem * stats.norm.ppf(1-aktualis_alfa/2)
    print(
        "Megbízhatóság: "+str((1-aktualis_alfa)*100)+"% - Konf. Int.: ["+
        str(round(also_hatar,2))+", "+str(round(felso_hatar,2))+"] ")

## Megbízhatóság: 80.0% - Konf. Int.: [153.24, 178.5]
## Megbízhatóság: 90.0% - Konf. Int.: [149.66, 182.09]
## Megbízhatóság: 95.0% - Konf. Int.: [146.55, 185.19]
## Megbízhatóság: 99.0% - Konf. Int.: [140.48, 191.26]
## Megbízhatóság: 99.9% - Konf. Int.: [133.43, 198.31]
```

Szépen láthatjuk, hogy az **átlagos átúszási időket 99%-os megbízhatósággal**

- $n = 20$ esetben 140 és 191 perc közé tesszük,
- $n = 100$ esetben pedig láttuk az előbb, hogy a becslés pontosabb (kisebb \triangle hibahatárú): 153 és 176 perc közé teszi a sokasági átlagidőt.

Nem meglepő az eredmény. Mivel a $\frac{\sigma}{\sqrt{n}}$ standard hiba **képlet nevezőjében van az n elemszám**, így növelése csökkenti a standard hibát, ezen keresztül pedig a teljes \triangle becslési hibahatárt. Ugyebár az 5. fejezetben megállapítottuk, hogy az átlag **konzisztenzs becslés**: elemszám növekedésével a SH -ja csökken, a 0-ra tart.

Emiatt a **választott $1 - \alpha$ megbízhatósági szint a mintaelemszám függvénye**:

- Nagyobb n elemszám esetén egy 99%-os megbízhatóság is elég pontos intervallumbecslést szolgáltathat,
- Kisebb mintaméret esetén valószínűleg meg kell elégedni valami moderáltható (pl. 90% – 95%) megbízhatósági szinttel is.

6.4. Intervallumbecslés a gyakorlatban

Ezen a ponton engedjük el a Balaton átúszókat, és próbáljuk ki az átlag konfidencia intervallum számítást olyan esetben, ahol nem ismerjük a teljes sokaságot, amiből mintát vettünk.

1. feladat: Altatók hatékonysága

A sztorink a következő.

Egy gyógyszergyár egy új altató készítmény hatását vizsgálja 10 véletlenszerűen kiválasztott inszomniában szenvedő páciensen. Mind a tíz páciens esetében feljegyezték, hogy hány órát növekedett az alvásidőük a készítmény használatát követően. Korábbi klinikai vizsgálatok alapján ismeretes, hogy az altató készítmények által kiváltott alvásidő-változás normális eloszlású, 2 óra szórással.

$$\text{Adatok} = \{1.9, 0.8, 1.1, 0.1, -0.1, 4.4, 5.5, 1.6, 4.6, 3.4\}$$

Készítsünk 95%-os megbízhatósággal intervallumbecslést a várható átlagos alvásidő-változásra:

- a) a megadott feltételek alapján;
- b) feltételezve, hogy az eloszlás normális, de a szórás ismeretlen!
- c) Mekkora mintára van szükség, ha ugyan ekkora megbízhatóság (95%) mellett a b) pontban kapott hibahatárt a felére kívánjuk csökkenteni?

1/a) feladat megoldás

Ebben az a) feladatban nagyon el vagyunk kényeztetve. Az altató hatékonyságához van egy $n = 10$ elemű mintánk, aminek az adatait tételesen ismerjük. Első páciens alvásidője 1.9 órával nőtt az altató használata után, másodiké 0.8 órával, stb. Van egy páciens, akinek csökkent az alvásidője a gyógyászerhasználat után: az 5. delikvensé, 0.1 órával. Ha ezeket az adatokat elrakjuk egy numpy tömbbe, akkor simán kiszámolható a megfigyelt 10 páciens esetében az átlagos alvásidő növekedés, azaz \bar{y}

```
MintaAdatok = np.array([1.9, 0.8, 1.1, 0.1, -0.1, 4.4, 5.5, 1.6, 4.6, 3.4])
MintaAtlag = np.mean(MintaAdatok)
MintaAtlag
```

```
## 2.3299999999999996
```

Tehát a megfigyelt páciensek esetében az átlagos alvásidő növekedés kb. $\bar{y} = 2.3$ óra. Ami szép és jó, de mennyi lehet az átlagos alvásidő növekedés az inszomniás páciensek összességére, a teljes sokaságra nézve? Ehhez kell a konfidencia-intervallum! Hogy olyan betegekre is tudjunk mondani valamit, akiket a mintában NEM figyeltünk meg!

Itt az a) esetben minden feltételezést elfogadhatunk, amit a feladat tesz. Ebben van egy olyan rész, ami szerint „korábbi kutatásokból” ismerjük, hogy az alvásidő-változások szórása 2. Ha ezt így elhíssük, akkor azt mondhatjuk, hogy az alvásidő-változások teljes sokaságra vonatkozó szórását vehetjük 2-nek. Azaz, $\sigma = 2$ -t „hazudunk” a számolások során. **FIGYELEM!** Ha a feladat szövege azt akarja közölni velünk, hogy van egy sokasági szórás, egy σ , amit ismerünk és használhatunk a számolásaink során, akkor azt mindig ilyen „korábbi kutatásokból ismeretes...” szövegrészbe fogja becsomagolni!

Ha elfogadjuk a feltételezéseinket, akkor minden adott a konfidencia-intervallum 3. fejezetben megismert képletének alkalmazásához. Hiszen, ha 95% a megbízhatóság, akkor $\alpha = 5\%$. Tegyük is ezt: alkalmazzuk a képleteket az adatainkra!

```
n = 10
alfa = 1-0.95
szigma = 2

k_szorzo = stats.norm.ppf(1-alfa/2)
SH = szigma/np.sqrt(n)
delta = k_szorzo*SH

also = MintaAtlag - delta
felso = MintaAtlag + delta

[also, felso]

## [1.0904099353908765, 3.5695900646091228]
```

Az eredmény alapján ez az új alatató készítmény az inszomniás páciensek teljes sokaságában a 10 elemű mintánk alapján legalább 1.09 óra és legfeljebb 3.6 óra alvásidő növekedést okoz 95%-os valószínűsséggel!

Ennek a cég vezetése nagyon örül, mert lehet olyan reklámslogoneket elsütni az eredményünk alapján, hogy „vizsgálatok igazolják, hogy altatónk 95%-os valószínűsséggel legalább 1 órával növeli a várható alvásidőt”. Egy ilyen mondat pedig minden marketinges álma úgymond. :)

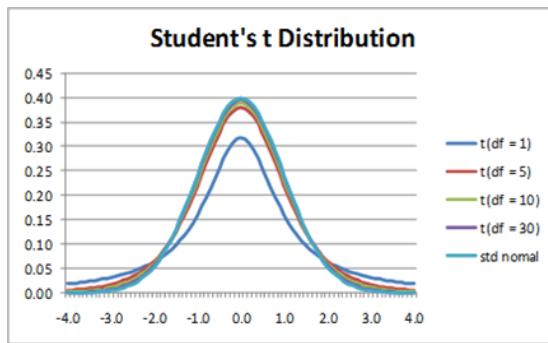
De minden ilyen szép marad-e ha az alvásidők szórását nem a feltételezett $\sigma = 2$ -vel, hanem a mintából számolt korrigált szórással (s) számítjuk?

1/b) feladat megoldás

Itt a feladat azt mondja, hogy ne higyjunk mindenféle kétes „korábbi kutatásnak”, ne fogadjuk el az általuk megadott σ -t, hanem **számoljunk magunknak egy korrigált mintaszórást**, azaz s -t (a korrigált mintaszórás ad torzítatlan becsést a valós, sokasági σ -ra), és számoljuk végig azzal a stanadard hibát $SH = \frac{s}{\sqrt{n}}$ módon.

Viszont, ha feloldjuk azt a feltevésünket, hogy ismerjük a sokasági szórást az SH számoláshoz, akkor a mintaátlagok normális helyett egy $n - 1$ szabadságfokú t-eloszlást követnek.

A t-eloszlás sűrűségsfüggvénye az alábbi alakot ölti. Az szabadságfokokat az angol *degrees of freedom* kifejezésből df -el jelöljük.



Ahogy a fenti sűrűségsfüggvényből is látszik Student-féle t-eloszlás valójában egy **ellapított standard normális eloszlás** (lásd fenti ábra). A lapítás azt akarja kifejezni, hogy az eloszlás szórása nagyobb. Hiszen nagyobb szórás esetén az eloszlás szélein lévő számértékek is nagyobb valószínűséggel következhetnek be (mivel a lapítás miatt a sűrűségsfüggvény magasabban fut ezeken a helyeken), így változatosabbá, jobban szóródóvá teszik az adatsorunk.

Viszont, ahogy növeljük az eloszlás szabadságfokát (df), egyre jobban „visszacsúsítjuk” az eloszlást a standard normális eloszlásba. Logikus, hogy ilyenkor ezt az eloszlást használjuk, mivel a standard hiba értékébe (ami a mintaátlagok normális eloszlásának szórás paramétere) egy biztosan ismert sokasági szórás érték helyett, annak egy mintából számított becslését rakjuk, így **nagyobb bizonytalanságot, nagyobb szórást viszünk az eloszlásba**.

Ebben a helyzetben a konfidencia-intervallum úgy módosul, hogy σ helyére s kerül a SH -ban, és a k megbízhatósági szorzót t-eloszlásból számoljuk $N(0, 1)$ eloszlás helyett:

$$P\left(\bar{y} - t_{1-\frac{\alpha}{2}}^{n-1} \times \frac{s}{\sqrt{n}} < \mu < +t_{1-\frac{\alpha}{2}}^{n-1} \times \frac{s}{\sqrt{n}}\right) = 1 - \alpha$$

Az $n - 1$ szabadságfokú t érték számításához a `scipy` csomag `t.ppf` függvényét vesszük elő. Teljesen hasonló logikával működik, mint a `norm.ppf` függvény

(vagy mint a `scipy` bármelyik eloszlás inverz értékét számoló függvénye). A t -eloszlás is szimmetrikus, tehát most is azt az értéket keressük az $n - 1$ szabadságfokú t -eloszlásunkban, ahol az „alá esési” valószínűség $1 - \frac{\alpha}{2}$. A `t.ppf` függvény `df` paraméterével állítható a szabadságfok. Ezt a mi $n = 10$ elemű mintánkra, 95%-os megbízhatósági szint mellett az alábbi módon számoljuk

```

n = 10
alfa = 1-0.95

k_szorzo_z = stats.norm.ppf(1-alfa/2)
k_szorzo_t = stats.t.ppf(1-alfa/2, df = (n-1))

[k_szorzo_z, k_szorzo_t]

## [1.959963984540054, 2.2621571628540993]

```

Láthatjuk, hogy a t -eloszlású k szorzó értéke érdemben nagyobb, mint a standard normális eloszlású k szorzóé ugyan arra a megengedett hibavalószínűségre! Mivel a t -eloszlás nagyobb valószínűséget tulajdonít az extrém magas+alacsony értékek bekövetkezésének, így ha ezt alkalmazzuk, akkor ugyan ahhoz a megbízhatósági szinthez egy magasabb k megbízhatósági szorzót kapunk a SH -hoz!

Ezek után igazából csak annyi a feladatunk, hogy a teljes becslési hibahatárt kiszámítsuk $\Delta = t_{1-\frac{\alpha}{2}}^{n-1} \times \frac{s}{\sqrt{n}}$ módon, és ezt rámérjük \pm a mintaátlagra, \bar{y} -ra. Ez már mehet ugyan úgy, mint az a) feladatban.

```

korr_szoras = np.std(MintaAdatok, ddof = 1)

k_szorzo_t = stats.t.ppf(1-alfa/2, df = (n-1))
SH = korr_szoras/np.sqrt(n)
delta = k_szorzo_t * SH

also = MintaAtlag - delta
felso = MintaAtlag + delta

[also, felso]

## [0.8976775393413148, 3.7623224606586847]

```

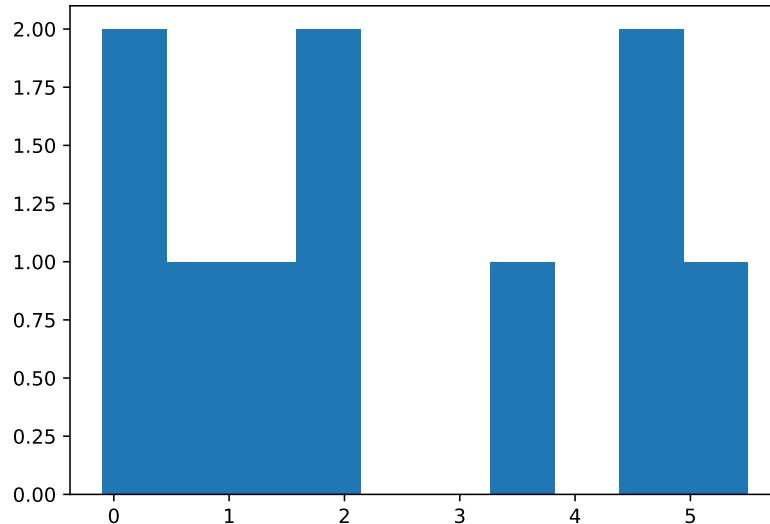
Nem meglepő módon, a **t -eloszlású megbízhatósági szorzó miatt a 95%-os kinfidencia-intervallum kitágult**. Annak ellenére is, hogy az SH -ban gyakorlatilag nincs változás, mivel $s = 2.0022$. Az altató átlagos alvásidő növekedése a betegek teljes sokaságban, már 0.9 és 3.76 óra közé tehető 95%-os

valószínűséggel. Szóval, ebben a reálisabb helyzetben, amikor már a szórást magunknak számoljuk a mintaadatokból, és nem pedig „*elhisszük*” valakinek, akkor a bizonytalanság megnövekedése miatt alkalmazott t-eloszlásnak köszönhetően, a becslési hibahatár, a Δ kitágul. A marketingesek pedig már nem mondhatnak olyan szépeket, hogy „95%-os valószínűséggel legalább átlag 1 órát növeli az alvásidőt az altató”. RIPSZ! :(

Viszont, amíg a **mintánk elemszámi kicsi**, $n \leq 30$, addig a **t-eloszlású konfidencia intervallum számításának van egy előfeltétele: az adatsor, amiből a mintát vesszük normális eloszlású kell, hogy legyen!!** Esetünkben ez annyit tesz, hogy az inszomniás páciensek sokaságában az *alvásidő normális eloszlású* hisztogramot mutat.

Na, most itt ezt kemény $n = 10$ esetben marha nehéz értelmesen megvizsgálni a minta alapján, de lessünk egy hisztogramot a megfigyelt 10 db alvásidőre! A hisztogram osztályközeinek számát most nem optimalizáljuk, elfogadjuk az alapbeállításokat.

```
plt.hist(MintaAdatok)
plt.show()
```

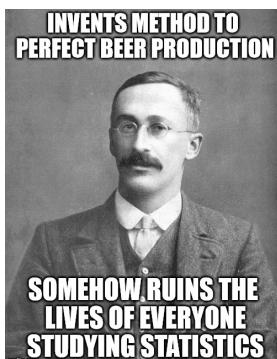


Hát, a fene se tudja ennyiből eldönteni, de tényleg. Akár normális eloszlásúak lehetnek az alvásidők. :) Bízzunk a CHT-ben: az egyéni alvásidők valószínűleg véletlen hatások összegeként állnak elő, így azok adatsora a sokaságban, a nem megfigyelt 10 elemű mintán *kívüli* világban, lehet normális eloszlású is akár. :)

Maga a *t-eloszlás* egyébként egy *William Gosset* nevű angol statisztikus kreatúrája. Mr. Gosset a *Guiness* sörgyárak minőségbiztosítási vezetője volt, és azt a feladatot kapta, hogy oldja meg, hogy az üveges sörök minőségbiztosítása kis mintákból is megoldható legyen. Mivel a minőségbiztosításhoz ki kell bontani az üveget és megmérni benne az összetevőket, aztán utána ezek átlagára néznek konfidencia-intervallumokat. A minőségbiztosításra kinyitott sör pedig már nem elfogyasztható. *Kellemetlen*. Így érhető, hogy minél kevesebb üvegből akarják az egész minőségbiztit megúszni. Erre adta megoldásként Gosset a *t-eloszlást*. Mivel a sörökben a különböző alapanyagok értéke normális eloszlást követ.

Azban emberünk el akart büszkélkedni a saját kis eloszlásával a nagy világban, ezért *Student álnéven* publikálta is az eredményeit...innen lett az eloszlás neve *Student-féle t-eloszlás*. :) Viszont nagyon nem volt előrelátó az úriemberünk, hiszen akkoriban nyilván csak a *Guiness* sörgyár tudott elég kevés üvegből ugyan olyan „jó” megbízhatósággal minőségbiztosítani, mintha több elemű mintát vizsgáltak volna, így *Mr. Gosset* gyorsan lebukott.

Tehát, ahogy az alábbi ábra is mutatja, szegény emberünk csak a sörtermelést szerette volna hatékonyabbá tenni, de végül mindenki életét tönkretette, aki statisztikai becsléselméletet tanul. :)



1/c) feladat megoldás

Node, térjünk vissza az altató készítményünk hatékonyságára. Ugyebár a *t-eloszlással* készített konfidencia-intervallum eléggé elkeserítő eredményt hozott: nem tudjuk azt mondani 95%-os megbízhatósággal, hogy készítményünk legalább 1 órával növeli az alvásidőt!

Mi ennek az oka? Hát a **magas becslési hiba, az átlagra épített konfidencia-intervallum hossza**, tehát a Δ . Ez most ugye nekünk ± 1.43 óra.

```
delta
```

```
## 1.4323224606586848
```

Mondjuk ezt a becslési **hibahatárt szeretnénk levinni a felére**, 1.43/2, azaz kb. $\Delta' = \pm 0.7$ órára. Ezt kétféleképpen lehet elérni. Hiszen „*magasról nézve*”, a becslési hibahatár egy kéttényezős szorzat

$$\Delta = k \times SH$$

1. Addig állítgatom a megbízhatósági-szintet, azaz valójában α -t, amíg a t-eloszlásból származó k megbízhatósági szorzó olyan alacsony nem lesz, hogy az eddigi SH -val szorozgatva ki nem adja a 0.7-es értéket Δ -re. Ez az **illetlen megoldás!** Hiszen lehet, hogy ehhez az α -t nagyon fel kell engedni, és a becslésnek nagyon alacsony lesz a megbízhatósága...az „55%-os megbízhatósággal állítható az, hogy...” nem hangzik olyan jól marketing szempontból sem.
2. Az **intelligens megoldás** a mintaelemszám, azaz n növelése. Az α -t adottnak vesszük, és így k -t nem bántjuk. Persze a t-eloszlás szabadságfoka miatt az elemszám k -ra is hat, de ahogy az eloszlás sűrűségfüggvényéből is látszik, kellően nagy n esetén a t-eloszlás igazából a standard normálissal lesz ekvivalens, tovább tehát nem tudjuk csökkenteni k -t adott α mellett. Tehát ekkor a k lényegében fix. Ellenben n növekedése miatt az SH része a szorznak biztosan csökkeni fog, hiszen az átlagra konzisztens becslést adunk, tehát SH a 0-ba tart n növelése esetén. Ez ugye a $SH = \frac{s}{\sqrt{n}}$ összefüggésből adódik egyértelműen. Szóval az s -t (vagy éppen σ -t, ha azt ismerjük) adottnak vesszük, akkor kiszámolhatjuk, hogy makkora n szükséges a $\Delta' = \pm 0.7$ óra eléréséhez hibahatár fronton.

Szóval, a 2. megoldást, az elemszám növelését választva a következő összefüggésből ki kell fejezni n -t:

$$\Delta = k \times SH = z_{1-\frac{\alpha}{2}} \times \frac{s}{\sqrt{n}}$$

Hiszen Δ helyére beírhatjuk nemes egyszerűséggel az elérni kívánt Δ -t. A k helyére azért írtam $z_{1-\frac{\alpha}{2}}$ -t és nem a t-eloszlású szprzót, mert a standard normális eloszlású megbízhatóságú szorzó értéke alá úgysem tudunk menni a t-eloszlással adott α mellett, hiszen a t-eloszlású sűrűségfüggvényt végtelen szabadságfok mellett is csak a standard normális eloszlásba lehet maximum „*visszacsúcsosítani*”, ahogy a b) feladat megoldásában szereplő ábrán láthattuk is.

Ha pedig a fent formulából kifejeztük az elemszámot, akkor ehhez az összefüggéshez jutunk:

$$n = \frac{z_{1-\frac{\alpha}{2}}^2 \times s^2}{\Delta^2}$$

Ebbe pedig gond nélkül be tudunk helyettesíteni minden.

```
delta_uj = 0.7
n_uj = k_szorzo_z**2 * korr_szoras**2 / delta_uj**2
n_uj
## 31.429404922781128
```

Tehát a becslési ha felezéséhez $n = 31.4$ elemű minta kéne... mivel egyik betegből sem vágnánk ki 0.4 részt a vizsgálatra, így logikus módon az eredményt kerekítsük fel $n = 32$ -re. **Azaz**, $32 - 10 = 22$ páciensen kéne még extrában megvizsgálni az altató hatását ahhoz, hogy az átlagos alvásidő növekedést 95%-os megbízhatósággal és ± 0.7 órás becslési hibával meg lehessen adni. Persze, ha a mintaátlag vagy éppen a korrigált mintaszórás úgy módosul az új mintaelemek hatására, hogy a konfidencia-intervallum alsó határa továbbra sem éri el a vágyott 1 órát, akkor tervünk dugába dölt. Viszont, akkor ez inkább a gyógyszert fejlesztő vegyészek sara, mint a hatékonyságvizsgálatot végző statisztikusé, aki rávilágít a problémákra. :)

2. feladat

Az ESS2020.xlsx fájlban található adatbázis a 2020-ban végzett európai szociális felmérés (European Social Survey 2020 = ESS2020) 1849 magyar kitöltjének válaszait tartalmazza 14 kérdésre (plusz van egy *id* oszlop).

Ha valamelyik oszloban üres értéket találunk, akkor az adott sorban lévő kitöltő nem válaszolt a kérdésre. Az adatbázisban szereplő kitöltők a teljes 18 év feletti magyar népességből vett véletlen mintaként kezelhetők. Most feltesszük, hogy ez a véletlen minta visszatevéses, azaz *FAE* is. A 6. heti tananyagban látni fogjuk, hogy ez nem is valóságtól elrugászkodott feltevés.

Az adatbázis nyers formában eről a linkről elérhető egy ingyenes regisztráció után.

Feladatunk a mintavétel alapján megbecsülni, hogy hogy egy átlagos magyar polgár 97%-os megbízhatósággal várhatóan legalább és legfeljebb hány percet internetezik naponta!

2. feladat megoldás

Először is töltök be az adatbázist `pandas` data frame-be és nézzük meg az elemszámot a feladat szempontjából releváns `NetUsePerDay_Minutes` oszlopban!

```
ESS = pd.read_excel("ESS2020.xlsx")
ESS.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 1849 entries, 0 to 1848
## Data columns (total 15 columns):
## #   Column           Non-Null Count  Dtype  
## ---  --  
## 0   id               1849 non-null    int64  
## 1   PoliticalRadioTVPerDay_Minutes 1796 non-null    float64 
## 2   NetUsePerDay_Minutes          1099 non-null    float64 
## 3   TrustInParlament          1849 non-null    object  
## 4   PoliticalPartyPref         1849 non-null    object  
## 5   Education_Years           1830 non-null    float64 
## 6   WeeklyWork_Hours          685 non-null    float64 
## 7   Region                  1849 non-null    object  
## 8   County                  1849 non-null    object  
## 9   SecretGroupInfluenceWorldPol 1849 non-null    object  
## 10  ScientistsDecievePublic  1849 non-null    object  
## 11  COVID19                 1849 non-null    object  
## 12  ContactCOVID19          1849 non-null    object  
## 13  GetVaccince             1849 non-null    object  
## 14  SomeContactCOVID19       1849 non-null    object  
## dtypes: float64(4), int64(1), object(10)
## memory usage: 216.8+ KB

n = ESS.NetUsePerDay_Minutes.count() # így nem számoljuk be az üres értékeket
n

## 1099
```

Szóval, az üres értékeket nem számolva, a releváns oszlop `count` metódusával megtudhattuk, hogy az internetezési idő esetében $n = 1099$ elemű mintából főzhetünk. **Ez bőven nagy minta**, mivel ebből a tárgyból közmegegyezéssel az $n > 30$ eseteket **nagy mintának** tekintjük, ahogy az 1/b) feladat megoldásában megadtuk. Emiatt **nem kell az intrenetezési idők normális eloszlását sem vizsgálni az intervallumbecslés elvégzéséhez s-t és t-eloszlást használva!**

Mehetünk simán előre a $\Delta = t_{1-\frac{\alpha}{2}}^{n-1} \times \frac{s}{\sqrt{n}}$ képletet használva.

```

n = ESS.NetUsePerDay_Minutes.count() # így nem számoljuk be az üres értékeket

minta_atlag = np.mean(ESS.NetUsePerDay_Minutes)
korr_szoras = np.std(ESS.NetUsePerDay_Minutes, ddof = 1)

SH = korr_szoras / np.sqrt(n)

alfa = 1-0.97
k_szorzo_t = stats.t.ppf(1-alfa/2, df = (n-1))

delta = k_szorzo_t * SH

also = minta_atlag - delta
felso = minta_atlag + delta

[also, felso]

## [171.77362310362497, 190.81418399373626]

```

Tehát, az $n = 1099$ elemű mintánk alapján azt mondhatjuk, hogy az átlagos magyar 97%-os megbízhatósággal legalább napi 171.8 percet és legfeljebb napi 190.8 percet tölt internevezéssel.

Az eredményünk nem nagyon változik, ha a k megbízhatósági szorzót standard normális eloszlásból számoljuk t-eloszlás helyett, hiszen az n olyan nagy, hogy az $n - 1$ szbadságfokú t-eloszlás sűrűségfüggvénye lényegében semmiben nem fog különbözni a standard normális eloszlás sűrűségfüggvényétől.

```

alfa = 1-0.97
k_szorzo_z = stats.norm.ppf(1-alfa/2)

delta = k_szorzo_z * SH

also = minta_atlag - delta
felso = minta_atlag + delta

[also, felso]

## [171.785998274992, 190.80180882236922]

```

Láthatjuk, hogy csak a tizedesjegyekben vannak nagyon minimális eltérések. Szóval **nagy mintaelemszám esetén a megbízhatósági szorzó nyugodtan számolható standard normális eloszlásból is a t-eloszlás helyett**.

Szerencsénkre, a `scipy` csomagban az átlag konfidencia-intervallumának számítására standard normális és t-eloszlású k szorzók esetében is **vannak**

beépített függvények. Ezek értelemszerűen `stats.norm.interval` és `stats.t.interval` nevekre hallgatnak.

- A függvények `scipy` paraméterében adjuk meg a **megbízhatósági szintet**, tehát NEM a „rendes” statisztika által α -val jelölt megengedett hibavalószínűséget!!! Ezt sajnos **meg kell szokni!!**
- A `loc` paraméterben érkezik a **mintaátlag**.
- A `scale` paraméterbe a **standard hibát rakjuk**, ami az **adatokból közvetlenül** is számítható a `stats.sem` függvénnyel.
- Csak a `stats.t.interval` esetén kell használni a `df` paramétert, ami értelemszerűen a szabadságfok helye.

Először nézzük meg, hogy tényleg használható a `stats.sem` függvény *SH* számításra. A `nan_policy = 'omit'` paraméterbe állíttassal szólunk neki, hogy a hiányzó értékeket ne vegye figyelembe a számítás során, ahogy mi is tettük.

```
SH_Adatokbol = stats.sem(ESS.NetUsePerDay_Minutes, nan_policy = 'omit')

[SH_Adatokbol, SH]
```

```
## [4.381340690645104, 4.381340690645097]
```

Egyezük, szuper! :)

Most pedig szépen használhatjuk a kétféle eloszlássa dolgozó konfidencia-intervallum számító függvényt is.

```
stats.norm.interval(
    confidence=0.97,
    loc=np.mean(ESS.NetUsePerDay_Minutes),
    scale=stats.sem(ESS.NetUsePerDay_Minutes, nan_policy = 'omit'))
```

```
## (171.78599827499198, 190.80180882236925)
```

```
stats.t.interval(
    confidence=0.97,
    loc=np.mean(ESS.NetUsePerDay_Minutes),
    scale=stats.sem(ESS.NetUsePerDay_Minutes, nan_policy = 'omit'),
    df = (ESS.NetUsePerDay_Minutes.count() - 1))
```

```
## (171.77362310362494, 190.8141839937363)
```

Egyezünk a korábbi eredményeinkkel, szuperek vagyunk! :)

6.5. Összefoglalás az átlag konfidencia-intervallumairól

Ezen a ponton érdemes összefoglalni, hogy az átlag intervallumbecslése milyen feltételek mellett milyen konkrét formulák vannak.

Az alapképlet a konfidencia-intervallum hosszára, azaz a becslési hibahatárra:

$$\Delta = k \times SH$$

1. A sokasági szórás, alias σ ismert.

- $k = z_{1-\frac{\alpha}{2}}$
- $SH = \frac{s}{\sqrt{n}}$

2. Az adatsor, amiből a mintát vettük *normális eloszlású*

- $k = t_{1-\frac{\alpha}{2}}^{n-1}$
- $SH = \frac{s}{\sqrt{n}}$

3. A mintánk elemszáma nagy: $n > 30$

- $k = t_{1-\frac{\alpha}{2}}^{n-1}$ vagy $k = z_{1-\frac{\alpha}{2}}$
- $SH = \frac{s}{\sqrt{n}}$

És ennyi, igazából, ha végiggondoljuk a 4. fejezet feladatait, ezt a 3 esetet tudjuk elkülöníteni a Δ számítása során.

6.6. Esettanulmány

Vizsgáljuk meg 99%-os megbízhatósággal hogyan alakul az átlagos internevezéssel töltött idő a teljes magyar népességen pártpreferencia szerint!

Érdekes megnézni, hogy egy $n = 1099$ elemű mintában mért átlagos különbségek az egyes pártok intenetezési idejében mennyire általánosíthatók ki a pártokat támogató népesség egészére, tehát azokra az emberekre, akiket a mintában még NEM figyeltünk meg, azaz a **mintaelemeken kívüli világra**.

Ehhez az út ott kezdődik, hogy a nominális ismérvünk, azaz a pártpreferencia egyedi értékei szerint vesszük az átlaggal vizsgált mennyiségi ismérv részátlagait, és ezen részátlagok SH -it. Ezt a kiinduló data frame `groupby` metódusával gyorsan meg tudjuk oldani.

```
# Először megtisztítjuk a 'NetUsePerDay_Minutes' oszlop üres értékeitől a data frame-t
df_szurt = ESS.loc[ESS.NetUsePerDay_Minutes.notna(), :]

PartPrefKimutatas = df_szurt.groupby("PoliticalPartyPref").agg(
    Elemszam = ('NetUsePerDay_Minutes', 'count'),
    Atlag = ("NetUsePerDay_Minutes", np.mean),
    SH = ("NetUsePerDay_Minutes", stats.sem)
)
```

<string>:2: FutureWarning: The provided callable <function mean at 0x000002140ED913>

PartPrefKimutatas

	Elemszam	Atlag	SH
## PoliticalPartyPref			
## Egyesült Ellenzék	182	228.791209	12.500299
## Egyéb	13	129.230769	20.489570
## Fidesz-KDNP	215	158.711628	8.225458
## Nem Ismert	689	176.776488	5.486396

Ha ezzel megvagyunk, akkor nekiállhatunk kiszámolni a 99% megbízhatóságú konfidencia-intervallumokat. Az *Egyéb* pártokat preferálók kivételével szép, nagy mintánk van mindenhol, így nincs nagy jelentősége, hogy t vagy *standard normális* eloszlásból számoljuk a k megbízhatósági szorzót. De mivel az *Egyéb* esetben csak $n = 13$, így ott biztonságosabb, ha a magasabb k -t eredményező $t(13 - 1)$ eloszlás alapján dolgozunk a konfidencia-intervallum számolásakor. Mivel a többi párt esetében ennek nincs jelentősége, így egyszerűbb az elétünk, ha a többi esetben a k -t a megfelelő $n - 1$ szabadságfokú t -eloszlásból számoljuk ki. Ezt a számolási lépést egyszerűen a `groupby`-al létrehozott data frame oszlopok bővítésével tudjuk megtenni.

alfa = 1-0.99

```
PartPrefKimutatas['ci_low_t'] = PartPrefKimutatas['Atlag'] - PartPrefKimutatas['SH'] * *
PartPrefKimutatas['ci_upp_t'] = PartPrefKimutatas['Atlag'] + PartPrefKimutatas['SH'] *
```

PartPrefKimutatas

	Elemszam	Atlag	SH	ci_low_t	ci_upp_t
## PoliticalPartyPref					
## Egyesült Ellenzék	182	228.791209	12.500299	196.249626	261.332792
## Egyéb	13	129.230769	20.489570	66.644566	191.816972
## Fidesz-KDNP	215	158.711628	8.225458	137.333678	180.089577
## Nem Ismert	689	176.776488	5.486396	162.605159	190.947816

Remek! :) Ezek alapján azt mondhatjuk, hogy az egyesült ellenzék esetén egy átlagos kérdőívkitöltő 228.8 percet netezik naponta, míg egy átlag Fidesz kitöltő csak napi 158.7 percet. Azonban, a teljes ellenzéki népességben az átlag netezési idő 196 és 261 perc között van 99%-os valószínűséggel. A teljes fideszes táborban pedig az átlagos neten töltött idő csak 137 és 180 perc között mozog, szintén 99%-os valószínűséggel. Tehát, 99%-os valószínűséggel az átlag netezési idő az Ellenzék esetében magasabb, mint a Fidesz esetében a teljes népességben is, hiszen a „legrosszabb” ellenzéki átlagérték is már 196 perc, míg a Fidesz esetén a „legjobb” átlagidő is csak 180 perc. Mindez persze a különböző felületeken (tv, rádió, internet) működő médiumok irányultságát figyelembe véve egyáltalán nem meglepő. :)

Ezek után telepítünk egy **statsmodels** néven futó csomagot!

```
pip install statsmodels
```

A **statsmodels** csomag szépsége, hogy van egy **dot_plot** című függvénye, amivel csudiszép vizuális megjelenítés adható a csoportátlagoknak és a konfidencia-intervallumaiknak. Ezeket az ábrákat **forest diagramoknak** hívják.

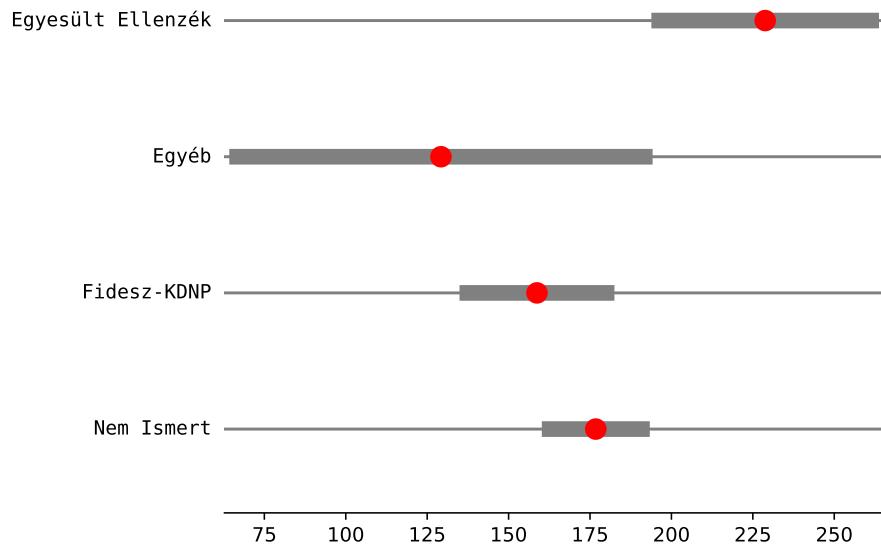
A **dot_plot** függvény **points** paraméterébe kerülnek a mintaátlagok, míg az **intervals** paraméterbe egy olyan 2 oszlopos data frame kerül, ami tartalmazza a konfidencia-intervallum alsó és felső határának távolságát a mintaátlaguktól abszolút értékként, azaz lényegében a Δ -t. Igen, itt egy olyan data frame kell neki, ahol minden oszlopban ugyan az az érték szerepel, mert ezt méri rá majd a gépállat a **points** paramétere, mint egy *sugár* \pm . Ahhoz, hogy szépen megjelenjenek a pártpreferenciák is, a függvénynek a **lines** paraméteren meg kell kapnia az eredménytábla soreindexeit is, hiszen a pártpreferencia értékek ott kerültek eltárolásra.

```
from statsmodels.graphics.dotplots import dot_plot

# konfidencia-intervallumok méretének megadása az ábrához
intervallumok = np.abs(PartPrefKimutatas[["ci_low_t", "ci_upp_t"]]) - PartPrefKimutatas[["Atlag"]]

dot_plot(
    points=PartPrefKimutatas["Atlag"],
    intervals = intervallumok,
    lines=PartPrefKimutatas.index)

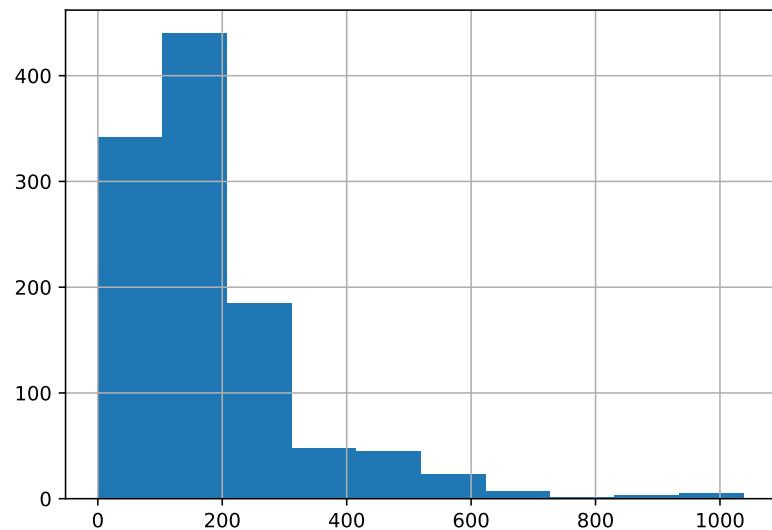
plt.tight_layout() # ne csússzanak ki a tengelyfeliratok
plt.show()
```



Mint láthatjuk, igazából az Ellenzék átlagos netezési ideje az, ami konfidencia-intervallumokkal együtt is magasabb, mint a többi párt szavazónak átlagos netezési ideje a teljes népességben. Sőt, a **többi párt esetében azt mondhatjuk, hogy egyáltalán nem biztos, hogy a teljes népességben ezek az átlag netezési idők különböznek 99%-os megbízhatósággal, mivel a konfidencia-intervallumok metszik egymást.** Szóval, hiába van a megfigyelt adatok körében a Fidesznek magasabb átlag netideje, mint az Egyéb pártoknak, de kevesebb, mint a Nem ismert eseteknek, a nem megfigyelt, mintán kívüli adatok körében, azaz a teljes népességben simán lehet (99%-os megbízhatósággal), hogy a Fideszes szavazók neteznek átlag a legkevesebbet egy napon. Ehhez csak annyi kell, hogy az a Fidesz valós, sokasági átlaga a konfidencia-intervallum alsó határa körül legyen a teljes népességben, míg a többi párt (nem ismert és egyéb) átlagideje pedig a konfidencia-intervallumuk tetejénél „kössön ki” a teljes népesség körében.

Érdemes még megfigyelni, hogy az Egyéb pártoknak a legszélesebb a konfidencia-intervalluma. Ez ugye azért van, mert mint láttuk a fejezet elején ide csak $n = 13$ kitöltő tartozik. Így az $SH = \frac{s}{\sqrt{n}}$ összefüggés miatt érthető, hogy nagyon bizonytalanok vagyunk ennél a csoportnál az átlagos netezési idő becslésében. Ráadásul ez nem is egy teljesen pontos becslés, amit itt a konfidencia-intervallum hosszára látunk! Hiszen a kis elemszám miatt a k megbízhatósági szorzót itt 13–1 szabadságfokú t-eloszlásból számoltuk ki, de ehhez kellene az, hogy maguk a netezési idők normális eloszlást kövesssenek. Viszont, az a helyzet, ahogyan az alábbi hisztogram is szemlélteti, a napi netezési idők nem normális, hanem jobbra elnyúló eloszlásúak.

```
ESS.NetUsePerDay_Minutes.hist()  
plt.show()
```



7. fejezet

További FAE becslések és a Bootstrap módszer

7.1. A Konfidencia-intervallumok két általános tulajdonsága

Az átlagra vonatkozó konfidencia-intervallumokkal kapcsolatos számolások során megállapítottunk két olyan általános tulajdonságot a konfidencia-intervallumok hosszára, azaz a teljes becslési hibahatárra Δ -re vonatkozóan, amelyek igazak lesznek az összes többi - a tényban vizsgált - statisztikai mutató konfidencia-intervallumára is:

1. A megbízhatóság növelésével a konfidencia-intervallum egyre csak tágul, azaz a becslési hibahatár folyamatosan nő. Tehát, **nagyobb megbízhatóságú becslés csak pontatlanabb konfidencia-intervallum árán érhető el.**
2. Mivel a továbbiakban is konzisztensen viselkedő becslőfüggvényekkel ($\hat{\theta}$ -kal) fogunk dolgozni, így kijelenthető, hogy a **mintaelemszám (n) növelésével**, a SH értéke csökken. A csökkenő SH miatt pedig **az egész konfidencia-intervallum pontosabb lesz**. Magyarul az elemszám növelésével a konfidencia-intervallum hossza, leánykori nevén **becslési hibahatár (Δ) csökken**.

A következő két fejezetben figyeljük meg, hogy **minden újabb statisztikai mutató konfidencia-intervalluma a fenti két tulajdonságot betartva fog viselkedni!**

7.2. Arányok konfidencia-intervalluma

Vegyük elő újra a ESS2020.xlsx fájlban található adatbázist! Emlékeztetőül álljon itt, hogy ez az adatbázis a 2020-ban végzett európai szociális felmérés (European Social Survey 2020 = ESS2020) 1849 magyar kitöltjének válaszait tartalmazza 14 kérdésre (plusz van egy *id* oszlop).

Ugyebár a 6.4. fejezet 2. feladatában azt mondta, hogy ha az adatbázis valamelyik oszlopában üres értéket találunk, akkor az azt jelenti, hogy az adott sorban lévő kitöltő nem válaszolt a kérdésre. Az adatbázisban szereplő kitöltők a teljes 18 év feletti magyar népességből vett véletlen mintaként kezelhetők. Most feltesszük, hogy ez a véletlen minta visszatevéses, azaz *FAE* is. A következő tananyagban látni fogjuk, hogy ez nem is valóságtól elrugaszkodott feltevés.

Először is töltsök be az adatbázist ismét az Excelből egy **pandas** data frame-be és nézzük meg az **info** metódussal milyen oszlopaink (azaz ismérveink) vannak!

```
# Elemzéshez és ábrázoláshoz szükséges csomagok betöltése
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
from statsmodels.stats.meta_analysis import combine_effects
from statsmodels.graphics.dotplots import dot_plot

# Adatbeolvasás data frame-be
ESS = pd.read_excel("ESS2020.xlsx")
ESS.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 1849 entries, 0 to 1848
## Data columns (total 15 columns):
## #   #   Column           Non-Null Count  Dtype  
## ---  --  -- 
## #   0   id              1849 non-null   int64  
## #   1   PoliticalRadioTVPerDay_Minutes 1796 non-null   float64
## #   2   NetUsePerDay_Minutes          1099 non-null   float64
## #   3   TrustInParlament          1849 non-null   object 
## #   4   PoliticalPartyPref        1849 non-null   object 
## #   5   Education_Years          1830 non-null   float64
## #   6   WeeklyWork_Hours         685 non-null   float64
## #   7   Region                  1849 non-null   object 
## #   8   County                  1849 non-null   object 
## #   9   SecretGroupInfluenceWorldPol 1849 non-null   object 
## #   10  ScientistsDecievePublic    1849 non-null   object 
## #   11  COVID19                 1849 non-null   object
```

```
## 12 ContactCOVID19           1849 non-null   object
## 13 GetVaccince             1849 non-null   object
## 14 SomeContactCOVID19      1849 non-null   object
## dtypes: float64(4), int64(1), object(10)
## memory usage: 216.8+ KB
```

Láthatjuk, hogy megvan mind a 14+1 oszlopunk a megfelelő adattípusokkal. Hurrá! :)

Feladatunk ezúttal az lenne, hogy **99%-os megbízhatóságú konfidencia-intervallumot** építünk a **Fideszt támogatók arányára!**

Szerencsére ezt aránylag könnyű megtenni, hiszen **egy adott tulajdonsággal bíró egyedek aránya lényegében egy átlag!** Konkrétan egy olyan változó átlaga, ahol a tulajdonsággal bíró egyedek 1 értéket, míg a tulajdonsággal **NEM rendelkező** egyedek 0 értéket kapnak.

Ezt könnyű is szemléltetni Python-ban. Vegyük a feladat szempontjából releváns a PoliticalPartyPref ismérő **relatív gyakoriságait** a value_counts metódus segítségével:

```
# Teljes mintaelemszám megadása
n = ESS.PoliticalPartyPref.count()

# Relatív Gyakoriságok
ESS.PoliticalPartyPref.value_counts() / n
```

```
## PoliticalPartyPref
## Nem Ismert          0.643050
## Fidesz-KDNP         0.197404
## Egyesült Ellenzék   0.148729
## Egyéb                0.010817
## Name: count, dtype: float64
```

Ez alapján ugye a Fidesz támogatóinak aránya a megfigyelt 1849 elemű mintában 19.7%. Ezt az eredményt pedig úgy is megkaphatjuk, hogy csinálunk egy új Fidesz nevű oszlopot az ESS nevű data frame-be, amiben a Fidesz támogatók 1 értéket kapnak, a többiek 0-t, és vesszük az új oszlop átlagát. Az új oszlop létrehozásához a numpy csomag where néven futó függvényét használjuk. Ez lényegében olyan, mint az Excel HA függvénye: egy logikai feltétel megadása után értéket adunk az új oszlopan a *feltétel igaz* ágon, majd utána a *feltétel hamis* ágon.

```
# Létrehozzuk a Fidesz nevű oszlopot!
ESS['Fidesz'] = np.where(ESS.PoliticalPartyPref=='Fidesz-KDNP', 1, 0)
```

2127. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER

```
# És vesszük az új oszlop átlagát  
np.average(ESS.Fidesz)
```

```
## 0.19740400216333154
```

Ismét megkaptuk a 19.7%-os támogatottsági arányt. Ez alapján pedig könnyen elkészíthetjük rá a 99%-os megfeszítőszínűságú konfidencia-intervallumot a `scipy` csomag `stats.norm.interval` függvényével. Hiszen a nagy mintaelemszám miatt nem szükséges a k szorzót t-eloszlásból számolni, bőven megfelel nekünk a standard normális eloszlás alkalmazása is.

```
stats.norm.interval(  
    confidence=0.99,  
    loc=np.mean(ESS.Fidesz),  
    scale=stats.sem(ESS.Fidesz, nan_policy = 'omit'))
```

```
## (0.1735537723743666, 0.2212542319522965)
```

Tehát, a mintánk alapján a magyar népesség egészét tekintve az mondható el, hogy 99%-os valószínűséggel legalább 17.4%-uk támogatja a Fidesz-KDNP-t, viszont szintén 99%-os valószínűséggel kijelenthető, hogy a támogatottsági arányuk nem magasabb 22.1%-nál.

A teljes népességre nézve vett Fidesz támogatottság vizsgálható a `statsmodels` csomag `combine_effects` függvényével **regionális bontásban** is.

Mielőtt a `combine_effects` függvényt használnánk, csináljuk meg a kiinduló kimutatást az `ESS` data frame `groupby` metódusával:

1. A nominális ismérünk, azaz most a `Region` egyedi értékei szerint vesszük az átlaggal vizsgált mennyiségi ismérv (most `Fidesz`) részátlagait
2. Utána ezen részátlagok SH^2 -eit, azaz *becslési varianciáit* is megadjuk.

```
RegionKimutatas = ESS.groupby("Region").agg(  
    Elemszam = ('Fidesz', 'count'),  
    Atlag = ("Fidesz", np.mean),  
    SH = ("Fidesz", stats.sem)  
)
```

```
## <string>:1: FutureWarning: The provided callable <function mean at 0x000002140ED913
```

```
# SH négyzetre emelés
RegionKimutatas['BecslesiVar'] = RegionKimutatas['SH']**2
```

```
RegionKimutatas
```

	Elemszam	Atlag	SH	BecslesiVar
## Region				
## Budapest	298	0.248322	0.025069	0.000628
## Dél-Alföld	167	0.125749	0.025734	0.000662
## Dél-Dunántúl	143	0.153846	0.030278	0.000917
## Közép-Dunántúl	242	0.219008	0.026641	0.000710
## Nyugat-Dunántúl	221	0.167421	0.025171	0.000634
## Pest	265	0.222642	0.025604	0.000656
## Észak-Alföld	305	0.144262	0.020152	0.000406
## Észak-Magyarország	208	0.264423	0.030653	0.000940

Látható, hogy a mintán belül a Fidesz támogatottsága az Dél-Dunántúlon csak 15.4%, míg Nyugat-Dunántúlon 16.7%. Kérdés, hogy ezek a különbségek a mintavételi hiba, azaz a 99%-os megbízhatóságú konfidencia-intervallum figyelembe vételevel is megmaradnak-e!

Ami fontos még az előbb elkészített kimutatásból, hogy **minden régióban megvan a nagy elemszám ($n > 30$)**, így nyugodtan használható a k megbízhatósági szorzót standard normális eloszlásból számító **combine_effects** függvény. A függvény paramétereitől szabályait lásd a 6.6. fejezetben.

```
eredmeny = combine_effects(
  effect = RegionKimutatas.Atlag,
  variance = RegionKimutatas.BecslesiVar,
  row_names = RegionKimutatas.index,
  alpha = (1-0.99)
)

# a summary_frame eredményéből az első 8 sor kell nekünk,
# mert 8 régió szerint vizsgáljuk a támogatottsági arányokat!
eredmeny_tabla = eredmeny.summary_frame().iloc[0:8,0:4]
eredmeny_tabla
```

	eff	sd_eff	ci_low	ci_upp
## Budapest	0.248322	0.025069	0.199187	0.297457
## Dél-Alföld	0.125749	0.025734	0.075310	0.176187
## Dél-Dunántúl	0.153846	0.030278	0.094503	0.213189
## Közép-Dunántúl	0.219008	0.026641	0.166794	0.271223
## Nyugat-Dunántúl	0.167421	0.025171	0.118086	0.216756

2147. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER

```
## Pest           0.222642  0.025604  0.172458  0.272825
## Észak-Alföld  0.144262  0.020152  0.104766  0.183759
## Észak-Magyarország 0.264423  0.030653  0.204344  0.324503
```

Az eredményül kapott táblából ismételten látható, hogy a **mintán belül a Fidesz támogatottsága az Dél-Alföldön csak 12.6%**, míg Dél-Dunántúlon már 15.4%. Azonban, ha a konfidencia-intervallum segítségével a teljes népességet vizsgáljuk, akkor ez 99%-os valószínűséggel egy NEM szignifikáns (jelentős) eltérés, mivel a két konfidencia-intervallum metszi egymást! Tehát a teljes népességen belül a legjobb esetben egy 17.6%-os támogatottság is a Dél-Alföldön, míg a legrosszabb esetben belefér a Dél-Dunántúlon 9.5%-os támogatottság is. Tehát, az, hogy a Dél-Dunántúlon magasabb a Fidesz támogatottsági arány a mintában, az lehet csak a mintavételi hiba műve 99%-os megbízhatósággal! Ellenben az **Észak-Magyarországi Fidesz támogatottság 99% valószínűséggel a sokaságban is magasabb, mint a Dél-Alföldi**, hiszen a Dél-Alföldön a támogatottság legjobb esetben is csak 17.6%, míg Észak-Magyarországon legrosszabb esetben is már 20.4%. Tehát a két konfidencia-intervallum NEM metszi egymást, a mintában mért eltérések 99% valószínűséggel megmaradnak a sokaságban is!

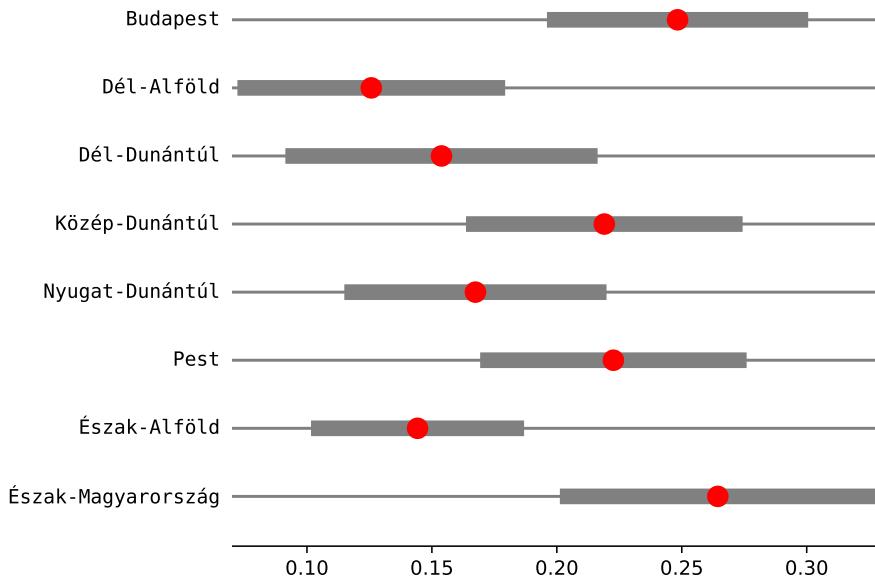
Az eredményekről ugyan úgy készíthetünk forest diagramot, mint a pártpreferenciák szerint bontott átlagos netezési időkről a 6.6. fejezetben. Itt ugyebár a statsmodels csomag dot_plot függvényét vetjük be. Ennek a használatáról is bővebb információk a 6.6. fejezetben találhatók.

```
# konfidencia-intervallumok méretének megadása az ábrához
intervallumok = np.abs(eredmeny_tabla[["ci_low", "ci_upp"]] - eredmeny_tabla[["eff"]]).round(4)
intervallumok

##                         ci_low    ci_upp
## Budapest            0.049135  0.049135
## Dél-Alföld          0.050439  0.050439
## Dél-Dunántúl        0.059343  0.059343
## Közép-Dunántúl      0.052215  0.052215
## Nyugat-Dunántúl     0.049335  0.049335
## Pest                 0.050183  0.050183
## Észak-Alföld          0.039496  0.039496
## Észak-Magyarország   0.060080  0.060080

# ábra elkészítése
dot_plot(
  points=eredmeny_tabla[["eff"]],
  intervals = intervallumok,
  lines=eredmeny_tabla.index)
```

```
# ábra megjelenítése
plt.tight_layout() # ne csússzanak ki a tengelyfeliratok
plt.show()
```



Az előbb taglalt, 99%-os megbízhatósággal a sokaságban is szignifikáns eltérés Dél-Alföld és Észak-Magyarország között. Az is látszik, hogy **hasonló szignifikáns különbség még ezen kívül Dél-Alföld és Budapest Fidesz támogatottsági aránya között található.**

7.2.1. Mintaelemszám meghatározása aránybecsléshez

Érdemes az arány konfidencia-intervallumának számítása során felhasználni azt az információt, hogy egy csak 0-ból és 1-ből álló változó korrigált mintaszórása $s = \sqrt{p(1-p)}$ módon számítható, ahol p az 1 értékek aránya a mintában! Nézzük is meg, hogy igaz-e ez! Ugyebár a Fidesz támogatottsági aránya a teljes 1849 elemű mintában $p = 19.7\%$. Ez alapján a szórása a Fidesz nevű 0-1-ből álló változónak $s = \sqrt{p(1-p)} = \sqrt{0.197 \times (1 - 0.197)} = 0.3977323$.

Nézzük meg az eredményt a `numpy` csomag `std` függvényével is:

```
np.std(ESS.Fidesz)
```

```
## 0.3980397745115843
```

2167. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER

A kétféleképp számolt érték némi kerekítési hibát leszámítva egyezik! :) De hát ez nem meglepő, hogy így alakult, hiszen az 5.5. fejezetben éppen azt mondta, hogy a mintaarányok (a p -k) standard mintavételi hibája a $SH(p) \approx \sqrt{\frac{p(1-p)}{n}}$ képlettel megadható. :)

Ez azt jelenti, hogy **az arány konfidencia-intervallumának hossza** a $\Delta = SH \times k$ képlet alapján $\sqrt{\frac{p(1-p)}{n}} \times k$ módon számítható, hiszen az átlag standard hibája $\frac{s}{\sqrt{n}}$ volt, és most felhasználtuk, hogy csak 0-1-et tartalmazó változóra $s = \sqrt{p(1-p)}$. A k megbízhatósági szorzó pedig ugyan úgy $N(0, 1)$ eloszlással számolható nagy méretű minták esetén, mint az átlag konfidencia-intervallumánál. Hiszen magas n esetén a $t(n - 1)$ eloszlás sűrűségfüggvénye már lényegében egybeesik az $N(0, 1)$ eloszlás sűrűségfüggvényével, ahogy a 6.4. fejezet 2. feladatában is láttuk.

Ennyi információ alapján pedig képesek vagyunk arra, hogy még mintavétel ELŐTT meghatározzuk, hogy az arány egy adott pontosságú és megbízhatóságú becsléséhez mekkora elemszámú mintára van szükségünk.

Hiszen 99%-os megbízhatósági szint mellett a szükséges megbízhatósági k szorzó a standard normális, azaz $N(0, 1)$ eloszlás inverz értéke alapján megadható $z_{1-\frac{\alpha}{2}}$ módon:

```
alfa=1-0.99
stats.norm.ppf(1-alfa/2)
```

```
## 2.5758293035489004
```

Vegyük az értéket kerekítve $k = 2.6$ -nak!

Ugyebár azt tudjuk, hogy a jelenlegi 1849 elemű mintánk esetén Fidesz támogatottsági aránya $p = 19.7\%$, **amitől a támogatottsági arány valós sokasági értéke 99%-os valószínűsséggel ± 2.4 százalékpontos hibahatárral** térhet el:

$$\pm \Delta = SH \times k = \sqrt{\frac{p(1-p)}{n}} \times k = \sqrt{\frac{0.197 \times (1 - 0.197)}{1849}} \times 2.6 = 0.0240$$

De mi a helyzet, ha a hibahatár értékét **1 százalékpontra akarom csökkenteni** és meg akarom őrizni a **99%-os megbízhatósági szintet?** Ekkor **nagyobb mintát kell venni, kérdés, hogy mennyivel nagyobbat.** Ezek alapján a kívánt Δ érték 0.01 és a $k = 2.6$ értékből sem akarok engedi. Azaz:

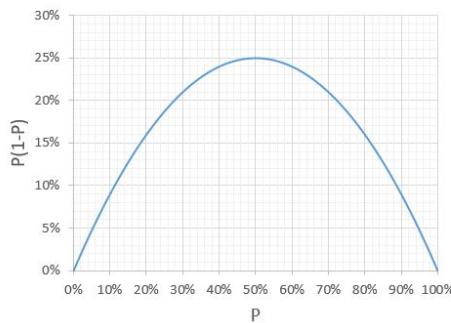
$$0.01 = \sqrt{\frac{p(1-p)}{n}} \times 2.6$$

Ebből n -t kifejezve:

$$n = \frac{2.6^2 \times p(1-p)}{0.01^2}$$

Ezen a ponton nagy a csábítás, hogy a képletből úgy számoljuk ki $n - t$, hogy $p = 19.7\%$ -kal dolgozzunk. De ezt **ne tegyük!** Mivel nem tudjuk, hogy a megnövelt elemszámú mintában mennyi is lesz p . Plusz, ha a **szükséges mintaelemszámot AZELŐTT akarjuk meghatározni**, hogy a kérdéses p arány becslésére már vettük mintát, akkor aztán tényleg lövésünk nincs a p értékéről!

Szerencsére, **rájöhetünk, hogy a $p(1-p)$ kifejezésnek könnyen meg tudjuk adni a maximumát**, hiszen az $f(p) = p(1-p) = p - p^2$ függvény egy fordította parabola, melynek maximuma $p = 0.5$ -nél kerül felvételre és értéke $\max(p(1-p)) = 0.25$:



Szóval az $N = \frac{2.6^2 \times p(1-p)}{0.01^2}$ formulába **mindig beírhatjuk a 0.25-öt, hiszen ez a legrosszabb szituációnk, ekkor lesz aránybecslés esetén maximális a standard hibánk**. Ha elégsgé nagy mintát veszünk, hogy a maximális SH mellett is $\Delta = 0.01$ -et érjünk el, akkor minden egyéb esetben is jók vagyunk.

Tehát, **az 1 százalékpontos hibahatár eléréséhez szükséges elemszám 99%-os megbízhatóság mellett $N = \frac{2.6^2 \times 0.25}{0.01^2} = 16900$ fő**. Ennek fényében különösen érdekes megléni ezen a linken hogy hány fős mintából dolgoztak a 2016-os Brexit népszavazás eredményének előrejelzése során a közvéleménykutatók, ahol lehetett tudni, hogy nagyon kiélezett a verseny a maradás és elszakadás pártok között, így a két párt támogatottsági arányának becslése során **nagyon szükség lett volna erre az 1 százalékpontos hibahatárra és a 99%-os megbízhatósági szintre, ami a 16900 elemű minták biztosítanak arányok becslése során**.

További érdekes példaként vegyük a Momentum Mozgalom 2021. október 11-én megosztott plakátját:

2187. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER



Ha megnézzük a kép jobb alsó sarkát láthatjuk, hogy az adatok forrása a Medián közvélemény kutató intézet. A dátum alapján rájöhetünk, hogy erről a Medián közvéleménykutatásról szól a plakát. A linkelt HVG cikkből megtudhatjuk, hogy a Medián közvéleménykutatása egy $n = 1000$ elemű véletlen mintát takar. Ebben az esetben számoljuk ki a különböző pártlisták támogatottsági arányaihoz a maximális Δ hibahatárt, azaz vegyük $p(1 - p)$ -t 0.25-nek:

$$\pm \Delta = SH \times k = \sqrt{\frac{p(1-p)}{n}} \times k = \sqrt{\frac{0.25}{1000}} \times 2.6 = 0.0411$$

Tehát a hibahatár durván **4 százalékpont**. Ha ezt az értéket Dobrev Klára esetén levonjuk a Fidesz támogatottsági arányából és hozzáadjuk az Ellenzék támogatottsági arányához, majd ha MZP esetén az Ellenzéki arányból levonjuk és a Fideszhez hozzáadjuk, akkor láthatjuk, hogy a **plakáton** kimutatott Fidesz és Ellenzéki pártlisták támogatottsági arányai közötti különbség MZP és Dobrev esetén is bőven a mintavételi hibahatáron belül van! Tehát, egyik esetben sem mondható el a konfidencia-intervallum alapján, hogy a teljes néppességen reális lenne a plakát állítása, miszerint csak MZP-vel verhető a Fidesz. Az **plakát üzenete csak a megfigyelt 1000 elemű mintán belül vehető igaznak!** Amit meg is erősít a 2022-es országgyűlési választások eredménye...

7.2.2. Szükséges minimális elemszám aránybecsléshez

Az arány intervallumbecslés esetén van a 2.1. fejezetben taglaltak mellett egy **minimális elemszám követelménye is, aminél kisebb mintákban az intervallumbecslés egyáltalán NEM elvégezhető!!**

Ez a követelemény abból jön, hogy az aránybecslést gyakorlatilag egy átlagbecslésre vezetjük vissza. Hiszen átlagbecslés esetén **kis elemszámú mintáknál feltételezzük az alapsokaság** (tehát, amiből a mintát vettük)

normális eloszlását, még akkor is, ha a megbízhatósági szorzót t-eloszlásból számítjuk! Egy csak 0-ból és 1-ből álló adatsor pedig bajosan fog normális eloszlást követni! :)

Azt, hogy mi számít aránybecslés esetén nagy mintának, a következő szabály adja meg:

- Legyen az arány szempontjából *kedvező* esetek száma több, mint 10 a mintában, azaz: $n \times p > 10$
- Legyen az arány szempontjából *kedvezőtlen* esetek száma is több, mint 10 a mintában, azaz: $n \times (1 - p) > 10$

Ez a Fidesz támogatók arányának korúabbi példájára nézve úgy néz ki, hogy a teljes mintánk elemszáma $n = 1849$ fő:

```
len(ESS) # tábla sorainak száma
```

```
## 1849
```

Míg az arány szempontjából *kedvező* esetek, azaz a Fidesz-KDNP támogatók száma 365 fő:

```
ESS.loc[ESS['PoliticalPartyPref'] == 'Fidesz-KDNP', 'PoliticalPartyPref'].count()
```

```
## 365
```

Tehát a két feltétel itt a következőképpen teljesül:

- A *kedvező* esetek száma $365 > 10 \rightarrow$ feltétel teljesül
- A *kedvezőtlen* esetek száma $(1849 - 365) = 1484 > 10 \rightarrow$ feltétel teljesül

Tehát, minden feltétel teljesül, a Fidesz támogatók arányának intervallumbecslése elvégezhető volt, mivel megvan a minimális mintaelemszám. Yeah! :)

7.2.3. Aránybecslés a `statsmodels` csomaggal

Érdemes megemlékezni arról is, hogy Pythonban nem csak a `scipy` csomag segítségével lehet arányra vonatkozó intervallumbecslést készíteni hanem a `statsmodels` csomagban is van erre beépített megoldás.

A `scipy` csomagban a `stats.norm.interval` függvény alkalmazásához ugyebár készíteni kellett egy külön új oszlopota data frame-ben az aránybecsléshez, ahol

2207. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER

1-gyel jelöltük az arányban vizsgált tulajdonsággal rendelkező mintaelemeket, 0-val a tulajdonsággal nem rendelkezőket. És a trükk az volt, hogy az arányra úgy tekintünk, mint az új 0-kat és 1-eket tartalmazó oszlop átlagára.

Ha ezt az **új oszlop létrehozást meg akarjuk úszni**, akkor jöhet jól a `statsmodels` csomag `stats.proportion_confint` függvénye. Ebben a függvényben csak a következő paramétereket kell megadni:

- `count`: Kedvező eseteket száma a mintában. Tehát azon értékek száma, amiket a `scipy`-os megoldásban 1-gyel kódolnánk az új oszlopan.
- `nobs`: Minta teljes elemszáma, tehát az n .
- `alpha`: A megengedett becslési hibavalósínség, tehát az α . Szóval, itt tényleg az $1 - \text{megbízhatósági szint}$ kell.
- `method`: Itt kell megadni, hogy a k megbízhatósági szorzót milyen eloszlásból számítjuk. Mi nagy mintaméretet feltételezve minden maradunk a standard normális eloszlásnál, ami itt a '`normal`' beállításnak fog megfelelni.

A fenti paraméterek megértése után nézzük meg hogyan tudjuk az ESS adatbázison keresztül megbecsülni a magyar népességen a Fidesz támogatók arányát a teljes népességen ezzel a `stats.proportion_confint` függvényel! :)

Először is kell egy kedvező esetek száma, ez a Fidesz-KDNP-t támogatók gyakorisága a data frame-ben a `PoliticalPartyPref` oszlop alapján. Utána pedig lövünk egy n -t, ami természetesen a data frame sorinak a száma:

```
kedvezo_esetek = ESS.loc[ESS['PoliticalPartyPref'] == 'Fidesz-KDNP', 'PoliticalPartyPref']
osszes_eset = len(ESS) # tábla sorainak száma

kedvezo_esetek

## 365

osszes_eset

## 1849
```

Szuper, megvagyunk! Van összesen $n = 1849$ megfigyelés és ebből 365 Fidesz támogatónk. Ez utóbbi akkor a kedvező esetek száma. :)

Paraméterezzük akkor ezzel fel a `stats.proportion_confint` függvényt, a korábbiakhoz haosnlóan 99%-os megbízhatósági szintet használva:

```
# Betöltjük a csomagot
import statsmodels.api as sm

# Használjuk a függvényt
sm.stats.proportion_confint(count=kedvezo_esetek, nobs=osszes_eset, alpha=(1-0.99), method='norma
## (0.17356022274089133, 0.22124778158577174)
```

Nagyon szépen láthatjuk a korább eredményünk: a Fidesz támogatók aránya a teljes magyar népességben 17.4% és 22.1% között található 99% valószínűsséggel.

Sőt, ha figyelembe vesszük, hogy a teljes magyar népesség 2020. január 1-jén 9 772 756 fő volt, akkor megkaphatjuk, hogy **konkrétan hány főnyi Fidesz támogató lehet a magyar népességben 99%-os megbízhatósággal**. Egyszerűen csak **az arány konfidencia-intervallum két határát kell felszorozni az $N = 9772756$ -os sokasági elemszámmal**.

```
[int(0.174 * 9772756), int(0.221 * 9772756)]
## [1700459, 2159779]
```

Tehát Magyarországon 99% valószínűsséggel 170 ezer és 216 ezer fő közötti a Fidesz támogatók száma. Éljen! :)

7.3. A Bootstrap becslések általános elve

Eddig a konfidencia-intervallumokkal kapcsolatban elég könnyű dolgunk volt úgymond, mert az **átlag és arány esetében** is a konfidencia inztervallum hosszát (Δ -t) ki tudtuk számolni standard hiba (SH) szorozva megbízhatósági szorzó (k) elven:

$$\Delta = SH \times k$$

Azért tudott ez a formula működni, mert a **standard hibára tudtunk adni egy egyszerű képletet** ($\frac{s}{\sqrt{n}}$ vagy $\sqrt{\frac{p(1-p)}{n}}$) és a k -t ki tudtuk számolni **valami konkrét eloszlásból** (standard normális vagy t-eloszlás).

NODE, mi a helyzet ha ezek az eszközök NEM állnak rendelkezésre?
Tehát, mi van akkor, ha

1. A standard hibáját egy statisztikai mutatónak (paraméternek, azaz θ -nak) nem lehet egysegrű képpel kiszámolni.

2227. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER

2. A k számolásához nincs konkrét eloszlás, vagy ami van, az csak lehetetlen feltételekkel alkalmazható (pl. a vizsgált alapsokaság, amiből a mintát vettük az legyen normális eloszlású, mintamérettől függetlenül)

Ezekben az esetekben **segít rajtunk a Bootstrap becslés!** Nézzük meg ennek a módszernek mi az általános alapelve z átlag standard hibáján keresztül.

Ugyebár az átlag standard hibája úgy jön ki egy mintavételekből, hogy fogjuk a minta korrigált szórását, s -t, és ezt elosztjuk a mintaelemszám (n) gyökével:

$$SH(\bar{y}) = \frac{s}{\sqrt{n}}$$

Számoljuk ki az ESS adatbázisból a napi netezési idő (NetUsePerDay_Minutes oszlop) átlagának standard hibáját! Csak emlékezzünk, hogy a NetUsePerDay_Minutes oszlopan volt pár hiányzó érték, ezeket ne vegyük figyelembe az n meghatározásakor!

```
n = ESS.NetUsePerDay_Minutes.count() # így nem számoljuk be az üres értékeket  
korrekciós = np.std(ESS.NetUsePerDay_Minutes, ddof = 1)  
  
SH_Keplet = korrekciós / np.sqrt(n)  
SH_Keplet  
  
## 4.381340690645097
```

Szuper, a mintaátlag várható eltérése a valós, sokasági átlagos netezési időtől várhatóan ± 4.38 perc!

Hogyan jön ki ez az eredmény Bootstrap módon?

A **Bootstrap becslés alapötlete a standard hiba alap definíciójából** jön, amit az 5.5. fejezetben néztünk. Vegyük ki a sokaságból nagyon-nagyon sok (pl. 1000 vagy 10000 db) visszatevéses (azaz FAE) mintát, minden mintából számoljuk mi a mintaátlagot és a mintaátlagok szórása a standard hiba az átlagbecslés *torzítatlansága* miatt. Ugyebár ezzel a megközelítéssel az a baj, hogy a gyakorlatban csak egyetlen egy darab mintánk van, és nem ismerjük a sokaságot, így nem tudunk belőle nagyon-nagyon sok FAE mintát kivenni. Nos, a **Bootstrap módszer** azt mondja, hogy ezt az alap szórás-elvű SH számolást tudjuk **SZIMULÁLNI** akár egyetlen egy darab mintavételeből is!

Ha van egy n elemű mintánk, akkor abból vegyük ki nagyon-nagyon sok (pl. 1000 db) szintén n elemű FAE almintát!! A FAE, azaz visszatevéses elv miatt, az n elemű alminták összetétele

véletlenszerűen meg fog változni, és ezek a véletlen változások épp a mintavételi hiba tendenciáit követik le!! Ezek után más dolgunk nincs, mint kiszámolni minden almintából a mintaátlagot, és venni ezek sima, korrigálatlan szórását, és ez lesz a *SH*!!

Lássuk akkor ezt a dolgot a gyakorlatban! Számoljuk ki az átlag standard hibáját Bootstrap elven!

Először is vegyük a `NetUsePerDay_Minutes` oszlopból mondjuk 1000 db **FAE** mintát, és tároljuk le ezeket az almintákat egy olyan data frame-ben, aminek 1000 sorában lesznek a különböző mintavételek, míg n db oszlopában az n db mintaelem minden egyes mintavételezési körben. Itt gyakorlatilag ugyanazokat a megoldásokat követjük Pythonban, mint a 4.3.1. fejezetben.

Készítsük el először az almintákat tároló üres data frame-t! Mivel 1099 db nem hiányzó értéke van a `NetUsePerDay_Minutes` oszlopnak, így $n = 1099$ oszlopa lesz az új data frame-nek!

```
n = ESS.NetUsePerDay_Minutes.count() # így nem számoljuk be az üres értékeket

oszlopnevek = (np.array(range(n))+1)
oszlopnevek = oszlopnevek.astype(str)
oszlopnevek = np.core.defchararray.add("Elem", oszlopnevek)
oszlopnevek

## array(['Elem1', 'Elem2', 'Elem3', ..., 'Elem1097', 'Elem1098', 'Elem1099'],
##       dtype='<U15')

MintaDF = pd.DataFrame(columns = oszlopnevek)
MintaDF

## Empty DataFrame
## Columns: [Elem1, Elem2, Elem3, Elem4, Elem5, Elem6, Elem7, Elem8, Elem9, Elem10, Elem11, Elem12]
## Index: []
## [0 rows x 1099 columns]
```

Majd elkészítünk egy olyan verziót az `ESS` data frame-ból, amiben **nincsenek benne a `NetUsePerDay_Minutes` oszlop hiányzó értékei**. Így egy $n = 1099$ soros data frame-ünk lesz.

```
ESS_Szurt = ESS[ESS['NetUsePerDay_Minutes'].notna()]
ESS_Szurt.info()
```

2247. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER

```

## <class 'pandas.core.frame.DataFrame'>
## Index: 1099 entries, 0 to 1847
## Data columns (total 16 columns):
## #   Column           Non-Null Count  Dtype  
## ---  --  
## 0   id               1099 non-null   int64  
## 1   PoliticalRadioTVPerDay_Minutes 1082 non-null   float64 
## 2   NetUsePerDay_Minutes          1099 non-null   float64 
## 3   TrustInParlament           1099 non-null   object  
## 4   PoliticalPartyPref         1099 non-null   object  
## 5   Education_Years            1090 non-null   float64 
## 6   WeeklyWork_Hours           541 non-null   float64 
## 7   Region                      1099 non-null   object  
## 8   County                      1099 non-null   object  
## 9   SecretGroupInfluenceWorldPol 1099 non-null   object  
## 10  ScientistsDecievePublic    1099 non-null   object  
## 11  COVID19                     1099 non-null   object  
## 12  ContactCOVID19             1099 non-null   object  
## 13  GetVaccince                1099 non-null   object  
## 14  SomeContactCOVID19         1099 non-null   object  
## 15  Fidesz                      1099 non-null   int32  
## dtypes: float64(4), int32(1), int64(1), object(10)
## memory usage: 141.7+ KB

```

Szuper, ezzel megvagyunk!

Akkor most **jöhet az 1000 db $n = 1099$ elemű alminta generálása for ciklusból** és a tárolás az újonnan létrehozott data frame-ben.

Most is megnövekedett futásiidőre kell itt készülni! :)

```

for index in range(1000):
    AktualisMinta = ESS_Szurt['NetUsePerDay_Minutes'].sample(n = len(ESS_Szurt), replace = True)
    AktualisMinta.index = oszlopnevek
    MintaDF = MintaDF.append(AktualisMinta, ignore_index = True)

MintaDF

##      Unnamed: 0   Elem1   Elem2   Elem3   ...   Elem1096   Elem1097   Elem1098   Elem1099
## 0           0    150     80     90   ...     120     360     105     330
## 1           1     60    150    180   ...     240      60      90      60
## 2           2     90     60    120   ...     120     140     150     150
## 3           3     60     90    120   ...     360     120     180      60
## 4           4     80    240     60   ...     120     180      95     330
## ...
## 995        995    240     90    300   ...     510     150     240     120

```

```

## 996      996    180     74    120   ...      45      60    240     90
## 997      997    420     60    120   ...     240      90     45    240
## 998      998     90    120     60   ...     120    330    180    300
## 999      999    960    180    180   ...    150    120    180    180
##
## [1000 rows x 1100 columns]

```

Akkor meg is van az 1000 db almintánk! :) Számoljuk ki **mindegyik almintában a mintátlagot!** Figyeljünk itt is arra, hogy mivel a data frame oszlopai folyamatosan bővülnek amjd, így manuálisan le kell szorítani a numpy statisztikai függvények alkalmazását mindig az első $n = 1099$ db oszlopra az `iloc` metódussal!

```
MintaDF['Atlagok'] = np.mean(MintaDF.iloc[:,0:n], axis=1)
```

```
MintaDF
```

```

##      Unnamed: 0   Elem1   Elem2   Elem3   ...   Elem1097   Elem1098   Elem1099   Atlagok
## 0          0    150     80     90   ...      360      105      330  181.517743
## 1          1     60    150    180   ...       60      90      60  174.091902
## 2          2     90     60    120   ...      140      150      150  176.921747
## 3          3     60     90    120   ...      120      180      60  184.478617
## 4          4     80    240     60   ...      180      95      330  184.444950
## ...
## 995      995    240     90    300   ...      150      240      120  182.411283
## 996      996    180     74    120   ...       60      240      90  182.355778
## 997      997    420     60    120   ...       90      45      240  178.429481
## 998      998     90    120     60   ...     330      180      300  182.266606
## 999      999    960    180    180   ...     120      180      180  173.399454
##
## [1000 rows x 1101 columns]

```

A standard hibánk pedig akkor ezeknek az alminta-átlagoknak lesz a sima korrigálatlan szórása!

```
SH_Bootstrap = np.std(MintaDF.Atlagok)
```

```
SH_Bootstrap
```

```
## 4.511376672501348
```

E voilá! Némi kerekítési hiba mellett ez kb. ugyan annyi, mint a $\frac{s}{\sqrt{n}}$ képlettel kapott verzió! :) Illetve, értelemszerűen **itt mindenki más értéket kaphatott**, mert nem fixáltuk a véletlenszám generátor magját a mintavételek során.

2267. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER

```
[SH_Keplet, SH_Bootstrap]
```

```
## [4.381340690645097, 4.511376672501348]
```

Nyilván, ha több almintát veszünk, akkor pontosabb lesz a közelítés. Ezt úgy mondjuk szépen szakszóval, hogy növeljük a Bootstrap becslés replikációinak számát. :) Tehát a replikációk száma az alminták számát jelenti.

NODE, mire volt jó, hogy ezt a fránya *SH*-t ilyen körülményesen számoltuk ki a képlet helyett? Nos, átlag esetében a világon SEMMIRE! Viszont, ezzel a Bootstrap elvvel ki tudjuk számolni pl. a napi netezési időnk mediánjának standard hibáját, amire amúgy NINCS olyan egyszerű képlet, mint az átlag standard hibájára! Ugyebár ezt megtehetjük, mert megnéztük az 5.4. fejezetben, hogy a medián is egy *torzítatlanul* becsülhető statisztikai paraméter, mint az átlag.

```
MintaMedian = np.median(ESS_Szurt.NetUsePerDay_Minutes)  
MintaMedian
```

```
## 120.0
```

```
MintaDF['Medianok'] = np.median(MintaDF.iloc[:,0:n], axis=1)  
SH_Median = np.std(MintaDF.Medianok)
```

```
SH_Median
```

```
## 11.718634732766454
```

Tehát tudjuk, hogy a megfigyelt $n = 1099$ elemű mintában a medián napi netezési idő 120 perc, és a **standard hibából tudjuk, hogy ez az érték a teljes magyar népesség** (sokaság) valós medián napi netezési idejétől várhatóan ± 11.72 perccel különbözik.

Innen pedig már csak egy lépés, hogy legyen valami $1 - \alpha$ megbízhatósági szintű **konfidencia-intervallumunk a mediánra** ezzel a Bootstrap módszerrel!

7.4. A Medián Bootstrap intervallumbecslése

Ahhoz, hogy megtudjuk a Bootstrap módszerrel előállított 1000 db alminta alapján a medián $1 - \alpha$ megbízhatóságú konfidencia-intervallumát, akkor egyszerűen vennünk kell az **almintákból kiszámolt 1000 db medián**

adatsorának $\alpha/2$ és $1-\alpha/2$ percentiliseit! Hiszen, az $1-\alpha$ megbízhatóságú konfidencia intervallumnak azt kell megadnia, hogy milyen két érték között mozoghat a valós, sokasági medián $1-\alpha$ valószínűséggel. Ezt pedig pl. **átlag esetében úgy állapítottuk meg, hogy vettük a sok-sok mintaátlag eloszlásának, konkrétan az $N(\mu, \frac{\sigma}{\sqrt{n}})$ eloszlásnak a középső $1-\alpha$ százalékát!!** A Bootstrap mintavételezés segítségével pedig pont a **sok-sok minta-medián eloszlását akartuk szimulálni, tehát ennek kell venni a középső $1-\alpha$ százalékát!** Ezt pedig a minta-medián értékek $\alpha/2$ és $1-\alpha/2$ percentilisei adják ki.

Nézzük is meg az eredményt a netezési idők mediánjának 95%-os megbízhatóságú konfidencia-intervallumára! Pythonban a data frame `Medianok`-nak elnevezett oszlopának `quantile` metódusát tudjuk használni a keresett két percentilis kiszámításához. Ugyebár ekkor $1-\alpha = 95\% = 0.95$, tehát $\alpha = 0.05 = 5\%$:

```
alfa = 0.05

[MintaDF['Medianok'].quantile(alfa/2), MintaDF['Medianok'].quantile(1-alfa/2)]

## [120.0, 150.0]
```

Tehát, a **teljes magyar népesség körében a medián napi netezési idő 120 és 150 perc között van 95% valószínűséggel..**

Az eredményből láthatjuk, hogy a medián konfidencia-intervalluma **NEM szimmetrikus** a minta egészéből számított mediánra (ami szintén 120 perc volt, mint itt az alsó határ), mint ahogy az átlag konfidencia-intervalluma a minta egészéből számított átlagra szimmetrikus volt!

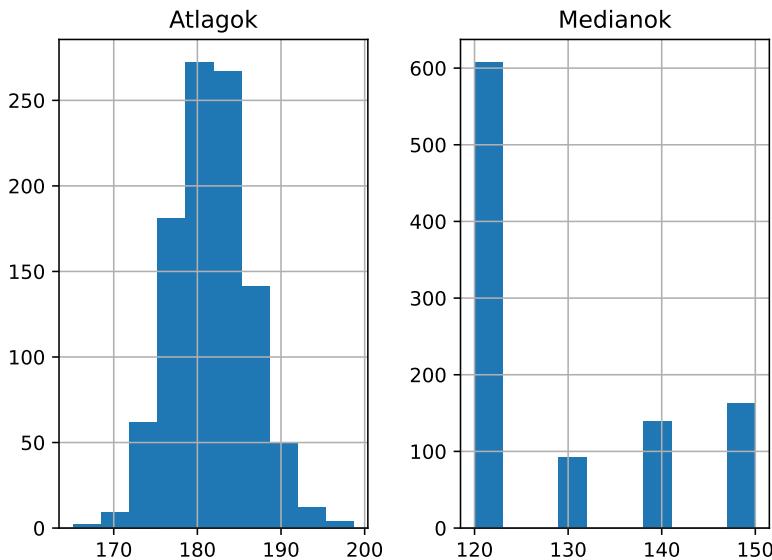
Ennek hátterében az áll, hogy a **mintá-mediánok eloszlása nem szép szimmetrikus normális eloszlás, mint a mintaátlagoké**. Ez a szimulált alminták átlagainak és mediánjainak hisztogramjaiból rögtön szépen látszódik.

```
MintaDF[['Atlagok', 'Medianok']].hist()

## array([[<Axes: title={'center': 'Atlagok'}>,
##          <Axes: title={'center': 'Medianok'}>]], dtype=object)

plt.show()
```

2287. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER



És tényleg: a mediánok esetében a szimulált mintavételi eloszlás nagyon koncentrál a 120-ra, míg az átlag esetében megkapjuk a jól ismert, csudiszép normális eloszlásunkat! :) Medián esetén a torzabb eloszlás kép részben összefügg azzal, amit Stat. I-ből tanultunk: kis értékkészletű ismérvre a medián nem túl informatív mutatószám! Mindenesetre ezt a „torzságot” a Bootstrap elven számolt konfidencia-intervallumkezeli. :)

Egyébként le is tudjuk csekkolni, hogy **átlag esetében a Bootstrap elven számított 95%-os megbízhatóságú konfidencia-intervallum ugyan azt az eredményt hozza némi kerekítési hibával, mint a standard normális eloszlású megbízhatóságú szorzóval dolgozó konfidencia-intervallum számolás.** Ugyebár most $n = 1099$ nagy minta, így lehet standard normális eloszlású megbízhatóságú szorzóval dolgozni a t-eloszlású helyett ismeretlen sokasági szórás esetén is.

```
# Standard normális eloszlású megbízhatóságú szorzós verzió
stats.norm.interval(
    confidence=0.95,
    loc=np.mean(ESS.NetUsePerDay_Minutes),
    scale=stats.sem(ESS.NetUsePerDay_Minutes, nan_policy = 'omit'))
```

```
## (172.70663359101636, 189.88117350634488)
```

```
# Bootstrap verzió
[MintaDF['Atlagok'].quantile(0.05/2), MintaDF['Atlagok'].quantile(1-0.05/2)]
## [1] 173.6656050955414, 190.59504094631484
```

Tényleg, csak minimális eltérés van a két eredmény között, ami amúgy csökkenthető, ha a Bootstrap módszerben több almintával, azaz magasabb replikációszámmal dolgozunk.

Szerencsére, a **Bootstrap konfidencia intervallumokat** nem kell minden ilyen szenvédős módon kiszámolni Pythonban, mint ahogy most tettük a külön data frame generálásra for ciklusból az almintáknak, hanem a **scipy csomagban létezik rá egy beépített függvény stats.bootstrap néven**. Ennek az is az előnye, hogy a **futásidéje is sokkal jobb, mint a mi összebarkálcsolt megoldásunknak**. Hiszen ezen a függvényen több fejlesztő is dolgozott több hónapot, halálra van az egész kód mögötte optimalizálva. A függvény paraméterei a következőképpen működnek:

1. Megadjuk az **adatsort, amiből** majd a Bootstrap **almintákat kell generálni**.
2. Megadjuk annak a **statisztikai paraméternek** (mutatószámnak) a **numpy csomagban** lévő függvényét, **amire a konfidencia-intervallumot** akarunk számolni.
3. A **random_state** paraméterben **megahdatunk** a mintavételezés véletlenszám-generátorának egy ***véletlen magot**, amivel minden gépen ugyan azt az eredményt kapjuk.
4. A **method** paraméterben kikötjük, hogy ugyan úgy **precentilisekkel határozza meg a konfidencia-intervallumot** a gépszellem, ahogyan mi is tettük ezt manuálisan.
5. A **replikációszámot** egy **n_resamples** paraméterben lehet variálni, de mivel alapesetben 10000 db almintát készít a függvény, így **nem érdemes ebbe manuálisan belenyúlni ebbe a paraméterbe**. A 10000 replikáció általában elég szokott lenni. :)

Lássuk akkor a függvényt akcióban, a napi netezési idők mediánjának 95%-os megbízhatóságú konfidencia-intervalluma esetében! Annyi technikai előfeltétel van, hogy az első paraméteren átadott adatsort a függvénynek **sequence** formátumban kell átadni, így a függvény alkalmazása előtt még ezt a konverziós lépést lehet látni a kódban:

```
# Konvertálás sequence típusra
NetUse_Seq = (ESS_Szurt.NetUsePerDay_Minutes,)

stats.bootstrap(NetUse_Seq, np.median, confidence_level=0.95,
                random_state=1992, method='percentile')
```

2307. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER

```
## BootstrapResult(confidence_interval=ConfidenceInterval(low=120.0, high=150.0), boot
```

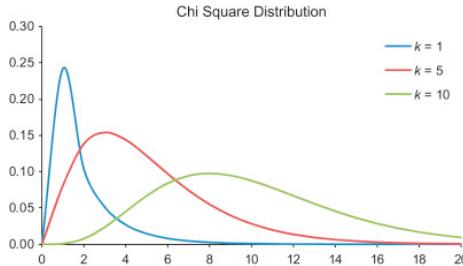
E voilá! Itt is van a $120 - 150$ perces konfidencia-intervallum és a 11.7 perc körüli SH is! :)

7.5. A Szórás Bootstrap intervallumbecslése

A szórás $1 - \alpha$ konfidencia-intervallumára van egy zárt formulás képletünk, ami elég egyszerűen néz ki:

$$P\left(\sqrt{\frac{(n-1)s^2}{\chi_{\alpha/2}^2(n-1)}} < \sigma < \sqrt{\frac{(n-1)s^2}{\chi_{1-\alpha/2}^2(n-1)}}\right) = 1 - \alpha$$

A képletben a $\chi^2(n-1)$ az egyetlen ismeretlen betű. Ez azért van itt, mert a mintaszórásnégyzetek eloszlása egy $n-1$ szabadságfokú χ^2 -eloszlás, úgy mint ahogy a mintaátlagok eloszlása egy $n-1$ szabadságfokú t -eloszlás, ha a sokasági szórást nem ismerjük előre. A $\chi^2(df)$ -eloszlás egy **jobbra elnyúló normális eloszlás**. Az **elnyúlás mértékét a szabadságfoka** (angolul degrees of freedom = df) **szabályozza**: minél nagyobb a szabadságfok, annál kevésbé jobbra elnyúló az eloszlás:



A $\chi_{\alpha/2}^2(n-1)$ és $\chi_{1-\alpha/2}^2(n-1)$ értékek egy $n-1$ (mintaelemszám mínusz egy) szabadságfokú χ^2 -eloszlás **inverz értékei $\alpha/2$ és $1 - \alpha/2$ aláesési valószínűségek mellett**. Ezeket Pythonban kiszámolni pofon egyszerű, hiszen van beépített függvény rá a `scipy` csomagban, teljesen logikusan a normális, exponenciális és t-eloszlások után `stats.chi2.ppf` néven. A függvény első paramétere az ismert alá esési valószínűség, második paramétere a szabadságfok, pont mint a `stats.t.ppf` esetében.

A konfidencia-intervallum ezek után pedig csak annyi, hogy kiindulunk a mintaelemszám mínusz egy szorozva korrigált mintabeli szórásnégyzet $((n-1)s^2)$ értékből, és azt leosztjuk az intervallum alsó határához $\chi_{1-\alpha/2}^2(n-1)$ -gyel, a felső határához pedig $\chi_{\alpha/2}^2(n-1)$ -gyel, és az egész hányadosból gyököt vonunk.

Nézzük meg az elvet **alkalmazás közben a napi netezési idők szórásának 97-os megbízhatóságú konfidencia-intervallumára!** Arra emlékezzünk csak technikai oldalon, hogy a $\chi^2_{\alpha/2}(n - 1)$ ad majd kisebb inverz értéket, így ővele a felső határnál kell osztani (hogy az eredmény nagyobb legyen), míg a $\chi^2_{1-\alpha/2}(n - 1)$ érték ad magasabb inverz értéket, így ővele az alsó határnál kell osztani (hogy az eredmény kisebb legyen).

```
# Mintabeli korrigált szórás számítása
s = np.std(ESS_Szurt.NetUsePerDay_Minutes, ddof=1)
s

## 145.24656528126647

# Konfidencia-intervallum számítása
n = len(ESS_Szurt)
alfa = 1-0.97

khi_negyzet_also = stats.chi2.ppf(alfa/2, df = (n-1))
khi_negyzet_felso = stats.chi2.ppf(1-alfa/2, df = (n-1))

kozos_szamlalo = (n-1)*(s**2)

[np.sqrt(kozos_szamlalo/khi_negyzet_felso), np.sqrt(kozos_szamlalo/khi_negyzet_also)]

## [138.81196985448224, 152.28945163668794]
```

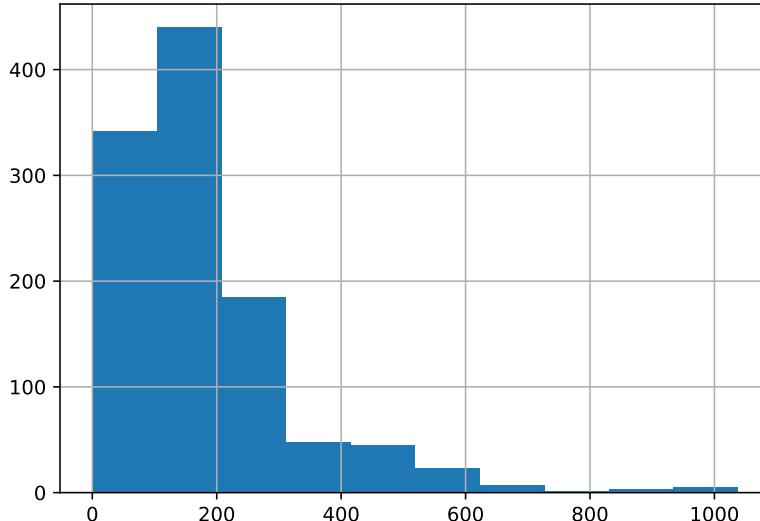
Tehát, a megfigyelt mintában a napinetezási idők szórása 145.2 perc, ami a teljes magyar népesség körében 97% valószínűsséggel 138.8 és 152.3 perc között mozoghat.

Érdemes megfigyelni, hogy a konfidencia-intervallum NEM szimmetrikus a mintából számított szórásra azaz s -re, mint ahogy az átlag konfidencia-intervalluma a minta egészéből számított átlagra szimmetrikus volt! Közelebb van valamivel a mintából számolt szórás (s) a konfidencia-intervallum alsó határához, mint a felsőhöz! Itt ennek az oka az, hogy a χ^2 -eloszlás egy jobbra elnyúló eloszlás, tehát azt gondolja, hogy a kisebb értékek jellemzőek inkább a mintaszórások eloszlására, így a mintából mért szórást ($s-t$) is inkább a konfidencia-intervallum „aljára teszi”.

Ez mind szép és jó, de ez a χ^2 -eloszlásos konfidencia-intervallum képlet jelen esetben feltételezi, hogy az adatsor, amiből a mintát vettük (tehát a napi netezési idők) az normális eloszlású a sokaságban!!! Ez pedig marhára NEM TELJESÜL, ahogy azt egy gyors hisztogram készítés után láthatjuk is!

2327. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER

```
ESS_Szurt.NetUsePerDay_Minutes.hist()  
plt.show()
```



Jobbra elnyúló ez az időeloszlás, mint a fene. Ebből aztán semmilyen sokaságban **NEM** lesz szép szimmetrikus normális eloszlás!

Tehát, a szórás konfidencia-intervallum képlete nem reális eredményeket mutat a valós, sokasági szórásra, mert a képlet mögötti feltételezés NEM teljesül! Mit lehet tenni? Hát alkalmazzuk a Bootstrap becslést, mivel annak semmilyen csúnya előfeltétele nincs. Csak az kell itt is, hogy az eredeti sokaságból FAE mintavételünk legyen, de megbeszéltük a 6.4. fejezet 2. feladatában, hogy az ESS2020 mintavétel esetén ez a FAE mintavétel feltételezhető.

Szóval, akkor csináljuk meg a Bootstrap becslést a napi netezési idők szórására is, 97%-os megbízhatósági szinten! A beépített `scipy` csomagos függvénykel most is simán tudunk dolgozni, mint a medián esetében is tettük:

```
stats.bootstrap(NetUse_Seq, np.std, confidence_level=0.97,  
                random_state=1992, method='percentile')
```

```
## BootstrapResult(confidence_interval=ConfidenceInterval(low=131.27407547315042, high=  
##                  137.40271278, 140.21897524]), standard_error=6.453404340995032)
```

Ez alapján pedig a napi netezési idők szórása a teljes magyar népesség körében 97% valószínűséggel 131.3 és 159.0 perc között mozoghat. Ami

érdemben más, mint a χ^2 -es formulából kapott 138.8 és 152.3 perc közötti intervallum, de a **Bootstrap verzió a reálisabb, hiszen az NEM feltételezi a netezési idők normális eloszlását a teljes sokaságban!**

Szóval ez a Bootstrap becslés elég menő dolog. Pl. lehet vele intervallumbecslést készíteni átlag esetében is kis mintákra ($n \leq 30$), amikor a mintavételezett adatsor eloszlása asokaságban NEM normális eloszlású és NEM tudjuk a valós sokasági szórást:

GF: "baby come over"
Me: "I can't, my sample size is small and
not normally distributed"
GF: "I'm home alone"
Me: "I'll be right over"
Me:



2347. FEJEZET. TOVÁBBI FAE BECSLÉSEK ÉS A BOOTSTRAP MÓDSZER

8. fejezet

Becslések EV és R mintából

8.1. Intervallumbecsles visszatevés nélküli egyszerű véletlen (EV) mintákból

Az eddigi hetek tananyagában végig FAE, azaz *visszatevéses* mintavételekkel dolgoztunk. Most nézzük meg, hogy mia helyzet akko, ha a **mintavételünk visszatevés nélküli véletlen mintavétel**, azaz **EV** minta.

Az okoskodásunk onnan indul, hogy azt mondjuk, hogy ha a minta kiválasztási aránya kicsi, azaz **ha a teljes adatsokaságunknak csak egy nagyon kis százalékát választjuk ki, akkor a FAE és EV mintavétel lényegében ugyan az**. Hiszen ebben az esetben még FAE mintavétrel esetén is nagyon kicsi az esélye, hogy a visszatevés miatt ténylegesen ismétlődés legyen, azaz többször is kiválasszuk ugyan azt az egyedet a mintánkba.

A minta **kiválasztási arányát** korábbi jelöléseinkkel élve $\frac{n}{N}$ -nek jelöljük. Tehát, ha $\frac{n}{N} \rightarrow 0$, akkor $FAE \approx EV$.

Ha viszont $\frac{n}{N}$ egy nagyobb érték, akkor viszont az **EV mintának pontosabbnak, azaz kisebb mintavételi hibájúnak kell lennie, minta FAE mintának**, hiszen minden mintaelem az EV mintában biztos, hogy új információt hoz be a mintába, míg FAE esetben lehet ismétlődés a visszatevés miatt.

Ezt a tényt pedig **úgy képezzük le a konfidencia-intervallum képleteinkben**, hogy FAE mintavételeknél tanult standard hiba képleteket egyszerűen beszorozzuk $\sqrt{1 - \frac{n}{N}}$ -nel. Tehát:

$$SH_{EV} = SH_{FAE} \times \sqrt{1 - \frac{n}{N}}$$

Ez a képlet azért végzi azt, amit mi akarunk, mert **ha nagyon 0 közeli a kiválasztási arány**, akkor az SH_{FAE} -t gyakorlatilag 1-gyel szorozzuk, azaz **nem változtatunk rajta semmit**. Ha pedig a kiválasztási arány **érdemben nagyobb, mint 0**, akkor pedig egy 0 és 1 közti számmal szorozzuk SH_{FAE} -t, így a **szorzat ekkor biztosan kisebb lesz, mint SH_{FAE} volt**.

Aki nem hiszi, nyugodtan **kirajzoltathatja Pythonban, az EV korrekciós tényező, azaz a $\sqrt{1 - \frac{n}{N}}$ viselkedését** különböző $\frac{n}{N}$ kiválasztási arányok mellett:

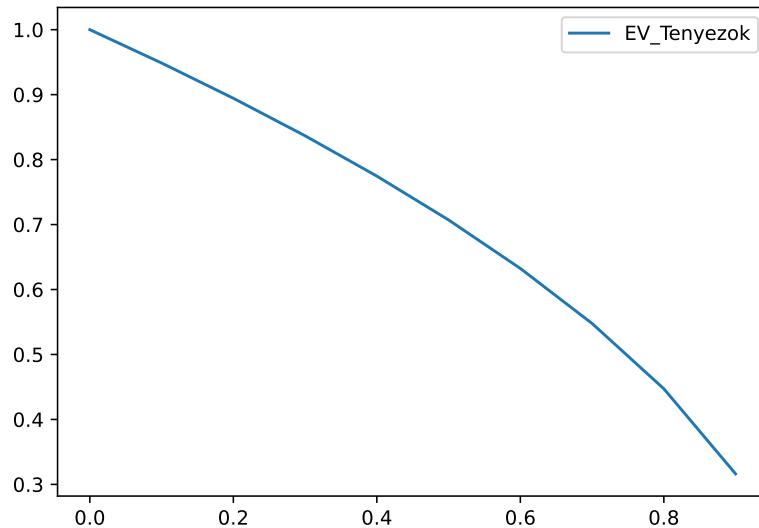
```
# Elemzéshez és ábrázoláshoz szükséges csomagok betöltése
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats

# Üres lista létrehozása az EV tényezők tárolására
EV_Lista = []
# Vizsgált kiválasztási arányok listájának létrehozása
# 0% és 90% közötti arányokat nézünk 10%-pontos lépésközzel
# Felső határ 1 = 100% a nyílt intervallum miatt
KivalArany_Lista = np.arange(0, 1, 0.1)

# Ciklus indítása
for AktualisKivalArany in KivalArany_Lista:
    EV_Lista.append(np.sqrt(1-AktualisKivalArany))

# Vizsgált kiválasztási arányok és a mért EV korrekciós tényezők data frame-be rendezése
# Ahol a kiválasztási arányok a sorindexek
EV_Data = pd.DataFrame(EV_Lista, columns=['EV_Tenyezok'], index = np.arange(0, 1, 0.1))

# Ábrázolás a 'plot' metódussal: nem kell paraméterezni, mert csak egy oszlopunk van
EV_Data.plot()
plt.show()
```



Látszik, hogy ahogy nő a kiválasztási arány, annál kisebb százalékát kell venni SH_{FAE} -nek az SH_{EV} számolása során. Éljen! :)

Nézzük meg ezt az egész rendszert a gyakorlatban az átlag intervallumbecslése során, nagy mintás esetben!

8.1.1. Átlag becslése EV mintákóból

Töltsük be a HKF_Jovedelem.xlsx című Excel fájl tartalmát egy **pandas** data frame-be! A táblában $n = 8306$ db magyar háztartás két ismérvét (oszlopát) látjuk:

1. Milyen típusú településen található a háztartás (Község, Többi város, Nagyváros, Budapest)
2. A háztartás éves jövedelmét 2019-ben, ezer Ft-ban kifejezve

Az adatok a KSH Jövedelmi és Életkörülmény Adatfelvételéből (rövidítve *HKÉF* vagy *HKF*) származnak. A KSH linkelt leírása szerint az **adatok a magyar háztartások sokaságából visszatevés nélkül véletlen, azaz EV mintaként kezelhetők**. Ez alapján egy nagyon fontos infó, hogy 2019-ben a magyar háztartások teljes száma 4 111 240 db volt. Tehát, a **sokaságunk teljes elemszáma $N = 4111240$ db**.

Lássuk is a beolvasást a **pandas** data frame-be!

```

HKF = pd.read_excel("HKF_Jovedelem.xlsx")
HKF.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 8306 entries, 0 to 8305
## Data columns (total 3 columns):
##   #   Column      Non-Null Count  Dtype  
##   --  --          -----          ----- 
##   0   HaztartasID    8306 non-null   int64  
##   1   TelepulesTipus 8306 non-null   object  
##   2   EvesJovEft     8306 non-null   float64
## dtypes: float64(1), int64(1), object(1)
## memory usage: 194.8+ KB

```

Remek: megvan a két keresett oszlopunk mind az $n = 8306$ db háztartásra, plusz egy ID oszlop az elején, amit nem fogunk semmire sem használni. :)

Ezek után simán bevethetjük az átlagos háztartási jövedelem 99%-os megbízhatóságú intervalumbecslésére a jól bevált `stats.norm.interval` című függvényt. Hiszen a $\Delta = k \times SH$ képletben a nagy, $n = 8306$ -os elemszám miatt a megbízhatósági szorzó simán jöhet standard normális eloszlásból, azaz $k = z_{1-\frac{\alpha}{2}}$. Az SH pedig akkor egy EV minta SH -ja, ami az előző fejezet alapján $SH_{FAE} \times \sqrt{1 - \frac{n}{N}}$ módon meghadható a függvény `scale` paraméterében.

```

N = 4111240
n = len(HKF)

stats.norm.interval(
    confidence=0.99,
    loc=np.mean(HKF.EvesJovEft),
    scale=stats.sem(HKF.EvesJovEft)*np.sqrt(1-n/N))

```

```
## (4567.729318773609, 4730.116789870932)
```

Tehát, viszzatevés nélküli véletlen mintavételeként kezelve a HKF adatokat, azt kapjuk, hogy **egy átlagos magyar háztartás éves jövedeleme 4568 és 4730 ezer Ft, azaz 4.5 és 4.7 millió Ft között mozog 99%-os valószínűséggel.**

A kiválasztási arányunk ugyebármost nagyon kicsi: $\frac{n}{N} = \frac{8306}{4111240} = 0.002$, szóval az eredmény lényegében nem különbözik attól, mintha az egészet az EV korrekciós szorzó nélkül csináltuk volna végig a műveletet:

```

stats.norm.interval(
    confidence=0.99,
    loc=np.mean(HKF.EvesJovEft),
    scale=stats.sem(HKF.EvesJovEft))

```

8.1. INTERVALLUMBECSLÉS VISSZATEVÉS NÉLKÜLI EGYSZERŰ VÉLETLEN (EV) MINTÁKBÓL 239

```
## (4567.647175824164, 4730.198932820376)
```

A várható éves jövedelem így is kb. 4.5 és 4.7 millió Ft között mozog 99%-os valószínűséggel.

8.1.2. Arány becslése EV mintákból

Ugyan ez a logika teljesen jól működik az arányok becslése esetén is. Ez nem véletlen, hiszen letisztáztuk a 7. fejezetben, hogy az arány, mint statisztikai mutató valójában egy olyan adatsor átlaga, hogy az arányban vizsgált tuljadonsággal bíró megfigyelések 1-gyel vannak kódolva, a többiek pedig 0-val.

Csináljuk is meg egy új oszlopan azt a 0 – 1 adatsort, amivel mondjuk a budapesti háztartások aránya becsülhető lesz.

```
HKF['BP_e'] = np.where(HKF.TelepulesTipus=="Budapest", 1, 0)  
np.mean(HKF.BP_e)
```

```
## 0.18552853359017576
```

Szuper, meg is vagyunk, a megfigyelt mintában a budapesti háztartások aránya 18.55%.

Ezek után, ha ezen az új oszlopon alkalmazzuk a `stats.norm.interval` függvényt az 1.1. fejezetben látott módon a `scale` paraméterben a $\sqrt{1 - \frac{n}{N}}$ korrekciós tényezővel, akkor meg is van a **budapesti háztartások sokaságjárának 99%-os megbízhatóságú intervallumbecslése**:

```
N = 4111240  
n = len(HKF)  
  
stats.norm.interval(  
    confidence=0.99,  
    loc=np.mean(HKF.BP_e),  
    scale=stats.sem(HKF.BP_e)*np.sqrt(1-n/N))
```

```
## (0.1745523489357643, 0.19650471824458723)
```

Szóval az **összes magyar háztartásból kb. 17.5% – 19.6% lehet budapesti**, 99% valószínűséggel.

8.2. Átlag becslése Arányosan Rétegzett (AR) mintákból

A KSH a HKF adatbázisáról nem csak simán azt állítja, hogy egyszerű véletlen (EV) minta a magyar háztartásokból, hanem, hogy **településtípus szerint arányosan rétegzett EV mita**. Ez azt jelenti, hogy a 4 db településtípusból (Budapest, Nagyváros, Többi város, Község) **külön-külön** vettek EV mintát, úgy, hogy a végső $n = 8306$ elemű mintában az egyes településtípusok aránya annyi legyen, mint a teljes sokaságban (összes magyar háztartás) a településtípusok aránya. Tehát a minta tökéletesen reprezentatívrá lett beállítva a településtípusra.

Ennek a mintavételi technikának a lényege, hogy ha **meg akarom becsülni a magyar háztartások átlagos éves jövedelmét**, akkor a standard hibában **NEM KELL** számolnom a településtípusok **KÖZÖTTI** jövedelem szórással! Hiszen ezt a külső szórást már kezelte a településtípusonkénti **külön-különmintavétel!!** Magyarul, az

$$SH_{EV} = \frac{s}{\sqrt{n}} \times \sqrt{1 - \frac{n}{N}}$$

Képlet helyett elég a standard hibában csak a belső korrigált **mintaszórással**, s_b -vel számolnom:

$$SH_{AR} = \frac{s_b}{\sqrt{n}} \times \sqrt{1 - \frac{n}{N}}$$

Mivel Stat. I-ből és az 3.4. fejezetből tudjuk, hogy egy **számértékű ismérvteljes varianciája, a külső és belső variancia összege**, így ha a külső variancával nem kel számolni a standard hibában, csak a belsővel, akkor **kijelenthető, hogy az arányosan rétegzett mintavétel becslési hibája az átlagra kisebb, mint az EV (vagy a FAE) mintavételek esetén:** $SH_{AR} \leq SH_{EV}$

Nézzük is meg akkor a dolgot élesben! Ugyebár tudjuk, hogy a háztartások átlagos éves jövedeleme a teljes magyar népességben 99%-os valószínűséggel 4.5 és 4.7 millió Ft körül mozog, ha a HKF adatokat EV mintavéteként kezeljük. Most a SH-t rendesen a $SH_{EV} = \frac{s}{\sqrt{n}} \times \sqrt{1 - \frac{n}{N}}$ képlettel számoljuk ki, és nem vetjük be a `stats.sem` függvényt.

```

N = 4111240
n = len(HKF)
s = np.std(HKF.EvesJovEFT)

sh_ev = s/np.sqrt(n)*np.sqrt(1-n/N)

```

8.2. ÁTLAG BECSLÉSE ARÁNYOSAN RÉTEGZETT (AR) MINTÁKBÓL 241

```
stats.norm.interval(
    confidence=0.99,
    loc=np.mean(HKF.EvesJovEft),
    scale=sh_ev)

## (4567.734206576373, 4730.111902068167)
```

Viszont, azt mondja az AR minta, hogy a standard hiba $\frac{s}{\sqrt{n}}$ részében elég csak a településtípusok szerinti belső szórással, s_b -vel számolni. Ehhez viszont kell egy aggregált tábla településtípusonként a

- mintaelemszámokról (n_j)
- mintaátlagokról (\bar{y}_j)
- korrigált mintaszórásokról (s_j)
- sokasági elemszámokról (N_j)

Láthatjuk, hogy a településtípusokat, azaz a rétegzett mintavétel rétegeit j indexszel jelöljük. Ezt az aggregált táblát a data frame-k groupby és agg metódusaival könnyen összehozhatjuk, ahogy a 3.4. fejezetben is láthattuk pl.

```
Segéd = HKF.groupby('TelepulesTipus').agg(
    Elemszam = ('EvesJovEft', 'count'),
    Reszatlagok = ('EvesJovEft', np.mean),
    KorrigalatlanSzorasok = ('EvesJovEft', np.std)
)
```

```
## <string>:1: FutureWarning: The provided callable <function mean at 0x000002140ED913A0> is currentl
## <string>:1: FutureWarning: The provided callable <function std at 0x000002140ED914E0> is currentl
```

Segéd

	Elemszam	Reszatlagok	KorrigalatlanSzorasok
## TelepulesTipus			
## Budapest	1541	5395.202989	3686.786788
## Község	2765	4344.417552	2580.432303
## Nagyváros	1786	4790.141025	2803.906596
## Többi város	2214	4395.863000	2516.663955

A fenti kód nem korrigált szórásokat számol technikailag, de a mi szempontunkból ez most nem oszt nem szoroz, mert a legkisebb elemszámú rétegen ($n_{BP} = 1541$) is a korrekció csak $\sqrt{\frac{n}{n-1}} = \sqrt{\frac{1541}{1541-1}} = 1.0006$, tehát elhanyagolható szorzótényező.

De bármikor tudjuk alkalmazni a korrekciós tényezőt a szórásokon egy új oszloppban:

```
Segéd['KorrigaltSzorasok'] = np.sqrt(Segéd.Elemszam/(Segéd.Elemszam-1)) * Segéd.Korrig
```

	Elemszam	Reszatlagok	KorrigalatlanSzorasok	KorrigaltSzorasok
## TelepulesTipus				
## Budapest	1541	5395.202989	3686.786788	3687.983602
## Község	2765	4344.417552	2580.432303	2580.899054
## Nagyváros	1786	4790.141025	2803.906596	2804.691894
## Többi város	2214	4395.863000	2516.663955	2517.232500

Viszont a rend kedvéért **egészítsük ki** az aggregált segédtáblát a **sokasági elemszámokkal**. Itt **kijelöljük, hogy arányosan rétegzett mintavételről van szó**. Tehát, a rétegek sokasági elemszáma a teljes sokasági elemszámhoz úgy aránylik, mint a rétegek mintabeli elemszáma a teljes mintavétel elemszámához, azaz

$$\frac{n_j}{n} = \frac{N_j}{N}, \forall j$$

Ebből gyorsan ki tudjuk számolni minden réteg sokasági elemszámát, hiszen tudjuk, hogy $N = 4111240$ háztartást jelent. A végén természetesen egészre kerekítjük az N_j -ket:

```
Segéd['SokasagiElemszam'] = round(Segéd.Elemszam/np.sum(Segéd.Elemszam) * N, 0)
```

	Elemszam	Reszatlagok	...	KorrigaltSzorasok	SokasagiElemszam
## TelepulesTipus			...		
## Budapest	1541	5395.202989	...	3687.983602	762752.0
## Község	2765	4344.417552	...	2580.899054	1368598.0
## Nagyváros	1786	4790.141025	...	2804.691894	884021.0
## Többi város	2214	4395.863000	...	2517.232500	1095869.0
##					
## [4 rows x 5 columns]					

Ezzel is megvagyunk. Akkor minden adott, hogy kiszámoljuk a belső szórást a következő képlettel, ami szintén ismerős az 3.4. fejezetből.

$$s_b = \sqrt{\frac{\sum_j n_j \times s_j^2}{n}}$$

Ezt gyorsan ki is tudjuk számolni Pythonban `numpy` függvények segítségével.

8.2. ÁTLAG BECSLÉSE ARÁNYOSAN RÉTEGZETT (AR) MINTÁKBÓL

```
belső_szórás = np.sqrt(np.sum(Segéd.Elemszam * Segéd.KorrigaltSzorasok**2)/np.sum(Segéd.Elemszam))
belső_szórás
```

```
## 2849.7867055139272
```

Tehát, egy konkrét **háztartás** éves **jövdeleme a megfigyelt mintában várhatóan kb.** $s_b = \pm 2.849$ milió Ft-tal (2849 ezer Ft-tal)** tér el saját településtípusának átlagos jövedelmétől**.

Ha az 5.3. fejezetben bemutatott számolási módját veszem figyelembe a korrigált mintaszórásnak, ami szerint $s^2 = \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2 / (n - 1)}$, akkor úgy lenne korrekt, hogy ha a belső szórást számoló formulában is mindenhol az n -t és n_j -ket $n - 1$ -nek és $n_j - 1$ -eknek venni, mint a „sima” s^2 nevezőjében. Tehát, a korrigált szórás sajátosságait figyelembe vevő belső szórás képlet a következő:

$$s_b = \sqrt{\frac{\sum_j (n_j - 1) \times s_j^2}{n - 1}}$$

Ezt is gyorsan ki is tudjuk számolni Pythonban numpy függvények segítségével.

```
belső_szórás_korrecióval = np.sqrt(np.sum((Segéd.Elemszam-1) * Segéd.KorrigaltSzorasok**2)/(len(Segéd.Elemszam)-1))
belső_szórás_korrecióval
```

```
## 2849.2301124754485
```

Látjuk, hogy az eredmények érdemben különbségek egymástól, mert a rétegek mintaelemszámai nagyok (ezres nagyságrendűek), így ± 1 ide-oda nem számít. :)

```
[belső_szórás, belső_szórás_korrecióval]
```

```
## [2849.7867055139272, 2849.2301124754485]
```

Kisebb elemszámoknál (olyan $n_j \approx 30$ körül) viszont már jelentős lehet a különbség, és érdemesebb az $n_j - 1$ -eket használó formulát alkalmazni, mert az konzisztens a korrigált szórás formulájával. Mi most a rétegek nagy elemszámai, tehát az ezres nagyságrendű n_j -k miatt az először kiszámolt **belső_szórás** értékkel megyünk tovább.

Ha van első szórásunk, akkor pedig gyorsan meg is van az **arányosan rétegzett standard hiba**, ami kb. 0.28 ezer Ft-tal (280 Ft-tal) **kisebb, mint a sima EV módra számolt standard hiba**:

```
N = 4111240
n = len(HKF)

sh_ar = belső_szórás/np.sqrt(n) * np.sqrt(1-n/N)
sh_ar

## 31.23757005707566
```

```
sh_ev
```

```
## 31.519498452027698
```

Ez pedig érezte magát a **némlileg szűkebb, azaz kisebb** $\Delta = SH_{AR} \times z_{1-\frac{\alpha}{2}}$ becslési hibahatárral bíró 99%-os konfidencia-intervallumban:

```
stats.norm.interval(
    confidence=0.99,
    loc=np.mean(HKF.EvesJovEFT),
    scale=sh_ar)
```

```
## (4568.460405997593, 4729.385702646948)
```

Arányos rétegzéssel a változatlan megbízhatóságú konfidencia-intervallum 4567 – 4730 ezer Ft-ról 4568 – 4729 ezer Ft-ra **szükül**.

Na jó, ne áltassunk magunkat: **ez szinte nulla csökkenés mind a standard hibában, mind a Δ -ben!** Tehát kb az AR mintavétel nem ért semmit. Miért van ez? Lássuk a magyarázatot!

8.2.1. Az AR mintavételek hatékonysága

Az, hogy mennyire jó az AR mintavétel az EV-hez képest attól függ, hogy mekkora a külső szórás mértéke, amit el tudunk hagyni azzal, hogy csak a belső szórással számolunk a standard hibában a teljes szórás helyett. Ezt pedig az határozza meg, hogy a rétegzéshez használt minőségi ismérő hány százalékban határozza meg annak a mennyiségi ismérvnek az alakulását, aminek az átlagát becsülni szeretnénk. Magyarul a **variancia-hányados** kell nekünk:

$$H^2 = \frac{s_k^2}{s^2} = 1 - \frac{s_b^2}{s^2}$$

És hát valóban a 2. képlettel kiszámolva azt láthatjuk, hogy a **településtípus alig határozza csak meg a háztartások éves jövdelemének alakulását**:

8.2. ÁTLAG BECSLÉSE ARÁNYOSAN RÉTEGZETT (AR) MINTÁKBÓL245

```
round((1-belső_szórás**2/s**2) * 100, 2)
```

```
## 1.78
```

Mindössze 1.78%-ban magyarázza a jövdelemek alakulását a településtípus. Ez megegyezik azzal hány **SZÁZALÉKKAL** csökkenti az **AR** mintavétel az **EV** standard hiba négyzetét:

```
round((sh_ar**2/sh_ev**2 - 1) * 100, 2)
```

```
## -1.78
```

Másképpen fogalmazva az **arányos rétegzés annyi SZÁZALÉKRA csökkenti a SH^2 -et, amennyi az $1 - H^2$ értéke.** Innentől kezdve ezt a csökkentés hívjuk **relatív hatásfoknak**, azaz:

$$Rel = \frac{SH_{AR}^2}{SH_{EV}^2} = 1 - H^2 = \frac{s_b^2}{s^2}$$

Ezek az összefüggések Pythonban is szépen kiszámíthatók.

```
(sh_ar**2/sh_ev**2)
```

```
## 0.9821908631514376
```

```
belso_szorras**2/s**2
```

```
## 0.9821908631514374
```

Tehát, az **arányos rétegzés miatt 98.2%-RA csökken csak a standard hiba négyzete.**

Természetesen az, hogy maga a **standard hiba** hány **százalékRA csökken,** azt a fenti összefüggés, zóval a **relatív hatásfok gyöke** adja meg.

```
(sh_ar/sh_ev)
```

```
## 0.991055428899634
```

belső_szórás/s

```
## 0.991055428899634
```

Szóval, a standard hiba az arányos rétegzés miatt csak 99.1%-RA, azaz 0.9%-KAL csökkent csak le, mivel a rétegzéshez használt településtípus csak 1.78%-ban magyarázza csak az éves jövedelmek alakulását, aminek az alakulását becsülni akartuk.

Mindazonáltal az átlagbecslés háromféle mintavételi mód (FAE, EV, AR) szerinti standard hibája között felállítható a következő összefüggés:

$$SH_{FAE} \geq SH_{EV} \geq SH_{AR}$$

Hiszen az EV standard hiba valamivel kisebb, mint a FAE a $\sqrt{1 - \frac{n}{N}}$ korrekciós tényező miatt, és az AR standard hiba valamivel kisebb, mint az EV amiatt, mert csak belső szórást használunk a képletben a teljes helyett:

$$\frac{s}{\sqrt{n}} \geq \frac{s}{\sqrt{n}} \times \sqrt{1 - \frac{n}{N}} \geq \frac{s_b}{\sqrt{n}} \times \sqrt{1 - \frac{n}{N}}$$

9. fejezet

Hipotézisvizsgálat alapjai

9.1. A hipotézisvizsgálat alapgondolata

Istenigazából a **hipotézisvizsgálat** csak egy **alternatív mód a konfidencia-intervallumok** mellett egy **statisztikai mutatószám (paraméter)** mintavételi hibájának figyelembe vételére, amikor a mutatószám egy mintából számított értékéből akarjuk megismerni a valós, sokasági értékét. Azonban, a **hipotézisvizsgálat gondolatvilága** a kérdést némi legeltérően közelíti meg, mint az eddig tanult konfidencia-intervallumok.

Hipotézisvizsgálat során megfogalmazunk egy **állítást vagy szebb szóval hipotézist** egy **statisztikai mutatószám** (átlag, arány, medián, szórás, stb.) **valós, sokasági értékéről**, és utána **megpróbáljuk állítani, hogy ezt a felvetést/hipotézist a megfigyelt mintaadatok alátámasztják-e vagy sem**. Most így alapesetben a paraméteres statisztikai hipotézisvizsgálatokat vagy rövidebben az úgynevezett **paraméteres (statisztikai) próbákat vizsgáljuk**, mert a **megfogalmazott hipotézisünk mindig egy statisztikai paraméter** (azaz statisztikai mutatószám) **valós, sokasági értékéről** szól majd egyelőre.

Nézzünk meg ismét egy $n = 100$ elemű **FAE** mintát a **2022-es Balaton átúszást teljesítők sokaságából** a LIDLBalaton2022.xlsx fájl alapján. Ahogy az 5. fejezetben is néztük, ebben a fájlban a Balaton átúszás résztvevőinek neve, neme és percben mért időeredménye található. Ez az adatsor lesz most is a **sokaságunk**.

```
# Elemzéshez és ábrázoláshoz szükséges csomagok betöltése
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
```

```
# Adatbeolvasás data frame-be
Balcsi = pd.read_excel("LIDLBalaton2022.xlsx")

Balcsi.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 9751 entries, 0 to 9750
## Data columns (total 3 columns):
## #   Column Non-Null Count Dtype 
## ---  -----  --------------  ----- 
## 0   Nev     9751 non-null   object 
## 1   Nem     9751 non-null   object 
## 2   PERC    9751 non-null   float64
## dtypes: float64(1), object(2)
## memory usage: 228.7+ KB
```

Szuper, vegyük is ki azt az $n = 100$ elemű FAE mintát! Rögzítsük meg a `random_state` paraméter 1992-nek, hogy mindenkor ugyan azt a 100 elemű véletlen mintát kapjuk!

```
BalcsiMinta = Balcsi.sample(n = 100, replace = True, random_state = 1992)

BalcsiMinta.info()

## <class 'pandas.core.frame.DataFrame'>
## Index: 100 entries, 7991 to 5727
## Data columns (total 3 columns):
## #   Column Non-Null Count Dtype 
## ---  -----  --------------  ----- 
## 0   Nev     100 non-null   object 
## 1   Nem     100 non-null   object 
## 2   PERC    100 non-null   float64
## dtypes: float64(1), object(2)
## memory usage: 3.1+ KB
```

Namámost. A teljes sokaság átlagos átúszási idejéről 6 db különböző állítást tudok megfogalmazni egy konkrét érték, pl. az 2.5 óra, azaz 150perc viszonylatában. A valós, sokasági átlagos időeredményt most is μ -vel jelöljük:

1. Optimista vagyok, és azt mondjam, hogy gyorsak voltak a népek és az átlagos átúszási idő a sokaságban kisebb, mint 150 perc: $\mu < 150$
2. Pesszimista magyarként azt is mondhatom, hogy lassú volt az úszótömeg és az átlagos időeredmény a sokaságban nagyobb, mint 150 perc: $\mu > 150$

3. De még azt is el tudom képzelni, hogy a valós átlag úszási idő épp 150 perc: $\mu = 150$
4. Vagy abban is hihetek, hogy a sokaságban az átlag időeredmény minden csak nem 150 perc: $\mu \neq 150$
5. A *félíg optimista* mondhatja azt, hogy a sokasági átlagos időeredmény *legfeljebb* 150 perc: $\mu \leq 150$
6. Kizárással alapon a *félíg pessimista* pedig úgy fog vélekedni, hogy a sokasági átlagos úszási idő *legalább* 150 perc: $\mu \geq 150$

Oké, akkor van 6 db elméletem...azaz *hipotézisem*. Nézzük meg mennyi az **áttag a megfigyelt 100 elemű mintában**:

```
BalcsiMinta.PERC.mean()
```

```
## 164.4403333333334
```

No, a megfigyelt 100 átúszó alapján a mintaátlag kb. 164 perc. Ez alapján a 2. állítás, a pessimista ürge mondása tünik igaznak, az **áttag nagyobb, mint** 150 perc. NODE! Amit itt látunk az csak 100 ember átlagos ideje, a teljes sokaságban az átlagos átúszási idő ettől eltérhet! Szebben fogalmazva, a 164 és 150 közti különbség simán betudható a mintavételi hibának **is!! Ezért végünk hipotézisvizsgálatot**, hogy eldöntsük, hogy a $\mu > 150$ hipotézis (állítás) igaznak vehető-e még a mintavételi hibával együtt is.

Igazából, ha belegondolunk, akkor minden a 6 állítás visszavezethető ahhoz a kérdéshez, hogy a valós, sokasági átlag, μ **eltér-e** annyira a 150-től, hogy az **eltérés már meghaladja a mintavételi hibát**. Ezt úgy mondjuk szépen, hogy **azt vizsgáljuk, SZIGNIFIKÁNS-e az eltérés a valós μ és 150 között!**

Most itt éppen ki tudjuk számolni a teljes sokaság ismeretében, hogy $\mu = 167$ perc, ami **tényleg magasabb, mint** 150.

```
Balcsi.PERC.mean()
```

```
## 167.52914060096398
```

A gyakorlatban viszont mindezt NEM tudjatjuk, hiszen csak az $n = 100$ elemű minta áll rendelkezésünkre!! Szóval marad az, hogy **hipotézisvizsgálattal ellenőrizzük**, hogy a $\bar{y} = 164$ megfigyelt mintaátlag **szignifikánsan eltér-e** a 150-től.

Az eddigi logikánk alapján létre tudunk hozni egy úgynevezett **nullhipotézist** (H_0) és **alternatív hipotézist** (H_1) az **eredeti állításunkból**.

A pontos logika itt a következő. A H_0 -ban minden olyan állítást fogalmazunk meg, ami a **vizsgált statisztikai paraméter tekintetében megengedi az egyenlőséget**. A H_1 pedig az eredeti állítástól függ:

- Ha az eredeti állítás megengedi az egyenlőséget, akkor az állítás H_0 -ba kerül és a H_1 -ben tagadjuk az eredeti állítást. Ebből adódóan azt várjuk, hogy a hipotézisvizsgálat végén a H_0 állítás bizonyuljon igaznak. Ebben az esetben azt szeretnénk, hogy a valós sokasági átlag átúszási idő ne különbözzön szignifikánsan a 150 perctől.
- Ha az eredeti állítás nem engedi meg az egyenlőséget, akkor az a H_1 -be „költözik”, és a H_0 -ban az eredeti állítást tagadjuk. Ebből adódóan azt várjuk, hogy a hipotézisvizsgálat végén a H_1 állítás bizonyuljon igaznak. Ebben az esetben azt szeretnénk, hogy a valós sokasági átlag átúszási idő szignifikánsan különbözzön a 150 perctől.

Ezen elveket figyelembe véve a 6 eredeti állításunkhoz a következő H_0 és H_1 párok adhatók meg:

1. Állítás: $\mu < 150 \parallel H_0 : \mu \geq 150 \parallel H_1 : \mu < 150 \parallel$ Állítás a H_1 -ben található.
2. Állítás: $\mu > 150 \parallel H_0 : \mu \leq 150 \parallel H_1 : \mu > 150 \parallel$ Állítás a H_1 -ben található.
3. Állítás: $\mu = 150 \parallel H_0 : \mu = 150 \parallel H_1 : \mu \neq 150 \parallel$ Állítás a H_0 -ban található.
4. Állítás: $\mu \neq 150 \parallel H_0 : \mu = 150 \parallel H_1 : \mu \neq 150 \parallel$ Állítás a H_1 -ben található.
5. Állítás: $\mu \leq 150 \parallel H_0 : \mu \leq 150 \parallel H_1 : \mu > 150 \parallel$ Állítás a H_0 -ban található.
6. Állítás: $\mu \geq 150 \parallel H_0 : \mu \geq 150 \parallel H_1 : \mu < 150 \parallel$ Állítás a H_0 -ban található.

Az 1-2. és 5-6. állításokban, ahol nem „tiszta” egyenlőség van H_0 -ban szoktunk úgynevezett technikai nullhipotézist, H_0^T -t alkalmazni. Ez csak annyit jelent, hogy a nem „tiszta” egyenlőséggel adott H_0 -t átírjuk tiszta egyenlőségre. Tehát, pl. az 1. és 5. állítás technikai nullhipotézissel:

1. Állítás: $\mu < 150 \parallel H_0^T : \mu = 150 \parallel H_1 : \mu < 150 \parallel$ Állítás a H_1 -ben található.
2. Állítás: $\mu \leq 150 \parallel H_0^T : \mu = 150 \parallel H_1 : \mu > 150 \parallel$ Állítás a H_0 -ban található.

A hipotézisvizsgákat vagy más néven **statisztikai próbákat** a H_1 -ben adott relációs jelek alapján három csoportba kategórizálják:

- Ha H_1 -ben \neq jel van, akkor **kétoldali próba**
- Ha H_1 -ben $<$ jel van, akkor **baloldali próba**
- Ha H_1 -ben $>$ jel van, akkor **jobboldali próba**

9.1.1. A p-érték fogalma

Ha szépenfelírtuk a megfelelő H_0 és H_1 párokat, akkor meghatározzuk, hogy a megfigyelt mintánk alapjánk melyik tekinthető igaznak a kettő közül. Erre a célra egy **p-érték** nevű statisztikai mutatószámot használunk. A **p-érték** megadja mekkora a valószínűsége, hogy a H_0 elutasításával hibás döntést hozunk Szóval, ha azt kapom, hogy **p-érték = 30%**, akkor azt mondhatom, hogy **30% valószínűséggel HIBÁZOM**, ha azt mondomb, hogy H_0 egy hamis állítás a mintánk alapján.

Ez alapján a **döntési szabály** H_0 és H_1 között a következő. Amennyiben a **p-érték túl magas, elfogadjuk H_0 -t**, mivel elutasítása túl nagy hibavalószínűsséggel járna. Ellenben ha a **p-érték túl alacsony**, akkor elutasítjuk H_0 -t és **elfogadjuk H_1 -et**, mivel H_0 elutasítása nagyon alacsony hibavalószínűsséggel jár.

A kérdés az hogyan döntöm el, hogy egy p-érték túl magas/alacsony-e? Erre van nekünk az úgynevezett **szignifikancia-szint**, amit α -val jelölünk. A szignifikancia-szint a **maximális elfogadott hibavalószínűség H_0 elutasításához**.

Tehát, ha azt mondjuk, hogy $\alpha = 5\%$, akkor 5% alatti p-érték esetén **elutasítjuk H_0 -t, mivel elutasítása kisebb hibával jár, mint a megengedett maximum**. Másrészről, ha a p-értékünk 5% feletti, akkor meg **elfogadjuk H_0 -t, mivel elutasítása nagyobb hibával jár, mint a megengedett maximum**.

Ezen a ponton tehát a **következő hinnénk**:

- **p-érték > $\alpha \rightarrow H_0$**
- **p-érték $\leq \alpha \rightarrow H_1$**

De természetesen **AZ ÉLET SOSEM ILYEN EGYSZERŰ!!** a fő probléma itt abból jön, hogy **simán lehetnek olyan mintaadataink, hogy a meghozott döntés marhára függ attól mit mondok pontosan α -nak!** Pl. oké, maximum megenegedett hibavalószínűségem $\alpha = 5\%$. A p-értékre meg kapok egy 4.8%-ot. Na akkor $\alpha = 5\%$ esetén elutasítanánk H_0 -t, de egy $\alpha = 3\%-nál$ már épp elutasítanánk H_0 -t. Emiatt alkalmazzuk a **szokásos szignifikancia-szintek tartományát, ami 1% – 10% között mozog**. Nem akarunk 10%-nál nagyobb megenegedett α hibát, mert az már tényleg magas hibavalószínűség. Másrészről meg 0% hibavalószínűségünk meg csak úgy lehet, ha megfigyeljük az egész sokaságot, és tudjuk mennyi a vizsgált statisztikai paraméter valós, sokasági értéke. Ekkor meg pont a mintavétel lényegét veszítjük el, hogy nem kell megfigyelni minden múltbeli és jövőbeli adatpontot ahhoz, hogy döntenи tudjunk a vizsgált mutatószámunk valós, sokasági értékéről. Ez utóbbi $\alpha = 0\%$ eset hasonló ahhoz, amikor azt mondunk, hogy a 100% megbízhatóságú konfidenzia-intervallumnak sincs

semmi értéelme, mert akkor azt mondánánk, hogy a mintavételünk alapján a vizsgált statisztikai mutatószámunk (paraméterünk) bárhol lehet $\pm\infty$ között... :) Továbbá, majd **szimulációkból látni fogjuk, hogy 1% és 10% közti p-értéket simán produkál a valóságban igaz és hamis H_0 állítás is!**

Szóval, az előbbi családregényünk alapján az tűnik egy normálisabb megoldásnak, ha **azt mondjuk, hogy 1% és 10% közti p-érték esetén nem választunk H_0 és H_1 között**, mert ekkor a döntésünk nagyon érzékeny lenne a konkrét α megválasztására. Ha egy 1% és 10% közti p-értéket produkálnak a mintaadataink, akkor a legjobb, amit tehetünk, hogy **addig növeljük a mintaméretet (n -t)** amíg a **p-érték egyértelműen 1% alá vagy 10% fölé nem kerül**. Ezt a módszert hívja a statisztika szekvenciális analízisnek, és a klinikai gyógyszerkísérletekben és az ipari minőségbiztosításban rendszeresen alkalmazzák. A módszer alapjait pedig egy Wald Ábrahám nevű magyar statisztikus dolgozta ki az Amerikai Légierőnek a II. világháború alatt (zsidó származása miatt itthon „nem kellett”). Szóval, mint minden fontos területet a világon, ezt is magyarok találták ki. :)

Tehát, az egy fokkal **korrektebb döntés szabályok hipotézisvizsgálat esetén:**

- **p-érték** $> 10\% \rightarrow H_0$
- $1\% < \text{p-érték} \leq 10\% \rightarrow \text{nincs döntés}$
- **p-érték** $\leq 1\% \rightarrow H_1$

További fejfájásokat okoz, hogy nagyon óvatosan kell bálni az 1% és 10% közti p-értékkel azért is, mert a **p-érték és az α csak egy nézőpontból írják le a hipotézisvizsgálat során elkövethető döntési hibát!** A p-érték és az α csak az úgynevezett **ELSŐFAJÚ hibáról ad információt: a valóságban igaz H_0 elutasításának valószínűségéről!** NODE, van nekünk **MÁSODFAJÚ hibánk** is, ami a **valóságban hamis H_0 téves elfogadását jelenti**. Az a rossz hír, hogy ennek a másodfajú hibának a valószínűségét nem **tudjuk kiszámolni a megfigyelt minta alapján!!** Ez a hipotézisvizsgálat nagy átka: ha **elfogadunk egy H_0 -t**, akkor igazából **nem tudjuk, hogy mekkora valószínűsséggel hibázunk**, csak azt **tudjuk, hogy mennyi a H_0 elutasításának hibavalószínűsége**, mert ez pont a **p-érték**. Ezért nagyon sokan fogalmaznak szándékosan ilyen furán a H_0 -ról, hogy nem elfogadjuk a H_0 -t, hanem **nem tudjuk elutasítani a H_0 -t**.

A probléma alaposabb megértésében szerintem remek szolgálatot tesz az alábbi mém. :)



Most pedig nézzük meg hogyan kell kiszámolni a p-értéket a megfigyelt minta alapjánabban az esetben, amikor a vizsgált statisztikai paraméter az átlag, azaz μ !

9.2. A sokasági átlagra vonatkozó *t*-próba

A p-érték számításához mindenkor szükségünk van egy próbafüggvény nevű statisztikai mutatóra, amit csak a megfigyelt mintaadatokból ki tudunk mindenkor számolni. Eztán a p-értéket a próbafüggvényből egy nevezetes valószínűségi eloszlás (standard normális, t-eloszlás, khi-négyzet, stb.) alapján tudjuk kiszámolni.

Jelöljük a sokasági átlag **HIPOTETIKUS** értékét μ_0 -nak. Ez az érték, amit az állításban feltételezünk a valós, sokasági átlagról. Az 1. fejezetben ez volt a 150 perc. Ezzel a jelöléssel az alábbi módon néz ki az átlagra vonatkozó próbafüggvény:

$$\frac{\bar{y} - \mu_0}{\frac{s}{\sqrt{n}}}$$

Tehát, a próbafüggvényünk most nem más, mint a **minta és hipotetikus átlag különbsége osztva az átlag standard hibájával**.

Ha a H_0 -ban jól okoskodunk, és **tényleg sikerült a valós μ sokasági átlagot venni a μ_0 elvi** (hipotetikus) átlagnak, akkor sok-sok mintavétel esetén a próbafüggvényünk $n - 1$ szabadságfokú t-eloszlást követ:

$$\frac{\bar{y} - \mu}{\frac{s}{\sqrt{n}}} \sim t(n - 1)$$

A képlet mögött az az elv nyugszik, hogy a $\mu = \mu_0$ **technikai nullhipotézist** (H_0^T) alkalmazzuk mindenkor a 6 különböző típusú H_0 és H_1 párosnál, így meg tudjuk majd adni a próbafüggvény eloszlását sok-sok mintavétel esetén.

Ezt az egészet gyorsan ellenőrizni tudjuk, ha **kiveszünk** 10000 db $n = 100$ elemű FAE mintát a Balaton átúszók sokaságából, ahogy az 5. fejezetben is tettük. Most csak visszatölthjük a 10000 db FAE mintát tartalmazó Excel táblát egy data frame-be, amit a 4.3.1. fejezetben hoztunk létre. Ez az Excel fájl innen elérhető.

```
MintaVetelek100Elem = pd.read_excel("MintaDataFrame.xlsx")
MintaVetelek100Elem
```

```
##           Elem1        Elem2        Elem3      ...       Elem98       Elem99     Elem100
## 0    164.800000  129.066667  156.166667  ...  207.350000  159.666667  165.883333
## 1    152.516667  212.483333  152.900000  ...  307.266667  119.783333  128.216667
## 2    145.666667  185.266667  169.516667  ...  167.733333  228.366667  215.633333
## 3    185.683333  120.333333  201.250000  ...  182.766667  177.666667  112.450000
## 4    117.483333  142.350000  320.266667  ...  188.566667  189.166667  99.916667
## ...
## 9995  162.333333  83.350000  146.750000  ...  164.250000  131.933333  128.183333
## 9996  100.416667  161.433333  187.366667  ...  160.483333  168.416667  209.300000
## 9997  146.450000  160.783333  165.483333  ...  158.816667  167.733333  183.400000
## 9998  139.250000  140.466667  130.933333  ...  153.616667  112.366667  196.050000
## 9999  147.316667  173.450000  106.100000  ...  185.616667  171.800000  135.166667
##
## [10000 rows x 100 columns]
```

Oké, az eredményből látjuk is, hogy úgy néz ki a data frame, hogy **1 sor tartalmaz 1 db 100 elemű mintát és a mintaelemeket** (tehát a mintába besorolt versenyző percben mért időeredményét) **az oszlopokban tároljuk**.

Kiszámoljuk mindenegyes mintavételre a minta átlagos időeredményét és azok korrigált szórását. Csak figyeljünk, hogy az alkalmazott numpy függvényeket **axis = 1** paraméterrel használjuk, hogy ne oszlopok, hanem sorok szerint vegyük az átlagokat és a szórásokat. Plusz, figyeljünk, hogy a függvényeket minidg csak az első 100 oszlopra engedjük rá, hiszen ott vannak a tényleges mintaelemek. Ezt ugyebár a data framek **iloc** metódusával tudjuk biztosítani, ahogy az 5.2. fejezetben csináltuk.

```
MintaVetelek100Elem['Atlagok'] = np.mean(MintaVetelek100Elem.iloc[:,0:100], axis=1)
MintaVetelek100Elem['Szorasok'] = np.std(MintaVetelek100Elem.iloc[:,0:100], axis=1, dd

MintaVetelek100Elem

##           Elem1        Elem2        Elem3      ...       Elem100     Atlagok     Szorasok
## 0    164.800000  129.066667  156.166667  ...  165.883333  171.261667  42.397078
## 1    152.516667  212.483333  152.900000  ...  128.216667  158.026500  44.288357
## 2    145.666667  185.266667  169.516667  ...  215.633333  172.278833  44.809282
```

```

## 3    185.683333 120.333333 201.250000 ... 112.450000 163.434167 45.453383
## 4    117.483333 142.350000 320.266667 ... 99.916667 166.552833 45.814202
## ...
## ...
## 9995 162.333333 83.350000 146.750000 ... 128.183333 163.682333 41.282298
## 9996 100.416667 161.433333 187.366667 ... 209.300000 164.332833 42.020990
## 9997 146.450000 160.783333 165.483333 ... 183.400000 163.702333 37.947348
## 9998 139.250000 140.466667 130.933333 ... 196.050000 166.423833 39.749948
## 9999 147.316667 173.450000 106.100000 ... 135.166667 173.050333 48.227237
##
## [10000 rows x 102 columns]

```

Ezek után pedig ki tudjuk számolni a próbafüggvényünket egy olyan H_0 és H_1 párosra, amiről tudjuk, hogy a sokaságban H_0 igaz, és egy olyanra, ahol tudjuk, hogy a sokaságban H_0 hamis:

- **IGAZ** H_0 esete: $H_0 : \mu = 167$ és $H_1 : \mu < 167$
- **HAMIS** H_0 esete: $H_0^T : \mu = 150$ és $H_1 : \mu > 150$

```

mu_0_igaz = np.mean(Balcsi.PERC) # ez ekkor a valós sokasági átlag
mu_0_hamis = 150 # tetszőleges érték
n = 100 # mintaméret

```

```

MintaVetelek100Elem['ProbaFv_H0Igaz'] = (MintaVetelek100Elem.Atlagok - mu_0_igaz)/(MintaVetelek100Elem['SokasagiAtlag'])
MintaVetelek100Elem['ProbaFv_H0Hamis'] = (MintaVetelek100Elem.Atlagok - mu_0_hamis)/(MintaVetelek100Elem['SokasagiAtlag'])

```

MintaVetelek100Elem

```

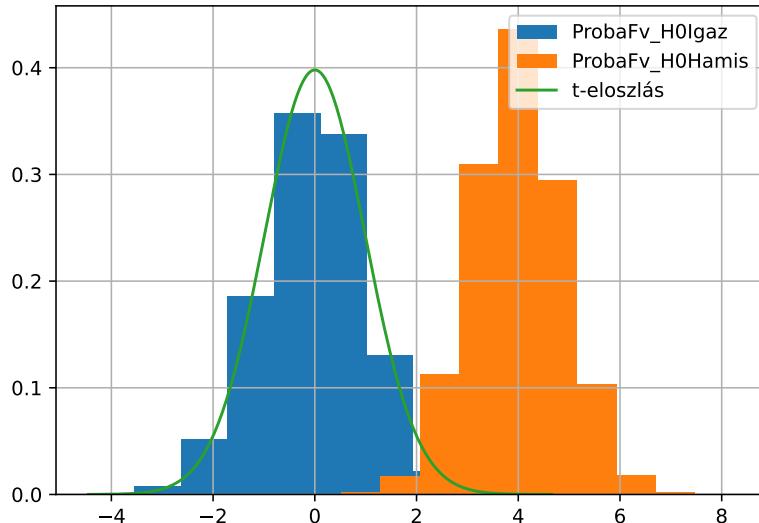
##           Elem1      Elem2 ... ProbaFv_H0Igaz ProbaFv_H0Hamis
## 0    164.800000 129.066667 ...     0.880373      5.014890
## 1    152.516667 212.483333 ...    -2.145630      1.812327
## 2    145.666667 185.266667 ...     1.059980      4.971924
## 3    185.683333 120.333333 ...    -0.900917      2.955592
## 4    117.483333 142.350000 ...    -0.213101      3.613035
## ...
## ...
## 9995 162.333333 83.350000 ...    -0.931830      3.314334
## 9996 100.416667 161.433333 ...    -0.760645      3.410875
## 9997 146.450000 160.783333 ...    -1.008452      3.610880
## 9998 139.250000 140.466667 ...    -0.278065      4.131787
## 9999 147.316667 173.450000 ...     1.144829      4.779526
##
## [10000 rows x 104 columns]

```

Okké! Akkor miután megvannak a kétféle esetben a próbafüggvényeink, nézzük meg a kétféle próbafüggvények hisztogramja a 10000 db

mintavétel alapján hogyan alakul a $t(n - 1)$, azaz $t(100 - 1)$ eloszlás súrűségfüggvényéhez képest. A súrűségfüggvény hisztogramhoz való illeszkedését ábrázoló kód teljes mértékben az 3.2.3. és 3.3. fejezetekben lévő kódok logikáját követi.

```
# Igaz és Hamis H0 esetén próbafüggvények hisztogramja
# A 'label' paraméter majd a színcímkékkel adjuk az ábrán
MintaVetelek100Elem.ProbaFv_H0Igaz.hist(density = True, label="ProbaFv_H0Igaz")
MintaVetelek100Elem.ProbaFv_H0Hamis.hist(density = True, label="ProbaFv_H0Hamis")
# A súrűségfüggvény x tengelye az Igaz H0 esetén vett próbafüggvények min-max tartományában
x_tengely = np.arange(np.min(MintaVetelek100Elem.ProbaFv_H0Igaz), np.max(MintaVetelek100Elem.ProbaFv_H0Hamis))
# A t-eloszlású súrűségfüggvény (pdf) értékek az 'y' tengelyen
y_tengely = stats.t.pdf(x_tengely, df = n-1)
# Súrűségfüggvény felrakása a hisztogramra
plt.plot(x_tengely, y_tengely, label = "t-eloszlás")
# Színcímkék megjelenítése a jobb felső sarokban
plt.legend(loc="upper right")
# Ábra mutatása
plt.show()
```



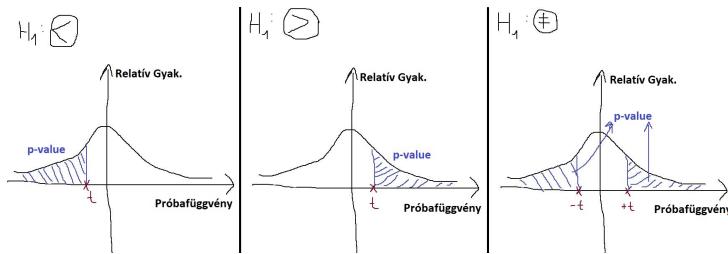
Szuper! Tehát igaz H_0 esetén tényleg t-eloszlást követ a próbafüggvény **a sok-sok mintavételünk esetén**. Hamis H_0 esetén pedig valami más szimmetrikus eloszlás rajzolódik ki a hisztogramon, ami NEM a t-eloszlás. De hogy ez most konkrétan mi, nem annyira érdekel minket. :)

Ebből, hogy a próbafüggvény t-eloszlást követ sok-sok mintavétel esetén, ha H_0

igaz, ki tudjuk számolni a p-értéket. gyakorlatilag területeket számolunk a t -eloszlás sűrűségfüggvénye alatt. A konkrét számoláshoz végig kell gondolnunk, hogy mi a legjobb eset H_0 szempontjából a próbafüggvényre nézve:

- **Kétoldali** próbák ($H_0 : \mu = \mu_0$ és $H_1 : \mu \neq \mu_0$) esetén: ha a **próbafüggvény pontosan** $0 \rightarrow$ ez az elvi eset arra utal, hogy a hipotetikus átlag és a valós, sokasági átlag ugyanaz, így a H_1 0 valószínűséggel következik be.
- **Baloldali** próbák ($H_0 : \mu \geq \mu_0$ és $H_1 : \mu < \mu_0$) esetén: ha a **próbafüggvény =** $+\infty \rightarrow$ ez az elvi eset arra utal, hogy a sokasági átlag (μ) az pont $+\infty$, aminél minden μ_0 kisebb, így a H_1 0 valószínűséggel következik be.
- **Jobboldali** próbák ($H_0 : \mu = \mu_0$ és $H_1 : \mu > \mu_0$) esetén: ha a **próbafüggvény =** $-\infty \rightarrow$ ez az elvi eset arra utal, hogy a sokasági átlag (μ) az pont $-\infty$, aminél minden μ_0 nagyobb, így a H_1 0 valószínűséggel következik be.

Ez alapján, ha van egy ismert próbafüggvény értékem, amit t -vel jelölök, akkor a p-érték a $t(n - 1)$ eloszlásból az alábbi ábrán látható módon számítható ki:



Látható, hogy minden esetben a p-érték úgy lesz kiszámolva, hogy ha a konkrét t próbafüggvényünk messzebb kerül a H_0 számára legjobb esettől, akkor a p-érték csökken → hiszen egyre kisebb és kisebb hibát vétünk, ha elutasítjuk a H_0 -t!!

Mindezek alapján akkor számoljuk ki a `scipy` csomag `stats.t.cdf` függvénye segítségével a p-értéket mind a 10000 db mintában a korábban vizsgált két H_0 és H_1 párnakra, ahol egyszer igaz, egyszer pedig hamis volt a H_0 :

- **IGAZ** H_0 esete: $H_0 : \mu = 167$ és $H_1 : \mu < 167$
- **HAMIS** H_0 esete: $H_0 : \mu = 150$ és $H_1 : \mu > 150$

Figyeljünk arra, hogy az 1– rész a `stats.t.cdf` függvényen a hamis H_0 esetben a **jobboldali próba** miatt van:

```

n = 100 # mintaméret

MintaVetelek100Elem['p_ertek_H0Igaz'] = stats.t.cdf(MintaVetelek100Elem.ProbaFv_H0Igaz
MintaVetelek100Elem['p_ertek_H0Hamis'] = 1-stats.t.cdf(MintaVetelek100Elem.ProbaFv_H0Hamis

MintaVetelek100Elem

##           Elem1      Elem2 ... p_ertek_H0Igaz p_ertek_H0Hamis
## 0    164.800000 129.066667 ...     0.809605     0.000001
## 1    152.516667 212.483333 ...     0.017174     0.036483
## 2    145.666667 185.266667 ...     0.854133     0.000001
## 3    185.683333 120.333333 ...     0.184909     0.001950
## 4    117.483333 142.350000 ...     0.415843     0.000239
## ...
## 9995 162.333333 83.350000 ...     0.176846     0.000642
## 9996 100.416667 161.433333 ...     0.224338     0.000469
## 9997 146.450000 160.783333 ...     0.157848     0.000241
## 9998 139.250000 140.466667 ...     0.390771     0.000038
## 9999 147.316667 173.450000 ...     0.872480     0.000003
##
## [10000 rows x 106 columns]

```

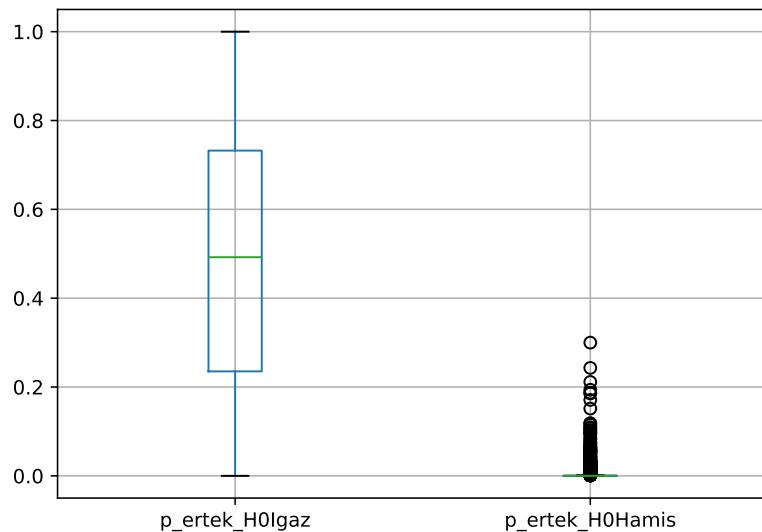
Remek! Az első és az utolsó 5 db minta alapján az elvárt eredményeket kaptuk a p-értékekre: magas értékek igaz H_0 , alacsony értékek hamis H_0 esetén. Pontosabb értelmezéshez nézzük meg **az 5., azaz a data frame-ben 4-es sorindexszel rendelkező minta** esetét:

- Igaz H_0 esetén 41.58%-os valószínűséggel hibázunk, ha elutasítjuk H_0 -t ezen minta alapján.
- Hamis H_0 esetén 0.02%-os valószínűséggel hibázunk, ha elutasítjuk H_0 -t ezen minta alapján.

Ugyanakkor a 2. mintában látjuk, hogy a p-érték csak 1.7% igaz H_0 és 3.6% hamis H_0 esetén. Szóval, a „**nem hozunk döntést, ha a p-érték 1% – 10% között mozog**”, egy elég jogos szabálynak tűnik! :)

Hasonlóan megerősíti az 1% – 10% közötti p-értékek esetén vett óvatosságot, ha az igaz és hamis H_0 esetén vizsgált p-értékeket doboz ábrán vizsgáljuk:

```
MintaVetelek100Elem.boxplot(column=['p_ertek_H0Igaz', 'p_ertek_H0Hamis'])
plt.show()
```



Ahogyan vártuk is, igaz H_0 esetén a p-értékek középső 50%-a, azaz interkavrtlis trejedelme (*IKT*) látványosan magasabban van, mint hamis H_0 esetében. Ugyanakkor, azt is nagyon fontos látni az ábrán, hogy **a nagy p-értékek kilógó értéknek számítanak hamis H_0 esetén, de a kis p-értékek nem kilógó értékek igaz H_0 esetében!** Tehát, néha megérhető szigorúbb α -t használni, mint 1%. Sokan csak akkor utasítják el H_0 -t, ha a p-érték kisebb, mint 0.1% = 0.001, szóval $\alpha = 0.001$ mellett dolgoznak.

9.2.1. A *t*-próba beépített függvényivel

Azért van némi szerencsénk, mert az **átlagra vonatkozó t-próba p-értékét szépen ki lehet számolni beépített scipy függvényel** is!

Vegyük például az 1. fejezet elején kivett $n = 100$ elemű mintát, amit a **BalcsiMinta** című Python objektumban tároltunk le, ami egy 100×3 -as data frame:

```
BalcsiMinta.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 100 entries, 7991 to 5727
## Data columns (total 3 columns):
## #   Column Non-Null Count Dtype
## ---  -----  -----  -----
```

```
## 0 Nev      100 non-null   object
## 1 Nem      100 non-null   object
## 2 PERC     100 non-null   float64
## dtypes: float64(1), object(2)
## memory usage: 3.1+ KB
```

Vizsgáljuk meg ezen a mintán az alábbi H_0 és H_1 párost:

- $H_0^T : \mu = 150$
- $H_1 > 150$

Mivel H_1 -ben $>$ relációs jel lakik, így **jobboldali** lesz a hipotézisvizsgálat. Tudjuk, hogy itt a H_1 **lesz igaz, hiszen láttuk korábban, hogy a teljes sokaságban az átlagos átúszási idő 167.5 perc**. Tehát, valami **jó alacsony p-értéket** várunk!

Ez szépen ki is jön manuálisan is! Először kiszámoljuk a $\frac{\bar{y} - \mu_0}{\frac{s}{\sqrt{n}}}$ **próbafüggvényt** $\mu_0 = 150$ mellett, hiszen a hipotézisekben azt nézzük, hogy a valós, sokasági átlag szignifikánsan nagyobb-e, mint ez a feltételezett (hipotetikus) 150 perces átlag.

```
elviátlag = 150
mintaátlag = np.mean(BalcsiMinta.PERC)
s = np.std(BalcsiMinta.PERC, ddof=1)
n = len(BalcsiMinta)

próbafv = (mintaátlag - elviátlag) / (s/np.sqrt(n))
próbafv
```

```
## 3.719542604812644
```

Remek! Ebből gyorsan is meg is van a $t(n - 1)$ eloszlásból (`stats.t.cdf` függvényel) a p-érték. Most ugye a **próbafüggvény felé esés valószínűségét számoljuk, hiszen jobboldali a próba**.

```
p_ertek_t_elo = 1 - stats.t.cdf(próbafv, df = n-1)
p_ertek_t_elo * 100 # százalékban írom ki
```

```
## 0.01655831079531156
```

A p-érték alapján elmonható, hogy a most vizsgált 100 elemű minta alapján csak 0.017% a valószínűsége a hibának, ha H_0 -t elutasítjuk. Ez kisebb még a legkisebb szokásos szignifikancia-szintnél, az $\alpha = 1\%$ -nál

is, így meghozzuk ezt a döntést és H_0 -t elutasítjuk. Ez alapján pedig H_1 elfogadható, azaz a sokaságban az átlagos átúszási idő szignifikánsan (mintavételi hibát meghaladó mértékben) nagyobb, mint 150 perc. Most tudjuk, hogy helyesen döntöttünk, mivel láttuk, hogy a sokaságban az átlagos átúszási idő 167.5 perc.

Mindez szerencsére elintézhető a `scipy` csomag `stats.ttest_1samp` függvényével is. Ha a függvény

- első paraméterében megadjuk az aktuális mintánk elemeit tartalmazó oszlopot egy data frame-ból,
- második paraméterben az elvi átlagot, azaz μ_0 -t,
- az `alternative` paraméterben pedig a próba oldalát H_1 relációs jele alapján
 - lehetséges értékek: ‘two-sided’, ‘less’, ‘greater’, rendre a \neq , $<$, $>$ relációs jeleket jelölök H_1 -ben

Akkor a függvény megadja a t-próba p-értékét és a próbafüggvényt is:

```
stats.ttest_1samp(BalcsiMinta.PERC, 150, alternative='greater')
```

```
## TtestResult(statistic=3.7195426048126436, pvalue=0.00016558310795314698, df=99)
```

Szuper, ugyan úgy megvan a próbafüggvény (3.7195) és a 0.017%-os p-érték is! :)

9.2.2. A *t*-próba előfeltételei

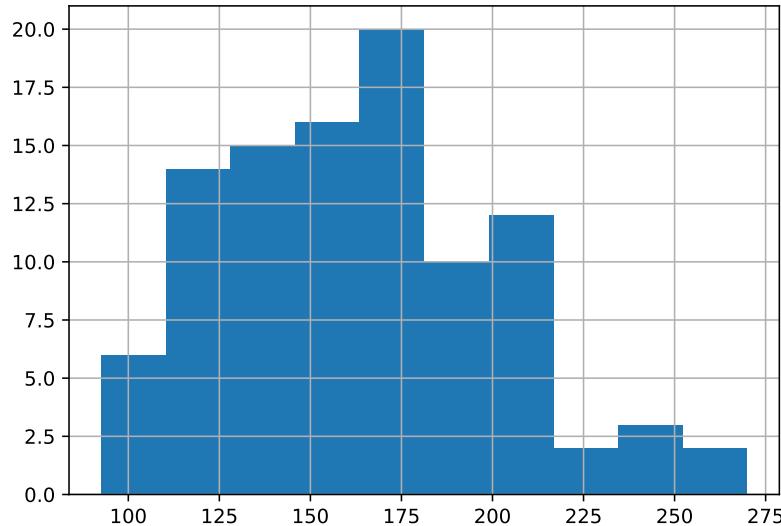
Természetesen, ennek a t-próbás hipotézisvizsgálatnak is vannak előfeltételei, mint ahogy a különböző konfidencia-intervallum képleteknek is voltak. **Ha ezek a feltételek nem teljesülnek, akkor bármilyen p-értékre, azt nem használhatjuk döntésre H_0 -ról és H_1 -ról**, hiszen az alkalmazott képletek mögötti követelmények az adatokkal szemben NEM teljesülnek.

A t-próbának alapvetően két feltételére kell odafigyelni:

1. Amennyiben **kis mintánk** ($n \leq 30$) van, akkor feltesszük, hogy **az adatok, amikből a mintát vettük normális eloszlást követnek**
2. Ha a **minta elemszáma nagy** ($n > 30$), akkor **nem kell semmi extra előfeltételre figyelni**.

Most az $n = 100$ elemű mintánk a Balaton átúszók sokaságából nagy minta, hiszen $100 > 30$, de azért nézzünk rá az **időeredmény adatok eloszlására a minta alapján**:

```
BalcsiMinta.PERC.hist()
plt.show()
```



Hm, bár a nagy mintaelemszám miatt ez nem számít, de azért megnyugtató, hogy az átúszási idők eloszlása nem olyan durván tér el a normális eloszlás sűrűségfüggvényétől. Csak nagyon enyhén van egy kis jobbra elnyúló jellege, de semmi vészes. :)

9.3. A sokasági átlagra vonatkozó *z*-próba

Ugyebár az átlagra vonatkozó konfidencia-intervallumok esetén megfigyeltük, hogy **nagy szabadságfok** ($df > 30$) esetén a **t-eloszlás lényegében megegyezik a standard normális eloszlással**.

Azaz, kicsit matekosabban fogalmazva: **ha $n > 30$, akkor igaz nullhipotézis (H_0) esetén** (tehát, amikor μ_0 helyére épp a valós, sokasági átlagot, azaz μ -t helyettesítjük), a **próbafüggvényünk standard normális ($N(0,1)$) eloszlást követ**:

$$\frac{\bar{y} - \mu}{\frac{s}{\sqrt{n}}} \sim N(0, 1)$$

Ez alapján pontosan **ugyan azon az elven számíthatunk a próbafüggvényből p-értéket, amit a 2. fejezetben a t-eloszlásnál is alkalmaztunk**. Mivel a null- és alternatív hipotézis nem változott meg ($H_0^T : \mu = 150$ és $H_1 > 150$),

így marad a jobboldali próba, és mivel a mintaelémek sem változtak a 100 elemű mintánkban, így a **próbafüggvény is megegyezik a 2.1. fejezetben használittal**. Mivel maradt a jobboldali próba, így a **próbafüggvény fölé esés valószínűségét kell megadni a standard normális eloszlásban** a `stats.norm.cdf` függvény 1– verziójával.

```
p_ertek_norm_elo = 1 - stats.norm.cdf(próbafv)
p_ertek_norm_elo * 100 # százalékban írom ki

## 0.009979194034781536

p_ertek_t_elo * 100 # p-érték a t-eloszlából százalékban

## 0.01655831079531156
```

Mindkét p-érték nagyjából 0.01% körüli, szóval a standard normális és t-eloszlás között tényleg nincs érdemi különbség a p-érték számolás tekintetében.

Természetesen, mindez beépített függvényel is meg tudjuk oldani, csak ez most a `statsmodels` csomag `stats.weightstats` moduljában lakó `ztest` függvény lesz. A név (`z test = z próba`) onnan jön, hogy a standard ormális eloszlás jele a matekban z , így ez a hipotézisvizsgálat **z-próba** néven fut, míg az előző **t-próba** néven, mivel ott a p-értéket t-eloszlásból számoltuk. Paramétereiben a függvény ugyan úgy működik, mint a `scipy`-os `stats.ttest_1samp`:

- először megadjuk aktuális mintánk elemeit tartalmazó oszlopot egy `data frame`-ból
- utána jön a `value` paraméterben az elvi átlag (μ_0)
- végül pedig az alteranítv hipotézisben lévő relációs jelet adjuk meg szövegesen
 - lehetséges értékek: ‘two-sided’, ‘smaller’, ‘larger’, rendre a \neq , $<$, $>$ relációs jeleket jelölik H_1 -ben

Ezek alapján az alábbi módon néz ki a függvény használata Pythonban.

```
# függvény importja
from statsmodels.stats.weightstats import ztest

# függvény használata
ztest(BalcsiMinta.PERC, value = 150, alternative = 'larger')

## (3.7195426048126436, 9.979194034785673e-05)
```

Meg is van a 3.7-es próbafüggvényünk és a $0.0099\% \approx 0.01\%$ körüli p-értékünk. Győzelem!

9.4. A sokasági arányra vonatkozó *z-próba*

Természetesen, nem csak sokasági átlagokra (μ), hanem sokasági arányokra (P) is tudunk hipotéziseket tenni. Majd ehhez is tudunk p-értéket számolni és eldöntení, hogy az eredeti állításunkból felírt H_0 vagy H_1 tekinthető-e igaznak egy adott α szignifikancia-szinten.

Nézzük meg, hogy **vehető-e 1/3-nak a Balatont 3 óra felett átúszók aránya**. Matematikailag az állításom azt mondja, hogy a 3 óra = 180 perc felett átúszók sokasági aránya egyenlő 1/3-dal: $P = 1/3$. Mivel az **állítás engedi az egyenlőséget, így az a H_0 -ba kerül**. Ez alapján pedig a H_1 -ben **az eredeti állítást tagadjuk, és a H_0 -nak szurkolunk** a hipotézisvizsgálat során. Tehát, a H_0 és H_1 párunk az alábbi módra fog kinézni:

- $H_0 : P = 1/3$
- $H_1 : P \neq 1/3$
- Szurkolunk: H_0

A H_0 és H_1 párunkból következik, hogy az **elvi arány, azaz P_0 , amihez képest vizsgálódunk most az 1/3 érték!!**

Ezek után itt most mondanánk a 7.2. fejezet alapján, hogy készítünk a megfigyelt mintánkba egy olyan 0 – 1 értékekkel álló oszlopot, ahol 1 jelenti a 3 óránál hosszabb idő alatt átúszókat a mintában (arány szempontjából kedvező esetek a mintában), míg 0 a többieket (arány szempontjából kevdezőtlen esetek a mintában), és ennek az oszlopnak az átlagára vonatkozó hipotézisvizsgálatot végezünk (mivel $n = 100 > 30$ ez simán lehet z-próba).

Ezzel a fenti elvvel viszont van egy kis gond! Ugyanis **árányra vonatkozó hipotézisvizsgálat esetén a próbafüggvényünk az alábbi módon néz ki:**

$$\frac{p - P_0}{\sqrt{\frac{P_0(1-P_0)}{n}}} \sim N(0, 1)$$

A „**kis gond**” a képlet nevezőjével, azaz a **standard hibával van**. A képlet madártávlatból nézve azt mondja, hogy vegyük minta és hipotetikus arány különbségét ($p - P_0$) osztva az arány standard hibájával. Ami tényleg az átlagra vonatkozó hipotézisvizsgálat próbafüggvényének működési elve, amit láttunk is a 2. fejezetben. DE! A **standard hibában a képlet az elvi arányt** (P_0) használja a mintából számolt arány (p) helyett. Viszont, ha egy 0 – 1 értékekkel álló oszlop átlagára vonatkozó hipotézisvizsgálatként számolnánk végig ezt a próbafüggvényt, akkor bizony a nevezőben $\sqrt{\frac{p(1-p)}{n}}$ -gyel számolnánk $\sqrt{\frac{P_0(1-P_0)}{n}}$ helyett. Szóval mindenképpen egy külön függvényt kell itt használni a próbafüggvény számítására. A p-érték számolása standard normális ($N(0, 1)$) eloszlásból történik már úgy, mint az átlag esetében láttuk a

2. és 3. fejezetekben. Ezért is *z-próbaként* emlegetjük ezt a hipotézisvizsgálatot is. De ezt a p-érték számítást is elintézi majd nekünk a **statsmodels** csomag **proportions_ztest** függvénye.

A függvényhez először ki kell számolni a minta teljes elemszámát (n) és az arány szempontjából kedvező esetek számát (k), ami most esetünkben 180 percnél tovább úszók száma. Ezekből már akár a mintabeli arányt (p) is kiszámíthatjuk.

```
# mintaelemszám
n = len(BalcsiMinta)

# arány szempontjából kedvező esetek száma
kedvezo = np.sum(BalcsiMinta.PERC > 180)

# mintaarány
kedvezo/n

## 0.3
```

Ezzel látjuk, hogy a mintában a 180 percnél tovább úszók aránya kereken 30%, ami nem egyenlő az $1/3 \approx 0.333 = 33.3\%$ -al, így a H_1 néz ki igaz állításnak. Viszont, ez simán lehet a mintavételi hiba műve is, hiszen csak egy $n = 100$ elemű mintából dolgozom, nem látom az összes átúszó sokaságát. Épp ezért kell a p-érték, ami megadja, hogy mennyire valószínű, hogy az a tény, hogy a megfigyelt mintában $0.3 \neq 0.333$ szignifikáns eltérés mintaarány és elvi arány között.

Importáljuk is a függvényt a **statsmodels** csomagból, és használjuk is az előbb kiszámolt kedvező esetszámmal és teljes elemszámmal. Amit még meg kell adni neki az a **value** paraméteren az elvi arány, és az **alternative** paraméterben a H_1 relaciós jelét ugyan azokkal a kulcszavakkal, mint amiket a 3. fejezetben is használtuk a sima, átlagra vonatkozó **ztest** függvényben.

```
# függvény importja
from statsmodels.stats.proportion import proportions_ztest

# elvi arány megadása
P_0 = 0.25

# függvény használata
proportions_ztest(
    count = kedvezo,
    nobs = n,
    value = P_0,
    alternative = 'two-sided')
```

```
## (1.0910894511799616, 0.27523352407483437)
```

A p-értékünk magas, 46.6%, szóval ennyi esélyünk van a hibára, ha a mintavétel alapjánelutasítanánk H_0 -t. Ez jó magas hibaválsázínség, magasabb, mint a legmagasabb szokásos szignifikancia-szint (10%), így **azt mondjuk, hogy a H_0 -t nem tudjuk elutasítani**. Valós, sokasági aránya a 180 percnél lassabban úszóknak simán vehető 1/3-nak a mintaarány és az 1/3 közötti eltérés **NEM szignifikáns**.

A hipotézisvizsgálatnak, mivel z-próba és standard normális eloszlásból számol p-értéket, előfeltétele, hogy nagy mintánk van. Ez most azt jelenti, hogy a $n \times P_0$ és $n \times (1 - P_0)$ szorzat is legyen nagyobb mint 10. Tehát, **az arány szempontjából kedvező és kedvezőtlen esetek száma is legyen legalább 10 igaz nullhipotézis esetén.**

Lássuk, hogy teljesülnek-e akkor a feltételek:

```
# mintaelemszám
n = len(BalcsiMinta)
# elvi arány megadása
P_0 = 1/3

# feltétel ellenőrzése
n*P_0
```

```
## 33.33333333333333
```

```
n*(1-P_0)
```

```
## 66.66666666666667
```

A 33 és a 66 is 10-nél nagyobb szám, szóval rendben vagyunk! :)

9.5. A hipotézisvizsgálat menete madártávlatból

Az eddigiek alapján akkor azt mondhatjuk el, hogy igazából **bármilyen statisztikai mutatóra/paramétere** (átlag, arány, stb.) **vonatkozik a hipotézisvizsgálatunk**, annak minden 4 fő lépése van.

1. Az alapállításból **null-** és **alternatív hipotézisek** (H_0 és H_1) felírása
2. Megfigyelt n elemű mintaadatokból **próbafüggvény** számítása
3. A próbafüggvény és egy nevezetes eloszlás alapján **p-érték** számítása

4. A p-érték alapján annak eldöntése, hogy H_0 vagy H_1 vehető-e igaz állításnak a teljes sokaságban

Ez a 4 lépés tényleg minden hipotézisvizsgálat során ugyan ez lesz, és az 1. és 4. pontoknak is mindig ugyan az az elve, amiket eddig láttunk, ezek nem változnak. Bármi is történik, a H_0 és H_1 felírás és közöttük a döntés *p*-érték alapján fix szabályok szerint működik. Ami az egyes esetekben eltérő lehet, az a 2. és 3. pont, ahol a próbafüggvényeket és a p-értékeket számoljuk. De ezekre a pontokra szinte minden lesz beépített Python függvény, csak azoknak a paraméterezését és a használatok előfeltételét kell igazából jól tudni.

10. fejezet

Egymintás és Kétmintás próbák

10.1. A szórásra vonatkozó χ^2 -próba

Vegyük elő újra a ESS2020.xlsx fájlban található adatbázist! Emlékeztetőül álljon itt, hogy ez az adatbázis a 2020-ban végzett európai szociális felmérés (European Social Survey 2020 = ESS2020) 1849 magyar kitöltjének válaszait tartalmazza 14 kérdésre (plusz van egy *id* oszlop).

Ugyebár a 6.4. fejezet 2. feladatában azt mondta, hogy ha az adatbázis valamelyik oszlopában üres értéket találunk, akkor az azt jelenti, hogy az adott sorban lévő kitöltő nem válaszolt a kérdésre. Az adatbázisban szereplő kitöltők a teljes 18 év feletti magyar népességből vett véletlen mintaként kezelhetők.

Először is töltök be az adatbázist ismét az Excelból egy `pandas` data frame-be és nézzük meg az `info` metódussal milyen oszlopaink (azaz ismérveink) vannak!

```
# Elemzéshez és ábrázoláshoz szükséges csomagok betöltése
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats

# Adatbeolvasás data frame-be
ess = pd.read_excel("ESS2020.xlsx")
ess.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 1849 entries, 0 to 1848
```

```

## Data columns (total 15 columns):
## #   Column           Non-Null Count  Dtype  
## --- 
## 0   id               1849 non-null    int64  
## 1   PoliticalRadioTVPerDay_Minutes 1796 non-null    float64 
## 2   NetUsePerDay_Minutes          1099 non-null    float64 
## 3   TrustInParlament           1849 non-null    object  
## 4   PoliticalPartyPref         1849 non-null    object  
## 5   Education_Years            1830 non-null    float64 
## 6   WeeklyWork_Hours           685  non-null    float64 
## 7   Region                      1849 non-null    object  
## 8   County                      1849 non-null    object  
## 9   SecretGroupInfluenceWorldPol 1849 non-null    object  
## 10  ScientistsDecievePublic    1849 non-null    object  
## 11  COVID19                     1849 non-null    object  
## 12  ContactCOVID19             1849 non-null    object  
## 13  GetVaccince                1849 non-null    object  
## 14  SomeContactCOVID19        1849 non-null    object  
## dtypes: float64(4), int64(1), object(10)
## memory usage: 216.8+ KB

```

Láthatjuk, hogy megvan mind a 14+1 oszlopunk a megfelelő adattípusokkal. Hurrá! :)

Nézzünk most egy olyan **hipotézisvizsgálatot**, ami **szórásra vonatkozik**. Lehet egy olyan állításunk, ami szerint a teljes népességben a **heti munkaidő szórása legalább 15 óra**. Ezt a szokásos elvek alapján át tudjuk alakítani null- és alternatív hipotézissé. Ez az állítás a **legalább' kulcsszó miatt engedi az egyenlőséget (\geq)**, így az állítás maga egy H_0 lesz. Míg a H_1 ennek a tagadása lesz $<$, hiszen a H_0 és H_1 egymást kizáró állításként kell megfogalmazni. Tehát, ezek alapján az állítás a heti munkaidő sokasági szórásáról, a σ -ról az alábbi módon néz ki H_0 -al és H_1 -el:

- $H_0 : \sigma \geq 15$
- $H_1 : \sigma < 15$
- Szurkolunk: H_0

Ebben a felírásban az **elvi szórásunk a 15 óra**, hiszen a valós sokasági szórásnak a viszonyát ehhez képest vizsgáljuk majd a mintánk alapján. Ezt az elvi szórást $\sigma_0 = 15$ -nek jelöljük.

Ezzel megvolnánk a hipotézisvizsgálatunk 1. pontjával, a H_0 és H_1 felírásával. Most ugrunk a 2. és igazából a 3. pontra is, a próbafüggvény meghatározásához, majd a p-érték számolásához. **Szórás esetében a próbafüggvény** a következő alakot ölti:

$$\frac{(n-1)s^2}{\sigma_0^2} \sim \chi^2(n-1)$$

$A \sim \chi^2(n-1)$ jelölés azt mondja, hogy a p-értéket egy $n-1$ szabadságfokú χ^2 eloszlásból kell majd kiszámolnunk.

Sajnos, erre a próbafüggvényre és p-értékre nincs Python beépített függvény, így muszáj kiszámolnunk a dolgokat manuálisan!

Viszont, a próbafüggvényben minden betűt ismerünk. Ugye az előbb letisztéztük, hogy a σ_0 az elvi szórás, az n szokásos müödon a mintánk elemszáma és az s a korrigált (tehát torzítatlan becslést adó) mintaszórás. Ezeket az értékeket gyorsan ki tudjuk számolni. Az ESS adatbázisban a heti munkaórára vonatkozó adatai a kitöltőknek a `WeeklyWork_Hours` oszlopban laktaknak. Arra kell a számolásnál figyelni, hogy az ESS adatbázis `info()` metódusának eredményéből láthattuk, hogy **ebben a `WeeklyWork_Hours` oszlopban van egy rakat hiányzó érték**, így n -ünk az nem 1849 (data frame sorainak a száma), hanem csak 685. Ezt úgy tudjuk legkönnyebben figyelembe venni, hogy az n -t a `WeeklyWork_Hours` oszlop `count()` metódusával számítjuk ki, ami megadja az oszlopban a nem üres értékek számát.

```
elvi_szórás = 15
n = ess.WeeklyWork_Hours.count()
s = np.std(ess.WeeklyWork_Hours, ddof=1)
s
```

```
## 15.375457568374705
```

Láthatjuk, hogy a mintában a szórás $s = 15.37$, ami nagyobb mint a 15-ös elvi szórás. De **nem vehetjük automatikusan igaznak H_0 -t, hiszen ez a jelenség lehet csak a mintavételi hiba műve!** Épp azért kell p-értéket számolni, hogy megtudjuk, mekkora valószínűséggel lehet **ez a jelenség** (hogy a mintaszórás nagyobb, mint a 15-ös elvi érték) **csak a mintavételi hiba műve**.

Szóval, számoljuk csak ki ezt a próbafüggvényt! :)

```
próbafv = (n-1)*s**2/elvi_szórás**2
próbafv
```

```
## 718.6702741281487
```

Szép és jó ez az érték, de önmagában nem sokat mond. :) Ugorjunk is a 3. pontra, a p-érték számítására. Ugye azt megállapítottuk már az előbb, hogy a p-értéket most $n-1$ szabadságfokú χ^2 eloszlásból számoljuk. Plusz, **mivel a H_1 -ben < jel van, így a számítás balodali módon történik**, azaz a $\chi^2(n-1)$ eloszlásban a próbafüggvény alá esés valószínűsége érdekel. Az elv itt ugyan az, amit láttunk a 9.2. fejezetben.

Ezek alapján meg is van a p-értékünk a `stats.chi2.cdf` függvénnyel.

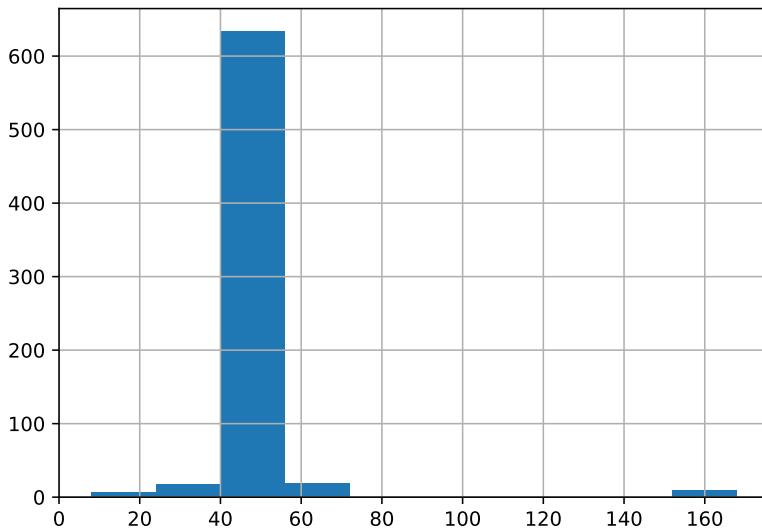
```
stats.chi2.cdf(próbafv, df=n-1)
```

```
## 0.8263764489186204
```

A p-érték = 82.6% lett, ami szerint a megfigyelt minta alapján a H_0 -t elutasítani 82.6% hibavalószínűsggel jár. Ez jó magas hibavalószínűség így „szabad szemmel” is, de technikailag ha nézem, akkor **ez a p-érték magasabb még a legmagasabb szokásos szignifikancia-szintnél**, a 10%-nál is, így a H_0 -t nem utasítjuk el, mert ez a döntés túl nagy hibával járna. Ez alapján pedig azt mondhatjuk, hogy a **heti munkaidők szórása a teljes népességenben (sokaságban) is legalább 15 órának kinéz**.

Sajnos ennek a χ^2 próbának ugyan az a baja, mint a χ^2 eloszlásos intervallumbecslésnek volt a 7.5. fejezetben: A próba elvégzésének előfeltétele, hogy az adataink normális eloszlásúak legyenek. tehát, a számított p-értékünk, a 82.6%, az akkor valid, ha az adatsor, aminek a szórását vizsgáljuk normális eloszlást követ. Ezt egy hisztogrammal gyorsan ellenőrizni is tudjuk.

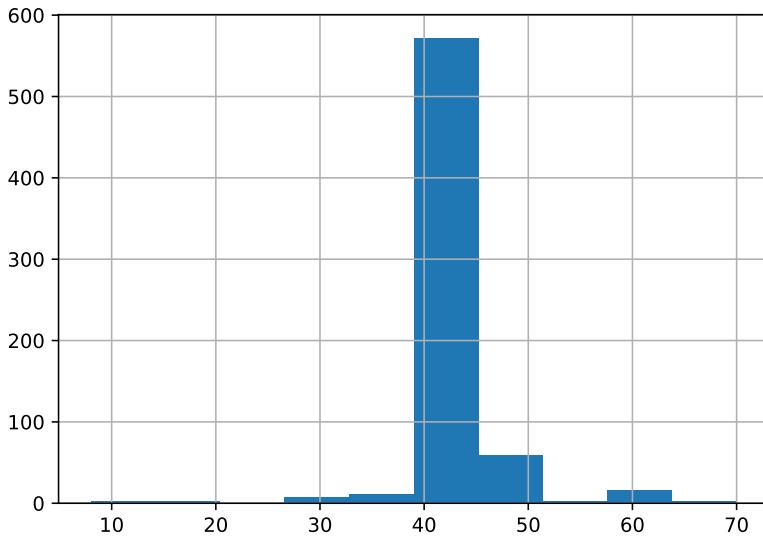
```
ess.WeeklyWork_Hours.hist()
```



Ajjaj, a munkaórák eloszlása csúcsosnak tűnik, mint a fene. Ez így nem egy normális eloszlás, az előfeltételünk nem teljesül, a számított p-értékünk nem megbízható ... *BigRIP* :(Esetleg próbálkozhatunk az estrém outlier 160 óra

kiszűrésével, hátha csak az okozza a csúcsosságot, de sajnos nem járunk sikkerrel, ez a hisztogram az outlier szűrés után is elég csúcsos.

```
ess.WeeklyWork_Hours[ess.WeeklyWork_Hours < 160].hist()
```



De amúgy ez nem meglepő: a legtöbb kitöltő valószínűleg a „standard” heti 40 órát (vagy ahhoz közelí értéket) adott meg válaszként, és ez okozza a csúcsosságot.

Amúgy, ez az előfeltétel az oka, hogy nincs erre a hipotézisvizsgálatra beépített függvény Pythonban: nagyon ritka, hogy egy adatsor szép normális eloszlású legyen, így a legtöbb esetben a próba nem megbízható, mivel a $\chi^2(n - 1)$ eloszlásból számított p-érték mögötti feltételek nem teljesülnek a legtöbb esetben. Emiatt ritka, hogy a próba valid eredményeket produkál, így egy programozó sem kódolta le Pythonban, mert ritkán kell ez a hipotézisvizsgálat gyakorlatban. Mi is inkább „történelmi okokból” tanítjuk. Meg hogy minél több dolgot lehessen számonkérni. :)

10.2. A kétmintás hipotézisvizsgálatok elve

Az eddig tanult hipotézisvizsgálatok az átlagra, arányra és szórásra úgynevezett egymintás hipotézisvizsgálatok voltak. Ez azt jelentette, hogy egy db statisztikai mutató (egy db átlag, egy db arány vagy éppen egy db szórás)

viszonyát vizsgáltuk valami elvi értékhez képest. Ez alapján fogalmaztuk meg az előfeltevéseinket, azaz hipotéziseinket.

Mostantól viszont úgynevezett **kétmintás hipotézisvizsgálatokkal** vagy más néven **kétmintás próbákkal** foglalkoznunk. Ez pedig azt fogja jelenti, hogy a **hipotéziseinkben** (előfeltevéseinkben) minden **két db statisztikai mutató** (két db átlag, két db arány, stb.) **viszonyát vizsgáljuk majd egymáshoz képest**.

Tehát, ha **példának az átlagot veszem**, mint statisztikai mutatót, akkor **egy olyan kétmintás állítást** mondhatok, hogy „**a férfi átlagfizetés magasabb, mint a női átlagfizetés Magyarországon**”. Kicsit matekosabban felírva: **Átlagfizu(Férfi) > Átlagfizu(Nő)**

Azért lesznek az ilyen két statisztikai mutatós állításokhoz tartozó hipotézisvizsgálatok kétmintás próbák, mert **az a logika, hogy 1 db mintánka van a férfi átlaghoz és 1 db mintánk van a női átlaghoz**. Tehát, összességében 2 db mintával dolgozunk. :)

A kérdés, hogyan lesznek ezekből a kétmintás állításokból null- és alternatív hipotézisek. A trükkünk az, hogy **átrendezzük az állítást leíró egyenlőtlenséget** úgy, hogy a két statisztikai mutató **KÜLÖNBSÉGÉNEK a viszonyát vizsgáljuk egy elvi értékhez** képest. Ez az elvi érték lesz az elvi eltérés, általában δ_0 -nak jelöljük. Arányok esetében néha szokás ϵ_0 -t is használni az elvi eltérés jelölésére.

Az előbbi átlagfizetéses példában, ha a sokasági átlagokat továbbra is μ -vel jelölöm, mint a korábbi hetek tananyagalban, akkor az **átrendezett állítás** a következő lesz:

$$\mu_{Fr_{fi}} - \mu_{Ni} > 0$$

És akkor itt $\delta_0 = 0$ esetünk van. Ebből az **eddigி elvek mentén lehet a különbségre felírni H_0 -t és H_1 -et**:

1. Ha az állítás megengedi az egyenlőséget, akkor az H_0 , egyébként H_1 -be megy.
2. A H_0 és H_1 egymást kizáró, ellentétes állítások.

Szóval, a férfi-női átlagkeresettes példánkon a H_0 és H_1 állítások a következők:

- $H_0 : \mu_{Fr_{fi}} - \mu_{Ni} \leq 0$
- $H_1 : \mu_{Fr_{fi}} - \mu_{Ni} > 0$
- Szurkolunk: H_1

Nézzünk egy olyan példát, amikor a $\delta_0 \neq 0$! Legyen az az állításunk, hogy „**a férfi átlagkereset legalább 100 ezer Ft-tal magasabb, mint a női**”. Ekkor azt csináljuk, hogy első körben írjuk fel úgy az állítást, mintha csak annyi

lenne, hogy „*a férfi átlagkereset legalább annyi, mint a női*”. Tehát, mintha a különbségre adott számértékünk (a 100 ezer Ft) ott sem lenne. A „*legalább*” kifejezés miatt ez egy \geq -s állítás lesz:

$$\mu_{Fr_{fi}} \geq \mu_{Ni}$$

És ezt a fenti egyenlőtlenséget egészítsük úgy ki, hogy belevesszük a 100 ezer Ft-os különbséget. A „*férfit átlagkereset legalább 100 ezer Ft-tal magasabb, mint a nőit*” azt jelenti másiképpen, hogy ha a **női átlagkeresethez még 100 ezret hozzáadok, akkor is nagyobb vagy egyenlő ennél a férfi átlagkereset**. Ez a megfogalmazás pedig alábbi módon néz ki az egyenlőtlenségünkben:

$$\mu_{Fr_{fi}} \geq \mu_{Ni} + 100$$

Innen től kezdve megvan az állításom, amit már könnyű **átrendezni** úgy, hogy **azt vizsgálja az egyenlőtlenség, hogy az átlagok különbsége hogyan viszonyul valami elvi, δ_0 eltéréshez**, ami most nekem a 100-as érték:

$$\mu_{Fr_{fi}} - \mu_{Ni} \geq 100$$

Tehát, ebben az állításban akkor $\delta_0 = 100$. És ezt az állítást a szokásos két alapelt mellett már gyorsan tudom H_0 -ra és H_1 -re szétszedni. Mivel az állítás engedi az egyenlőséget most, így az állításunkból egy nullhipotézis lesz, és az alternatív hipotézisben ezt az állítást meg tagadjuk, és vélgül a H_0 -nak szurkolunk, mert abban lakik az eredeti állításunk:

- $H_0 : \mu_{Fr_{fi}} - \mu_{Ni} \geq 100$
- $H_1 : \mu_{Fr_{fi}} - \mu_{Ni} < 100$
- Szurkolunk: H_0

Akkor, lássuk hogy működik a kétmintás H_0 és H_1 párokhoz a p-érték számolás pár gyakorlati példán! :)

10.3. Az átlagokra vonatkozó kétmintás z-próba

Az ESS adatbázis SecretGroupInfluenceWorldPol és Education_Years című oszlopait felhasználva ellenőrizhetünk egy olyan állítást a teljes magyar népességre (azaz sokaságra), ami szerint „**átlagban jobban iskolázottak** (több évet töltöttek tanulással), **akik nem hisznek abban, hogy egy titkos társaság irányítja a világpolitikát**”.

Ezt az alááítást úgy írhatjuk fel matematika egyenlőtlenséggé mint $\mu_{NemHisz} > \mu_{Hisz}$. Ha ezt az állítást az átlagok különbségére rendezzük át, akkor a $H_1 : \mu_{NemHisz} - \mu_{Hisz} > 0$ egyenlőtlenséget kapjuk. Ez alapján pedig

már látjuk is, hogy az átlagok elvi eltérése a sokaságban $\delta_0 = 0$. Ezt az állítást a szokásos két alapely mellett már gyorsan tudom H_0 -ra és H_1 -re szétszedni. Mivel az állítás *nem engedi* az egyenlőséget most, így az állításunkból egy H_1 lesz, és a H_0 -ban ezt az állítást meg tagadjuk, és vélgül a H_1 -nek szurkolunk, mert abban lakik az eredeti állításunk:

- $H_0 : \mu_{NemHisz} - \mu_{Hisz} \leq 0$
- $H_1 : \mu_{NemHisz} - \mu_{Hisz} > 0$
- Szurkolunk: H_1

A hipotézisvizsgálatok 2. pontja alapján most kezdjük el megvizsgálni a megfigyelt mintánk adatait. Esetünkben az átlagos oktatási évek eltérését a két csoport (*Hisz* és *NemHisz* a titkos társaságok uralmában a világpolitika felett) között könnyen ki tudjuk számolni egy data frame `groupby` metódussal. Közben az elvi eltérés (δ_0) 0-ás értékét is elrakom egy külön objektumba.

```
elvi_eltérés_átlagban = 0

# két mintaátlag
ess.groupby('SecretGroupInfluenceWorldPol').Education_Years.mean()

## SecretGroupInfluenceWorldPol
## No      12.289596
## Yes     11.966790
## Name: Education_Years, dtype: float64
```

Láthatjuk, hogy a mintában a *NemHisz* csoport átlagos oktatásban töltött éveinek száma nemileg magasabb, mint a *Hisz* csoport esetén ez az átlag. De **nem vehetjük automatikusan igaznak H_1 -et, hiszen ez a jelenség lehet csak a mintavételi hiba műve!** Épp azért kell p-értéket számolni, hogy megtudjuk, mekkora valószínűséggel lehet ez a jelenség (hogy a *NemHisz* csoport esetén magasabb az átlag) **csak a mintavételi hiba műve**.

Szóval, számoljuk csak ki ezt a próbafüggvényt! :)

A próbafüggvényünk formulája alább látható, és azt is megfigyelhetjük az következő képletből, hogy a nevezetes eloszlásunk a p-érték számításához standard normális ($N(0, 1)$):

$$\frac{(\bar{y}_1 - \bar{y}_2) - \delta_0}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \sim N(0, 1)$$

Láthatjuk, hogy a próbafüggvény képlete nem egy bonyolult konstrukció. Fogjuk a két mintaátlag különbségét $(\bar{y}_1 - \bar{y}_2)$ és megnézzük, hogy ez mennyire tér el az elvi különbségtől (δ_0) és ezt arányítjuk a kétféle átlaghöz tartozó

standard hiba négyzetek összegéhez a gyök alatt. Viszont, van erre az úgynevezett **kétmintás z-próbára beépített függvény** a `statsmodels` csomagban, ami a próbafüggvényt és a p-értéket kiszámolja, így **nem kell ezzel manuálisan szórakoznunk** majd. :) Ugye ismét azért lesz *z-próba* a hipotézisvizsgálatunk neve, mert a pórbafüggvényből standard normális, azaz *z* eloszláson keresztül lesz p-értékünk.

Viszont, azt **mindenképpen látnunk kell, hogy mivel a p-érték számításhoz vett eloszlás standard normális ($N(0, 1)$), így minden mintának (*Hisz* és *Nem Hisz* is) nagynak ($n > 30$) kell lennie!** Ezt a `SecretGroupInfluenceWorldPol` oszlop `value_counts()` metódusával tudjuk talán a legkönnyebben ellenőrizni.

```
ess.SecretGroupInfluenceWorldPol.value_counts()
```

```
## SecretGroupInfluenceWorldPol
## No      1301
## Yes     548
## Name: count, dtype: int64
```

Mivel $1301 > 30$ és $548 > 30$, így a **nagy minta előfeltételünk teljesült**.

Lássuk hát a **p-érték számolását** a `ztest` beépített függvényvel a `statsmodels` csomagból. A függvényben a **következő paramétereket** kell megadni:

- **x1** és **x2**: A két minta elemeit tartalmazó tömbök *hiányzó értékek nélkül*.
 - Tehát, a biztonság kedvéért, miután leszűrjük az `Education_Years` oszlop értékeit a *Hisz* és *Nem Hisz* csoportokra a `SecretGroupInfluenceWorldPol` oszlop alapján, még a biztonság kedvéért megkínáljuk a dolgot egy `dropna()` metódussal is.
 - Arra kell figyelni, hogy a függvényben a kivonás iránya **$x1 - x2$** . Tehát, a két csoport átadásának sorrendje következetes kell legyen a H_0 és H_1 -ben alkalmazott kivonások irányával. Nekünk ez most azt jelenti, hogy elsőnek a *Nem Hisz* csoport `Education_Years` értékeit adjuk át (az **x1** paraméteren), mert a hipotéziseinkben $\mu_{Nem Hisz} - \mu_{Hisz}$ kivonás szerepel.
- **value**: Az átlagok közti elvi eltérés értéke a hipotéziseinkben. Ez most nekünk $\delta_0 = 0$.
- **alternative**: Az alteranítv hipotézisben lévő relációs jelet adjuk meg szövegesen.
 - lehetséges értékek: ‘two-sided’, ‘smaller’, ‘larger’, rendre a \neq , $<$, $>$ relációs jeleket jelölik H_1 -ben

Ezzel a működés a következőképpen néz ki.

```
# függvény importja
from statsmodels.stats.weightstats import ztest

ztest(x1= ess.Education_Years[ess.SecretGroupInfluenceWorldPol=='No'].dropna(),
       x2= ess.Education_Years[ess.SecretGroupInfluenceWorldPol=='Yes'].dropna(),
       value=elvi_eltérés_átlagban,
       alternative='larger')

## (1.8415694788119368, 0.03276907446405458)
```

Az eredményül kapott lista első eleme a próbafüggvény, és a második elem a p-érték. Tehát, a **p-értékünk most 3.3% lett**. Tehát, a H_0 elutasításával 3.3% a hibavalószínűségem. Ha a megengedett hibavalószínűségem (α szignifikancia-szintem) 1% akkor így nem tudom elutasítani a H_0 -t, mert többet hibáznék vele, mint a megengedett, de ha a szignifikancia-szintem $\alpha = 5\%$, akkor már H_0 elutasítható, hiszen kevesebb hibával jár, mint a megengedett. Azaz, itt most a p-értékkel bekerülttem a szokásos szignifikancia szintek tartományába (1% és 10% közé), így a felelős „mondás” most az, hogy nem döntök és inkább kérek egy NAGYOBBA mintát, mert a döntés túl érzékeny lenne a konkrét szignifikancia-szint megválasztására. Végső soron tehát **NEM tudom egyértelműen kijelenteni, hogy a megfigyelt mintában mért eltérés az átlagos iskolaévekben a Hisz és NemHisz csoportok között szignifikáns volna** (azaz, hogy az eltérés nem csak a mintavételi hiba műve). A kérdés eldöntésére még nagyobb mintát kellene vizsgálni.

10.4. Az arányokra vonatkozó kétmintás *z-próba*

Az ESS adatbázis SecretGroupInfluenceWorldPol és TrustInParlament című oszlopait felhasználva ellenőrizhetünk egy olyan állítást a teljes magyar népességre (azaz sokaságra), ami szerint „**több, mint 2 %-ponttal alacsonyabb a parlamentben bízók aránya azok körében, akik hisznek a titkos társaságokban**”.

A fentri bekezdésben félkövérrel szedett állítás elég agymegsüllyesztő lehet első olvasatra, de talán az elől látszik, hogy valamiknek a sokasági arányairól, azaz P -ről szól az állítás. Ha ilyen arányokra vonatkozó állítást látunk, akkor két dolgot érdemes kihámozni a szövegből:

1. **Mi a vizsgált arányunk?** → Ez most nekünk a **parlamentben bízók aránya**.
2. **Mi a 2 minta?** → Ez most nekünk az, hogy valaki **hisz / nem hisz a titkos társaságok uralmában** a világpolitika felett.

Ezek alapján az **állítás matematikailag megadva** a parlamentben bízók arányára vonatkozóan:

$$P_{Hisz} < P_{NemHisz} - 0.02$$

Ugye itt **az volt az elv, amit a 2. fejezetben is láttunk**. Ha simán az állítás szövegének „*alacsonyabb*” kifejezésére koncentrálunk, akkor az egyenlőtlenség $P_{Hisz} < P_{NemHisz}$ lenne. De a „*több, mint 2 %-ponttal*” rész miatt azt mondjuk, hogy ha a $P_{NemHisz}$ -ból levonok még 2 %-pontot (0.02-t), még akkor is alacsonyabb (azaz kisebb) marad a P_{Hisz} . És így lett a végső egyenlőtlenségünk, ami leírja matematikailag az alapállításunk $P_{Hisz} < P_{NemHisz} - 0.02$.

Ezt a kiinduló egeynlőtlenséget, ha **átrendezünk az arányok különbségére**, akkor a $P_{Hisz} - P_{NemHisz} < -0.02$ egyenlőtlenséget kapjuk.

Ezt az állítást a szokásos két alapelt mellett már gyorsan tudom H_0 -ra és H_1 -re szétszedni. Mivel az állítás *nem* engedi az egyenlőséget most, így az állításunkból egy H_1 lesz, és a H_0 -ban ezt az állítást meg tagadjuk, és végül a H_1 -nek szurkolunk, mert abban lakkik az eredeti állításunk:

- $H_0 : P_{Hisz} - P_{NemHisz} \geq -0.02$
- $H_1 : P_{Hisz} - P_{NemHisz} < -0.02$
- Szurkolunk: H_1

Jöhet is a hipotézisvizsgálatok 2. pontja, ami szerint elkezdjük megvizsgálni a megfigyelt mintánk adatait. **Számoljuk minden mintában** ($1 = Hisz$ és $2 = NemHisz$) a parlamentben bízók arányát (p_1 és p_2)! Ehhez minden mintában meg kell nézni a teljes elemszámokat (n_1 és n_2), és az arány szempontjából kedvező esetek (k_1, k_2) számát is.

```

k_1 = np.sum((ess.TrustInParlament=='Yes') &
              (ess.SecretGroupInfluenceWorldPol=='Yes'))
n_1 = np.sum(ess.SecretGroupInfluenceWorldPol=='Yes')

k_2 = np.sum((ess.TrustInParlament=='Yes') &
              (ess.SecretGroupInfluenceWorldPol=='No'))
n_2 = np.sum(ess.SecretGroupInfluenceWorldPol=='No')

# mintaarányok

k_1/n_1 # Hivők mintája

```

```
## 0.11131386861313869
```

k_2/n_2 # Nem hívők mintája

```
## 0.14681014604150652
```

Láthatjuk, hogy a mintában a *Hisz* csoportban a parlamentben bízók aránya tényleg több, mint 2 %-ponttal alacsonyabb, mint a *NemHisz* csoport esetén ez az arány ($11.1 - 14.68 = -3.58$). De **nem vehetjük automatikusan igaznak H_1 -et, hiszen ez a jelenség lehet csak a mintavételi hiba műve!** Épp azért kell p-értéket számolni, hogy megtudjuk, mekkora valószínűsséggel lehet ez a jelenség (hogy a *Hisz* csoport esetén ennyivel alacsonyabb a parlamentben bízók aránya) **csak a mintavételi hiba műve.**

A próbafüggvényünk formulája alább látható, és azt is megfigyelhetjük az következő képletből, hogy a nevezetes eloszlásunk a p-érték számításához standard normális ($N(0, 1)$):

$$\frac{(p_1 - p_2) - \epsilon_0}{\sqrt{\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}}} \sim N(0, 1)$$

Láthatjuk, hogy a próbafüggvény képlete itt sem egy bonyolult konstrukció. Fogjuk a két mintaarány különbségét ($p_1 - p_2$) és megnézzük, hogy ez mennyire tér el az elvi különbségtől (ϵ_0) és ezt arányítjuk a kétféle átlaghoz tartozó standard hiba négyzetek összegéhez a gyök alatt. Viszont, van erre az **arányos kétmintás z-próbára is beépített fiuggvény a statsmodels csomagban**, ami a próbafüggvényt és a p-értéket kiszámolja, így **nem kell ezzel manuálisan szórakoznunk** majd. :) Ugye ismét azért lesz *z-próba* a hipotézisvizsgálatunk neve, mert a pórbafüggvényből standard normális, azaz *z* eloszláson keresztül lesz p-értékünk.

Viszont, azt **minden képpen látnunk kell, hogy mivel a p-érték számításhoz vett eloszlás standard normális ($N(0, 1)$)**, így minden mintának nagynak kell lennie! Ez itt kétmintás arányra vonatkozó hipotézisvizsgálat esetén azt jelenti, hogy minden mintában legyen legalább 10 db arány szempontjából kedvező és kedvezőtlen eset is.

Ezt a korábban kiszámolt k_1 , k_2 és n_1 , n_2 értékekből már könnyen tudjuk ellenőrizni.

Nézzük az arány szempontjából kedvező eseteket.

k_1 > 10

```
## True
```

```
k_2 > 10
```

```
## True
```

Mindkét mintára igaz a feltétel, rendben vagyunk.

S most az arány szempontjából kedvezőtlen esetek jönnek. Nyilván itt az az alapelvek, hogy ami a mintában *nem* arány szempontjából kedvező eset, az minden kedvezőtlennek minősül.

```
n_1 - k_1 > 10
```

```
## True
```

```
n_2 - k_2 > 10
```

```
## True
```

Itt is minden mintára igaz a feltétel, rendben vagyunk. Juhé! :)

Lássuk hát a **p-érték számolását** a `test_proportions_2indep` beépített függvénytel a `statsmodels` csomagból. A **függvényben a következő paramétereket** kell megadni:

- **count1, count2 + nobs1, nobs2:** A két minta arány szempontjából kedvező eseteinek száma (`count1, count2`) és a két minta teljes elemszámai (`nobs1, nobs2`).
 - Arra kell figyelni, hogy a függvényben a kivonás iránya 1 – 2. Tehát, a két csoport átadásának sorrendje következetes kell legyen a H_0 és H_1 -ben alkalmazott kivonások irányával. Nekünk ez most azt jelenti, hogy elsőnek a *Hisz* csoport esetszámait adjuk át (a `count1` és `nobs1` paramétereken), mert a hipotéziseinkben $P_{Hisz} - P_{NemHisz}$ kivonás szerepel.
- **value:** Az arányok közti elvi eltérés értéke a hipotéziseinkben. Ez most nekünk $\epsilon_0 = -0.02$.
- **alternative:** Az alteranítv hipotézisben lévő relációs jelet adjuk meg szövegesen.
 - lehetséges értékek: ‘two-sided’, ‘smaller’, ‘larger’, rendre a ≠, <, > relációs jeleket jelölik H_1 -ben

Ezzel a működés a következőképpen néz ki.

```
# függvény importja
from statsmodels.stats.proportion import test_proportions_2indep

test_proportions_2indep(count1 = k_1, nobs1=n_1,
                        count2=k_2, nobs2=n_2,
                        value = -0.02, alternative='smaller')

## <class 'statsmodels.stats.base.HolderTuple'>
## statistic = -0.8767111411557769
## pvalue = 0.19032177690414032
## compare = 'diff'
## method = 'agresti-caffo'
## diff = -0.035496277428367834
## ratio = 0.7582164558413269
## odds_ratio = 0.7279314533902407
## variance = 0.0002782775559788205
## alternative = 'smaller'
## value = -0.02
## tuple = (-0.8767111411557769, 0.19032177690414032)
```

Szépen látszik, hogy a próbafüggvény (az outoutban **statistics**) -0.8767 , de ami fontosabb, hogy a p-értékünk 19%-ra jön ki. Tehát a H_0 -t elutasítani 19%-os valószínűsséggel eredményez hibát. Ez magasabb, mint a legnagyobb megengedett szokásos szignifikancia-szint, az $\alpha = 10\%$, így a H_0 -t nem utasítom el. Mivel nagyobb hibával járna a H_0 elutasítása, mint a mengedett hiba. Tehát, végső soron a két arány közti különbség a népességen kisebbnek vehető, mint 2%-pont. Azaz, a mintaarányok között tapasztalt, 2 %-pontnál nagyobb eltérés NEM szignifikáns a megfigyelt mintánk alapján!

11. fejezet

Nemparaméteres próbák

11.1. Nemparaméteres próbák elve

Továbbra is hipotézisvizsgálatokat végünk.

EDDIG az úgynevezett **paraméteres próbák** esetét vizsgáltunk. Ekkor statisztikai mutatószámokra, azaz statisztikai paraméterek (pl. átlagra, szórásra, aránya, stb.) lehetséges sokasági értékeire fogalmaztunk meg állításokat, *hipotéziseket*.

MOST az úgynevezett **nemparaméteres próbák** esetét kezdjük vizsgálni. Ekkor **ismérvek** sokasági eloszlására vonatkozó állításokat, **hipotéziseket teszünk**. Ugyebár egy ismérv eloszlása gyakorlatilag azt jelenti, hogy megadjuk, hogy a lehetséges ismérvértékek milyen arányban fordulnak elő. Szóval valójában nagyon **sok arány értékéről egyszerre** mondunk itt valamit a hipotézisben megfogalmazott alapállításainkban.

De a **négy alaplépés változatlan** az eddigiekhez képest!

1. H_0 és H_1 felírása
2. Próbafüggvény számítása a megfigyelt mintából
3. A p-érték számítása próbafüggvény és egy nevezetes eloszlás alapján
4. Döntés p-érték alapján $\rightarrow H_0$ vagy H_1 vehető-e igaznak a sokaságban?

A nemparaméteres próbák témakörét a StackOverflowHungary2020.xlsx adattáblán járjuk majd végig, ami a Stack Overflow programozói közösségi oldal 2020-as felmérése a világ amatőr és profi programozóról 60 változó szerint. A teljes adatbázis (és a korábbi+újabb évek felmérései) erről a linkről elérhető. A mi Moodle-n található Excel fájlunkban csak a 2020-as felmérés 210 magyar kitöltőjének válaszai szerepelnek az alábbi 9 változó szerint :

- **Age:** A válaszadó életkora (év)
- **Age1stCode:** A válaszadó életkora első sor programkódjának megírásakor (év)
- **YearsCodePro:** Programozási tapasztalat a tanulmányokat nem beleszámítva (év)
- **MonthlyHuf:** Havi bruttó fizetés Forintban
- **Gender:** Válaszadó neme
- **EdLevel:** Legmagasabb befejezett iskolai végzettség
- **Employment:** Foglalkoztatási státusz (teljes munkaidő; részmunkaidő; egyéni vállalkozó)
- **JobSat:** Elégedettség a jelenlegi munkahelyen
- **OpSys:** Használt operációs rendszer (Windows; Linux; MacOS)

Töltsök be az adatbázist az Excelből egy **pandas** data frame-be és nézzük meg az **info** metódussal megvan-e minden fenteb felsorolt ismérvünk!

```
# Elemzéshez és ábrázoláshoz szükséges csomagok betöltése
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats

# StackOverflow 2020-as kérdőív magyar kitöltő adatainak beolvasása
sfH = pd.read_excel("StackOverflowHungary2020.xlsx")
sfH.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 210 entries, 0 to 209
## Data columns (total 9 columns):
## #   Column      Non-Null Count  Dtype  
## --- 
## #   0   Age         210 non-null    int64  
## #   1   Age1stCode  210 non-null    int64  
## #   2   YearsCodePro 210 non-null    float64 
## #   3   MonthlyHuf  210 non-null    float64 
## #   4   Gender       210 non-null    object  
## #   5   EdLevel      210 non-null    object  
## #   6   Employment   210 non-null    object  
## #   7   JobSat       210 non-null    object  
## #   8   OpSys        210 non-null    object  
## #   dtypes: float64(2), int64(2), object(5)
## #   memory usage: 14.9+ KB
```

Olybá tűnik, mind a 210 megfigyelésünk és a vizsgált 9 változónk. Yeah! :)

11.2. Illeszkedésvizsgálatok

A nemparaméteres próbák egyik nagy alesete az illeszkedésvizsgálatok esete. Ekkor minden azt vizsgáljuk, hogy a **megfigyelt mintaelémek eloszlása illeszkedik-e valami általunk megadott elméleti eloszláshoz** (pl. egyenletes eloszlás vagy normális eloszlás, ilyesmik).

11.2.1. Reprezentativitás vizsgálat

Reprezentativitás egy ismérv szerint: a minta eloszlása egy konkrét ismérv szerint kb. ugyan az, mint az ismérv eloszlása a teljes adatsokaságban.

A KSH 2020-as adatai szerint a magyar infokommunikációs-szektörben tevékenykedők

- 85%-a teljes állásban foglalkoztatott,
- 4% részmunkaidős
- 11% egyéni vállalkozó.

Ezen munkakör típus arányok (azaz munkakör típus eloszlás) mellett **reprezentatív-e a StackOverflow kérdőív magyar mintája munkakör típusra?**

Ekkor:

- H_0 : A minta **reprezentatív**
- H_1 : A minta **NEM reprezentatív**

Próbafüggvényünk és p-értékhez az eloszlásunk a következő

$$\sum_{j=1}^k \frac{(f_j - f_j^*)^2}{f_j^*} \sim \chi^2(k-1)$$

A képletként szereplő betűk jelentése:

- k : A vizsgált ismérv lehetséges rtékeinek (kategóriáinak) száma
- f_j : tény gyakoriságok
- f_j^* : reprezentativitás esetén fennálló elvi gyakoriságok a mintában

A **p-értéket** a $\chi^2(k-1)$ eloszlásból **mindig jobboldali** módon számítjuk. Részletek a jobboldali p-érték számításról a 9.2. fejezetben.

Számítsuk ki a szükséges f_j gyakoriságokat.

```
tény_gyak = sfH.Employment.value_counts()
tény_gyak
```

```
## Employment
## Employed full-time                                167
## Independent contractor, freelancer, or self-employed    33
## Employed part-time                                    10
## Name: count, dtype: int64
```

Jöjjönek az elvi f_j^* gyakoriságok! Mi lenne ha az $n = 210$ elemű minta teljesen reprezentatív lenne?

```
elvi_gyak = 210 * np.array([0.85, 0.11, 0.04])
elvi_gyak
```

```
## array([178.5, 23.1, 8.4])
```

Az eltérés elvi és tény gyakoriságok között betudható-e a mintavételi hibának?
→ Hipotézisvizsga :)

Próbafüggvény és p-érték beépített `scipy` függvényivel. Most a szabadságfok $df = k - 1 = 3 - 1 = 2$. Ez a szabadságfok értelmezés (tehát a $szf = k - 1$) a függvény alapértelmezése, így ezt nem kell külön paraméterben beállítanunk.

```
stats.chisquare(tény_gyak, elvi_gyak)
```

```
## Power_divergenceResult(statistic=5.288515406162464, pvalue=0.07105808036191753)
```

A p-értékünk 7.1% Ez beza benne van a szokásos szignifikancia-szintek tartományában (1%-10%). Nagyobb minta alapján kéne döntenünk. De közelebb vagyunk a 10%-oz, mint az 1%-hez → inkább tűnik a minta reprezentatívnak, mint nem. :)

Előfeltételünk, hogy minden elvi gyakoriság legalább 5, azaz $f_j^* \geq 5$. Ellenőrizzük le gyorsan!

```
elvi_gyak >= 5
```

```
## array([ True,  True,  True])
```

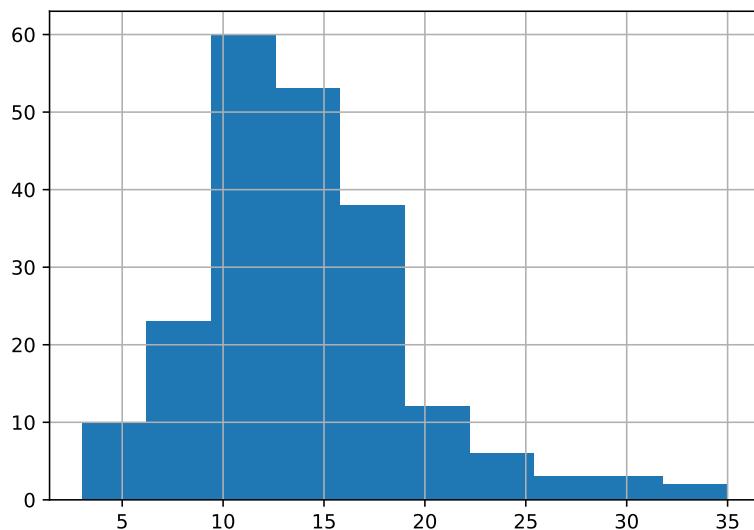
Mind a három esetben van legalább 5 megfigyelés az elvi esetben, jók vagyunk!
Wuhú! :)

11.2.2. Normalitás vizsgálat

Állításunk, hogy a magyar programozók sokaságának az első kód megírásakor a koreloszlása normális eloszlású.

Szemmelverés = Hisztogram

```
sfH.Age1stCode.hist()
```



Nagyjából normális eloszlású a hisztogram, de kicsit jobbra nyúlik.

Kérdés enyhe jobbra nyúlás a mintavételi hiba műve-e? → Hipotézisvizsgálat!
:)

- H_0 : **Normális** az eloszlás
- H_1 : **NEM normális** az eloszlás

A próbafüggvény és p-érték számoláshoz trükközünk! Kiszámoljuk az Age1stCode ismérvre legjobban illeszkedő normális eloszlás kvintiliseit, azaz ötödőlő pontjait! Pl. K_2 az az érték, aminél az adatok 40%-a (2/5) kisebb, 60%-a (3/5) nagyobb. Az Age1stCode ismérvre legjobban illeszkedő normális eloszlás: az a normális eloszlás, aminek átlaga és szórása ugyan az, mint az Age1stCode ismérvén.

A megfelelő átlag és szórás számolása. Szórás korrigált, hiszen mintában vagyunk, nem szeretnénk torzítást! :)

```
átlag = np.mean(sfH.Age1stCode)
s = np.std(sfH.Age1stCode, ddof=1)
```

Normális eloszlás ötödölő pontjai a `scipy` csomag `stats`. függvényévek. Ötödölő pontok (kvintilisek) listaként való átadása. Technikai okok miatt kell a 0 és 1 = 100% sztópont is a 0.2 egységekre bontó pontok = ötödölő pontok = kvintilisek mellett.

```
norm_kvint=stats.norm.ppf([0,0.2, 0.4, 0.6, 0.8,1], loc=átlag, scale=s)
norm_kvint
```

```
## array([-inf,  9.60607718, 12.62407404, 15.22354501, 18.24154186,
##        inf])
```

Tapasztalati gyakorisági tábla, f_j -k megadása a normális eloszlás kvintiliseire.

```
gyak_tábla = np.histogram(sfH.Age1stCode, bins=norm_kvint)
gyak_tábla
```

```
## (array([33, 60, 53, 38, 26], dtype=int64), array([-inf,  9.60607718, 12.62407404,
```

Elvi gyakoriság, ha H_0 ,azaz a normális eloszlás igaz lenne $f_j^* = \frac{n}{5} = \frac{210}{5} = 42$. Ez a `stats.chisquare` alapbeállítása f_j^* -re.

Számoljuk ki a próbafüggvényt és p-értéket ezzel az alapfeltételezéssel f_j^* -re. Van minden: f_j és f_j^* is. De most az eloszlás p-értékhez $\chi^2(k - 1 - b)$, ahol b a becsült paraméterek száma. Ez most a legjobban illeszkedő normális eloszlás átlaga és szórása volt, így $b = 2$. Ezzel a $b = 2$ értékkel felül kell írni a `stats.chisquare` függvény alapbeállítását a szabadságfok (ddof) paraméterre. Ezen a ddof paraméteren csak a $b = 2$ -t kell átadni, így tudni fogja a függvény, hogy a szabadságfokot $k - 1 - b$ módon kell kiszámolnia.

```
stats.chisquare(gyak_tábla[0], ddof=2)
```

```
## Power_divergenceResult(statistic=19.0, pvalue=7.485182988770057e-05)
```

A p-értékünk 0.0075%, ami kisebb még a legkisebb szokásos szignifikanciaszintnél, az 1%-nál is. Azaz, a H_0 stabilan elvethető, így az eloszlás **nem tekinthető normálisnak**

Tehát, a hisztogramon észrevehető enyhe jobbra nyúlás a normális eloszláshoz képest NEM mintavételi hiba műve, hanem egy szignifikáns (jelentős) eltérés, ami megmarad a mintán kívüli világban is!

11.3. Függetlenségvizsgálatok (homogenitásvizsgálatok)

Vizsgáljuk meg azt az állítást, miszerint, az összes magyar programozó sokaságában az **egyes operációs rendszereket használó fejlesztők ugyan olyan munkahelyi elégedettségi arányokkal bírnak**. Másképpen fogalmazva a munkahelyi elégedettség és az operációs rendszer között **összefüggés l l fenn**.

Más szóval, függetlenségvizsgálat esetén azt vizsgáljuk, hogy a sokaságban, azaz a **mintán kívüli világban**, két **nominális** (szöveges) **ismérv között összefüggés áll fenn**. Ezt pedig a következő null- és alternatív hipotézis párossal írjuk le.

- **Két nominális** ismérv: JobSat és OpSys
- H_0 : A két ismérv **független**
- H_1 : A két ismérv **összefügg**

Úgy járunk el, mint Statisztika I-en két nominális ismérv kapcsoltának vizsgálatánál: készítünk egy kereszttávlát, ami megadja a két ismérv *együttet* gyakoriságait.

```
kereszt = pd.crosstab(sfH.JobSat, sfH.OpSys)
kereszt
```

	Linux-based	MacOS	Windows
## OpSys			
## JobSat			
## Neither satisfied nor dissatisfied	3	4	16
## Slightly dissatisfied	10	6	23
## Slightly satisfied	18	7	44
## Very dissatisfied	2	2	11
## Very satisfied	25	8	31

Tehát, pl. 31 főnyi munkájával nagyon elégedett Windows felhasználó van a megfigyelt mintánkban.

Sajnos, a kereszttáblában van olyan elem (azaz van olyan i sor és j oszlop), ahol a gyakoriság kisebb, mint 5, azaz $\exists f_{ij} < 5$. Láasd pl. A semleges érzelmi Linuxosok esetét. Ekkor **valószínűleg NEM fog teljesülni a nagy minta előfeltétel**, mert nem lesz minden kereszttábla gyakoriság legalább 5.

Emiatt a problémát úgy hidaljuk át, hogy a *Very satisfied* és *Slightly satisfied* kategóriákból készítünk egy *Satisfied* kategóriát, a többiekből pedig egy *Not satisfied* kategóriát. Ezt a *numpy* csomag *where* függvényével tudjuk intézni, mint az 7.2. fejezetben az arányok konfidenzia-intervallum számolásához a $0 - 1$ értékű új oszlop létrehozását.

```

sfH['JS_v2'] = np.where(sfH.JobSat.isin(['Very satisfied',
                                         'Slightly satisfied']),
                        'Satisfied', 'Not satisf.')
kereszt = pd.crosstab(sfH.JS_v2, sfH.OpSys)
kereszt

## OpSys      Linux-based  MacOS  Windows
## JS_v2
## Not satisf.      15      12      50
## Satisfied        43      15      75

```

Remek, már csak két érték van a munkahelyi elégdettséggel kapcsoaltos új oszlopból (JS_v2). Innentől kezdve **homogenitásvizsgálatot** végzünk. Ugyanis, ha az egyik nominális ismérvnek csak 2 lehetséges értéke van, akkor *függetlenségvizsgálat* = *homogenvizsgálat*. Mivel ekkor két nominális csoporton belül vizsgáljuk, hogy egy másik nominális arányai azonosnak lehetők-e. Azaz vizsgáljuk, hogy a két nominális ismérv független-e.

Nézzük meg az elégdettségi arányok oprendszeren belül!

```

pd.crosstab(sfH.JS_v2, sfH.OpSys, normalize='columns')

## OpSys      Linux-based  MacOS  Windows
## JS_v2
## Not satisf.  0.258621  0.444444    0.4
## Satisfied    0.741379  0.555556    0.6

```

Ezen arányok eltérése oprendszerök között csak a mintavételi hiba műve-e? → Hiptézisvizsgálat! :)

Próbafüggvényünk a Stat. I-en megismert χ^2 mutató, és igaz H_0 esetén a próbafüggvény eloszlása sok-sok mintavételből $\chi^2((r-1)(c-1))$, ahol r a kereszttábla sorainak (*rows*), míg c a kereszttábla oszlopainak (*columns*) a darabszáma:

$$\sum_{i=1}^r \sum_{j=1}^c \frac{(f_{ij} - f_{ij}^*)^2}{f_{ij}^*} \sim \chi^2((r-1)(c-1))$$

A p-érték itt is **jobboldali módon** számolható, mint az illeszkedésvizsgálatok esetében.

Próbafüggvény és p-érték számolás a `stats.chi2_contingency` beépített függvénytellyel. Bemenet csak a kereszttábla, a szabadságfokot is ebből ki tudja számolni a függvény.

11.3. FÜGGETLENSÉGVIZSGÁLATOK (HOMOGENITÁSVIZSGÁLATOK)291

```
stats.chi2_contingency(kereszt)
```

```
## Chi2ContingencyResult(statistic=4.217125886817357, pvalue=0.12141231749833191, dof=2, expected
##      [36.73333333, 17.1          , 79.16666667]))
```

A p-értékünk itt 12%, ami nagyobb még a legnagyobbszokásos szignifikancia-szintnél a 10%-nál is. A mintán kívüli világban, azaz a **sokaságban a két ismérv függetlennek tekintíthő**. Az a tény, hogy a megfigyelt mintában az elégedettségi arányok eltérnek oprendszerek között csak a mintavételi hiba műve, nem szignifikáns!

A függvény eredményéül kapott listában a 4. elem az a kereszttábla, ami igaz H_0 , azaz a két ismérv függetlensége esetén lenne.

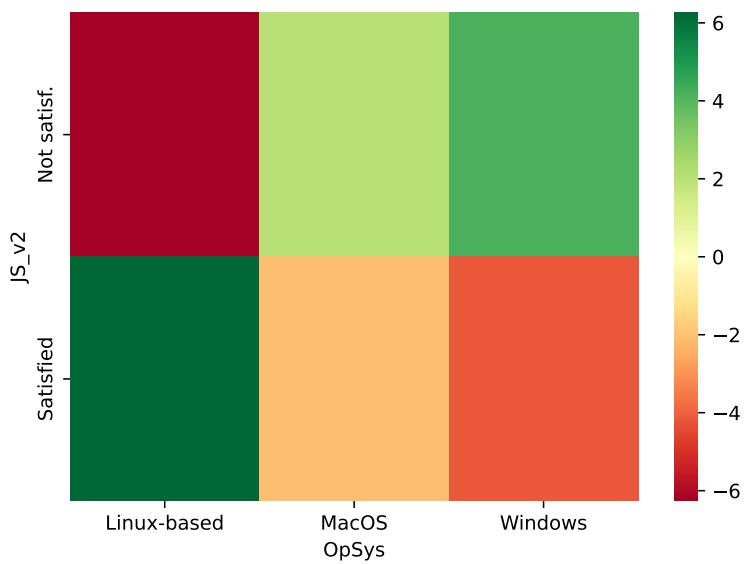
Ha H_0 elutasítható lenne, akkor érdekes lenne, hogy a megfigyelt mintában a kereszttábla (az f_{ij} -k) hol tér el a legjobban a függetlenség esetén várt gyakoriságuktól, az f_{ij}^* -ktől.

Ezeket az $f_{ij} - f_{ij}^*$ eltérések szépen lehetne vizualizálni hőterképen a **seaborn** csomag segítségével.

```
eltérés = kereszt - stats.chi2_contingency(kereszt)[3]
eltérés
```

```
## OpSys      Linux-based  MacOS   Windows
## JS_v2
## Not satisf. -6.266667    2.1  4.166667
## Satisfied   6.266667   -2.1 -4.166667
```

```
import seaborn as sns # seaborn csomag importja
sns.heatmap(eltérés, cmap='RdYlGn') # eltérés vizualizáció hőterképen
```



Látható, hogy kb. 6-tal több elégedett Linuxos van a mintában, mint függetlenség (H_0) esetén lennie kéne, és kb. 4-gyel kevesebb elégedet Windowsos van a mintában, mint függetlenség esetén kéne.

12. fejezet

Kétváltozós lineáris regresszió

12.1. Budapesti lakások vizsgálata

A BP_Lakas.csv fájl egy olyan adattábla, ami 1406 budapesti lakásról 10 változó (oszlop) adatát tárolja:

- KinArMFt: lakás ára millió Ft-ban (MFt)
- Terulet: lakás területe négyzetméterben
- Terasz: teraszok száma a lakásban
- Szoba: szobák száma a lakásban
- Felszoba: félszobák száma a lakásban
- Furdoszoba: fürdőszobák száma a lakásban
- Emelet: emeletek száma a lakásban
- DeliTaj: lakás déli fekvésű-e? (1 = igen; 0 = nem)
- Buda: lakás déli fekvésű-e? (1 = igen; 0 = nem)
- Kerulet: lakás kerülete (1 - 22)

Olvassuk be az adattáblát egy pandas `data frame`-be egy `readr_csv` függvény segítségével! A fájl olyan értelemben „jól vislekedő” `csv`, hogy az oszlopokat vesszők választják el egymástól, és a tizedes helyet tizedes pont jelöli, így nem kell a függvényben semmit sem külön paraméterezn. A beolvasás után egy `info` metódussal nézzük meg, hogy rendben van-e minden a betöltött `data frame`-el.

```
# Elemzéshez és ábrázoláshoz szükséges csomagok betöltése
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```

import scipy.stats as stats

# Budapesti lakások adatainak beolvasása
BP_Lakas = pd.read_csv("BP_Lakas.csv")
BP_Lakas.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 1406 entries, 0 to 1405
## Data columns (total 10 columns):
## #   Column      Non-Null Count  Dtype  
## #   --   --          --           --    
## #  0   KinArMFT    1406 non-null   float64
##  1   Terulet     1406 non-null   float64
##  2   Terasz       1406 non-null   float64
##  3   Szoba        1406 non-null   int64  
##  4   Felszoba     1406 non-null   int64  
##  5   Furdoszoba   1406 non-null   int64  
##  6   Emelet        1406 non-null   int64  
##  7   DeliTaj      1406 non-null   int64  
##  8   Buda          1406 non-null   int64  
##  9   Kerulet      1406 non-null   int64  
## dtypes: float64(3), int64(7)
## memory usage: 110.0 KB

```

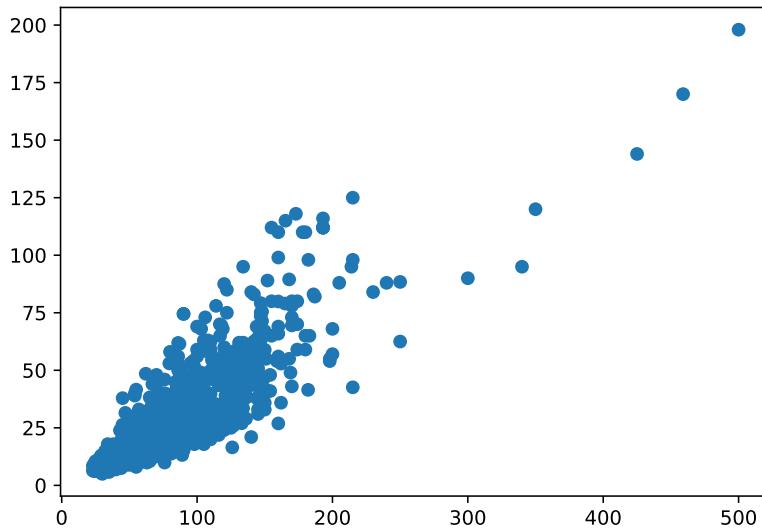
Első ránézésre rendben vagyunk: 10 oszlopunk = változónk van a megfelelő oszlopnevekkel, mindenhol 1406 **non-null** megfigyeléssel.

Vizsgáljuk meg a lakások kínálati árának és a területének a kapcsolatát! Mivel logikusan azt gondolhatjuk, hogy a terület fogja meghatározni az árak alakulását, és nem fordítva, így a terület lesz a magyarázóváltozónk (*x tengely*) és az ár az eredményváltozónk (*y tengely*). A pontdiagramos ábrázoláshoz a **scatter** című **matplotlib** függvényt tudjuk bevetni. A függvényben az első paraméter legyen az az oszlop, ami a diagram x tengelyét adja, míg a második az, amelyik az y tengelyét szolgáltatja az ábrának.

```

plt.scatter(BP_Lakas.Terulet, BP_Lakas.KinArMFT)
plt.show()

```



Láthatjuk, hogy a terület növekedésével nőnek az árak, és ez a kapcsolat szoros is lehet, mert a pontokra gondolatban is elég jó pontossággal rá tudunk illeszteni egy pozitív meredekségű egyenest. Mindezt Stat. I-es ismereteink alapján egy korreláció kiszámításával tudjuk megerősíteni. Itt Pythonban korrelációt a `scipy` csomag `stats` moduljában lakó `pearsonr` függvényel tudunk számolni, aminek a bemenete a két numerikus változó, amik között korrelációt akarunk számolni. Itt a sorrend mindegy, nem számít hogy a magyarázóváltozót vagy az eredményváltozót adom meg a függvénynek első paraméterként.

```
stats.pearsonr(BP_Lakas.Terulet, BP_Lakas.KinArMft)
```

```
## PearsonRResult(statistic=0.8597347987405766, pvalue=0.0)
```

Az eredmény `statistics` része maga a korrelációs együttható, aminek jele r . Tehát, $r = +0.8597$. Ismétlésként meg kell emlékezni arról, hogyan is kell egy korrelációt értelmezni. Először is, egy korreláció mindenkor ± 1 közötti érték, azaz $-1 \leq r \leq +1$. Egyébként, korreláció kapcsán mindenkor kétdolgot kell vizsgálni:

1. Korreláció **előjele**: Ha ez *pozitív*, akkor a két változó közti kapcsolat *egyirányú* (ha nő az egyik érték, akkor várhatóan nő a másik is). Míg ha az előjel *negatív*, akkor a két változó közti kapcsolat *ellenétes irányú* (ha nő az egyik érték, akkor várhatóan csökken a másik is).
- Esetünkben pozitív korrelációról beszélünk, azaz ha nő a lakás területe, akkor várhatóan nő az ára is. Ez teljesen logikus.

2. Korreláció **abszolút értéke**: Ha az abszolút érték, 0.3 alatti ($|r| < 0.3$), akkor a megfigyelt kapcsolat *gyenge* erősséggű. Ha $0.3 - 0.7$ közötti ($0.3 \leq |r| \leq 0.7$), akkor *közepes*, míg 0.7 feletti abszolút érték ($|r| > 0.7$) esetén *erős* kapcsolatról beszélünk a két változó között.
- Esetünkben a korreláció abszolút értéke nagyobb, mint 0.7 ($0.86 > 0.7$), így a lakások területe és ára közötti kapcsolat erősnek minősíthető.

Erre a korreláció értelmezésre utalt az a jelenség a fenti pontdiagramon, hogy a pontokra gondolatban is elég jó pontossággal (korreláció abszolút értékben magas) rá tudunk illeszteni egy pozitív meredekségű (pozitív előjelű korreláció) egyenest.

Ezen a korrelációtól túl kapunk a függvényből egy p-értéket is. Ez egy olyan **nullhipotézishez tartozik, amely azt mondja, hogy ez a korreláció a sokaságban** (a nem megfigyelt lakások körében) lehet akár nulla is. Tehát, a H_0 itt azt mondja, hogy a mintában megfigyelt korreláció a sokaságban tök 0, a megfigyelt kapcsolat a sokaságban (a nem megfigyelt lakások körében) nem létezik. Szóval, itt a következő null- és alternatív hipotézis párosról van szó:

- $H_0 : r = 0$
- $H_1 : r \neq 0$

Ez az ominózus p-érték a fenti H_0 és H_1 pároshoz gyakorlatilag a függvény eredménye szerint 0.0, szóval ez a H_0 minden szokásos szignifikancia-szinten elvethető, a korreláció a sokaságban (a nem megfigyelt lakások körében) nem vehető nullának, a a megfigyelt kapcsolat a sokaságban igenis szignifikánsan létezik.

A korreláció (r) négyzete megadja a determinációs együttható, tehát az $R^2 = r^2$ értékét. Mivel a korreláció egy $[-1, +1]$ közötti szám, így négyzete egy $[0, 1]$ közötti érték lesz, és így *százalékosan* értelmezhető: megadja, hogy az x hány százalékban magyarázza y alakulását/ingadozását:

```
korrelacio = stats.pearsonr(BP_Lakas.Terulet, BP_Lakas.KinArMft)[0] # a két számból az
R_negyzet = korrelacio**2
R_negyzet
## 0.7391439241654997
```

Esetünkben tehát azt mondhatjuk, hogy a **Terulet** 73.914%-ban határozza meg a **KinAr** alakulását. Vagy másképpen: a **Terulet** ismeretében 73.914%-os pontossággal lehet megesülni a **KinAr** értékét. Ez így egy egészen jó előrejelző modell lenne, mivel a 10% alatti R^2 -ű előrejelző modell magyarázóereje *gyenge*, 10%-50% között a magyarázóerőt **közepesnek** vesszük, és 50% felett (tehát ha

a modell az eredményváltozóban lévő információ legalább felét megmagyarázta) mondjuk **erősnek**.

Ennyiből felmerülhet bennünk, hogy csinálunk egy olyan statisztikai modellt, ami a lakásárakat akarja előrejelezni a terület segítségével (hiszen abszolút értékben ezzel erősen korrelál a lakások kínálati ára)! Ez a modell lesz a **kétváltozós lineáris regresszió!** Ez a regresszió vagy más néven *regressziós egyenes*, az $y = \text{KinAr}$ és $x = \text{Terulet}$ pontdiagramon a pontokra legjobban illeszkedő egyenes.

Lássuk hát, hogyan is használható előrejelzésre ez a regressziós egyenes, és azt is nézzük meg, hogy miképpen tudja a `ggplot` berajzolni ezt az egyenest, mint a *pontdiagramra legjobban illeszkedő egyenest!*

12.2. A Kétváltozós Lineáris Regresszió OLS elvű becslése

Az előző fejezetben már bevezettük az alábbi általános jelöléseket, de most álljon itt még egyszer a dolog, emlékeztetőként:

- $y := \text{KinArMft}$ (**eredményváltozó**, amit előre akarunk jelezni)
- $x := \text{Terulet}$ (**magyarázóváltozó**, aminek ismeretében el akarjuk végezni az előrejelzést)

Középiskolában az x, y koordináta rendszerben egy egyenes egyenletét az alábbi általános jelölésekkel írta le az ember.

$$y = mx + b$$

Itt ugye m volt az egyenes *meredeksége*, míg b a konstans tag, vagy másnéven *tengelymetszet*. A b , azaz a tengelymetszet megadja, hogy az egyenes hol metszi el az y tengelyt, míg az m megadja, hogy ha +1 egységet előre lépünk az x tengelyen, akkor az y tengelyen mennyit kell előre haladni, hogy az egyenesen maradjunk. Tehát, a *meredekség* megmutatja, hogy milyen gyorsan emelkedik/csökken az egyenes.

Lineáris regresszióban a fenti egyenes egyenlet az alábbi általános alakot ölti:

$$\hat{y} = \beta_1 x + \beta_0$$

Ebben a felírásban az \hat{y} jelenti az egyenletből **becsült árakat**. Ez a legfontosabb módosítás az egyenletben, hiszen $\hat{y} \neq y!!$ Az y a valós ára a lakásnak, míg az \hat{y} a regressziós egyenes alapján, a *Terulet* ismeretében becsült ár. Csak akkor lehetne $\hat{y} = y$ minden pontra (azaz minden lakásra),

ha $R^2 = 100\%$ lenne, amit a pontdiagramon is láttunk, hogy nem áll fenn, mert **nem illeszkedik minden pont az egyenesre!** Ezen kívül tisztán látszik, hogy a második egyenletben a meredekséget és a tengelymetszetet csak „átbetűztük”: $\beta_1 = m$ és $\beta_0 = b$.

Az egyenletben x ismert minden lakásra, így nekünk csak a β -kat igazából valahogy meghatározni, hogy az egyenlet ténylegesen használható legyen a lakásárak megbecslésére. Ezt csinálja a `ggplot` is, amikor berajzolja a trendvonalat a pontdiagramon.

A β -k meghatározása teljesen logikus módon úgy történik, hogy az árakra (y) adott **becsléseink hibája minimális legyen**. A becslési hibát az ún. SSE mutatóval mérjük: $Sum of Squared Errors(SSE) = \sum_{i=1}^n (y_i - \hat{y}_i)^2$. Az SSE -ben, mint *hibafüggvényben* az egyedenkénti $y_i - \hat{y}_i$ hibatagot két okból emeljük négyzetre:

- Büntetni kell az alá- és fölébecslésekkel is...
- ...DE az abszolút érték függvény nem differenciálható, ami egy szükséges tulajdonság ha egy függvényt minimalizálni szeretnénk

Nézzük is meg, hogyan működik az SSE a gyakorlatban! Első körben adunk valami **kezdeti tippeket a β -akra**, majd kiszámoljuk a \hat{y} -okat mind az 1406 lakásra, tehát meglesz minden \hat{y}_i , $i = 1$ -től $i = n$ -ig:

```
# Kezdetben minden Bétára azt tippeljük, hogy az értéke = 1
Beta0 = 1
Beta1 = 1

# Kiszámoljuk az y kalapokat, azaz a becsült árakat ezekkel a Bétákkal
BP_Lakas['BecsultAr'] = Beta1*BP_Lakas.Terulet + Beta0

# Kiszámolhatjuk a becslések hibáját is minden lakásra
BP_Lakas['Hiba'] = BP_Lakas.KinArMft - BP_Lakas.BecsultAr

# Nézzük meg mit alkottunk
BP_Lakas.loc[:, ["KinArMft", "BecsultAr", "Hiba"]]
```

	KinArMft	BecsultAr	Hiba
## 0	10.7	33.0	-22.3
## 1	10.0	33.0	-23.0
## 2	10.5	33.0	-22.5
## 3	12.0	35.0	-23.0
## 4	13.0	35.0	-22.0
##
## 1401	37.0	81.0	-44.0
## 1402	21.0	84.0	-63.0

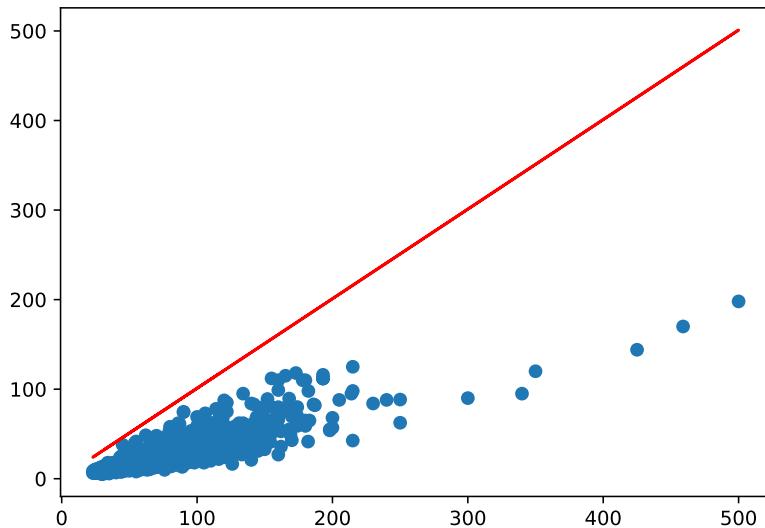
12.2. A KÉTVÁLTOZÓS LINEÁRIS REGRESSZIÓ OLS ELVŰ BECSLÉSE 299

```
## 1403      42.0      93.0 -51.0
## 1404      22.0     101.0 -79.0
## 1405      32.5     117.0 -84.5
##
## [1406 rows x 3 columns]
```

Láthatjuk, hogy a kezdeti tippeink a Bétákra még nem valami jók, 23-24 millió Ft-ot hibázik kb. a becslésünk lakásonként.

Nézzük meg, hogy a jelenlegi, csupa 1 β -k mellett milyen alakot ölt a **BecsultAr** oszlopban lévő \hat{y}_i -k által meghatározott regressziós egyenes! Itt egy olyan trükköt vetek be, hogy a korábban használt **matplotlib** csomag pontdiagrammos **scatter** függvényére egy sima **plot** függvénytel rárakom a regressziós egyenes x és \hat{y} koordinátáit, mint egy vonaldiagram. És ezzel akkor szépen ki is rajzolódik a jelenlegi β_0, β_1 értékekkel adódó regressziós egyenes:

```
plt.scatter(BP_Lakas.Terulet, BP_Lakas.KinArMft)
plt.plot(np.array(BP_Lakas.Terulet),
         np.array(BP_Lakas.BecsultAr), 'red')
plt.show()
```



Láthatjuk, hogy ez a regressziós egyenes még jó bána: a valós pontokhoz képest sokkal magasabban fut, nem jól illeszkedik rájuk.

Számoljuk is ki, hogy összességében mennyi a négyzetes hibánk, tehát az *SSE*! Itt azt használjuk, hogy a **data frame** oszlopokkal, mint **vectorokkal** tudunk az R-ben számolni:

```
# Egyik út
np.sum(BP_Lakas.Hiba**2)
```

```
## 4767764.1886
```

```
# Másik út
np.sum((BP_Lakas.KinArMFt - BP_Lakas.BecsultAr)**2)
```

```
## 4767764.1886
```

Giganagy az SSE, de ezen már az előbbiek alapján meg sem lepődünk. :)

Az előbbi pontdiagram alapján azt láthatjuk, hogy a regressziós egyenesnek az a bája, hogy túl magasról indul (túl magas a tengelymetszet) és túl gyorsan is emelkedik (túl magas a meredekség). Szóval minden két β -t csökkentsük. Szemmegértékre nem tűnik rossznak ha az egyenest $\beta_0 = -0.5$ -ról indítjuk, és az emelkedés ütemét megfelezzük: $\beta_1 = 0.5$. Nézzük is meg mit tudunk ezzel alkotni:

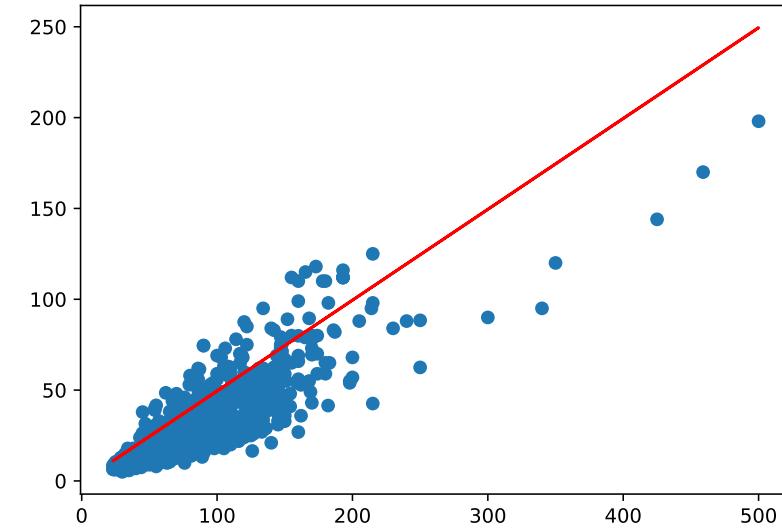
```
# Újra értékkadás a Bétáknak
Beta0 = -0.5
Beta1 = 0.5

# Kiszámoljuk az y kalapot, azaz a becsült árakat ezekkel a Bétákkal
BP_Lakas.BecsultAr = Beta1*BP_Lakas.Terulet + Beta0

# Kiszámolhatjuk a becslések hibáját is minden lakásra
BP_Lakas.Hiba = BP_Lakas.KinArMFt - BP_Lakas.BecsultAr

# Ábrázoljuk az új egyenest a pontdiagramon
plt.scatter(BP_Lakas.Terulet, BP_Lakas.KinArMFt)
plt.plot(np.array(BP_Lakas.Terulet),
         np.array(BP_Lakas.BecsultAr), 'red')
plt.show()
```

12.2. A KÉTVÁLTOZÓS LINEÁRIS REGRESSZIÓ OLS ELVŰ BECSLÉSE



Ez már sokkal jobban néz ki! De vajon csökkent az SSE is?

```
np.sum(BP_Lakas.Hiba**2)
```

```
## 352004.66865
```

Jepp, $352005 < 4767764$, szóval ez objektívan, és nem csak diagramról nézve is egy jobb illeszkedés! :)

Mielőtt továbblélünk bevezetek egy jelölést, ϵ -al fogjuk jelölni a regressziós modell hibatagját. Ezzel, a kódban lévő első képlet formálisan így néz ki: $SSE = \sum_{i=1}^n (\epsilon_i)^2$.

Akkor hát, hagyunk fel a „kézimunkával”, és kerestessük meg a gépállattal azokat a β -kat, amikkel a lehető legkisebb SSE -t kapjuk!

Ehhez először megcsinálunk egy külön R függvényt az SSE -re, ami $SSE(\beta_0, \beta_1)$ alakot ölt. Tehát, megadja az SSE -t a β -k függvényében:

```
# a függvény megadása
def SSE(x):
    return np.sum((BP_Lakas.KinArMft-(x[0]+x[1]*BP_Lakas.Terulet))**2)

# a függvény használata úgy, hogy minden Béta értéknek 1-et mondunk
SSE(np.array([1,1]))
```

```
## 4767764.1886
```

Láss csodát a csupa 1 β -kra ugyan azt az SSE -t kapjuk, mint korábban! :)

Na, ezt az SSE függvényt felhasználva meg tudjuk kerestetni gépállattal hol vannak azok a β -k, amik mellett a legkisebb összesített négyzetes modellhibát kapjuk. A használt `minimize` függvény a `scipy` csomag lakója:

```
from scipy.optimize import minimize # függvény importja scipy-ból
eredmeny = minimize(SSE,np.array([1,1])) # az hiba minimalizálást a minden Béta = 1 pozícióban végez
```

Az újonnan létrehozott, `eredmeny` objektumból, ami `dictionary` típusú, ki tudjuk olvasni, hogy mik lettek a β -k, és meg tudjuk nézni azt is, hogy mi az a legkisebb SSE érték, amit el tudtunk érni:

```
eredmeny.x # a Béták
```

```
## array([-4.31228957,  0.40019634])
```

```
eredmeny.fun # a minimalizált SSE érték
```

```
## 141310.92681266984
```

Ezzel a végső regressziós egyenletünk: $BecsultAr = -4.312 + 0.400 \times Terulet$. Ez így az úgynevezett **OLS** (**Ordinary Least Squares**) feladat megoldása.

Meg is nézhetjük, hogy az ezekkel a β -kal rajzolt egyenes miképpen néz ki a pontdiagramon:

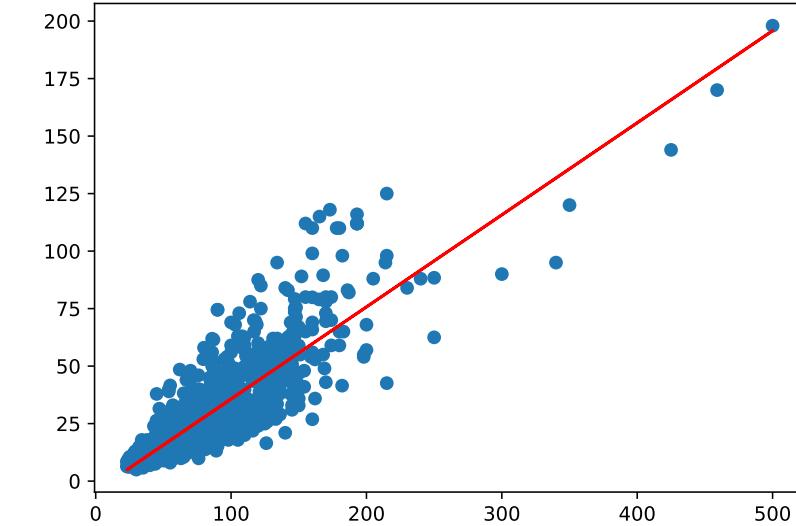
```
# Újra értékkedás a Bétáknak
Beta0 = eredmeny.x[0]
Beta1 = eredmeny.x[1]

# Kiszámoljuk az y kalapokat, azaz a becsült árakat ezekkel a Bétákkal
BP_Lakas.BecsultAr = Beta1*BP_Lakas.Terulet + Beta0

# Kiszámolhatjuk a becslések hibáját is minden lakásra
BP_Lakas.Hiba = BP_Lakas.KinArMft - BP_Lakas.BecsultAr

# Ábrázoljuk az új egyenest a pontdiagramon
plt.scatter(BP_Lakas.Terulet, BP_Lakas.KinArMft)
plt.plot(np.array(BP_Lakas.Terulet),
         np.array(BP_Lakas.BecsultAr), 'red')
plt.show()
```

12.2. A KÉTVÁLTOZÓS LINEÁRIS REGRESSZIÓ OLS ELVŰ BECSLÉSE



Ami nagyon fontos, hogy amiatt, hogy a hibát négyzetesen értelmeztük OLS-ben, ezek a β -k,a mit megkaptunk **egyértelműek!!** Akárhányszor futtam az optimalizálást minden ugyan ezeket az értékeket fogom kapni! Ha a hibatagok összeadásakor abszolút értéket alkalmaztam volna, akkor ez a luxus nem lenne meg! Ott minden optimalizálásnál fennállna az esélye, hogy más β -kat kapok, mint az előző esetben, és a β -król nem is tudnám eldönten, hogy ezekkel az értékekkel kapom-e ténylegesen a legkisebb becslési modellhibát!

A négyzetesen mért hiba esetében ez az egyértelműségi probléma azért nincs meg, mert valójában a gép itt **nem vaktában keresi** a β -kat! Az OLS feladat megoldása (tehát a legkisebb SSE -t adó β -k) kifejezhetők egy fix képlettel.

Valójában a kétváltozós $SSE(\beta_0, \beta_1) = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_i (y_i - \beta_0 - \beta_1 x_i)^2$ függvény minimumhelyének meghatározása történik ezen a ponton. Mivel a `data frame`-ben minden eredményváltozó (y_i) és magyarázóváltozó (x_i) érték ismert, így ezek a függvényben *konstansnak* vehetők, a függvény két változója csak a β_0 és a β_1 . Emiatt a függvény minimumhelyét úgy kereshetjük meg, hogy vesszük ennek az $SSE(\beta_0, \beta_1)$ hibafüggvénynek a parciális deriváltjait a két β szerint, és azokat egyenlővé tesszük nullával.

Magyarul megoldjuk az alábbi két egyenletből álló egyenletrendszeret:

$$\frac{\partial SSE(\beta_0, \beta_1)}{\partial \beta_0} = 0$$

$$\frac{\partial SSE(\beta_0, \beta_1)}{\partial \beta_1} = 0$$

Az egyenletrendszer megoldásával kifejezhető az a fix formula, amivel az OLS elven számított β_0 és β_1 értékek megadhatók:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 x$$

Azt, hogy konkrétan hogyan jön ki ez a csodaszép két formula szorgalmi feladatként levezethető +1 pontért. :) A gyakorlati szempontból azt kell látni, hogy minimalizálási feladathoz nem is kell az optim függvény, mert a megoldása van egy-egy fix képlet. **Emiatt használják az OLS regressziót mai napig előszeretettel: fix képpel megadhatók a β együtthatók, és nem kell a megadásukhoz optimalizálni!**

12.3. Az OLS Regresszió magyarázóerejének mérése

Megmérhetjük azt is, hogy a lakások területe hány %-ban magyarázza az áraik ingadozását az átlagos ár körül. Sőt, igazából ezt meg is mértük már a korreláció négyzetével, a determinációs együtthatóval (R^2) és 73.899%-nak adódott. De ez az érték meghatározható az *SSE* hibafüggvényünk alapján is!

Ehhez ki kell számolni, hogy mennyi az eredményváltozóban lévő teljes ingadozás, azaz **információtartalom**, ami megmagyarázható. Ezt úgy mérjük meg, hogy megnézzük mi lenne az *SSE*, ha az eredményváltozót (az árakat) 0 magyarázóváltozóval akarjuk megbecsülni, előrejelezni. Mondván ennél rosszabb becslést nem adhatunk az eredményváltozóra, így ennek a becslésnek az összhibáját tudjuk csökkenteni a magyarázóváltozókat használó regressziós modellel. Ha **nincs magyarázóváltozónk, akkor a becslésünk minden esetben az átlagár lesz: $\hat{y} = \bar{y}$.**

Ezzel azt mondhatjuk, hogy a magyarázóváltozók nélküli **nullmodell** *SSE*-je valójában az *SumOfSquaredTotals* = $SST = \sum_{i=1}^n (y_i - \bar{y})^2$, tehát az egyes lakások árainak négyzetes ingadozása az átlagos ár körül. Ezt hívjuk az **eredményváltozóban lévő teljes megmagyarázható információnak!**

Számoljuk is ezt ki a lásárakra!

```
SST = np.sum((BP_Lakas.KinArMft - np.mean(BP_Lakas.KinArMft))**2)
SST
## 541719.8980725462
```

12.4. A REGRESSZIÓ MAGYARÁZÓEREJE NEM MEGFIGYELETT ADATOK KÖRÉBEN305

Ha a modell hibászálékát akarjuk megnézni, akkor egyszerűen az SSE -t elosztom az SST -vel. Hiszen ekkor megkapom, hogy a modell az összes megmagyarázható információból SST mennyit **nem** tudott megmagyarázni (SSE). Nyilván, ha ennek a hányadosnak veszem a komplementerét (1-), akkor megkapom a modell által megmagyarázott információ-hányadot. Ez lesz a már ismerős **R-négyzet**, vagy szébb szóval **determinációs-együththató**:
$$R^2 = 1 - \frac{SSE}{SST}$$

Számoljuk is ki:

```
SSE = eredmeny.fun  
1 - SSE / SST
```

```
## 0.7391439241654998
```

Tehát, a lakások területe továbbra is **73.914%-ban magyarázza a lakások árának ingadozását** (alakulását). Ez egy elég jó modell még mindig! :)

Mindez képletek szintjén összefoglalva (szokjunk hozzá, hogy az angol könyvekben az SSE -t néha ESS -nek is jelölik :)):

$$\underbrace{\sum_{i=1}^n (Y_i - \bar{Y})^2}_{\substack{SST \\ = ESS_{Null}}} - \underbrace{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}_{SSE} = \underbrace{\sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2}_{\substack{SSR \\ = \text{Hibacsökkenés}}}$$

$$R^2 = \frac{TSS - ESS}{TSS} = 1 - \frac{ESS}{TSS} = \frac{RSS}{TSS}$$

12.4. A Regresszió magyarázóereje nem megfigyelt adatok körében

Szép és jó, hogy tudom, hogy a regresszióm magyarázóereje durván 74%, de ez csak azt mondja, hogy a **megfigyelt 1405 lakás esetében magyarázza kb. 74%-ban a terület az árak alakulását!** Szeretnék valamit mondani azért a modell magyarázóerejéről a nem megfigyelt lakások körében is! Hiszen az \hat{y} becsléseket arra akarom használni, hogy új, eddig még nem látott budapesti lakások esetében is meg tudjam becsülni az árakat a terület alapján! Tehát, a regresszió viszelkedésére vagyok kíváncsi a budapesti lakások **sokaságában**!

Ennek az eszköze a klasszikus Statisztikában a **hipotézisvizsgálat**!

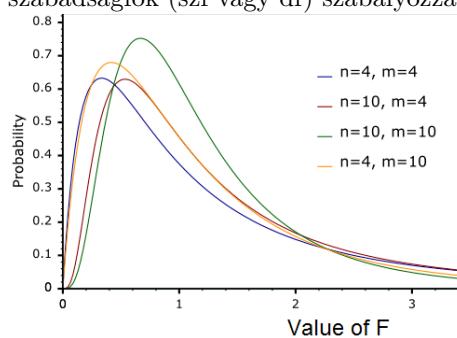
A hipotézisvizsgálatunk kérdése tehát az, hogy mi történik a megfigyeléseinken (mintánkon) túli világban (leánykori nevén sokaságban) ezzel a regressziós

modellel? Ha új lakásokra általánosítjuk ki az OLS regressziós modellt, akkor az R^2 értéke megmarad vagy összeomlik 0-ra?

Ezt a kérdést lehet megválaszolni egy R^2 -re adott hipotézisvizsgálattal, a **Globális F-próbával**.

Konkrétan ebben az esetben a hipotézisvizsgálat 4 lépéses folyamat így néz ki:

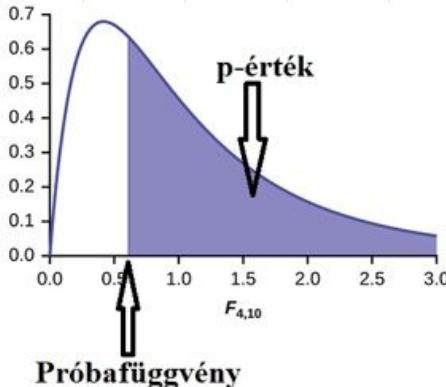
- Pesszimista emberek vagyunk, azt gondoljuk, hogy modellünk a mintán kívüli világban semmit nem magyaráz, azaz $R^2 = 0$
 - Hopp, ez egy egyenlőséggel adott állítás! Mehetsz H_0 -ba $\rightarrow H_0 : R^2 = 0$ (vagyis a modell nem szignifikáns a sokaságban)
 - Alternatív esetben pedig legyünk optimisták: ebből a magyarázóerőből, amit a mintában mértünk, fog maradni akkor is valami, ha új lakásokat kezdünk vizsgálni a sokaságból $\rightarrow H_1 : R^2 > 0$ (vagyis a modell szignifikáns a sokaságban)
- A próbafüggvényt számoljuk a mintában mért R-négyzet, a mintaellemszám (n) és az egyenletben használt β paraméterek száma (p) alapján!
 - Nekünk: $R^2 = 0.73914, n = 1406, p = 2$
 - Próbafüggvény képlete: $\frac{R^2/(p-1)}{(1-R^2)/(n-p)} = \frac{0.73914/(2-1)}{(1-0.73914)/(1406-2)} = 3978.197$
- Ez a **próbafüggvény igaz H_0 esetén** egy úgynevezett **F-eloszlást követ**.
 - Azt, hogy ez az eloszlás sűrűségfüggvényként hogy néz ki, két szabadságfok (szf vagy df) szabályozza:



- Ezt a két szabadságfokot mi az elemszámból és a magyarázóváltozók számából fogjuk megkapni a saját próbafüggvényünkre.
 - $- df_1 = p - 1 = 2 - 1 = 1$ és $df_2 = n - p = 1406 - 2 = 1404$
- H_0 akkor biztosan (100%-os valószínűséggel) igaz, ha R^2 már a mintában is 0.
- Ezért a p-értékkel azt mérjük le, hogy „milyen messze vagyunk” ettől a H_0 szempontjából ideális állapotról az igaz H_0 esetén fennálló eloszlásban.

12.4. A REGRESSZIÓ MAGYARÁZÓEREJE NEM MEGFIGYELETT ADATOK KÖRÉBEN307

- Egész konkrétan így:



- Tehát, a p-érték most a próbafüggvény feletti terület a megfelelő F-eloszlásban. Hiszen, így a *próbafüggvény* = 0 esetben 100%-ra jön ki a p-érték, ami azt jelenti, hogy H_0 -t elvetni tuti, hogy hibás döntés. A próbafüggvény pedig arányosan nőni fog, ha R^2 is nő, így ha kellően magas az érték, akkor már nagyon kicsi lesz a felette lévő terület, így alacsony annak a valószínűsége, hogy H_0 elvetésével hibát követünk el.
 - F-eloszlásban egy értékhez tartozó „felé esési” valószínűséget Pythonban a következő `scipy`-os `cdf` függvénnyel kapjuk meg (figyeljünk arra, hogy a szabadságfokok sorrendje a 2. és 3. paraméterben számít): `1-stats.f.cdf(3978.197, 1, 1404) = 1.1102e-16 = 0.`
4. Ez a p-érték jó alacsony, konkrétan 0%-nak vehető. Tehát még ha azt is, mondom, hogy csak $\alpha = 1\%$ valószínűsséggel engedek meg igaz H_0 -t elutasítani, ez a valószínűség még ennél is kisebb.
- Tehát nyugodtan elvethetem H_0 -t, mivel ezzel a döntéssel 0% az esélye, hogy hibázok.

!!!VIGYÁZAT!!! \rightarrow Kellően nagy minták esetén ebből a tesztből simán kijöhets egy 3,4%-os mintabeli R^2 esetén is, hogy H_1 állítást kell igaznak venni. Ami annyit jelent, hogy azt a 3,4%-os R^2 -et ki lehet általánosítani a megfigyeléseken túli világra is... de ez azért nem olyan jó eredmény, mintha egy 74%-os R^2 -re jön ki ugyan ez egy ilyen hipotézisvizsgálatból!

Megjegyzés: Globális F-próbában H_0 -t úgy is meg lehet fogalmazni, hogy a modellben a meredekség 0 a mintán kívüli világban: $\beta_1 = 0$. A H_1 -et meg ilyenkor úgy lehetne felírni, hogy a meredekség nem nulla: $\beta_1 \neq 0$.

12.5. A modell együtthatóinak értelemezése

A kétváltozós OLS modellünk két β paramétere tehát az alábbiak és a következő jelentéstartalommal bírnak:

- $\beta_0 = -4.312 \rightarrow$ Ő ugye a tengelymetszet, ami megadja, hogy a regressziós egyenes hol metszi az y tengelyt, ami az $x = 0$ helyen történik. Tehát, technikailag megadja, hogy $x = 0$ esetén mi a regresszió \hat{y} becslése.
- $\beta_1 = +0.400 \rightarrow$ Ez itt pedig a meredekség, ami megmutatja, hogy +1 egység elmozdulás esetén az x tengelyen hogyan változik az \hat{y} .

A mi példánkban ez azt jelenti, hogy egy 0 m^2 -ú lakás **becsült** ára -4.312 M Ft. Nyilván ezzel az értelmezéssel most nem kell foglalkozni, mivel az $x = 0$ helyen jelen adatbázisunkban nem létezik. :) A meredekség alapján pedig azt mondhatjuk, hogy +1 négyzetméter terület a lakás értékét **várhatóan** (hiszen \hat{y} -ról beszélünk) 0.400 M Ft-tal, azaz 400 E Ft-tal növeli. Ha nagyon közgázosak akarunk lenni, akkor ez a 400 ezer Ft a +1 négyzetméter **hasznossága**!

Kimondottan a β_1 értelmezésénél nagyon-nagyon figyeljünk oda, hogy **az értelmezés nagyon mértékegységszámos**!! Tehát, a +1 egység x az minden az x mértékegységében értendő, ami most nekünk négyzetméter. Míg maga a β_1 -nyi változás pedig értelemszerűen az y mértékegységében adott, ami a példánkban MFT!

12.6. Egy gyakorló példa COVID adatokon

Töltsük be a COVID_0326.xlsx c. Excel fájlt egy **covid** nevű pandas **data frame**-be!

```
covid = pd.read_excel("COVID_0326.xlsx")
covid.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 100 entries, 0 to 99
## Data columns (total 9 columns):
## #   Column           Non-Null Count  Dtype  
## ---  -- 
## #   0   Country        100 non-null    object 
## #   1   Pop            100 non-null    int64  
## #   2   PopDens        100 non-null    int64  
## #   3   Prop_UrbanPop  100 non-null    int64  
## #   4   PressLiberty   100 non-null    float64
## #   5   PerCapita_HealthExp_2016 100 non-null    float64
```

```
## 6 COVID19_CasesPerMillion    100 non-null    float64
## 7 ConstitutionalForm        100 non-null    object
## 8 HeadOfState                100 non-null    object
## dtypes: float64(3), int64(3), object(3)
## memory usage: 7.2+ KB
```

A fájlból betöltött tábla a Föld 100 országára, mint megfigyelési egységekre nézve tartalmazza az alábbi 9 változó adatait:

- Country: ország neve
- Pop: ország népessége (fő)
- PopDens: népsűrűség (fő/km²)
- Prop_UrbanPop: Városi népesség aránya 2019-ben (%)
- PressLiberty: Sajtószabadsági index 2019-ben (alacsonyabb = szabadabb sajtó)
- PerCapita_HealthExp_2016: Egy főre jutó eüg-i kiadások, vásárlóerő-paritáson számolva (2016)
- COVID19_CasesPerMillion: Egymillió főre jutó COVID-19 fertőzöttek száma 2020.03.26-án
- ConstitutionalForm: államforma
- HeadOfState: államfő jogkörei (hatalmat gyakorol vagy csak reprezentatív szerepkörű)

Nézzünk egy kétváltozós regressziót az egymillió főre jutó COVID esetszám (**COVID19_CasesPerMillion**) és a sajtószabadsági index (**PressLiberty**) között. Egy logikus feltételezés, hogy az ország sajtószabadsági indexéből következik az, hogy a COVID járvány korai szakaszában (2020. március 26-án) hány COVID fertőzöttet regisztráltak az országban népességarányosan. Mondván egy sajtónyilvánosabb országban kevésbé tudja eltitkolni a központi egészségügy a fertőzöttek valódi számát. Ebből adódóan a magyarázóváltozó lesz a regresszióban az **x = PressLiberty** és az eredményváltozó pedig az **y = COVID19_CasesPerMillion**. És akkor a következő regressziós egyenlet β_1, β_0 együtthatóit kéne megbecsülni a data frame-ben található 100 ország adatai alapján

$$\text{BecsltCOVID19} = \beta_1 \times \text{PressLiberty} + \beta_0$$

Szerencsére, nem kell újra a `minimize` függvényel szenvednünk ahhoz, hogy megkapjuk a β_1, β_0 regressziós együtthatók legkisebb négyzete elvű (OLS) becslését. A `numpy` csomag `polyfit` függvénye legyártja ezeket nekünk. A függvény első paramétere a magyarázóváltozó (*x*), második az eredményváltozó (*y*) a regresszióban. Míg a harmadik paraméterben, egy `deg=1` beállítással azt jelezük a gépállatnak, hogy a megadott *x* és *y* koordinátkra egyenest illesszen, ne valami bonyolultabb alakzatot (pl. másodfokú polinomot).

```
Beta1, Beta0 = np.polyfit(covid.PressLiberty, covid.COVID19_CasesPerMillion, deg = 1)
print([Beta1, Beta0])
```

```
## [-4.283796024297913, 214.90178214724114]
```

A 100 országra legjobban illeszkedő egyenes egyenlete tehát:

$$Becs\lt COVID19 = -4.28 \times PressLiberty + 214.9$$

Tehát, ha a sajtószabadsági index 1 egységgel nő (azaz a sajtó egy egységgel kevésbé lesz szabad) egy országban, akkor *várhatóan* 4.28 eset/millió fővel **csökken** az ország fertőzötteinek száma. Érdekes eredmény, de ez feltehetőleg azért van így, mert 2020.03.26-i állapotokat tükröz az adatbázis, és akkor még intenzív tesztelést inkább csak a gazdagabb, és szabadabb sajtóval bíró országok végeztek. Plusz, a sok kis észak-európai államban (Belgium, Hollandia, Dánia, stb.) nagy a sajtószabadság, de nagy a városi népesség aránya és a népesűrűség is, így ott nyilván jobban tud terjedni a vírus. De nyilván benne van ebben a negatív együtthatóban az is, hogy ekkor egy Pakisztán nagyon igyekezett még eltitkolni a fertőzöttek valós számát és a Pakisztánnál szabadabb sajtóval bíró Olaszországban nem igazán tudták eltitkolni, hogy mekkora a baj. Érdekességeképpen elmondhatjuk, hogy egy 0 szajtószabadsági indexsel (tehát a legszabadabb sajtóval) rendelkező országan a modell 214.9 fertőzöttet becsül 1 millió főre.

Nézzük meg a modell %-os magyarázóerejét, azaz az R^2 mutatóját.

```
(stats.pearsonr(covid.PressLiberty, covid.COVID19_CasesPerMillion)[0]**2)*100
```

```
## 24.40883012227656
```

Láthatjuk, hogy a modellnek csak egy közepes (de nem gyenge!) magyarázóereje van, hiszen a sajtószabadsági index csak 24.41%-ban magyarázza az egymillió főre jutó esetszámok alakulását, ami 10%-50% közötti érték.

Nézzük meg, hogy ez az R^2 lehet-e 0% nem megfigyelt országok esetén (a Föld országainak telejs sokaságában)? Azaz, nézzük meg a glogális F-próba p-értékét!

Ehhez először kell egy próbafüggvény a 4. fejezetből megismert képlet alapján:

```
probafuggveny_F = (0.2441/(2-1))/((1-0.2441)/(100-2))
```

Majd jobboldali módon jöhet a p-érték egy $F(2 - 1, 100 - 2)$ eloszlásból.

```
1-stats.f.cdf(probabafuggveny_F,1,98)
```

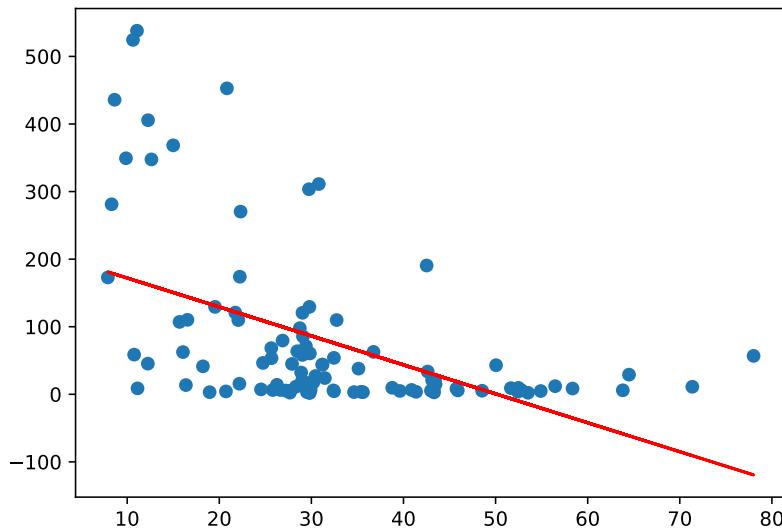
```
## 1.752395408782803e-07
```

Tehát, a modell F-próbájának p-értéke csupán 1.75×10^{-7} , ami kisebb még a legkisebb szokásosnak mondható $\alpha = 1\% = 0.01$ szignifikancia-szintnél is, így azt mondhatjuk, hogy a modell 24.41%-os magyarázóereje szignifikáns marad a 100 megfigyelt országon kívüli világban is.

Mindezen eredmények vizualizálva pontdiagramon:

```
# Kiszámoljuk az y kalapokat, azaz a becsült árakat ezekkel a Bétákkal
covid['BecsultCOVID'] = Beta1*covid.PressLiberty + Beta0

# Ábrázoljuk az új egyenest a pontdiagramon
plt.scatter(covid.PressLiberty, covid.COVID19_CasesPerMillion)
plt.plot(np.array(covid.PressLiberty),
         np.array(covid.BecsultCOVID), 'red')
plt.show()
```



Láthatjuk, hogy a pontokra legjobban illeszkedő egyenes negatív meredekségű, csökkenő trendet mutat, és aránylag jobban szóródnak az egyenes körül a pontok, és a 95%-os konfidiencia-intervalluma is viszonylag (de nem vészesen) tág, de az egymillió főre jutó esetszámok csökkenő trendje a **PressLiberty**

függvényében még szépen kirajzolódik. Mindez az R^2 által is jelzett közepek erősséggű kapcsolatra utal.

13. fejezet

Többváltozós OLS Regresszió alapjai

13.1. Magyar járások COVID-19 halálozási arányai

A COVID_JarasData.xlsx fájl egy olyan adattábla, ami 102 magyar járásról (azokról, ahol a kórházak 2019-ben átlagos vagy átlag alatti leterheltségűek voltak a NEAK adatai alapján) 5 változó (oszlop) adatát tárolja:

- **Jaras:** A járás neve
- **COVIDHalal:** COVID-19 halálozási arány: elhunytak / fertőzöttek hányadosa (%) 2021.03.04-én. Forrás: atlatszo.hu
- **Apolok:** Háziorvosi szakápolók/ápolók száma 10000 főre (2019) Forrás: KSH
- **Munkanelkuliseg:** Nyilvántartott álláskeresők száma 10000 főre (2019) Forrás: KSH
- **Nok65Felett:** Lakónépességből a 65 éves és idősebb nők aránya (%) (2019) Forrás: KSH

Olvassuk be a fájlt egy pandas data frame-be a pandas sima `read_excel` függvényével:

```
# Elemzéshez és ábrázoláshoz szükséges csomagok betöltése
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
```

```
# 102 járás adatainak beolvasása
covid = pd.read_excel("COVID_JarasData.xlsx")
covid.info()

## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 102 entries, 0 to 101
## Data columns (total 5 columns):
## #   Column      Non-Null Count  Dtype  
## ---  --          --          --    
## 0   Jaras       102 non-null    object 
## 1   COVIDHalal 102 non-null    float64
## 2   Apolok      102 non-null    float64
## 3   Munkanelkulisege 102 non-null    float64
## 4   Nok65Felett 102 non-null    float64
## dtypes: float64(4), object(1)
## memory usage: 4.1+ KB
```

Láthatjuk, hogy megvan minden oszlop, amit a leírásban megadtunk. Megvan mind a négy numerikus (azaz `float`, mert törtszámokról van szó) típusú oszlopunk, és a **Jaras** oszlop maradhat `object`, azaz „szöveges” adattípusban, hiszen ez a járások azonosítója, minden név csak egyszer fordul elő a táblában, statisztikai elemzéseknek értelemszerűen nem vetjük alá ezt a változót. :)

Adj a magát a doleg, hogy megpróbáljuk **kétváltozós regressziókkal** elemezni azt, hogy miképpen függ a vizsgált járásokban a COVID halálozási arány a másik három változótól (ápolók, munkanélküliek száma 10 ezer főre, 65 éven felüli nők aránya).

Első logikus gondolat azt diktálja, hogy hát minél több ápolója van népességarányosan egy járásnak, annál kevesebb lesz a COVID halálozási arány, hiszen a több ápoló jobba ellátást tud biztosítani. Teszteljük is le az elméletünket: csinálunk egy regressziót, ahol eredményváltozó (Y) a **COVIDHalal**, magyarázóváltozó (X) pedig az **Apolok**!

Első körben nézzük meg egy R^2 segítségével milyen szoros a két változónk köztü kapcsolat:

```
(stats.pearsonr(covid.Apolok, covid.COVIDHalal)[0]**2)*100
```

```
## 17.627405081896164
```

Alapvetően az R^2 szép, korrekt dolgokat mutat. Az ápolók száma kb. 18 %-ban megmagyarázza a COVID halálozás alakulását a vizsgált járások körében ($R^2 = 17.6\%$), ami egy közepes magyarázóérőnek minősíthető, hiszen $10\% < R^2 < 50\%$.

Nézzük meg, hogy néz ki a regressziós egyenes egyenlete! Használjuk nyugodtan a 12.6. fejezetben megismert `polyfit` függvényét a `numpy` csomagnak:

```
Beta1, Beta0 = np.polyfit(covid.Apolok, covid.COVIDHalal, deg = 1)
print([Beta1, Beta0])
```

```
## [0.3629321944430193, 2.02043692521212]
```

A 102 járás adatából illesztett regressziós modell szerint tehát:

$$BecsltCOVIDHalal = 0.36 \times Apolok + 2.02$$

Ajjajjaj! A modell **meredeksam** pozitív!! Konkrétan, ha a járásban az ápolók száma 10 ezer főre nő 1-gyel, akkor várhatóan a COVID halálozási arány is **növekedni** fog 0.36 százalékponttal! Ez alapján úgy tűnik, hogy mintha megérne kevesebb ápolót tartani, mert akkor fog csökkeni a COVID halálozás a járásban. Ezt az eredményt így nagyon nem eszi meg a gyomrunk!

A fura eredmény háttérben egy úgynevezett **confounding** nevű jelenség áll!

Lessük csak meg a minden **covid** data frame-ben lévő numerikus változó korrelációmátrixát! Ez egy olyan táblázat, amiben az egyes numerikus változók közti korrelációkat mutatja meg nekünk a gépállat. Simán egy **data frame** numerikus változóból a **data frame** numerikus oszlopain (mindegyik kivéve az elsőt) elsütött `cor` metódussal hívható elő.

```
covid.iloc[:, 1:5].corr()
```

	COVIDHalal	Apolok	Munkanelkulisegek	Nok65Felett
## COVIDHalal	1.000000	0.419850	0.495429	0.411443
## Apolok	0.419850	1.000000	0.540570	0.517462
## Munkanelkulisegek	0.495429	0.540570	1.000000	0.228315
## Nok65Felett	0.411443	0.517462	0.228315	1.000000

A korrelációkból látszik, hogy a **COVIDHalal** változóval a másik három változó egyirányú, és közepes erősséggű kapcsolatban állnak (mindegyik korreláció a COVIDHalal oszlopában, ami nem az önmagával vett korrelációt jelenti az +0.4 körüli érték). Ez a **Munkanelkulisegek** és **Nok65Felett** változók esetén logikus is, hiszen a COVID elsősorban a 65 év felettesek körében halálos, illetve azok körében ahol eleve több alapbetegség jelen volt. Ahol magas a munkanélküliség, ott pedig eleve rosszabb az egészségi állapot: alkoholizmus, szív- és érrendszeri problémák gyakoriak a magas munkanélküliségű járásokban (lásd pl. ezt a tanulmányt). NODE, az **Apolok** egyirányú módon és közepes erősségen összefügg a **Munkanelkulisegek** és **Nok65Felett** változókkal is! Tehát, valószínűleg a magas 10 ezer főre jutó ápolószámmal bíró járásokban

csak azért magas a COVID halálozás, mert ezekben a járásokban magas a munkanélküliség és az idős népesség aránya is! Tehát a népesség egészségi állapota ezen járásokban **eleve rosszabb!** Na, ez a jelenség a **confounding**: amikor egy változó csak azért korrelál egy másikkal, mert **valójában egy vagy több másik változó hatását közvetíti a saját hatásán túl.**

Szóval, a feladat adott: fejtsük meg, hogy ha leválasztjuk az **Apolok** változóról a **Munkanelkuliseg** és **Nok65Felett** változók hatását (azaz kiszűrjük/megtisztítjuk a *confounding* hatást), akkor hogyan hat az **Apolok** változó **önállóan** a **COVIDHalal**-ra! Erre a feladatra eszözünk a **többváltozós lineáris regresszió!** Ami egyszerűen a sima $\hat{Y} = \beta_1 X_1 + \beta_0$ kétváltozós regresszió kiterjesztése úgy, hogy az egyenletbe tetszőleges, konkrétan k db magyarázóváltozót rakunk be, és mindenkinet meglesz a maga β -ja:

$$\hat{Y} = \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_k X_k + \beta_0$$

Az előbbi egyenlet alapján a **többváltozós lineáris regressziót** úgy is értelmezhetjük, mint egy olyan statisztikai modellt, ami a COVID halálozási arányokat akarja előrejelezni **egyszerre az ápolószám, munkanélküliség és 65 év feletti női népesség aránya segítségével!**

13.2. A Többváltozós Lineáris Regresszió OLS elvű előállítása

A nagy szerencsénk, hogy a többváltozós regresszióban is lényegében változtatás nélkül működik a β_j -k OLS elvű meghatározási módja. Azaz, úgy választjuk meg a β_j -ket, hogy az COVID halálozási arányokra (Y) adott becsléseink hibája minimális legyen. A becslési hibát az ún. SSE mutatóval mérjük továbbra is: $\text{Sum of Squared Errors (SSE)} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$.

Az továbbra is igaz lesz, hogy négyzetesen mért hiba esetében a gép itt **sem vaktában keresi** a β -kat! Az OLS feladat megoldása (tehát a legkisebb SSE-t adó β -k) kifejezhetők egy fix képlettel itt is!

Az $\text{SSE} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_i (y_i - \sum_j \beta_j x_j)^2$ képlet kifejezhető mátrixosan is, ha bevezetjük az X mátrikot, aminek az első oszlopa csupa 1-öt tartalmaz (a tengelymetszet miatt) és a többi oszlopaiba az X_j -ket rakjuk, akkor az SSE-t így is fel tudjuk írni: $\text{SSE} = \|y - X\beta\|^2$. Ha ezt a mátrixosan kifejezett függvényt deriváljuk a β -k szerint, és a deriváltakat egyenlővé tesszük 0-val, akkor kifejezhető az a fix formula, amivel az OLS elven számított β -k vektora megadható: $\beta = (X^T X)^{-1} X^T y$.

Azt, hogy konkrétan hogyan jön ki ez a csodaszép mátrixos formula, nekünk annyira nem fontos. A gyakorlati szempontból azt kell látni, hogy minimalizálási feladathoz többváltozós regresszió esetén sem kell a 3. gyakorlaton látott optim

13.2. A TÖBBVÁLTOZÓS LINEÁRIS REGRESSZIÓ OLS ELVŰ ELŐÁLLÍTÁSA 317

függvény, mert a megoldása egy fix mátrixos képlet. **Emiatt használják az OLS regressziót mai napig előszeretettel: fix képlettel megadhatók a β együtthatók többváltozós esetben is!**

Az a nagy szerencse, hogy a mi három magyarázóváltozót használó modellünk együtthatóit ilyen OLS elven meg lehet becsülni Pythonban a `statsmodels` csomag `OLS` nevű függvényével. Ehhez először importáljuk a csomag függvényeit egy `sm` c. névtérbe.

```
import statsmodels.api as sm
```

A függvény használatához először el kell különíteni egy külön objektumba az eredményváltozó oszlopát a vizsgált `data frame`-ból. Hívjuk most ezt az objektumot `Y`-nak.

```
Y = covid.COVIDHalal
```

Majd ezek után egy külön `data frame`-be összegyűjtöm csak a magyarázóváltozókat, hívjuk ezt az új `data frame`-t most szimplán `X`-nek. Utána a `statsmodels` csomag `add_constant` függvényével hozzáadunk egy csupa 1-ből álló oszlopot, ahogy a regresszió korábban látott mátrixos felírásában is vettük az `X` mátrixot.

```
X = covid.loc[:,['Apolok','Munkanelkulisege','Nok65Felett']]  
X = sm.add_constant(X)  
X
```

```
##      const    Apolok  Munkanelkulisege  Nok65Felett  
## 0      1.0    7.354416    369.057966   14.735575  
## 1      1.0   10.385919    227.396961   14.873729  
## 2      1.0    6.256924    227.403274   13.129282  
## 3      1.0    4.421334     65.454965   11.351294  
## 4      1.0    6.876948    676.691729   12.350999  
## ...     ...       ...        ...          ...  
## 97     1.0    7.377035    625.503954   14.313164  
## 98     1.0    7.567731    586.751425   14.237425  
## 99     1.0    5.884826    287.876058   13.452231  
## 100    1.0    8.136283    520.722095   16.730231  
## 101    1.0    8.535544    338.983051   15.522497  
##  
## [102 rows x 4 columns]
```

És végül akkor az `sm` névtér `OLS` függvényébe berakjuk ezeket az újonnan létrehozott `Y` és `X` Python objektumokat (kötelezően ebben a sorrendben), és végül meghívunk az egészen egy `fit()` metódust. Az eredményt pedig egy külön Python objektumba kell elmenteni.

```
sokvalt_modell = sm.OLS(Y,X).fit()
```

A többváltozós regressziós modell legfontosabb statisztikai mutatóit egy nap összefoglaló táblázatban az az újonnan létrehozott `sokvalt_modell` objektum `summary()` metódusával lehet lekérdezni.

```
print(sokvalt_modell.summary())
```

```
##                                     OLS Regression Results
## =====
## Dep. Variable:      COVIDHalal    R-squared:           0.341
## Model:                 OLS        Adj. R-squared:       0.321
## Method:                Least Squares   F-statistic:         16.89
## Date:          H, 30 jún. 2025   Prob (F-statistic):  6.42e-09
## Time:                  15:25:07    Log-Likelihood:     -152.39
## No. Observations:      102        AIC:                  312.8
## Df Residuals:          98        BIC:                  323.3
## Df Model:                   3
## Covariance Type:    nonrobust
## =====
##            coef    std err      t    P>|t|    [0.025    0.975]
## -----
## const      -0.1542    0.937   -0.165    0.870   -2.015    1.706
## Apolok      0.0445    0.096    0.463    0.644   -0.146    0.235
## Munkanelkuliseg  0.0028    0.001    4.099    0.000    0.001    0.004
## Nok65Felett  0.2656    0.087    3.053    0.003    0.093    0.438
## =====
## Omnibus:             25.226 Durbin-Watson:           1.957
## Prob(Omnibus):        0.000 Jarque-Bera (JB):      36.179
## Skew:                  1.173 Prob(JB):            1.39e-08
## Kurtosis:                 4.735 Cond. No.        3.10e+03
## =====
## Notes:
## [1] Standard Errors assume that the covariance matrix of the errors is correctly specified
## [2] The condition number is large, 3.1e+03. This might indicate that there are
## strong multicollinearity or other numerical problems.
```

Nézzük meg mit látunk a `summary` eredményeként:

- Mivel az OLS elvű becslése, tehát a teljes modellhiba SSE elvű mérése megmaradt, így R^2 -et továbbra is tudunk számolni $R^2 = 1 - \frac{SSE}{SST}$ módon. Az értelmezése pedig annyiban módosul, hogy azt adjja meg hány százalékban magyarázza az eredményváltozót a

magyarázóváltozók összessége. Arra figyeljünk, az előbbi értelmezés miatt, hogy korreláció négyzeteként már **NEM** számolható a mutató, hiszen korreláció *egyszerre csak két változó kapcsolatát tudja leírni*, kettőnél több több változtót már nem. Ez alapján jelen szituációban azt mondhatjuk el, hogy az ápolók száma, 65 feletti női népesség aránya és munkanélküliség **együtt** 34%-ban magyarázzák a COVID halálozási arányok ingadozását a járások között. Ez érezhető magyarázóerő növekedés a kétváltozós modellhez képest, ahol csak az **Apolok** hatását vizsgáltuk.

- A Globális F-próba p-értéke most is jó alacsony, mivel $\text{Prob}(F\text{-statistic}) = 6.42 \times 10^{-9}$. Tehát, nem meglepő módon azt mondhatjuk, hogy minden szokásos α -n elutasíthatjuk, azt a H_0 -t, hogy az R^2 összeomlik 0-ba a mintán kívüli világban. Amit itt érdemes megfigyelni az a p-érték számoláshoz használt F-eloszlás szabadságfokai. Most nekünk ugye $k = 3$ magyarázóváltozónk van, az azt jelenti, hogy a regresszióban $p = 4$ paramétert, azaz β_j -t kellett megbecsülnie az OLS-nek a tengelymetszet miatt. Ez alapján az F-eloszlás első szabadságfoka $p-1 = 3 = k$, míg a második szabadságfok $n-p = 102-4 = 98 = n-k-1$.
- Ezen kívül az F-próba hipotéziseit úgy is fel lehet írni, mint: $H_0 : \beta_j = 0 \forall j$, azaz minden β a mintán kívüli világban 0-nak vehető, semminek nincs magyarázóereje Y -ra. És $H_1 : \exists j : \beta_j \neq 0$, tehát van legalább egy darab β , ami nem nulla a mintán kívüli világban, és a hozzá tartozó X_j -nek van hatása Y -ra a mintán kívüli világban is.

A **Coefficients** táblából ismét felírható a megbecsült modell egyenlete:

$$\text{BecsltCOVIDHalal} = 0.04 \times \text{Apolok} + 0.003 \times \text{Munkanelkuliseg} + 0.27 \times \text{Nok65Felett} - 0.15$$

Az egyenlet olyan szempontból furának tűnik, hogy az **Apolok** együtthatója még mindig pozitív, bár tény, hogy már csak 0.04 az értéke a kétváltozós modell 0.36-os meredeksége helyett. Ahhoz, hogy pontosan megértsük mit is takar ez a 0.04 és megadjuk hogyan lehet megmérni egy magyarázóváltozó (X_j) fontosságát a regressziós modellünk előrejelzéseiben, egy kicsit alaposabban meg kell érteni mit is takarnak ezek a β_j értékek a többváltozós regresszióban.

13.3. A magyarázóváltozók marginális hatása

Ahogyan a 2. fejezetben írtam, a az OLS regressziót azért használják a mai napig előszeretettel, mert fix képlettel megadhatók a β_j együtthatók. De ez csak az egyik oka a modell népszerűségének. A másik az, hogy a β_j együtthatók megfeleltethetők a hozzájuk tartozó X_j magyarázóváltozók **marginális hatásának**.

Az X_j magyarázóváltozó marginális hatása az a hatás, amit ō **egyedül** és **közvetlenül** fejt ki az Y eredményváltozó alakulására. OLS regresszióban ez a marginális hatás X_j magyarázóváltozó esetében épp a β_j együttható lesz.

Most ez alapján egy β_j általános értelmezése a következő: **Ha az adott bétához tartozó magyarázóváltozó értéke egy egységgel nő a többi magyarázóváltozó értékének változatlansága mellett, akkor az eredményváltozó értéke várhatóan bétányit változik.**

Ennek az értelmezésnek **3 kötelező eposzi kelléke** van:

1. minden változás az eredményváltozó és az éppen vizsgált magyarázóváltozó **saját mértékegységében** értendő
2. Az éppen nem vizsgált magyarázóváltozók értékéről feltesszük, hogy nem változnak. Ezzel az éppen vizsgált X_j közvetlen hatását mérjük meg Y -ra. Ez a **ceteris paribus elv**.
3. A β_j az Y -ban okozott **várható** változást mutatja csak! Ez a változás akkor lenne egész biztosan épp β_j -nyi, ha $R^2 = 100\%$ lenne.

Ha ez az értelemezés így általánosságban igaz, akkor az **Apolok** +0.04-es β ja a többváltozós regresszióinkban már megtisztult a munkanélküliség és a 65 év feletti népesség **confounding** hatásától, hiszen a 0.04 egy ezek megváltozása *nélküli* +1 egység **Apolok** növekedés esetén mutatja be a COVID halálozás várható változását.

Ezek alapján nézzük meg a jelenlegi modellünkben vett β_j -k értelmezése! Ha csak a modell β_j -it akarom lekérdezni és semmi egyebet, akkor azt a modellt tartalmazó Python objektum `params` tulajdonságának lekérdezésével tudom megtenni.

```
sokvált_modell.params
```

```
## const          -0.154215
## Apolok        0.044522
## Munkanelkulisege 0.002819
## Nok65Felett   0.265567
## dtype: float64
```

- $\beta_1 = 0.04$: Ha egy járásban a 10 ezer főre jutó ápolók száma nő 1 fővel változatlan munkanélküliség és 65 év feletti női népesség mellett, akkor a járás COVID halálozási aránya várhatóan 0.04 százalékponttal emelkedik. Ez még mindig nem tűnik szímpatikusnak, de majd mindenki a végére járunk.
- $\beta_2 = 0.003$: Ha egy járásban az álláskeresők száma 10 ezer fővel (azaz a 10 ezer főre jutó álláskeresők száma 1 fővel) nő változatlan ápolószám és 65 év feletti női népesség mellett, akkor a járás COVID halálozási aránya várhatóan 0.003 százalékponttal emelkedik. Egy fokkal magyarázásban: Ha két azonos ápolószámú és azonos 65 év feletti női népességgel bíró járás közül az egyiknek 10 ezer fővel több munkanélkülie van, akkor ott várhatóan 0.003 százalékponttal nagyobb esélye van egy COVID fertőzöttnek meghalni.

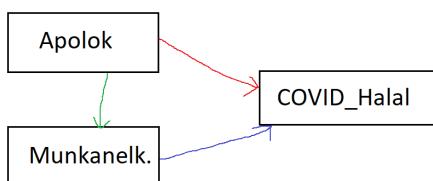
- $\beta_3 = 0.27$: Ha egy járásban a 65 év feletti női népesség aránya 1 százalékponttal nő változatlan ápolószám és munkanélküliség mellett, akkor a járás COVID halálozási aránya várhatóan 0.27 százalékponttal emelkedik. Egy fokkal magyarázatban: Ha két azonos ápolószámú és munkanélküliségű járás közül az egyik népességében 1 százalékponttal több lesz a 65 év feletti nők aránya, akkor ott várhatóan 0.27 százalékponttal nagyobb esélye van egy COVID fertőzöttnek meghalni.

Persze van még **egy β_0 tengelymetszetünk, ami hivatalosan azt mutatja meg, hogy mennyi lenne az \hat{Y} , ha a modellben minden magyarázóváltozó (X_j) 0 értéket venne fel.** Ez esetünkben azt jelenti, hogy egy 0 ápolójú, 65 év feletti nők nélküli és teljes foglalkoztatottságú járás COVID halálozási aránya a modellünk alapján -0.15% . Nyilván ezzel az értelmezéssel most nem kell foglalkozni, mivel a $\forall X_j = 0$ hely jelen témaerületen nem létezik. Ha valaki talál egy ilyen *csupa 0* járást, akkor viszont meneküljön onnan, mert jönnek a zombik :)) (negatív halálozási arány :))

13.3.1. Útlemzés

Az a tény, hogy többváltozós regresszióban a β_j együtthatók megfeleltethetők a hozzájuk tartozó X_j magyarázóváltozók **marginális hatásának**, elvezet minket az egyes X_j magyarázóváltozók közvetlen és közvetett hatásának fogalmához. Egy tetszőleges X_j magyarázóváltozó közvetlen és közvetett hatásait konkrétan, számszerűen is ki lehet fejezni a regressziós β -k segítségével.

Vegyük az eddig is vizsgált **Apolok** változó közvetlen és közvetett hatásait az alábbi ábrán szemléltetve az 1. fejezet korrelációmátrixából alapján kiindulva. Ugye a korrelációk alapján az volt az elképzelésünk, hogy **az ápolók száma csak azért lehet magas pozitív korrelációban a COVID halálozással**, mert a nagyobb munkanélküliség **jellemzően** nagyobb ápolószámmal jár együtt, és a magas munkanélküliség megnövelte a COVID halálozást, az ápolószámnak meg lehet, hogy önállóan semmi hatása nincs. Lássuk, hogy a regressziós együtthatók igazolják-e ezt az elképzelést!



Itt a *piros* nyíl jelöli az **Apolok** közvetlen hatását a **COVIDHalal**-ra, míg a *kék* nyíl a **Munkanelkuliseg** közvetlen hatását a **COVIDHalal**-ra. A *zöld* nyíl pedig az **Apolok** hatása a **Munkanelkuliseg**-re.

A piros és kék nyilakon lévő közvetlen hatások nagyságát megadja a $BecsltCOVIDHalal = \beta_1 \times Apolok + \beta_2 \times Munkanelkuliseg + \beta_0$ regresszió β_1 és β_2 együtthatója. Szóval nyejük is ki ezeket a β -kat külön R objektumokba:

```
# elõször legyártjuk a két magyarázóváltozós regressziót
X_ketvalt = covid.loc[:, ['Apolok', 'Munkanelkuliseg']]
X_ketvalt = sm.add_constant(X_ketvalt)
ketmagyvalt_modell = sm.OLS(Y, X_ketvalt).fit()

# megnézzük az együtthatókat tartalmazó tömböt
ketmagyvalt_modell.params

## const          2.326507
## Apolok        0.185685
## Munkanelkuliseg  0.002669
## dtype: float64

# a tömb 2. és 3. elemét a neveik alapján elmentjük: figyeljünk, hogy a 0. indexű elem
Beta_Apolok_COVID = ketmagyvalt_modell.params['Apolok']
Beta_Munkanelk_COVID = ketmagyvalt_modell.params['Munkanelkuliseg']
```

A zöld nyílon található hatás nagyságát pedig egyszerűen a $BecsltMunkanelkuliseg = \beta_1 \times Apolok + \beta_0$ kétváltozós regresszió β_1 együtthatója adja meg. Ezt is mentük le külön Python objektumokba:

```
# legyártjuk a két magyarázóváltozós regresszió együtthatóit, és a meredekség lesz a ".
Beta1_Apolok_Munkanelk, Beta0_Apolok_Munkanelk = np.polyfit(covid.Apolok, covid.Munkanelkuliseg, 1)
```

Ezzel pedig megadható az $Apolok \rightarrow COVIDHalal$ kapcsolat közvetlen és közvetett hatásai:

```
# közvetlen hatása az ápolók számának a COVID halálozásra (piros nyíl)
Kozvetlen_Apolok_COVID = Beta_Apolok_COVID
Kozvetlen_Apolok_COVID

## 0.18568467374999392

# közvetett hatása az ápolók számának a COVID halálozásra (zöld*kék nyíl)
Kozvetett_Apolok_COVID = Beta1_Apolok_Munkanelk * Beta_Munkanelk_COVID
Kozvetett_Apolok_COVID

## 0.17724752069302113
```

Hiszen a **közvetett** hatás csupán annyi, hogy annyiszor kell venni a *Munkanelkuliseg* → *COVIDHalal* közvetlen hatást (kék nyíl), ahányszor megváltozik a **Munkanelkuliseg** egy egység **Apolok** növekedésre (zöld nyíl). Ezt pedig éppen a **Beta_Apolok_Munkanelk** adja meg!

Ezzel pedig megadható a *Apolok* → *COVIDHalal* kapcsolat *teljes* hatása:

```
# teljes hatás = közvetlen + közvetett hatás
Teljes_Apolok_COVID = Kozvetlen_Apolok_COVID + Kozvetett_Apolok_COVID
Teljes_Apolok_COVID
```

```
## 0.36293219444301505
```

És jó, ez az érték pont ugyan az, mint *az eredeti *BecsultCOVIDHalal* = $\beta_1 \times \text{Apolok} + \beta_0$ kétváltozós regresszió β_1 meredeksége! :)

```
Beta1 # Ezt ugye még az 1. fejezetben hoztuk létre!
```

```
## 0.3629321944430193
```

Tehát, tényleg az eredeti kétváltozós regresszióban lévő **confounding** hatást tudtuk letisztítani a többváltozós regresszióval az **Apolok** változóra nézve!

Nyilván hasonló módon lehetne megnézni, hogy mennyi az *Apolok* → *COVIDHalal* kapcsolatban a **Nok65Felett** változó **közvetett hatása** miatt fellépő **confounding** hatása a *BecsultCOVIDHalal* = $\beta_1 \times \text{Apolok} + \beta_0$ kétváltozós regresszió β_1 meredekségében.

13.3.2. Parciális t-próba

A magyarázóváltozók fontosságát hipotézisvizsgálattal is meg tudjuk mérni. Egy tetszőleges X_j magyarázóváltozó fontosságát az alábbi null- alteranatív hipotézis páros tesztelésével tudjuk megállapítani:

- $H_0 : \beta_j = 0 \sim X_j$ hatása Y -ra a mintán kívül **nem szignifikáns**
- $H_1 : \beta_j \neq 0 \sim X_j$ hatása Y -ra a mintán kívül **szignifikáns**

Tehát ebben a hipotézisvizsgálatban, amit *parciális t-próbának* fogunk hívni, azt mondja a H_0 , hogy X_j hatása az eredményváltozóra csak egy mintavételi hiba, ha megfigyelnék új egyedeket (azaz új járásokat), akkor a mintában mért hatás megszűnne, azaz β_j kinullázódna.

Ehhez a hipotézisvizsgálathoz kell nekünk a Globális F-próbánál látottak alapján egy próbafüggvény és p-érték is.

A próbafüggvényhez ad nekünk a Python egy **standard mintavételei hibát, angolul standard errort**. Ez van a `summary` metódus által kigenerált együtthatótábla 2. oszlopában:

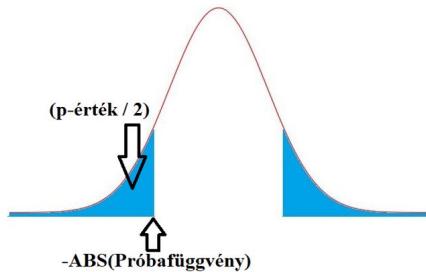
```
print(sokvalt_modell.summary())
```

```
##                                     OLS Regression Results
## =====
## Dep. Variable:                  COVIDHalal   R-squared:                 0.341
## Model:                          OLS          Adj. R-squared:            0.321
## Method:                         Least Squares   F-statistic:              16.89
## Date:                           H, 30 jún. 2025   Prob (F-statistic):       6.42e-09
## Time:                            15:25:09      Log-Likelihood:           -152.39
## No. Observations:                102         AIC:                      312.8
## Df Residuals:                   98          BIC:                      323.3
## Df Model:                        3
## Covariance Type:                nonrobust
## =====
##             coef    std err      t      P>|t|      [0.025      0.975]
## -----
## const        -0.1542     0.937   -0.165     0.870     -2.015     1.706
## Apolok        0.0445     0.096    0.463     0.644     -0.146     0.235
## Munkanelkuliseg  0.0028     0.001    4.099     0.000      0.001     0.004
## Nok65Felett   0.2656     0.087    3.053     0.003      0.093     0.438
## =====
## Omnibus:                   25.226   Durbin-Watson:            1.957
## Prob(Omnibus):               0.000   Jarque-Bera (JB):        36.179
## Skew:                       1.173   Prob(JB):                1.39e-08
## Kurtosis:                    4.735   Cond. No.                 3.10e+03
## =====
## Notes:
## [1] Standard Errors assume that the covariance matrix of the errors is correctly specified
## [2] The condition number is large, 3.1e+03. This might indicate that there are
## strong multicollinearity or other numerical problems.
```

Itt pl. a 0.096 érték azt jelenti, hogy az **Apolok** bétája a mintában 0.04, de ez az érték a megfigyelt mintán kívül, új járásokra is futtatva a regressziót, ingadozhat a 0.04 körül, *várhatóan* + 0.096-tal. A β_j és standard hibájának (SH_j) hányadosa lesz az ún. *t*-érték = *t value*, ami a parciális t-próba c. hipotéziszvizsgálat próbafüggvénye. Az **Apolok** változó esetében ez 0.463

A próbafüggvény eloszlása sok-sok mintavételel vizsgálva igaz H_0 esetén t-eloszlású, aminek a pontos alakját egy darab szabadságfok szabályozza, ami most $df = n - p = n - k - 1$, ami a Globális F-próbában a 2. szabadságfok volt.

Itt H_0 szempontjából a legjobb eset, ha $\beta_j = 0$ a mintában is, hiszen $t = \frac{\beta_j}{S\hat{H}_j}$. A t-eloszlás alakja miatt viszont így a p-értéket úgy kell számolni a 0 ponttól lefelé és felfelé vett eltérések esetén is csökkenjen a H_0 elutasításának hibavalószínűsége:



Így Pythonban a p-érték: `2*stats.t.cdf(-abs(0.0445/0.096), df = 102-4)` = 0.644. Ezt adja meg a `summary` metódus eredménytáblája a `P>|t|` oszlopan.

A `P>|t|` oszlopot nézegetve arra a kijelentésre juthatunk, hogy a COVID halálozásra a munkanélküliség és a 65 év feletti női népesség aránya is szignifikáns hatással van a *megfigyelt lakásokon kívüli világban is*, hiszen a parciális t-próba p-értéke az $\alpha = 1\%$ -os szignifikancia-szintnél is kisebb. Azaz a H_0 elvetésével mindenki magyarázóváltozó esetén elég alacsony valószínűséggel hibáznék. Ez a **Nok65Felett** változó esetében azt jelenti, hogy az a közepes-gyenge közvetlen hatása az árakra, amit a parciális korreláció (+0.29) kimutatott megmarad új járások vizsgálata esetén is. Ugyanakkor, azt is látni, hogy az **Apolok** változóhoz rendelt p-érték jóval nagyobb, mint a **Munkanelkuliseg** és **Nok65Felett** változóké. Konkrétan 64.4% a p-érték, ami olyan magas, hogy a legmagasabb szokásos α -nál, a 10%-nál is nagyobb, így stabilan elfogadhatom a próba H_0 -ját! Tehát a parciális t-próba is azt mondja, hogy az **Apolok** változók hatása a COVID halálozásra a mintén kívüli filágban **nem szignifikáns**.

Végkonklúzióként, tehát azt vonhatjuk le a modellünk alapján, hogy az **ápolók számának COVID halálozásra gyakorolt pozitív hatása a kétváltozós regresszióban látszólagos volt csupán**, és a munkanélküliség valamint **65 feletti női népesség arányának COVID halálozást növelő hatásait közvetítette csak**, és semmi önálló szignifikáns marginális hatása nem volt!

Ezek a változónkénti t-próba p-értékek azért nagyon jók, mert ezek alapján nagyon könnyű egy fontossági sorrendet felállítani a regresszió magyarázóváltozói között. Az eddigiek alapján röviden-tömören: minél kisebb a parciális t-próba p-értéke, annál fontosabb az adott magyarázóváltozó az eredményváltozó előrejelezésében (annál kevésbé vehető marginális hatása 0-nak a sokaságban)

13.4. Magyarázóváltozók fontossági sorrendjének megállapítása t-próba alapján

Vegyük elő újra a BP_Lakas.csv fájlt, és töltük be újra egy pandas data frame-be, ahogy a 12. fejezet elején is tettük!

Előtte egy kis emlékeztető az adattábla tartalmáról, változóról. A tábla 1406 budapesti lakásról 10 ismérő/változó (oszlop) adatát tárolja:

- KinArMFt: lakás ára millió Ft-ban (MFt)
- Terulet: lakás területe négyzetméterben
- Terasz: teraszok száma a lakásban
- Szoba: szobák száma a lakásban
- Felszoba: félszobák száma a lakásban
- Furdoszoba: fürdőszobák száma a lakásban
- Emelet: emeletek száma a lakásban
- DeliTaj: lakás déli fekvésű-e? (1 = igen; 0 = nem)
- Buda: lakás déli fekvésű-e? (1 = igen; 0 = nem)
- Kerulet: lakás kerülete (1 - 22)

Majd a konkrét beolvasás:

```
BP_Lakas = pd.read_csv("BP_Lakas.csv")
BP_Lakas.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 1406 entries, 0 to 1405
## Data columns (total 10 columns):
## #   Column      Non-Null Count  Dtype  
## #   --   --          --           --    
## #  0   KinArMFt    1406 non-null   float64 
## #  1   Terulet     1406 non-null   float64 
## #  2   Terasz       1406 non-null   float64 
## #  3   Szoba        1406 non-null   int64  
## #  4   Felszoba     1406 non-null   int64  
## #  5   Furdoszoba   1406 non-null   int64  
## #  6   Emelet        1406 non-null   int64  
## #  7   DeliTaj      1406 non-null   int64  
## #  8   Buda          1406 non-null   int64  
## #  9   Kerulet      1406 non-null   int64  
## dtypes: float64(3), int64(7)
## memory usage: 110.0 KB
```

Most nem szenveddünk az adattípusokkal, minden hagyunk numerikusan.

13.4. MAGYARÁZÓVÁLTOZÓK FONTOSÁGI SORRENDJÉNEK MEGÁLLAPÍTÁSA T-PRÓBA ALAPJÁN

Nézzünk most meg egy olyan modellt a lakásárak becslésére, amiben a **BP_Lakas** data frame minden változója szerepel magyarázóváltozóként a **Kerulet** kivételével:

```
# eredményváltozó megadása
Y = BP_Lakas.KinArMFT

# magyarázóváltozók megadása a konstanshoz szükséges csupa 1 oszlop hozzáadásával
X = BP_Lakas.iloc[:,1:9]
X = sm.add_constant(X)

# létrehozzuk az új regressziót
nagymodell = sm.OLS(Y, X).fit()

# lássuk az eredménytáblát
print(nagymodell.summary())

##                                     OLS Regression Results
## =====
## Dep. Variable:                 KinArMFT    R-squared:                   0.818
## Model:                          OLS        Adj. R-squared:             0.817
## Method:                         Least Squares   F-statistic:            783.3
## Date:          H, 30 jún. 2025   Prob (F-statistic):       0.00
## Time:              15:25:09      Log-Likelihood:         -4984.1
## No. Observations:            1406      AIC:                      9986.
## Df Residuals:                1397      BIC:           1.003e+04
## Df Model:                       8
## Covariance Type:            nonrobust
## =====
##            coef    std err          t      P>|t|      [0.025      0.975]
## -----
## const     -9.0189    0.779     -11.580      0.000     -10.547     -7.491
## Terulet      0.2975    0.010      29.153      0.000      0.277      0.318
## Terasz       0.3105    0.021      15.022      0.000      0.270      0.351
## Szoba        0.6898    0.349      1.977      0.048      0.005      1.374
## Felszoba     -0.3851    0.406     -0.949      0.343     -1.181      0.411
## Furdoszoba    5.1301    0.682      7.526      0.000      3.793      6.467
## Emelet        0.0545    0.132      0.412      0.680     -0.205      0.314
## DeliTaj       1.1916    0.458      2.604      0.009      0.294      2.089
## Buda          6.2197    0.475     13.091      0.000      5.288      7.152
## =====
## Omnibus:                  421.833   Durbin-Watson:            1.271
## Prob(Omnibus):            0.000    Jarque-Bera (JB):       2922.930
## Skew:                     1.211    Prob(JB):                  0.00
## Kurtosis:                  9.635   Cond. No.                  363.
```

```
## =====
## 
## Notes:
## [1] Standard Errors assume that the covariance matrix of the errors is correctly specified
```

Látjuk, hogy a modell magyarázóereje a mintában 81.76%, azaz a modellben lévő magyarázóváltozók együttesen kb. 82%-ban magyarázzák a lakásárak alakulását a megfigyelt 1406 lakás esetében. Ha a többi egyéb változó hatását is kiszűrjük, akkor a szobaszám közvetlen hatása $\alpha = 5\%$ -on még éppen szignifikáns, de $\alpha = 1\%$ -on már nem! A félszobák száma és az emeletek száma pedig egyik szokásos α -n sem szignifikáns magyarázóváltozó már a megfigyelt lakásokon túli világban.

A magyarázóváltozók fontossági sorrendjét a változók p-értéke alapján növekvő sorba rendezéssel tudom megadni, mert minél kisebb a p-érték annál kisebb az esélye, hogy az adott X_j -t nem szignifikáns magyarázóváltozónak venni hibás döntés.

Ennek a folyamata egy kicsit macerás Pythonban. Először elmentjük a `summary` függvény eredményét egy külön Python változóba, és ennek a `tables` tulajdonságának a második (azaz 1. indexű) elemét olvasom ki egy következő külön változóba `**html` kódként, azaz

tagekkel!!** Ezt utána arra tudom használni, hogy a `pandas` csomag `read_html` függvényével be tudom olvasni a regresszió nagy együtthatótábláját a p-értékekkel együtt egy `data frame`-be. Arra kell még figyelni, hogy a `html` tábla 0. indexű sora legyen a `read_html`-el beolvasott `data frame` sorindexe (`index_col` paraméter) és ugyan ezen tábla 0. indexű oszlopa pedig legyen ugyanitt az oszlopfejléc (`header` paraméter). Annyi technikai dologra kell még odafigyelni, hogy a `read_html` függvény egy egyelemű listaként adja vissza az eredményt, így ahhoz, hogy közvetlenül a `data frame`-t kapjuk vissza, ennek az eredménylistának a 0. indexű elemét külön ki kell venni [0] kódrészlettel a végén. Lássuk is hát a konkrét Python kódokat és az eredményül kapott `data frame`-et!

```
BetaTablaHTML = nagymodell.summary().tables[1].as_html()
BetaTabla_df = pd.read_html(BetaTablaHTML, header=0, index_col=0)[0]
```

```
## <string>:1: FutureWarning: Passing literal html to 'read_html' is deprecated and will be removed in a future version. Use read_html instead.
```

```
BetaTabla_df
```

	coef	std err	t	P> t	[0.025	0.975]
## const	-9.0189	0.779	-11.580	0.000	-10.547	-7.491
## Terulet	0.2975	0.010	29.153	0.000	0.277	0.318
## Terasz	0.3105	0.021	15.022	0.000	0.270	0.351

13.5. 5. KONFIDENCIA-INTERVALLUMOK ÉS A T-PRÓBA KAPCSOLATA329

```
## Szoba      0.6898   0.349   1.977   0.048   0.005   1.374
## Felszoba   -0.3851   0.406  -0.949   0.343  -1.181   0.411
## Furdoszoba 5.1301   0.682   7.526   0.000   3.793   6.467
## Emelet     0.0545   0.132   0.412   0.680  -0.205   0.314
## DeliTaj    1.1916   0.458   2.604   0.009   0.294   2.089
## Buda       6.2197   0.475  13.091   0.000   5.288   7.152
```

Ha megvan a `data frame`, akkor pedig rendezzük növekvő sorrendbe a 4. (3. indexű) oszlop, azaz a β_j együtthatók **p-értékei** szerint. Előtte a tengelymetszet (β_0) sorát, a `const`-t kihagyjuk a `data frame`-ből, mert ugyebár az nem egy magyarázóváltozó együtthatója a regressziós egyenletben.

```
BetaTabla_df.iloc[1:9,:].sort_values(by=BetaTabla_df.columns[3])
```

```
##           coef  std err      t  P>|t|  [0.025  0.975]
## Terulet    0.2975   0.010  29.153  0.000   0.277   0.318
## Terasz     0.3105   0.021  15.022  0.000   0.270   0.351
## Furdoszoba 5.1301   0.682   7.526  0.000   3.793   6.467
## Buda       6.2197   0.475  13.091  0.000   5.288   7.152
## DeliTaj   1.1916   0.458   2.604  0.009   0.294   2.089
## Szoba      0.6898   0.349   1.977  0.048   0.005   1.374
## Felszoba   -0.3851   0.406  -0.949  0.343  -1.181   0.411
## Emelet     0.0545   0.132   0.412   0.680  -0.205   0.314
```

Láthatjuk, hogy a Terület a legfontosabb magyarázóváltozó, míg a pl. szobák száma a 3. legkevésbé fontos magyarázóváltozó.

13.5. 5. Konfidencia-intervallumok és a t-próba kapcsolata

A többváltozós regressziós modell együtthatóira (béták) is lehet egy adott megbízhatósági szintű konfidencia-intervallumot is készíteni Pythonban a regressziós modellek `confint` metódusának segítségével. Pl. egy 97% megbízhatóságú konfidencia-intervallum β_j -re azt adja meg, hogy 97%-os valószínűséggel milyen értékhatárok között mozoghat β_j együttható a megfigyelt lakásokon túli világban.

Nézzük is meg a Python számítást:

```
nagymodell.conf_int() # alapból 95%-os megbízhatóságú
```

```
##          0          1
```

```

## const      -10.546693 -7.491181
## Terulet     0.277475  0.317511
## Terasz      0.269945  0.351035
## Szoba       0.005311  1.374306
## Felszoba    -1.180769  0.410579
## Furdoszoba  3.792974  6.467229
## Emelet      -0.204826  0.313728
## DeliTaj     0.293935  2.089167
## Buda        5.287680  7.151671

```

```
nagymodell.conf_int(alpha = 0.03) # de lehet pl. 97%-os is (alfa = 1-0.97)
```

```

##                  0          1
## const      -10.710745 -7.327129
## Terulet     0.275325  0.319660
## Terasz      0.265591  0.355389
## Szoba       -0.068191  1.447808
## Felszoba    -1.266210  0.496020
## Furdoszoba  3.649391  6.610811
## Emelet      -0.232668  0.341570
## DeliTaj     0.197548  2.185555
## Buda        5.187601  7.251750

```

Pl. az első eredménytábla alapján a **Terulet** változó β -ja 95%-os valószínűséggel 0.277 és 0.317 között kötne ki valahol, ha nem 1406 budapesti lakást vizsgálnánk, hanem az össeset (azaz a lakások sokaságát).

Az eredményből megjelenik egy olyan összefüggés is, miszerint, **ha egy X_j nem szignifikáns α szignifikancia-szinten, akkor az $1 - \alpha$ megbízhatóságú konfidencia-intervallumának határai előjelet váltanak.** legjobb példa a **Szoba** változó esete: azt mondta rá, hogy hatása az árakra 5%-on még épp szignifikáns -> a 95%-os megbízhatóságú intervalluma még épp nem vált előjelet. Viszont a változó 3%-on már nem szignifikáns, így az intervalluma előjelet vált.

Ezzel egy érdekes nézőponthoz jutottunk: **ha egy X_j változó nem szignifikáns α -n az azt jelenti, hogy $1 - \alpha$ megbízhatósági szinten azt sem tudom eldöntenni, hogy minden más változatlansága mellett +1 egység X_j növeli vagy csökkenti-e az eredmányváltozót a mintán kívüli világban.**

A háttérben amúgy annyi történik, hogy a Python kiszámolja a parciális t-próba esetén is alkalmazott $n - p = n - k - 1$ szabadságfokú t-eloszlás inverz értékét $1 - \alpha/2$ valószínűség mellett, és az ezzel beszorzott standard hibát adja hozzá és vonja le $\forall \beta_j$ -ből:

```
BetaTabla_df
```

13.6. NOMINÁLIS MAGYARÁZÓVÁLTOZÓK - HASZNÁLTAUTÓ ADATOK331

```

##          coef  std err      t  P>|t| [0.025  0.975]
## const     -9.0189   0.779 -11.580  0.000 -10.547 -7.491
## Terulet    0.2975   0.010  29.153  0.000   0.277  0.318
## Terasz     0.3105   0.021  15.022  0.000   0.270  0.351
## Szoba      0.6898   0.349   1.977  0.048   0.005  1.374
## Felszoba   -0.3851   0.406  -0.949  0.343  -1.181  0.411
## Furdoszoba 5.1301   0.682   7.526  0.000   3.793  6.467
## Emelet      0.0545   0.132   0.412  0.680  -0.205  0.314
## DeliTaj    1.1916   0.458   2.604  0.009   0.294  2.089
## Buda       6.2197   0.475  13.091  0.000   5.288  7.152

alfa = 0.02
p = len(BetaTabla_df) # így a tengelymetszetet is beszámítjuk p-be
n = len(BP_Lakas) # mintaelemszám
szorzo = stats.t.ppf(1-alfa/2, df = n-p)

Terulet_AH = BetaTabla_df.iloc[1,0] - szorzo * BetaTabla_df.iloc[1,1] # Terulet 98% alsó határ
Terulet_FH = BetaTabla_df.iloc[1,0] + szorzo * BetaTabla_df.iloc[1,1] # Terulet 98% felső határ
print([Terulet_AH, Terulet_FH])

## [0.2742097984822436,  0.3207902015177564]

```

Formális képlettel leírva a műveletet:

$$\hat{\beta}_i \pm t_{n-(k+1)}^{(1-\alpha/2)} \cdot \text{se}(\hat{\beta}_i)$$

13.6. Nominális magyarázóváltozók - Használtautó adatok

Most egy új, auto.xlsx fájlból fogunk dolgozni, ami 100 db használtautóról tartalmaz 8 változót:

- Vetelár: Vételár Ft-ban
- Kor: Az autó kora években
- Kilometerara: A kilóméteréra állása
- Teljes_tomeg: Az autó tömege kg-ban
- Hengerurtartalom: Hengerűrtartalom *ccm*-ben (köbcenti)
- Teljesitmeny: A motor teljesítménye Kw/h-ban
- Allapot: Az autó állapota szöveges minősítéssel
- Uzemanyag: Üzemanyag típusa (benzin/dízel)

Láthatjuk, hogy ez nem *csv*, hanem Excel, azaz *xlsx* fájl. Úgyhogy olvassuk is be egy pandas `data frame`-be!

```
autok = pd.read_excel("auto.xlsx")
autok.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 100 entries, 0 to 99
## Data columns (total 8 columns):
##   #   Column      Non-Null Count  Dtype  
##   --  --          -----          ----- 
##   0   Vetelar     100 non-null    int64  
##   1   Kor         100 non-null    int64  
##   2   Kilometerora 100 non-null    int64  
##   3   Teljes_tomeg 100 non-null    int64  
##   4   Hengerurtartalom 100 non-null    int64  
##   5   Teljesitmeny 100 non-null    int64  
##   6   Allapot     100 non-null    object 
##   7   Uzemanyag    100 non-null    object 
## dtypes: int64(6), object(2)
## memory usage: 6.4+ KB
```

Minden szupinak néz ki! Megvan mind a 100 megfigyelési egység és a 8 db változó.

Vessünk még egy pillantást arra, hogy az **Allapot** változónak milyen lehetséges értékei vannak:

```
autok.Allapot.unique()
```

```
## array(['Normal', 'Kituno', 'Megkimelt', 'Serulesmentes', 'Ujszeru'],
##       dtype=object)
```

13.7. Dummy változók és működésük Pythonban

No, ha egy sima OLS regresszióba magyarázóváltozóként nominális változót akarunk bevenni, mint amilyen jelen példánkban az **Allapot** vagy az **Uzemanyag**, akkor az R az úgynevezett **dummy-kódolást** veti be. A nominális változó lehetséges értékeiből kiválaszt egyet, mint **referencia kategória**, azt „*elteszi*”, és a maradéknak pedig csinál külön-külön egy *bináris = dummy* változót, aminek értéke 1, ha a megfigyelés a vizsgált szöveges változóban épp a dummy változónak megfelelő értéket vesz fel, egyébként 0:

	D_A	D_B	D_C	R_A	R_B
A	1	0	0	1	0
B	0	1	0	0	1
C	0	0	1	0	0

A fenti táblázatban az R_A és R_B lennének a regresszióban használt dummy változók, és a referencia kategória a C érték. Ezt a C kategóriát azért nem szabad a modellben szerepeltetni, mert egyértelmű, hogy ha $D_A = 0$ és $D_B = 0$, akkor $D_C = 1$. Ezzel egy teljesen redundáns magyarázóváltozót rakkának a modellbe, amitől az egész rendszer megdöglök, mert a β -k becslése során alkalmazott $\beta = (X^T X)^{-1} X^T y$ képletben az $(X^T X)^{-1}$ inverzet nem lehet elvégezni, ha X mátrix oszlopai olyanok, hogy az egyiket hibátlanul elő lehet állítani a többiből. Ez a jelenség az **egzakt multikollinearitás** nevet kapta a keresztségen.

Ezekkel a megfontolásokkal lássunk is neki a dummy változóink legyártásához a két nominális változónk (**Uzemanyag** és **Allapot**) esetében! Először a **pandas** csomag `get_dummies` függvényével létrehozzuk a két szöveges változó minden kategóriájához a megfelelő 0/1, azaz dummy változókat. A függvény első paramtere az érintett változókat tartalmazó `data frame`, a második (`columns`) azon oszlopnevek listája, amelyekből dummy változókat akarunk csinálni, még az utolsó, `dtype` paraméterben a dummy változók adattípusát lehet megadni. Ez utolsó paraméter azért szükséges, mert egy dummy változó logikusan lehet akár `bool` adattípusú, de ad abszurdum `int` is. Most mi ez utóbbi opciónal megyünk, mert a **statsmodels** csomag nem tud `bool` típusú változókkal dolgozni.

```
autok_dummy = pd.get_dummies(autok, columns=['Uzemanyag', 'Allapot'], dtype=int)
autok_dummy.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 100 entries, 0 to 99
## Data columns (total 13 columns):
## #   Column           Non-Null Count  Dtype  
## ---  -- 
## #   0   Vetelar        100 non-null   int64 
## #   1   Kor            100 non-null   int64 
## #   2   Kilometerora  100 non-null   int64 
## #   3   Teljes_tomeg   100 non-null   int64 
## #   4   Hengerurtartalom 100 non-null   int64 
## #   5   Teljesitmeny  100 non-null   int64 
## #   6   Uzemanyag_Benzin 100 non-null   int32 
## #   7   Uzemanyag_Dizel 100 non-null   int32 
## #   8   Allapot_Kituno  100 non-null   int32 
## #   9   Allapot_Megkimelt 100 non-null   int32
```

```
## 10 Allapot_Normal      100 non-null    int32
## 11 Allapot_Serulesmentes 100 non-null    int32
## 12 Allapot_Ujszeru     100 non-null    int32
## dtypes: int32(7), int64(6)
## memory usage: 7.6 KB
```

Szuper, megvan minden szükséges új változó, és a megfelelő, azaz `int` adattípuson is vannak!

Ezek után lássunk hozzá a regressziós modell legyártásához, amely az autók vételárát magyarázza az összes többi változóval a táblában. Arra kell figyelni, hogy a magyarázóváltozók (X mátrix) kijelölésénél ne válasszuk ki a két nominális változó refrencia kategóriáinak oszlopait. Legyen most az **Uzemanyag** változó esetén a kihagyott refrencia kategória a *Benzin* (6. oszlopindex), míg **Allapot** esetén a *Kituno* (8. oszlopindex).

```
Y = autok_dummy.Vetelar
X = autok_dummy.iloc[:, [1,2,3,4,5,7,9,10,11,12]] # megfelelő oszlopindexek kiválasztása
X = sm.add_constant(X)

# létrehozzuk az új regressziót
autos_modell = sm.OLS(Y, X).fit()

# lássuk az eredménytáblát
print(autos_modell.summary())

##                                     OLS Regression Results
## -----
## Dep. Variable:                  Vetelar   R-squared:           0.728
## Model:                          OLS      Adj. R-squared:        0.697
## Method:                         Least Squares   F-statistic:         23.81
## Date:                H, 30 jún. 2025   Prob (F-statistic):  4.10e-21
## Time:                    15:25:10    Log-Likelihood:   -1345.2
## No. Observations:             100      AIC:                 2712.
## Df Residuals:                  89      BIC:                 2741.
## Df Model:                      10
## Covariance Type:            nonrobust
## -----
##                   coef  std err      t    P>|t|  [0.025
## -----
## const          2.323e+06  6.15e+05   3.777   0.000  1.1e+06  3.5
## Kor            -7.273e+04  9353.425  -7.776   0.000 -9.13e+04 -5.4
## Kilometerora   -2.4542    0.362   -6.784   0.000  -3.173   -
## Teljes_tomeg    -164.4405  371.805  -0.442   0.659  -903.210  5
## Hengerurtartalom  42.0239   160.600   0.262   0.794  -277.084  3
## Teljesitmeny   4647.0879  1600.722   2.903   0.005  1466.488  782
```

```

## Uzemanyag_Dizel      -3.763e+04   4.67e+04    -0.806    0.422   -1.3e+05   5.51e+04
## Allapot_Megkimelt   4.186e+04   5.08e+04     0.824    0.412   -5.91e+04   1.43e+05
## Allapot_Normal       -1.192e+05  4.95e+04    -2.411    0.018   -2.18e+05  -2.1e+04
## Allapot_Serulesmentes -9418.8363  1.13e+05    -0.084    0.934   -2.33e+05  2.15e+05
## Allapot_Ujszeru      2.266e+05   1.14e+05     1.980    0.051   -742.588   4.54e+05
## =====
## Omnibus:              8.736   Durbin-Watson:          1.598
## Prob(Omnibus):        0.013   Jarque-Bera (JB):       8.434
## Skew:                  -0.646   Prob(JB):            0.0147
## Kurtosis:              3.597   Cond. No.           6.58e+06
## =====
## Notes:
## [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
## [2] The condition number is large, 6.58e+06. This might indicate that there are
## strong multicollinearity or other numerical problems.

```

Úgy néz ki rendben vagyunk, a magyarázóváltozók együttesen 72.8%-ban magyraázzák az autók vételárának alakulását.

Nézzük meg a nominális magyarázóváltozókhöz köthető dummy változók β_j együtthatónak értelmezését! Az eddigi okoskodásaink alapján alapján az egyes együtthatók értelmezése mindig a **referencia kategóriához** képest értedő:

- $\beta_{Dizel} = -37630$: minden más magyarázóváltozó változatlansága mellett egy dízelautó várhatóan 37630 Ft-al lesz olcsóbb, mint egy *benzines*.
- $\beta_{Normal} = -119200$: minden más magyarázóváltozó változatlansága mellett egy normál állapotú autó várhatóan kb. 119 200 Ft-tal lesz olcsóbb, mint egy *kitűnő* állapotban lévő.

13.7.1. Referencia kategória megváltoztatása

Az előző modellben nem feltétlenül a legszerencsesebb az, ha a *Kituno* a referencia kategória az **Allapot** változóban. Logikusabb lenne a *Normal* állapotot referenciának venni. A módosítást simán meg tudjuk tenni. Egyszerűen az X mátrixból a 10. oszlopindexet hagyom ki a 8. helyett, és így újrafuttatom a modellt.

Alapvetően a `levels` függvény eredményéből látható, hogy a referencia kategória az **Allapot** változóban azért a volt a *Kituno*, mert az van betűrend szerint a legelöl az **Allapot** változó lehetséges értékei közül:

```

X = autok_dummy.iloc[:, [1,2,3,4,5,7,8,9,11,12]] # megfelelő oszlopindexek kiválasztása
X = sm.add_constant(X)

```

```

# újra lefuttatjuk a regressziót
autos_modell = sm.OLS(Y, X).fit()

# lássuk az eredménytáblát
print(autos_modell.summary())

```

OLS Regression Results							
##	Dep. Variable:	Vetelar	R-squared:	0.728	##		
##	Model:	OLS	Adj. R-squared:	0.697	=====		
##	Method:	Least Squares	F-statistic:	23.81			
##	Date:	H, 30 jún. 2025	Prob (F-statistic):	4.10e-21			
##	Time:	15:25:10	Log-Likelihood:	-1345.2			
##	No. Observations:	100	AIC:	2712.			
##	Df Residuals:	89	BIC:	2741.			
##	Df Model:	10					
##	Covariance Type:	nonrobust					
##	coef	std err	t	P> t	[0.025	(
##	-----						
##	const	2.204e+06	6.08e+05	3.626	0.000	9.96e+05	3.4
##	Kor	-7.273e+04	9353.425	-7.776	0.000	-9.13e+04	-5.4
##	Kilometerora	-2.4542	0.362	-6.784	0.000	-3.173	-
##	Teljes_tomeg	-164.4405	371.805	-0.442	0.659	-903.210	57
##	Hengerurtartalom	42.0239	160.600	0.262	0.794	-277.084	30
##	Teljesitmeny	4647.0879	1600.722	2.903	0.005	1466.488	78
##	Uzemanyag_Dizel	-3.763e+04	4.67e+04	-0.806	0.422	-1.3e+05	5.4
##	Allapot_Kituno	1.192e+05	4.95e+04	2.411	0.018	2.1e+04	2.1
##	Allapot_Megkimelt	1.611e+05	4.35e+04	3.704	0.000	7.47e+04	2.4
##	Allapot_Serulesmentes	1.098e+05	1.09e+05	1.007	0.317	-1.07e+05	3.1
##	Allapot_Ujszeru	3.458e+05	1.15e+05	3.015	0.003	1.18e+05	5.7
##	=====						
##	Omnibus:	8.736	Durbin-Watson:	1.598			
##	Prob(Omnibus):	0.013	Jarque-Bera (JB):	8.434			
##	Skew:	-0.646	Prob(JB):	0.0147			
##	Kurtosis:	3.597	Cond. No.	6.50e+06			
##	=====						
##	Notes:						
##	[1] Standard Errors assume that the covariance matrix of the errors is correctly specified						
##	[2] The condition number is large, 6.5e+06. This might indicate that there are						
##	strong multicollinearity or other numerical problems.						

Így már azt mondhatom, hogy egy kitűnő állapotban lévő autó várhatóan $\beta_{Kituno} = +119200$ Ft-tal kerül többé egy normál állapotú autónál minden

egyéb változó változatlansága mellett. Teljesen logikus módon abszolút értékben ugyan akkora a hatás az árakra, mint az előző modellben, ahol a *Kituno* volt a referencia kategória és $\beta_{Normal} = -119200$ -t vizsgáltuk, csak ellentétes előjelű.

13.8. A korrigált R-négyzet mutató

Az előző fejezet végi modell eredménytáblázatát elnézegetve azt láthatjuk, hogy a modell két legkevésbé fontos, azaz legmagasabb p-értékű magyarázóváltozója a **Hengerurtartalom** és a **Teljes_tomeg**. Ha ezeknek a változóknak tényleg nincs közük az autók vételárához, akkor egy logikus gondolat, hogy potyogtassuk is ki őket a modellünkön! Ha emlékszünk még az **autok_dummy** táblára, akkor tudhatjuk, hogy ez a két változó rendre a 4. és 3. oszlopindexen futott, szóval csak ezeket az oszlopokat kell kiszedni az X mátrixból, és már meg is szabadultunk ezektől a azvaró tényezőktől.

```
X_szuk = autok_dummy.iloc[:, [1, 2, 5, 7, 8, 9, 11, 12]] # megfelelő oszlopindexek kiválasztása
X_szuk = sm.add_constant(X_szuk)

# újra lefuttatjuk a regressziót
autos_modell_szuk = sm.OLS(Y, X_szuk).fit()

# lássuk az eredménytáblát
print(autos_modell_szuk.summary())

##                                     OLS Regression Results
## -----
## Dep. Variable:                 Vetelar   R-squared:                  0.727
## Model:                          OLS      Adj. R-squared:               0.703
## Method:                         Least Squares   F-statistic:                 30.33
## Date:                H, 30 jún. 2025   Prob (F-statistic):        1.50e-22
## Time:                    15:25:11    Log-Likelihood:            -1345.4
## No. Observations:                  100    AIC:                      2709.
## Df Residuals:                      91    BIC:                      2732.
## Df Model:                           8
## Covariance Type:            nonrobust
## -----
##                  coef    std err          t      P>|t|      [0.025      0.975]
## -----
## const            1.968e+06   1.79e+05    10.982      0.000    1.61e+06    2.32e+06
## Kor             -7.279e+04   9027.563     -8.063      0.000   -9.07e+04   -5.49e+04
## Kilometerorra      -2.4477     0.353     -6.942      0.000     -3.148     -1.747
## Teljesitmeny       4754.5406   1086.245      4.377      0.000    2596.848    6912.234
## Uzemanyag_Dizel     -3.822e+04   4.02e+04     -0.952      0.344   -1.18e+05    4.15e+04
```

```

## Allapot_Kituno      1.164e+05   4.83e+04    2.408     0.018   2.04e+04   2.1
## Allapot_Megkimelt  1.619e+05   4.3e+04     3.762     0.000   7.64e+04   2.4
## Allapot_Serulesmentes 1.031e+05   1.07e+05     0.964     0.338  -1.09e+05   3.1
## Allapot_Ujszeru     3.45e+05   1.13e+05     3.055     0.003   1.21e+05   5.1
## =====
## Omnibus:             8.420 Durbin-Watson:           1.624
## Prob(Omnibus):       0.015 Jarque-Bera (JB):        8.060
## Skew:                 -0.639 Prob(JB):            0.0178
## Kurtosis:              3.550 Cond. No.          1.98e+06
## =====
## 
## Notes:
## [1] Standard Errors assume that the covariance matrix of the errors is correctly specified
## [2] The condition number is large, 1.98e+06. This might indicate that there are
## strong multicollinearity or other numerical problems.

```

Első ránézésre minden rendben, de nézzük csak össze ennek a szűkített modellnek az R^2 értékét az eredeti modell R^2 értékével, ami még tartalmazta a hengerűrtartalmat és a teljes tömeget! Ezt legkönyebbén úgy tudjuk megkérdezni, hogy minden modellnek lekérdezzük az `rsquared` tulajdonságát.

```
[autos_modell.rsquared, autos_modell_szuk.rsquared]
```

```
## [0.7279480731232686, 0.7272297252691152]
```

Ajjaj! A modellünk magyarázóereje leromlott 72.72%-ra az eredeti 72.79%-ról! De hát azért elégé nem szignifikánsnak kiméző változókat szedtünk ki! A legalacsonyabb t-próba p-érték 65.9% volt a kihagyott magyarázóváltozók között (**Teljes_tomeg**)! Miért csökken az R^2 ?

Sajnos a jó öreg R^2 nem fog soha javulni akkor, ha a modellből elhagyunk egy magyarázóváltozót. Legyen az bármennyire is haszontalan.

Miért is? Ugye $R^2 = 1 - \frac{SSE}{SST}$ és hát SSE minimalizálásával szüljük meg a β_j együtthatókat. Konkrétan úgy választjük meg az értéket, hogy a MEGFIGYELT y értékekre a lehető legjobban illeszkedjen a regresszióból nyert \hat{y} becslés!

Ezt az illeszkedést pedig egy újabb magyarázóváltozó bevonása biztosan NEM RONTJA el, legyen bármennyire irreleváns y szempontjából. Hiszen eggyel több β_j van arra, hogy minimalizáljuk a modellhibát (SSE). De ha kilépünk a megfigyeléseinken túli világba (sokaságba), akkor már az \hat{y} becslések jóságát elrontják ezek az irreleváns magyarázóváltozók a modellben. Ez a jelenség a **túltanítás** jelensége!

Nézzük ezt meg az alábbi képen:



A pontokra illesztett görbék mögött a következő egyenletek vannak:

- Lineáris = $5.13x - 0.46$
- Harmadfokú = $2.48x^3 + 3.17x^2 - 1.06x + 1.08$
- Hetedfokú = $-7426x^7 + 28047x^6 - 42886x^5 + 33991x^4 - 14814x^3 + 3457x^2 - 380x + 14$

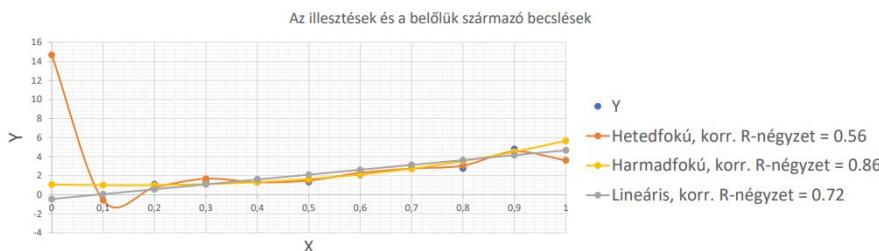
Az ábráról láthatjuk, hogy a pontokra a legreálisabb illeszkedést a 3 db magyarázó változót (x^3, x^2, x) használó *Harmadfokú* görbe adja. De az R^2 mégis a legtöbb magyarázó változót használó Hetedfokú opciót részesíti előnyben a *túltanulás* miatt!

Ennek kiküszöbölésére bevezetjük a korrigált R^2 mutatót:

$$\bar{R}^2 = 1 - (1 - R^2) \frac{n - 1}{n - k - 1}$$

A korrekciós képlet lényege, hogy ez a mutató csökkenhet is, ha irreleváns (erősen nem szignifikáns) magyarázó változókat vonunk be a modellbe. Ezért van a képletben a megfigyelések száma (n) mellett a magyarázó változók száma, k is.

Láthatjuk, hogy a korábbi példánkban \bar{R}^2 már a harmadfokú modellt preferálja, nagyon helyesen!



A \bar{R}^2 mutató értékét megtaláljuk a sima `summary` függvény eredménytáblájában is, de praktikusabb lekérdezni, mint a regressziós modellek `rsquared_adj` tulajdonsága.

```
[autos_modell.rsquared_adj, autos_modell_szuk.rsquared_adj]
```

```
## [0.697380440889928, 0.7032499208971693]
```

Hurrá! Láthatjuk, hogy a korrigált R-négyszet már heylesen azt mutatja, hogy a nem szignifikáns változók kihagyásával kapott szűkített modell magyarázóereje nagyobb (70.3%), mint az eredeti modellé (69.7%), amiben ez a két haszontalan változó is benne volt.