A könyvről

A *Tantusz Könyvek – Adatelemzés Pythonnal* célja elérni, hogy az adattudomány már ne legyen olyan ijesztő, annak bemutatásával, hogy az adatfeldolgozás módszerei nemcsak nagyon érdekesek, de a Pythonnal könnyen kivitelezhetők is. Lehet, hogy azt feltételezed, hogy informatikai zseninek kell lenned azoknak a bonyolult feladatoknak az elvégzéséhez, melyek általában az ember eszébe jutnak az adattudományról, de ez távol áll az igazságtól. A Pythonnak egy csomó olyan hasznos könyvtára van, amely a háttérben elvégzi helyetted a munka nehezét. Fogalmad sincs róla, hogy mi minden történik, és nem is kell ezzel foglalkoznod. Igazából csak annyit kell tudnod, hogy milyen feladatokat szeretnél elvégezni, és a Python könnyen hozzáférhetővé teszi számodra a feladat megoldását.

Ez a könyv többek között kiemelten foglakozik azzal, hogy a megfelelő eszközöket használd. A Jupyter Notebookkal (asztali rendszereken) vagy a Google Colabbal (a weben) láthatsz munkához – ez a két eszköz megkönnyíti a munkát a Pythonnal. A kód, amit a Jupyter Notebookban vagy a Google Colabban elhelyezel, prezentációs minőségű eredményt ad! Dokumentumodba közvetlenül behelyezheted a különböző prezentációs elemeket. Ez a megoldás más, mintha egy hagyományos fejlesztői környezetet használnál.

Ebben a könyvben néhány érdekes technikával is megismerkedhetsz. Például, hogy hogyan tudod diagramokon ábrázolni az összes adatfeldolgozási próbálkozásodat a Matplotlib használatával. Sokat foglalkozunk az elérhető erőforrások (például csomagok) bemutatásával és annak a leírásával is, hogy hogyan végezhetsz el néhány nagyon izgalmas számítást a Scikit-learn könyvtárral. Sokan szeretnék tudni, hogyan oldhatják meg a kézírás felismerését, és ha te is közéjük tartozol, ehhez a folyamathoz is kaphatsz egy kis segítséget ebben a könyvben.

Természetesen előfordulhat, hogy még mindig aggódsz az egész fejlesztőkörnyezetes ügy miatt, de ez a könyv ezzel kapcsolatban sem hagy a sötétben tapogatózni. Mindjárt az elején komplett módszereket találhatsz, amelyekre szükséged lesz ahhoz, hogy hozzálásss az adatfeldolgozási módszerek használatához a Jupyter Notebookban vagy a Google Colabban. A hangsúlyt arra helyeztük, hogy a lehető leggyorsabban hozzá tudj fogni, a példák pedig egyértelműek és egyszerűek legyenek, hogy a kód ne akadályozzon a tanulásban.

A könyv friss példákat tartalmaz, melyek a Python 3.x verzióján alapulnak, hogy olvasás közben a Python legmodernebb verzióját használd. Emellett hangsúlyosan figyeltünk rá, hogy a példák minél egyszerűbbek legyenek, de a mély tanulással kapcsolatos részek is megtalálhatók, hogy minél tágabb célközönség találja meg a számítását. Ennél is fontosabb, hogy a könyv friss adathalmazokat tartalmaz, hogy a lehető leg-

jobban tudjuk bemutatni, hogyan működnek az adattudományos módszerek napjainkban. A könyv olyan „napi” problémákkal is foglalkozik, mint például a személyazonosításra alkalmas adatok eltávolítása és az adatbiztonság növelése. A könyv igyekszik a lehető legtöbbet nyújtani az olvasóktól jött igények alapján.

A következő jelöléseket alkalmazzuk a könyvben, hogy minden egyértelmű és könnyen követhető legyen:

- Azok a szövegek, amelyeket ugyanúgy kell begépelned, ahogy a könyvben is szerepelnek, **`félkövérrel`** vannak szedve. A kivétel ez alól, amikor lépések egy listáján kell végighaladnod: mivel a lépések félkövérrel vannak szedve, itt a szöveg, amelyet be kell írnod, nem félkövér.
- Ha *`dőlt`* betűkkel szedett szavakat látsz egy szövegben, amelyet be kell gépelned, ezeket az értékeket olyasmire kell cserélned, ami a te esetedben megfelelő. Ha például azt látod, hogy „Írd be a *`Neved`*, majd nyomd meg az Entert”, a *`Neved`* helyére a tényleges nevedet kell beírnod.
- A webcímek és a programkódok `fix szélességű betűtípussal` vannak szedve, például így: `https://panem.hu/720-tantusz-konyvek`. Itt jegyezzük meg, hogy a hivatkozott weboldalak szinte mindig angol nyelvűek. A programkódokban a változóknak magyar neveket választunk, és a szöveges sztringeket is magyarul fogalmazzuk meg.
- Ha parancsok egy sorozatát kell begépelned, azokat egy speciális nyíllal elválasztva láthatod, valahogy így: Fájl ➪ Új fájl. Ebben a példában először meg kell nyitnod a Fájl menüt, majd ebben a menüben az Új fájl menüpontot kell választanod.

Feltételezéseink rólad

Nehéz lehet elhinni, hogy bármilyen feltételezésünk is van rólad – elvégre még nem is találkoztunk! Bár a legtöbb feltételezés valóban alaptalan, azért élünk ezekkel, hogy legyen miből kiindulnunk a könyvhöz.

Ismerned kell a platformot, amelyet használni szeretnél, mert a könyv ezzel kapcsolatban nem nyújt útmutatást. (Ugyanakkor a 3. fejezetben találsz segítséget az Anaconda telepítéséhez, amely támogatja a Jupyter Notebookot, a 4. fejezetben pedig a Google Colab használatának első lépéseit olvashatod.) Annak érdekében, hogy a lehető legtöbb információt nyújtsuk arról, hogy a Python hogyan használható adatfeldolgozási módszerekhez, ebben a könyvben nem foglalkozunk az adott platformok sajátosságaival. Mindenképpen tisztában kell lenned vele, hogyan kell alkalmazásokat telepíteni, alkalmazásokat és általánosságban az általad választott platformot használni.

Kezeljünk relációs adatbázisokból származó adatokat

Az adatbázisoknak rengeteg különböző formája létezik. Az AskSam (`http://asksam.en.softonic.com/`) például egyfajta szabad szöveges adatbázis. A szervezeteknél használt adatok túlnyomó többségéhez azonban relációs adatbázisokat alkalmaznak, mivel ezek az adatbázisok biztosítják az eszközöket hatalmas mennyiségű összetett adat szervezett struktúrába rendezéséhez, ami megkönnyíti az adatok kezelését. Az adatbázis-kezelő célja, hogy az adatokat könnyen kezelhetővé tegye. A legtöbb adattároló célja, hogy az adatok könnyen lekérhetők legyenek.

FONTOS

A relációs adatbázisok viszonylag könnyen teljesítik a kezelési és az adatlekérési célokat is. Mivel azonban a számítási platformok széles skáláján rendkívül sokféle és különböző méretű adattárolási igények jelentkeznek, számos különböző relációs adatbázis-kezelő eszköz létezik. Ami azt illeti, a változatos adatelrendezéseket használó, különböző adatbázis-kezelő rendszerek (Database Management System, röviden DBMS) elterjedése az egyik fő probléma, amellyel az adatelemzők találkoznak, amikor létrehozzák az elemzéshez szükséges, mindenre kiterjedő adathalmazt.

A relációs adatbázisokban az a közös, hogy mind ugyanannak a nyelvnek egy formáját használják az adatok kezeléséhez, ami megkönnyíti az adatelemzők munkáját. Az SQL (Structured Query Language, magyarul strukturált lekérdezési nyelv) használatával mindenféle kezelési feladatot elvégezhetsz egy relációs adatbázisban, szükség szerint lekérhetsz adatokat, sőt, meghatározott formába is alakíthatod őket, hogy ne legyen szükség további formázásra.

Egy adatbázissal kapcsolatot kiépíteni bonyolult feladat lehet. Először is tudnod kell, hogyan kell csatlakozni az adott adatbázishoz. A folyamatot azonban kisebb részekre oszthatod. Az első lépés az, hogy hozzáférést szerezz az adatbázismotorhoz. Ehhez két kódsort kell használnod, melyek az alábbi kódhoz hasonlóak (de az itt bemutatott kód nem egy feladat végrehajtására és elvégzésére szolgál):

```
from sqlalchemy import create_engine
motor = create_engine('sqlite:///:memory:')
```

Ha már hozzáférésed van egy motorhoz, annak használatával az adott DBMS-re vonatkozó feladatokat végezhetsz. Az olvasási metódus kimenete mindig egy `DataFrame` (Adatkeret) objektum lesz, amely a kért adatokat tartalmazza. Adatok írásához létre kell hoznod egy `DataFrame` objektumot, vagy egy meglévő `DataFrame` objektumot kell használnod.

By: John Paul Mueller and Luca Massaron
Updated: 10-03-2023

Tantusz Könyvek – Adatelemzés Pythonnal rövid összefoglaló

A Python egy hihetetlen programozási nyelv, amellyel minimális erőfeszítéssel végezhetsz el adattudományos feladatokat. Mivel hatalmas mennyiségű könyvtárhoz férhetsz hozzá, az alacsony szintű kód, amelyet általában meg kell írnod, valószínűleg már elérhető valamilyen más forrásból. Neked csak arra kell koncentrálnod, hogy elvégezd a feladatot. Ezt szem előtt tartva ebben a rövid összefoglalóban azokat az emlékeztetőket találod meg, amelyek a leggyakrabban szükségesek ahhoz, hogy gyors és könnyű programozási élményben legyen részed.

A nyolc leggyakoribb Python-programozási hiba

A világ összes programozója követ el hibákat. Viszont ha ismered a gyakori hibákat, azzal később időt és erőfeszítést takaríthatsz meg. Az alábbi listában megtalálod a leggyakoribb hibákat, amelyekkel a fejlesztők találkoznak, amikor Pythonban dolgoznak.

* **Helytelen behúzás használata:** Számos Python-funkcióhoz kell behúzást alkalmazni. Például, amikor új osztályt hozol létre, az osztály teljes tartalma behúzással szerepel az osztály deklarációja alatt. Ugyanez igaz a döntési, a ciklusszervező és más strukturált utasításokra is. Ha azt tapasztalod, hogy a kódod akkor is végrehajt egy feladatot, amikor valójában nem kellene, kezdd el átnézni az alkalmazott behúzásokat.
* **Az értékadó operátor használata az egyenlőségi operátor helyett:** Amikor összehasonlítasz két objektumot vagy értéket, az egyenlőségi operátort használd (==), és ne az értékadó operátort (=). Az értékadó operátor egy objektumot vagy értéket helyez el egy változóban, és nem hasonlít össze semmit.
* **A függvényhívások helytelen sorrendben történő elhelyezése összetett utasítások létrehozásakor:** A Python mindig balról jobbra haladva hajtja végre a függvényeket. Így a `Sztringem.strip().center(21, "*")` utasítás más eredményt ad, mint a `Sztringem.center(21, "*").strip()` utasítás. Ha olyan helyzetbe kerülsz, amikor egy sor egymás után fűzött függvény kimenete nem az, amit vártál, akkor ellenőrizned kell a függvények sorrendjét, és meg kell bizonyosodnod arról, hogy minden függvények a megfelelő helyen szerepel.
* **Rosszul elhelyezett írásjelek:** Előfordulhat, hogy rossz helyre teszel egy írásjelet, és így teljesen más eredményt kapsz. Ne feledd, hogy minden strukturált utasítás végén kettőspontnak kell állnia. Ezenkívül a zárójelek elhelyezése is kritikus kérdés. Például az `(1 + 2) * (3 + 4)`, az `1 + ((2 * 3) + 4)` és az `1 + (2 * (3 + 4))` képlet mind más és más eredményt ad.
* **Helytelen logikai operátor használata:** A legtöbb operátor nem okoz gondot a fejlesztőknek, a logikai operátorok viszont igen. Fontos megjegyezni, hogy az `and` (és) operátort használd akkor, ha mindkét operandusnak `True` (Igaz) értékűnek kell lennie, és az `or` (vagy) operátort akkor, ha elég az egyik operandusnak `True` értékűnek lennie.
* **Hibák tartományok és szeletek megadásánál:** Ne feledd, hogy a tartományok és a szeletek nem tartalmazzák a megadott záró sorszámot. Ha tehát az `[1:11]` vagy a

`range(1, 11)` tartományt adod meg, akkor valójában az 1 és 10 közötti indexek és értékek kimenetét fogod megkapni.

* **Rossz nagybetűs írásmód használata:** A Python megkülönbözteti a kis- és nagybetűket, szóval például a Változóm a VÁLTOZÓM nem ugyanaz. Ha azt tapasztalod, hogy nem férsz hozzá egy értékhez, mindig ellenőrizd a kis- és nagybetűket.
* **Elgépelés:** Időnként még a tapasztalt programozóknak is meggyűlik a baja az elgépelésekkel. Az segít, ha ügyelsz rá, hogy egységes megközelítést használj a változók, az osztályok és a függvények elnevezéséhez. Azonban még egy következetes elnevezési rendszerrel sem akadályozhatod meg mindig, hogy mondjuk Változót írj, amikor Változómat akartál írni.
* **A függvények alapértelmezett értékei működésének félreértése:** A függvények alapértelmezett értékét az első kiértékelésükkor állítja be a rendszer, nem pedig minden egyes alkalommal, amikor meghívod őket. Így ha deklarálod a következő függvényt:

```
def függvényem(lista=[]):
    lista.append("érték")
    return lista
```

az csak az első hívásakor fog üres listát visszaadni, nem pedig minden egyes hívásakor, ha nem adod meg a `lista` értékét. A további hívások egyszerűen hozzáadják az `"érték"` elemet egy folyamatosan növekvő listához. Tehát ha háromszor hívod meg a `függvényem()` függvényt, akkor a ~~`lista`~~ értéke valójában `["érték", "érték", "érték"]` lesz. Ezt a problémát úgy kerülheted el, hogy a kódban minden alkalommal ellenőrzöd a bemeneti értéket, és annak megfelelően jársz el, például:

```
def függvényem(lista=None):
    if lista is None:
        list = []
    lista.append("érték")
    return lista
```

* **Lista módosítása annak bejárása során:** Ha a programozó szerencsés, akkor ez a tévedés tartományon kívüli indexet jelző hibát okoz. Ez legalább jelzi valamennyire, hogy hol kell keresni a problémát. Amikor azonban olyan adattudományos problémákon dolgozunk, amelyekben nem a teljes listát használjuk, hanem egyszerűen csak a lista egyes részeit járjuk be, ez a tévedés mindenféle adattorzítási és -elemzési problémát okozhat, amelyeket rendkívül nehéz lehet megtalálni (feltéve, hogy egyáltalán feltűnik, hogy probléma áll fenn). A listaértelmezések használata gyakori módszer ennek a problémának az elkerülésére.
* **A Python egy szabványos könyvtármoduljával ütköző modulnév megadása:** Ha olyan modult hozol létre, amelynek a neve megegyezik egy meglévő Python-moduléval, előfordulhat, hogy a Python a kívánt modul helyett a saját modulodat importálja, ami nehezen feltárható hibákat okozhat. Ezt a problémát úgy kerülheted el a legkönnyebben, ha olyan modulneveket használsz, amelyek garantáltan egyediek, például ha előtagként a modul nevéhez illeszted a szervezeted nevét.

Vonaldiagram-stílusok

Amikor diagramot készítesz, az információforrásokat nem csupán vonalakkal kell jelölnöd. Ha különböző vonaltípusokkal és adatpontszimbólumokkal készítesz diagramot, akkor mások sokkal könnyebben tudják használni a diagramot. A következő táblázatban a vonaldiagramok stílusainak felsorolását láthatod.

Szín		***Jelölő***		***Stílus***	
Kód	***Vonalszín***	***Kód***	***Jelölőstílus***	***Kód***	***Vonalstílus***
`b`	kék	`.`	pont	`-`	folytonos
`g`	zöld	`o`	kör	`:`	pontozott

r	piros	x	x jel	-.	szaggatott-pontozott
c	ciánkék	+	pluszjel	--	szaggatott
m	magenta	*	csillag	(nincs)	nincs vonal
y	sárga	s	négyzet		
k	fekete	d	rombusz		
w	fehér	v	lefelé mutató háromszög		
		^	felfelé mutató háromszög		
		<	balra mutató háromszög		
		>	jobbra mutató háromszög		
		p	ötágú csillag		
		h	hatágú csillag		

Ne feledd, hogy ezeket a stílusokat más típusú diagramokhoz is használhatod. Egy pontdiagramon például ezekkel a stílusokkal határozhatod meg az egyes adatpontokat. Ha nem vagy biztos a dolgodban, próbáld ki a stílusokat, hogy lásd, használhatók-e az adott diagramon.

Az IPython gyakran használt mágikus függvényei

Eléggé hihetetlen belegondolni, hogy az IPythonban varázslatot használhatsz, a mágikus függvények viszont pontosan ezt biztosítják. A legtöbb mágikus függvény vagy `%` vagy `%%` jellel kezdődik. A `%` jellel kezdődőek a környezet szintjén működnek, a `%%` jellel kezdődőek pedig cellaszinten.

Van néhány speciális függvény, például a rendszerparancsra váltás (`!`), amely speciális szimbólumot vagy módszert igényel. Ezek közül a rendszerparancsra váltást a leglényegesebb, hogy ismerd. Egy másik hasznos lehetőség a változókiterjesztés (`$`), amelyet `$(változóm)` alakban használhatsz, hogy annak újbóli beírása nélkül adj értéket.

Fontos megjegyezni, hogy a mágikus függvények a Jupyter Notebookban működnek a legjobban. Akik alternatív lehetőségeket, például a Google Colabot használják, azt tapasztalhatják, hogy egyes mágikus függvények nem a kívánt eredményt adják vissza.

Az alábbi listában bemutatunk néhányat a leggyakrabban használt mágikus függvények közül, és hogy mire valók. Ha látni szeretnéd a teljes listát, írd be a **`%quickref`** parancsot és nyomd meg az Entert a Jupyter Notebookban vagy a Google Colabban, vagy a https://damontallen.github.io/IPython-quick-ref-sheets/ címen is megnézheted azt.

Mágikus függvény	***A típus önmagában megadja az állapotot?***	***Leírás***
`%%timeit` vagy `%%prun`	Nem	Az adott cellában szereplő összes utasítás legjobb időbeli teljesítményét számítja ki, kivéve azt, amelyik a sorszintű mágikus függvénnyel egy cellasorban található (ami így inicializáló utasítás is lehet). A `%%prun` változat részletesebb adatokat nyújt, mivel a Python profilkészítőjének kimenetére támaszkodik.
`%%writefile`	Nem	A megadott fájlba írja egy cella tartalmát.
`%alias`	Igen	Hozzárendeli egy rendszerparancs alias nevét, vagy megjeleníti azt.
`%autocall`	Igen	Lehetővé teszi, hogy zárójelek megadása nélkül hívj meg

		függvényeket. A lehetséges beállítások a következők: `Off` (Ki), `Smart` (Intelligens, ez az alapértelmezett beállítás) és `Full` (Teljes). A `Smart` beállítás csak akkor alkalmazza a zárójeleket, ha a híváshoz argumentumot is megadsz.
`%automagic`	Igen	Lehetővé teszi, hogy a `%` jel megadása nélkül hívj meg sorszintű mágikus függvényeket. A lehetséges beállítások a következők: `False` (Hamis, ez az alapértelmezett beállítás) és `True` (Igaz).
`%bookmark`	Nem	Egy meghajtó könyvtárrendszerén belüli aktuális hely követésére szolgáló könyvjelzőket állít be, vagy pedig listázza vagy törli azokat. A https://ipythonbook.com/magic/bookmark.html címen elérhető cikkben találhatsz további információt ennek a mágikus függvénynek a használatáról.
`%cd`	Igen	Egy új tárolási helyre váltja a könyvtárat. Ezzel a paranccsal a könyvtárak előzményeit is áttekintheted, vagy könyvjelzőre cserélheted a könyvtárakat.
`%cls` vagy `%clear`	Nem	Törli a képernyőt.
`%colors`	Nem	Megadja a felszólításokhoz, az információs rendszerhez és a kivételkezelőkhöz kapcsolódó szövegek megjelenítéséhez használt színeket. A `NoColor` (nincs szín, vagyis fekete-fehér), a `Linux` (ez az alapértelmezett beállítás) és a `LightBG` lehetőség közül választhatsz.
`%config`	Igen	A használatával konfigurálhatod az IPythont.
`%debug` vagy `%%debug`	Igen	Elindítja a Python interaktív hibakeresőjét, hogy a Notebook-környezetben tudj hibakeresést végezni egy alkalmazásban.
`%dhist`	Igen	Az aktuális munkamenet során meglátogatott könyvtárak listáját jeleníti meg.
`%env`	Igen	Lekéri, beállítja vagy listázza a környezeti változókat.
`%file`	Nem	Annak a fájlnak a nevét írja ki, amelyben az objektum forráskódja található.
`%hist`	Igen	Az aktuális munkamenet során kiadott, mágikus függvényekre vonatkozó parancsok listáját jeleníti meg.
`%install_ext`	Nem	Telepíti a megadott bővítményt.
`%load`	Nem	Egy másik forrásból, például egy online példából tölt be alkalmazáskódot.
`%load_ext`	Nem	Egy Python-bővítményt tölt be a modulneve alapján.
`%lsmagic`	Igen	Az aktuálisan elérhető mágikus függvények listáját jeleníti meg.
`%matplotlib`	Igen	A diagramokhoz használt háttérfeldolgozót állítja be. Ha az `inline` (beágyazott) értéket használod, akkor a diagram az IPython-jegyzetfüzetfájl celláján belül jelenik meg. A lehetséges értékek a következők: „`gtk`”, „`gtk3`”, „`inline`”, „`nbagg`”, „`osx`”, „`qt`”, „`qt4`”, „`qt5`”, „`tk`”, és „`wx`”.
`%more`	Nem	Megjelenít egy fájlt a lapozóban, hogy át tudj tekinteni egy adatfájlt, miközben a kódban használod.
`%paste`	Nem	Beilleszti a vágólap tartalmát az IPython-környezetébe.

`%pdef`	Nem	Azt adja meg, hogyan hívhatod meg az objektumot (feltéve, hogy az objektum meghívható).
`%pdoc`	Nem	Egy objektumhoz tartozó docstring dokumentációs sztringet jeleníti meg.
`%pinfo`	Nem	Részletes információkat jelenít meg az objektumról (amelyek gyakran bővebbek, mint ami önmagában a súgóban szerepel).
`%pinfo2`	Nem	Rendkívül részletes információkat jelenít meg az objektumról (ha azok elérhetők).
`%psource`	Nem	Az objektum forráskódját jeleníti meg (feltéve, hogy a forráskód elérhető).
`%reload_ext`	Nem	Újból betölt egy korábban telepített bővítményt.
`%timeit` vagy `%prun`	Nem	Egy utasítás legjobb teljesítményéhez tartozó időt számítja ki. A %prun változat részletesebb adatokat nyújt, mivel a Python profilkészítőjének kimenetére támaszkodik.
`%unalias`	Nem	Eltávolít egy korábban létrehozott alias nevet a listáról.
`%unload_ext`	Nem	Eltávolítja a megadott betöltött kiterjesztést a környezetből.