# Fake and Real News Detection using LSTM Deep Learning

**Cél:** egy olyan projekt létrehozása, amely segítségével meg tudjuk állapítani az igaz, illetve hamis híreket.

**A folyamat röviden:** egy nagy könyvtárt halmazunk. Ezután beolvassuk az adatbázisunkat. Ezen az adatbázison belül elemezzük melyek az igaz, és melyek a hamis hírek. Mindezek után kicsit megtisztítjuk az adatainkat, hogy később fel tudjuk dolgozni őket.

Ehhez egy igaz és hamis híreket tartalmazó adatbázist használunk. Két cvs fájlunk van: Fake.csv True.csv, amelyek a nevekhez megfelelően tartalmazzák a híreket.

Az adatainkat következőkre tudjuk elkülöníteni:

* cím
* szöveg
* tárgy
* dátum

# Projekt részletes leírása

Mielőtt megkezdjük a kód megírását, a következő módosításokat végezzük a felületen: Notebook settings-nél a Hardware accelerator típusát állítsuk át GPU-ra. Ez azért szükséges, hogy gyorsabban tudjuk elvégezni a training-eket. Másképp ez akár napokat is igénybe vehetne.

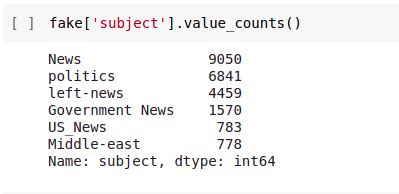
Következő könyvtárakat importáljuk:

* tensorflow
* numpy
* pandas
* matplotlib.pyplot
* seaborn
* re
* nltk
* wordcloud

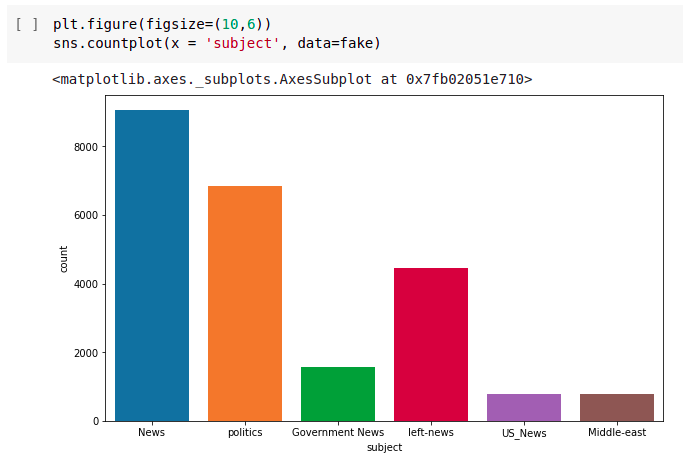
Tokenizer-eket és szekvenciákat importálunk (az utóbbira azért van szükség, hogy kitöltsük azokat az adathalmazokat, amelyek nem elég hosszúak).

Importáljuk a Fake/True híreket tartalmazó adathalmazainkat. Ezek egy-egy változóba mentjük el megfelelő névvel.

## Fake News importálása

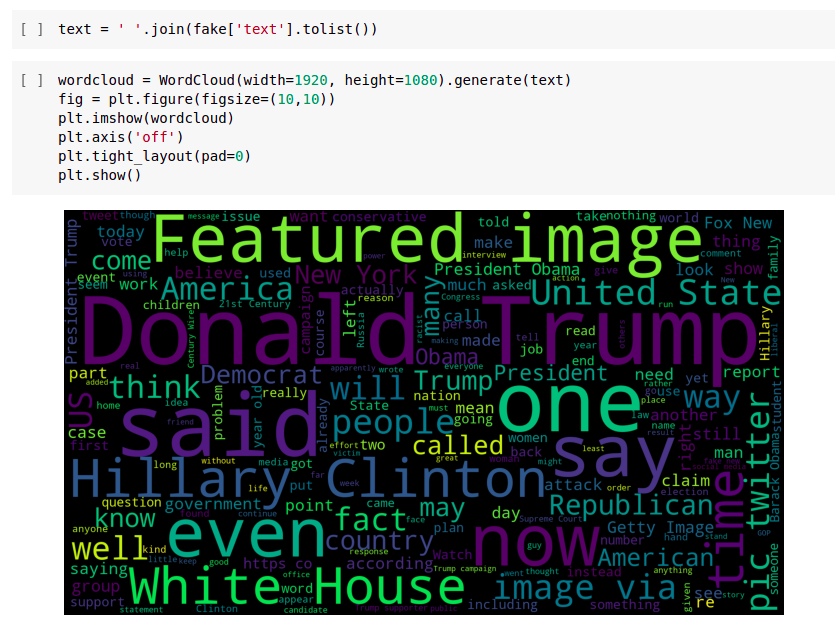
Már most is le tudunk dolgozni 1-2 dolgot. Például tárgy szerint összegezni tudunk, milyen híreink vannak.

Majd egy diagramon ábrázolhatjuk is tárgy szerint a híreket:

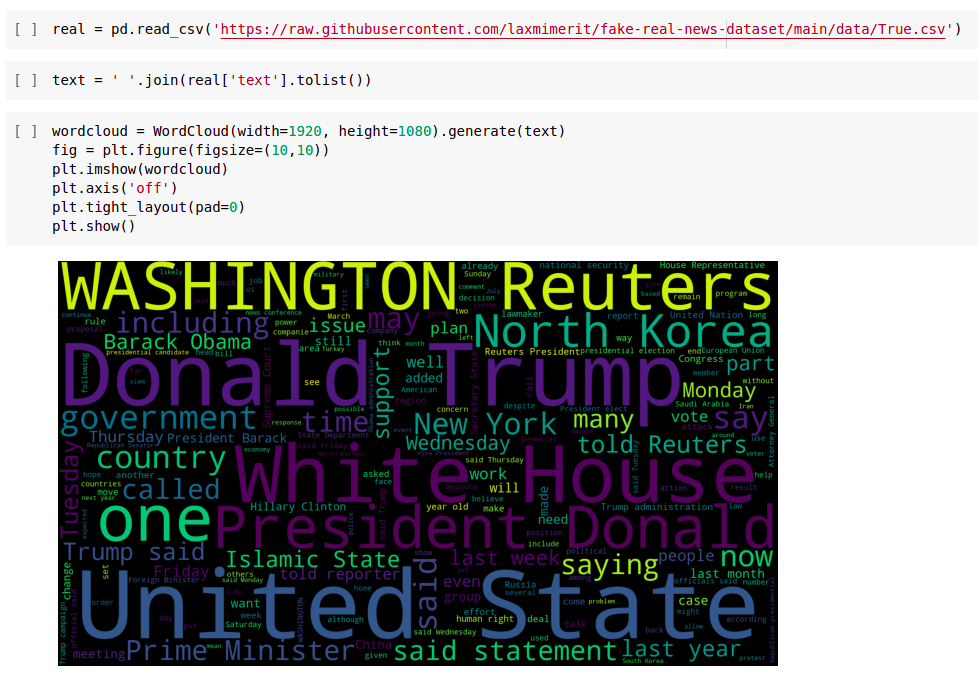
**Workcloud**

Az importálás lehetővé teszi, hogy megállapítsuk a szöveg szavainak számát vizualizáció segítségével.

Létrehozunk egy listát, aminek az elemei a hírek mondatai lesznek. Majd az így kapott listát egy szöveggé alakítjuk a join-nal.

A workcloud változóba pedig legeneráljuk a szöveget, majd megjelenítjük az imshow()-val egy képen. Megadjuk a paramétereit a képnek: szélességet, magasságot, a tengelyek és a kép közötti hely méretét (padding). Láthatjuk a betűméret egyenesen arányos az adott szó előfordulásának gyakoriságával:

## Real news importálása

Teljesen hasonló módon, mint a fake news-nál, beimportáljuk a csv-t, majd elkészítjük a true news szóhalmazának vizualizációját is.

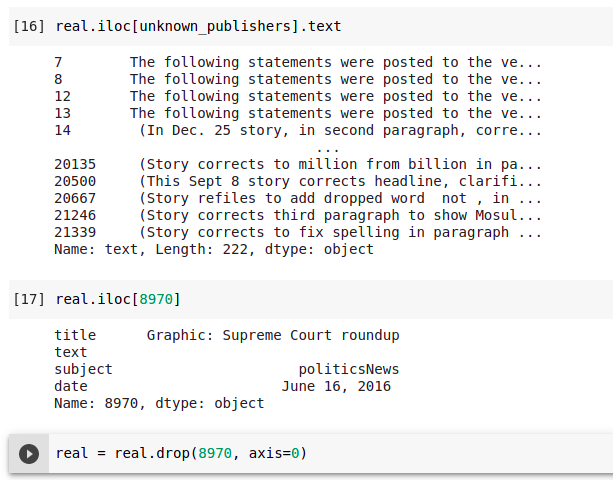
Ha összehasonlítjuk a két wordcloud-ot, láthatjuk, hogy bizonyos szavak, amik a valós hírekben szerepelnek (White House, WASHINGTON Reuters) az álhírekben szinte meg se jelennek. A WASHINGTON Reuters a publikáció forrása, amit érthető módon az álhíres verzió meg sem említ.

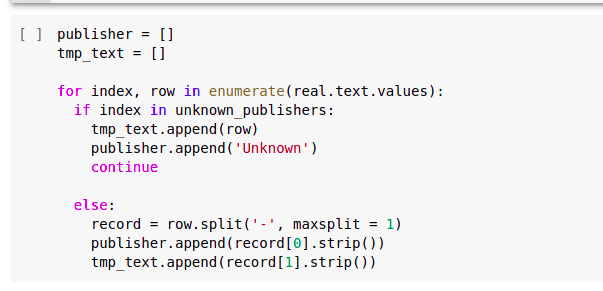
## Cleaning Data

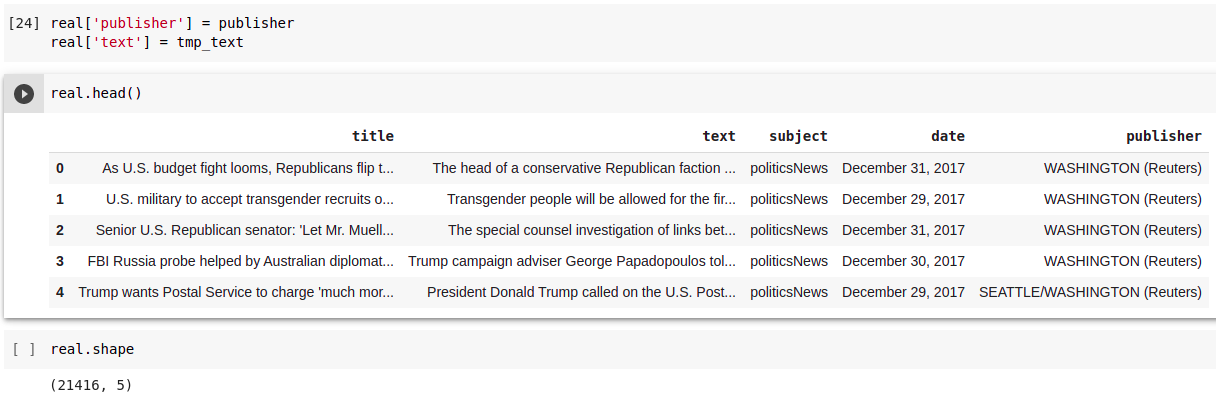
Az adathalmazunkat meg kell tisztítani. El akarjuk távolítani a hivatkozásokat, ezért ki kell szűrni, azokat a részeket a real news-ból, ahol hivatkozás történik a forrásra: (Reuters)-. A kötőjelek mentén fogjuk a szűrést elvégezni. Egy listába tároljuk el azokat a részeket, amik nem tartalmaznak forráshivatkozást. A listában indexelve helyezzük el az elemeket az enumerate-tel. Lekezeljük azt az esetet, ha nem tartalmaz az adott sor – jelet. Ha olyan sorhoz érkezünk, megvizsgáljuk, hogy a Twitter-ről származik-e az információ: azaz ha 120 karakternél kevesebb a szöveg mérete, akkor biztosan onnan származik.

Az unknown publishers listába így megkapjuk azokat az indexeket, amikre teljesül, hogy ismeretlen szerzőktől származnak az információk, azaz a Twitter-ről. A len függvénnyel pedig megkapjuk, hogy összesen hány ilyen ismeretlen szerző van.



Az ismeretlen szerzőkhöz tartozó szövegrészeket feltüntetjük, majd azokat a sorokat, ahol nem jelenik meg szöveg, eltávolítjuk a droppal a szerző indexére hivatkozva:

Az ismeretlen szerzőket az Unknown jelzővel látjuk el a publishers listában:

Majd készítünk két új oszlopot publisher és text néven, amikbe az előbb előállított két lista (publisher, text) elemeit helyezzük el:

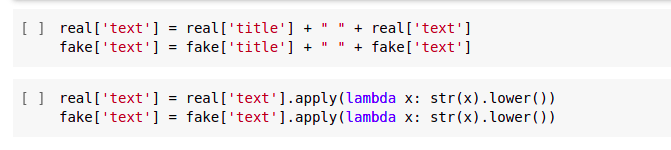
Ennél a pontnál az adathalmazunkat sikeresen bekategorizáltuk ismert és ismeretlen szerzők alapján. Megtörtént a két kategória sikeres elszeparálása egymástól.

Egy list comprehension-nel megvizsgáljuk, hogy a fake text elemei közül melyek üresek, és azoknak az indexeit kigyűjtjük az emmpty\_fake\_index listába:

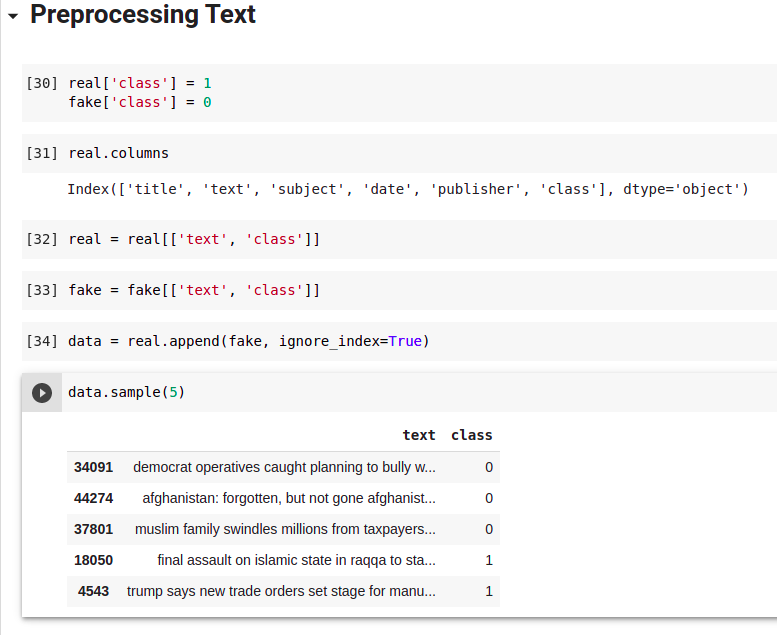
Fentebb láthatjuk azokat a címeket, amikhez nem tartoznak szövegrészek. Ezeket kell eltávolítanunk az adathalmazból, hogy elkészíthessük a modellünket a mélytanuláshoz.

A cél az, hogy ezeket a címeket összefésüljük a szövegrészekkel külön a fake és külön a real adathalmazokra is.

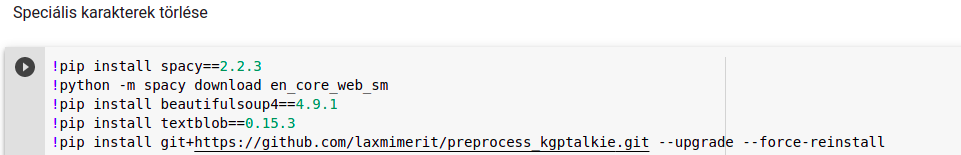
Ezt követően megkezdhetjük az előfeldolgozást. Láthatjuk, hogy a címek csupa nagybetűsen szerepelnek, ezeket először is kisbetűsökké alakítjuk.



## Preprocessing Text

Készítünk2 osztály oszlopot a fake és real news számára 1-es és 0-ás értékekkel. Ezt a két értéket fogjuk kombinálni: text és class, a fake és a real esetén is. Majd az így kapott új két adathalmazt egyesítjük a data adathalmazba. Így a kimeneten a class oszlopban láthatjuk hogy ha 0 szerepel, akkor az adott sor fake news, ha 1 akkor pedig real news.

A következőkben eltávolítjuk a speciális karaktereket, mint például az idézőjeleket, pontokat. Ehhez a preprocess\_kgptalkie-t használjuk, amit a következő módon installálhatunk:

****Sikeres telepítés után a beimportálást követően ebből a csomagból fogjuk alkalmazni a speciális karaktereket eltávolító metódust (remove\_special\_chars):

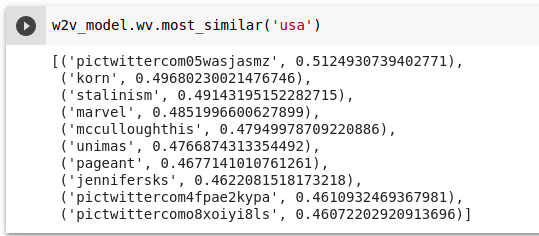
## Szöveges adat átalakítása numerikus adattá

Vektorizáció: az előbb kapott szöveges adatainkat numerikus adatokká konvertáljuk. Ehhez a Wolrd2Vec technikát alkalmazzuk, ami az adott szót vektorok szekvenciájává alakítja. Amennyivel dolgozunk, annyi lesz a dimenziónk száma.

Improtáljuk a gensim könyvtárat. Y-ban eltároljuk a class adatainkat (hogy 0 vagy 1, azaz fake vagy real a hír), illetve a text értékeket pedig listává alakítjuk list comprehension-nel, így megkapjuk a listák listáját, amit X-ben tárolunk.

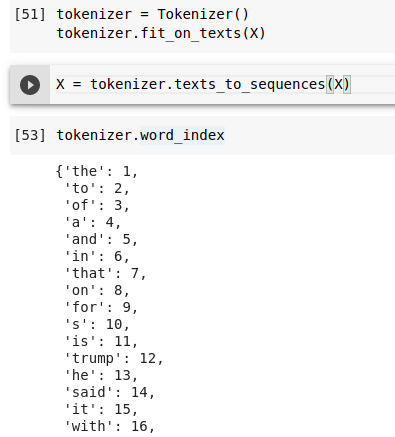
Megadjuk a kívánt dimenzió számát, ami azt jelenti hogy egy 100 vektoros szekvenciával dolgozunk. Használjuk a Word2Vec-et, beállítjuk, hogy 10 egység kerüljön egy vektorba, azaz 10 szó szerepeljen együtt. A len függvénnyel pedig lekérjük az így létrejött szótárunk szavainak számát. Ez a kapott szám az egyedi szavak összege.

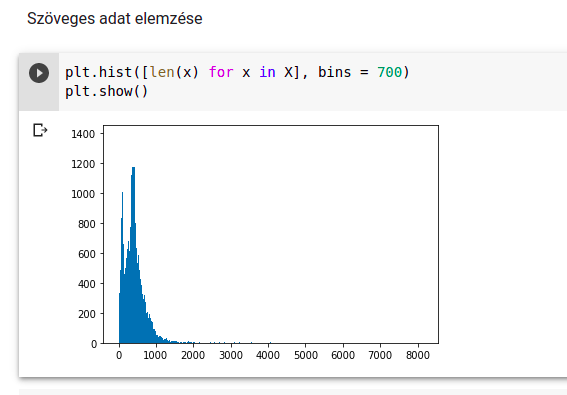
Ha megnézzük a vektorizációt a love szóra, akkor lárhatjuk hogy a kimenet a love vektorok szekvenciájaként állt elő numerikus adatokkal:

Azt is megtudhatjuk, hogy egy adott szóhoz mik a leghasonlóbb szavak a most\_similar tagfüggvénnyel:

Most már, hogy vannak vektoraink, 2 metódusra van szükségünk ahhoz, hogy feltanítsuk a modellünket. Az egyik metódus közvetlenül a vektorokkal fog dolgozni, a másik pedig a vektorokat fogja súlyozni a modellben.

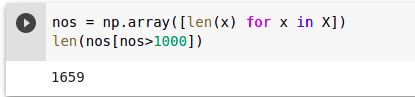
Készítünk egy Tokenizer-t. A tokenizáció után a szöveges adat egy szekvenciává konvertálódik, mint például 1 2 3 4 5. A szkevenciákat tartalmazó lista az X-ben található. A tokenizer.word\_index-szel pedig megkapjuk minden egyes szóhoz tartozó indexet:

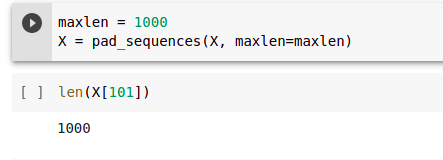


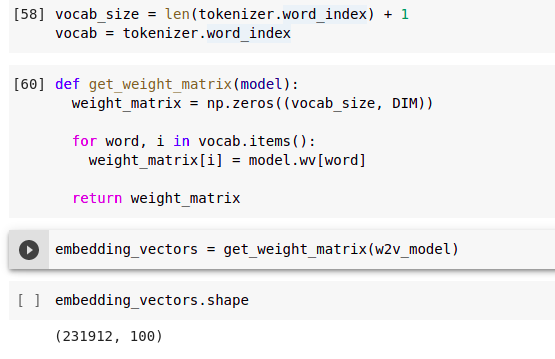
Megkezdhetjük a szöveg analizálását. Tudni szeretnénk, hogy hány szó van egy sorban. Ezért végigmegyünk a szekvenciákat tartalmazó X listán és lekérjük az adott szekvencia hosszát. Az így kapott eredményeket tartalmazó listát átadjuk a hisztogram készítő függvénynek.

Ez a hisztogram mutatja meg a szavak számát a hírekben. Látható, hogy nagyjából 1000 szószám alatti eredményeket kaptunk.

A következőkben lekérjük azokat a híreket, amik több mint 1000 szót tartalmaznak. Ezeknek a száma:

Majd beállítjuk, hogy a hírek szavainak a száma csakis 1000 lehet. Azaz, ha több, mint 1000, akkor csonkítunk, ha kevesebb, mint 1000, akkor 0-át adunk, ezt jelzi a pad\_sequences. Így akármelyik indexére hivatkozunk az X-nek, 1000-et kapunk mindig a len függvény visszatérési értékeként:

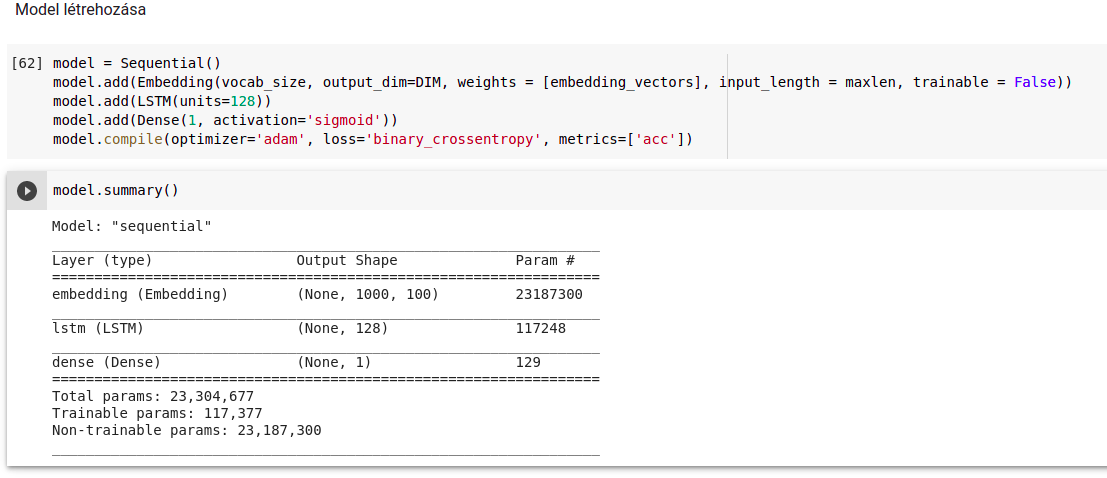
Mivel a szótárunk indexelése 1-től indul, így a szótár méreténél +1-et hozzá kell adni a tömb méretéhez. Elkészítjük a súlyokat tartalmazó vektorunkat is. Ehhez az input vektorok kezdeti súlyaihoz újabb súlyokat adunk, hogy minél pontosabb legyen a tanítás. Létrehozunk a szótár méretével megegyező 100 dimenziós mátrixot, kezdetben csupa nullás értékekkel. Végigmegyünk a szótár elemein, az indexeken és a szavakon, és beállítjuk a súlyokat.

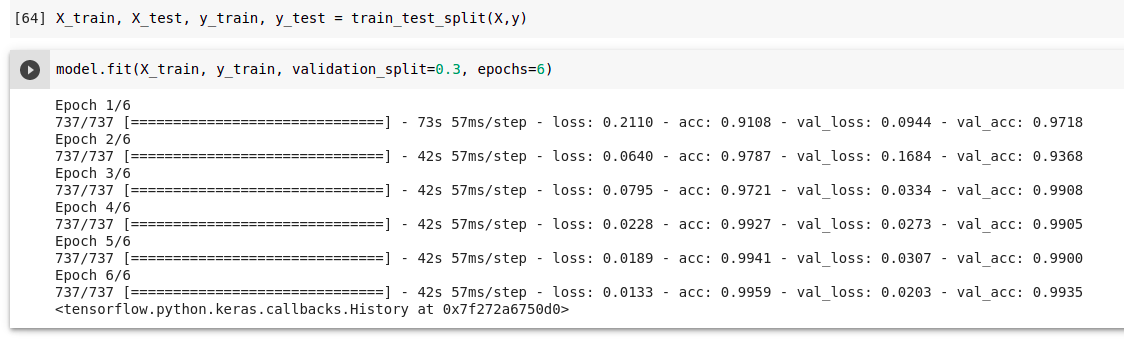
Az így kapott súlyozott mátrixot átadjuk a modellnek, és eltároljuk az embedding\_vectors-ben.

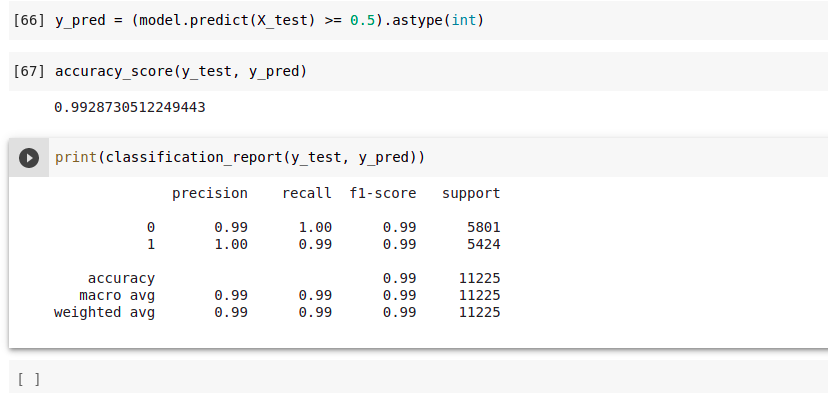
## Modell létrehozása

Létrehozzuk a modellünket, hozzáadjuk az lstm, és dense layer-eket, az aktivációs réteget, ami a sigmoid lesz (2 osztályunk van 0-1, ezért választjuk ezt).

Láthatunk egy summary-t a modellről.

Márcsak a tanítás és tesztelés maradt hátra. Az epoch-ok számát 6-ra állítva a következő eredményt kaptuk:

Majdem 100%-os pontosságot kaptunk az álhírek, illetve a valós hírek detektálásánál a validációs adathalmazon, ami egy eléggé biztos meghatározást jelent.

Nézzük meg mekkora pontosságot érünk el a teszt adathalmazunkon!

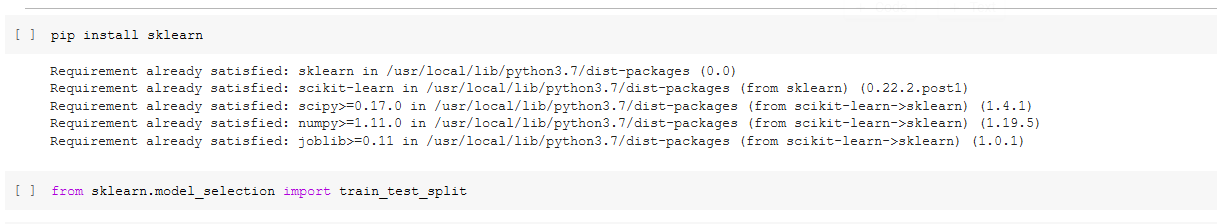
Mint látható, most is hatalmas pontosságot sikerült elérnünk: 99%.

Végül egy klasszifikációs jelentést láthatunk. Mindenhol 99%-os a pontosság, tehát egy tökéletes modellt kaptunk az álhírek detektálására.

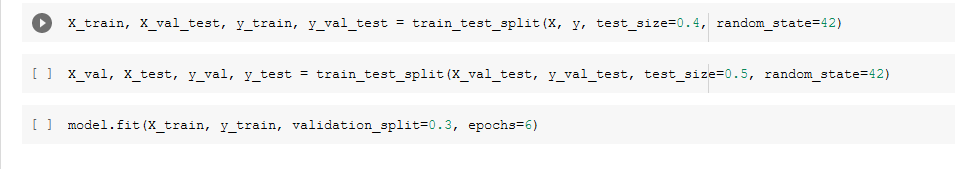
# Projektben való változtatások

## Validációs halmaz alkalmazása:

Először is szükséges volt feltelepíteni a következőket:



Ezután már tudtuk alkalmazni a validációs halmazunkat is:



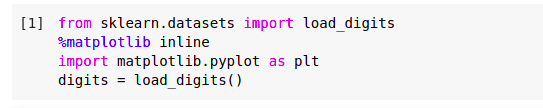
## CNN modell



# 2. feladat: Multiklasszifikáció

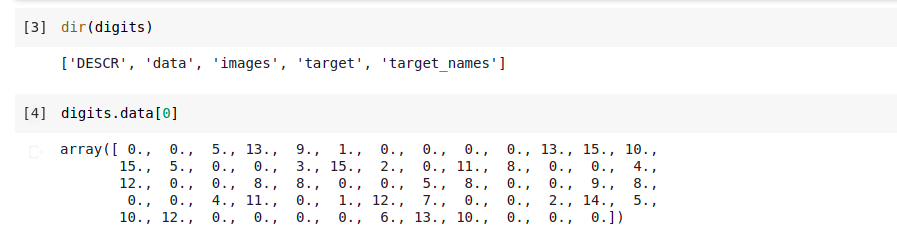
**Cél:** kézzel írt számok felismerése.

Logisztikus regressziós modellt építünk.

Importáljuk a számokat tartalmazó adathalmazunkat:

A load\_digits egy előre definiált adathalmaza az sklearn-nek, ami kifejezetten mélytanulási célból készült. Ez 1797 db 8x8-as képekből áll, amiken kézzel írt számok találhatók. Ezekre a képekre végezzük el a detektálást, hogy a modellünk képes-e felismerni, hogy milyen szám látható a képen.

Az importálásokat követően először is betöltjük a tanuló adathalmazunkat.

A tanuló adathalmazunk sok mindent tartalmaz:

A data egy tömb, ami a 8x8-as képeket tartalmazza, de egydimenziós tömbként van reprezentálva: azaz számszerint 68 elemű, hiszen 8x8=64.

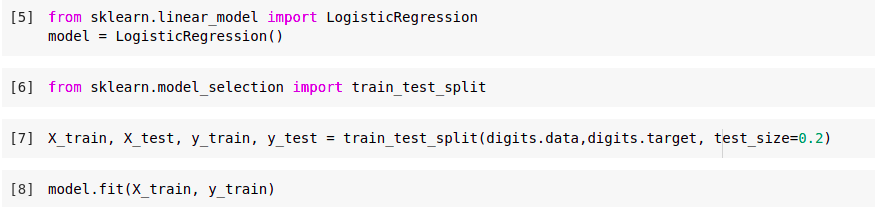
Az első öt ilyen kép megjelenítve:

A target mondja meg, hogy milyen szám található a képen a tanítás szerint.

Tehát a modellünket a data és a target alapján tanítjuk.

A tanítás előtt felosztjuk az adathalmazt tanuló és teszt adathalmazra, 20%-ra állítjuk a teszt méretét.

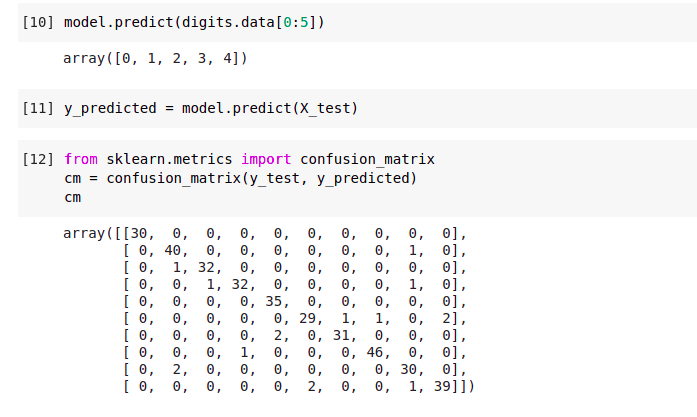
Elkészítjük a regressziós modellt, ehhez pedig importáljuk a LogisticRegression-t és megcsináljuk hozzá a modellobjektumot. Ezt fogjuk majd tanítani, amihez meghívjuk a fit metódust, aminek az x\_train és y\_traint adjuk át. Az x\_train tartalmazza a kézzel írt karaktereket, az y\_train pedig azt, hogy milyen karakter látható a képen.

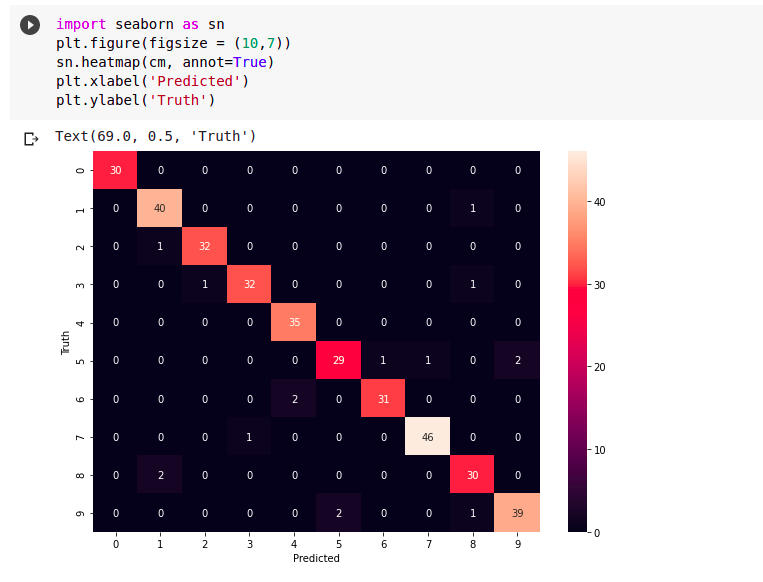
Mostmár, hogy kész a modellünk, kiszámíthatjuk a score-t. A score mondja meg azt, hogy milyen pontos a modellünk. (Az x\_test mondja meg a y\_test várható értékét. Ezt az y várható értéket hasonlítjuk össze az igazi értékkel). Láthatjuk, hogy viszonylag egész pontos, 96%.

Ha kipróbáljuk egy tetszőleges képre a predikciót a modellünkön, mondjuk a 67-es indexűre, akkor lárhatjuk, hogy a target-tel megegyező eredményt produkál, azaz valóban 6-osnak ismerte fel a képen látható számot:

 Próbáljuk meg az első öt számra is jól működik-e!

Nézzük meg, hogy ha egy confusion mátrixot készítünk, milyen eredményeket kapunk!

y\_test-et és az y\_predicted-et adjuk át a confiuson matrix-nak. Az y\_test-ben vannak a valós értékek, az y\_predictben pedig a modellünk álltal meghatározott értékek. Egy két dimenziós tömböt kapunk kimenetként, amit a seaborn importálásával vizualizálunk, hogy jobban átlátható legyen, mennyire dolgozik pontosan a modellünk.

Az ábrán azt láthatjuk, hogy a bal felső sarokban 30 alkalommal a valós érték, a truth 0 volt, és a modellünk is nullának prediktálta.

Alulról a második sor második eleme estén a valós, a truth 30-szor volt 8, de a modellünk azt mondta 2-szer, nem, ez 1. Tehát ezek alapján azt tudjuk elmondani, hogy ha egy sorban nem szerepel csupa nulla, akkor a modellünk nem dolgozott eléggé pontosan.