Bu not defterinde farklı karakter kodlamalarıyla çalışacağız.

Hadi başlayalım!

Get our environment set up[¶](https://www.kaggle.com/code/alexisbcook/character-encodings" \l "Get-our-environment-set-up" \t "_self)

Yapmamız gereken ilk şey, kullanacağımız kütüphaneleri yüklemek. Ama bu bizim veri setimiz değil: ona daha sonra değineceğiz!

*# modules we'll use*

import pandas as pd

import numpy as np

*# helpful character encoding module*

import charset\_normalizer

*# set seed for reproducibility*

np.random.seed(0)

# What are encodings?[¶](https://www.kaggle.com/code/alexisbcook/character-encodings#What-are-encodings?)

Karakter kodlamaları, ham ikili bayt dizilerini (şuna benzeyen: 0110100001101001) insan tarafından okunabilir metni oluşturan karakterlere ("hi" gibi) eşlemek için kullanılan belirli bir dizi kuraldır. Birçok farklı kodlama vardır ve metni başlangıçta yazıldığı kodlamadan farklı bir kodlamayla okumaya çalışırsanız, "mojibake" (mo-ci-ba-ke gibi okunur) adı verilen bozulmuş metinle karşılaşırsınız. İşte bir mojibake örneği: æ–‡å—åŒ–ã??

Ayrıca "bilinmeyen" karakterlerle de karşılaşabilirsiniz. Bunlar, kullanmakta olduğunuz kodlamada belirli bir bayt ile bir karakter arasında bir eşleşme olmadığında yazdırılan ve şöyle görünen karakterlerdir:

Karakter kodlama uyuşmazlıkları bugün eskisi kadar yaygın değildir, ancak kesinlikle hala bir sorundur. Birçok farklı karakter kodlaması vardır, ancak bilmeniz gereken en önemlisi **UTF-8**'dir.

**UTF-8**, standart metin kodlamasıdır. Tüm Python kodu UTF-8'dedir ve ideal olarak tüm verileriniz de öyle olmalıdır. İşler UTF-8'de olmadığında sorunlarla karşılaşırsınız.

Python 2'de kodlamalarla uğraşmak oldukça zordu, ancak neyse ki Python 3'te çok daha basittir. (Kaggle Notebook'ları yalnızca Python 3 kullanır.) Python 3'te metinle çalışırken karşılaşacağınız iki ana veri türü vardır. Biri, metnin varsayılan olarak sahip olduğu dize (string) türüdür.

*# start with a string*

before = "This is the euro symbol: €"

*# check to see what datatype it is*

type(before)

str

Diğer veri türü, tam sayılardan oluşan bir dizi olan bayt veri türüdür. Bir dizeyi, hangi kodlamada olduğunu belirterek baytlara dönüştürebilirsiniz:

*# encode it to a different encoding, replacing characters that raise errors*

after = before.encode("utf-8", errors="replace")

*# check the type*

type(after)

bytes

Bir "bytes" objesine bakarsanız, önünde **b** harfi ve sonrasında metin göreceksiniz. Bunun nedeni, baytların sanki ASCII ile kodlanmış karakterlermiş gibi yazdırılmasıdır. (ASCII, İngilizce dışında herhangi bir dilde yazmak için pek uygun olmayan eski bir karakter kodlamasıdır.) Burada, euro sembolümüzün, sanki bir ASCII dizesiymiş gibi yazdırıldığında "\xe2\x82\xac" gibi görünen bir "mojibake" ile değiştirildiğini görebilirsiniz.

*# take a look at what the bytes look like*

after

b'This is the euro symbol: \xe2\x82\xac'

Baytlarımızı doğru kodlamayla bir dizeye geri dönüştürdüğümüzde, metnimizin tamamının doğru şekilde orada olduğunu görebiliriz, bu harika! :)

*# convert it back to utf-8*

print(after.decode("utf-8"))

This is the euro symbol: €

Ancak, baytlarımızı bir dizeye (string) dönüştürmek için farklı bir kodlama kullanmaya çalıştığımızda bir hata alırız. Bunun nedeni, kullanmaya çalıştığımız kodlamanın, ona iletmeye çalıştığımız baytlarla ne yapacağını bilememesidir. Python'a, bayt dizesinin aslında hangi kodlamada olması gerektiğini söylemeniz gerekir.

Farklı kodlamaları, müziği kaydetmenin farklı yolları olarak düşünebilirsiniz. Aynı müziği bir CD'ye, kaset banda veya 8-track'e (eski bir ses kayıt formatı) kaydedebilirsiniz. Müzik az çok aynı ses çıkarsa da, her bir kayıt formatından müziği çalmak için doğru ekipmanı kullanmanız gerekir. Doğru kod çözücü (decoder), bir kaset çalar veya bir CD çalar gibidir. Bir kaseti CD çalarda oynatmaya çalışırsanız, çalışmaz.

*# try to decode our bytes with the ascii encoding*

print(after.decode("ascii"))

---------------------------------------------------------------------------

UnicodeDecodeError Traceback (most recent call last)

/tmp/ipykernel\_19/27547290.py in <module>

**1** # try to decode our bytes with the ascii encoding

----> 2 print(after.decode("ascii"))

UnicodeDecodeError: 'ascii' codec can't decode byte 0xe2 in position 25: ordinal not in range(128)

Metni baytlara dönüştürmek için yanlış kodlamayı kullanmaya çalıştığımızda da sorunlarla karşılaşabiliriz. Daha önce de söylediğim gibi, Python 3'te dizeler (stringler) varsayılan olarak UTF-8'dir, bu nedenle onları başka bir kodlamadaymış gibi ele almaya çalışırsak sorunlar yaratırız.

Örneğin, bir dizeyi encode() kullanarak ASCII için baytlara dönüştürmeye çalışırsak, metin ASCII'de olsaydı nasıl baytlara dönüşeceğini sorabiliriz. Ancak metnimiz ASCII'de olmadığı için, ASCII'nin ele alamayacağı bazı karakterler olacaktır. ASCII'nin işleyemediği bu karakterleri otomatik olarak değiştirebiliriz. Ancak bunu yaparsak, ASCII'de olmayan tüm karakterler bilinmeyen bir karakterle değiştirilir. Sonrasında, baytları tekrar bir dizeye dönüştürdüğümüzde, orijinal karakter yine bilinmeyen karakterle değiştirilmiş olacaktır. Bunun tehlikeli yanı, o karakterin **gerçekte** ne olması gerektiğini anlamanın hiçbir yolu olmamasıdır. Bu da verilerimizi kullanılamaz hale getirmiş olabileceğimiz anlamına gelir!

*# start with a string*

before = "This is the euro symbol: €"

*# encode it to a different encoding, replacing characters that raise errors*

after = before.encode("ascii", errors = "replace")

*# convert it back to utf-8*

print(after.decode("ascii"))

*# We've lost the original underlying byte string! It's been*

*# replaced with the underlying byte string for the unknown character :(*

This is the euro symbol: ?

Bu kötü bir şey ve bunu yapmaktan kaçınmak istiyoruz! Tüm metnimizi mümkün olan en kısa sürede UTF-8'e dönüştürüp bu kodlamada tutmak çok daha iyi. UTF-8 olmayan girdileri UTF-8'e dönüştürmek için en iyi zaman, dosyaları okuduğunuz zamandır; bundan birazdan bahsedeceğiz.

# Reading in files with encoding problems[¶](https://www.kaggle.com/code/alexisbcook/character-encodings" \l "Reading-in-files-with-encoding-problems" \t "_self)

Karşılaşacağınız dosyaların çoğu muhtemelen UTF-8 ile kodlanacaktır. Python varsayılan olarak bunu bekler, bu nedenle çoğu zaman sorun yaşamazsınız. Ancak bazen şuna benzer bir hata alırsınız:

*# try to read in a file not in UTF-8*

kickstarter\_2016 = pd.read\_csv("../input/kickstarter-projects/ks-projects-201612.csv")

---------------------------------------------------------------------------

UnicodeDecodeError Traceback (most recent call last)

/tmp/ipykernel\_19/3982885289.py in <module>

**1** # try to read in a file not in UTF-8

----> 2 kickstarter\_2016 = pd.read\_csv("../input/kickstarter-projects/ks-projects-201612.csv")

/opt/conda/lib/python3.7/site-packages/pandas/util/\_decorators.py in wrapper(\*args, \*\*kwargs)

**309** stacklevel=stacklevel,

**310** )

--> 311 return func(\*args, \*\*kwargs)

**312**

**313** return wrapper

/opt/conda/lib/python3.7/site-packages/pandas/io/parsers/readers.py in read\_csv(filepath\_or\_buffer, sep, delimiter, header, names, index\_col, usecols, squeeze, prefix, mangle\_dupe\_cols, dtype, engine, converters, true\_values, false\_values, skipinitialspace, skiprows, skipfooter, nrows, na\_values, keep\_default\_na, na\_filter, verbose, skip\_blank\_lines, parse\_dates, infer\_datetime\_format, keep\_date\_col, date\_parser, dayfirst, cache\_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, encoding\_errors, dialect, error\_bad\_lines, warn\_bad\_lines, on\_bad\_lines, delim\_whitespace, low\_memory, memory\_map, float\_precision, storage\_options)

**584** kwds.update(kwds\_defaults)

**585**

--> 586 return \_read(filepath\_or\_buffer, kwds)

**587**

**588**

/opt/conda/lib/python3.7/site-packages/pandas/io/parsers/readers.py in \_read(filepath\_or\_buffer, kwds)

**480**

**481** # Create the parser.

--> 482 parser = TextFileReader(filepath\_or\_buffer, \*\*kwds)

**483**

**484** if chunksize or iterator:

/opt/conda/lib/python3.7/site-packages/pandas/io/parsers/readers.py in \_\_init\_\_(self, f, engine, \*\*kwds)

**809** self.options["has\_index\_names"] = kwds["has\_index\_names"]

**810**

--> 811 self.\_engine = self.\_make\_engine(self.engine)

**812**

**813** def close(self):

/opt/conda/lib/python3.7/site-packages/pandas/io/parsers/readers.py in \_make\_engine(self, engine)

**1038** )

**1039** # error: Too many arguments for "ParserBase"

-> 1040 return mapping[engine](self.f, \*\*self.options) # type: ignore[call-arg]

**1041**

**1042** def \_failover\_to\_python(self):

/opt/conda/lib/python3.7/site-packages/pandas/io/parsers/c\_parser\_wrapper.py in \_\_init\_\_(self, src, \*\*kwds)

**67** kwds["dtype"] = ensure\_dtype\_objs(kwds.get("dtype", None))

**68** try:

---> 69 self.\_reader = parsers.TextReader(self.handles.handle, \*\*kwds)

**70** except Exception:

**71** self.handles.close()

/opt/conda/lib/python3.7/site-packages/pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_\_cinit\_\_()

/opt/conda/lib/python3.7/site-packages/pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_get\_header()

/opt/conda/lib/python3.7/site-packages/pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_tokenize\_rows()

/opt/conda/lib/python3.7/site-packages/pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.raise\_parser\_error()

UnicodeDecodeError: 'utf-8' codec can't decode byte 0x99 in position 7955: invalid start byte

UnicodeDecodeError, UTF-8 baytlarını ASCII olarak çözmeye çalıştığımızda aldığımız hatanın aynısı. Bu bize dosyanın aslında UTF-8 olmadığını söylüyor. Ancak hangi kodlamada olduğunu bilmiyoruz. Bunu anlamanın bir yolu, bir sürü farklı karakter kodlamasını denemek ve hangisinin işe yaradığını görmektir. Ancak daha iyi bir yol, **charset\_normalizer** modülünü kullanarak doğru kodlamanın ne olduğunu otomatik olarak tahmin etmeye çalışmaktır. %100 doğru olacağı garanti değildir, ancak genellikle sadece tahmin etmeye çalışmaktan daha hızlıdır.

Ben bu dosyanın ilk on bin baytına bakacağım. Bu, kodlamayı iyi bir şekilde tahmin etmek için genellikle yeterlidir ve tüm dosyaya bakmaya çalışmaktan çok daha hızlıdır. (Özellikle büyük bir dosyada bu işlem çok yavaş olabilir.) Dosyanın sadece ilk kısmına bakmanın bir başka nedeni de, hata mesajına bakarak ilk sorunun 11. karakterde olduğunu görmemizdir. Bu yüzden, ne olduğunu anlamak için muhtemelen dosyanın sadece ilk küçük kısmına bakmamız yeterlidir.

*# look at the first ten thousand bytes to guess the character encoding*

with open("../input/kickstarter-projects/ks-projects-201801.csv", 'rb') as rawdata:

result = charset\_normalizer.detect(rawdata.read(10000))

*# check what the character encoding might be*

print(result)

{'encoding': 'utf-8', 'language': 'English', 'confidence': 1.0}

Yani charset\_normalizer, doğru kodlamanın "Windows-1252" olduğundan %73 emin. Bakalım doğru mu:

*# read in the file with the encoding detected by charset\_normalizer*

kickstarter\_2016 = pd.read\_csv("../input/kickstarter-projects/ks-projects-201612.csv", encoding='Windows-1252')

*# look at the first few lines*

kickstarter\_2016.head()

/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3553: DtypeWarning: Columns (13,14,15) have mixed types.Specify dtype option on import or set low\_memory=False.

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

Evet, görünüyor ki charset\_normalizer haklıymış! Dosya sorunsuz bir şekilde okunuyor (her ne kadar veri tipleri hakkında bir uyarı alsak da) ve ilk birkaç satıra baktığımızda her şey yolunda görünüyor.

Peki ya charset\_normalizer'ın tahmin ettiği kodlama doğru değilse? charset\_normalizer temel olarak gelişmiş bir tahmin aracı olduğundan, bazen yanlış kodlamayı tahmin edebilir. Deneyebileceğiniz bir şey, dosyanın daha fazlasına veya daha azına bakmak ve farklı bir sonuç alıp almadığınızı görmektir, ardından o sonucu deneyebilirsiniz.

# Saving your files with UTF-8 encoding[¶](https://www.kaggle.com/code/alexisbcook/character-encodings" \l "Saving-your-files-with-UTF-8-encoding" \t "_self)

Son olarak, dosyanızı UTF-8 formatına dönüştürmek için gereken tüm zahmeti çektikten sonra, muhtemelen bu şekilde kalmasını isteyeceksiniz. Bunu yapmanın en kolay yolu, dosyalarınızı UTF-8 kodlamasıyla kaydetmektir. İyi haber şu ki, UTF-8 Python'da standart kodlama olduğundan, bir dosyayı kaydettiğinizde varsayılan olarak UTF-8 olarak kaydedilecektir:

*# save our file (will be saved as UTF-8 by default!)*

kickstarter\_2016.to\_csv("ks-projects-201801-utf8.csv")