Специалисты по обработке данных тратят много времени на очистку наборов данных и приведение их в форму, с которой они могут работать. Фактически, многие специалисты по данным утверждают, что начальные шаги по получению и очистке данных составляют 80% работы. Именно об этом мы и поговорим в данной главе. Но перед этим, нам необходимо установить и импортировать две библиотеки:

import pandas as pd  
import numpy as np

Кроме этого, в проекте мы создаем папку Datasets, куда размещаем наши исходные файлы, с которыми будет происходить работа. В моем случае это: **BL-Flickr-Images-Book.csv, olympics.csv и university\_towns.txt. Проделав все эти действия, можно приступать к работе.**

## Удаление столбцов в DataFrame

## Часто вы обнаруживаете, что не все категории данных в наборе данных вам нужны. Именно поэтому, библиотека Pandas предоставляет удобный способ удаления ненужных столбцов или строк из DataFrame с помощью функции drop(). Давайте посмотрим на простой пример, в котором мы удаляем несколько столбцов из DataFrame. В приведенных ниже примерах мы передаем относительный путь к pd.read\_csv, что означает, что все наборы данных находятся в папке с именем ****Datasets**** в нашем текущем рабочем каталоге:

df = pd.read\_csv('Datasets/BL-Flickr-Images-Book.csv')  
print('Вывод загруженного csv файла:')  
print(df.head())

## Результат:

## 

Сделав вывод нашего csv файла мы видим, что несколько столбцов предствляют собой вспомогательную информацию, которая была бы полезна для библиотеки, но не очень для описания самой книги: Edition Statement, Corporate Author, Corporate Contributors, Former owner, Engraver, Issuance type и Shelfmarks. Эту информацию мы можем удалить следующим образом:

to\_drop = ['Edition Statement',  
 'Corporate Author',  
 'Corporate Contributors',  
 'Former owner',  
 'Engraver',  
 'Contributors',  
 'Issuance type',  
 'Shelfmarks']  
  
df.drop(to\_drop, inplace=True, axis=1)  
print('Вывод csv файла с удаленными столбцами:')  
print(df.head())

Сначала мы определили список, который содержит имена всех столбцов, которые мы хотим удалить. Затем мы вызываем функцию drop() для нашего объекта, передавая параметр inplace как True и параметр оси как 1, что говорит Pandas об изменениях непосредственно в нашем объекте и что он должен искать значения, которые будут отброшены в столбцах объекта. Результат:



## Изменение индекса фрейма данных

## Индекс Pandas расширяет функциональность массивов NumPy, чтобы обеспечить более гибкое нарезание и маркировку. Во многих случаях полезно использовать однозначное идентифицирующее поле данных в качестве индекса. Давайте заменим существующий индекс в **BL-Flickr-Images-Book.csv** столбцом Identifier, используя set\_index:

df = df.set\_index('Identifier')  
print(' Замена существующего индекса столбцом Identifier:')  
print(df.head())

## Результат:

## 

## Кроме этого, мы можем получить доступ к каждой записи простым способом с помощью loc[]. Хотя loc[] может не иметь всего этого интуитивно понятного имени, он позволяет нам выполнять индексацию на основе меток, которая представляет собой маркировку строки или записи независимо от ее положения:

print('Получение доступа к каждой записи:')  
print(df.loc[206])

## Результат:

## 

## Другими словами, 206 — это первая метка индекса. Ранее нашим индексом был RangeIndex: целые числа, начинающиеся с 0, аналог встроенного диапазона Python. Передав имя столбца в set\_index, мы изменили индекс на значения в Identifier.

## Очистка полей в данных

## Пока что мы удалили ненужные столбцы и изменили индекс нашего DataFrame на что-то более разумное. В этом разделе мы очистим определенные столбцы и приведем их к единому формату, чтобы лучше понять набор данных и обеспечить согласованность. В частности, мы будем очищать дату публикации и место публикации. Давайте выведем поле, содержащее дату публикации, чтобы мы могли выполнять вычисления в будущем:

print('Вывод поля даты публикации для того, чтобы мы могли выполнять вычисления в будущем')  
print(df.loc[1905:, 'Date of Publication'].head(10))

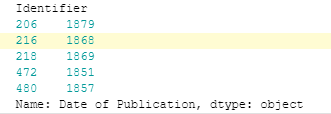
## Результат:

## 

Как известно, у конкретной книги может быть только одна дата публикации. Поэтому нам необходимо удалить лишние даты в квадратных скобках, преобразовать диапазоны дат в их «дату начала», полностью удалить даты, в которых мы не уверены и преобразовать строку nan в значение NaN NumPy. Для этого мы будем использовать следующее регулярное выражение: regex = r'^(\d{4})'. Данное выражение предназначено для поиска любых четырех цифр в начале строки, чего достаточно для нашего случая. Это необработанная строка, что является стандартной практикой с регулярными выражениями. \d представляет любую цифру, а {4} повторяет это правило четыре раза. Символ ^ соответствует началу строки, а круглые скобки обозначают группу захвата, которая сигнализирует Pandas, что мы хотим извлечь эту часть регулярного выражения. Сам код:

extr = df['Date of Publication'].str.extract(r'^(\d{4})', expand=False)  
print('Модернизированные поля даты публикации:')  
print(extr.head())

Результат:



## Объединение методов str с NumPy для очистки столбцов

## Для начала, давайте выведем содержимое столбца Place of Publication:

print('Вывод содержимого столбца Place of Publication')  
print(df['Place of Publication'].head(10))

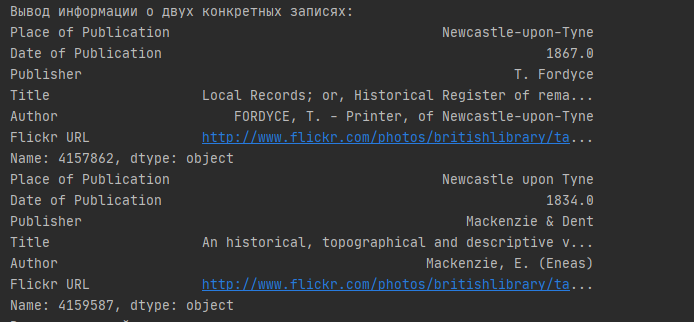
## Результат:

## 

Мы видим, что для некоторых строк место публикации окружено другой ненужной информацией. Если бы мы посмотрели на большее количество значений, мы бы увидели, что это справедливо только для некоторых строк, место публикации которых — ‘London’ или ‘Oxford’. Давайте взглянем на две конкретные записи:

print('Вывод информации о двух конкретных записях:')  
print(df.loc[4157862])  
print(df.loc[4159587])

Результат:



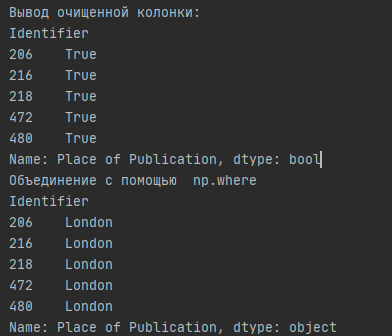
Эти две книги были изданы в одном месте, но одна имеет дефис в названии места, а другая — нет. Чтобы очистить этот столбец за один проход, мы можем использовать str.contains() для получения логической маски. Чистим колонку следующим образом:

pub = df['Place of Publication']  
london = pub.str.contains('London')  
print('Вывод очищенной колонки:')  
print(london[:5])  
oxford = pub.str.contains('Oxford')

Далее объединяем их с помощью np.where:

df['Place of Publication'] = np.where(london, 'London',  
 np.where(oxford, 'Oxford',  
 pub.str.replace('-', ' ')))  
print('Объединение с помощью np.where')  
print(df['Place of Publication'].head())

Результат:



Здесь функция np.where вызывается во вложенной структуре с условием, представляющим собой серию логических значений, полученных с помощью str.contains(). Метод contains() работает аналогично встроенному ключевому [слову in](https://www.programiz.com/python-programming/keyword-list#in), используемому для поиска вхождения объекта в итерируемом объекте (или подстроке в строке). Используемая замена — это строка, представляющая желаемое место публикации. Мы также заменяем дефисы пробелом с помощью str.replace() и переназначаем столбец в нашем **DataFrame.**

## Очистка всего набора данных с помощью функции applymap

В определенных ситуациях вы увидите, что «грязь» не локализована в одном столбце, а более разбросана. В некоторых случаях было бы полезно применить настраиваемую функцию к каждой ячейке или элементу DataFrame. Метод Pandas .applymap() похож на метод in-построил функцию map() и просто применяет функцию ко всем элементам в DataFrame. Давайте посмотрим на пример. Мы создадим DataFrame из ранее добавленного в проект файла «university\_towns.txt»:

$ head Datasets/univerisity\_towns.txt

Alabama[edit]

Auburn (Auburn University)[1]

Florence (University of North Alabama)

Jacksonville (Jacksonville State University)[2]

Livingston (University of West Alabama)[2]

Montevallo (University of Montevallo)[2]

Troy (Troy University)[2]

Tuscaloosa (University of Alabama, Stillman College, Shelton State)[3][4]

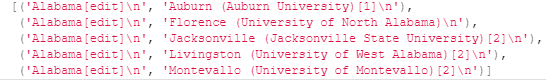
Tuskegee (Tuskegee University)[5]

Alaska[edit]

Мы видим, что у нас есть периодические названия штатов, за которыми следуют университетские города в этом штате: StateA TownA1 TownA2 StateB TownB1 TownB2 …. Если мы посмотрим на то, как названия штатов записаны в файле, мы увидим, что все они имеют в них подстрока [edit]. Мы можем воспользоваться этим шаблоном, создав список (state, city) кортежи и обертывание этого списка в DataFrame:

university\_towns = []  
with open('Datasets/university\_towns.txt') as file:  
 for line in file:  
 if '[edit]' in line:  
  
 state = line  
 else:  
  
 university\_towns.append((state, line))  
print('Вывод созданного списка, преобразованного в DataFrame:')  
print(university\_towns[:5])

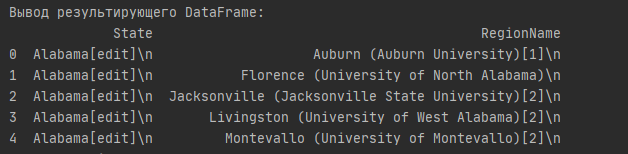
Результат:



Мы можем обернуть этот список в DataFrame и установить столбцы как «State» и «RegionName». Pandas возьмет каждый элемент в списке и установит State на левое значение, а RegionName — на правое значение:

towns\_df = pd.DataFrame(university\_towns,  
 columns=['State', 'RegionName'])  
print('Вывод результирующего DataFrame:')  
print(towns\_df.head())

Результат:



Хотя мы могли бы очистить эти строки в цикле for выше, Pandas упрощает это. Нам нужно только название штата и название города, а все остальное можно удалить. Хотя здесь мы могли бы снова использовать методы Pandas .str(), мы также могли бы использовать applymap() для сопоставления вызываемого Python с каждым элементом DataFrame.

## Переименование столбцов и пропуск строк

Часто наборы данных, с которыми вы будете работать, будут иметь либо имена столбцов, которые непросто понять, либо неважную информацию в первых нескольких и/или последних строках, такую ​​как определения терминов в наборе данных или сноски. В этом случае, мы хотели бы переименовать столбцы и пропустить определенные строки, чтобы можно было перейти к необходимой информации с помощью правильных и понятных меток. Чтобы продемонстрировать, как это сделать, давайте сначала взглянем на первые пять строк все также ранее добавленного набора данных **olympics.csv**:

$ head -n 5 Datasets/olympics.csv

0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

,? Summer,01 !,02 !,03 !,Total,? Winter,01 !,02 !,03 !,Total,? Games,01 !,02 !,03 !,Combined total

Afghanistan (AFG),13,0,0,2,2,0,0,0,0,0,13,0,0,2,2

Algeria (ALG),12,5,2,8,15,3,0,0,0,0,15,5,2,8,15

Argentina (ARG),23,18,24,28,70,18,0,0,0,0,41,18,24,28,70

Теперь мы прочитаем его в DataFrame Pandas:

olympics\_df = pd.read\_csv('Datasets/olympics.csv')  
print('Вывод olympics.csv:')  
print(olympics\_df.head())

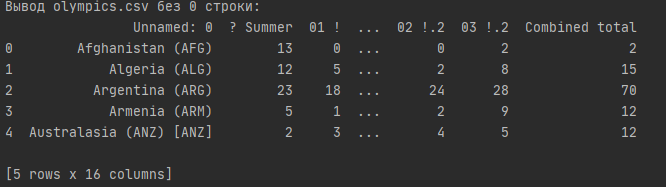
## Результат:

## 

Это действительно грязно! Поэтому, мы должны пропустить одну строку и установить заголовок как первую (с нулевым индексом) строку и переименовать столбцы. Для того, чтобы удалить 0-ю строку мы используем:

olympics\_df = pd.read\_csv('Datasets/olympics.csv', header=1)  
print('Вывод olympics.csv без 0 строки:')  
print(olympics\_df.head())

Результат:



Теперь у нас есть правильная строка, установленная в качестве заголовка, и все ненужные строки удалены. Обратите внимание на то, как Pandas изменил имя столбца, содержащего названия стран, с NaN на Unnamed: 0. Чтобы переименовать столбцы, мы будем использовать метод rename() DataFrame, который позволяет вам изменить метку оси на основе сопоставления (в данном случае dict). Начнем с определения словаря, который сопоставляет текущие имена столбцов (как ключи) с более удобными (значениями словаря):

new\_names = {'Unnamed: 0': 'Country',  
 '? Summer': 'Summer Olympics',  
 '01 !': 'Gold',  
 '02 !': 'Silver',  
 '03 !': 'Bronze',  
 '? Winter': 'Winter Olympics',  
 '01 !.1': 'Gold.1',  
 '02 !.1': 'Silver.1',  
 '03 !.1': 'Bronze.1',  
 '? Games': '# Games',  
 '01 !.2': 'Gold.2',  
 '02 !.2': 'Silver.2',  
 '03 !.2': 'Bronze.2'}

Далее вызываем  функцию rename() для нашего объекта:

olympics\_df.rename(columns=new\_names, inplace=True)

Установка inplace в True указывает, что наши изменения будут внесены непосредственно в объект. Результат:

