**# Welcome to Knowledge Check 2!**

We're going to clean some data today. I would guess about 70% of data science is cleaning and wrangling data, so we want to make sure you can do it. I haven't looked through this data set at all, so we'll see if we can find some issues to clean up. Here's what you need to do to complete the knowledge check:

1. Make a .py (or .ipynb) file that contains the following (your choice of editor does not matter!) and do the following:

- find and access a data set in any way you want. You can use an API, a CSV, anything.

- Fix TWO issues with the data set using techniques you've learned in class. Here are some common fixes:

  - Remove null values

  - Fill in null values with 0's or blanks

  - fill in blanks

  - fix character strings that aren't formatted correctly (you could use regex for this)

  - correct column names if they're misnamed

  - correct spelling (for example, you might have a Country column with an entry that says "Unted States of America".)

  -  There are hundreds of other things you could fix depending on the issues, so don't worry about whether or not your fix "counts" for the check. It most likely will if you're fixing something.

2. Commit your changes.

3. Push your changes to GitHub, and make sure to turn in the GitHub link into Google Classroom.

**## Cleaning Data**

Data taken from [Louisville Metro Data](https://data.louisvilleky.gov/dataset/parking-citations). This parking data set from 2018 looked interesting and I haven't looked at it before so let's get started. The interesting thing about data analysis is that you don't have to look at just raw science topics, you can look at sociological data to answer questions about your community and area. For example, this data set is about parking citations. If there were a ton of citations clumped together geographically, that might inform your decision to add more parking to the area.

import pandas as pd

import numpy as np

I usually check the shape of the data and print the first few rows so I know what I'm looking at. Maybe the columns as well.

print(df.shape)

df.head()

(20336, 8)

|  | **Cite Number** | **Issue Date** | **Violat** | **Sublocatio** | **Street** | **Meter #** | **Is Wa** | **Due** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | M201507360 | 02/16/2018 | PSW | 700 | YORK ST | NaN | NO | 0.0 |
| 1 | M201306003 | 01/05/2018 | HO | 100S | 1ST ST | NaN | NO | 0.0 |
| 2 | M202705546 | 01/18/2018 | X | N200 | 1ST ST | 3012 | NO | 0.0 |
| 3 | M202802800 | 01/10/2018 | NPP | 500W | LIBERTY ST | NaN | NO | 0.0 |
| 4 | M200208507 | 02/23/2018 | NP | 200E | CHESTNUT ST | NaN | NO | 0.0 |

Index(['Cite Number', 'Issue Date', 'Violat', 'Sublocatio', 'Street',

'Meter #', 'Is Wa', 'Due'],

dtype='object')

Index(['citation\_number', 'issue\_date', 'violation', 'sublocation', 'street',

'meter\_number', 'is\_warning', 'due'],

dtype='object')

df.columns

**Index(['Cite Number', 'Issue Date', 'Violat', 'Sublocatio', 'Street',**

**'Meter #', 'Is Wa', 'Due'],**

**dtype='object')**

(20336, 8)

We have an issue already! The column names are misspelled, or at least, they were truncated at some point in the creation of the set. If you look at the link above, you can see the data dictionary of what these values mean, so I'm going to fix them accordingly. Of course, you could just reference these incorrect names in the code but it might get confusing.

fixed\_columns = {

    'Cite Number':'citation\_number',

    'Issue Date':'issue\_date',

    'Violat':'violation',

    'Sublocatio':'sublocation',

    'Street':'street',

    'Meter #':'meter\_number',

    'Is Wa':'is\_warning',

    'Due':'due'

}

df.rename(columns=fixed\_columns,inplace=True)

df.columns

Index(['citation\_number', 'issue\_date', 'violation', 'sublocation', 'street',

'meter\_number', 'is\_warning', 'due'],

dtype='object')

Great! Our fix worked. Python mostly uses snake case, the typing style that's like\_this, which has underscores and lowercase. It's not stricly necessary but I'm used to it. Notice I also used ```inplace=True```. That modified the data set without having to set a new variable.

Sometimes it's useful to look at unique values, like this:

df.street.unique()

array(['YORK ST', '1ST ST', 'LIBERTY ST', 'CHESTNUT ST',

'MUHAMMAD ALI BLVD', 'BROOK ST', 'MAIN ST', 'FLOYD ST',

'BARDSTOWN RD', 'JEFFERSON ST', 'BAXTER AVE', '8TH ST', '6TH ST',

'4TH ST', 'MARKET ST', '7TH ST', '5TH ST', 'HIGHLAND AVE',

'WASHINGTON ST', 'BROADWAY', '3RD ST', 'DOUGLASS BLVD',

'DUNDEE RD', 'KENTUCKY ST', 'ABRAHAM FLEXNER WAY', 'LEE ST',

'BLOOM ST', '2ND ST', 'GRAY ST', 'ARMORY PL', 'PRESTON ST',

'ELLWOOD AVE', 'BRANDYWYNE DR', 'BARRET AVE', 'LUCIA AVE',

'HARVARD DR', 'EVERETT AVE', 'GUTHRIE ST', 'COLORADO AVE',

'LIBRARY PL', 'HANCOCK ST', 'GAULBERT AVE', 'CARDINAL BLVD',

'CEDAR ST', 'GARAGE - 6TH & MAIN', 'RODMAN ST', 'CLAY ST',

'RUBEL AVE', 'STORY AVE', 'CRITTENDEN DR', 'JACKSON ST',

'WEBSTER ST', 'HILL ST', '10TH ST', 'STANSBURY PARK',

'LIBRARY LOT', 'DEERWOOD AVE', 'WITHERSPOON ST', 'ARTHUR ST',

'BRONNER CIR', 'BRADLEY AVE', 'EDENSIDE AVE',

'COVENTRY COMMONS DR', 'ST ANTHONY PL', 'YALE DR', 'PATTERSON AVE',

'SARATOGA DR', 'GARAGE - MARKET ST', 'BRANDEIS AVE',

'EDGELAND AVE', 'BARINGER AVE', 'LILLY AVE', 'ALTA AVE',

'BEECHWOOD AVE', 'CAWTHON ST', 'LOT - CITY HALL',

'BONNYCASTLE AVE', 'SHERWOOD AVE', 'LOT - MUD', 'JACKSON AVE',

'BRYNWOOD LN', 'MELLWOOD AVE', 'EDEN AVE', 'BICKEL AVE',

'GARAGE - MUHAMMAD ALI CENTER', 'HAMES TRCE', 'READING RD',

'GARAGE - SEELBACH', 'SPRING ST', 'FORT ST', '9TH ST', 'SPEED AVE',

'HAROLD AVE', "PLACE D'OR null", 'FLEMING AVE', 'FISK CT',

'LIBERTY AVE', 'LAUDERDALE RD', 'JEFFERSON AVE', 'OUTER LOOP',

'BUNGALOW DR', 'BRINKEY WAY', 'HIKES LN', 'PAYNE ST',

'WATTERSON TRL', 'PLACE NOIR null', 'WOODLAND AVE',

'PLACE ROUGE null', 'CHARLBURY CT', 'MARY DELL LN', 'CAWOOD DR',

'CRAIG AVE', 'RIEDLING DR', 'BEECH AVE', 'WALTER AVE'],

dtype=object)

*Output is truncated. View as a* [*scrollable element*](command:cellOutput.enableScrolling?189983d7-1d67-4e53-ae48-0cac7f46778c) *or open in a* [*text editor*](command:workbench.action.openLargeOutput?189983d7-1d67-4e53-ae48-0cac7f46778c)*. Adjust cell output* [*settings*](command:workbench.action.openSettings?%5B%22%40tag%3AnotebookOutputLayout%22%5D)*...*

Confusingly, some of those just have the word 'null' in them. This is not the same as a null value, like the ```np.NaN``` values you see under meter number, these are just misplaced text. Let's try to remove them.

df[df.street.str.contains('null')].head()

|  | **citation\_number** | **issue\_date** | **violation** | **sublocation** | **street** | **meter\_number** | **is\_warning** | **due** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1559 | 94783496 | 01/19/2018 | EX | 1200 | PLACE D'OR null | NaN | NO | 50.0 |
| 1689 | 94942481 | 01/15/2018 | U | I-265 | I 71 NORTH null | NaN | NO | 50.0 |
| 1750 | 94941571 | 02/20/2018 | NP | 12MM | I 264 EAST null | NaN | NO | 50.0 |
| 1978 | 94725046 | 01/11/2018 | NPJ | 23.2 MM | I 64 EAST null | NaN | NO | 50.0 |
| 1990 | 94944872 | 02/06/2018 | EX | 9MM | I 264 WEST null | NaN | NO | 50.0 |

df.street.replace({' null':''}, regex=True,inplace=True)

df[df.street.str.contains('null')].head()

| **citation\_number** | **issue\_date** | **violation** | **sublocation** | **street** | **meter\_number** | **is\_warning** | **due** |
| --- | --- | --- | --- | --- | --- | --- | --- |

Hurray! When we tried the command again to find the bad values that contained the string ' null', there weren't any. Above. we used Pandas ability to use regex to find anything that matched ' null' and replace it with '' which is nothing. Notice also that I have a space in ' null'. This is so we didn't end up with a trailing white space, which can be very frustrating to deal with if you don't remember to check or trim the values.

These are a couple quick examples of cleaning. Remember, you can do basically anything you want to fulfill this knowledge check as long as you perform two cleaning operations on any set of data.