

Membership constraints

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @DataCamp

Chapter 2 - Text and categorical data problems

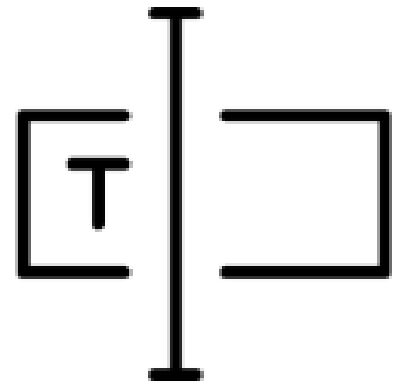
Categories and membership constraints

Predefined finite set of categories

Type of data	Example values	Numeric representation
Marriage Status	unmarried , married	0 , 1
Household Income Category	0-20K , 20-40K , ...	0 , 1 , ..
Loan Status	default , payed , no_loan	0 , 1 , 2

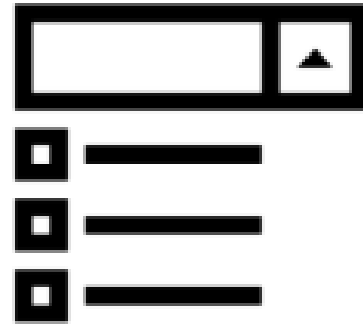
*Marriage status can **only** be* unmarried _or_ married

Why could we have these problems?



Free text

Or



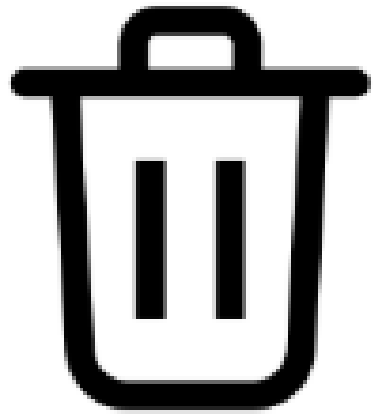
Dropdowns

Data Entry Errors

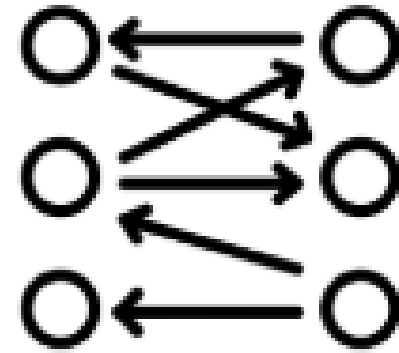


Parsing Errors

How do we treat these problems?



***Dropping
Data***



***Remapping
Categories***



***Inferring
Categories***

An example

```
# Read study data and print it
study_data = pd.read_csv('study.csv')
study_data
```

	name	birthday	blood_type
1	Beth	2019-10-20	B-
2	Ignatius	2020-07-08	A-
3	Paul	2019-08-12	O+
4	Helen	2019-03-17	O-
5	Jennifer	2019-12-17	Z+
6	Kennedy	2020-04-27	A+
7	Keith	2019-04-19	AB+

```
# Correct possible blood types
categories
```

	blood_type
1	O-
2	O+
3	A-
4	A+
5	B+
6	B-
7	AB+
8	AB-

An example

```
# Read study data and print it
study_data = pd.read_csv('study.csv')
study_data
```

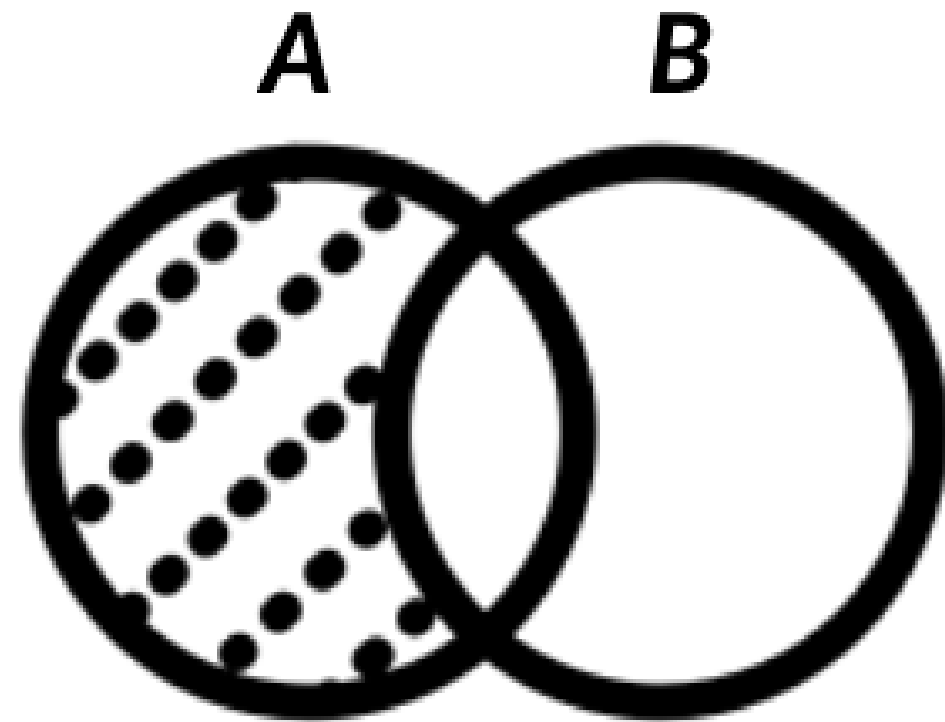
	name	birthday	blood_type
1	Beth	2019-10-20	B-
2	Ignatius	2020-07-08	A-
3	Paul	2019-08-12	O+
4	Helen	2019-03-17	O-
5	Jennifer	2019-12-17	Z+ <--
6	Kennedy	2020-04-27	A+
7	Keith	2019-04-19	AB+

```
# Correct possible blood types
categories
```

	blood_type
1	O-
2	O+
3	A-
4	A+
5	B+
6	B-
7	AB+
8	AB-

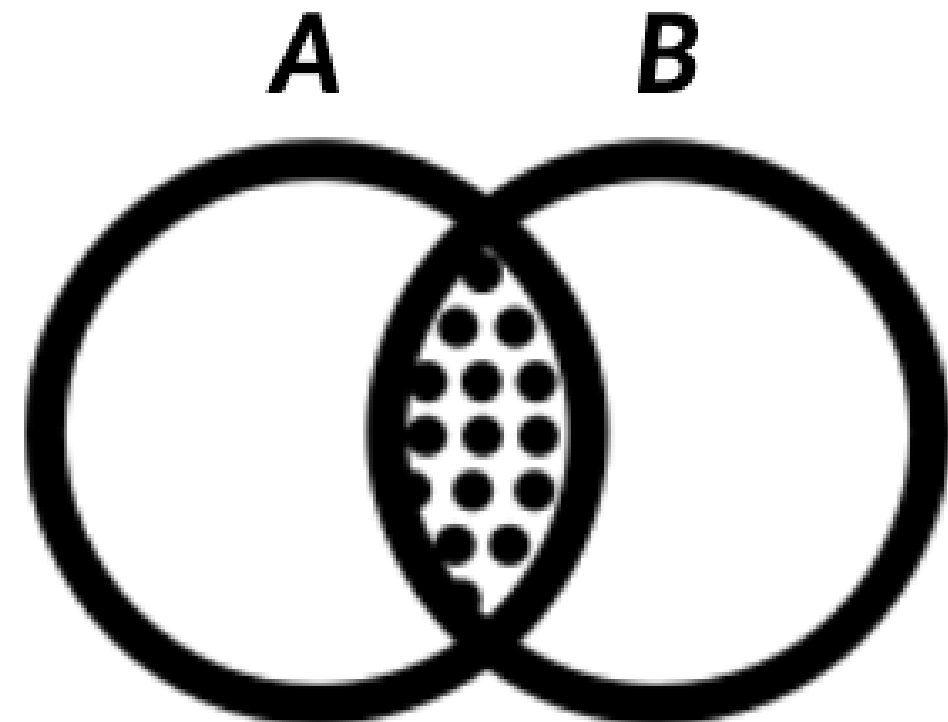
A note on joins

Anti Joins



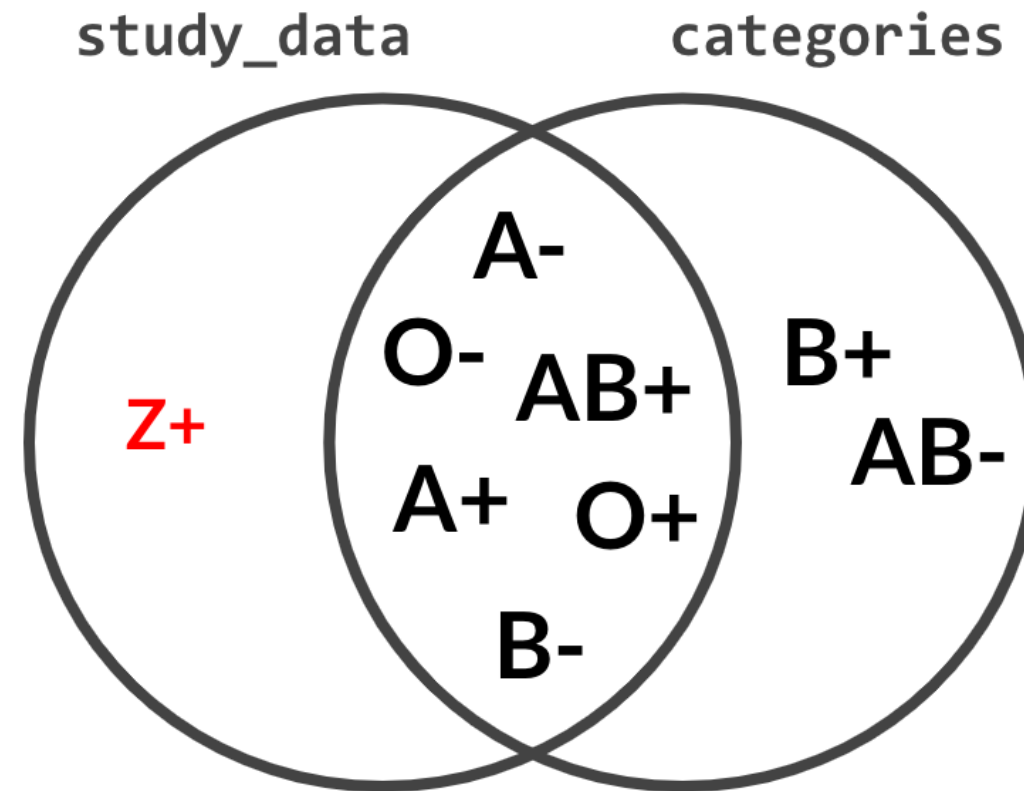
What is in A and not in B

Inner Joins



What is in both A and B

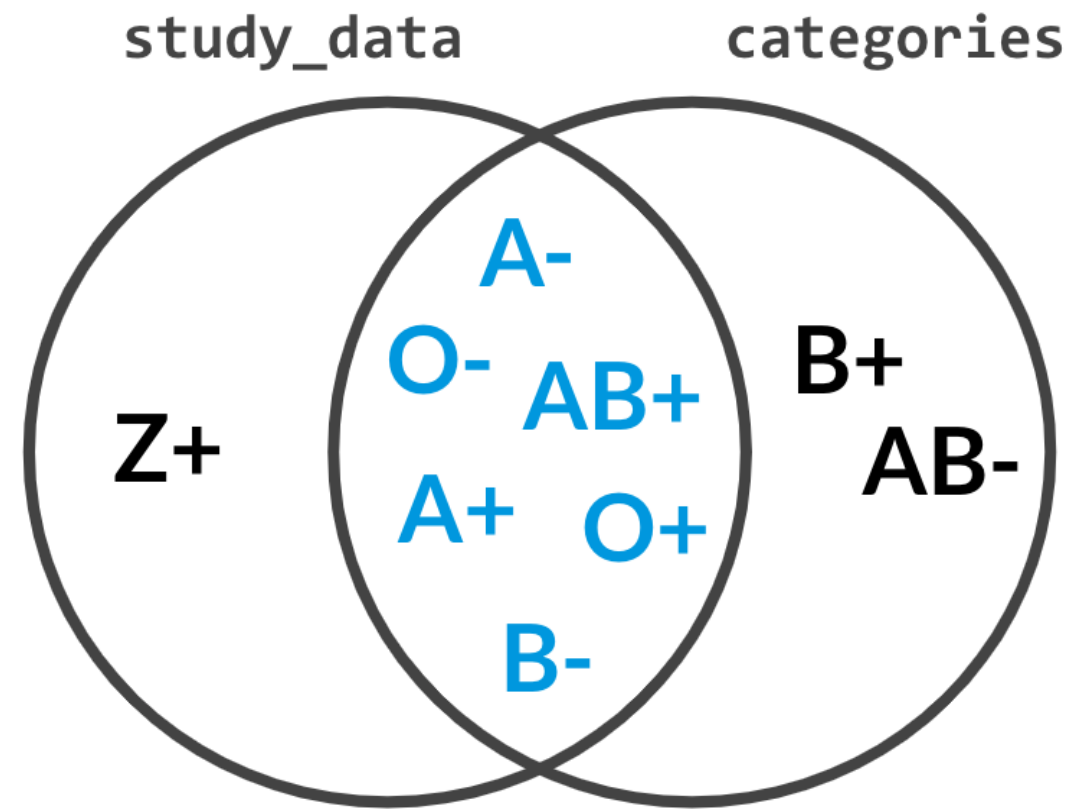
A left anti join on blood types



What is in study_data only

*Returns only rows
containing Z+*

An inner join on blood types



What is in `study_data` and `categories` only

*Returns all the rows except those
containing **Z+**, **B+** and **AB-***

Finding inconsistent categories

```
inconsistent_categories = set(study_data['blood_type']).difference(categories['blood_type'])  
print(inconsistent_categories)
```

```
{'Z+'}
```

```
# Get and print rows with inconsistent categories  
inconsistent_rows = study_data['blood_type'].isin(inconsistent_categories)  
study_data[inconsistent_rows]
```

```
   name  birthday blood_type  
5 Jennifer 2019-12-17      Z+
```

Dropping inconsistent categories

```
inconsistent_categories = set(study_data['blood_type']).difference(categories['blood_type'])
inconsistent_rows = study_data['blood_type'].isin(inconsistent_categories)
inconsistent_data = study_data[inconsistent_rows]
# Drop inconsistent categories and get consistent data only
consistent_data = study_data[~inconsistent_rows]
```

	name	birthday	blood_type
1	Beth	2019-10-20	B-
2	Ignatius	2020-07-08	A-
3	Paul	2019-08-12	O+
4	Helen	2019-03-17	O-
...

Let's practice!

CLEANING DATA IN PYTHON

Categorical variables

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @DataCamp

What type of errors could we have?

I) Value inconsistency

- *Inconsistent fields:* 'married' , 'Married' , 'UNMARRIED' , 'not married' ..
- *_Trailing white spaces:* _ 'married ' , ' married ' ..

II) Collapsing too many categories to few

- *Creating new groups:* 0-20K , 20-40K categories ... from continuous household income data
- *Mapping groups to new ones:* Mapping household income categories to 2 'rich' , 'poor'

III) Making sure data is of type category (seen in Chapter 1)

Value consistency

Capitalization: 'married', 'Married', 'UNMARRIED', 'unmarried' ..

```
# Get marriage status column
marriage_status = demographics['marriage_status']
marriage_status.value_counts()
```

```
unmarried    352
married      268
MARRIED      204
UNMARRIED    176
dtype: int64
```


Value consistency

```
# Get value counts on DataFrame  
marriage_status.groupby('marriage_status').count()
```

	household_income	gender
marriage_status		
MARRIED	204	204
UNMARRIED	176	176
married	268	268
unmarried	352	352

Value consistency

```
# Capitalize
```

```
marriage_status['marriage_status'] = marriage_status['marriage_status'].str.upper()  
marriage_status['marriage_status'].value_counts()
```

```
UNMARRIED    528  
MARRIED      472
```

```
# Lowercase
```

```
marriage_status['marriage_status'] = marriage_status['marriage_status'].str.lower()  
marriage_status['marriage_status'].value_counts()
```

```
unmarried    528  
married      472
```

Value consistency

Trailing spaces: 'married ', 'married', 'unmarried', ' unmarried' ..

```
# Get marriage status column
marriage_status = demographics['marriage_status']
marriage_status.value_counts()
```

```
unmarried    352
unmarried    268
married      204
married      176
dtype: int64
```

Value consistency

```
# Strip all spaces
demographics = demographics['marriage_status'].str.strip()
demographics['marriage_status'].value_counts()
```

```
unmarried    528
married      472
```

Collapsing data into categories

Create categories out of data: `income_group` column from `income` column.

```
# Using qcut()
import pandas as pd
group_names = ['0-200K', '200K-500K', '500K+']
demographics['income_group'] = pd.qcut(demographics['household_income'], q = 3,
                                       labels = group_names)

# Print income_group column
demographics[['income_group', 'household_income']]
```

```
category household_income
0  200K-500K    189243
1      500K+    778533
..
```

Collapsing data into categories

Create categories out of data: `income_group` column from `income` column.

```
# Using cut() - create category ranges and names
ranges = [0,200000,500000,np.inf]
group_names = ['0-200K', '200K-500K', '500K+']
# Create income group column
demographics['income_group'] = pd.cut(demographics['household_income'], bins=ranges,
                                      labels=group_names)
demographics[['income_group', 'household_income']]
```

	category	Income
0	0-200K	189243
1	500K+	778533

Collapsing data into categories

Map categories to fewer ones: reducing categories in categorical column.

operating_system column is: 'Microsoft', 'MacOS', 'IOS', 'Android', 'Linux'

operating_system column should become: 'DesktopOS', 'MobileOS'

```
# Create mapping dictionary and replace
mapping = {'Microsoft': 'DesktopOS', 'MacOS': 'DesktopOS', 'Linux': 'DesktopOS',
          'IOS': 'MobileOS', 'Android': 'MobileOS'}
devices['operating_system'] = devices['operating_system'].replace(mapping)
devices['operating_system'].unique()
```

```
array(['DesktopOS', 'MobileOS'], dtype=object)
```

Let's practice!

CLEANING DATA IN PYTHON

Cleaning text data

CLEANING DATA IN PYTHON



Adel Nehme

VP of AI Curriculum, DataCamp

What is text data?

Type of data	Example values
Names	Alex , Sara ...
Phone numbers	+96171679912 ...
Emails	`adel@datacamp.com`..
Passwords	...

Common text data problems

1) *Data inconsistency:*

+96171679912 or 0096171679912 or ..?

2) *Fixed length violations:*

Passwords needs to be at least 8 characters

3) *Typos:*

+961.71.679912

Example

```
phones = pd.read_csv('phones.csv')  
print(phones)
```

	Full name	Phone number
0	Noelani A. Gray	001-702-397-5143
1	Myles Z. Gomez	001-329-485-0540
2	Gil B. Silva	001-195-492-2338
3	Prescott D. Hardin	+1-297-996-4904
4	Benedict G. Valdez	001-969-820-3536
5	Reece M. Andrews	4138
6	Hayfa E. Keith	001-536-175-8444
7	Hedley I. Logan	001-681-552-1823
8	Jack W. Carrillo	001-910-323-5265
9	Lionel M. Davis	001-143-119-9210

Example

```
phones = pd.read_csv('phones.csv')
print(phones)
```

	Full name	Phone number	
0	Noelani A. Gray	001-702-397-5143	
1	Myles Z. Gomez	001-329-485-0540	
2	Gil B. Silva	001-195-492-2338	
3	Prescott D. Hardin	+1-297-996-4904	<-- Inconsistent data format
4	Benedict G. Valdez	001-969-820-3536	
5	Reece M. Andrews	4138	<-- Length violation
6	Hayfa E. Keith	001-536-175-8444	
7	Hedley I. Logan	001-681-552-1823	
8	Jack W. Carrillo	001-910-323-5265	
9	Lionel M. Davis	001-143-119-9210	

Example

```
phones = pd.read_csv('phones.csv')  
print(phones)
```

	Full name	Phone number
0	Noelani A. Gray	0017023975143
1	Myles Z. Gomez	0013294850540
2	Gil B. Silva	0011954922338
3	Prescott D. Hardin	0012979964904
4	Benedict G. Valdez	0019698203536
5	Reece M. Andrews	NaN
6	Hayfa E. Keith	0015361758444
7	Hedley I. Logan	0016815521823
8	Jack W. Carrillo	0019103235265
9	Lionel M. Davis	0011431199210

Fixing the phone number column

```
# Replace "+" with "00"
phones["Phone number"] = phones["Phone number"].str.replace("+", "00")
phones
```

	Full name	Phone number
0	Noelani A. Gray	001-702-397-5143
1	Myles Z. Gomez	001-329-485-0540
2	Gil B. Silva	001-195-492-2338
3	Prescott D. Hardin	001-297-996-4904
4	Benedict G. Valdez	001-969-820-3536
5	Reece M. Andrews	4138
6	Hayfa E. Keith	001-536-175-8444
7	Hedley I. Logan	001-681-552-1823
8	Jack W. Carrillo	001-910-323-5265
9	Lionel M. Davis	001-143-119-9210

Fixing the phone number column

```
# Replace "-" with nothing
phones["Phone number"] = phones["Phone number"].str.replace("-", "")
phones
```

	Full name	Phone number
0	Noelani A. Gray	0017023975143
1	Myles Z. Gomez	0013294850540
2	Gil B. Silva	0011954922338
3	Prescott D. Hardin	0012979964904
4	Benedict G. Valdez	0019698203536
5	Reece M. Andrews	4138
6	Hayfa E. Keith	0015361758444
7	Hedley I. Logan	0016815521823
8	Jack W. Carrillo	0019103235265
9	Lionel M. Davis	0011431199210

Fixing the phone number column

```
# Replace phone numbers with lower than 10 digits to NaN
digits = phones['Phone number'].str.len()
phones.loc[digits < 10, "Phone number"] = np.nan
phones
```

	Full name	Phone number
0	Noelani A. Gray	0017023975143
1	Myles Z. Gomez	0013294850540
2	Gil B. Silva	0011954922338
3	Prescott D. Hardin	0012979964904
4	Benedict G. Valdez	0019698203536
5	Reece M. Andrews	NaN
6	Hayfa E. Keith	0015361758444
7	Hedley I. Logan	0016815521823
8	Jack W. Carrillo	0019103235265
9	Lionel M. Davis	0011431199210

Fixing the phone number column

```
# Find length of each row in Phone number column
sanity_check = phone['Phone number'].str.len()
```

```
# Assert minmum phone number length is 10
assert sanity_check.min() >= 10
```

```
# Assert all numbers do not have "+" or "-"
assert phone['Phone number'].str.contains("+|-").any() == False
```

Remember, `assert` returns nothing if the condition passes

But what about more complicated examples?

```
phones.head()
```

	Full name	Phone number
0	Olga Robinson	+(01706)-25891
1	Justina Kim	+0500-571437
2	Tamekah Henson	+0800-1111
3	Miranda Solis	+07058-879063
4	Caldwell Gilliam	+(016977)-8424

Supercharged control + F

Regular expressions in action

```
# Replace letters with nothing
phones['Phone number'] = phones['Phone number'].str.replace(r'\D+', '')
phones.head()
```

```
      Full name Phone number
0   Olga Robinson  0170625891
1   Justina Kim    0500571437
2  Tamekah Henson   08001111
3  Miranda Solis   07058879063
4 Caldwell Gilliam  0169778424
```

Let's practice!

CLEANING DATA IN PYTHON