Cleaning Data

1405 Instructor: Ruiqing (Sam) Cao

Review: Data Sources & Data Cleaning

- Common data sources
  - Tabular Data (e.g., *.csv, *.dta) read directly from your local disk
  - Data from APIs or web sources, often in formats like JSON or HTML.
  - Data from SQL databases or large-scale cloud-based storage systems.
  - Data created by your programs, e.g., through string parsing or scraping

→ Raw data is often unstructured or incomplete, **requiring substantial cleaning** before use. Even relatively clean data must be **filtered and transformed** at the right level for meaningful analysis.

→ In reality, a large portion of time is spent ensuring **data integrity and quality**, often more than performing analytics or machine learning tasks.

Motivating Task

- Below is a user registration dataset from a FinTech payment company

- We want to obtain the number of user sign-ups every month 2011–2019

This requires us to

- Clean the data and ensure every id is linked to one user only
- Aggregate the data at the appropriate level

primary key

first_name	is_group	id	date_joined
Sion	False	2082497001160704615	2016-11-13T07:09:48.000Z
Kari	False	2538731244355584742	2018-08-04T18:45:48.000Z
Jessie	False	1921315569139712500	2016-04-04T21:51:04.000Z
Dion	False	2019594260709376161	2016-08-18T12:13:17.000Z
Alec	False	20808953306808325 65	2016-11-11T02:07:34.000Z
...	...	...	...
Matthew	False	17746703616573444 64	2015-09-15T13:53:13
Matt	False	10805729443184646 79	2013-01-30T21:45:33
Jerson	False	18749831900037123 96	2016-01-31T23:36:54
Xinrong	False	20465811424542726 53	2016-09-24T17:51:24
Paul	False	13335745108377601 38	2014-01-14T23:34:30

Data Cleaning Process (General)

- Make a decision about what columns are chosen as primary key
- Modify, create and remove columns based on current data to achieve these goals:
  - Data homogenization: each column contains same type of data
  - Create variables that are wanted for analysis
  - Remove redundant columns not wanted for analysis
- Deal with missing values and duplicate data to satisfy 1NF (or go higher if needed)
- Sort the data in some order that helps make sense of the data (e.g., by the primary key)

Commonly Used String Methods

- Fetch a substring `DataFrame.col.str[b:e]`
- Substring membership `DataFrame.col.str.contains(s)`
- Split a string into parts `DataFrame.col.str.split(s)`
- Other common methods (e.g., findall, replace, strip)

String Methods: Fetch a Substring

	`DataFrame.col.str[b:e]`
Parameters	b: starting position (can be omitted if 0), e: one plus the ending position (can be omitted if end of string)
Returns	a Pandas Series object containing the substrings

`data=`

	first_name	is_group	id	date_joined
0	Sion	False	2082497001160704615	2016-11-13T07:09:48.000Z
1	Kari	False	2538731244355584742	2018-08-04T18:45:48.000Z
2	Jessie	False	1921315569139712500	2016-04-04T21:51:04.000Z
3	Dion	False	2019594260709376161	2016-08-18T12:13:17.000Z
4	Alec	False	2080895330680832565	2016-11-11T02:07:34.000Z

```
data.date_joined.str[:19]
```

output

```
0    2016-11-13T07:09:48
1    2018-08-04T18:45:48
2    2016-04-04T21:51:04
3    2016-08-18T12:13:17
4    2016-11-11T02:07:34
```

String Methods: Substring Membership

	`DataFrame.col.str.contains(s)`
Arguments	s: substring to be recognized in the values of the column `DataFrame.col`
Returns	a Pandas Series object containing Boolean values

data=

	first_name	is_group	id	date_joined
0	Sion	False	2082497001160704615	2016-11-13T07:09:48
1	Kari	False	2538731244355584742	2018-08-04T18:45:48
2	Jessie	False	1921315569139712500	2016-04-04T21:51:04
3	Dion	False	2019594260709376161	2016-08-18T12:13:17
4	Alec	False	2080895330680832565	2016-11-11T02:07:34

```
data.date_joined.str.contains('2018')
```

output

```
0    False
1     True
2    False
3    False
4    False
```

String methods: Split a String Into Parts

	`DataFrame.col.str.split(s)`
Arguments	s: character or (short) string that splits `DataFrame.col` into a list of substrings
Returns	a Pandas Series object containing lists of substrings

data=

	first_name	is_group	id	date_joined
0	Sion	False	2082497001160704615	2016-11-13T07:09:48
1	Kari	False	2538731244355584742	2018-08-04T18:45:48
2	Jessie	False	1921315569139712500	2016-04-04T21:51:04
3	Dion	False	2019594260709376161	2016-08-18T12:13:17
4	Alec	False	2080895330680832565	2016-11-11T02:07:34

```
>>> data.date_joined.str[:10].str.split('-')
```

output

```
0    [2016, 11, 13]
1    [2018, 08, 04]
2    [2016, 04, 04]
3    [2016, 08, 18]
4    [2016, 11, 11]
```

Other Common String Methods

Pattern matching: `DataFrame.col.str.findall(pattern).str[0]`

- Match the regular expression using *pattern* for all values of DataFrame.col, and return the first match

Replace substring: `DataFrame.col.str.replace(old,new)`

- Replace any values equal to *old* by the value *new*

Left trimming: `DataFrame.col.str.lstrip()`

- Remove any blank spaces (' ') in the beginning

Right trimming: `DataFrame.col.str.rstrip()`

- Remove any blank spaces (' ') at the end

Trimming (both sides): `DataFrame.col.str.strip()`

- Remove any trailing blank spaces (' ') on both sides

Two Ways to Append a New Column

Suppose we want to add a new column to the DataFrame **`data`** with values **`newcol`** and name **`'v'`**. There are two way:

```
data['v'] = newcol
```

OR

```
data = pd.concat([data, newcol.rename('v')], axis=1)
```

- They are basically equivalent, but the second approach is more general: it also works for a DataFrame with more than 1 column
- For example:

```
data = pd.concat([data, newdata], axis=1)
```

Add New Columns to a DataFrame

If this were **`0`**, we'd be adding new rows instead!

	`pd.concat(DataFrames,axis=1)`
Arguments	`DataFrames`: a list of DataFrames with the same number of observations (and non-overlapping column names) to be concatenated `axis`: must be `1` or `'columns'` in this application
Returns	a Pandas DataFrame that combines all the data in the list `DataFrames`, by stacking them together as columns.

Add New Columns to a DataFrame

```
data['joined2018'] = data.date_joined.str.contains('2018')
data= pd.concat([data,data.date_joined.str[:10].str.split('-
').rename('ymd_list')],axis=1)
```

data

	first_name	is_group	id	date_joined	joined2018	ymd_list
0	Sion	False	20824970011607046l5	2016-11-13T07:09:48	False	[2016, 11, 13]
1	Kari	False	25387312443555847 42	2018-08-04T18:45:48	True	[2018, 08, 04]
2	Jessie	False	19213155691397125 00	2016-04-04T21:51:04	False	[2016, 04, 04]
3	Dion	False	20195942607093761 61	2016-08-18T12:13:17	False	[2016, 08, 18]
4	Alec	False	20808953306808325 65	2016-11-11T02:07:34	False	[2016, 11, 11]

Apply Any Function to a Column

`f` can be a built-in Python function

	`DataFrame.col.apply(f)`
Arguments	`f`: a function (built-in, external, or a lambda function)
Returns	a Pandas Series object resulting from applying `f` element-wise to values in `DataFrame.col`

```
data.date_joined.apply(int)
```

→ *Casts all values in the column* **`date_joined`** *to integers, e.g., 1.00 becomes 1*

Apply Any Function to a Column

`f` can be a user-defined function

	`DataFrame.col.apply(f)`
Arguments	`f`: a function (built-in, external, or a lambda function)
Returns	a Pandas Series object resulting from applying `f` element-wise to values in `DataFrame.col`

```
def f(date):
    ...
data.date_joined.apply(f)
```

→ *Applies a user-defined function* **`f`** *on the column* **`date_joined`**

Apply Any Function to a Column

`f` can be a lambda function

	`DataFrame.col.apply(f)`
Arguments	`f`: a function (built-in, external, or a lambda function)
Returns	a Pandas Series object resulting from applying `f` element-wise to values in `DataFrame.col`

```
data.date_joined.apply(lambda x: x if type(x)==list else [])
```

→ *Keep the value unchanged if its type is a* **`list`**, *and turn everything else into an empty string* `[]`

Handle Missing Values & Duplicate Data

- Decide and identify the ***primary keys*** for each DataFrame. The primary keys must be *non-missing*, and they must be *unique identifiers* of observations in the data.

- Data cleaning usually requires:

➔Removing all the observations with missing primary keys

➔Keeping exactly one observation for each primary key

Count Frequencies of the Primary Key

	`DataFrame.value_counts(dropna)`
Arguments	**`dropna`:** must be set to **`False`**, because the default is `True` and we need to see the missing values
Returns	a Pandas Series object with all distinct values of `DataFrame.col` as index and their frequency as value

- `value_counts()` automatically *sorts the unique values by their* ***counts in descending order***, so any values>1 appear at the top

For example,

```
data[['id']].value_counts(dropna=False)
```

Count Frequencies of the Primary Key

```
data[['id']].value_counts(dropna=False)
```

output

```
id
1759830821830656677    2
2017628776300544678    2
1934351583412224902    2
2104942223425536361    2
1919028113178624252    2
                      ..
9935168009666656551    1
9957598392483842O4     1
9958405879889923 72    1
9445081371115520 09    1
NaN                    1
Name: count, Length: 19993, dtype: int64
```

Clearly, there are duplicate observations for some values of the primary key “id”

Drop Rows with Missing Primary Key

	`DataFrame.dropna(subset)`
Arguments	`subset`: a list of one or more column names
Returns	a DataFrame object that drops observations with missing value in at least one variable among `subset` and keeps all other observations

- The method can also be performed *inplace*

For example,

```
data.dropna(subset=['id'],inplace=True)
```

Modifies **`data`** directly by dropping observations with missing **`'id'`** values

Drop Rows with Missing Primary Key

	first_name	is_group	id	date_joined
0	Sion	False	2082497001160704615	2016-11-13T07:09:48
1	Kari	False	2538731244355584742	2018-08-04T18:45:48
2	Jessie	False	1921315569139712500	2016-04-04T21:51:04
3	Dion	False	2019594260709376161	2016-08-18T12:13:17
4	Alec	False	2080895330680832565	2016-11-11T02:07:34
...	...	...	...	...
19995	Matthew	False	1774670361657344464	2015-09-15T13:53:13
19996	Matt	False	1080572944318464679	2013-01-30T21:45:33
19997	Jerson	False	1874983190003712396	2016-01-31T23:36:54
19998	Xinrong	False	2046581142454272653	2016-09-24T17:51:24
19999	Paul	False	1333574510837760138	2014-01-14T23:34:30

20000 rows × 8 columns

```
data.dropna(subset=['id'],inplace=True)
```

	first_name	is_group	id	date_joined
0	Sion	False	2082497001160704615	2016-11-13T07:09:48
1	Kari	False	2538731244355584742	2018-08-04T18:45:48
2	Jessie	False	1921315569139712500	2016-04-04T21:51:04
3	Dion	False	2019594260709376161	2016-08-18T12:13:17
4	Alec	False	2080895330680832565	2016-11-11T02:07:34
...	...	...	...	...
19995	Matthew	False	1774670361657344464	2015-09-15T13:53:13
19996	Matt	False	1080572944318464679	2013-01-30T21:45:33
19997	Jerson	False	1874983190003712396	2016-01-31T23:36:54
19998	Xinrong	False	2046581142454272653	2016-09-24T17:51:24
19999	Paul	False	1333574510837760138	2014-01-14T23:34:30

19999 rows × 8 columns

The one observation with missing `'id'` *is dropped from the data*

Drop Duplicates by Primary Key

	`DataFrame.drop_duplicates(subset,keep)`
Arguments	`subset`: a list of one or more column names `keep`: `'first'` (default), `'last'`, `False`
Returns	a DataFrame object that keeps exactly one observation for each unique value of `subset`

- The method can also be performed *inplace*. For example,

```
data.drop_duplicates(subset=['id'],keep='first',inplace=True)
```

Only the *first occurrence* (**`keep='first'`**) of each unique value of **`'id'`** is kept, and **`data`** is directly modified

Drop Duplicates by Primary Key

	first_name	is_group	id	date_joined
0	Sion	False	2082497001160704615	2016-11-13T07:09:48
1	Kari	False	2538731244355584742	2018-08-04T18:45:48
2	Jessie	False	1921315569139712500	2016-04-04T21:51:04
3	Dion	False	2019594260709376161	2016-08-18T12:13:17
4	Alec	False	2080895330680832565	2016-11-11T02:07:34
...	...	...	...	...
19995	Matthew	False	1774670361657344464	2015-09-15T13:53:13
19996	Matt	False	1080572944318464679	2013-01-30T21:45:33
19997	Jerson	False	1874983190003712396	2016-01-31T23:36:54
19998	Xinrong	False	2046581142454272653	2016-09-24T17:51:24
19999	Paul	False	1333574510837760138	2014-01-14T23:34:30

19999 rows × 8 columns

```
data.drop_duplicates(subset=['id'],
keep='first', inplace=True)
```

	first_name	is_group	id	date_joined
0	Sion	False	2082497001160704615	2016-11-13T07:09:48
1	Kari	False	2538731244355584742	2018-08-04T18:45:48
2	Jessie	False	1921315569139712500	2016-04-04T21:51:04
3	Dion	False	2019594260709376161	2016-08-18T12:13:17
4	Alec	False	2080895330680832565	2016-11-11T02:07:34
...	...	...	...	...
19995	Matthew	False	1774670361657344464	2015-09-15T13:53:13
19996	Matt	False	1080572944318464679	2013-01-30T21:45:33
19997	Jerson	False	1874983190003712396	2016-01-31T23:36:54
19998	Xinrong	False	2046581142454272653	2016-09-24T17:51:24
19999	Paul	False	1333574510837760138	2014-01-14T23:34:30

19993 rows × 4 columns

6 observations were removed, and the updated data now has unique `'id'` *values*

Tips on Data Cleaning

- As a first step, very important to choose a **primary key**: drop *missing values* and *duplicate rows* relative to the primary key

- Be careful with different missing value *types*
  - **NaN** (**`np.isnan`**): missing value for **numeric** types
  - **None**: a generic data object (e.g., string, but not numeric)
  - **Empty string** (`''` or `""`): sometimes treated as the missing value for string variables

Tips on Data Cleaning

- Make sure to **harmonize data formats**
  - E.g., `"fifteen"`, `15`, `"0015"`, and `15.00` are different ways to express *the same* number 15, so they should all become `15` with `int` as the data type
- **Remove redundant columns** (in principle 3NF, but not strictly)
  - E.g., delete intermediate columns created to clean or process data
- **Sort the rows** in some order that is useful for data analysis
  - E.g., often in ascending order of the primary key, but not always

Exercise: Data Cleaning (Advanced)

Work on part of **Assignment 1, Problem 2(f)**: “Write Python code to execute your relational model database re-design and produce the data tables corresponding to your proposed design.”

1. Read the CSV file **assignment1_venmo_dataset_jul2018.csv** into a Pandas DataFrame
2. Define the **primary key** to uniquely identify *each transaction*
3. Design and create a **transactions table** that satisfies the **Third Normal Form (3NF)**: each column has atomic values (1NF), and no partial or transitive dependencies (2NF and 3NF).