Introduction to Data Science

Data Wrangling Continued

Joanna Bieri DATA101

Important Information

- Email: joanna_bieri@redlands.edu
- Office Hours: Duke 209 Click Here for Joanna's Schedule

Last Class - Definitions

Make sure you can define and come up with examples for

- Raw Data
- Tidy Data
- Displayed Data
- Summarized Data

Last Class - Code

We learned how to select subsets of the data and sort the data. You should be able to:

- Find all the column names and print them.
- Slice the data by selecting a subset of the columns
- Sort the data either ascending or descending
- Mask the data using logical operators

This class

Our goal is to continue learning commands that helps us retrieve pieces of the data. Today we will:

- Learn how to do more advanced masking
- Find count unique values with value_counts() and drop_duplicates()
- Add new columns
- Use group_by() to do statistics on groups

Load the Data and look at the basic info

Same data as last class!

- Data from two hotels: one resort and one city hotel
- Observations: Each row represents a hotel booking

Data Information: Tidy Tuesday

Look at the Column names!

```
columns_list = list(DF_raw_hotels.keys())
for i in columns_list:
    print(i)
```

```
hotel
is_canceled
lead_time
arrival_date_year
arrival_date_month
arrival_date_week_number
arrival_date_day_of_month
stays_in_weekend_nights
stays_in_week_nights
adults
children
babies
```

More advanced boolean masks - combining masks

Last time we learned about boolean masks. We were able to use a mask to only show certain rows or columns in the data.

Basic Operators

Operator	Definition
$<$	less than
$>$	greater than
$<=$	less than or equal to
$>=$	greater than or equal to
$==$	exactly equal to
$!=$	not equal to

More advanced boolean masks - combining masks

Advanced Operators

Operator	Definition
and	check if two things are both true
or	check if one of two things is true
in	checks if something is in another thing
!	not checks if something is false

More advanced boolean masks - Applying one at a time

What does this code do?

```
mask1 = (DF_raw_hotels['hotel']=='Resort Hotel')
DF_hotels_mask=DF_raw_hotels[mask1]
mask2 = (DF_hotels_mask['is_canceled']==1)
DF_final = DF_hotels_mask[mask2]
DF_final
```

More advanced boolean masks - Combining Masks

You can merge this command into one line using:

Operator	Definition
&	check if two things are both true (bitwise = works on multiple values)
|	check if one of two things is true (bitwise = works on multiple values)

```
mask = (DF_raw_hotels['hotel']=='Resort Hotel')
        &
        (DF_raw_hotels['is_canceled']==1)
DF_raw_hotels[mask]
```

Another Example of Combining Masks

This time we will ask:

- Show only results for visitors from USA or GBR.
- Can you figure out how to do this?

Another Example of Combining Masks - solution

```
mask = (DF_raw_hotels['country']=='USA')
        |
        (DF_raw_hotels['country']=='GBR')
DF_raw_hotels[mask]
```

Finding unique values

Often we want to see how many different categories are contained in a categorical column. This lets us see what the choices are for that entry.

Unique in one column

- select just one column

```
my_columns = ['market_segment']
```

- grab that column of the data frame

```
DF_raw_hotels[my_columns]
```

- then applying the drop_duplicates() command

```
DF_raw_hotels[my_columns].drop_duplicates()
```

Unique in one column - Result

	market_segment
0	Direct
3	Corporate
4	Online TA
9	Offline TA/TO
125	Complementary
413	Groups
40600	Undefined
49013	Aviation

Unique across two columns

- **.drop_duplicates()** can look at more than one column for duplicates.
- If we read down just one column, but there are no duplicates if we consider both columns.

```
my_columns = ['market_segment','customer_type']
DF_raw_hotels[my_columns].drop_duplicates()
```

Unique across two columns - results

	market_segment	customer_type
0	Direct	Transient
3	Corporate	Transient
4	Online TA	Transient
9	Offline TA/TO	Transient
16	Offline TA/TO	Contract
47	Offline TA/TO	Transient-Party
125	Complementary	Transient
127	Online TA	Transient-Party
260	Direct	Transient-Party
413	Groups	Transient-Party
417	Groups	Transient
539	Online TA	Group
1530	Corporate	Transient-Party
1539	Direct	Group
1587	Offline TA/TO	Group

Counting unique values in one column

Counts of how often labels were seen in the data = Frequency, Value Counts, Frequency Counts, etc.

In Python we can use the **.value_counts()** command.

```
my_columns = ['market_segment']
DF_raw_hotels[my_columns].value_counts()
```

Counting unique values in one column - results

```
market_segment
Online TA         56477
Offline TA/TO     24219
Groups            19811
Direct            12606
Corporate          5295
Complementary       743
Aviation            237
Undefined             2
Name: count, dtype: int64
```

Counting unique values in two columns

```
my_columns = ['market_segment','customer_type']
DF_raw_hotels[my_columns].value_counts()
```

Counting unique values in two columns - results

market_segment	customer_type	
Online TA	Transient	51299
Offline TA/TO	Transient	14054
Direct	Transient	11336
Groups	Transient-Party	10633
	Transient	8427
Offline TA/TO	Transient-Party	8137
Corporate	Transient	3576
Online TA	Transient-Party	3513
Offline TA/TO	Contract	1817
Corporate	Transient-Party	1668
Online TA	Contract	1486
Direct	Transient-Party	1122
Groups	Contract	735
Complementary	Transient	703
Aviation	Transient	218

Counting unique values in two columns - results

This lets us separate Online TA into four subgroups:

Online TA	Transient	51299
Online TA	Transient-Party	3513
Online TA	Contract	1486
Online TA	Group	179

Counting unique values - Unsorted

Sometimes the way that .value_counts() sorts the data in descending order, actually makes the data harder to read!

You can use the flag **sort=False** to stop this sorting.

```
my_columns = ['market_segment','customer_type']
DF_raw_hotels[my_columns].value_counts(sort=False)
```

Counting unique values - Unsorted - Results

market_segment	customer_type	
Aviation	Group	2
	Transient	218
	Transient-Party	17
Complementary	Contract	2
	Group	6
	Transient	703
	Transient-Party	32
Corporate	Contract	22
	Group	29
	Transient	3576
	Transient-Party	1668
Direct	Contract	14
	Group	134
	Transient	11336
	Transient-Party	1122

Counting unique values - Ascending

You can also ask to sort the data ascending with **ascending=True**

```
my_columns = ['market_segment','customer_type']
DF_raw_hotels[my_columns].value_counts(ascending=True)
```

Counting unique values - Ascending - Results

market_segment	customer_type	
Aviation	Group	2
Complementary	Contract	2
Undefined	Transient-Party	2
Complementary	Group	6
Direct	Contract	14
Groups	Group	16
Aviation	Transient-Party	17
Corporate	Contract	22
	Group	29
Complementary	Transient-Party	32
Direct	Group	134
Online TA	Group	179
Offline TA/TO	Group	211
Aviation	Transient	218
Complementary	Transient	703

Saving Unique Values to a Data Frame

- First we do the value counts like normal and then save the information in the variable my_counts

```
my_columns = ['market_segment','customer_type']
my_counts = DF_raw_hotels[my_columns].value_counts()
```

- Then we do **.reset_index().rename(columns={"index": "value", 0: "count"})**. In this class you can just copy and past this after the variable name.

```
my_counts = my_counts.reset_index().rename(columns={"i
```

Saving Unique Values to a Data Frame - Results

	market_segment	customer_type	count
0	Online TA	Transient	51299
1	Offline TA/TO	Transient	14054
2	Direct	Transient	11336
3	Groups	Transient-Party	10633
4	Groups	Transient	8427
5	Offline TA/TO	Transient-Party	8137
6	Corporate	Transient	3576
7	Online TA	Transient-Party	3513
8	Offline TA/TO	Contract	1817
9	Corporate	Transient-Party	1668
10	Online TA	Contract	1486
11	Direct	Transient-Party	1122
12	Groups	Contract	735
13	Complementary	Transient	703
14	Aviation	Transient	218

Adding a column to a Data Frame

In many cases your data will have information across multiple columns that you will want to combine. You can do this by adding a new column to your data frame.

In the **raw data** there are two columns 'children' and 'babies', but what if I actually wanted to know the total number of little ones that families were traveling with. In other words, I wanted to combine the information about children and babies (add it up). We do this my adding a new column!

Adding a column to a Data Frame

- First you call your data frame and put in a name that is not already in the columns:

```
DF_raw_hotels['little_ones']
```

- Then you set that equal to the calculation you want to do

```
DF_raw_hotels['little_ones']=
DF_raw_hotels['children'] + DF_raw_hotels['babies']
```

- This will save the new calculation in a column with the name 'little_ones'

Grouping the data frame into chunks

Grouping allows us to grab data in categorical groups and then apply a function to all of the variables in those groups.

Using the **.groupby** command: we are grouping by the 'hotel' column which has two different options (City Hotel and Resort Hotel).

```
DF_raw_hotels.groupby(by=['hotel'])
```

Once the data is grouped we can do different operations.

```
DF_raw_hotels.groupby(by=['hotel']).sum()
```

Grouping the data frame into chunks - results

hotel	is_canceled	lead_time	arrival_date_year	arrival_date_m
City Hotel	33102	8705335	159943106	JulyJulyJulyJuly
Resort Hotel	11122	3712588	80765825	JulyJulyJulyJuly

```
hotel
City Hotel      JulyJulyJulyJulyJulyJulyJulyJulyJulyJulyJulyJu
Resort Hotel    JulyJulyJulyJulyJulyJulyJulyJulyJulyJulyJulyJu
Name: arrival_date_month, dtype: object
```

Grouping the data frame into chunks

Let's select just columns we care about, columns that are numerical:

```
my_columns = ['adults','children','little_ones','babies']
DF_raw_hotels.groupby(by=['hotel'])[my_columns].sum()
```

hotel	adults	children	little_ones	babies
City Hotel	146838	7248.0	7640.0	392
Resort Hotel	74798	5155.0	5712.0	557

Grouping the data frame into chunks

There are lots of different operations you could use:

- .min()
- .max()
- .mean()
- .median()
- .sum()
- .prod()
- .count()
- .describe()

Grouping the data frame into chunks - describe

```
DF_raw_hotels.groupby(by=['hotel'])[my_columns].describe()
```

	adults							
	count	mean	std	min	25%	50%	75%	m
hotel								
City Hotel	79330.0	1.850977	0.509292	0.0	2.0	2.0	2.0	4
Resort Hotel	40060.0	1.867149	0.697285	0.0	2.0	2.0	2.0	5

Grouping to quickly generate data

- The .groupby() function lets you quickly generate data.
- In just one command we are able to look at the total number of occupants for each of the columns (adults, children, little_ones, and babies) for each month in the data set.

```
DF_raw_hotels.groupby(by=['arrival_date_month'])
                              [my_columns].sum()
```

Grouping to quickly generate data - results

arrival_date_month	adults	children	little_ones	babies
April	20806	1141.0	1194.0	53
August	27795	2780.0	2976.0	196
December	12382	736.0	814.0	78
February	14450	790.0	849.0	59
January	10024	452.0	500.0	48
July	25164	2322.0	2443.0	121
June	20353	1057.0	1128.0	71
March	17675	700.0	757.0	57
May	21539	845.0	917.0	72
November	11488	279.0	324.0	45
October	20279	703.0	765.0	62
September	19681	598.0	685.0	87

What's next?

Once you have a nice data summary, you are ready to make some plots!

You can start asking more questions of your data.

Use a sense of curiosity and exploration