

Data Cleaning and Summary Statistics

module 5

Evaluation

- Online Python Tutorial 10%
- Homework 20%
 - Individual assignments, total 5 or 6 HW assignments
- Quizzes 15%
 - total 6 quizzes
- Midterm Exam 20%
 - On 5/4 during class time
- Group Project and presentation 15%
 - Groups of 2 people
 - The last two lectures will be dedicated to project presentations
 - Presentations are mandatory for everyone
- Final Exam 20%
 - In Final Exam week, 6/8 during class

Data cleaning

- Definition: prepares the data for analysis
- One of the most time-consuming tasks in data science

Today's data set

- Let's consider the survey taken by students at the beginning of the term (prior quarter)
- data science survey.csv

What don't we like of this table?

- Column headers are too long
- Some cell values are too long
- Some cell values are yes/no, but we prefer 1/0

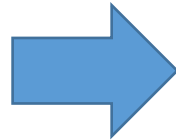
```
df = pd.read_csv('data/data science survey.csv')
```

```
df.head()
```

	Timestamp	Do you have a job?	How long ago did you get your Bachelor degree?	What program are enrolled in?	How would you rate your computer programming background?	Have you ever programmed in C?	Have you ever programmed in C++?	Have you ever programmed in C#?	Have you ever programmed in Java?
0	2017/01/09 1:48:20 PM PST	No, I'm not working at the moment	longer than 1 year ago but less than 3 years ago	MSIS	4	Yes	Yes	No	Yes
1	2017/01/09 2:15:59 PM PST	Yes, I have a part-time job	over 5 years ago	MSIS	3	Yes	Yes	No	Yes
2	2017/01/09 3:48:48 PM PST	No, I'm not working at the moment	longer than 3 years ago but less than 5 years	MSIS	3	No	No	No	Yes

Change the column names

```
df.columns = ['Timestamp',  
              'Job',  
              'BachTime',  
              'Program',  
              'ProgSkills',  
              'C',  
              'CPP',  
              'CS',  
              'Java',  
              'Python',  
              'JS',  
              'R',  
              'SQL',  
              'SAS',  
              'Excel',  
              'Tableau',  
              'Regression',  
              'Classification',  
              'Clustering']
```



	Timestamp	Job	BachTime	Program	ProgSkills	C	CPP	CS	Java
0	2017/01/09 1:48:20 PM PST	No, I'm not working at the moment	longer than 1 year ago but less than 3 years ago	MSIS	4	Yes	Yes	No	Yes
1	2017/01/09 2:15:59 PM PST	Yes, I have a part- time job	over 5 years ago	MSIS	3	Yes	Yes	No	Yes
2	2017/01/09 3:48:48 PM PST	No, I'm not working at the moment	longer than 3 years ago but less than 5 years ago	MSIS	3	No	No	No	Yes

Remove column “Timestamp”

```
df.drop('Timestamp', axis=1, inplace=True)
```

Remove a column



Make the change on df
(instead of returning a
modified copy of df)

	Job	BachTime	Program	ProgSkills	C	CPP	CS	Java	Python
0	No, I'm not working at the moment	longer than 1 year ago but less than 3 years ago	MSIS	4	Yes	Yes	No	Yes	Yes
1	Yes, I have a part-time job	over 5 years ago	MSIS	3	Yes	Yes	No	Yes	No

Change values in column “Job”

We want to perform this transformation:

- No job → 0
- Part-time → 0.5
- Full-time → 1

	Job
0	No, I'm not working at the moment
1	Yes, I have a part-time job
2	No, I'm not working at the moment
3	No, I'm not working at the moment
4	No, I'm not working at the moment
5	Yes, I have a full-time job
6	No, I'm not working at the moment
7	No, I'm not working at the moment
8	Yes, I have a full-time job
9	Yes, I have a part-time job
10	Yes, I have a part-time job



	Job
0	0.0
1	0.5
2	0.0
3	0.0
4	0.0
5	1.0
6	0.0
7	0.0
8	1.0
9	0.5
10	0.5

Change values in column “Job”: solution 1

```
df.loc[df['Job'] == 'No, I\'m not working at the moment', 'Job1'] = 0
```

```
df.loc[df['Job'] == 'Yes, I have a part-time job', 'Job1'] = 0.5 #df.Job works as well
```

```
df.loc[df['Job'] == 'Yes, I have a full-time job', 'Job1'] = 1
```

Pros:

- Easy

Cons:

- Sometimes such solution cannot be used; example: if the new value of the column is retrieved through a complex calculation (e.g., simulation)

Change values in column “Job”: solution 2

Define the transformation by writing a function whose **input** is the old value of the column and whose **output** is the new value of the column. Then, use the DataFrame function “apply”

```
def job2number (jobDescr):  
    if jobDescr.startswith('No'):  
        return 0  
    elif 'part-time' in jobDescr:  
        return 0.5  
    else:  
        return 1
```

```
df['Job'] = df['Job'].apply(job2number)
```

A function with:

- Input = old value
- Output = new value

Pros:

- Flexible

Cons:

- Writing a function takes space and time

Apply this function to df['Job']

Change values in column “Job”: solution 3 (optional)

Instead of declaring a function, use lambda (or anonymous) functions

```
df['Job'] = df['Job'].apply(lambda x : 0 if x.startswith('No') else 1 if 'full-time' in x else 0.5)
```

The keyword `lambda` defines a short function without the need to give it a name. The function takes x as input and gives a number 0, 0.5, or 1 as output

Pros:

- Flexible

Cons:

- Can only write one-line functions

Change values in column “Job”: Result

[illegible]

Make BachTime a dummy variable

We want to perform this transformation:
do the “dummy” first then simplify the column names

	BachTime
0	longer than 1 year ago but less than 3 years ago
1	over 5 years ago
2	longer than 3 years ago but less than 5 years ago
3	over 5 years ago
4	longer than 3 years ago but less than 5 years ago
5	longer than 1 year ago but less than 3 years ago
6	longer than 1 year ago but less than 3 years ago
7	< 1 year ago
8	over 5 years ago
9	over 5 years ago
10	over 5 years ago



	Bach_0to1	Bach_1to3	Bach_3to5	Bach_5Plus
0	0	1	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	0	1
4	0	0	1	0
5	0	1	0	0
6	0	1	0	0
7	1	0	0	0
8	0	0	0	1
9	0	0	0	1
10	0	0	0	1

Step 1: use *get_dummies*

```
dumTime = pd.get_dummies(df, columns=[ 'BachTime' ])
dumTime
```

Create a modified copy of the dataframe df where the column BachTime is replaced by dummy columns

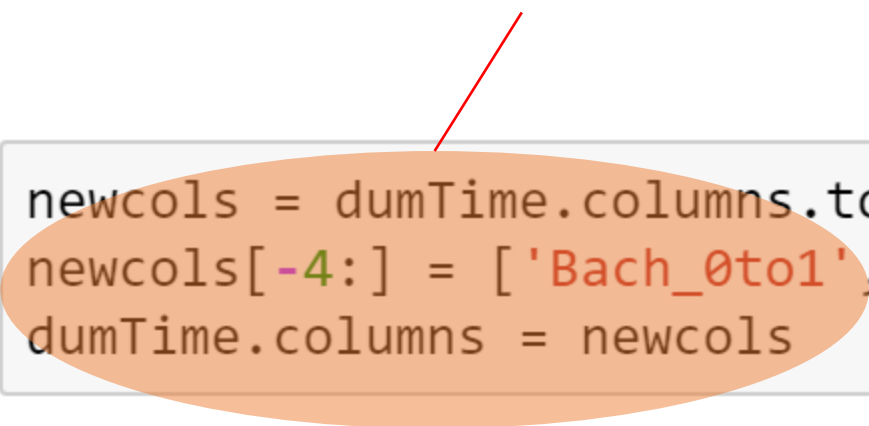
	Job	Program	ProgSkills	...	BachTime_< 1 year ago	BachTime_longer than 1 year ago but less than 3 years ago	BachTime_longer than 3 years ago but less than 5 years ago	BachTime_over 5 years ago
0	0.0	MSIS	4		0	1	0	0
1	0.5	MSIS	3	...	0	0	0	1
2	0.0	MSIS	3		0	0	1	0
3	0.0	MSIS	3		0	0	0	1
4	0.0	MSIS	3		0	0	1	0
5	1.0	Supply Chain Mgmt & Analytics	1	...	0	1	0	0
6	0.0	MSIS	3		0	1	0	0
7	0.0	MSIS	2		1	0	0	0
8	1.0	MBA	1		0	0	0	1
9	0.5	MSIS	3	...	0	0	0	1
10	0.5	MBA	4		0	0	0	1

Step 2: change the column names

Note that we cannot modify a column individually: that is, `df.columns[0] = 'newname'` will give an error. However, we can replace the whole set of columns.

Instead, we:

1. Create a list *newcols* with the original column names
2. Modify the last four elements with the new names
3. Replace the whole set of columns.



```
newcols = dumTime.columns.tolist()
newcols[-4:] = ['Bach_0to1', 'Bach_1to3', 'Bach_3to5', 'Bach_5Plus']
dumTime.columns = newcols
```

```
df = dumTime
```

Change “Yes” and “No” to 1 and 0

	Job	Program	ProgSkills	C	CPP	CS	Java	Python	JS
0	0.0	MSIS	4	Yes	Yes	No	Yes	Yes	Yes
1	0.5	MSIS	3	Yes	Yes	No	Yes	No	No
2	0.0	MSIS	3	No	No	No	Yes	Yes	No
3	0.0	MSIS	3	Yes	No	No	Yes	Yes	No
4	0.0	MSIS	3	Yes	No	No	Yes	Yes	No



	Job	Program	ProgSkills	C	CPP	CS	Java	Python	JS
0	0.0	MSIS	4	1	1	0	1	1.0	1.0
1	0.5	MSIS	3	1	1	0	1	0.0	0.0
2	0.0	MSIS	3	0	0	0	1	1.0	0.0
3	0.0	MSIS	3	1	0	0	1	1.0	0.0
4	0.0	MSIS	3	1	0	0	1	1.0	0.0

Change “Yes” and “No” to 1 and 0: *Replace*

```
df.replace('Yes', 1, inplace=True)  
df.replace('No', 0, inplace=True)
```

Change the value of all cells equal to
“Yes” to 1

Inplace = True makes
the change on df
rather than creating
a copy

Add a calculated column

- Let's add a column **Languages** that counts the programming languages known by each student:
- *$Languages = C + CPP + CS + Java + Python + JS + R + SQL + SAS$*

	C	CPP	CS	Java	Python	JS	R	SQL	SAS
0	1	1	0	1	1.0	1.0	0.0	1	0.0
1	1	1	0	1	0.0	0.0	0.0	1	0.0
2	0	0	0	1	1.0	0.0	0.0	1	0.0
3	1	0	0	1	1.0	0.0	1.0	1	0.0
4	1	0	0	1	1.0	0.0	0.0	1	0.0



	Languages
0	6.0
1	4.0
2	3.0
3	5.0
4	4.0

Add a calculated column

```
df['Languages'] = df.C+df.CPP+df.CS+df.Java+df.Python+df.JS+df.R+df.SQL + df.SAS
```



Create a column by simply declaring it

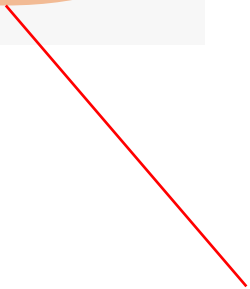
Add a more complex column

- Let's add a column **Expert**, which is 1 if the student knows at least 3 languages and 0 otherwise.
- One possible way is to use the function *DataFrame.apply*
- The function passed has:
 - Input = one row of the DataFrame
 - Output = the value of the new column Expert for that row

Solution with apply 1/2: define a function

```
def expertFunction(row):  
    if row['Languages'] >= 3:  
        return 1  
    else:  
        return 0
```

```
toadd=df.apply(expertFunction, axis = 1)  
df['Expert'] = toadd
```



Axis=1 indicates that the function takes a row as input (summarizes columns)

Solution with apply 2/2: lambda function

```
toadd=df.apply(lambda row: 1 if row['Languages'] >= 3 else 0, axis = 1)  
df['Expert'] = toadd
```

Performance of *DataFrame.apply*

- It is slow!!
- If called with axis=1, it is basically a for-loop through the rows
- Do not use it unless it is absolutely needed
 - Example: for each row, we need to run a simulation
- If possible, use operations among Series and scalars

```
df[ 'Expert' ] = (df[ 'Languages' ] >= 3) * 1.0
```

DataFrame.apply vs *Series.apply* ¶

- As of pandas 0.20, *DataFrame.apply* is slow whereas *Series.apply* is fast.
- Example: compute the lower case value of the column *Program*.

Solution 1: DataFrame.apply (slow! Do not use!)

```
%timeit df.apply(lambda r : r['Program'].lower(), axis = 1)
```

1000 loops, best of 3: 1.36 ms per loop



Apply to all rows of the dataframe

Solution 2: Series.apply (fast because it uses vectorization)

```
%timeit df.Program.apply(lambda v : v.lower())
```

10000 loops, best of 3: 137 µs per loop



Apply to all values of the series

Problems

- How many students know SQL?
- What's the average programming skills of MSIS students? Compare it to that of MBA students
- How many students know classification better than clustering? And how many clustering better than classification?

Summary statistics

describe

- The method **describe** returns summary statistics for each column

How many not null

percentiles

```
df.describe()
```

	ProgSkills	C	CPP
count	30.000000	30.000000	30.000000
mean	2.733333	0.633333	0.433333
std	1.080655	0.490133	0.504007
min	1.000000	0.000000	0.000000
25%	2.000000	0.000000	0.000000
50%	3.000000	1.000000	0.000000
75%	3.000000	1.000000	1.000000
max	5.000000	1.000000	1.000000

corr

- Finds the correlation for each pair of columns

Looks like C and C++ are correlated

```
df.corr()
```

	ProgSkills	C	CPP	CS
ProgSkills	1.000000	0.590268	0.409410	8.366077e-02
C	0.590268	1.000000	0.665375	2.305715e-02
CPP	0.409410	0.665375	1.000000	1.569570e-01
CS	0.083661	0.023057	0.156957	1.000000e+00

Problem to solve together

- What are the top 10 correlations and the top 10 anti-correlations?
Hint: understand what this does first:

```
cor = df.corr()  
cor.stack()
```