Introduction to Data Science

Lecture 2.1 Table and Pandas

Instructor: Sangryul Jeon

School of Computer Science and Engineering

srjeonn@pusan.ac.kr

Notice

- There will be **no class at 03/21**
  - Make-up class will be in a form of the **recorded video**, which will be uploaded at **PLATO**
  - There will be **no attendance check** for make-up class
  - The contents of make-up class will be **included in the scope of midterm exam, HW, and others**.

Floating Point

❖ Floating point (http://en.wikipedia.org/wiki/Floating_point)

- describes a method of representing an approximation to real numbers in a way that can support a wide range of values. The numbers are, in general, represented approximately to a fixed number of significant digits (the mantissa) and scaled using an exponent. The base for the scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form:

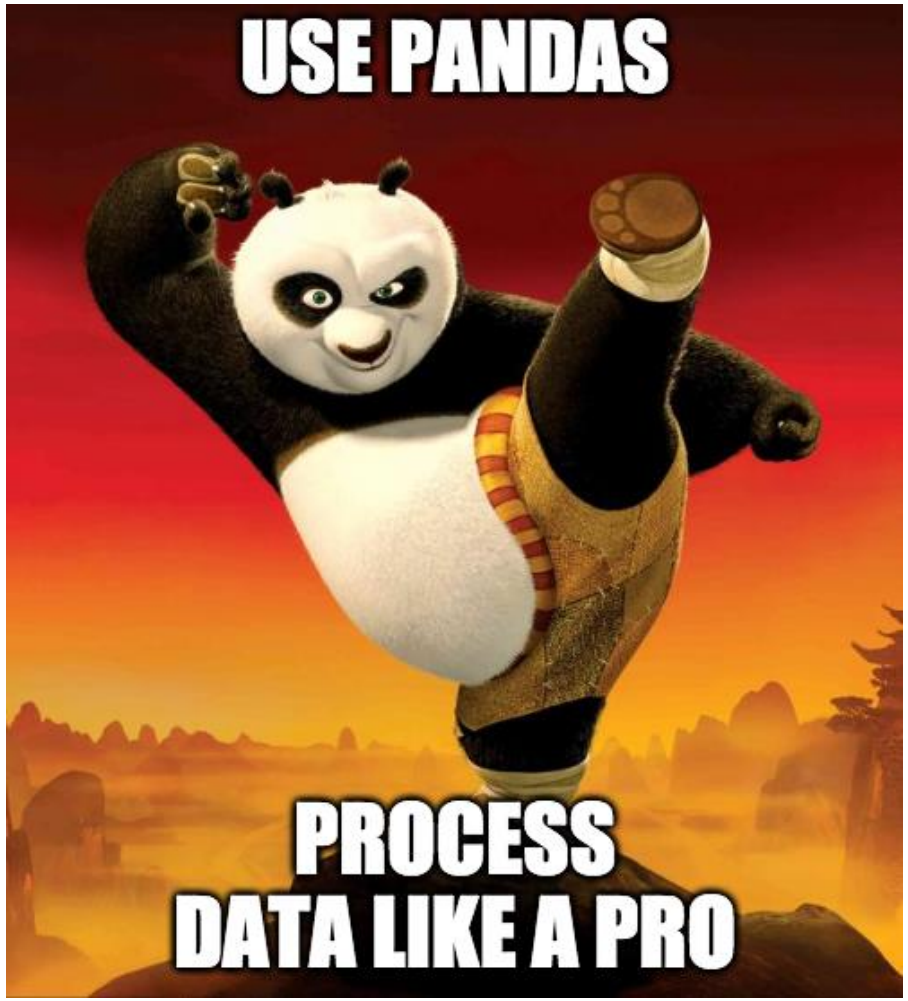
$$Significant\ digits \times base^{exponent}$$

$$1.2345 = \underbrace{12345}_{\text{mantissa}} \times 10^{\overbrace{-4}^{\text{exponent}}}$$

- IEEE floating point – IEEE 754

Name	Common name	Base	Digits	E min	E max
binary16	Half precision	2	10+1	−14	+15
binary32	Single precision	2	23+1	−126	+127
binary64	Double precision	2	52+1	−1022	+1023
binary128	Quadruple precision	2	112+1	−16382	+16383

Pandas



- Standard library for representing relational data for data scientists
- Created by Wes McKinney in 2008, Open source project

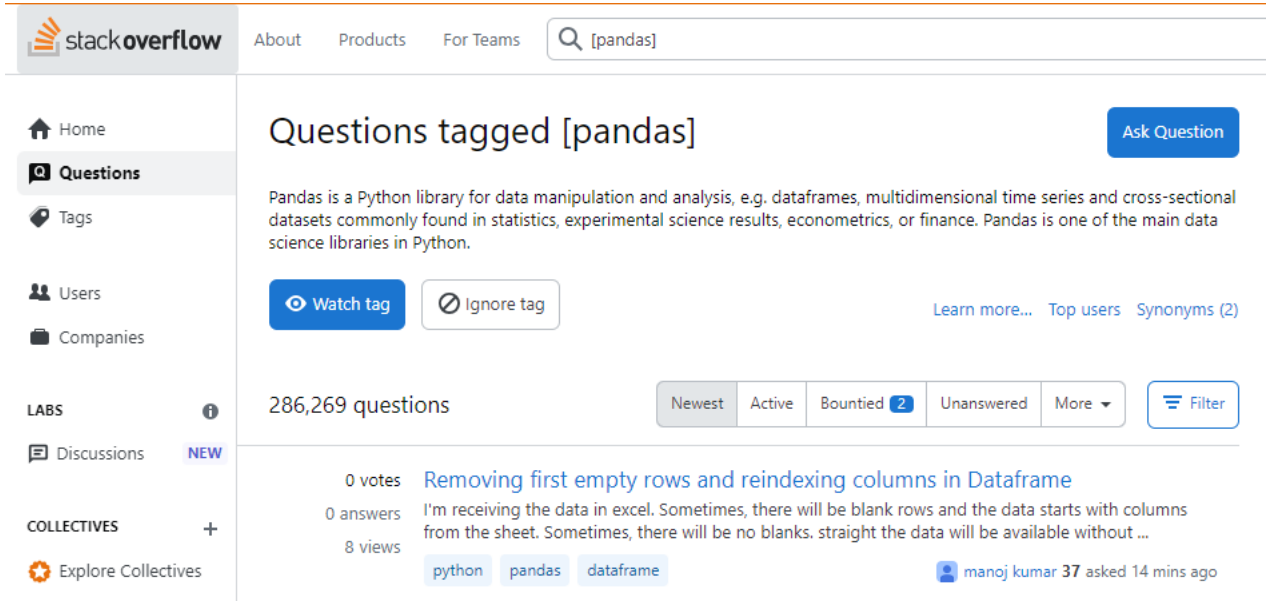
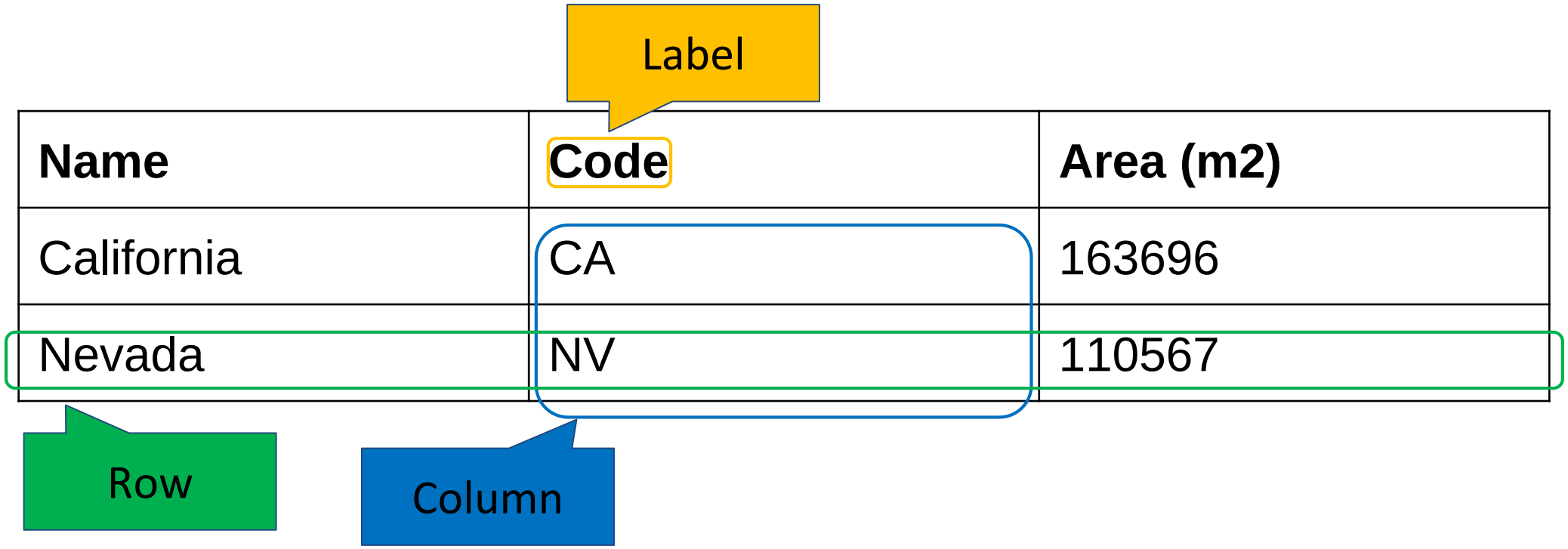


Table Structure

- A Table is a sequence of labeled columns
- Each row represents one individual
- Data within a column represents one attribute of the individuals



Ways to create a table

❖ Pandas

```
cones1 = pd.read_csv('cones.csv',
         index_col=0)
cones1
```

Flavor	Color	Price	Rating
strawberry	pink	3.55	1
chocolate	light brown	4.75	4
chocolate	dark brown	5.25	3
strawberry	pink	5.25	2
chocolate	dark brown	5.25	5
bubblegum	pink	4.75	1

❖ From scratch

```
# Define cones' attributes
conesAttributes = ["Flavor", "Color", "Price"]

# Define cones' data, must be same sequence as
attributes
conesData = [["strawberry", "pink", 3.55],
          ["chocolate", "light brown", 4.75],
          ["chocolate", "dark brown", 5.25],
          ["strawberry", "pink", 5.25],
          ["chocolate", "dark brown", 5.25],
          ["bubblegem", "pink", 4.75]]

# Define cones
cones3 = pd.DataFrame(data=conesData,
          columns=conesAttributes)
cones3
```

Creating a table

```
class pandas.DataFrame(data=None, index=None, columns=None,
dtype=None, copy=None)
```

```
# pandas
pd.DataFrame([8,34,5], columns=['Number of petals'])
```

```
pd.DataFrame({'Number of petals':[8,34,5]})
```

```
d = {'Number of petals':[8,34,5]} #dictionary
pd.DataFrame(data=d)
```

	Number of petals
0	8
1	34
2	5

Creating a table with two columns

❖ Pass a list of individuals (row) as data or label-value pairs as dict

```
pd.DataFrame(data = [[8,'lotus'],[34,'sunflower'],[5,'rose']],
                    columns=['Number of petals', 'Name'])
```

```
pd.DataFrame({'Number of petals':[8,34,5],
                    'Name':['lotus','sunflower','rose']})
```

❖ Checking the size of the table

```
df = df_1.copy(deep=True)
df.shape
```

```
[88] df = df_1.copy(deep=True)
     df.shape

     (8, 5)
```

Adding a new column to an existing table

- Use insert() method

```
flowers_df = pd.DataFrame({'Number of petals':[8,34,5],
                                    'Name':['lotus','sunflower','rose']})
flowers_df.insert(2, 'Color', ['pink','yellow','red'])
print(flowers_df)
```

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.insert.html

- Or simply,

```
flowers_df['Color'] = ['pink','yellow','red']
```

Column labels

- df.columns returns the column labels
  - df.index returns the row labels

```
# list(df.columns)
df.columns
```

```
Index(['Longitude', 'Latitude', 'Assign', 'Direction', 'Survivors'], dtype='object')
```

- df.rename replaces the column labels

```
df.rename(columns={'City':'City Name'}, inplace = True)
```

```
df = df.rename(columns={'City':'City Name'})
```

Column data

❖ Access to the column data

- iloc(integer location)
- iloc[:4]: index 0 … 4

```
df['Survivors'] # Series
df.iloc[:, 4]
df['Survivors'][0]
df.iloc[0, 4]

df.loc[2] # 2th row
```

❖ Choosing sets of columns

```
df[['City Name','Survivors']]
df.iloc[:, [2,4]]
```

	Longitude	Latitude	City Name	Direction	Survivors
0	32.0	54.8	Smolensk	Advance	145000
1	33.2	54.9	Dorogobouge	Advance	140000
2	34.4	55.5	Chjat	Advance	127100
3	37.6	55.8	Moscou	Advance	100000
4	34.3	55.2	Wixma	Retreat	55000
5	32.0	54.6	Smolensk	Retreat	24000
6	30.4	54.4	Orscha	Retreat	20000
7	26.8	54.3	Moiodexno	Retreat	12000

Working with the Data in a Column

❖ Implement TODO block for adding a new column called ‘Percent Surviving’ by dividing the initial value with each item in the ‘Survivors’ Column

```
##TODO##
initial =
ratio =
##End of block##

percent = np.array(["{:.2%}".format(val) for val in ratio])
df['Percent Surviving'] = percent
```

Sorting Rows

❖ Use sort_values()

```
df_nba = pd.read_csv('nba_salaries.csv')
df_nba = df_nba.rename(columns={'2015-2016 SALARY':'SALARY'})
df_nba.sort_values(by = ['SALARY'], axis=0, ascending=False).head(5)
# or [:5]
```

	PLAYER	POSITION	TEAM	SALARY
169	Kobe Bryant	SF	Los Angeles Lakers	25.000000
29	Joe Johnson	SF	Brooklyn Nets	24.894863
72	LeBron James	SF	Cleveland Cavaliers	22.970500
255	Carmelo Anthony	SF	New York Knicks	22.875000
131	Dwight Howard	C	Houston Rockets	22.359364

Selecting Rows

- Often, we need to extract rows that match specific conditions
  - e.g., players in a specified team or players who earned more than $10M

- For specified rows

```
df[3:6]
```

	PLAYER	POSITION	TEAM	SALARY
3	Jeff Teague	PG	Atlanta Hawks	8.000000
4	Kyle Korver	SG	Atlanta Hawks	5.746479
5	Thabo Sefolosha	SF	Atlanta Hawks	4.000000

Selecting Rows (cont.)

❖ Implement a code for selecting rows corresponding to a specified feature, e.g., >$10M

```
##TODO##
```

	PLAYER	POSITION	TEAM	SALARY
169	Kobe Bryant	SF	Los Angeles Lakers	25.000000
29	Joe Johnson	SF	Brooklyn Nets	24.894863
72	LeBron James	SF	Cleveland Cavaliers	22.970500
255	Carmelo Anthony	SF	New York Knicks	22.875000
131	Dwight Howard	C	Houston Rockets	22.359364
...	...	...	...	...

Selecting Rows (cont.)

❖ Implement a code for selecting players who played with Stephen Curry

```
##TODO##
```

PLAYER	POSITION	TEAM	SALARY
Klay Thompson	SG	Golden State Warriors	15.501
Draymond Green	PF	Golden State Warriors	14.2609
Andrew Bogut	C	Golden State Warriors	13.8
Andre Iguodala	SF	Golden State Warriors	11.7105
Stephen Curry	PG	Golden State Warriors	11.3708
Jason Thompson	PF	Golden State Warriors	7.00847
Shaun Livingston	PG	Golden State Warriors	5.54373

Selecting Rows (cont.)

❖ Multiple Features

```
df[(df['POSITION']=='PG') & (10<=df['SALARY']) & (df['SALARY']<12)]
```

	PLAYER	POSITION	TEAM	SALARY
121	Stephen Curry	PG	Golden State Warriors	11.370786
241	Jrue Holiday	PG	New Orleans Pelicans	10.595507

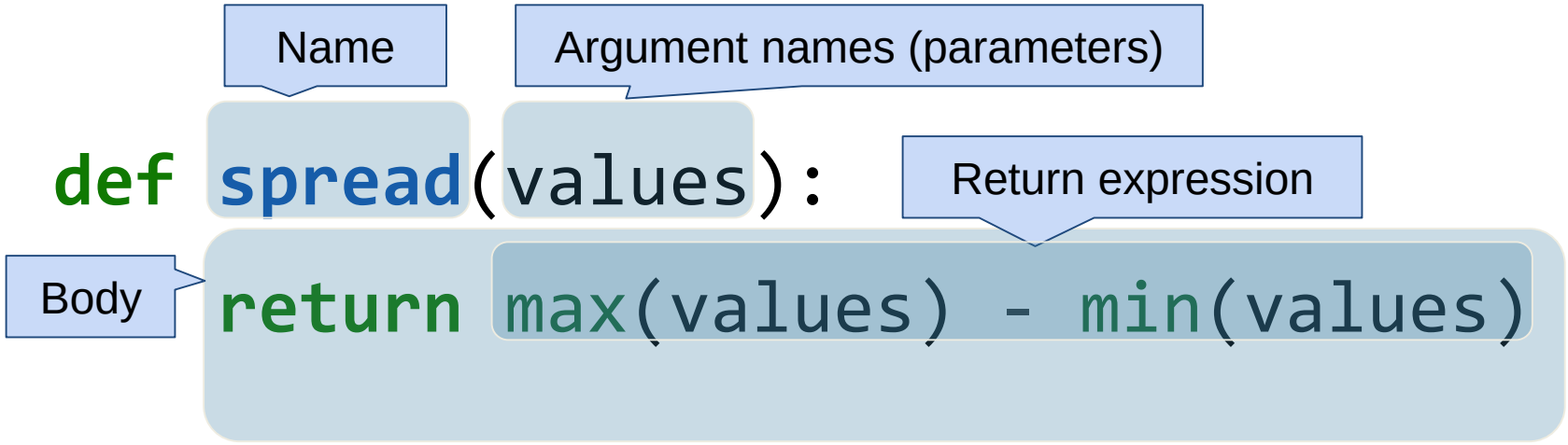
Selecting Rows (cont.)

❖ str.contains() method for partial matching

```
df[df['TEAM'].str.contains('Warriors')]
```

Def Statements

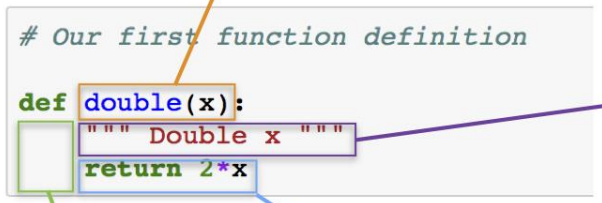
❖ User-defined functions give names to blocks of code



Defining a Function

Signature

- Calls to the function will look like this (with the same name and number of arguments). Example: `double(3)`.
- When you call `double`, the argument can be any expression. (The name `x` doesn't affect calls.)
- In the body of the function, `x` is the name of the argument, as if the body included the code `x = <the first argument>`.



Documentation ("docstring")

- Text that describes what the function does.
- Can be any string, traditionally triple-quoted so it can span several lines.
- Traditionally, the first line describes what the function does, briefly.
- Subsequent lines can give more detail and examples.
- Running `double?` will show this text, just like `max?` will show the documentation for the built-in function `max`.

Body

- All the code in here runs each time you call the function.
- The special statement `return` tells Python what the value of each call to this function is: it's the value of the expression after `return`.
- For example, the value of `double(3)` is `6`. (Remember, when the argument is `3`, it's like the body starts with `x = 3`.)
- Often, the body will have multiple lines of code that build up to computing the `return`ed value. You can write any Python code here that you could write anywhere else.

Indentation

- Each line of code in the body is indented (that is, it's preceded by spaces).
- Traditionally, we use 2 or 4 spaces. They only need to be consistent.
- This tells Python that those lines are part of the body.
- The function's body ends at any unindented line.

Discussion Question

- What does this function do? What kind of input does it take? What output will it give? What's a reasonable name?

```
def f(s):
        return np.round(s/sum(s)*100,2)
```

Apply Demo (Pandas DataFrame)

```
def cut_off_at_100(x):
        """The smaller of x and 100"""
        return min(x, 100)
```

```
ages = pd.DataFrame({'Person': np.array(['A', 'B', 'C', 'D', 'E', 'F']),
        'Age': np.array([17, 117, 52, 100, 6, 101])})

ages['Cut Off Age'] = ages['Age'].apply(cut_off_at_100)
```

	Person	Age	Cut Off Age
0	A	17	17
1	B	117	100
2	C	52	52
3	D	100	100
4	E	6	6
5	F	101	100

Prediction using Apply

❖ Data on hand

	family	father	mother	children	childNum	sex	childHeight
0	1	78.5	67.0	4	1	male	73.2
1	1	78.5	67.0	4	2	female	69.2
2	1	78.5	67.0	4	3	female	69.0
3	1	78.5	67.0	4	4	female	69.0
4	2	75.5	66.5	4	1	male	73.5
...	...	...	...	...	...	...	...
929	203	62.0	66.0	3	1	male	64.0
930	203	62.0	66.0	3	2	female	62.0
931	203	62.0	66.0	3	3	female	61.0
932	204	62.5	63.0	2	1	male	66.5
933	204	62.5	63.0	2	2	female	57.0

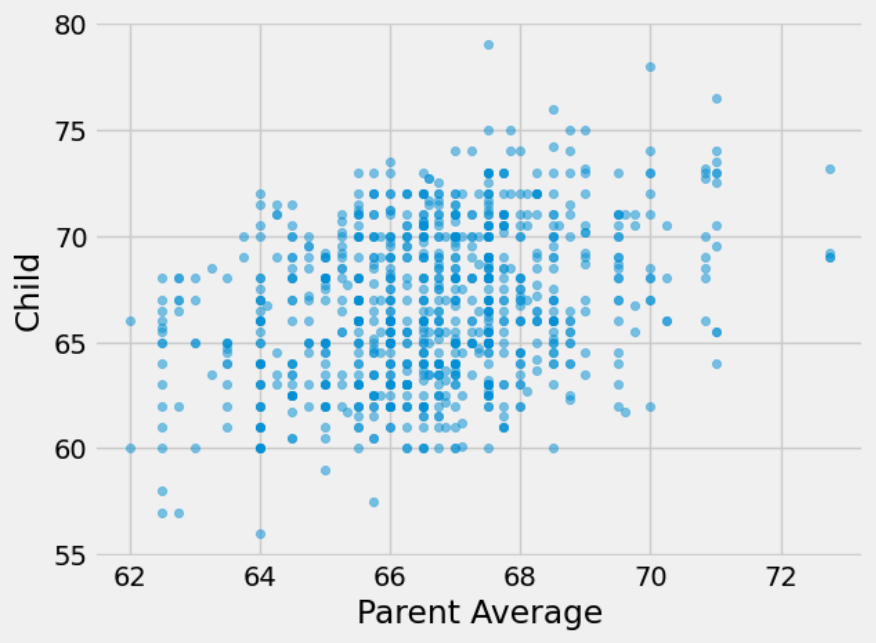
934 rows × 7 columns

Naïve approach (averaging over nearest points)

❖ You have an intuition that the child's height

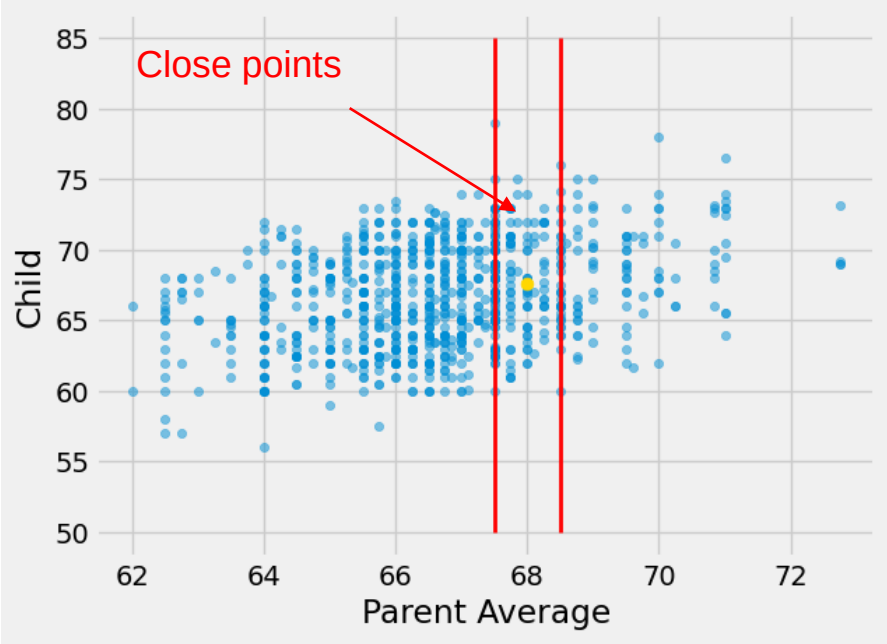
is related to the parent's height

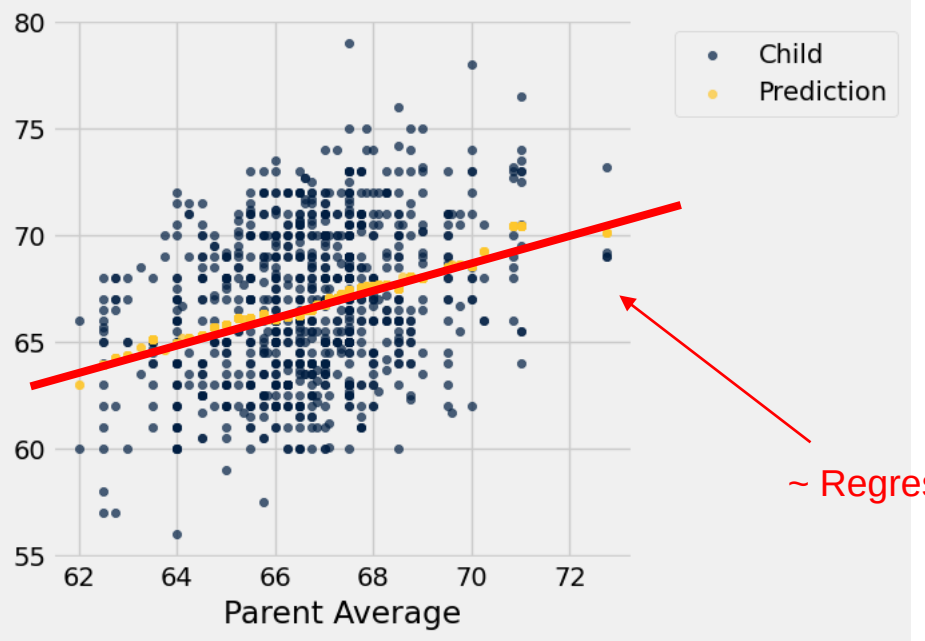
```
parent_averages = (family_heights['father'] + family_heights['mother'])/2
heights = pd.DataFrame({'Parent Average': parent_averages,
                        'Child':family_heights.childHeight})
heights.plot.scatter(x='Parent Average',y='Child', alpha=0.5)
```



Define your prediction function

```
def predict_child(p_avg):
        filter = (p_avg-0.5 <= heights['Parent Average'])\
        & (heights['Parent Average'] < p_avg+0.5)
        close_points = heights.loc[filter]
        return np.average(close_points.Child)
```





Q&A

Prof. Jeon, Sangryul

Computer Vision Lab.

Pusan National University, Korea

Tel: +82-51-510-2423

Web: http://sr-jeon.github.io/

E-mail: srjeonn@pusan.ac.kr