Code for QSS python Chapter 2: Causality

Kosuke Imai, Python code by Jeff Allen

First Printing

Section 2.1: Racial Discrimination in the Labor Market

[ ]:
```python
import pandas as pd
import numpy as np
```

[ ]:
```python
resume = pd.read_csv('resume.csv')

resume.shape
```

[ ]:
```
(4870, 4)
```

[ ]:
```python
resume.head()
```

[ ]:
```
  firstname     sex   race  call
0   Allison  female  white     0
1   Kristen  female  white     0
2   Lakisha  female  black     0
3   Latonya  female  black     0
4    Carrie  female  white     0
```

[ ]:
```python
resume.dtypes # firstname, sex, and race are currently strings
```

[ ]:
```
firstname    object
sex          object
race         object
call          int64
dtype: object
```

[ ]:
```python
resume.describe() # by default, only summarizes numeric variables
```

[ ]:
```
              call
count  4870.000000
mean      0.080493
std       0.272083
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       1.000000
```

In 2.2.5, when we discuss categorical variables, we will also explore overriding the `describe()` default behavior and alternatives for summarizing non-numeric data.

[ ]:
```
# contingency table (crosstab)
race_call_tab = pd.crosstab(resume.race, resume['call'])
# note the two ways to access a column in a data frame

race_call_tab
```

[ ]:
```
call      0    1
race
black  2278  157
white  2200  235
```

[ ]:
```
type(race_call_tab) # a data frame
```

[ ]:
```
pandas.core.frame.DataFrame
```

[ ]:
```
# the data frame's index and columns both have names
print(race_call_tab.columns)
print(race_call_tab.index)
```

```
Index([0, 1], dtype='int64', name='call')
Index(['black', 'white'], dtype='object', name='race')
```

[ ]:
```
# crosstab with margins
pd.crosstab(resume.race, resume.call, margins=True)
```

[ ]:
```
call      0    1   All
race
black  2278  157  2435
white  2200  235  2435
All    4478  392  4870
```

[ ]:
```
# overall callback rate: total callbacks divided by sample size
# using positional selection and number of rows
race_call_tab.iloc[:,1].sum() / resume.shape[0]
```

[ ]:
```
0.08049281314168377
```

[ ]:
```
# callback rate for each race race
race_call_tab.loc['black', 1] / race_call_tab.loc['black'].sum() # black
```

[ ]:
```
0.06447638603696099
```

[ ]:
```
race_call_tab.loc['white', 1] / race_call_tab.loc['white'].sum() # white
```

[ ]:
```
0.09650924024640657
```

[ ]:
```
race_call_tab.iloc[0] # the first row, using positions
```

[ ]:
```
call
0    2278
1     157
Name: black, dtype: int64
```

[ ]:
```
race_call_tab.loc['black'] # the first row, using names
```

[ ]:
```
call
0    2278
1     157
Name: black, dtype: int64
```

[ ]:
```
race_call_tab.iloc[:,1] # the second column, using positions
```

[ ]:
```
race
black    157
white    235
Name: 1, dtype: int64
```

[ ]:
```
race_call_tab.loc[:,1] # the second column, using names
```

[ ]:
```
race
black    157
white    235
Name: 1, dtype: int64
```

By coincidence, the name of the second column is also the number 1. In pandas, column names can be numeric.

[ ]:
```
resume['call'].mean() # overall callback rate
```

[ ]:
```
0.08049281314168377
```

Section 2.2: Subsetting Data in pandas

Section 2.2.1: Boolean Values and Logical Operators

[ ]:
```
type(True)
```

[ ]:
```
bool
```

[ ]:
```
int(True)
```

[ ]:
```
1
```

[ ]:
```
int(False)
```

[ ]: 0

[ ]:
```
x = pd.Series([True, False, True]) # a vector with boolean values

x.mean().round(2) # proportion of True values
```

[ ]: 0.67

[ ]:
```
x.sum() # number of True values
```

[ ]: 2

[ ]:
```
False & True
```

[ ]: False

[ ]:
```
True & True
```

[ ]: True

[ ]:
```
True | False
```

[ ]: True

[ ]:
```
False | False
```

[ ]: False

[ ]:
```
True & False & True
```

[ ]: False

[ ]:
```
# Parentheses evaluate to False
(True | False) & False
```

[ ]: False

[ ]:
```
# Parentheses evaluate to True
True | (False & False)
```

[ ]: True

[ ]:
```
# Vector-wise logical operations
TF1 = pd.Series([True, False, False])
TF2 = pd.Series([True, False, True])
TF1 | TF2
```

[ ]:
```
0     True
1    False
```

```
2     True
dtype: bool
```

[ ]:
```
TF1 & TF2
```

[ ]:
```
0     True
1    False
2    False
dtype: bool
```

Section 2.2.2: Relational Operators

[ ]:
```
4 > 3
```

[ ]:
```
True
```

[ ]:
```
"Hello" == "hello" # Python is case-sensitive
```

[ ]:
```
False
```

[ ]:
```
"Hello" != "hello"
```

[ ]:
```
True
```

[ ]:
```
x = pd.Series([3, 2, 1, -2, -1])

x >= 2
```

[ ]:
```
0     True
1     True
2    False
3    False
4    False
dtype: bool
```

[ ]:
```
x != 1
```

[ ]:
```
0     True
1     True
2    False
3     True
4     True
dtype: bool
```

[ ]:
```
# logical conjunction of two vectors with boolean values
(x > 0) & (x <= 2)
```

[ ]:
```
0    False
1     True
2     True
3    False
4    False
dtype: bool
```

[ ]:
```
# logical disjunction of two vectors with boolean values
(x > 2) | (x <= -1)
```

[ ]:
```
0     True
1    False
2    False
3     True
4     True
dtype: bool
```

[ ]:
```
x_int = (x > 0) & (x <= 2) # logical vector

x_int
```

[ ]:
```
0    False
1     True
2     True
3    False
4    False
dtype: bool
```

[ ]:
```
x_int.mean() # proportion of True values
```

[ ]:
```
0.4
```

[ ]:
```
x_int.sum() # number of True values
```

[ ]:
```
2
```

Section 2.2.3: Subsetting

[ ]:
```
# callback rate for black-sounding names
resume['call'][resume['race'] == 'black'].mean()
```

[ ]:
```
0.06447638603696099
```

[ ]:
```
# race of the first 5 observations
resume['race'][0:5]
```

[ ]:
```
0    white
1    white
```

```
2    black
3    black
4    white
Name: race, dtype: object
```

```
[ ]: # comparison of first 5 observations
     resume['race'][0:5] == 'black'
```

```
[ ]: 0    False
     1    False
     2     True
     3     True
     4    False
     Name: race, dtype: bool
```

```
[ ]: resume.shape # dimensions of the original data frame
```

```
[ ]: (4870, 4)
```

```
[ ]: # subset blacks only
     resumeB = resume.loc[resume['race'] == 'black'].copy()

     resumeB.shape # this data frame has fewer rows than the original
```

```
[ ]: (2435, 4)
```

```
[ ]: resumeB['call'].mean() # callback rate for blacks
```

```
[ ]: 0.06447638603696099
```

```
[ ]: # subset observations with black, female-sounding names
     # keep only the "call" and "firstname" variables
     resumeBf = (resume.loc[(resume.race == 'black') &
                            (resume.sex == 'female'), ['call', 'firstname']])

     '''
     Notice, we can use parentheses to break up a line and circumvent the python
     white space rules.
     '''

     resumeBf.head(n=6)
```

```
[ ]:     call firstname
     2      0   Lakisha
     3      0   Latonya
     7      0     Kenya
     8      0   Latonya
     10     0     Aisha
```

```
12     0     Aisha
```

```
[ ]: # black male
resumeBm = resume.loc[(resume.race == 'black') & (resume.sex == 'male')]

# white female
resumeWf = resume.loc[(resume.race == 'white') & (resume.sex == 'female')]

# white male
resumeWm = resume.loc[(resume.race == 'white') & (resume.sex == 'male')]
```

```
[ ]: # racial gaps
resumeWf['call'].mean() - resumeBf['call'].mean() # among females
```

```
[ ]: 0.0326468944913853
```

```
[ ]: resumeWm['call'].mean() - resumeBm['call'].mean() # among males
```

```
[ ]: 0.030407856181199011
```

Section 2.2.4: Simple Conditional Statements

```
[ ]: # where() from numpy implements vectorized if-else
resume['BlackFemale'] = (np.where((resume.race == 'black') &
                                  (resume.sex == 'female'), 1, 0))

# three-way crosstab
pd.crosstab([resume.race, resume.sex], resume.BlackFemale)
```

```
[ ]: BlackFemale      0     1
race   sex
black  female     0  1886
       male     549     0
white  female  1860     0
       male     575     0
```

```
[ ]: # drop the BlackFemale column in place
resume.drop('BlackFemale', axis=1, inplace=True)
```

Section 2.2.5: Categorical Variables

Recall, firstname, sex, and race are currently strings, but for analytical purposes, they are categorical variables because values in these columns belong to one of a limited number of groups. Let's convert firstname, sex, and race to the pandas categorical data type.

```
[ ]: # first, store the variable names in a list for more compact code
cat_vars = ['firstname', 'sex', 'race']
```

```
resume[cat_vars] = resume[cat_vars].astype('category')

resume.dtypes # now the variables are categorical
```

[ ]:
```
firstname    category
sex          category
race         category
call            int64
dtype: object
```

[ ]:
```
resume['race'][0:5]
```

[ ]:
```
0    white
1    white
2    black
3    black
4    white
Name: race, dtype: category
Categories (2, object): ['black', 'white']
```

[ ]:
```
resume['race'].cat.categories
```

[ ]:
```
Index(['black', 'white'], dtype='object')
```

[ ]:
```
resume['race'].cat.codes
```

[ ]:
```
0       1
1       1
2       0
3       0
4       1
       ..
4865    0
4866    0
4867    1
4868    0
4869    1
Length: 4870, dtype: int8
```

[ ]:
```
resume['race'].value_counts()
```

[ ]:
```
race
black    2435
white    2435
Name: count, dtype: int64
```

[ ]:
```
resume['race'].value_counts(normalize=True)
```

[ ]:
```
race
black    0.5
white    0.5
Name: proportion, dtype: float64
```

[ ]:
```
resume[cat_vars].describe()
```

[ ]:

	firstname	sex	race
count	4870	4870	4870
unique	36	2	2
top	Tamika	female	black
freq	256	3746	2435

[ ]:
```
resume.describe(include='all') # output is not visually appealing
```

[ ]:

	firstname	sex	race	call
count	4870	4870	4870	4870.000000
unique	36	2	2	NaN
top	Tamika	female	black	NaN
freq	256	3746	2435	NaN
mean	NaN	NaN	NaN	0.080493
std	NaN	NaN	NaN	0.272083
min	NaN	NaN	NaN	0.000000
25%	NaN	NaN	NaN	0.000000
50%	NaN	NaN	NaN	0.000000
75%	NaN	NaN	NaN	0.000000
max	NaN	NaN	NaN	1.000000

[ ]:
```
# create a new factor variable
resume['type'] = np.nan
(resume.loc[(resume.race == "black") &
            (resume.sex == "female"), 'type']) = 'BlackFemale'
(resume.loc[(resume.race == "black") &
            (resume.sex == "male"), 'type']) = 'BlackMale'
(resume.loc[(resume.race == "white") &
            (resume.sex == "female"), 'type']) = 'WhiteFemale'
(resume.loc[(resume.race == "white") &
            (resume.sex == "male"), 'type']) = 'WhiteMale'
```

[ ]:
```
# A faster alternative:

# create a list of n-1 conditions
conditions = [
      (resume.race == "black") & (resume.sex == "female")
    , (resume.race == "black") & (resume.sex == "male")
    , (resume.race == "white") & (resume.sex == "female")
]
```

```python
# create a list of choices corresponding to the conditions
choices  = ['BlackFemale', 'BlackMale', 'WhiteFemale']

# create a new column in the data frame based on the conditions
# the third argument is the default value if none of the conditions is met
resume["type_alt"] = np.select(conditions, choices, 'WhiteMale')

# check that the results are the same
resume['type'].equals(resume['type_alt'])
```

[ ]: True

[ ]:
```python
# drop the alternative column
resume.drop('type_alt', axis=1, inplace=True)

resume.dtypes # type is still a string
```

[ ]:
```
firstname    category
sex          category
race         category
call            int64
type           object
dtype: object
```

[ ]:
```python
# coerce the new variable into a categorical variable
resume['type'] = resume['type'].astype('category')

# list the categories
resume['type'].cat.categories
```

[ ]: Index(['BlackFemale', 'BlackMale', 'WhiteFemale', 'WhiteMale'], dtype='object')

[ ]:
```python
# obtain the number of observations in each category
resume['type'].value_counts(sort=False)
```

[ ]:
```
type
BlackFemale    1886
BlackMale       549
WhiteFemale    1860
WhiteMale       575
Name: count, dtype: int64
```

[ ]:
```python
# compute callback rate for each category
resume.groupby('type')['call'].mean()
```

[ ]:
```
type
BlackFemale    0.066278
BlackMale      0.058288
```

```
WhiteFemale    0.098925
WhiteMale      0.088696
Name: call, dtype: float64
```

[ ]:
```
# compute callback rate for each first name
callback_name = resume.groupby('firstname')['call'].mean()

# look at the names with the lowest callback rates
callback_name.sort_values().head(n=10)
```

[ ]:
```
firstname
Aisha       0.022222
Rasheed     0.029851
Keisha      0.038251
Tremayne    0.043478
Kareem      0.046875
Darnell     0.047619
Tyrone      0.053333
Hakim       0.054545
Tamika      0.054688
Lakisha     0.055000
Name: call, dtype: float64
```

[ ]:
```
# look at the names with the highest callback rates
callback_name.sort_values(ascending=False).head(n=10)
```

[ ]:
```
firstname
Brad        0.158730
Jay         0.134328
Kristen     0.131455
Carrie      0.130952
Meredith    0.101604
Sarah       0.098446
Laurie      0.097436
Jermaine    0.096154
Ebony       0.096154
Allison     0.094828
Name: call, dtype: float64
```

Section 2.3: Causal Effects and the Counterfactual

[ ]:
```
resume.iloc[0]
```

[ ]:
```
firstname    Allison
sex           female
race           white
call               0
```

```
type         WhiteFemale
Name: 0, dtype: object
```

Section 2.4: Randomized Controlled Trials

Section 2.4.1: The Role of Randomization

Section 2.4.2: Social Pressure and Voter Turnout

[ ]:
```
social = pd.read_csv('social.csv')

social.describe().round(2)
```

[ ]:

	yearofbirth	primary2004	primary2006	hhsize
count	305866.00	305866.00	305866.00	305866.00
mean	1956.21	0.40	0.31	2.18
std	14.45	0.49	0.46	0.79
min	1900.00	0.00	0.00	1.00
25%	1947.00	0.00	0.00	2.00
50%	1956.00	0.00	0.00	2.00
75%	1965.00	1.00	1.00	2.00
max	1986.00	1.00	1.00	8.00

[ ]:
```
social.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 305866 entries, 0 to 305865
Data columns (total 6 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   sex          305866 non-null  object
 1   yearofbirth  305866 non-null  int64
 2   primary2004  305866 non-null  int64
 3   messages     305866 non-null  object
 4   primary2006  305866 non-null  int64
 5   hhsize       305866 non-null  int64
dtypes: int64(4), object(2)
memory usage: 14.0+ MB
```

[ ]:
```
# convert sex and messages to categorical variables
social[['sex', 'messages']] = social[['sex', 'messages']].astype('category')

social['messages'].cat.categories
```

[ ]:
```
Index(['Civic Duty', 'Control', 'Hawthorne', 'Neighbors'], dtype='object')
```

[ ]:
```
# re-order the categories, so the control group is first
social['messages'] = social['messages'].cat.reorder_categories(
    ['Control', 'Civic Duty', 'Hawthorne', 'Neighbors'])
```

```
social['messages'].cat.categories
```

[ ]: Index(['Control', 'Civic Duty', 'Hawthorne', 'Neighbors'], dtype='object')

[ ]:
```
'''
Even though we re-ordered the levels, we have not converted messages to an
ordered categorical variable.
'''
social['messages'].cat.ordered
```

[ ]: False

[ ]:
```
# turnout for each group
social.groupby('messages')['primary2006'].mean()
```

[ ]:
```
messages
Control       0.296638
Civic Duty    0.314538
Hawthorne     0.322375
Neighbors     0.377948
Name: primary2006, dtype: float64
```

[ ]:
```
# turnout for control group
social['primary2006'][social.messages == 'Control'].mean()
```

[ ]: 0.2966383083302395

[ ]:
```
# subtract control group turnout from each group
(social.groupby('messages')['primary2006'].mean() -
 social['primary2006'][social.messages == 'Control'].mean())
```

[ ]:
```
messages
Control       0.000000
Civic Duty    0.017899
Hawthorne     0.025736
Neighbors     0.081310
Name: primary2006, dtype: float64
```

[ ]:
```
social['age'] = 2006 - social['yearofbirth'] # create age variable

# calculate mean of age for each message type
social.groupby('messages')['age'].mean()
```

[ ]:
```
messages
Control       49.813546
Civic Duty    49.659035
Hawthorne     49.704795
```

```
Neighbors     49.852936
Name: age, dtype: float64
```

[ ]:
```
# calculate the mean of primary2004 for each message type
social.groupby('messages')['primary2004'].mean()
```

[ ]:
```
messages
Control       0.400339
Civic Duty    0.399445
Hawthorne     0.403230
Neighbors     0.406665
Name: primary2004, dtype: float64
```

[ ]:
```
# calculate the mean of hhsize for each message type
social.groupby('messages')['hhsize'].mean()
```

[ ]:
```
messages
Control       2.183667
Civic Duty    2.189126
Hawthorne     2.180138
Neighbors     2.187770
Name: hhsize, dtype: float64
```

Section 2.5: Observational Studies

Section 2.5.1: Minimum Wage and Unemployment

If we know that certain variables should be categorical ahead of time, we can specify that in pd.read_csv() using the dtype argument and a dictionary.

[ ]:
```
minwage = pd.read_csv('minwage.csv',
                      dtype={'chain': 'category', 'location': 'category'})

minwage.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 358 entries, 0 to 357
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   chain       358 non-null    category
 1   location    358 non-null    category
 2   wageBefore  358 non-null    float64
 3   wageAfter   358 non-null    float64
 4   fullBefore  358 non-null    float64
 5   fullAfter   358 non-null    float64
 6   partBefore  358 non-null    float64
 7   partAfter   358 non-null    float64
```

```
dtypes: category(2), float64(6)
memory usage: 17.8 KB
```

[ ]:
```
minwage.shape
```

[ ]:
```
(358, 8)
```

[ ]:
```
minwage.describe().round(2)
```

[ ]:

	wageBefore	wageAfter	fullBefore	fullAfter	partBefore	partAfter
count	358.00	358.00	358.00	358.00	358.00	358.00
mean	4.62	4.99	8.47	8.36	18.75	18.69
std	0.35	0.26	8.70	7.81	10.29	10.57
min	4.25	4.25	0.00	0.00	0.00	0.00
25%	4.25	5.05	2.12	2.00	11.00	11.00
50%	4.50	5.05	6.00	6.00	16.25	17.00
75%	4.99	5.05	12.00	12.00	25.00	25.00
max	5.75	6.25	60.00	40.00	60.00	60.00

[ ]:
```
minwage['chain'].value_counts()
```

[ ]:
```
chain
burgerking    149
roys           88
kfc            75
wendys         46
Name: count, dtype: int64
```

[ ]:
```
minwage['location'].value_counts()
```

[ ]:
```
location
northNJ      146
PA            67
southNJ       67
centralNJ     45
shoreNJ       33
Name: count, dtype: int64
```

[ ]:
```
# subsetting the data into two states
minwageNJ = minwage.loc[minwage.location != 'PA'].copy()
minwagePA = minwage.loc[minwage.location == 'PA'].copy()

# proportion of restaurants whose wage is less than $5.05
(minwageNJ['wageBefore'] < 5.05).mean() # NJ before
```

[ ]:
```
0.9106529209621993
```

[ ]:
```
(minwageNJ['wageAfter'] < 5.05).mean() # NJ after
```

[ ]: 0.003436426116838488

[ ]:
```
(minwagePA['wageBefore'] < 5.05).mean() # PA before
```

[ ]: 0.9402985074626866

[ ]:
```
(minwagePA['wageAfter'] < 5.05).mean() # PA after
```

[ ]: 0.9552238805970149

[ ]:
```
# create a variable for proportion of full-time employees in NJ and PA
minwageNJ['fullPropAfter'] = (
    minwageNJ['fullAfter'] / (minwageNJ['fullAfter'] + minwageNJ['partAfter'])
    )

minwagePA['fullPropAfter'] = (
    minwagePA['fullAfter'] / (minwagePA['fullAfter'] + minwagePA['partAfter'])
    )

# compute the difference in means
minwageNJ['fullPropAfter'].mean() - minwagePA['fullPropAfter'].mean()
```

[ ]: 0.04811886142291416

Section 2.5.2: Confounding Bias

[ ]:
```
minwageNJ['chain'].value_counts(sort=False, normalize=True)
```

[ ]:
```
chain
burgerking    0.405498
kfc           0.223368
roys          0.250859
wendys        0.120275
Name: proportion, dtype: float64
```

[ ]:
```
minwagePA['chain'].value_counts(sort=False, normalize=True)
```

[ ]:
```
chain
burgerking    0.462687
kfc           0.149254
roys          0.223881
wendys        0.164179
Name: proportion, dtype: float64
```

[ ]:
```
# subset Burger King only
minwageNJ_bk = minwageNJ.loc[minwageNJ.chain == 'burgerking'].copy()
minwagePA_bk = minwagePA.loc[minwagePA.chain == 'burgerking'].copy()
```

```
# comparison of full-time employment rates
minwageNJ_bk['fullPropAfter'].mean() - minwagePA_bk['fullPropAfter'].mean()
```

[ ]: 0.03643933939149829

[ ]:
```
minwageNJ_bk_subset = (
    minwageNJ_bk.loc[(minwageNJ_bk.location != 'shoreNJ') &
                     (minwageNJ_bk.location != 'centralNJ')].copy()
)

(minwageNJ_bk_subset['fullPropAfter'].mean() -
 minwagePA_bk['fullPropAfter'].mean())
```

[ ]: 0.031498534750908636

Section 2.5.3: Before-and-After and Difference-in-Differences Designs

[ ]:
```
# full-time employment proportion in the previous period for NJ
minwageNJ['fullPropBefore'] = (
    minwageNJ['fullBefore'] /
    (minwageNJ['fullBefore'] + minwageNJ['partBefore'])
)

# mean difference before and after the minimum wage increase for NJ
NJdiff = minwageNJ['fullPropAfter'].mean() - minwageNJ['fullPropBefore'].mean()

NJdiff
```

[ ]: 0.0238747402131399

[ ]:
```
# full-time employment proportion in the previous period for PA
minwagePA['fullPropBefore'] = (
    minwagePA['fullBefore'] /
    (minwagePA['fullBefore'] + minwagePA['partBefore'])
)

# mean difference before and after the minimum wage increase for PA
PAdiff = minwagePA['fullPropAfter'].mean() - minwagePA['fullPropBefore'].mean()

# difference-in-differences
NJdiff - PAdiff
```

[ ]: 0.06155831231224712

Section 2.6: Descriptive Statistics for a Single Variable

Section 2.6.1: Quantiles

[ ]:
```
# cross-section comparison between NJ and PA
minwageNJ['fullPropAfter'].median() - minwagePA['fullPropAfter'].median()
```

[ ]: 0.07291666666666669

[ ]:
```
# before and after comparison
NJdiff_med = (minwageNJ['fullPropAfter'].median() -
              minwageNJ['fullPropBefore'].median())

NJdiff_med.round(3)
```

[ ]: 0.025

[ ]:
```
# median difference-in-differences
PAdiff_med = (minwagePA['fullPropAfter'].median() -
              minwagePA['fullPropBefore'].median())

NJdiff_med - PAdiff_med
```

[ ]: 0.037019230769230804

[ ]:
```
# describe() shows quartiles as well as minimum, maximum, and mean
minwageNJ['wageBefore'].describe().round(2)
```

[ ]:
```
count    291.00
mean       4.61
std        0.34
min        4.25
25%        4.25
50%        4.50
75%        4.87
max        5.75
Name: wageBefore, dtype: float64
```

[ ]:
```
minwageNJ['wageAfter'].describe().round(2)
```

[ ]:
```
count    291.00
mean       5.08
std        0.11
min        5.00
25%        5.05
50%        5.05
75%        5.05
max        5.75
Name: wageAfter, dtype: float64
```

```
[ ]: # find the interquartile range (IQR)
     (minwageNJ['wageBefore'].quantile(0.75) -
      minwageNJ['wageBefore'].quantile(0.25)).round(2)
```

```
[ ]: 0.62
```

```
[ ]: minwageNJ['wageAfter'].quantile(0.75) - minwageNJ['wageAfter'].quantile(0.25)
```

```
[ ]: 0.0
```

```
[ ]: # deciles (10 groups)
     # use np.arange(start, stop, step) to generate sequence; stop is not included
     minwageNJ['wageBefore'].quantile(np.arange(0, 1.1, 0.1))
```

```
[ ]: 0.0    4.25
     0.1    4.25
     0.2    4.25
     0.3    4.25
     0.4    4.50
     0.5    4.50
     0.6    4.65
     0.7    4.75
     0.8    5.00
     0.9    5.00
     1.0    5.75
     Name: wageBefore, dtype: float64
```

```
[ ]: minwageNJ['wageAfter'].quantile(np.arange(0, 1.1, 0.1))
```

```
[ ]: 0.0    5.00
     0.1    5.05
     0.2    5.05
     0.3    5.05
     0.4    5.05
     0.5    5.05
     0.6    5.05
     0.7    5.05
     0.8    5.05
     0.9    5.15
     1.0    5.75
     Name: wageAfter, dtype: float64
```

Section 2.6.2: Standard Deviation

```
[ ]: (np.sqrt((minwageNJ['fullPropAfter'] -
               minwageNJ['fullPropBefore']).pow(2).mean()))
```

```
[ ]: 0.3014668578470611
```

[ ]:
```
(minwageNJ['fullPropAfter'] - minwageNJ['fullPropBefore']).mean()
```

[ ]: 0.023874740213139886

[ ]:
```
# standard deviation
minwageNJ['fullPropBefore'].std()
```

[ ]: 0.23045922465419544

[ ]:
```
minwageNJ['fullPropAfter'].std()
```

[ ]: 0.25100159189283716

[ ]:
```
# variance
minwageNJ['fullPropBefore'].var()
```

[ ]: 0.053111454228212916

[ ]:
```
minwageNJ['fullPropAfter'].var()
```

[ ]: 0.06300179913273839