

# Intro to Programming for Public Policy Week 5

## More Pandas

Eric Potash

April 27, 2018

## Read a CSV

```
>>> df = pd.read_csv('salaries.csv')  
>>> df.shape  
(33183, 8)
```

```
In [4]: df = pd.read_csv('~/.salaries.csv')
df
```

```
Out[4]:
```

	Name	Job Titles	Department	Full or Part-Time	Salary or Hourly	Typical Hours	Annual Salary	Hourly Rate
0	AARON, JEFFERY M	SERGEANT	POLICE	F	Salary	NaN	\$101442.00	NaN
1	AARON, KARINA	POLICE OFFICER (ASSIGNED AS DETECTIVE)	POLICE	F	Salary	NaN	\$94122.00	NaN
2	AARON, KIMBERLEI R	CHIEF CONTRACT EXPEDITER	GENERAL SERVICES	F	Salary	NaN	\$101592.00	NaN
3	ABAD JR, VICENTE M	CIVIL ENGINEER IV	WATER MGMNT	F	Salary	NaN	\$110064.00	NaN
4	ABASCAL, REECE E	TRAFFIC CONTROL AIDE-HOURLY	OEMC	P	Hourly	20.0	NaN	\$19.86
5	ABBASI,	STAFF ASST TO THE	CITY	F	Salary	NaN	\$50436.00	NaN

Figure 1: Jupyter Notebook Representation of DataFrame

# Columns

```
>>> df.columns
Index(['Name', 'Job Titles', 'Department',
       'Full or Part-Time', 'Salary or Hourly',
       'Typical Hours', 'Annual Salary',
       'Hourly Rate'],
      dtype='object')
```

# Data Types

```
>>> df.dtypes
Name                object
Job Titles           object
Department           object
Full or Part-Time    object
Salary or Hourly     object
Typical Hours        float64
Annual Salary        object
Hourly Rate          object
dtype: object
```

describe()

```
>>> df['Typical Hours'].describe()
count      8022.000000
mean       34.507604
std        9.252077
min        10.000000
25%        20.000000
50%        40.000000
75%        40.000000
max        40.000000
Name: Typical Hours, dtype: float64
```

mean()

We can get just the mean:

```
>>> df['Typical Hours'].mean()  
34.507604088755919
```

## Series.isnull()

Again we can look at the null values:

```
>>> df['Typical Hours'].isnull()
...
33175    False
33176     True
33177     True
33178     True
33179     True
33180     True
33181     True
33182     True
Name: Typical Hours, dtype: bool
```

Instead of summing, it can be more useful to take the mean, which corresponds to the *proportion* of missing values:

```
>>> df['Typical Hours'].isnull().mean()
0.75824970617484855
```



## DataFrame.isnull()

- ▶ Similar to a Series, we can call `isnull()` on a DataFrame
- ▶ Result is now a DataFrame, with the same rows and columns, but all values are booleans indicating whether the value in the original table was null

```
In [5]: df.isnull()
```

```
Out[5]:
```

	Name	Job Titles	Department	Full or Part-Time	Salary or Hourly	Typical Hours	Annual Salary	Hourly Rate
0	False	False	False	False	False	True	False	True
1	False	False	False	False	False	True	False	True
2	False	False	False	False	False	True	False	True
3	False	False	False	False	False	True	False	True
4	False	False	False	False	False	False	True	False
5	False	False	False	False	False	True	False	True
6	False	False	False	False	False	False	True	False
7	False	False	False	False	False	False	True	False
8	False	False	False	False	False	True	False	True
9	False	False	False	False	False	True	False	True

## Missing proportions

When all columns are numeric (or boolean), we can call `mean()` on the whole DataFrame:

```
>>> df.isnull().mean()
Name                0.00000
Job Titles          0.00000
Department          0.00000
Full or Part-Time   0.00000
Salary or Hourly    0.00000
Typical Hours       0.75825
Annual Salary       0.24175
Hourly Rate         0.75825
dtype: float64
```

## value\_counts

Given a series, we can get a new series which is a histogram of the original using the `value_counts` function:

```
>>> df.Department.value_counts()
```

POLICE	13414
--------	-------

FIRE	4641
------	------

STREETS & SAN	2198
---------------	------

OEMC	2102
------	------

WATER MGMNT	1879
-------------	------

AVIATION	1629
----------	------

TRANSPORTN	1140
------------	------

PUBLIC LIBRARY	1015
----------------	------

GENERAL SERVICES	980
------------------	-----

FAMILY & SUPPORT	615
------------------	-----

FINANCE	560
---------	-----

HEALTH	488
--------	-----

CITY COUNCIL	411
--------------	-----

LAW	407
-----	-----

BUILDINGS	269
-----------	-----

## Full or Part-Time

```
>>> df['Full or Part-Time'].value_counts()
F      31090
P       2093
Name: Full or Part-Time, dtype: int64
```

## Number of detectives

```
>>> df[df.Department == 'POLICE']['Job Titles'].value_counts()
POLICE OFFICER                                     9520
SERGEANT                                           1202
POLICE OFFICER (ASSIGNED AS DETECTIVE)            989
LIEUTENANT                                         265
POLICE OFFICER / FLD TRNG OFFICER                 231
DETENTION AIDE                                    221
POLICE ADMINISTRATIVE CLERK                       126
POLICE OFFICER (ASSIGNED AS EVIDENCE TECHNICIAN)  103
SENIOR DATA ENTRY OPERATOR                       83
COMMANDER                                          45
CLERK III                                         45
POLICE OFFICER/EXPLSV DETECT K9 HNDLR            44
CAPTAIN                                           33
PROPERTY CUSTODIAN                               33
POLICE OFFICER (ASGND AS MARINE OFFICER)         31
TIMEKEEPER - CPD                                 30
POLICE OFFICER (ASSIGNED AS CANINE HANDLER)       23
FREEDOM OF INFORMATION ACT OFFICER              12
```

# Type conversion

- ▶ We could prefer to store full or part time status as a boolean.

```
>>> df['Full-time'] = df['Full or Part-Time'] == 'F'
```

How can we subset to the full-time employees?

# Type conversion

- ▶ We could prefer to store full or part time status as a boolean.

```
>>> df['Full-time'] = df['Full or Part-Time'] == 'F'
```

How can we subset to the full-time employees?

- ▶ 

```
>>> df[df['Full-time']]
```

How can we subset to part-time employees?

# Type conversion

- ▶ We could prefer to store full or part time status as a boolean.

```
>>> df['Full-time'] = df['Full or Part-Time'] == 'F'
```

How can we subset to the full-time employees?

- ▶ 

```
>>> df[df['Full-time']]
```

How can we subset to part-time employees?

- ▶ 

```
>>> df[~df['Full-time']]
```



## apply

A very powerful tool for extending pandas is `apply`. This produces a new series by calling a function on each element in an existing series.

```
>>> df['Typical Hours'].apply(np.sqrt)
```

```
...
```

```
33168    4.472136
```

```
33169    6.324555
```

```
33170         NaN
```

```
33171         NaN
```

```
33172         NaN
```

```
33173         NaN
```

```
33174    6.324555
```

```
33175    6.324555
```

```
33176         NaN
```

```
33177         NaN
```

```
33178         NaN
```

```
33179         NaN
```

```
33180         NaN
```

```
33181         NaN
```

## More interesting example

```
def get_first_name(name):  
    first_middle = name.split(', ')[1]  
    first = first_middle.split(' ')[0]  
  
    return first
```

```
>>> df.Name.apply(get_first_name)  
...  
33176      ARTUR  
33177      DAWID  
33178  KATARZYNA  
33179      LAURA  
33180      MARK  
33181      CARLO  
33182  DARIUSZ  
Name: Name, Length: 33183, dtype: object
```

## Count first names

```
>>> df['First Name'] = df.Name.apply(get_first_name)
>>> df['First Name'].value_counts()
MICHAEL      1151
JOHN          856
JAMES         645
ROBERT        587
JOSEPH        540
DAVID         511
DANIEL        510
THOMAS        471
ANTHONY       411
WILLIAM       376
KEVIN         354
...
```

## String columns

Another way to work with string columns is through the `.str` attribute. For example:

```
>>> (df['Job Titles'].str.find('DETECTIVE') > 0)
0      False
1       True
2      False
3      False
4      False
5      False
6      False
7      False
8      False
9      False
...
```

```
>>> (df['Job Titles'].str.find('DETECTIVE') > 0).sum()
989
```

## Salaries

Also use the `.str` attribute to help parse the salaries.

- ▶ First by dropping the first character:

```
>>> df['Annual Salary'].str[1:]  
...  
33180      90024.00  
33181      93354.00  
33182     115932.00  
Name: Annual Salary, dtype: object
```

# Salaries

Also use the `.str` attribute to help parse the salaries.

- ▶ First by dropping the first character:

```
>>> df['Annual Salary'].str[1:]  
...  
33180      90024.00  
33181      93354.00  
33182     115932.00  
Name: Annual Salary, dtype: object
```

- ▶ Next by converting to float using the `astype` method:

```
>>> df['Annual Salary'].str[1:].astype(float)  
...  
33180      90024.0  
33181      93354.0  
33182     115932.0  
Name: Annual Salary, dtype: float64
```

## Salaries continues

- Replace the original column with the numeric conversion:

```
>>> df['Annual Salary'] = \
...     df['Annual Salary'].str[1:].astype(float)
```

- Now can get descriptive statistics:

```
>>> df['Annual Salary'].describe()
count      25161.000000
mean       86786.999790
std        21041.354602
min         7200.000000
25%        76266.000000
50%        90024.000000
75%        96060.000000
max        300000.000000
Name: Annual Salary, dtype: float64
```

## sort\_values

Now that salaries are numeric, we can meaningfully sort the table by them:

```
In [17]: df.sort_values('Annual Salary')
```

```
Out[17]:
```

	Name	Job Titles	Department	Full or Part-Time	Salary or Hourly	Typical Hours	Annual Salary	Hourly Rate
2386	BLONSKI, KATHERINE E	ALDERMANIC AIDE	CITY COUNCIL	P	Salary	NaN	7200.0	NaN
29311	TERRELL HART, ADRIENNE	ALDERMANIC AIDE	CITY COUNCIL	F	Salary	NaN	12840.0	NaN
6673	DAVIS, PATRICIA A	ALDERMANIC AIDE	CITY COUNCIL	P	Salary	NaN	13800.0	NaN
7808	DUKES, DOROTHY L	ALDERMANIC AIDE	CITY COUNCIL	F	Salary	NaN	15000.0	NaN
29203	TAYLOR, INES W	ALDERMANIC AIDE	CITY COUNCIL	F	Salary	NaN	15000.0	NaN
3535	BURKS, BRITTANY S	ALDERMANIC AIDE	CITY COUNCIL	F	Salary	NaN	15012.0	NaN



# sort\_values(ascending=False)

```
In [18]: df.sort_values('Annual Salary', ascending=False)
```

```
Out[18]:
```

	Name	Job Titles	Department	Full or Part-Time	Salary or Hourly	Typical Hours	Annual Salary	Hourly Rate
8439	EVANS, GINGER S	COMMISSIONER OF AVIATION	AVIATION	F	Salary	NaN	300000.0	NaN
14221	JOHNSON, EDDIE T	SUPERINTENDENT OF POLICE	POLICE	F	Salary	NaN	260004.0	NaN
8198	EMANUEL, RAHM	MAYOR	MAYOR'S OFFICE	F	Salary	NaN	216210.0	NaN
26424	SANTIAGO, JOSE A	FIRE COMMISSIONER	FIRE	F	Salary	NaN	202728.0	NaN
9226	FORD II, RICHARD C	FIRST DEPUTY FIRE COMMISSIONER	FIRE	F	Salary	NaN	197736.0	NaN

# head and tail

```
In [20]: df.sort_values('Annual Salary').head(3)
```

Out[20]:

	Name	Job Titles	Department	Full or Part-Time	Salary or Hourly	Typical Hours	Annual Salary	Hourly Rate
2386	BLONSKI, KATHERINE E	ALDERMANIC AIDE	CITY COUNCIL	P	Salary	NaN	7200.0	NaN
29311	TERRELL HART, ADRIENNE	ALDERMANIC AIDE	CITY COUNCIL	F	Salary	NaN	12840.0	NaN
6673	DAVIS, PATRICIA A	ALDERMANIC AIDE	CITY COUNCIL	P	Salary	NaN	13800.0	NaN

```
In [21]: df.sort_values('Annual Salary').tail(3)
```

Out[21]:

	Name	Job Titles	Department	Full or Part-Time	Salary or Hourly	Typical Hours	Annual Salary	Hourly Rate
33169	ZWARYCZ, THOMAS J	POOL MOTOR TRUCK DRIVER	WATER MGMNT	F	Hourly	40.0	NaN	\$35.60
33174	ZYGADLO, JOHN P	MACHINIST (AUTOMOTIVE)	GENERAL SERVICES	F	Hourly	40.0	NaN	\$46.35
33175	ZYGADLO, MICHAEL J	FRM OF MACHINISTS - AUTOMOTIVE	GENERAL SERVICES	F	Hourly	40.0	NaN	\$48.85

# Series.hist()

```
In [22]: df['Annual Salary'].hist()
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c93f2a4d0>
```

