Pandas

- It is an IPython library for data manipulation
- Has functions to work with any kind of data such as tabular data

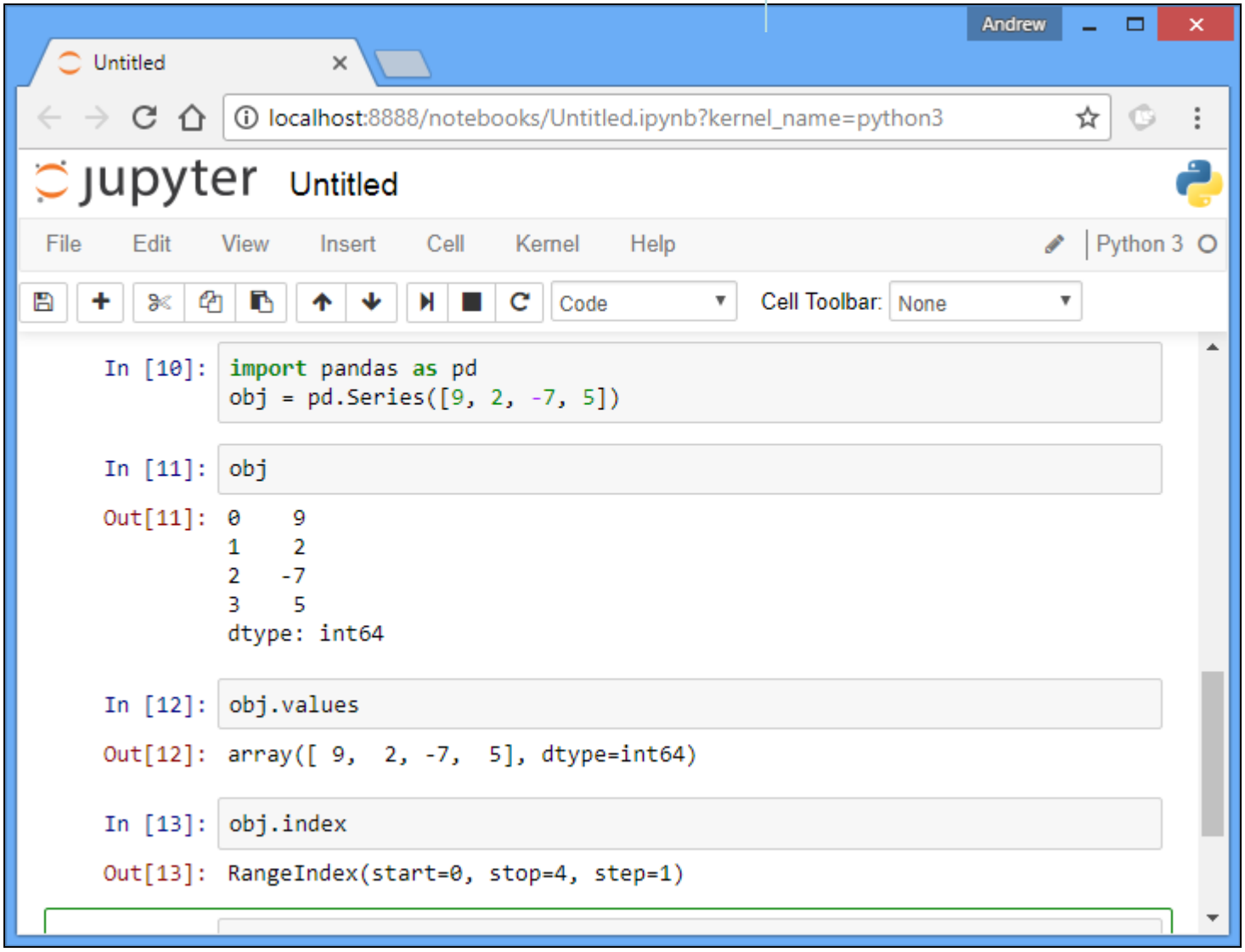
Data Structures of Pandas

- All data representation in Pandas is done using one of two data structures:
  - Series
  - DataFrame

Series

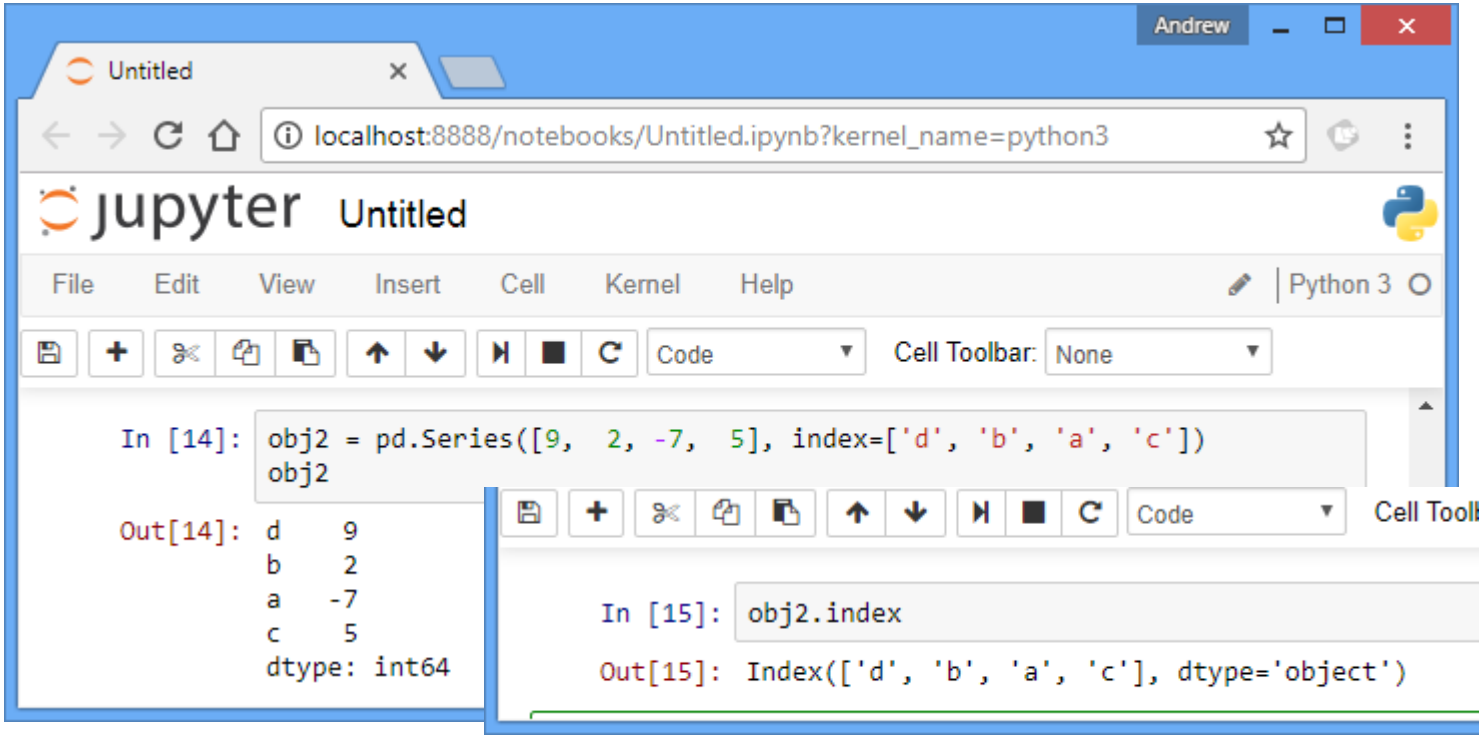
- A Series is a one-dimensional array-like **object** containing a sequence of values and an associated array of data labels, called its *index*
- A Series object wraps both its values and indices

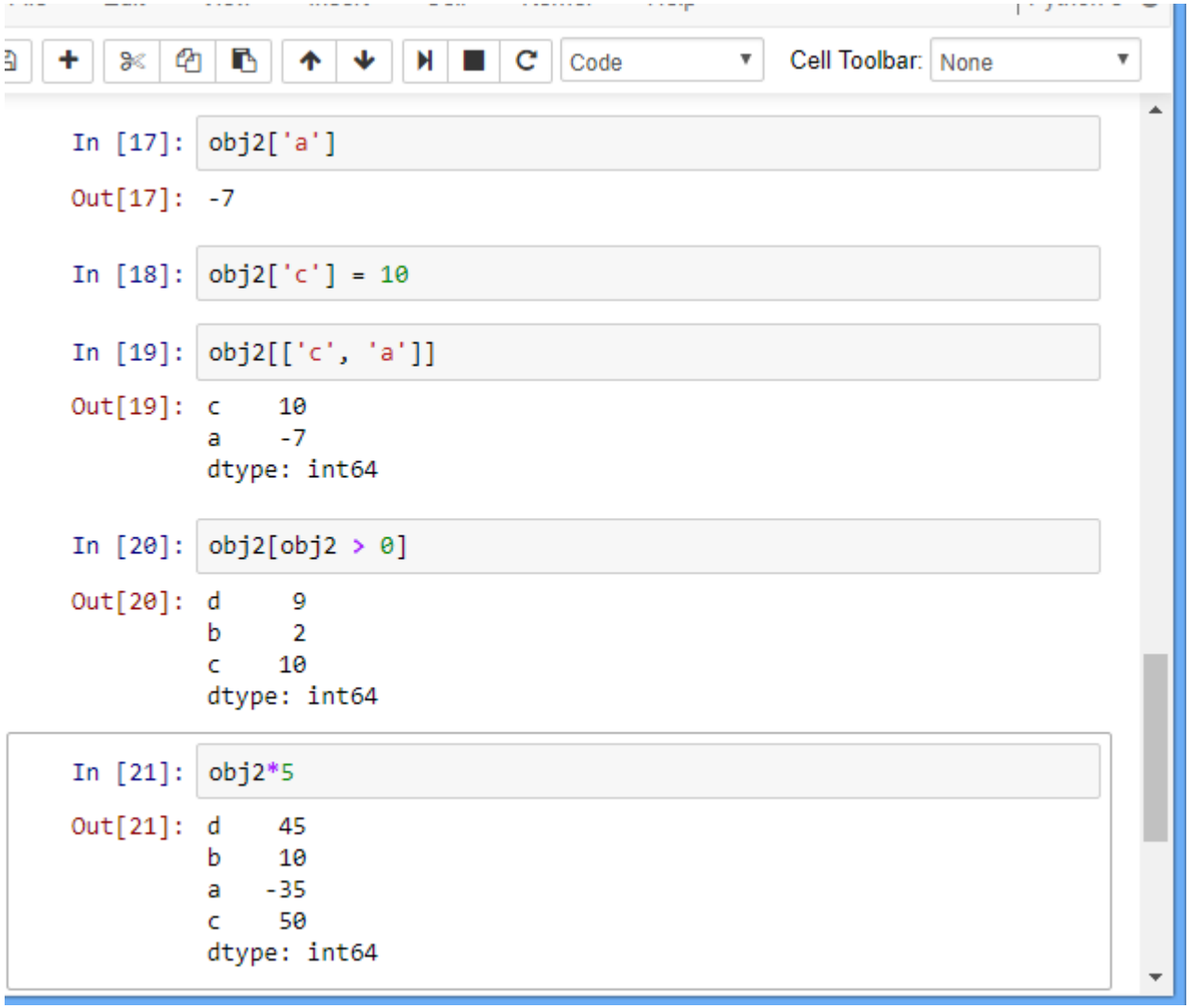
Series


Andrew
Untitled
localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3
jupyter Untitled
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None
In [10]:
import pandas as pd
obj = pd.Series([9, 2, -7, 5])
In [11]:
obj
Out[11]:
0 9
1 2
2 -7
3 5
dtype: int64
In [12]:
obj.values
Out[12]:
array([9, 2, -7, 5], dtype=int64)
In [13]:
obj.index
Out[13]:
RangeIndex(start=0, stop=4, step=1)

Series

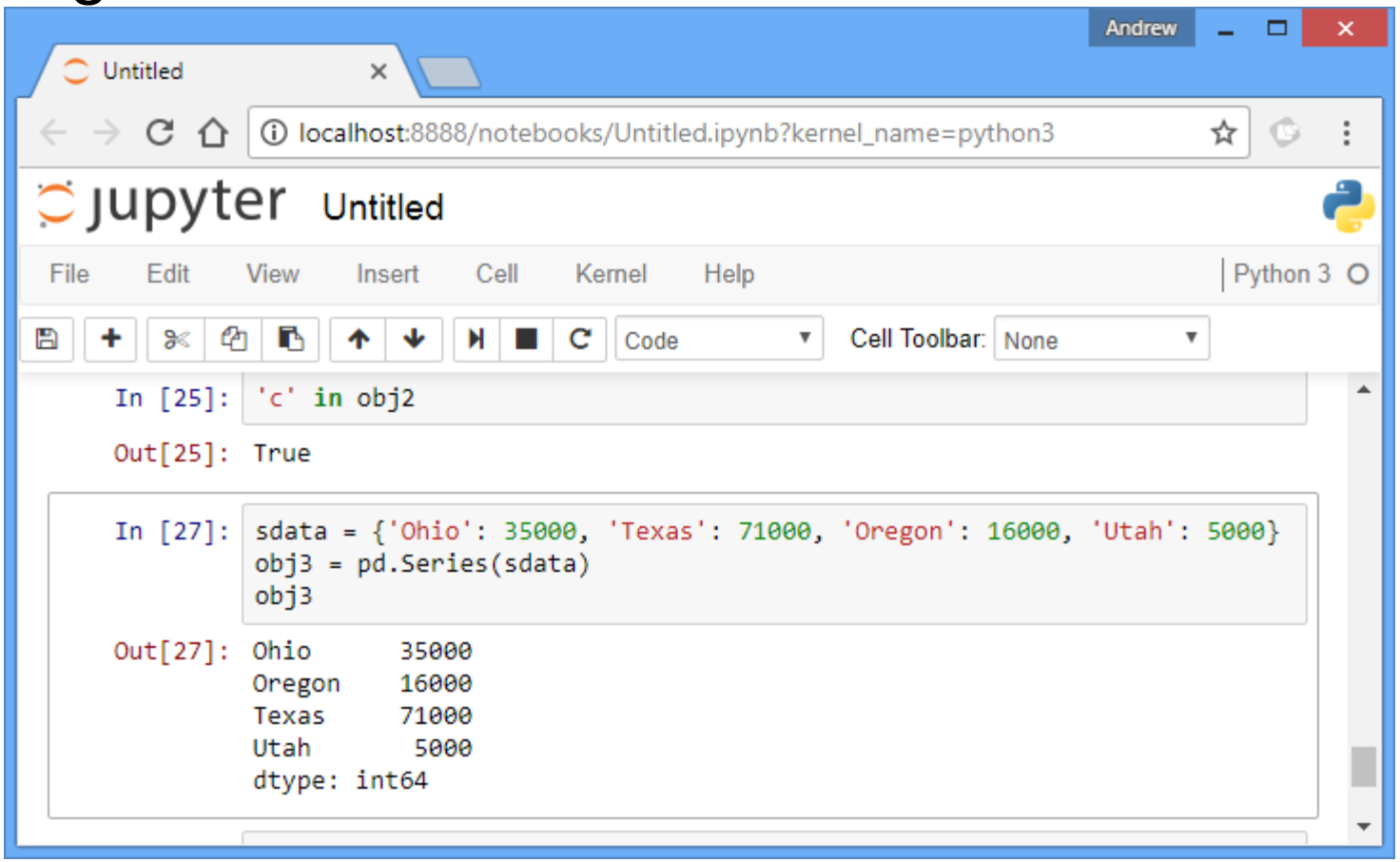
- You can create a Series with an index identifying each data point with a label



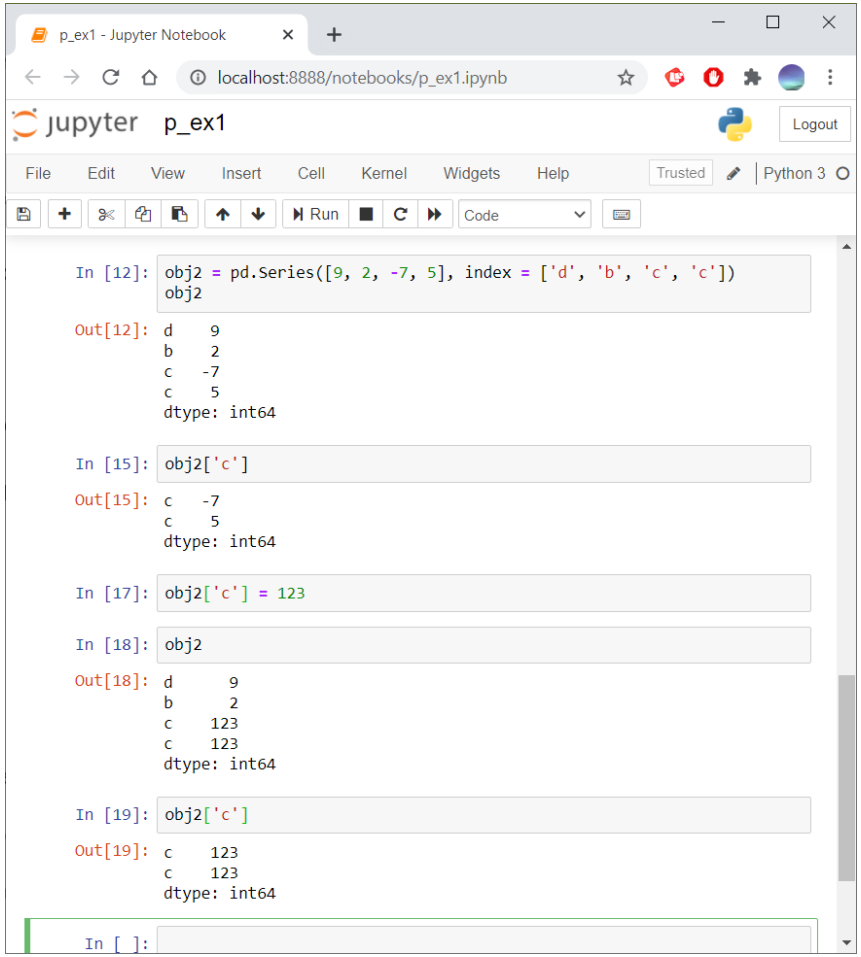

Code
Cell Toolbar:
None
In [17]: obj2['a']
Out[17]: -7
In [18]: obj2['c'] = 10
In [19]: obj2[['c', 'a']]
Out[19]: c 10
a -7
dtype: int64
In [20]: obj2[obj2 > 0]
Out[20]: d 9
b 2
c 10
dtype: int64
In [21]: obj2*5
Out[21]: d 45
b 10
a -35
c 50
dtype: int64

Series

- Another way to think about a Series is as a dict, as it is a mapping of index values to data values. It can be used in many contexts where you might use a dict



The index/label values in a Series do not have to be unique



DataFrame

- Dataframe (DF) is the most important and useful data structure in pandas
- It is used for almost all kind of data representation and manipulation in pandas
- A DF is a table containing rows (representing objects/observations) and columns (representing features/variables/attributes)
- Unlike numpy arrays, it can contain heterogeneous data
- DataFrames are useful for representing raw datasets and processed datasets in ML and data science

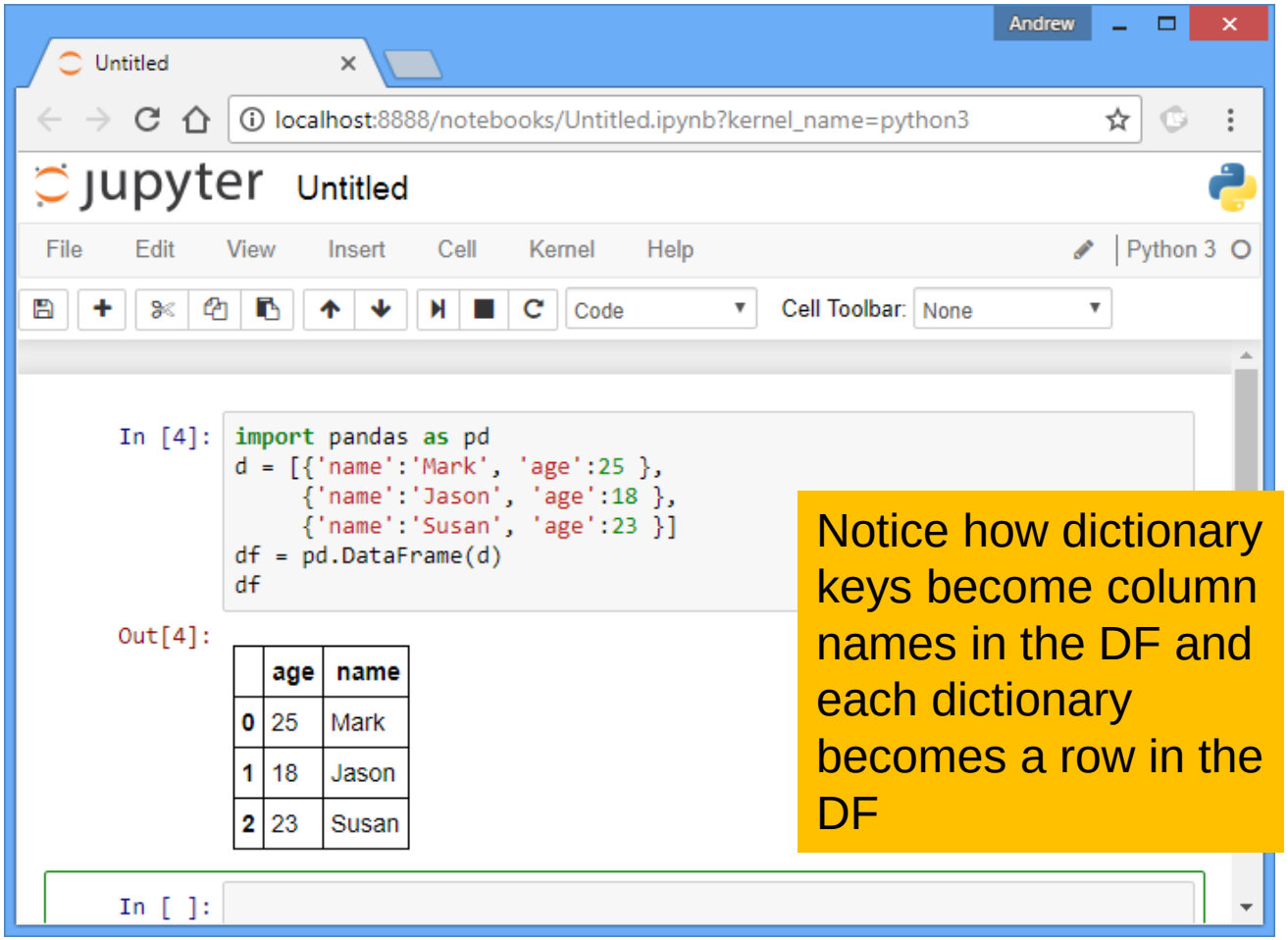
Relationship between DataFrames and Series

- We can think of a DataFrame as a sequence of aligned Series objects
  - aligned means that they share the same index

Constructing Data Frame Objects

- A DataFrame object can be constructed in different ways, including:
  - from a list of dicts
  - from a dict of dicts
  - from a list of Series objects
  - from a dict of Series objects
  - from a 2-D NumPy array
  - from a 2-D Python list

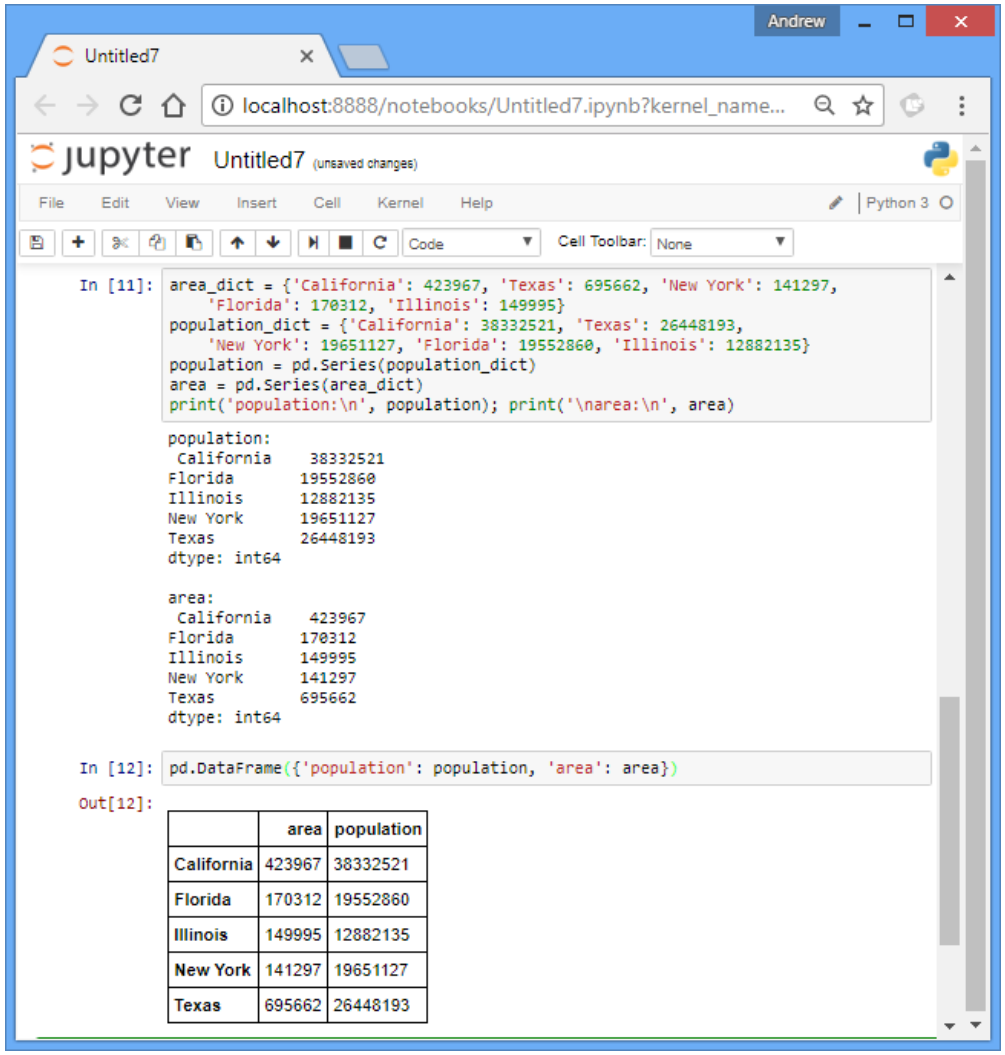
List of Dictionaries to a Dataframe



	age	name
0	25	Mark
1	18	Jason
2	23	Susan

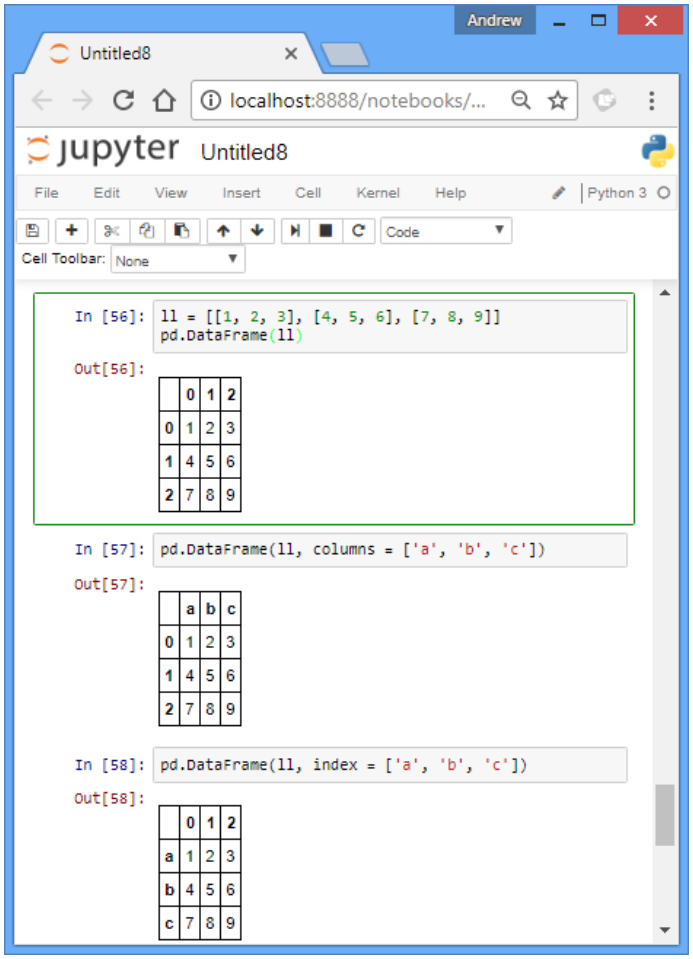
Notice how dictionary keys become column names in the DF and each dictionary becomes a row in the DF

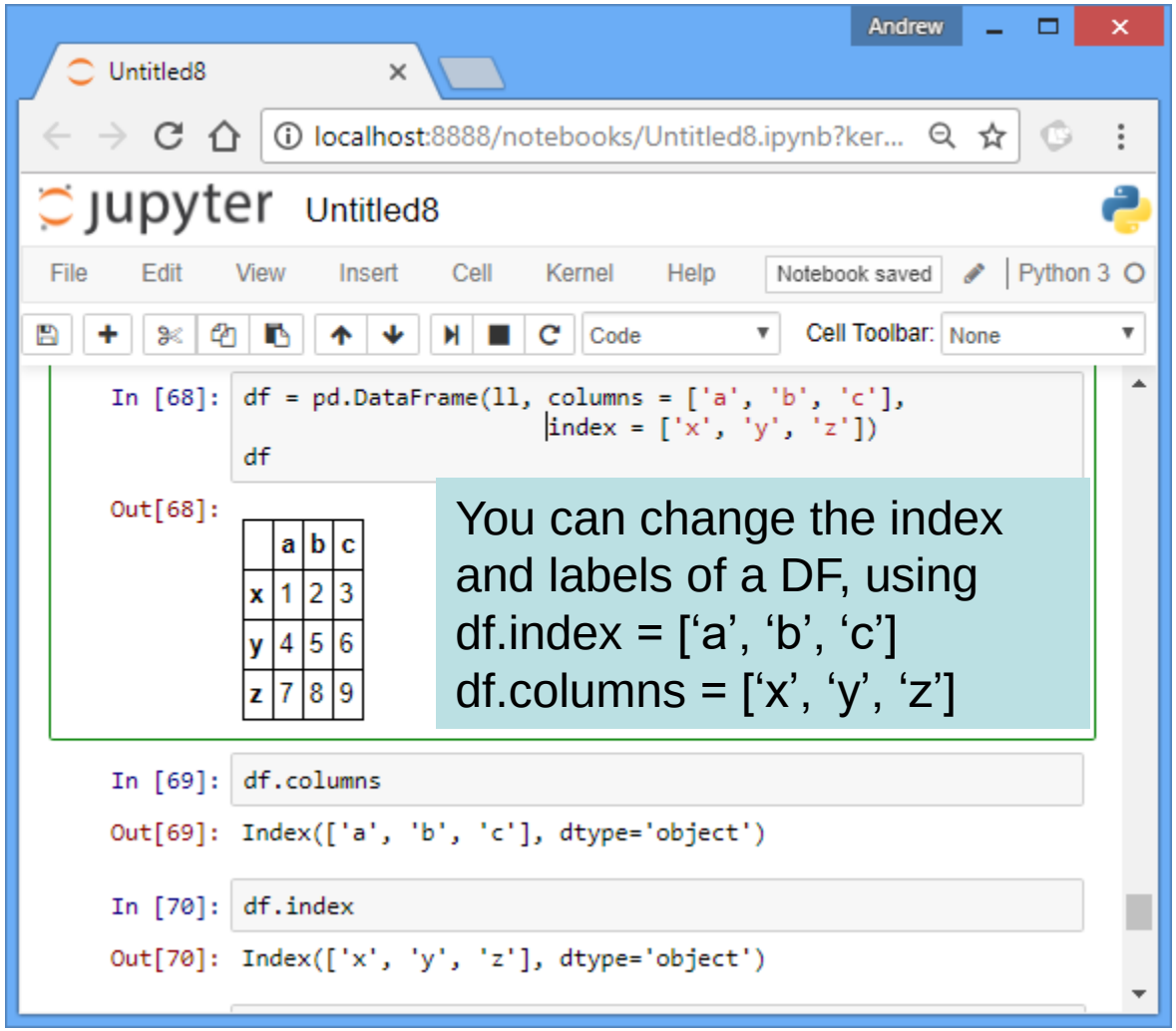
A Dictionary of Series Objects to a Dataframe



	area	population
California	423967	38332521
Florida	170312	19552860
Illinois	149995	12882135
New York	141297	19651127
Texas	695662	26448193

A 2-D List to a DataFrame





You can change the index and labels of a DF, using
df.index = [‘a’, ‘b’, ‘c’]
df.columns = [‘x’, ‘y’, ‘z’]

Data Retrieval

- Pandas provides different ways to retrieve data into a DataFrame. For example, we can read flat files, csv files, database tables, …etc.

CSV Files to a Dataframe

- We can read a CSV, or any delimited file, using pandas and get the result as a DF
- This is done using panda's read_csv() function
- Usually, the only parameter needed is the name of the input file but sometimes you may need to add more parameters

Examples

- We are using the files simplemaps-worldcities-basic.csv and nyc_taxi data

Examples

Code | Cell Toolbar: None

```
In [30]: import pandas as pd
city_data = pd.read_csv('simplemaps-worldcities-basic.csv')
city_data.head(3)
```

Out[30]:

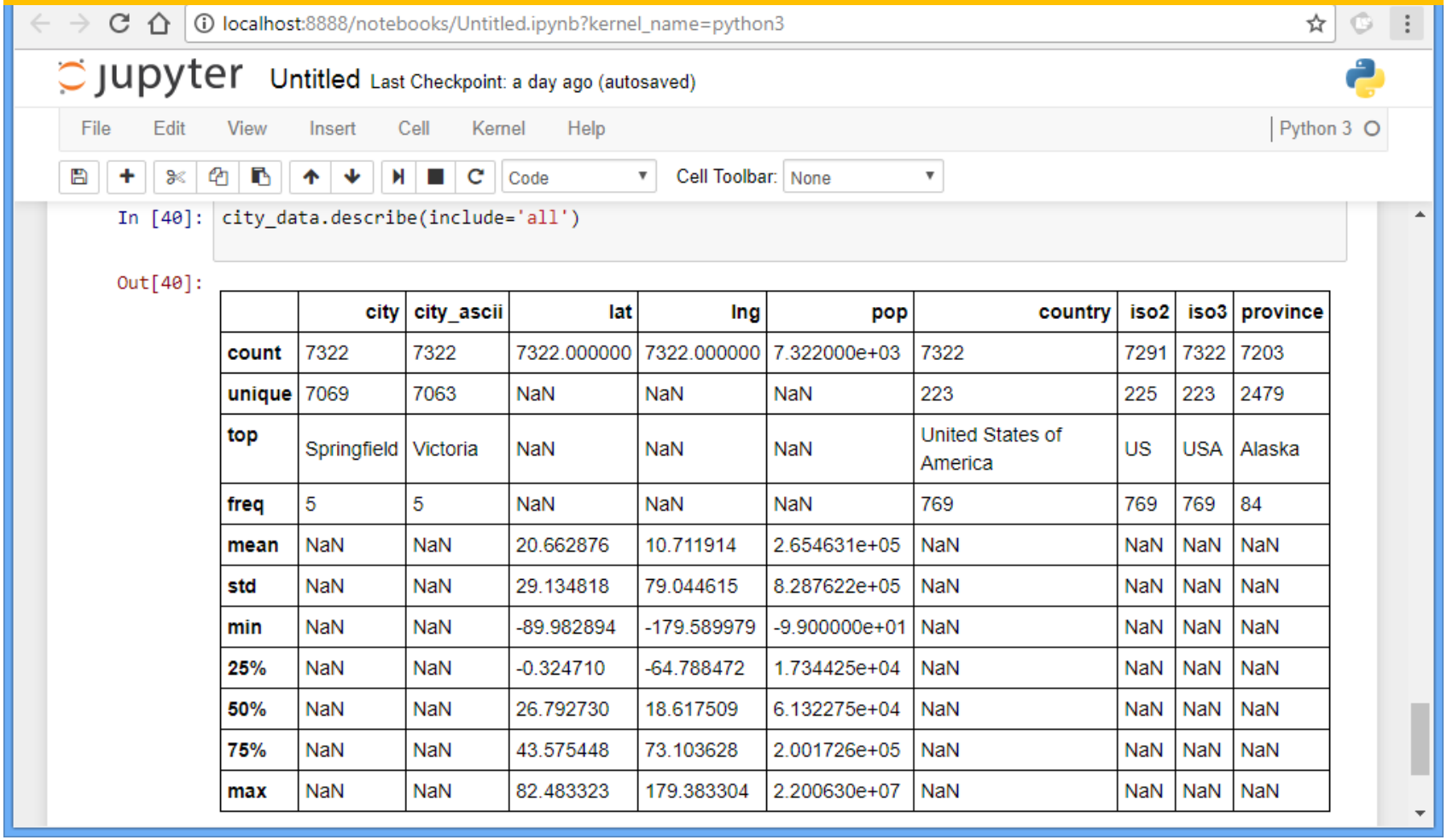
	city	city_ascii	lat	lng	pop	country	iso2
0	Qal eh-ye Now	Qal eh-ye	34.983000	63.133300	2997.0	Afghanistan	AF
1	Chaghcharan	Chaghcharan	34.516701	65.250001	15000.0	Afghanistan	AF
2	Lashkar Gah	Lashkar Gah	31.582998	64.360000	201546.0	Afghanistan	AF

```
In [32]: city_data.tail()
```

Out[32]:

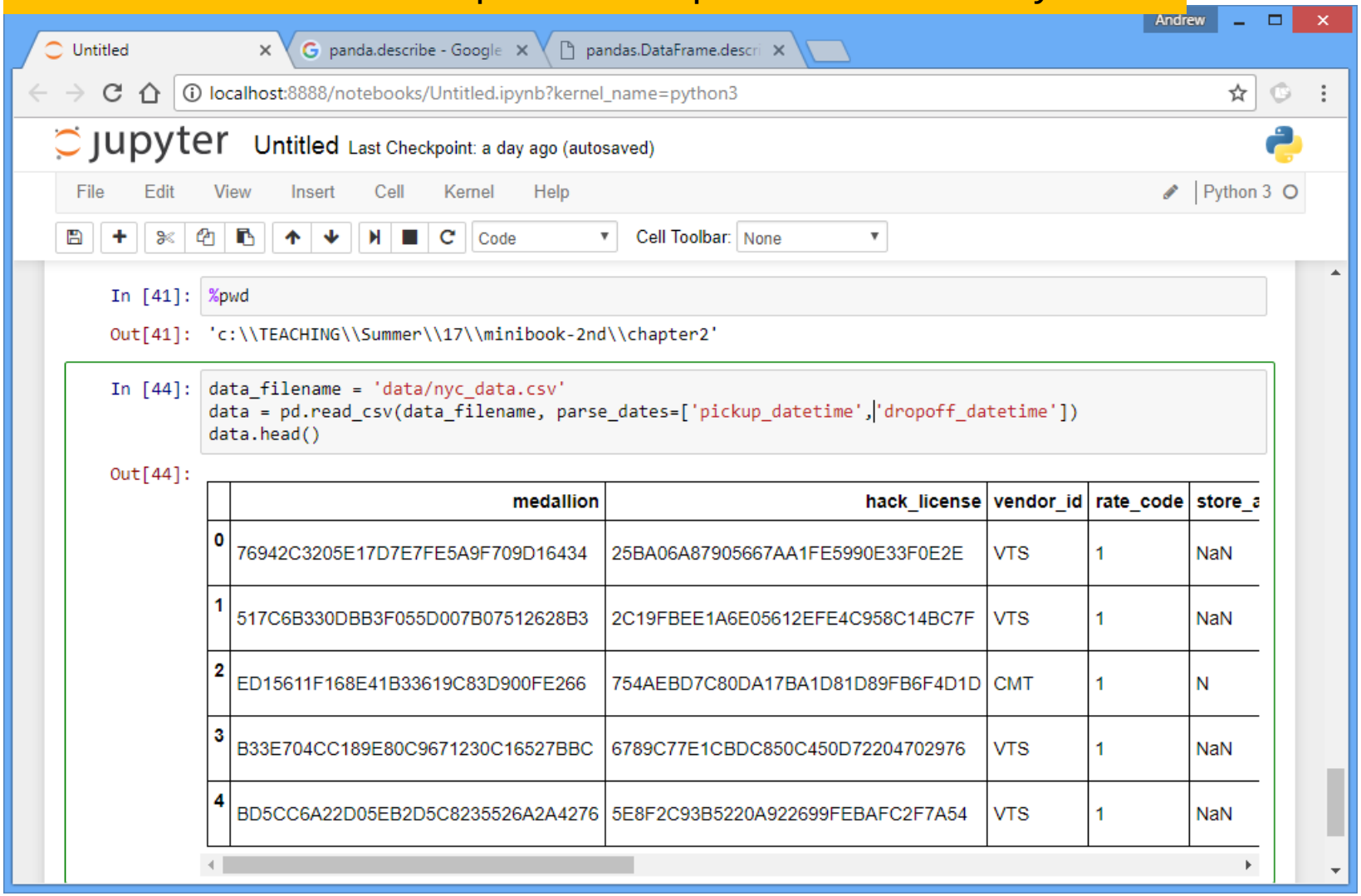
	city	city_ascii	lat	lng	pop	country	iso2
7317	Mutare	Mutare	-18.970019	32.650038	216785.0	Zimbabwe	ZW
7318	Kadoma	Kadoma	-18.330006	29.909947	56400.0	Zimbabwe	ZW
7319	Chitungwiza	Chitungwiza	-18.000001	31.100003	331071.0	Zimbabwe	ZW
7320	Harare	Harare	-17.817790	31.044709	1557406.5	Zimbabwe	ZW
7321	Bulawayo	Bulawayo	-20.169998	28.580002	697096.0	Zimbabwe	ZW

DF.describe() gives basic statistics of the columns
w/o param include='all', function shows only columns with numeric types

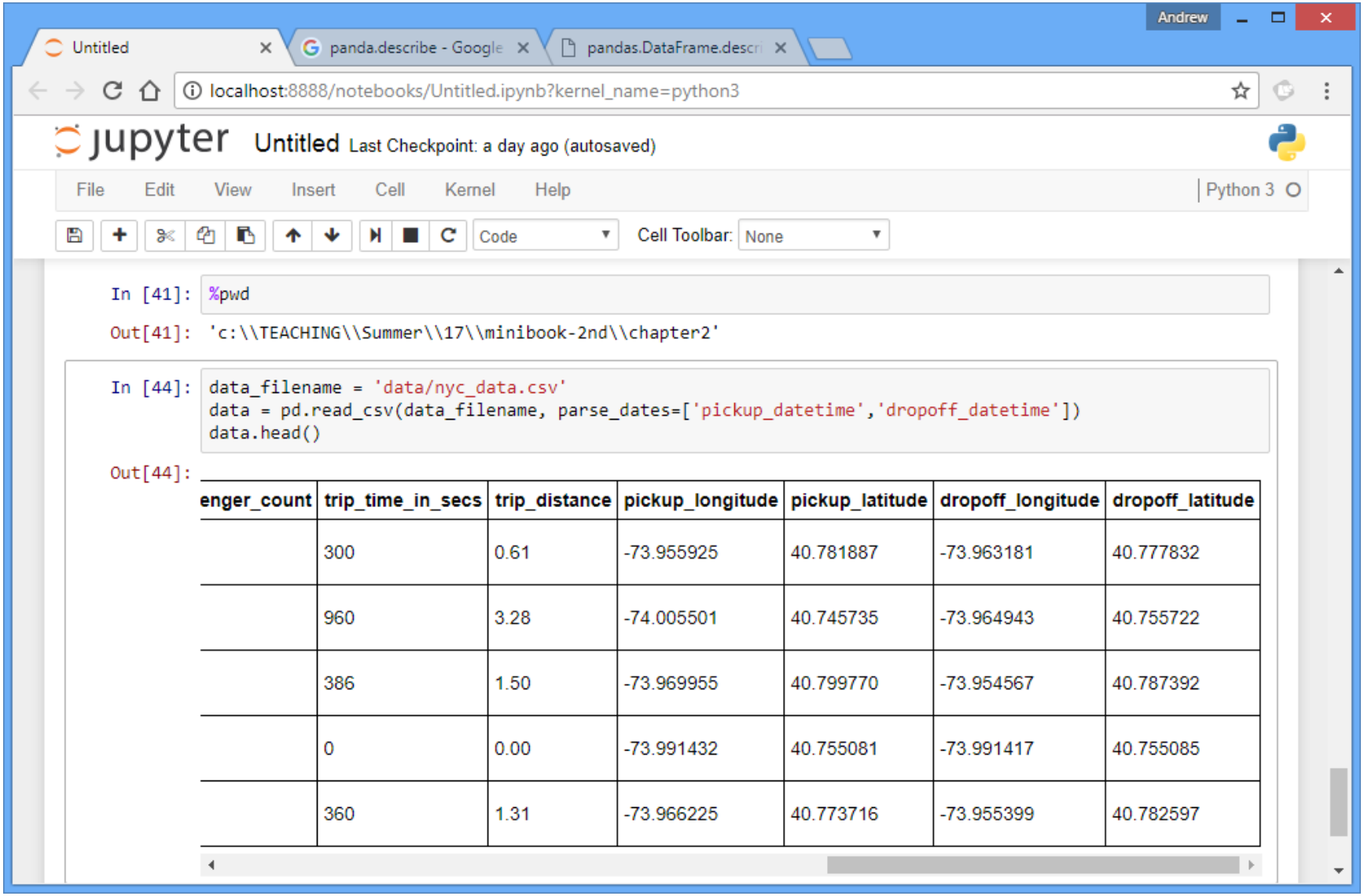


	city	city_ascii	lat	lng	pop	country	iso2	iso3	province
count	7322	7322	7322.000000	7322.000000	7.322000e+03	7322	7291	7322	7203
unique	7069	7063	NaN	NaN	NaN	223	225	223	2479
top	Springfield	Victoria	NaN	NaN	NaN	United States of America	US	USA	Alaska
freq	5	5	NaN	NaN	NaN	769	769	769	84
mean	NaN	NaN	20.662876	10.711914	2.654631e+05	NaN	NaN	NaN	NaN
std	NaN	NaN	29.134818	79.044615	8.287622e+05	NaN	NaN	NaN	NaN
min	NaN	NaN	-89.982894	-179.589979	-9.900000e+01	NaN	NaN	NaN	NaN
25%	NaN	NaN	-0.324710	-64.788472	1.734425e+04	NaN	NaN	NaN	NaN
50%	NaN	NaN	26.792730	18.617509	6.132275e+04	NaN	NaN	NaN	NaN
75%	NaN	NaN	43.575448	73.103628	2.001726e+05	NaN	NaN	NaN	NaN
max	NaN	NaN	82.483323	179.383304	2.200630e+07	NaN	NaN	NaN	NaN

Keyword argument **parse_dates** is used to specify the columns that contain dates so that pandas can parse them correctly



	medallion	hack_license	vendor_id	rate_code	store_a
0	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E	VTS	1	NaN
1	517C6B330DBB3F055D007B07512628B3	2C19FBEE1A6E05612EFE4C958C14BC7F	VTS	1	NaN
2	ED15611F168E41B33619C83D900FE266	754AEBD7C80DA17BA1D81D89FB6F4D1D	CMT	1	N
3	B33E704CC189E80C9671230C16527BBC	6789C77E1CBDC850C450D72204702976	VTS	1	NaN
4	BD5CC6A22D05EB2D5C8235526A2A4276	5E8F2C93B5220A922699FEBAFC2F7A54	VTS	1	NaN



enger_count	trip_time_in_secs	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
	300	0.61	-73.955925	40.781887	-73.963181	40.777832
	960	3.28	-74.005501	40.745735	-73.964943	40.755722
	386	1.50	-73.969955	40.799770	-73.954567	40.787392
	0	0.00	-73.991432	40.755081	-73.991417	40.755085
	360	1.31	-73.966225	40.773716	-73.955399	40.782597

Andrew

Untitled

localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3

jupyter Untitled Last Checkpoint: a day ago (unsaved changes)

File Edit View Insert Cell Kernel Help Python 3

Code Cell Toolbar: None

In [45]:
```
data.columns
```

Out[45]:
```
Index(['medallion', 'hack_license', 'vendor_id', 'rate_code',
       'store_and_fwd_flag', 'pickup_datetime', 'dropoff_datetime',
       'passenger_count', 'trip_time_in_secs', 'trip_distance',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude'],
      dtype='object')
```

In [50]:
```
p_lat = data.pickup_latitude
p_lat.head()
```

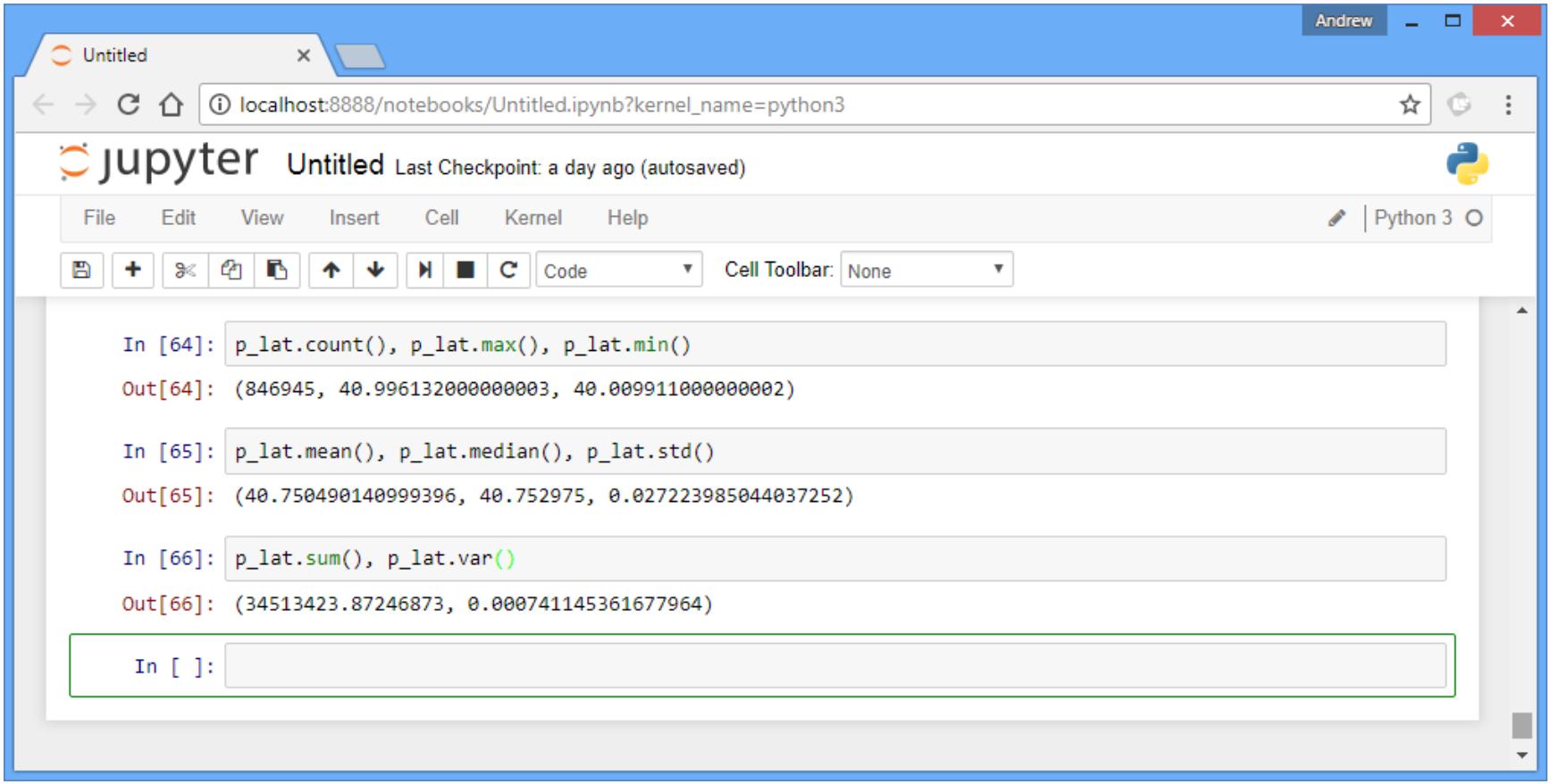
Out[50]:
```
0    40.781887
1    40.745735
2    40.799770
3    40.755081
4    40.773716
Name: pickup_latitude, dtype: float64
```

In [52]:
```
p_long = data['pickup_longitude']
p_long.tail()
```

Out[52]:
```
846940   -73.992058
846941   -73.994949
846942   -73.993492
846943   -73.978477
846944   -73.987206
Name: pickup_longitude, dtype: float64
```

Descriptive Statistics with Pandas

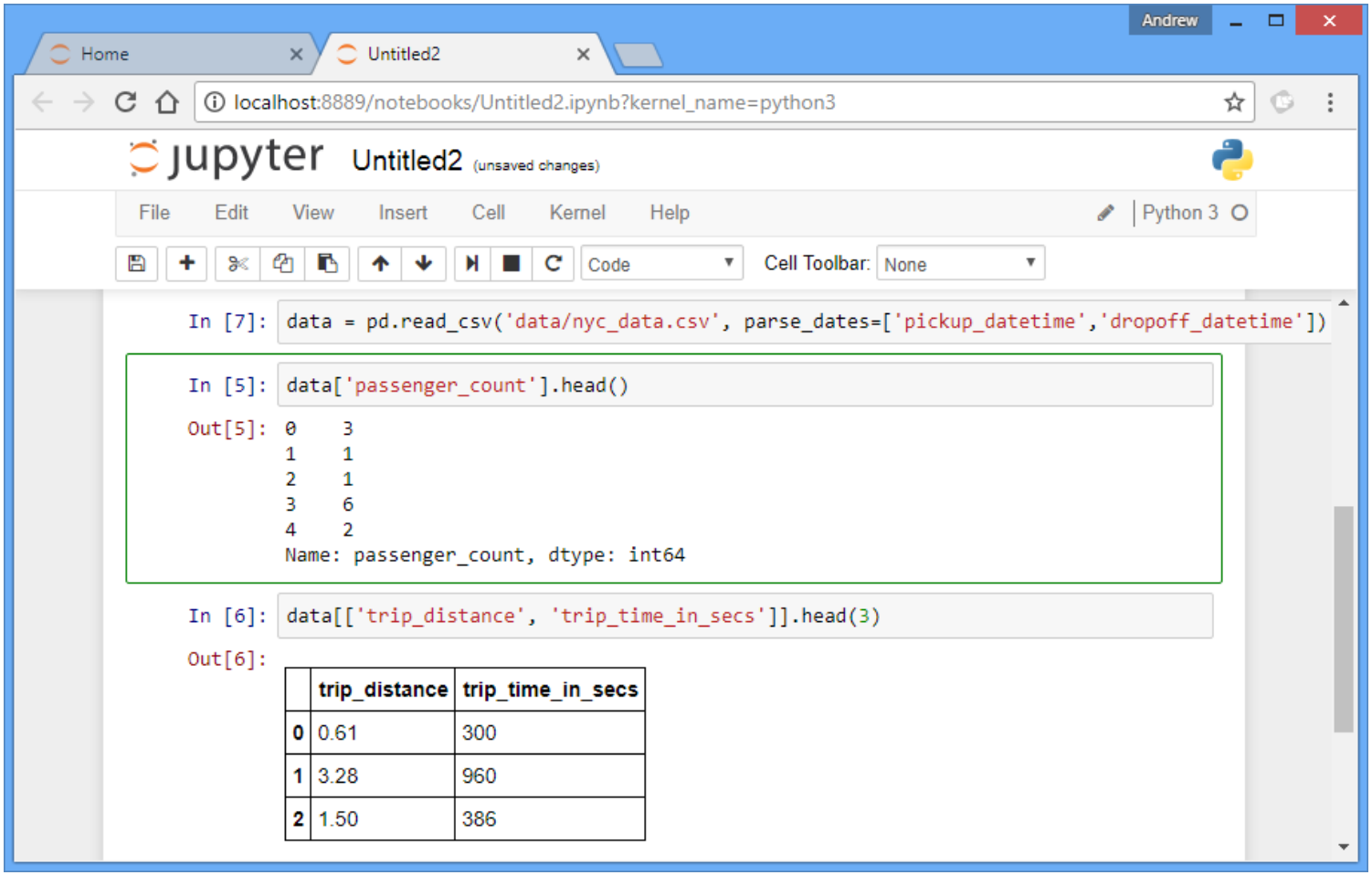
- The following functions do as their names indicate



Selecting Data

- Pandas allows us to select rows, columns, and combinations of rows and columns, … etc.

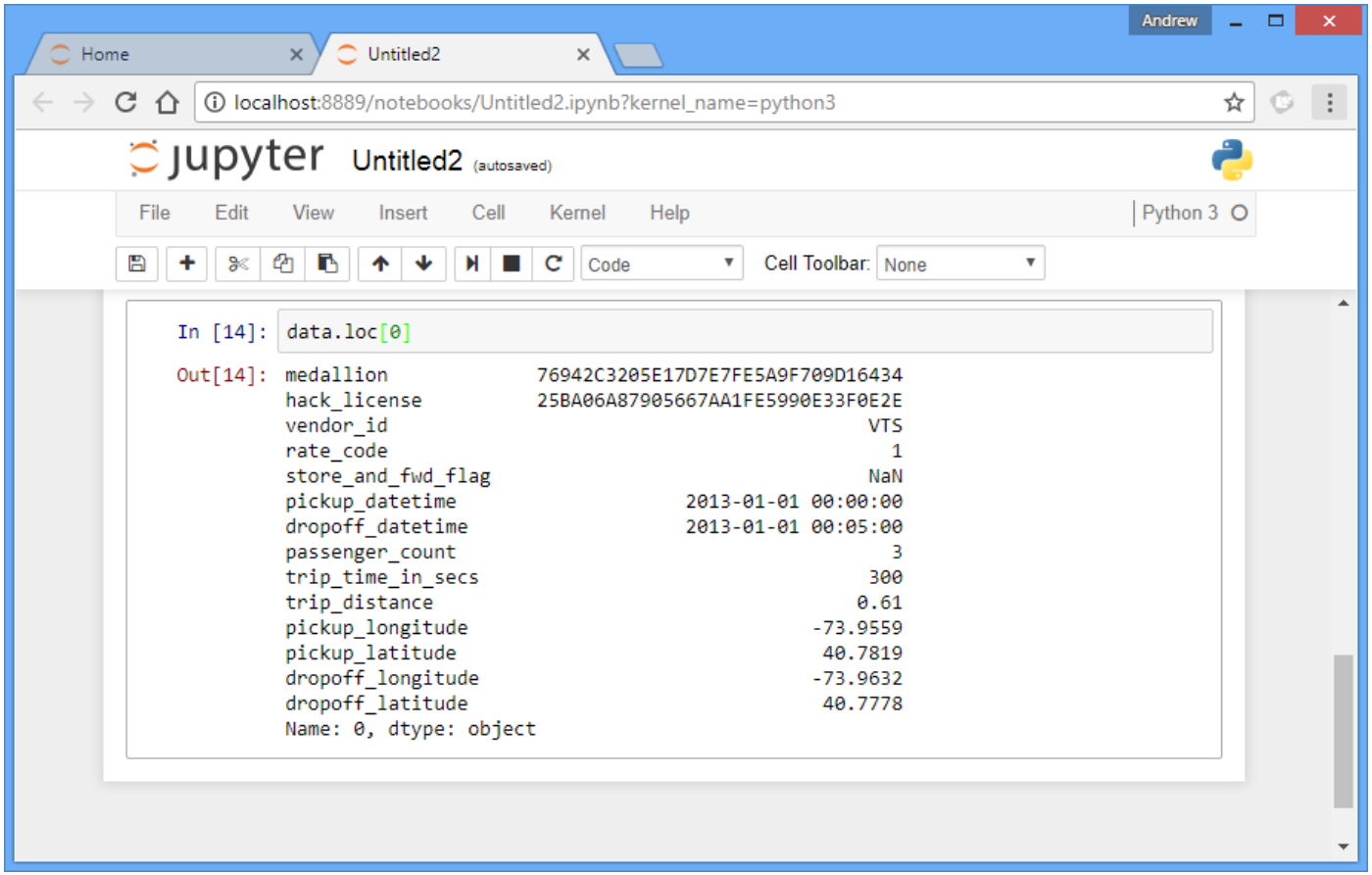
Selecting Columns



	trip_distance	trip_time_in_secs
0	0.61	300
1	3.28	960
2	1.50	386

Selecting Rows

- Rows can be retrieved by position or name
  - Use loc attribute to select row(s) from their labels
  - Use iloc attribute to select row(s) from their positions
- The following example show how to select the first row:



Remark: The **row selection** syntax data[:2] is provided as a convenience.
Passing a single element or a list to the [] operator selects columns

In [27]:
```
data[0:2]
```

Out[27]:

	medallion	hack_license	vendo
0	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E	VTS
1	517C6B330DBB3F055D007B07512628B3	2C19FBEE1A6E05612EFE4C958C14BC7F	VTS

In [29]:
```
data['passenger_count'].head(3)
```

Out[29]:
```
0    3
1    1
2    1
Name: passenger_count, dtype: int64
```

In [30]:
```
data.iloc[0:2]
```

Out[30]:

	medallion	hack_license	vendo
0	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E	VTS
1	517C6B330DBB3F055D007B07512628B3	2C19FBEE1A6E05612EFE4C958C14BC7F	VTS

Using DF.loc[rows_labels_names] or DF.iloc[rows_indexes] is the proper way to select rows.
The slice notation, DF[:], is provided for convenience.

```
In [31]: data.loc[0:2]
```

Out[31]:

	medallion	hack_license	vend
0	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E	VTS
1	517C6B330DBB3F055D007B07512628B3	2C19FBEE1A6E05612EFE4C958C14BC7F	VTS
2	ED15611F168E41B33619C83D900FE266	754AEBD7C80DA17BA1D81D89FB6F4D1D	CMT

```
In [39]: data[['trip_distance', 'trip_time_in_secs']].head(3)
```

Out[39]:

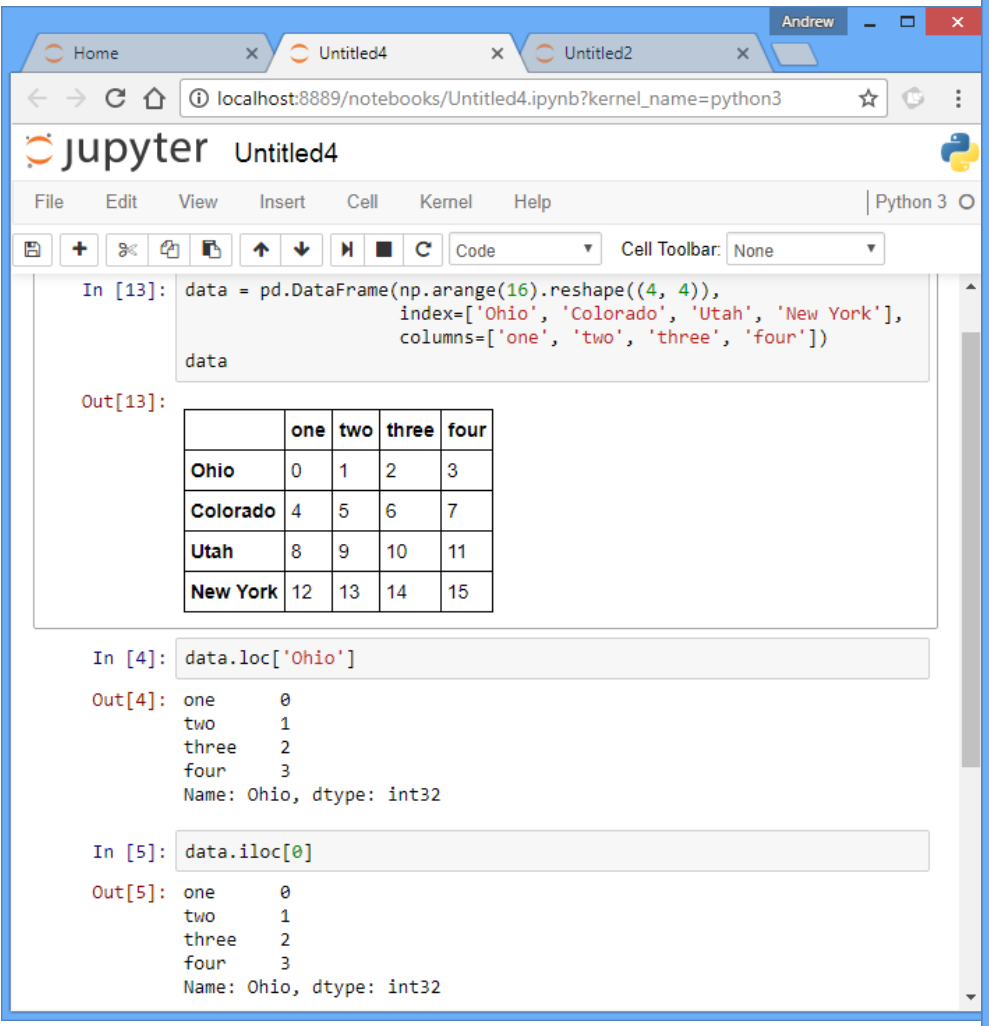
	trip_distance	trip_time_in_secs
0	0.61	300
1	3.28	960
2	1.50	386

Here, you needed to give the column labels, because writing DF[] is the notation to select columns.

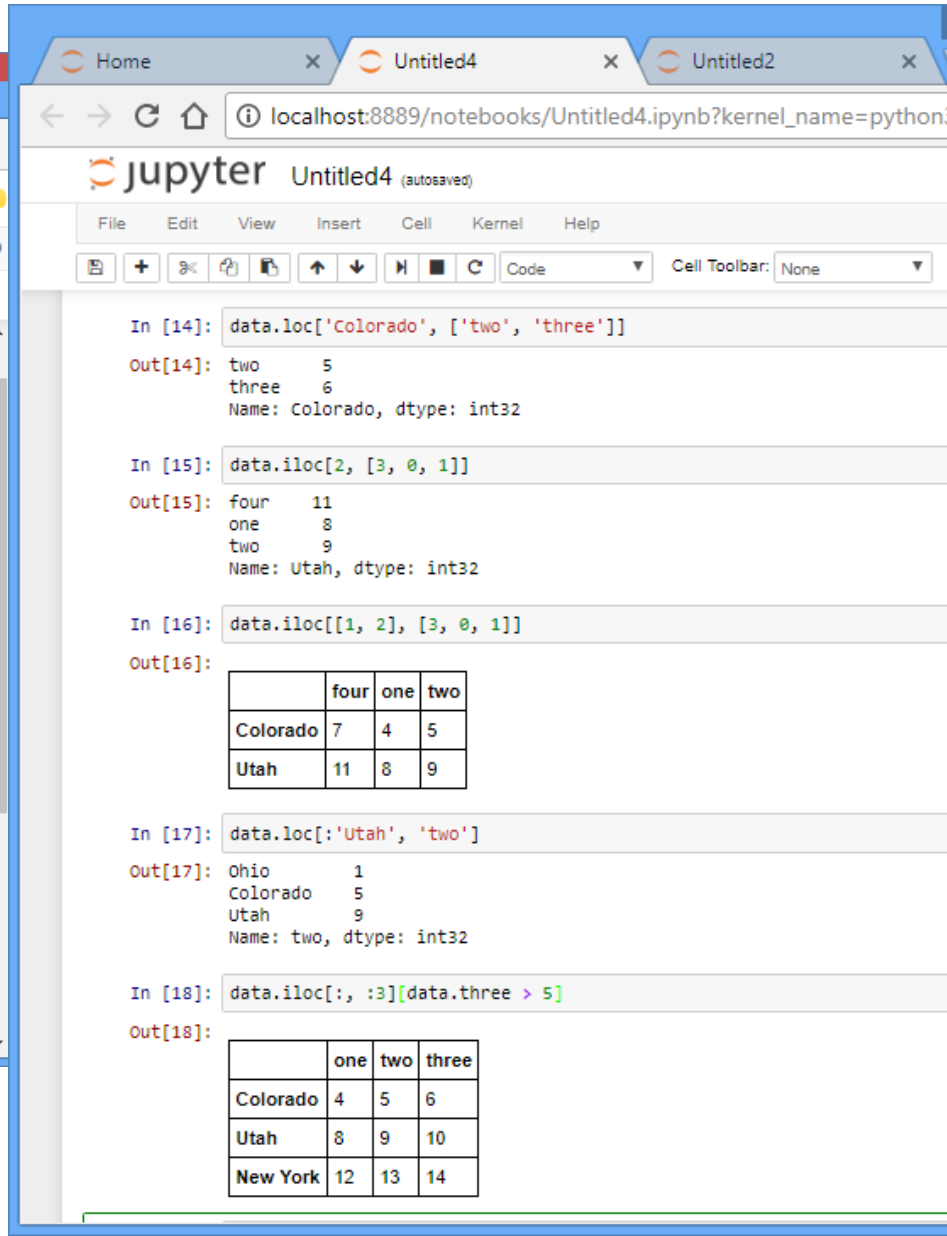
```
In [40]: data[[1,2]].head(3)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
```

Aside: More about loc and iloc



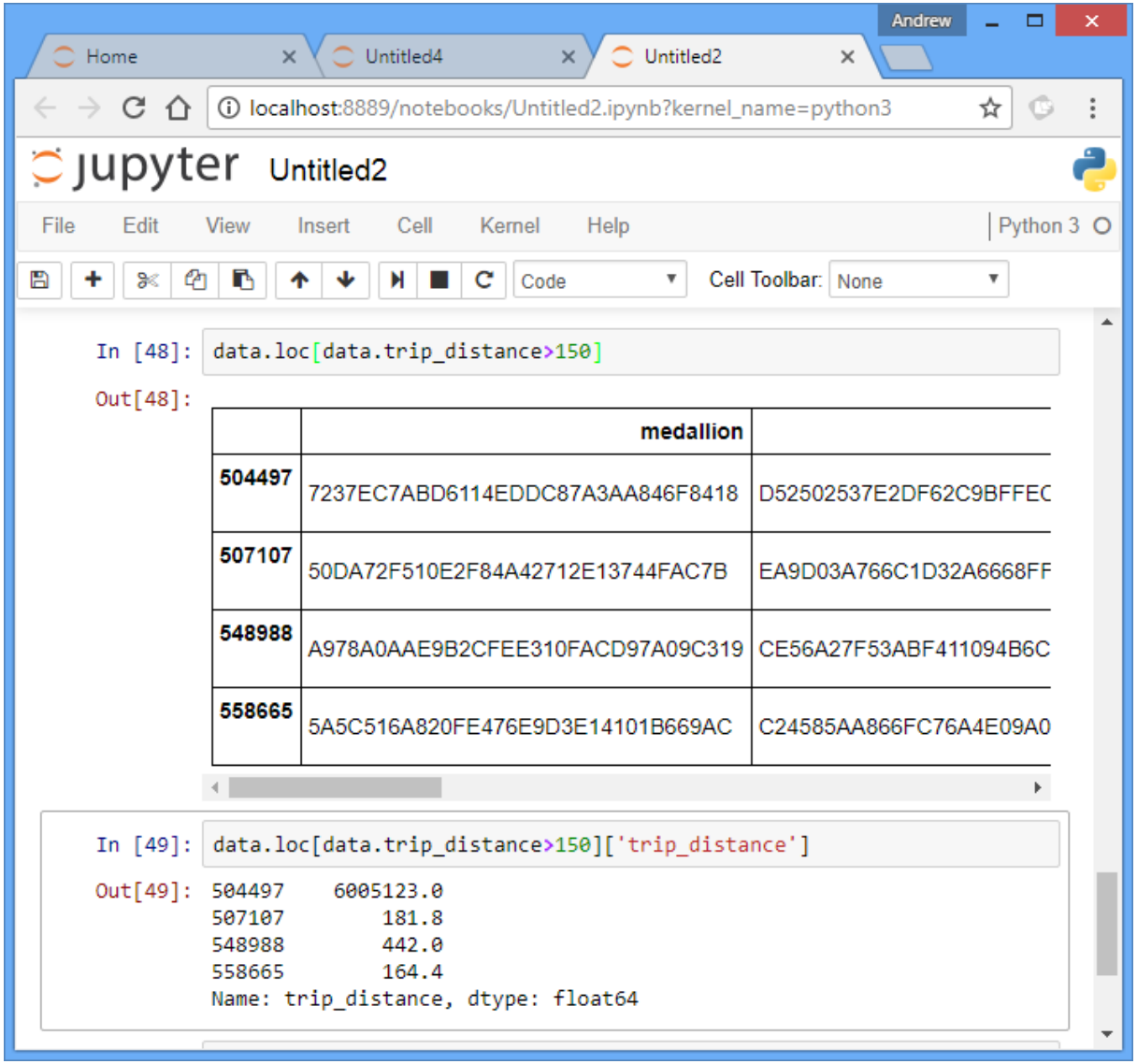
	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15



	four	one	two
Colorado	7	4	5
Utah	11	8	9

	one	two	three
Colorado	4	5	6
Utah	8	9	10
New York	12	13	14

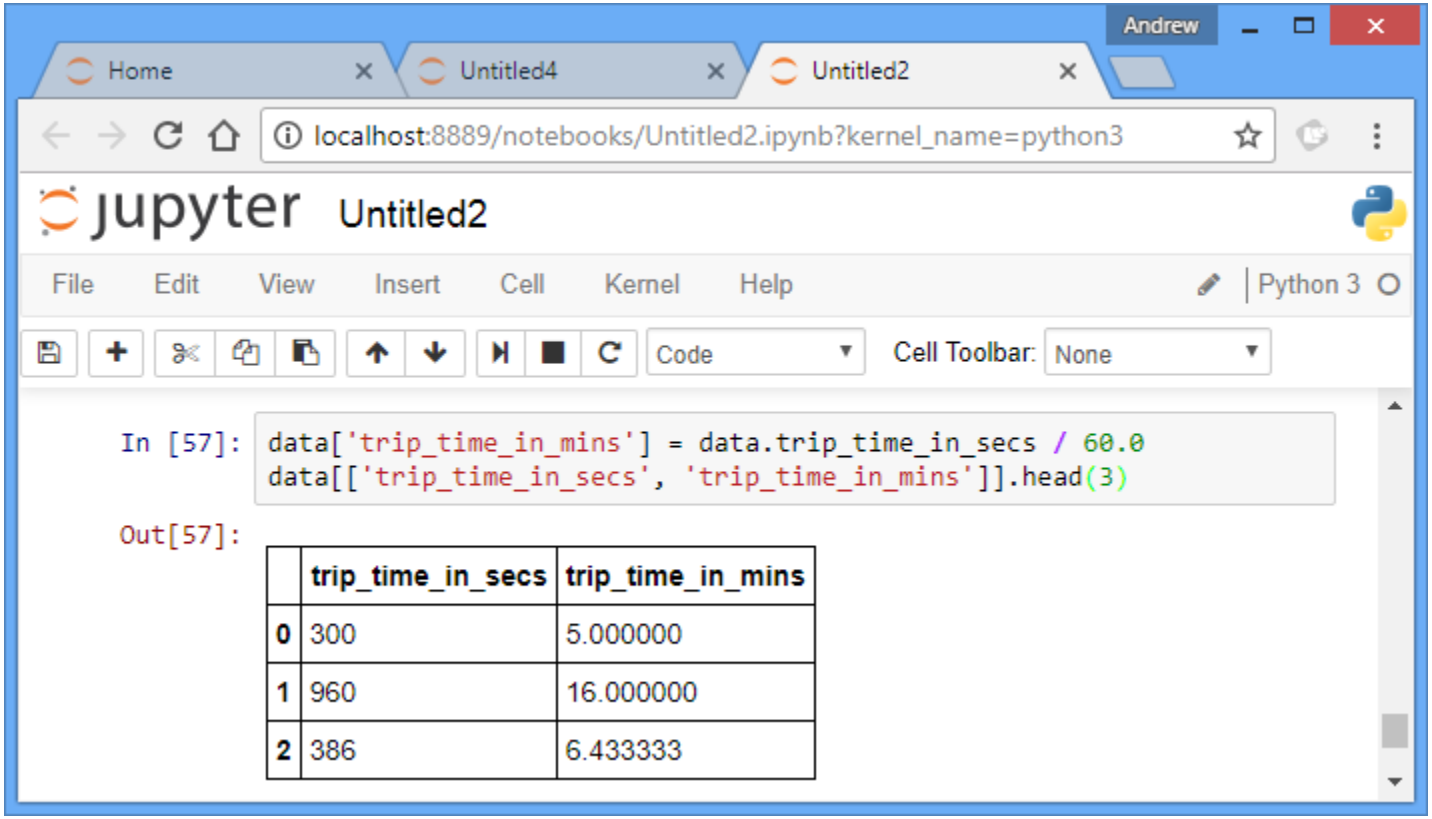
Filtering with Boolean Indexing



	medallion	
504497	7237EC7ABD6114EDDC87A3AA846F8418	D52502537E2DF62C9BFFEC
507107	50DA72F510E2F84A42712E13744FAC7B	EA9D03A766C1D32A6668FF
548988	A978A0AAE9B2CFEE310FACD97A09C319	CE56A27F53ABF411094B6C
558665	5A5C516A820FE476E9D3E14101B669AC	C24585AA866FC76A4E09A0

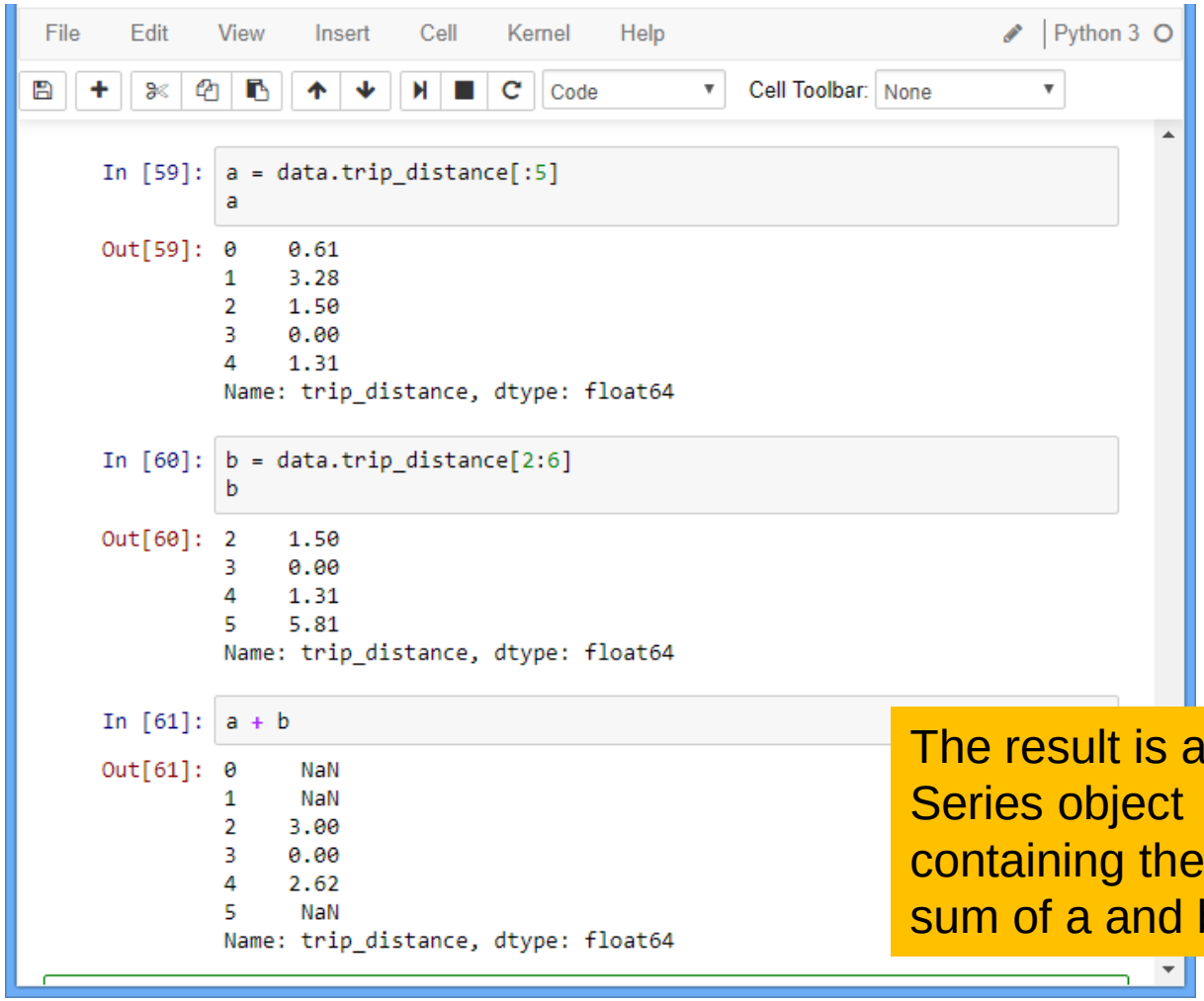
Operating on Data

- Vector arithmetic can be used with DataFrames and Series objects
- Example: adding a new column to the DF with the trip duration in minutes



	trip_time_in_secs	trip_time_in_mins
0	300	5.000000
1	960	16.000000
2	386	6.433333

More on Vector Operations



The result is a new Series object containing the aligned sum of a and b

More on Vector Operations

In [135]:
```
a = np.arange(12).reshape(3, 4)
df = pd.DataFrame(a, columns=list('ABCD'))
df
```

Out[135]:

	A	B	C	D
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

In [138]:
```
b = df.iloc[0]
b
```

Out[138]:
```
A    0
B    1
C    2
D    3
Name: 0, dtype: int32
```

In [139]:
```
df - b
```

Out[139]:

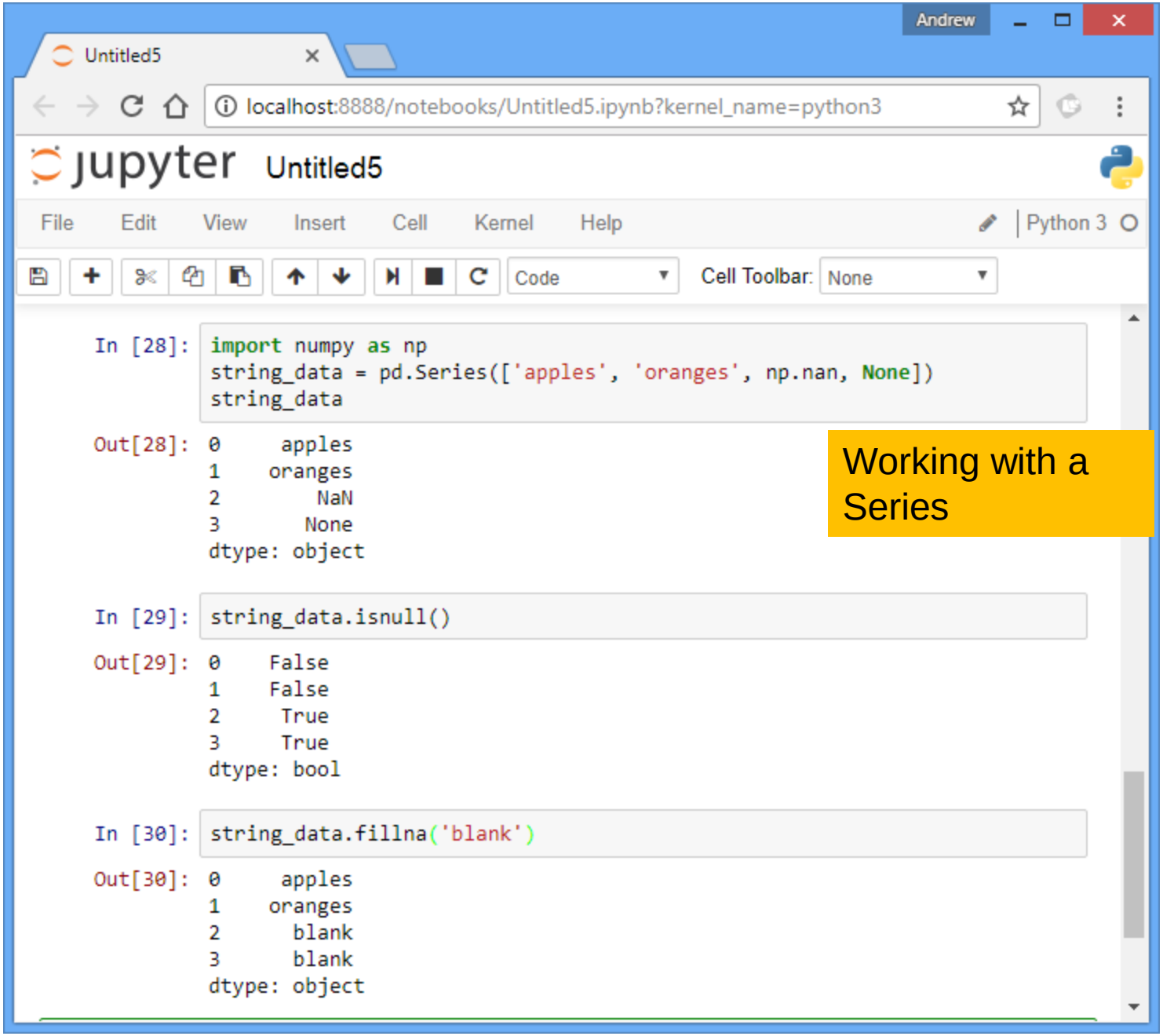
	A	B	C	D
0	0	0	0	0
1	4	4	4	4
2	8	8	8	8

```
[[0, 1, 2, 3],
 [4, 5, 6, 7],
 [8, 9, 10, 11]]
```

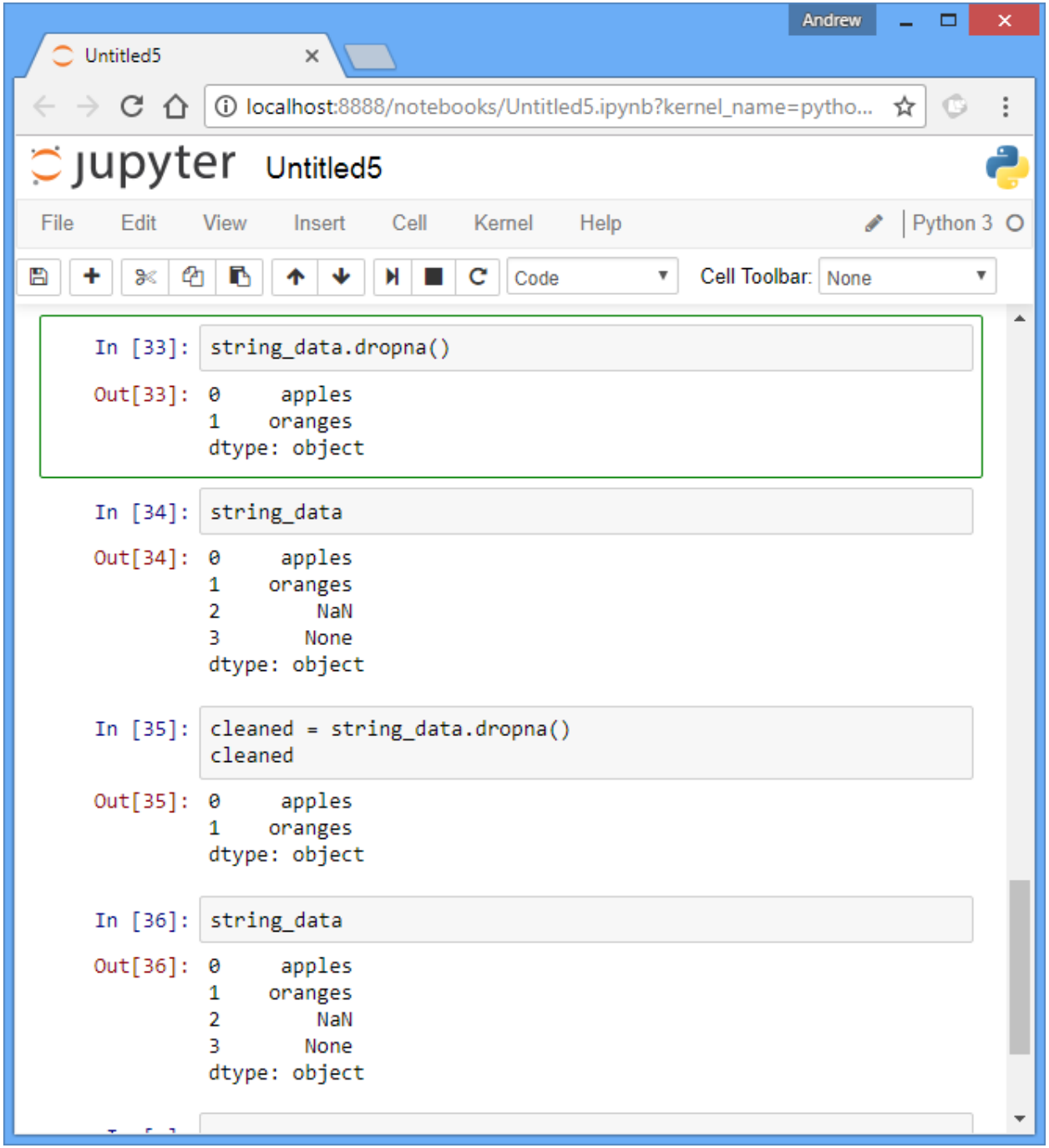
```
[[0, 1, 2, 3],
 [0, 1, 2, 3],
 [0, 1, 2, 3]]
```

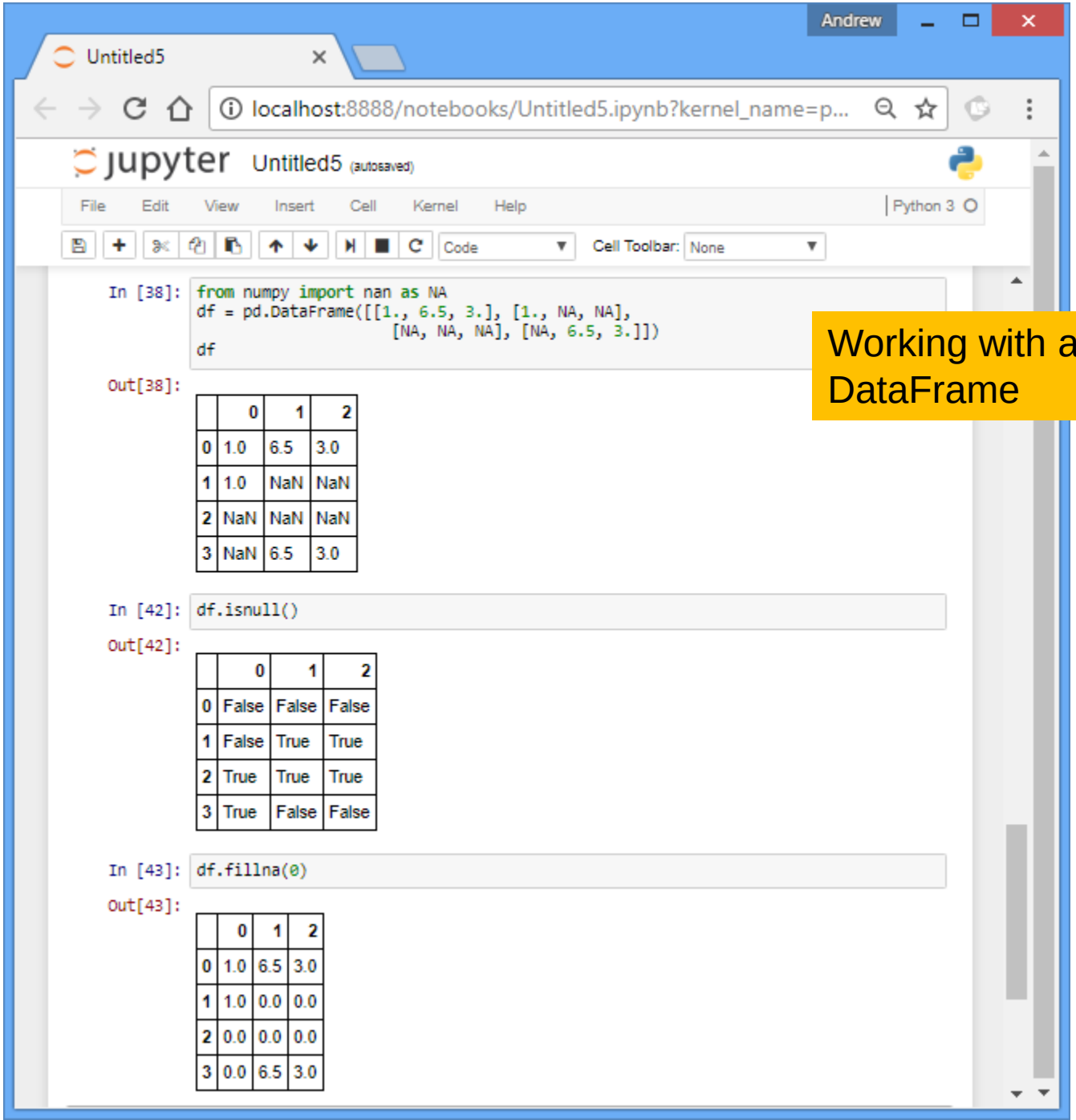
Handling Missing Data

- In pandas, missing data is represented by NaN (Not a Number) or None
- Pandas provides several Series and DataFrame methods to deal with missing data, including:
  - isnull() indicates whether values are null or not
  - notnull() the opposite of isnull()
  - dropna() removes missing data
  - fillna(some_default_value) replaces missing data with a default value



Working with a Series


Andrew
Untitled5
localhost:8888/notebooks/Untitled5.ipynb?kernel_name=pytho...
jupyter Untitled5
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None
In [33]: string_data.dropna()
Out[33]: 0 apples
1 oranges
dtype: object
In [34]: string_data
Out[34]: 0 apples
1 oranges
2 NaN
3 None
dtype: object
In [35]: cleaned = string_data.dropna()
cleaned
Out[35]: 0 apples
1 oranges
dtype: object
In [36]: string_data
Out[36]: 0 apples
1 oranges
2 NaN
3 None
dtype: object

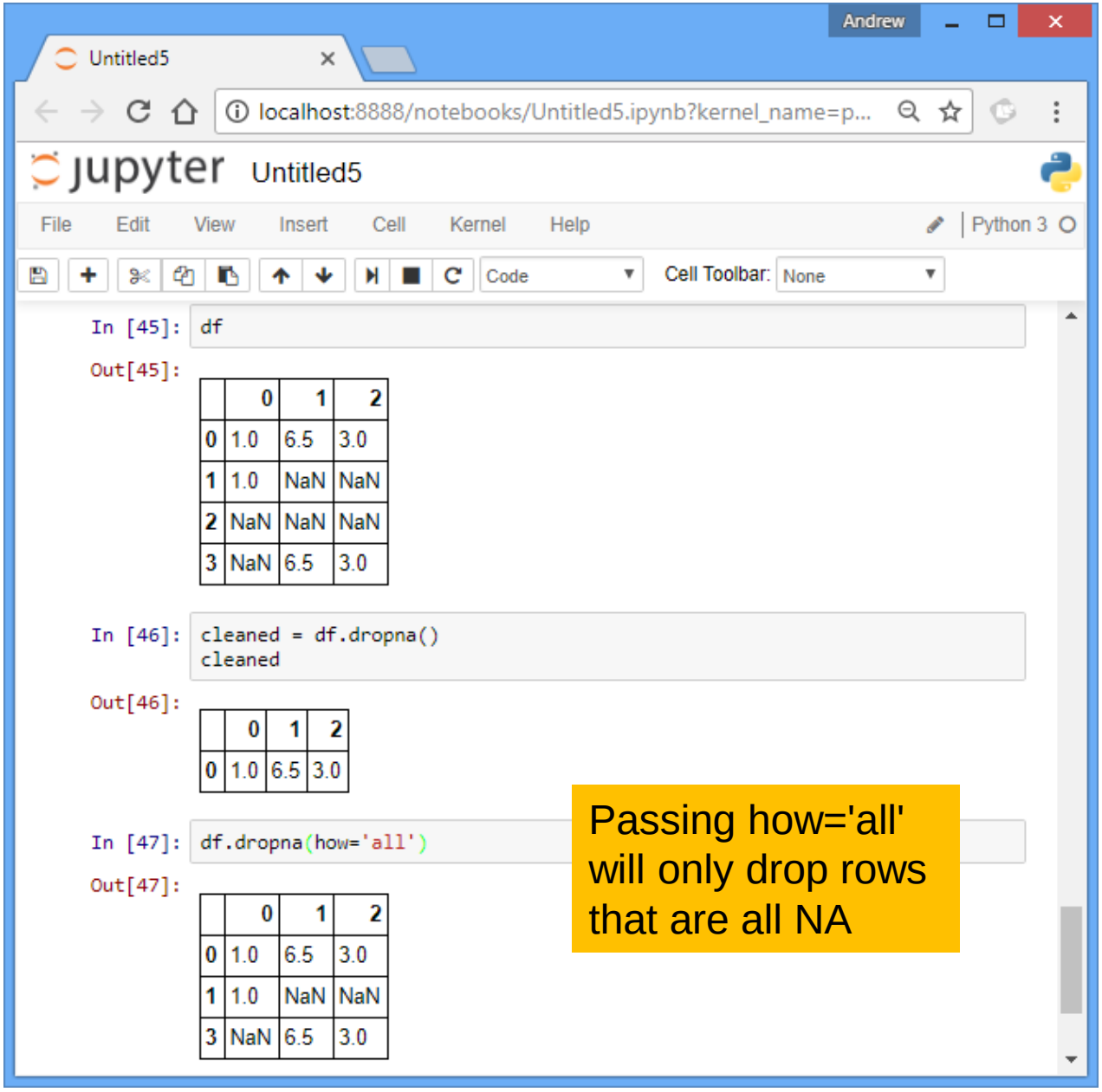


	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

	0	1	2
0	False	False	False
1	False	True	True
2	True	True	True
3	True	False	False

	0	1	2
0	1.0	6.5	3.0
1	1.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	6.5	3.0

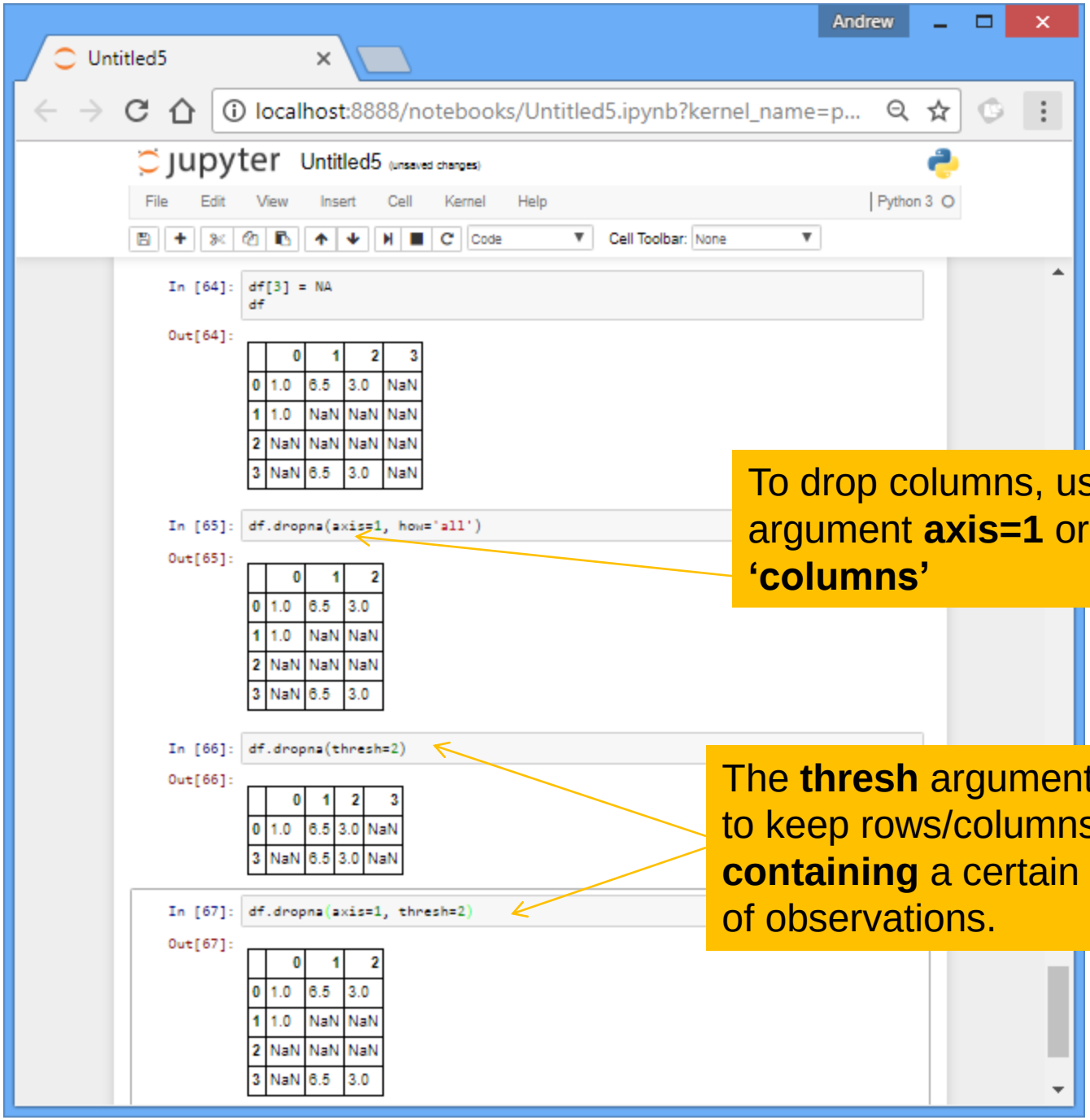
Working with a DataFrame



	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

	0	1	2
0	1.0	6.5	3.0

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0



	0	1	2	3
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

	0	1	2	3
0	1.0	6.5	3.0	NaN
3	NaN	6.5	3.0	NaN

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

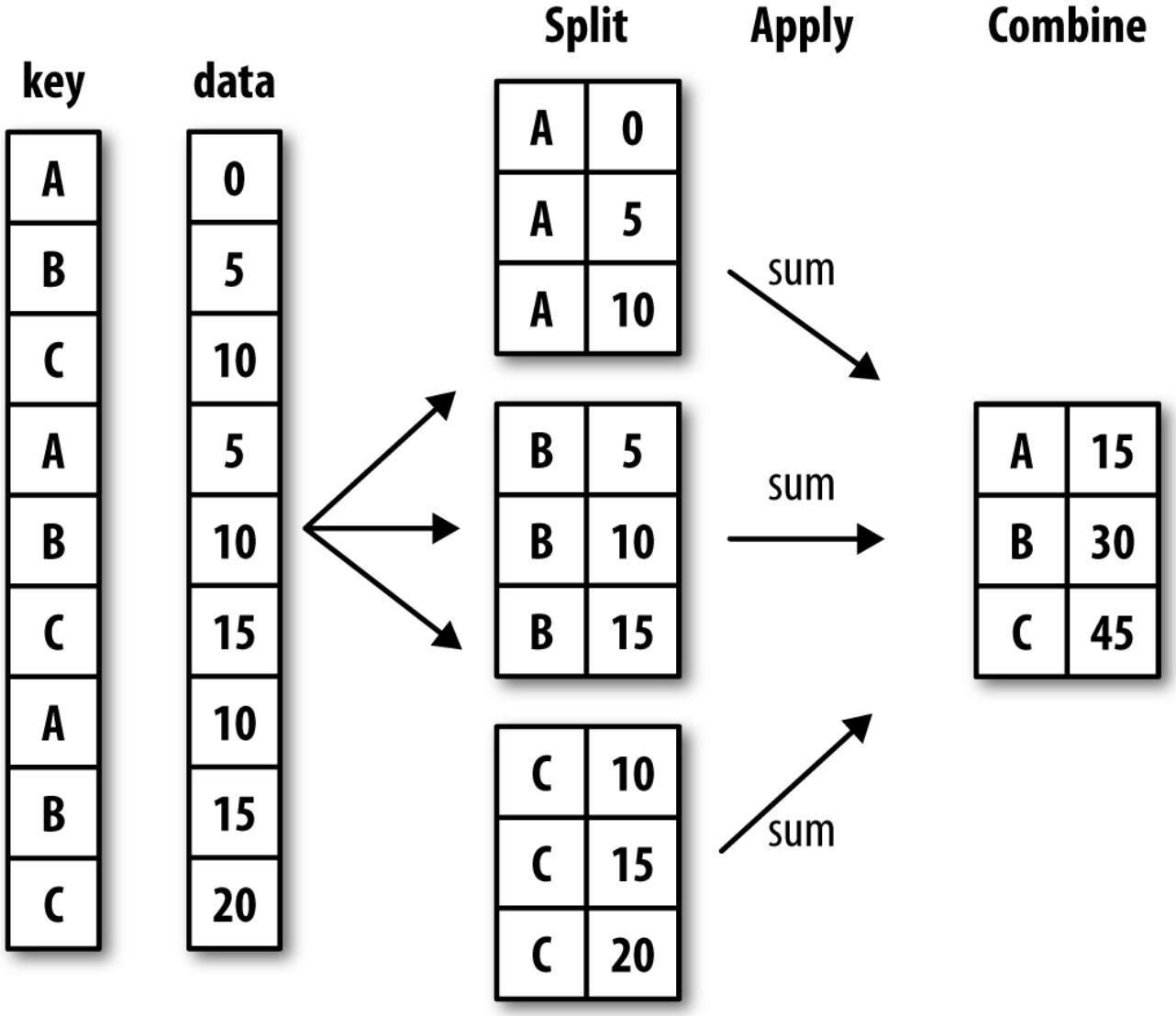
To drop columns, use the argument **axis=1** or **axis = 'columns'**

The **thresh** argument is used to keep rows/columns **containing** a certain number of observations.

The groupby Operation

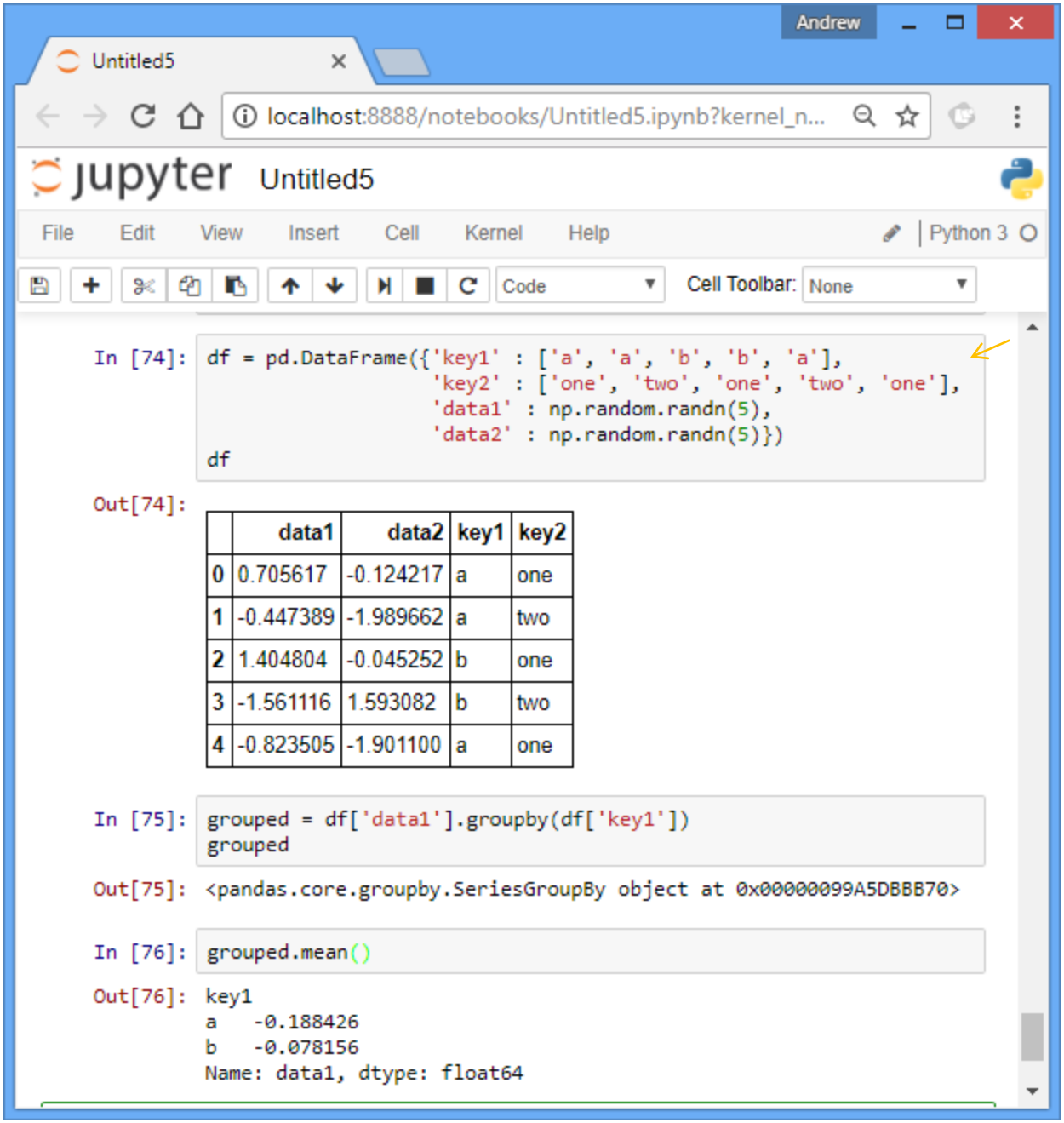
- Similar to the *group by* operation in SQL where:
  - data is split into groups that share common attributes
  - a function is applied to each group
  - function results are combined into a result object

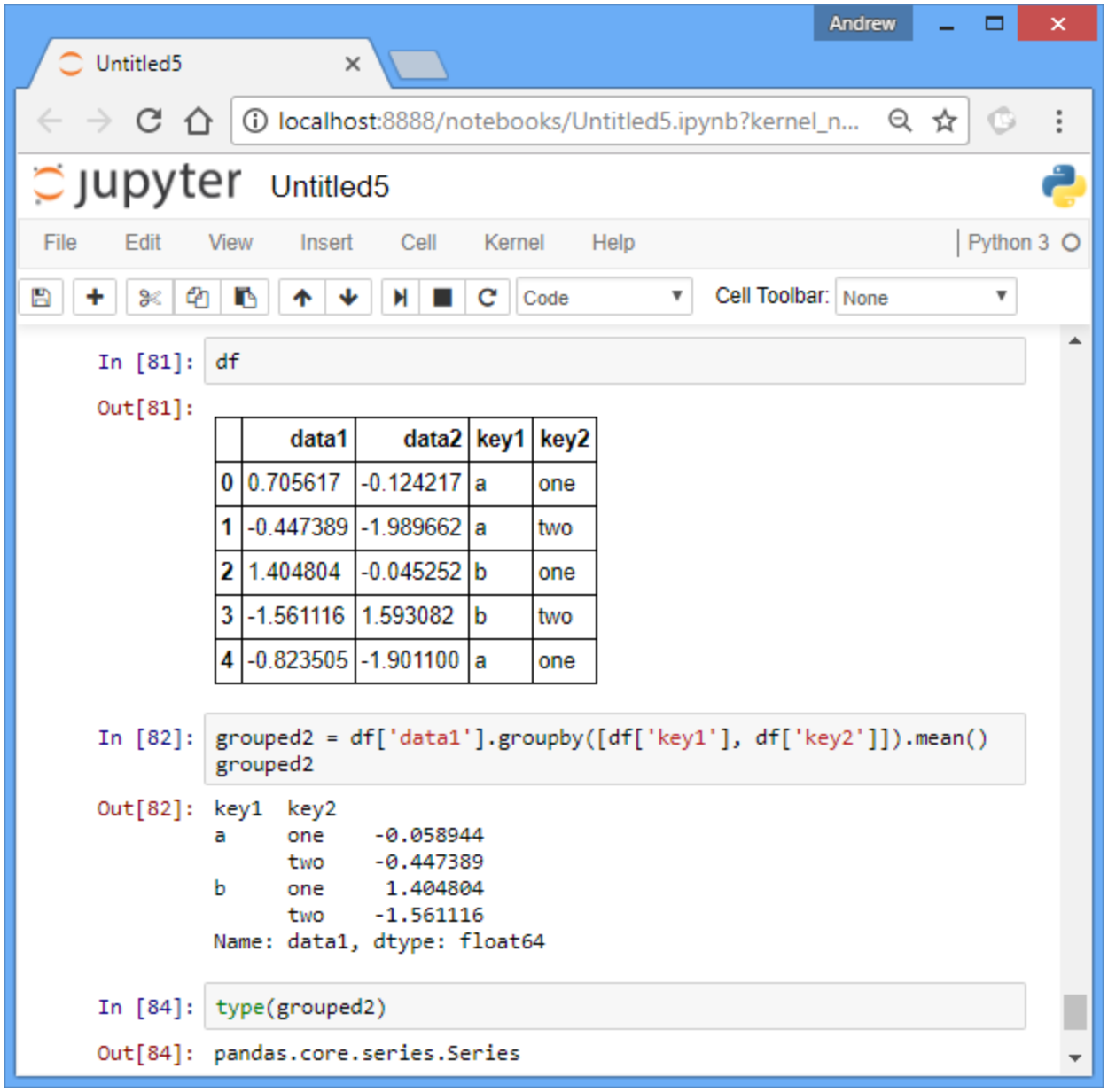
Illustration of a group aggregation



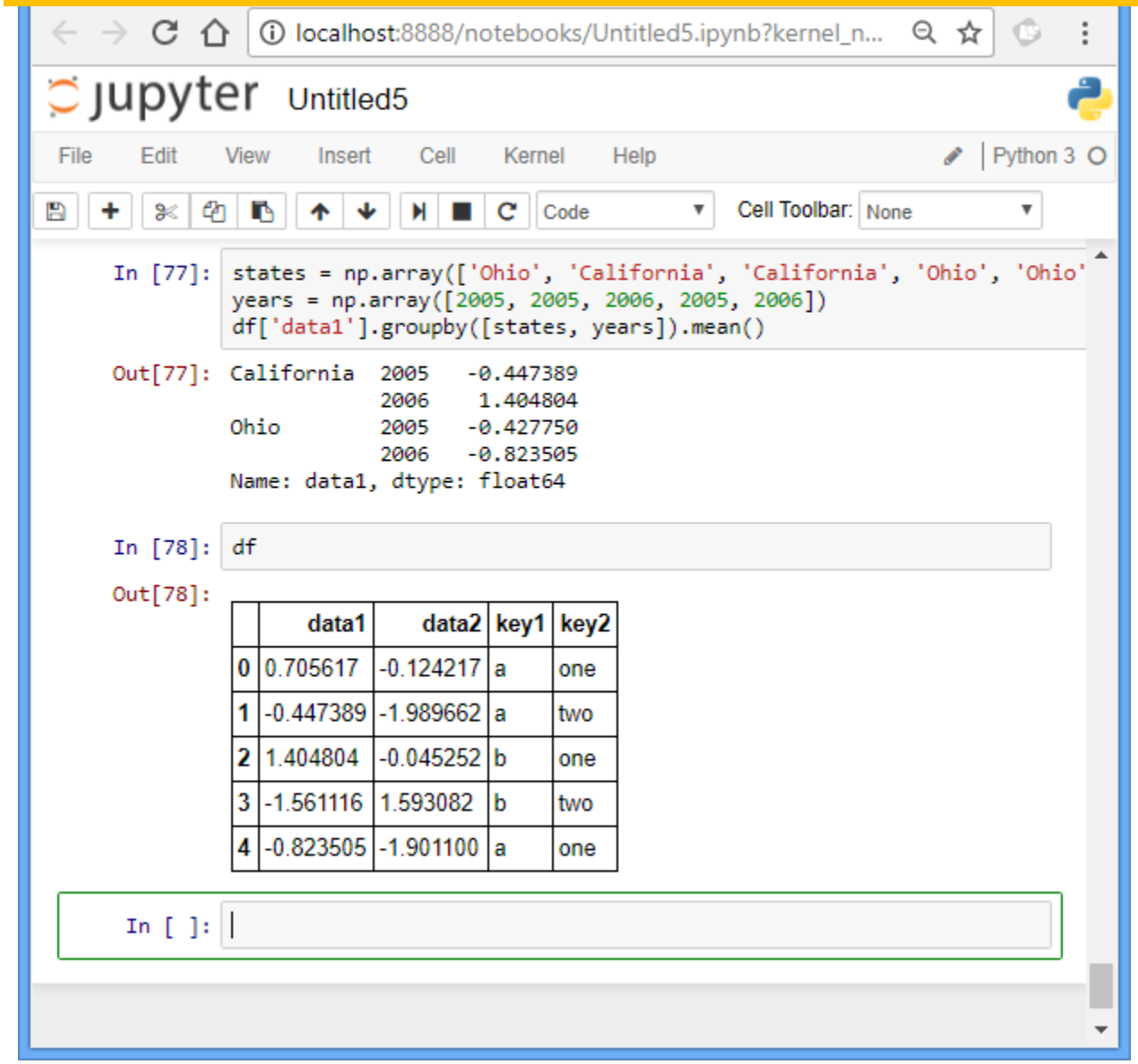
Grouping Key

- Data is split into groups based on the vales of a key or several keys
- Each grouping key can be
  - a list, array, dict, or series object
  - a column name in a DataFrame
- The keys do not have to be all of the same type


Andrew
Untitled5
localhost:8888/notebooks/Untitled5.ipynb?kernel_n...
jupyter Untitled5
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None
In [74]:
df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
'key2' : ['one', 'two', 'one', 'two', 'one'],
'data1' : np.random.randn(5),
'data2' : np.random.randn(5)})
df
Out[74]:
data1 data2 key1 key2
0 0.705617 -0.124217 a one
1 -0.447389 -1.989662 a two
2 1.404804 -0.045252 b one
3 -1.561116 1.593082 b two
4 -0.823505 -1.901100 a one
In [75]:
grouped = df['data1'].groupby(df['key1'])
grouped
Out[75]:
<pandas.core.groupby.SeriesGroupBy object at 0x00000099A5DBBB70>
In [76]:
grouped.mean()
Out[76]:
key1
a -0.188426
b -0.078156
Name: data1, dtype: float64


Andrew
Untitled5
localhost:8888/notebooks/Untitled5.ipynb?kernel_n...
jupyter Untitled5
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None
In [81]: df
Out[81]:
data1 data2 key1 key2
0 0.705617 -0.124217 a one
1 -0.447389 -1.989662 a two
2 1.404804 -0.045252 b one
3 -1.561116 1.593082 b two
4 -0.823505 -1.901100 a one
In [82]: grouped2 = df['data1'].groupby([df['key1'], df['key2']]).mean()
grouped2
Out[82]: key1 key2
a one -0.058944
two -0.447389
b one 1.404804
two -1.561116
Name: data1, dtype: float64
In [84]: type(grouped2)
Out[84]: pandas.core.series.Series

Using arrays as the grouping keys



	data1	data2	key1	key2
0	0.705617	-0.124217	a	one
1	-0.447389	-1.989662	a	two
2	1.404804	-0.045252	b	one
3	-1.561116	1.593082	b	two
4	-0.823505	-1.901100	a	one

Andrew

Untitled5

localhost:8888/notebooks/Untitled...

jupyter Untitled5

File Edit View Insert Cell Kernel Help Python 3

Code

Cell Toolbar: None

In [86]:
```
df
```

Out[86]:

	data1	data2	key1	key2
0	0.705617	-0.124217	a	one
1	-0.447389	-1.989662	a	two
2	1.404804	-0.045252	b	one
3	-1.561116	1.593082	b	two
4	-0.823505	-1.901100	a	one

In [87]:
```
df.groupby('key1').mean()
```

Out[87]:

	data1	data2
key1		
a	-0.188426	-1.338326
b	-0.078156	0.773915

Notice: there is no key2 column in the result as df['key2'] is not numeric data

In [88]:
```
df.groupby(['key1', 'key2']).mean()
```

Out[88]:

		data1	data2
key1	key2		
a	one	-0.058944	-1.012658
	two	-0.447389	-1.989662
b	one	1.404804	-0.045252
	two	-1.561116	1.593082

Andrew

Untitled5

localhost:8888/notebooks/Untitle...

jupyter Untitled5

File Edit View Insert Cell Kernel Help Python 3

Code

Cell Toolbar: None

In [90]:
```
df.groupby('key1').size()
```

Out[90]:
```
key1
a    3
b    2
dtype: int64
```

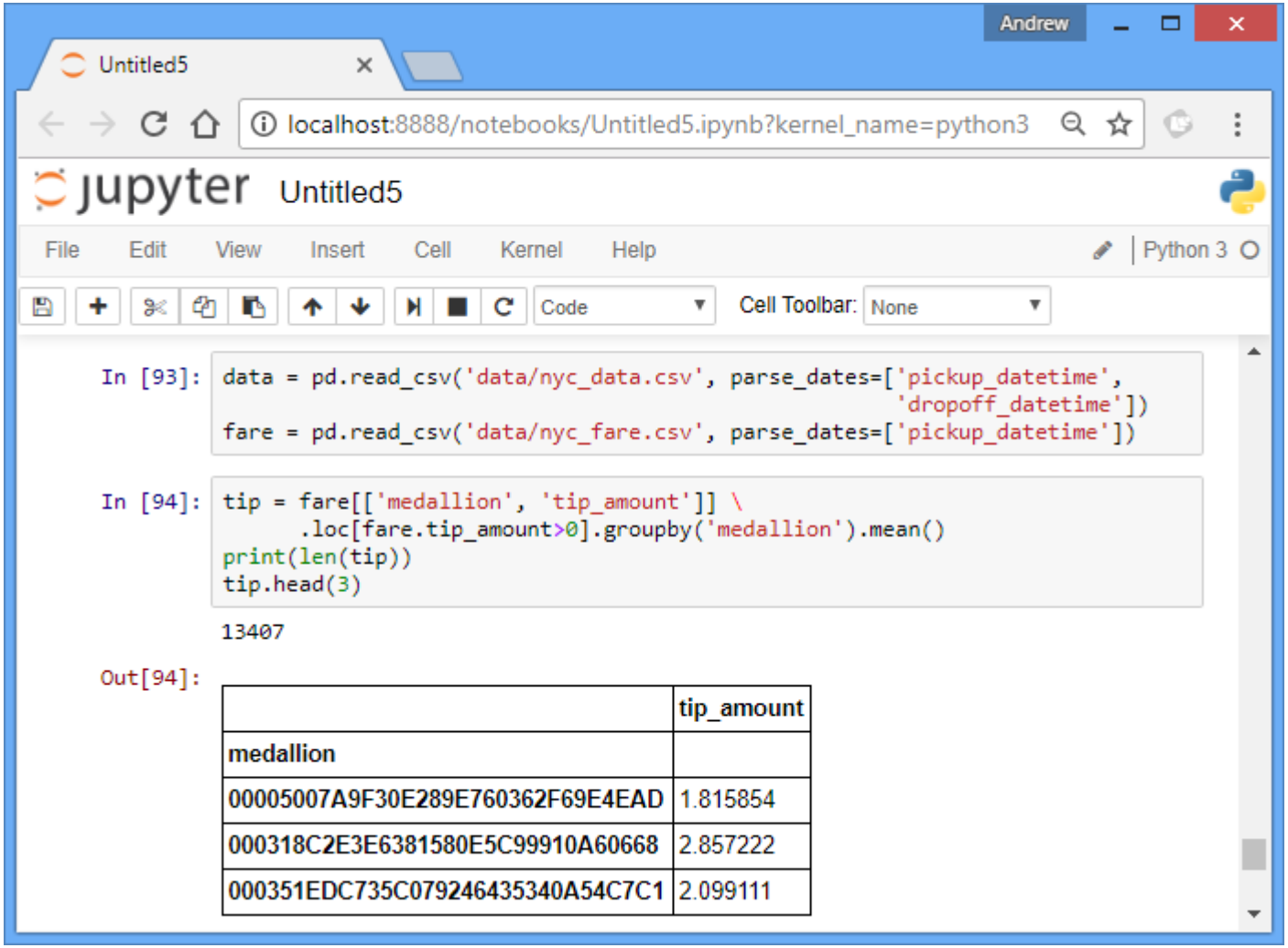
In [91]:
```
df.groupby(['key1', 'key2']).size()
```

Out[91]:
```
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

The GroupBy method **size()** returns group sizes

Joins

- pd.merge
- The idea is similar to joins in databases. That is to join data from more than one table
- Consider the NY city taxi dataset
- The DFs data and fare have same number of rows (one row for each ride)
- We want to get the average tip from the fare DF
- We will then join the average tip into the data DF



	tip_amount
medallion	
00005007A9F30E289E760362F69E4EAD	1.815854
000318C2E3E6381580E5C99910A60668	2.857222
000351EDC735C079246435340A54C7C1	2.099111

Andrew

Untitled10

localhost:8889/notebooks/Untitled10.ipynb?kernel_name=python3

jupyter Untitled10 Last Checkpoint: 17 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Help Python 3

Code Cell Toolbar: None

In [13]:
```
data_merged = pd.merge(data, tip, left_on='medallion', right_index=True)
data_merged.head(3)
```

Out[13]:

	medallion	hack_license	vendor_id	rate_code	store_
0	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E	VTS	1	NaN
4761	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E	VTS	1	NaN
22991	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E	VTS	1	NaN

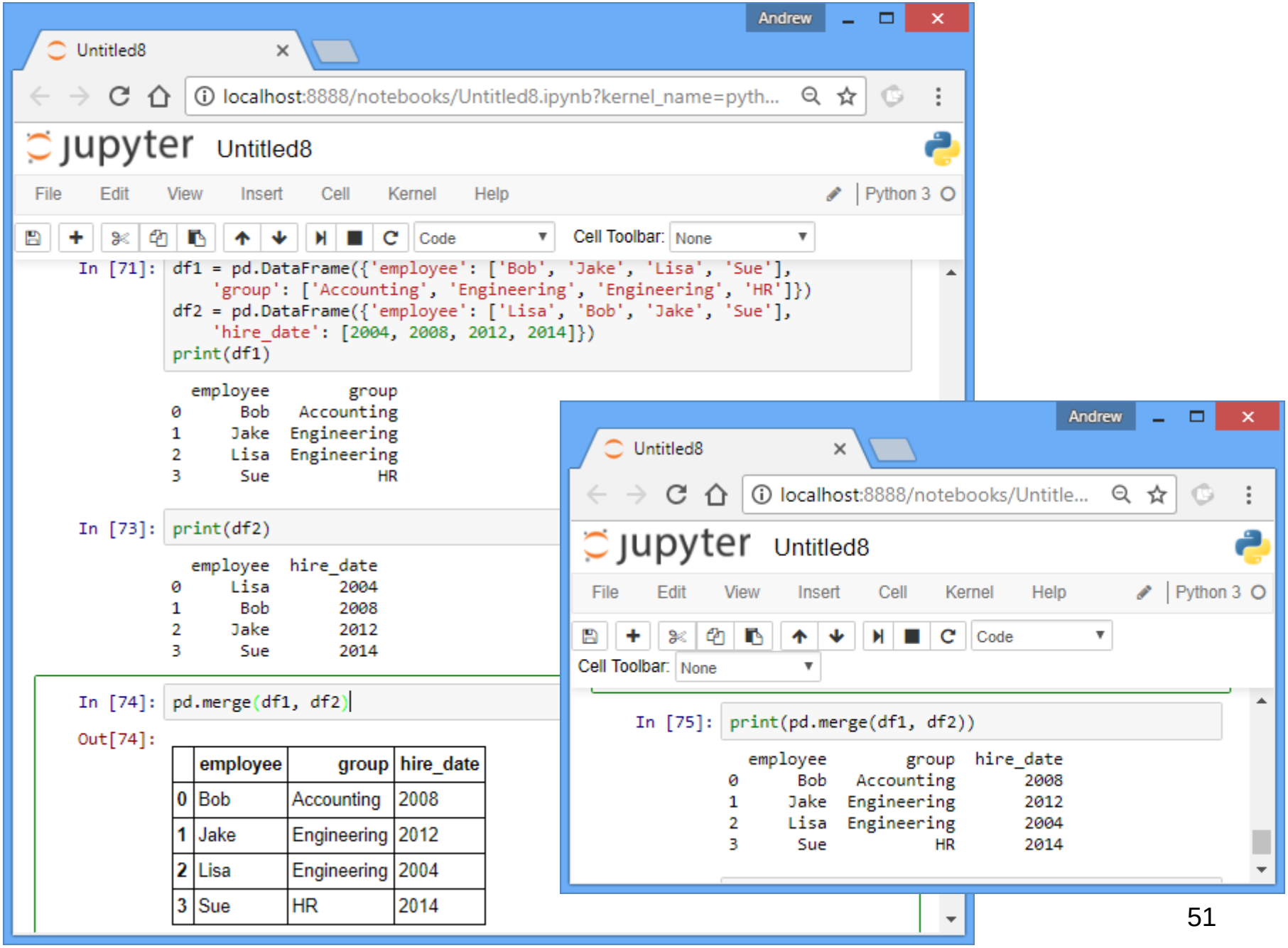
In [14]:
```
data_merged[['medallion', 'tip_amount']].head(3)
```

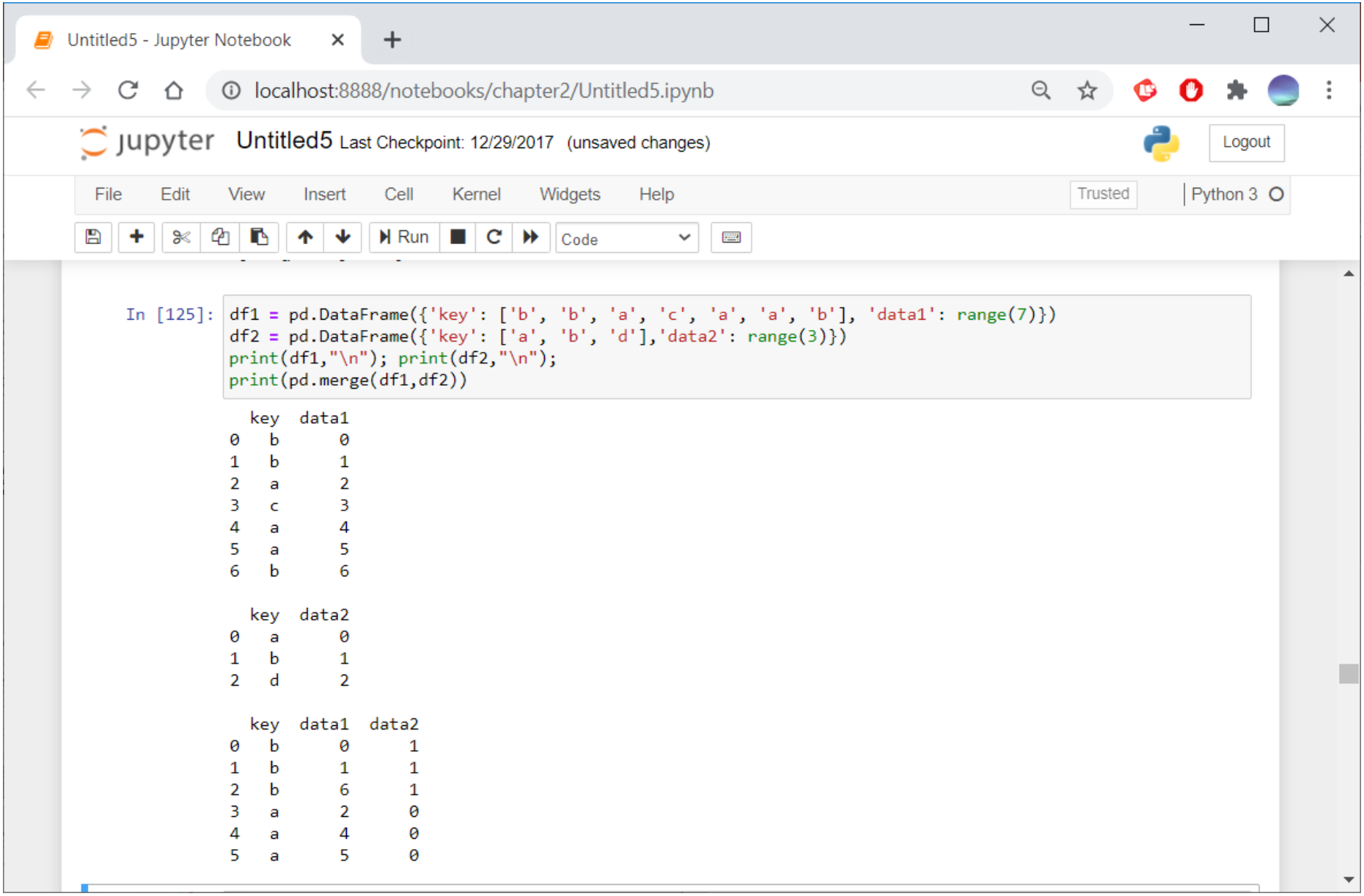
Out[14]:

	medallion	tip_amount
0	76942C3205E17D7E7FE5A9F709D16434	3.180417
4761	76942C3205E17D7E7FE5A9F709D16434	3.180417
22991	76942C3205E17D7E7FE5A9F709D16434	3.180417

More about the merge Operation

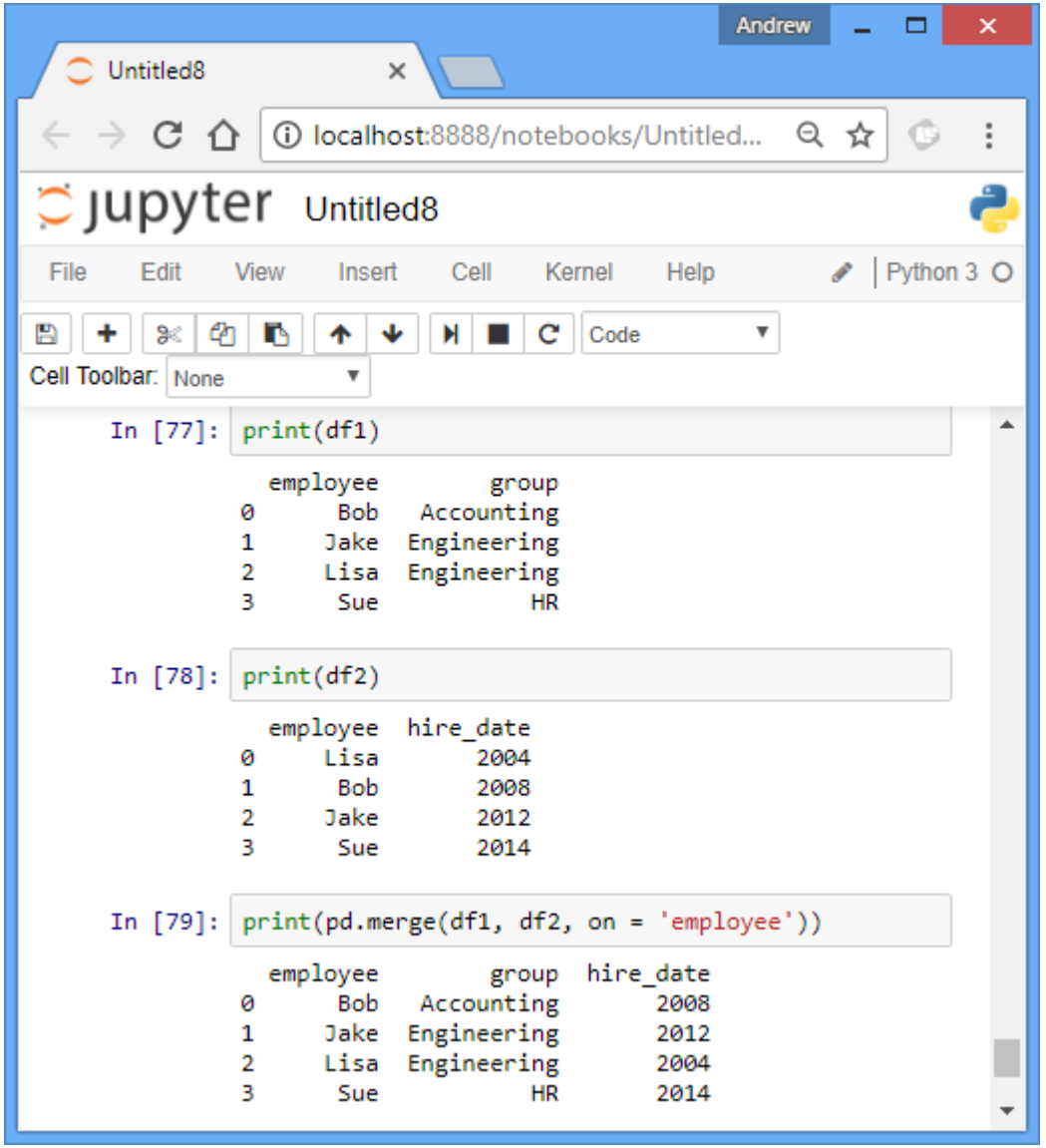
- Idea is similar to the join operation in SQL where pandas will combine two DFs where values under common columns are the same


Andrew
Untitled8
localhost:8888/notebooks/Untitled8.ipynb?kernel_name=pyth...
jupyter Untitled8
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None
In [71]:
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
'hire_date': [2004, 2008, 2012, 2014]})
print(df1)
employee group
0 Bob Accounting
1 Jake Engineering
2 Lisa Engineering
3 Sue HR
In [73]:
print(df2)
employee hire_date
0 Lisa 2004
1 Bob 2008
2 Jake 2012
3 Sue 2014
In [74]:
pd.merge(df1, df2)
Out[74]:
employee group hire_date
0 Bob Accounting 2008
1 Jake Engineering 2012
2 Lisa Engineering 2004
3 Sue HR 2014
Andrew
Untitled8
localhost:8888/notebooks/Untitle...
jupyter Untitled8
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None
In [75]:
print(pd.merge(df1, df2))
employee group hire_date
0 Bob Accounting 2008
1 Jake Engineering 2012
2 Lisa Engineering 2004
3 Sue HR 2014

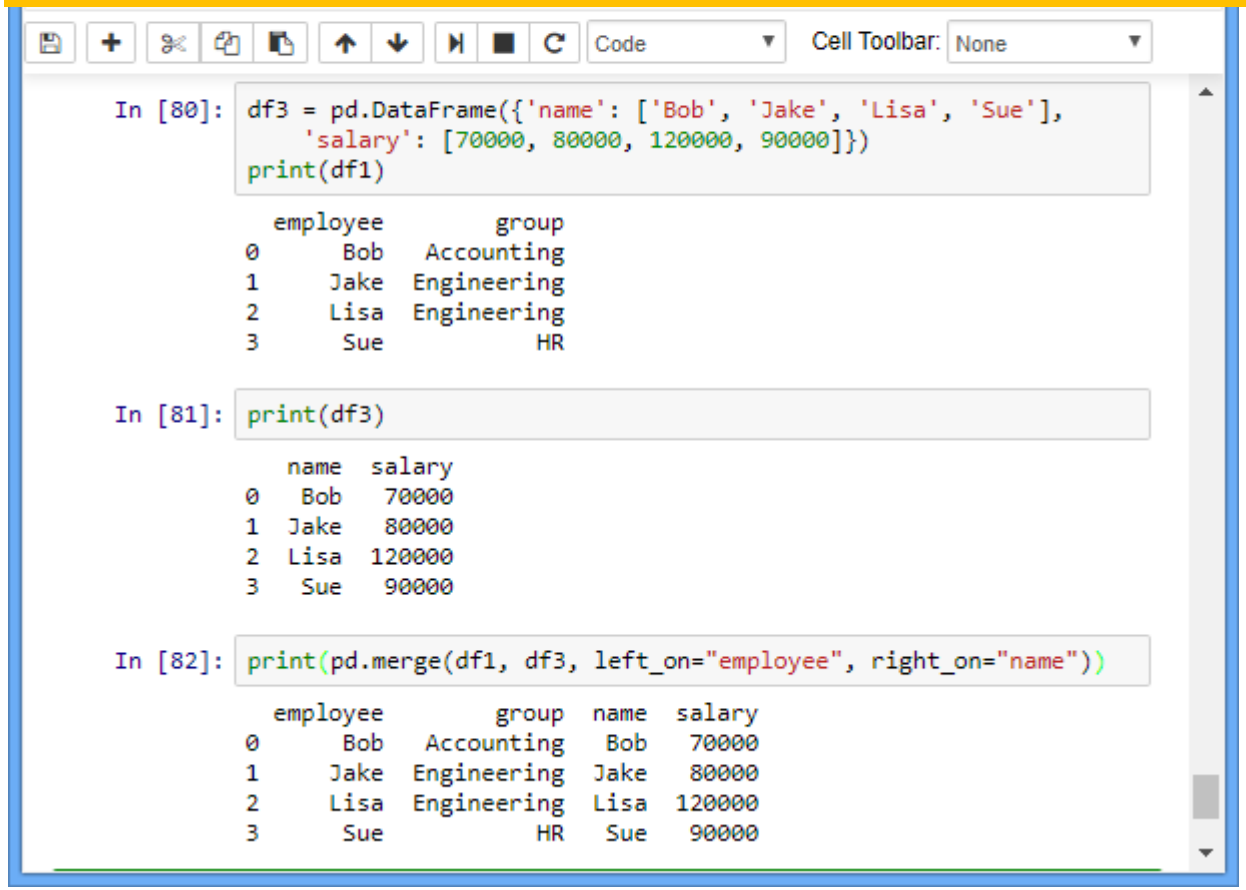

Untitled5 - Jupyter Notebook
localhost:8888/notebooks/chapter2/Untitled5.ipynb
jupyter Untitled5 Last Checkpoint: 12/29/2017 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [125]:
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'], 'data1': range(7)})
df2 = pd.DataFrame({'key': ['a', 'b', 'd'],'data2': range(3)})
print(df1,"\n"); print(df2,"\n");
print(pd.merge(df1,df2))
key data1
0 b 0
1 b 1
2 a 2
3 c 3
4 a 4
5 a 5
6 b 6
key data2
0 a 0
1 b 1
2 d 2
key data1 data2
0 b 0 1
1 b 1 1
2 b 6 1
3 a 2 0
4 a 4 0
5 a 5 0

The on Keyword Argument

- The attribute **on** allows us to explicitly specify the name of the key column(s)
- It takes either a column name or a list of column names



If the columns to join on have different names, we can use the **left_on** and **right_on** parameters to specify the column names



We can use the DataFrame's **drop()** method to remove redundant columns

Andrew

Untitled8

localhost:8888/notebooks/Untitled8.ipynb?kernel_name=python3

jupyter Untitled8 Last Checkpoint: a day ago (autosaved)

File Edit View Insert Cell Kernel Help Python 3

Code Cell Toolbar: None

```
In [82]: print(pd.merge(df1, df3, left_on="employee", right_on="name"))
```

```
  employee        group  name  salary
0      Bob   Accounting   Bob   70000
1     Jake  Engineering  Jake   80000
2     Lisa  Engineering  Lisa  120000
3      Sue           HR   Sue   90000
```

```
In [84]: print(pd.merge(df1, df3, left_on="employee", right_on="name").drop('name', axis = 1))
```

```
  employee        group  salary
0      Bob   Accounting   70000
1     Jake  Engineering   80000
2     Lisa  Engineering  120000
3      Sue           HR   90000
```

```
In [86]: pd.merge(df1, df3, left_on="employee", right_on="name").drop(2)
```

Out[86]:

	employee	group	name	salary
0	Bob	Accounting	Bob	70000
1	Jake	Engineering	Jake	80000
3	Sue	HR	Sue	90000

Removing a row

DataFrame.set_index

```
In [87]: print(df1)
```

```
  employee        group
0      Bob   Accounting
1     Jake  Engineering
2     Lisa  Engineering
3      Sue           HR
```

```
In [88]: print(df2)
```

```
  employee  hire_date
0     Lisa       2004
1      Bob       2008
2     Jake       2012
3      Sue       2014
```

```
In [89]: df1a = df1.set_index('employee')
         print(df1a)
```

```
                group
employee
Bob        Accounting
Jake      Engineering
Lisa      Engineering
Sue                HR
```

```
In [92]: df2a = df2.set_index('employee')
         print(df2a)
```

```
          hire_date
employee
Lisa           2004
Bob            2008
Jake           2012
Sue            2014
```

Untitled5 - Jupyter Notebook

localhost:8888/noteboo...

upyter Untitled5

Logout

Trusted | Python 3

Edit View Insert Cell Kernel Widgets Help

Run Code

In [111]:
```
df
```

Out[111]:

	month	year	sale
0	1	2012	55
1	4	2014	40
2	7	2013	84
3	10	2014	31

In [118]:
```
df.set_index(pd.Index(["a", "b", "c", "d"]), inplace = True)
```

In [119]:
```
df
```

Out[119]:

	month	year	sale
a	1	2012	55
b	4	2014	40
c	7	2013	84
d	10	2014	31

Untitled5 - Jupyter Notebook

localhost:8888/noteboo...

upyter Untitled5

Edit View Insert Cell Kernel W

Run

In [120]:
```
df.drop("a")
```

Out[120]:

	month	year	sale
b	4	2014	40
c	7	2013	84
d	10	2014	31

In [121]:
```
df
```

Out[121]:

	month	year	sale
a	1	2012	55
b	4	2014	40
c	7	2013	84
d	10	2014	31

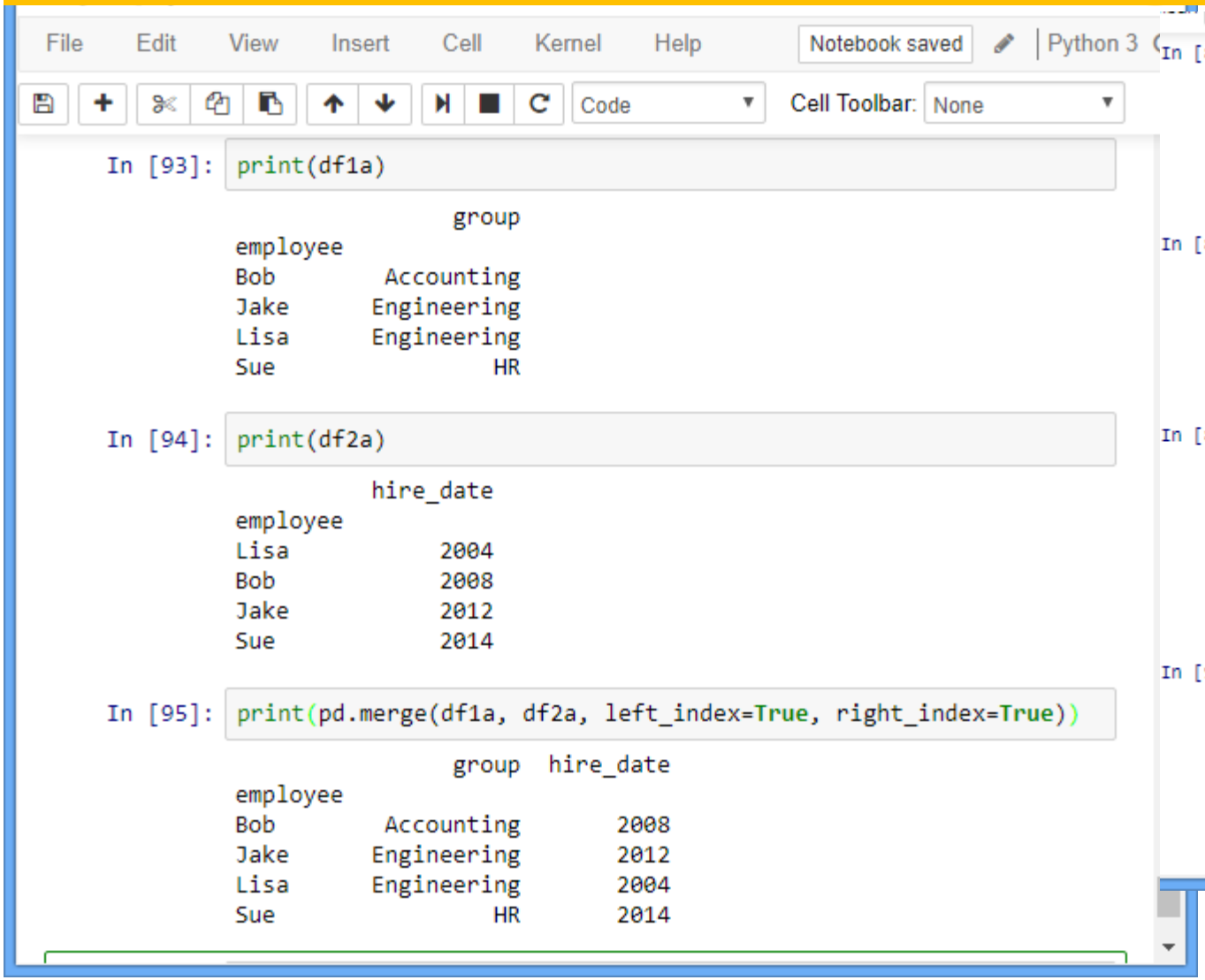
In [123]:
```
df.drop('a', inplace = True)
```

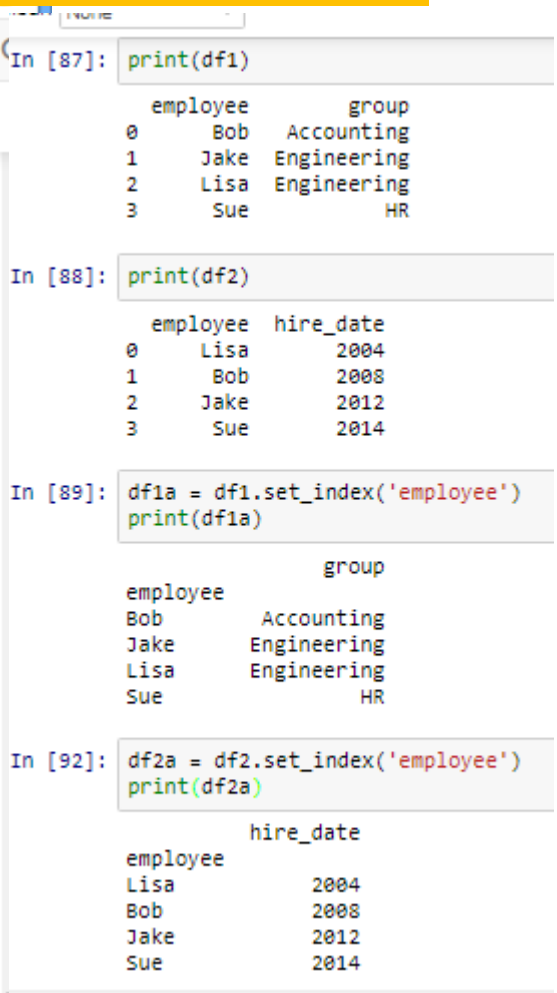
In [124]:
```
df
```

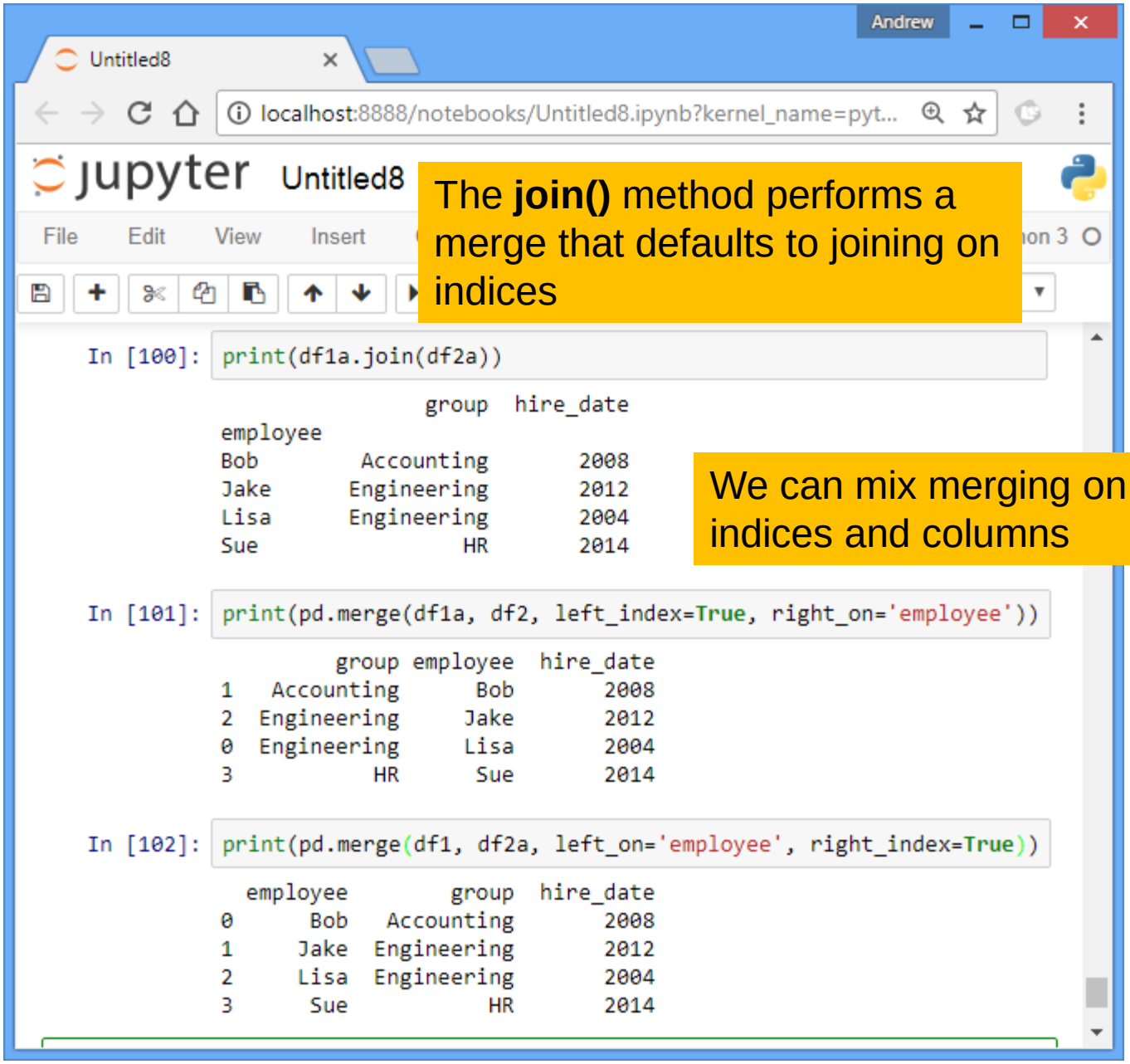
Out[124]:

	month	year	sale
b	4	2014	40
c	7	2013	84
d	10	2014	31

Joining on Indexes






Andrew
Untitled8
localhost:8888/notebooks/Untitled8.ipynb?kernel_name=pyt...
jupyter Untitled8
File Edit View Insert
The **join()** method performs a merge that defaults to joining on indices
In [100]: print(df1a.join(df2a))
group hire_date
employee
Bob Accounting 2008
Jake Engineering 2012
Lisa Engineering 2004
Sue HR 2014
We can mix merging on indices and columns
In [101]: print(pd.merge(df1a, df2, left_index=True, right_on='employee'))
group employee hire_date
1 Accounting Bob 2008
2 Engineering Jake 2012
0 Engineering Lisa 2004
3 HR Sue 2014
In [102]: print(pd.merge(df1, df2a, left_on='employee', right_index=True))
employee group hire_date
0 Bob Accounting 2008
1 Jake Engineering 2012
2 Lisa Engineering 2004
3 Sue HR 2014

Attributes lsuffix and rsuffix

```
In [33]: df1 = pd.DataFrame({'name':['Mike', 'Bob', 'Jim'], 'age': [21, 25, 17]})
         df1
```

Out[33]:

	name	age
0	Mike	21
1	Bob	25
2	Jim	17

```
In [34]: df2 = pd.DataFrame({'name':['Mike', 'Bob', 'Jim'], 'group': ['HR', 'Accounting', 'Sales']})
         df2
```

Out[34]:

	name	group
0	Mike	HR
1	Bob	Accounting
2	Jim	Sales

```
In [47]: df1.join(df2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
```

p_ex1 - Jupyter Notebook

localhost:8888/notebooks/p_ex1.ipynb

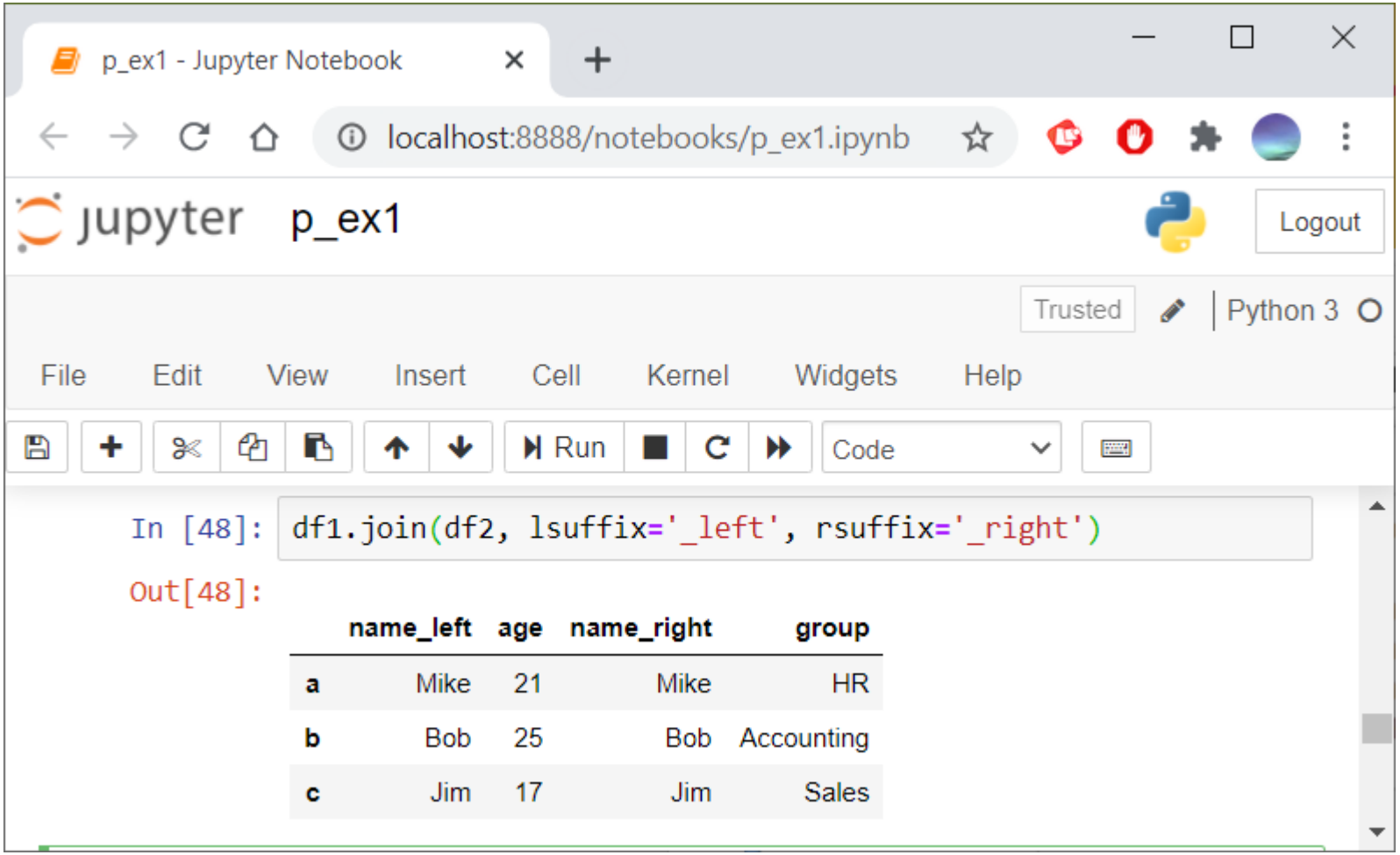
jupyter p_ex1 Last Checkpoint: Last Sunday at 3:28 PM (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
1973
1974         def lrenamer(x):

ValueError: columns overlap but no suffix specified: Index(['name'], dtype='object')
```

Attributes lsuffix and rsuffix



	name_left	age	name_right	group
a	Mike	21	Mike	HR
b	Bob	25	Bob	Accounting
c	Jim	17	Jim	Sales

The how Attribute

- Notice, by default pandas performs **inner** join (intersect)
- For full outer join (union), use the attribute how = 'outer'
- Inner join may be specified explicitly using how = 'inner'
- Other possible values for **how** are **left** and **right** for left/right outer joins

Andrew

Untitled8

localhost:8888/notebooks/Untitled8.ipynb?kernel_name=python3

jupyter Untitled8 Last Checkpoint: Yesterday at 2:52 PM (autosaved)

File Edit View Insert Cell Kernel Help Python 3

Code Cell Toolbar: None

In [105]:
```
df6 = pd.DataFrame({'name': ['Peter', 'Paul', 'Mary'], 'food': ['fish', 'beans', 'bread']},
                   columns=['name', 'food'])
print(df6)
```

```
    name   food
0  Peter   fish
1   Paul  beans
2   Mary  bread
```

In [106]:
```
df7 = pd.DataFrame({'name': ['Mary', 'Joseph'], 'drink': ['water', 'juice']},
                   columns=['name', 'drink'])
print(df7)
```

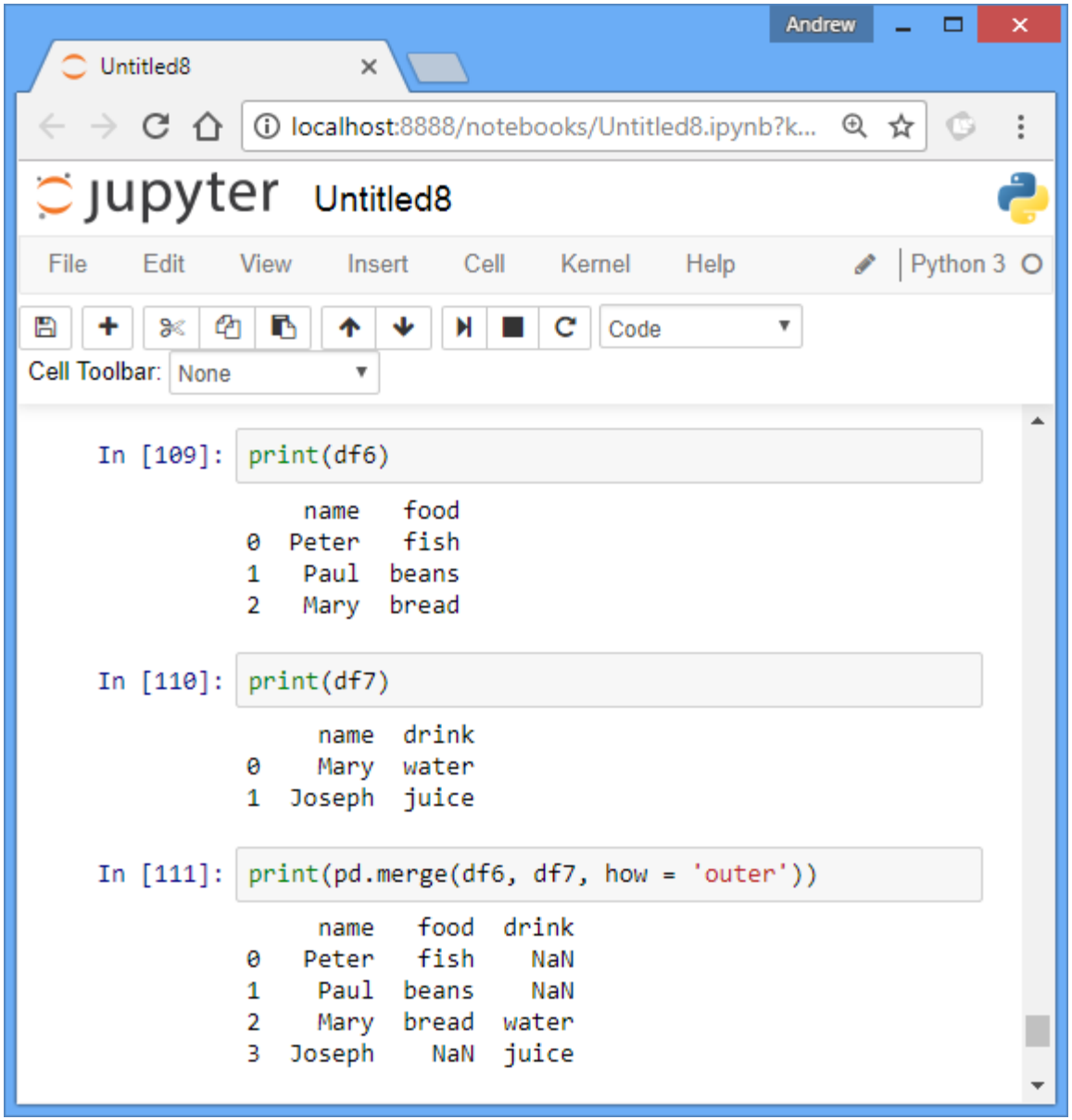
```
     name  drink
0    Mary  water
1  Joseph  juice
```

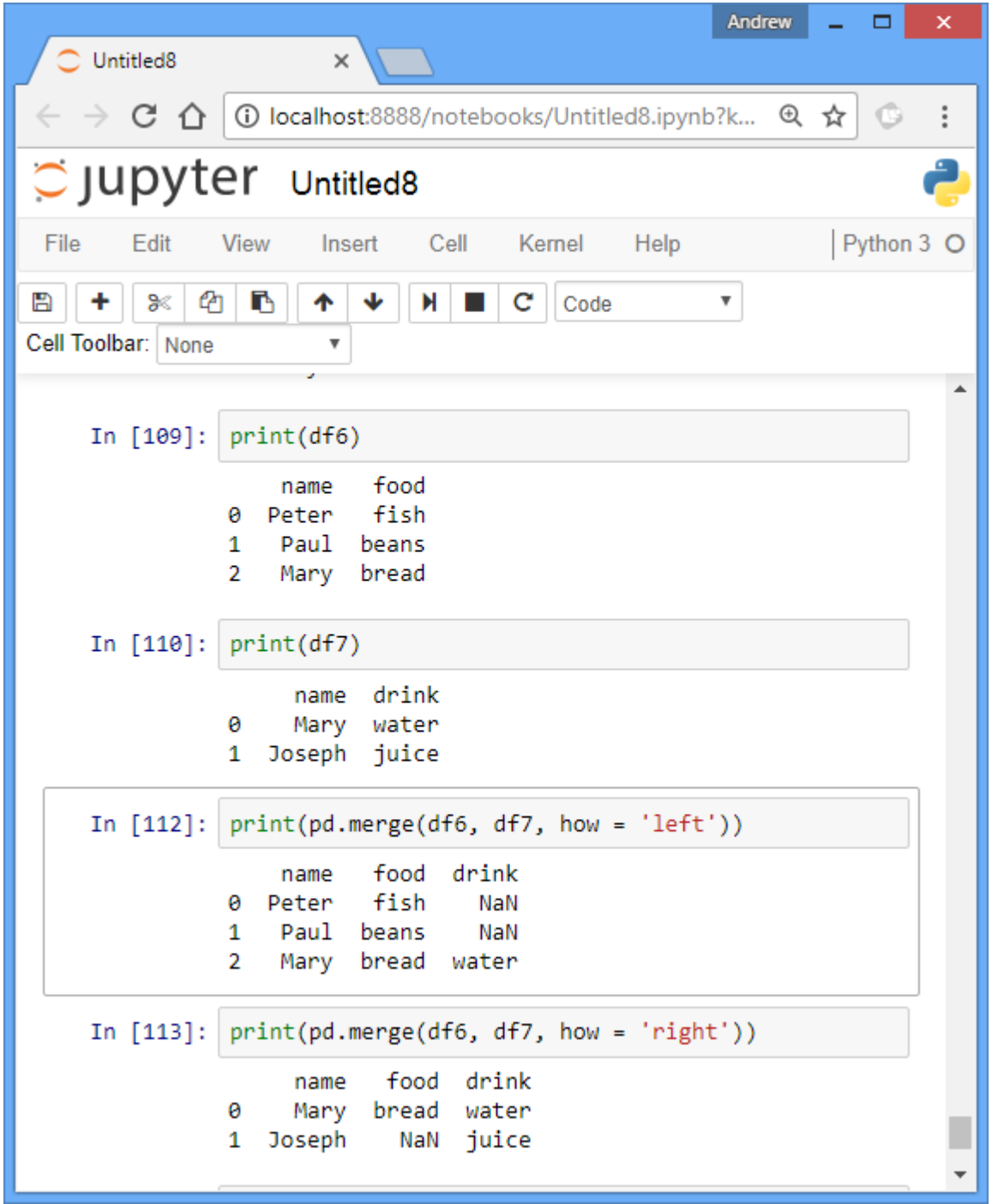
In [107]:
```
print(pd.merge(df6, df7))
```

```
   name   food  drink
0  Mary  bread  water
```

In [108]:
```
print(pd.merge(df6, df7, how = 'inner'))
```

```
   name   food  drink
0  Mary  bread  water
```


Andrew
Untitled8
localhost:8888/notebooks/Untitled8.ipynb?k...
jupyter Untitled8
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None
In [109]: print(df6)
name food
0 Peter fish
1 Paul beans
2 Mary bread
In [110]: print(df7)
name drink
0 Mary water
1 Joseph juice
In [111]: print(pd.merge(df6, df7, how = 'outer'))
name food drink
0 Peter fish NaN
1 Paul beans NaN
2 Mary bread water
3 Joseph NaN juice


Andrew
Untitled8
localhost:8888/notebooks/Untitled8.ipynb?k...
jupyter Untitled8
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None
In [109]: print(df6)
name food
0 Peter fish
1 Paul beans
2 Mary bread
In [110]: print(df7)
name drink
0 Mary water
1 Joseph juice
In [112]: print(pd.merge(df6, df7, how = 'left'))
name food drink
0 Peter fish NaN
1 Paul beans NaN
2 Mary bread water
In [113]: print(pd.merge(df6, df7, how = 'right'))
name food drink
0 Mary bread water
1 Joseph NaN juice

Renaming Columns in a DF – Function rename

jupyter p_ex1 Last Checkpoint: Last Sunday at 3:28 PM (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [61]:
```
df1 = pd.DataFrame({'name':['Mike', 'Bob', 'Jim'], 'group': ['HR', 'Accounting', 'Sales']})
df1
```

Out[61]:

	name	group
0	Mike	HR
1	Bob	Accounting
2	Jim	Sales

In [62]:
```
# Renaming one column
df1.rename(columns = {'group': 'department'}, inplace = True)
df1
```

Out[62]:

	name	department
0	Mike	HR
1	Bob	Accounting
2	Jim	Sales

In [63]:
```
# Renaming several columns
df1.rename(columns = {'name':'person', 'group': 'department'}, inplace = True)
df1
```

Out[63]:

	person	department
0	Mike	HR
1	Bob	Accounting
2	Jim	Sales

DF.rename

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [74]:
```
# Renaming several columns
df1.rename(columns = {'name':'person', 'group': 'department'}, inplace = True)
df1
```

Out[74]:

	person	department
0	Mike	HR
1	Bob	Accounting
2	Jim	Sales

In [75]:
```
df1.rename({'person': 'Preson-Name'}, axis = 'columns')
```

Out[75]:

	Preson-Name	department
0	Mike	HR
1	Bob	Accounting
2	Jim	Sales

In [76]:
```
df1.rename({'person': 'Preson-Name'}, axis = 1)
```

Out[76]:

	Preson-Name	department
0	Mike	HR
1	Bob	Accounting
2	Jim	Sales

DF.rename

In [81]:
```
df1
```

Out[81]:

	person	department
0	Mike	HR
1	Bob	Accounting
2	Jim	Sales

In [77]:
```
df1.rename({1:4, 2:8}, axis = 'index')
```

Out[77]:

	person	department
0	Mike	HR
4	Bob	Accounting
8	Jim	Sales

In [78]:
```
df1.rename({1:4, 2:8}, axis = 0)
```

Out[78]:

	person	department
0	Mike	HR
4	Bob	Accounting
8	Jim	Sales

In [79]:
```
df1.rename({1:4, 2:8})
```

Out[79]:

	person	department
0	Mike	HR
4	Bob	Accounting
8	Jim	Sales

In [80]:
```
df1.rename({1:4, 2:8}, axis = 'rows')
```

Out[80]:

	person	department
0	Mike	HR
4	Bob	Accounting
8	Jim	Sales

Accessing an Element in a DF