Chater 5 - Getting Started with pandas

W. Mckinney, “Getting Started with pandas,” in *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Jupyther*, 3rd ed., O’Reilly, 2022, pp. 123-173.

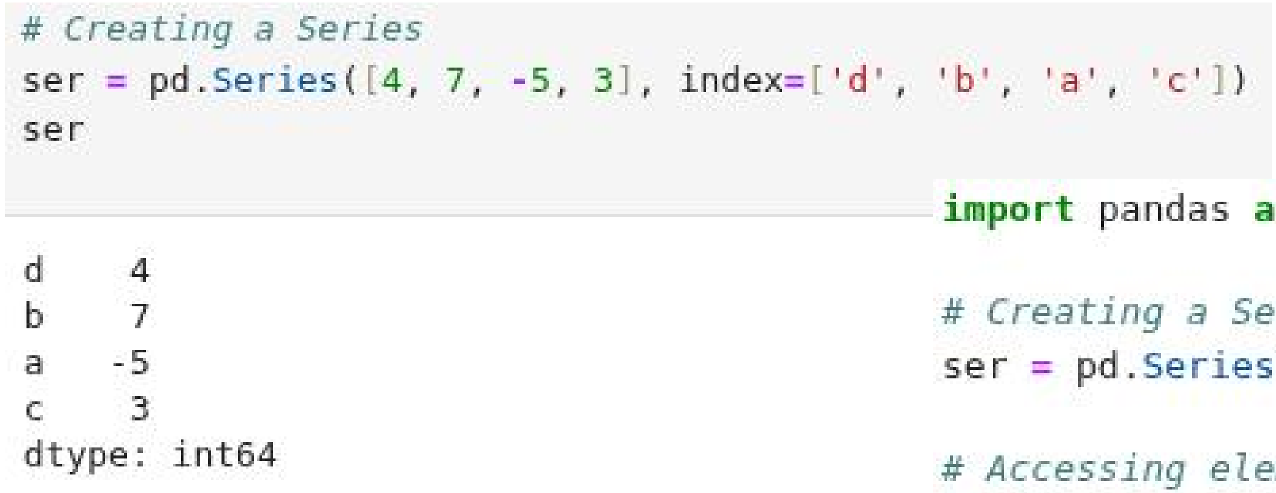
https://datamineaz.org/

Pandas

- High-Performance Library: Pandas is a Python library for fast data manipulation.
- Core Structures: It introduces DataFrame and Series for data handling.
- Data Processing: Ideal for cleaning and analyzing datasets.
- Versatile I/O: Offers extensive file format compatibility for data I/O.

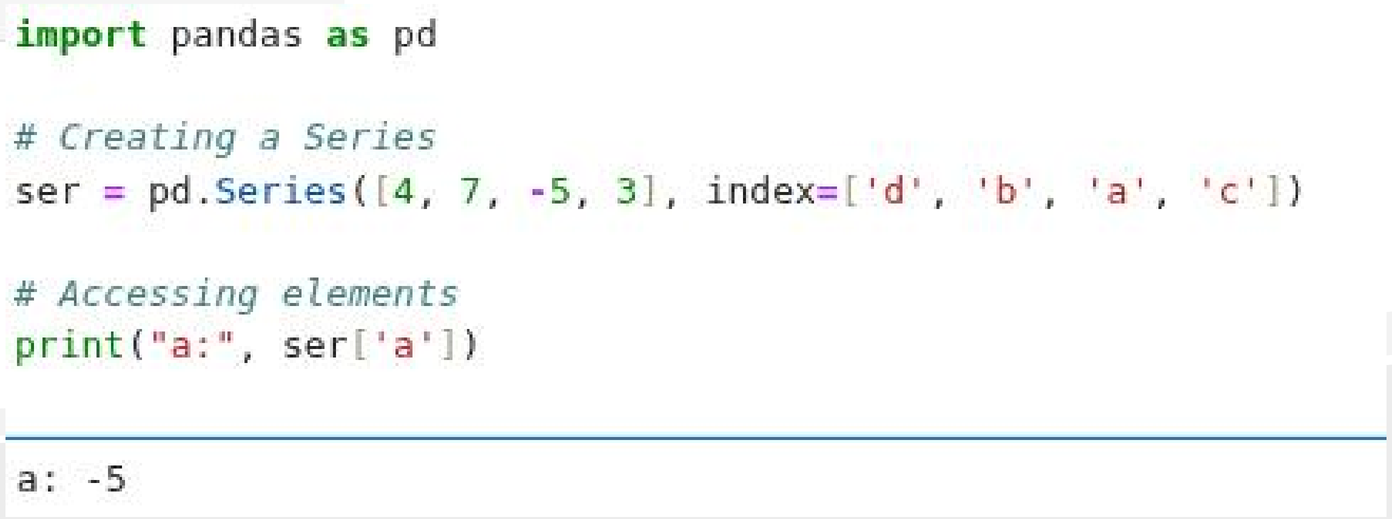
Pandas Data Structures

- Series
  - One-dimensional array-like object containing a sequence of values with an associated array of labels, its index.



```
# Creating a Series
ser = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
ser
```

```
d    4
b    7
a   -5
c    3
dtype: int64
```

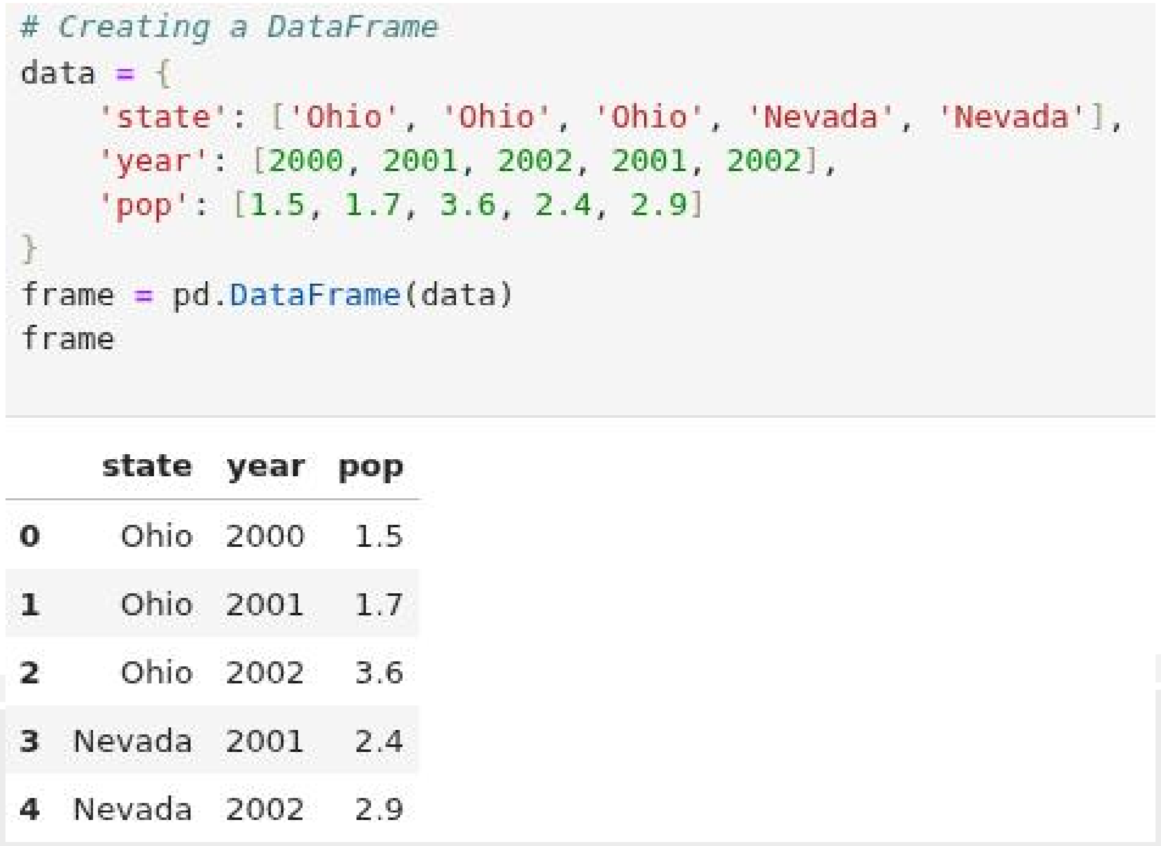


```
import pandas as pd

# Creating a Series
ser = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])

# Accessing elements
print("a:", ser['a'])
```

```
a: -5
```

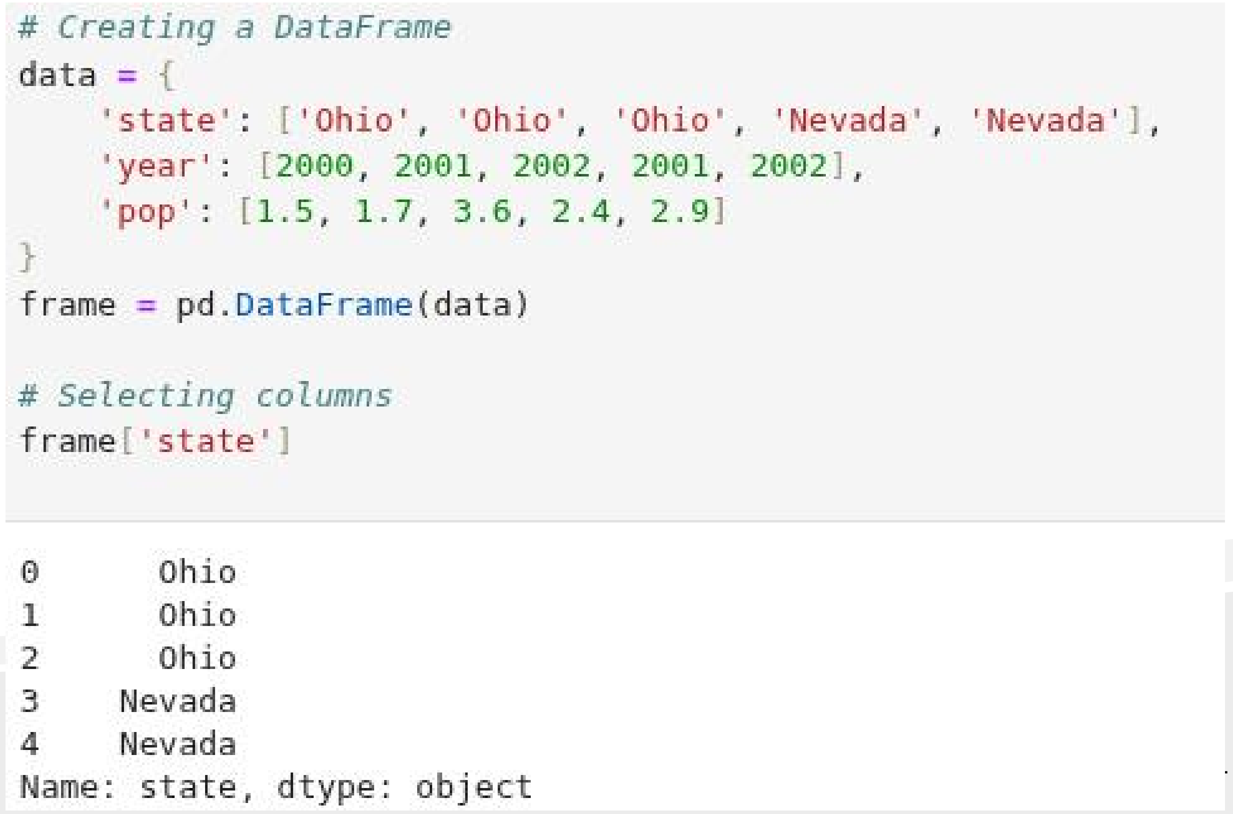
Pandas Data Structures

- DataFrame
  - A rectangular table of data with an ordered collection of columns



```python
# Creating a DataFrame
data = {
    'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
    'year': [2000, 2001, 2002, 2001, 2002],
    'pop': [1.5, 1.7, 3.6, 2.4, 2.9]
}
frame = pd.DataFrame(data)
frame
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9

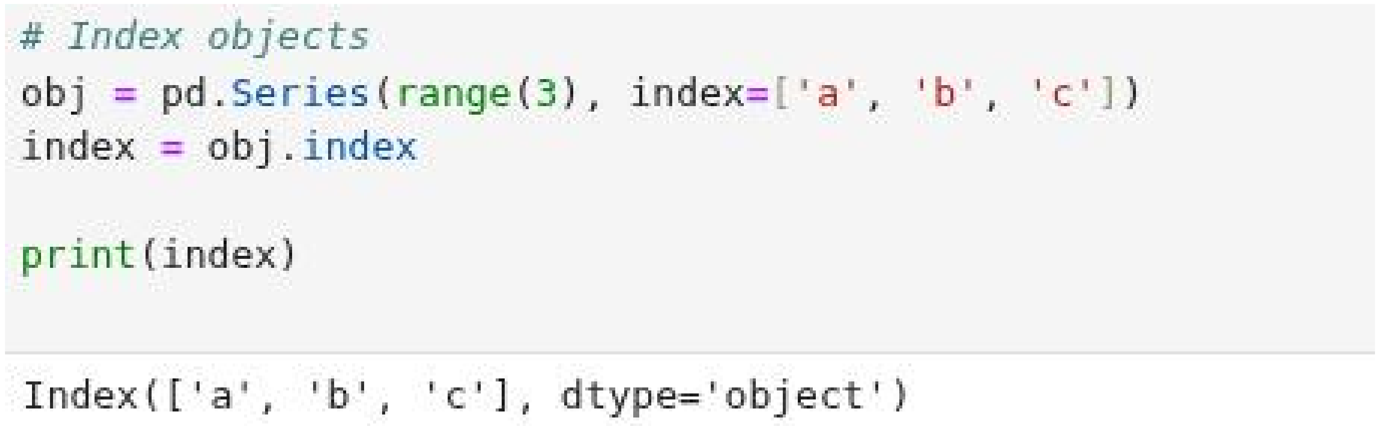


```python
# Creating a DataFrame
data = {
    'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
    'year': [2000, 2001, 2002, 2001, 2002],
    'pop': [1.5, 1.7, 3.6, 2.4, 2.9]
}
frame = pd.DataFrame(data)

# Selecting columns
frame['state']
```

```
0      Ohio
1      Ohio
2      Ohio
3    Nevada
4    Nevada
Name: state, dtype: object
```

Pandas Data Structures

- Index objects
  - Immutable, can’t be modified by a user



```
# Index objects
obj = pd.Series(range(3), index=['a', 'b', 'c'])
index = obj.index

print(index)
```

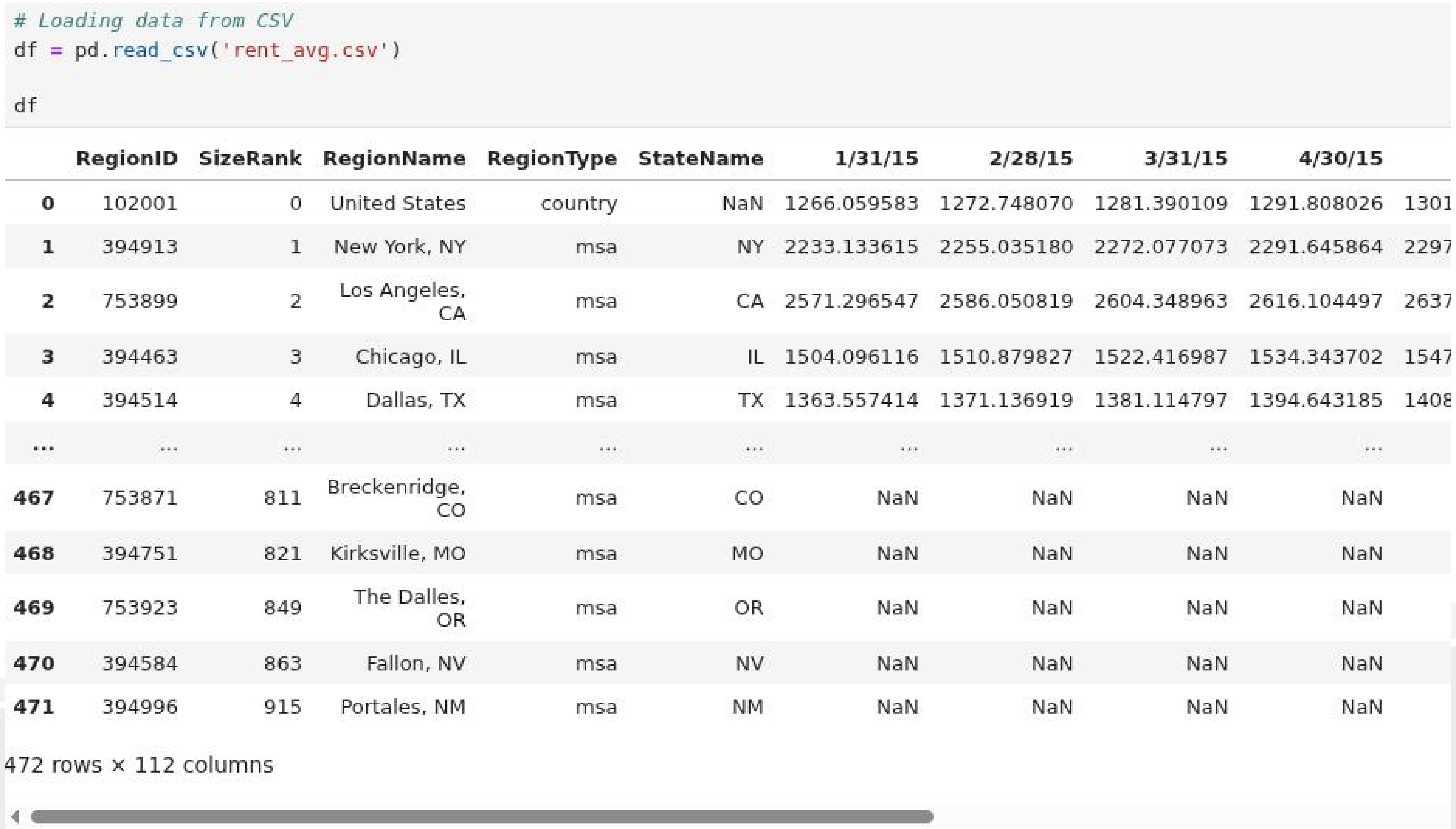
```
Index(['a', 'b', 'c'], dtype='object')
```

Reading & Inspecting Data

- .csv Files
  - More reproducible - can see changes on GitHub
  - Simple file structure
  - Standardized
  - Non-proprietary (e.g., Excel)

Reading & Inspecting Data

- Reading .csv files



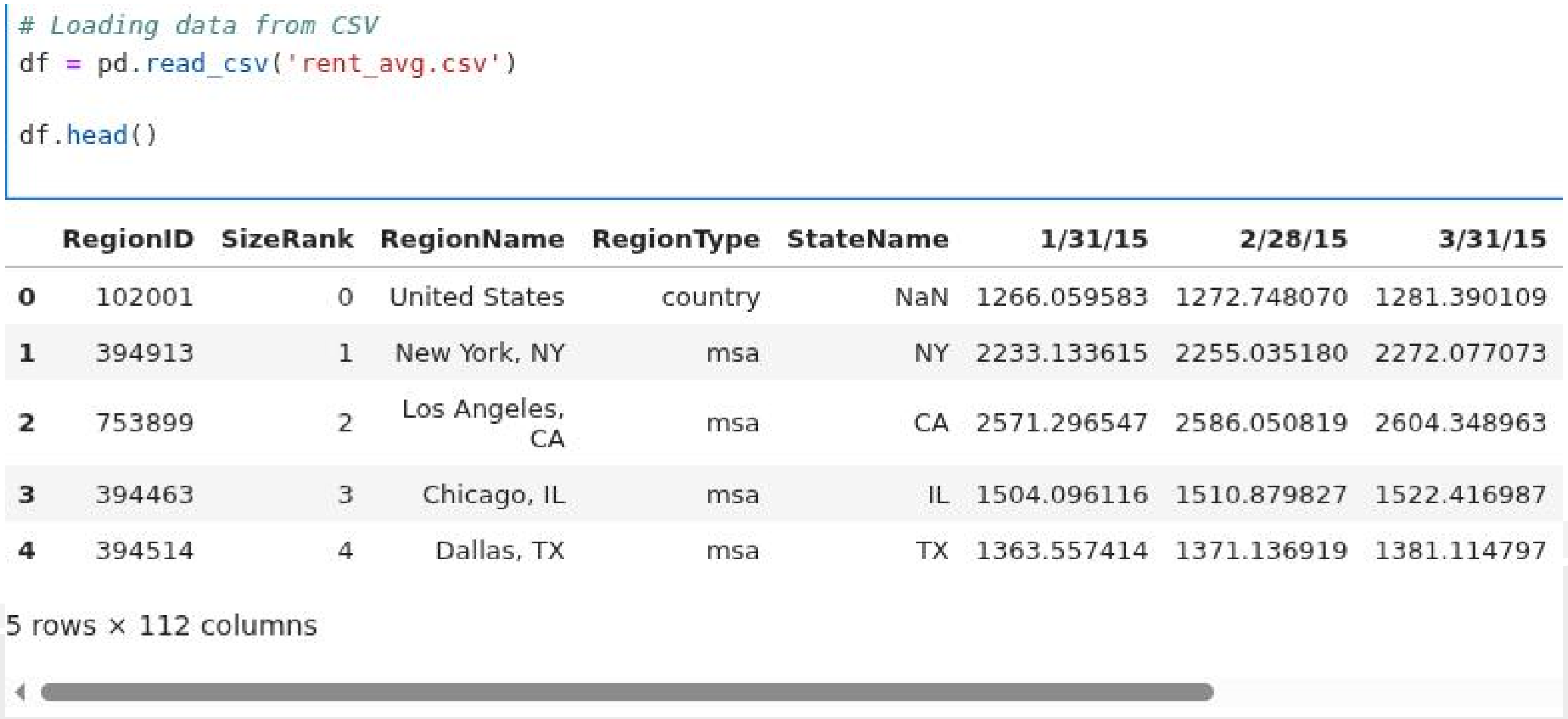
```python
# Loading data from CSV
df = pd.read_csv('rent_avg.csv')

df
```

	RegionID	SizeRank	RegionName	RegionType	StateName	1/31/15	2/28/15	3/31/15	4/30/15	
0	102001	0	United States	country	NaN	1266.059583	1272.748070	1281.390109	1291.808026	1301
1	394913	1	New York, NY	msa	NY	2233.133615	2255.035180	2272.077073	2291.645864	2297
2	753899	2	Los Angeles, CA	msa	CA	2571.296547	2586.050819	2604.348963	2616.104497	2637
3	394463	3	Chicago, IL	msa	IL	1504.096116	1510.879827	1522.416987	1534.343702	1547
4	394514	4	Dallas, TX	msa	TX	1363.557414	1371.136919	1381.114797	1394.643185	1408
...	...	...	...	...	...	...	...	...	...	
467	753871	811	Breckenridge, CO	msa	CO	NaN	NaN	NaN	NaN	
468	394751	821	Kirksville, MO	msa	MO	NaN	NaN	NaN	NaN	
469	753923	849	The Dalles, OR	msa	OR	NaN	NaN	NaN	NaN	
470	394584	863	Fallon, NV	msa	NV	NaN	NaN	NaN	NaN	
471	394996	915	Portales, NM	msa	NM	NaN	NaN	NaN	NaN	

472 rows × 112 columns

Reading & Inspecting Data

- Inspecting data
  - Head



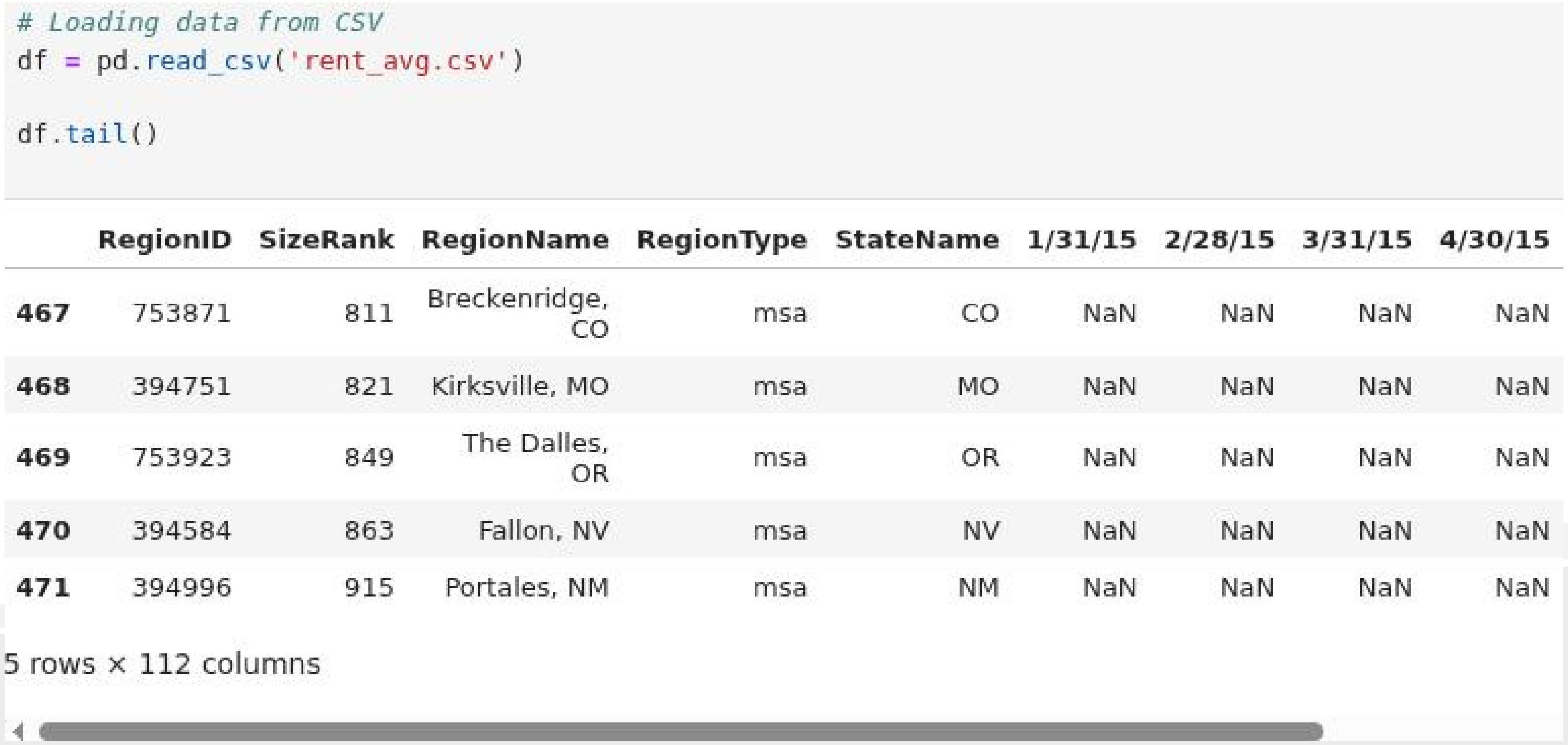
```python
# Loading data from CSV
df = pd.read_csv('rent_avg.csv')

df.head()
```

	RegionID	SizeRank	RegionName	RegionType	StateName	1/31/15	2/28/15	3/31/15
0	102001	0	United States	country	NaN	1266.059583	1272.748070	1281.390109
1	394913	1	New York, NY	msa	NY	2233.133615	2255.035180	2272.077073
2	753899	2	Los Angeles, CA	msa	CA	2571.296547	2586.050819	2604.348963
3	394463	3	Chicago, IL	msa	IL	1504.096116	1510.879827	1522.416987
4	394514	4	Dallas, TX	msa	TX	1363.557414	1371.136919	1381.114797

5 rows × 112 columns

Reading & Inspecting Data

- Inspecting data
  - Tail



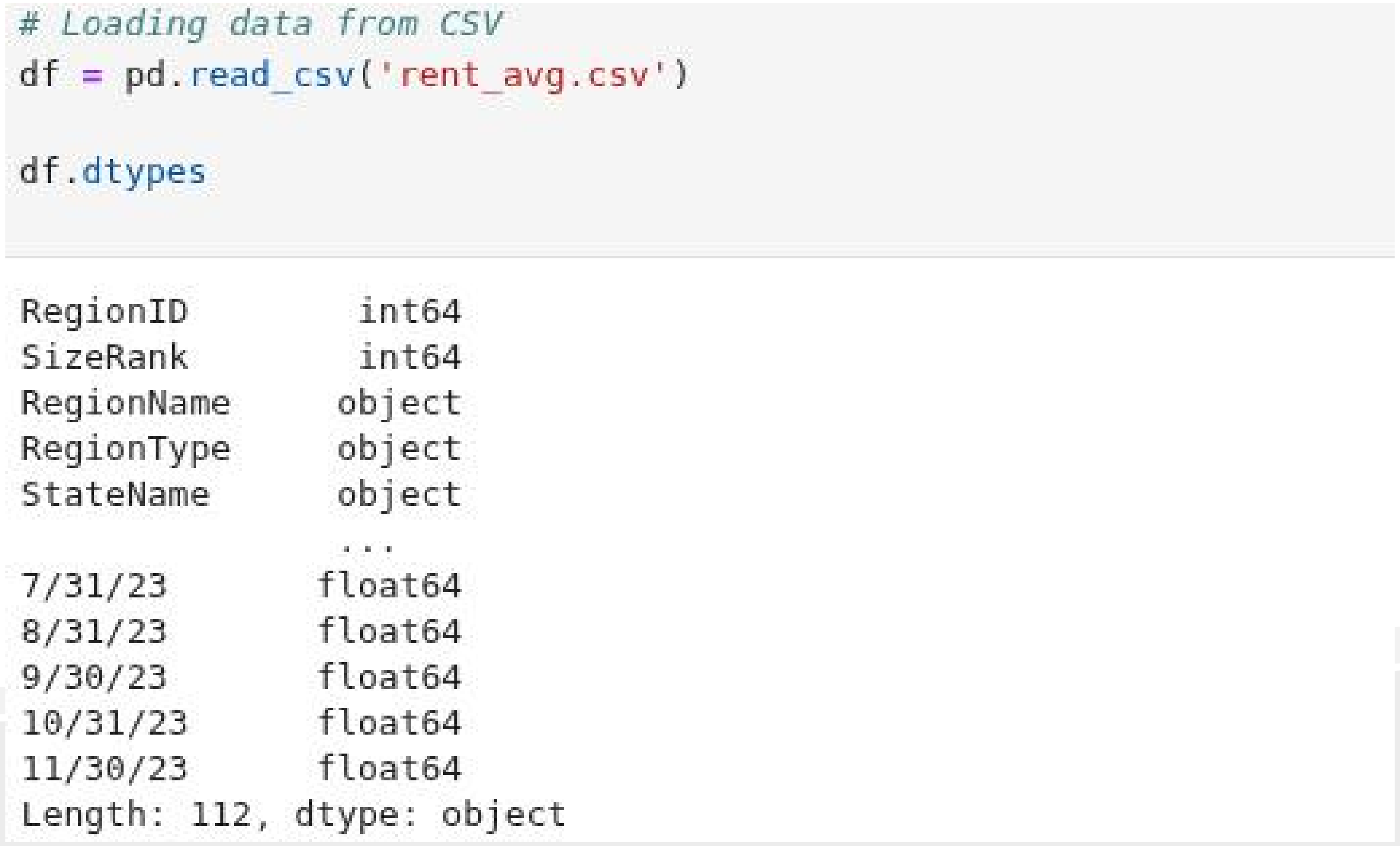
```
# Loading data from CSV
df = pd.read_csv('rent_avg.csv')

df.tail()
```

	RegionID	SizeRank	RegionName	RegionType	StateName	1/31/15	2/28/15	3/31/15	4/30/15
467	753871	811	Breckenridge, CO	msa	CO	NaN	NaN	NaN	NaN
468	394751	821	Kirksville, MO	msa	MO	NaN	NaN	NaN	NaN
469	753923	849	The Dalles, OR	msa	OR	NaN	NaN	NaN	NaN
470	394584	863	Fallon, NV	msa	NV	NaN	NaN	NaN	NaN
471	394996	915	Portales, NM	msa	NM	NaN	NaN	NaN	NaN

5 rows × 112 columns

Reading & Inspecting Data

- Inspecting data
  - Data types

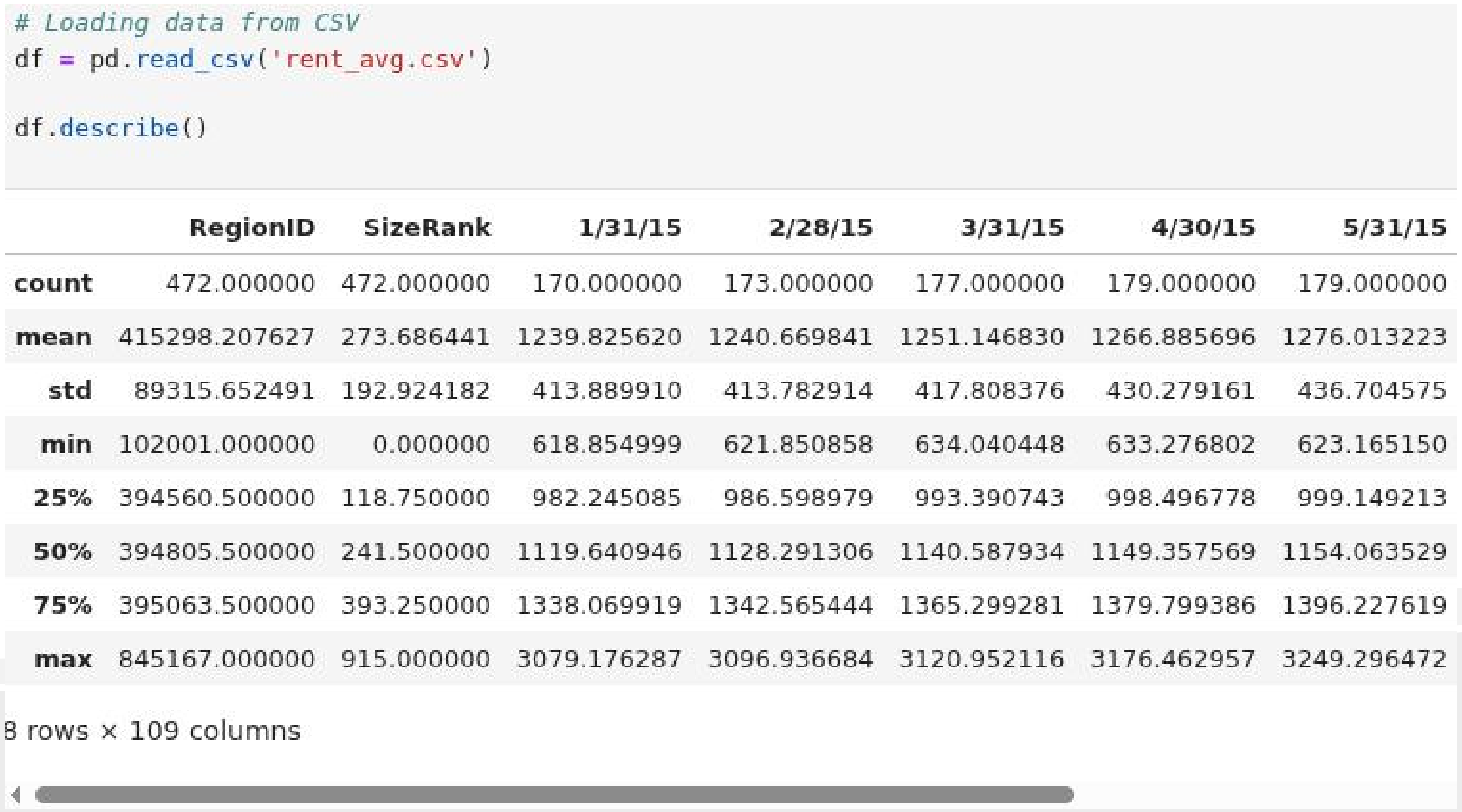


```
# Loading data from CSV
df = pd.read_csv('rent_avg.csv')

df.dtypes
```

```
RegionID        int64
SizeRank        int64
RegionName     object
RegionType     object
StateName      object
                ...
7/31/23       float64
8/31/23       float64
9/30/23       float64
10/31/23      float64
11/30/23      float64
Length: 112, dtype: object
```

Reading & Inspecting Data

- Inspecting data
  - Describe



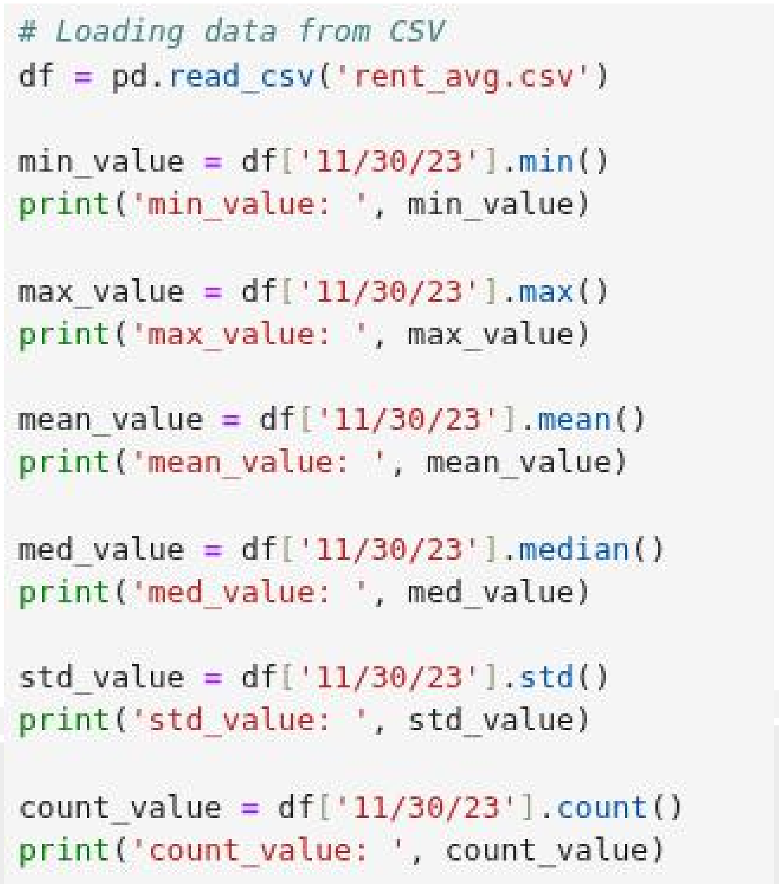
```python
# Loading data from CSV
df = pd.read_csv('rent_avg.csv')

df.describe()
```

	RegionID	SizeRank	1/31/15	2/28/15	3/31/15	4/30/15	5/31/15
count	472.000000	472.000000	170.000000	173.000000	177.000000	179.000000	179.000000
mean	415298.207627	273.686441	1239.825620	1240.669841	1251.146830	1266.885696	1276.013223
std	89315.652491	192.924182	413.889910	413.782914	417.808376	430.279161	436.704575
min	102001.000000	0.000000	618.854999	621.850858	634.040448	633.276802	623.165150
25%	394560.500000	118.750000	982.245085	986.598979	993.390743	998.496778	999.149213
50%	394805.500000	241.500000	1119.640946	1128.291306	1140.587934	1149.357569	1154.063529
75%	395063.500000	393.250000	1338.069919	1342.565444	1365.299281	1379.799386	1396.227619
max	845167.000000	915.000000	3079.176287	3096.936684	3120.952116	3176.462957	3249.296472

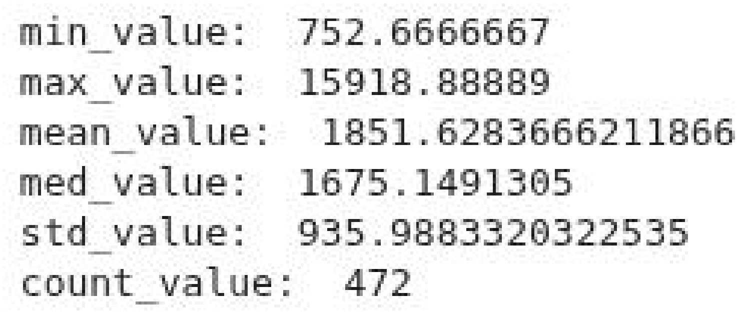
8 rows × 109 columns

Reading & Inspecting Data

- Inspecting data
  - Summary statistics



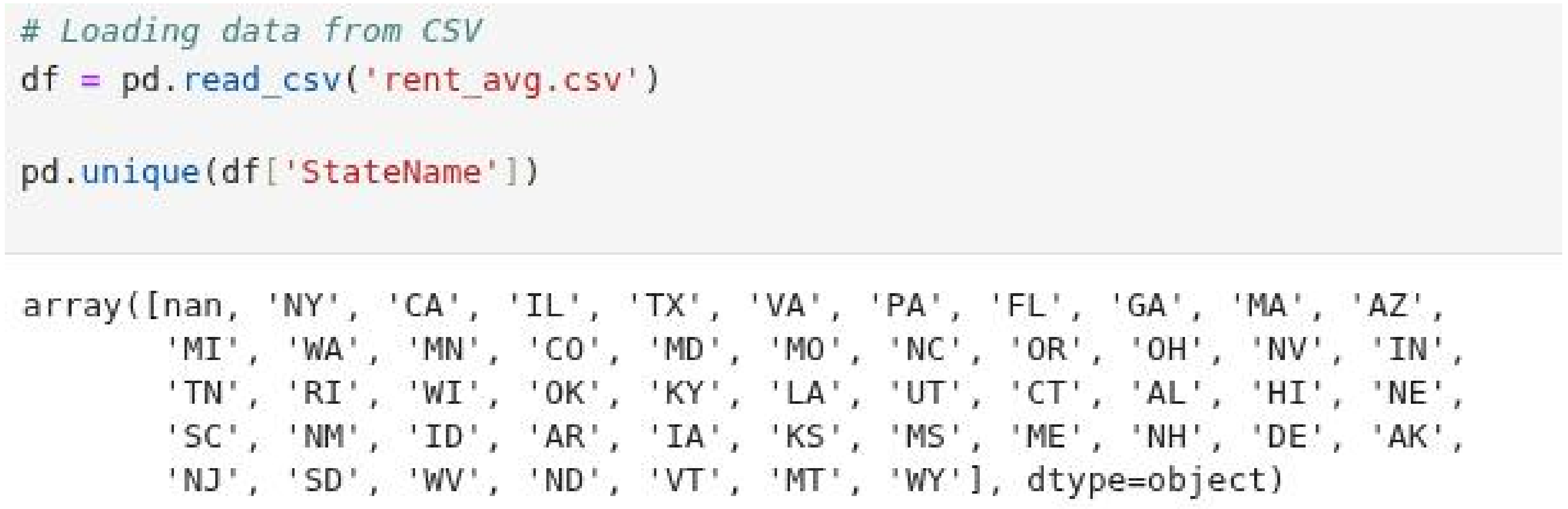
```python
# Loading data from CSV
df = pd.read_csv('rent_avg.csv')

min_value = df['11/30/23'].min()
print('min_value: ', min_value)

max_value = df['11/30/23'].max()
print('max_value: ', max_value)

mean_value = df['11/30/23'].mean()
print('mean_value: ', mean_value)

med_value = df['11/30/23'].median()
print('med_value: ', med_value)

std_value = df['11/30/23'].std()
print('std_value: ', std_value)

count_value = df['11/30/23'].count()
print('count_value: ', count_value)
```



```
min_value:  752.6666667
max_value:  15918.88889
mean_value:  1851.628366211866
med_value:  1675.1491305
std_value:  935.9883320322535
count_value:  472
```

Reading & Inspecting Data

- Inspecting data
  - Unique values

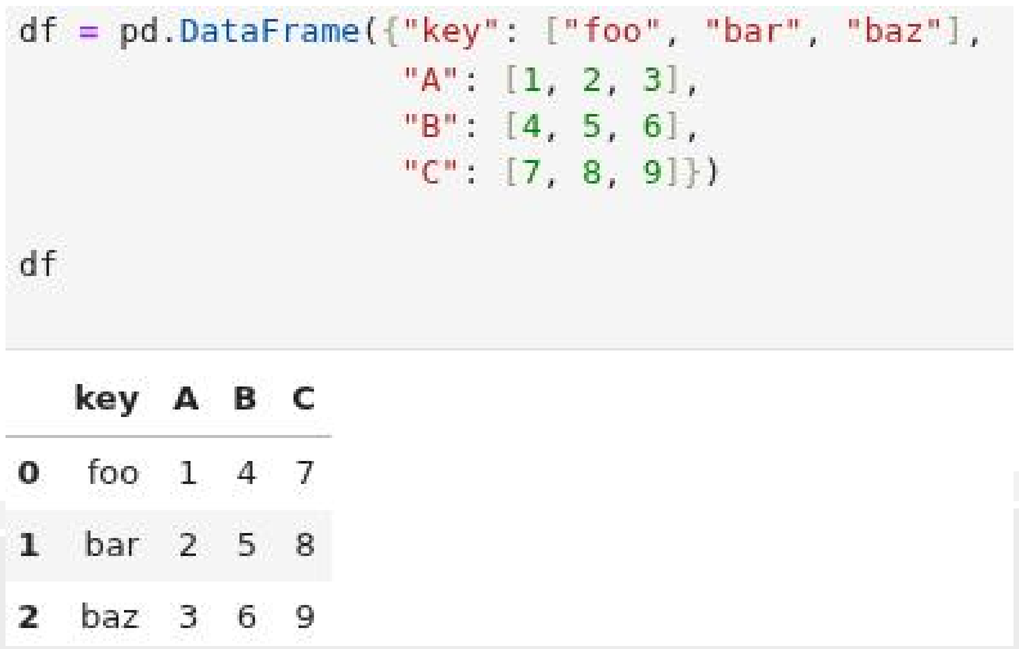


```
# Loading data from CSV
df = pd.read_csv('rent_avg.csv')

pd.unique(df['StateName'])
```

```
array([nan, 'NY', 'CA', 'IL', 'TX', 'VA', 'PA', 'FL', 'GA', 'MA', 'AZ',
       'MI', 'WA', 'MN', 'CO', 'MD', 'MO', 'NC', 'OR', 'OH', 'NV', 'IN',
       'TN', 'RI', 'WI', 'OK', 'KY', 'LA', 'UT', 'CT', 'AL', 'HI', 'NE',
       'SC', 'NM', 'ID', 'AR', 'IA', 'KS', 'MS', 'ME', 'NH', 'DE', 'AK',
       'NJ', 'SD', 'WV', 'ND', 'VT', 'MT', 'WY'], dtype=object)
```

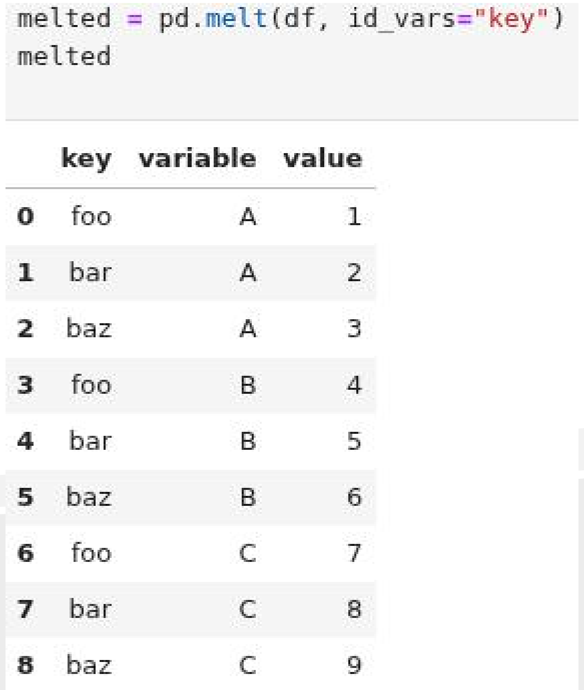
Melting

- Pandas melt() function is useful to massage a DataFrame into a format where one or more columns are identifier variables, while all other columns, considered measured variables, are unpivoted to the row axis, leaving just two non-identifier columns, variable and value.



```
df = pd.DataFrame({"key": ["foo", "bar", "baz"],
                   "A": [1, 2, 3],
                   "B": [4, 5, 6],
                   "C": [7, 8, 9]})

df
```

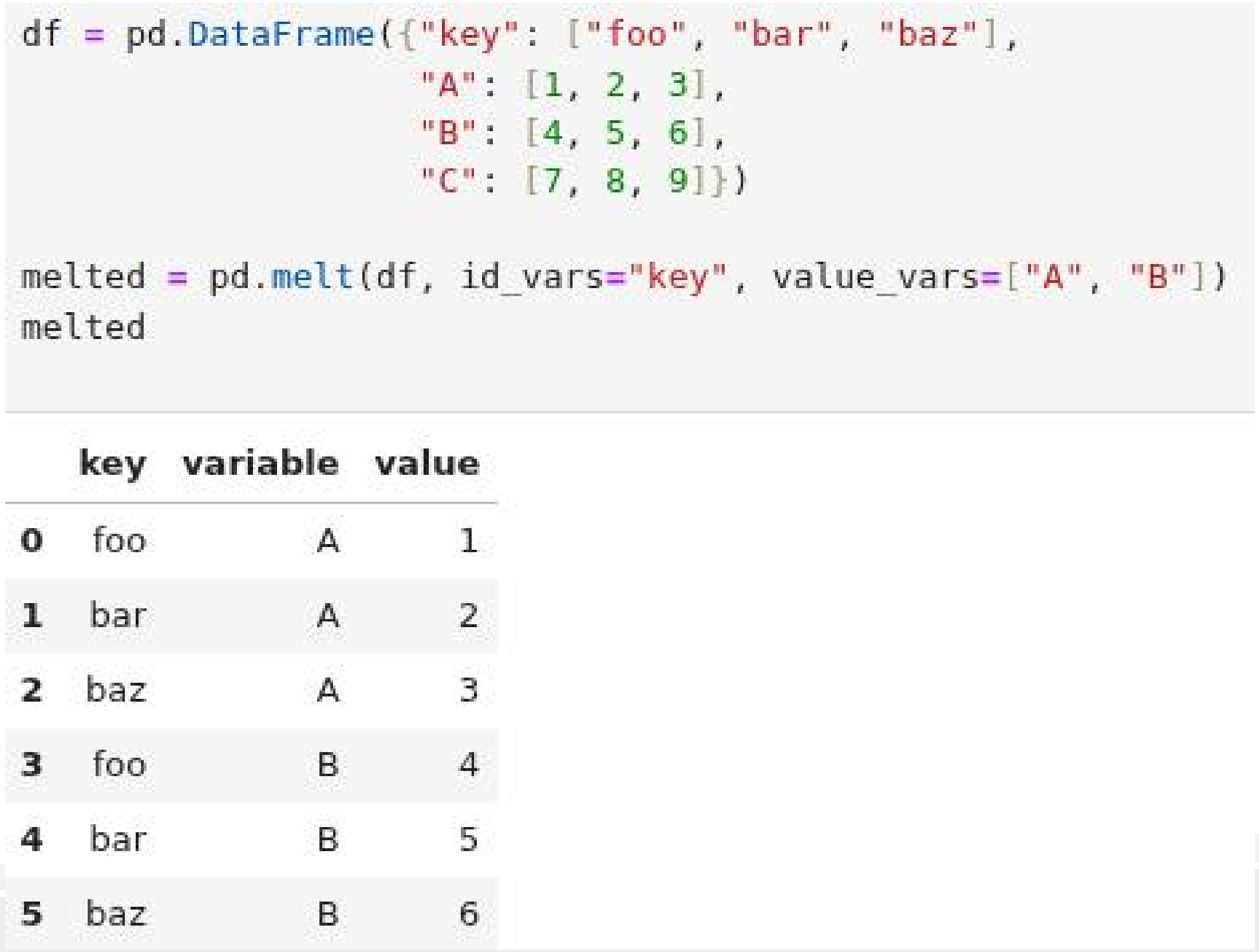
	key	A	B	C
0	foo	1	4	7
1	bar	2	5	8
2	baz	3	6	9



```
melted = pd.melt(df, id_vars="key")
melted
```

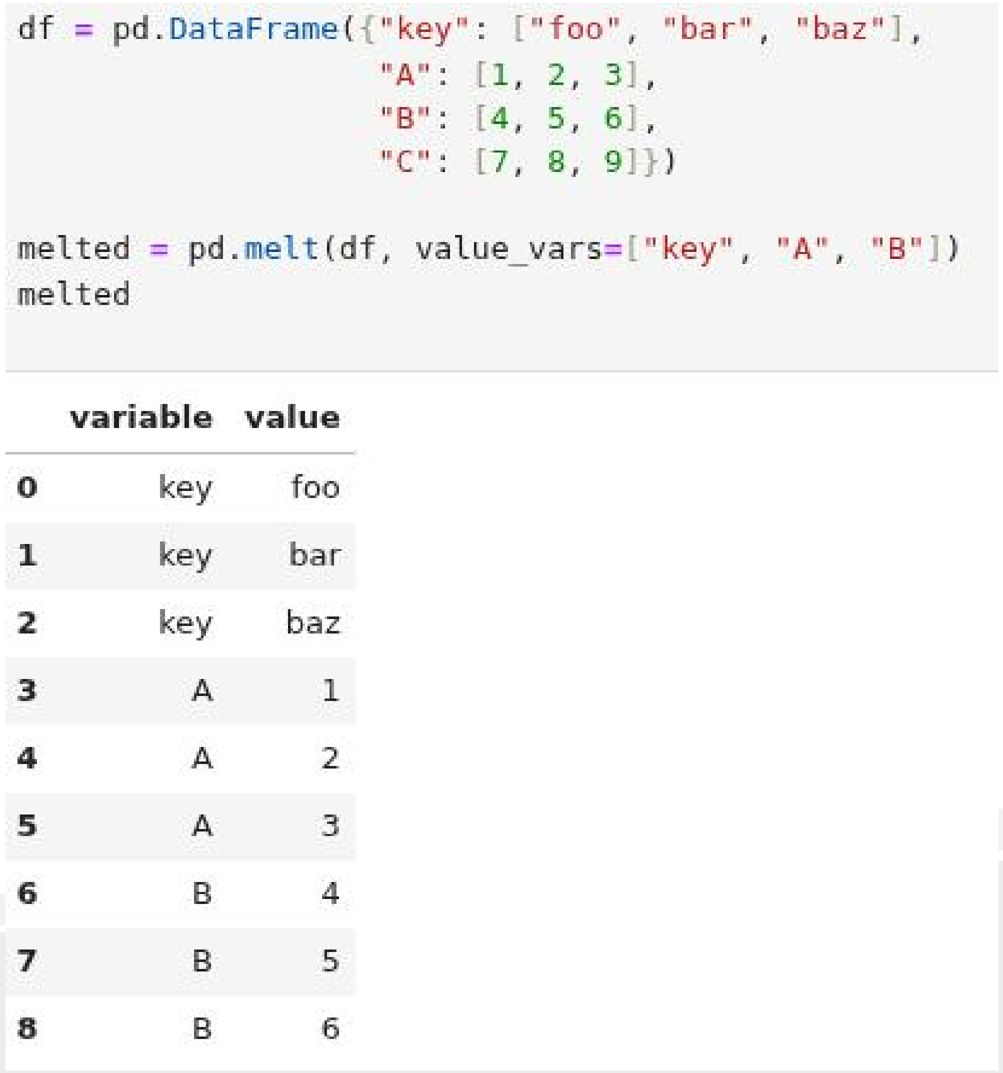
	key	variable	value
0	foo	A	1
1	bar	A	2
2	baz	A	3
3	foo	B	4
4	bar	B	5
5	baz	B	6
6	foo	C	7
7	bar	C	8
8	baz	C	9

Melting



```python
df = pd.DataFrame({"key": ["foo", "bar", "baz"],
                   "A": [1, 2, 3],
                   "B": [4, 5, 6],
                   "C": [7, 8, 9]})

melted = pd.melt(df, id_vars="key", value_vars=["A", "B"])
melted
```

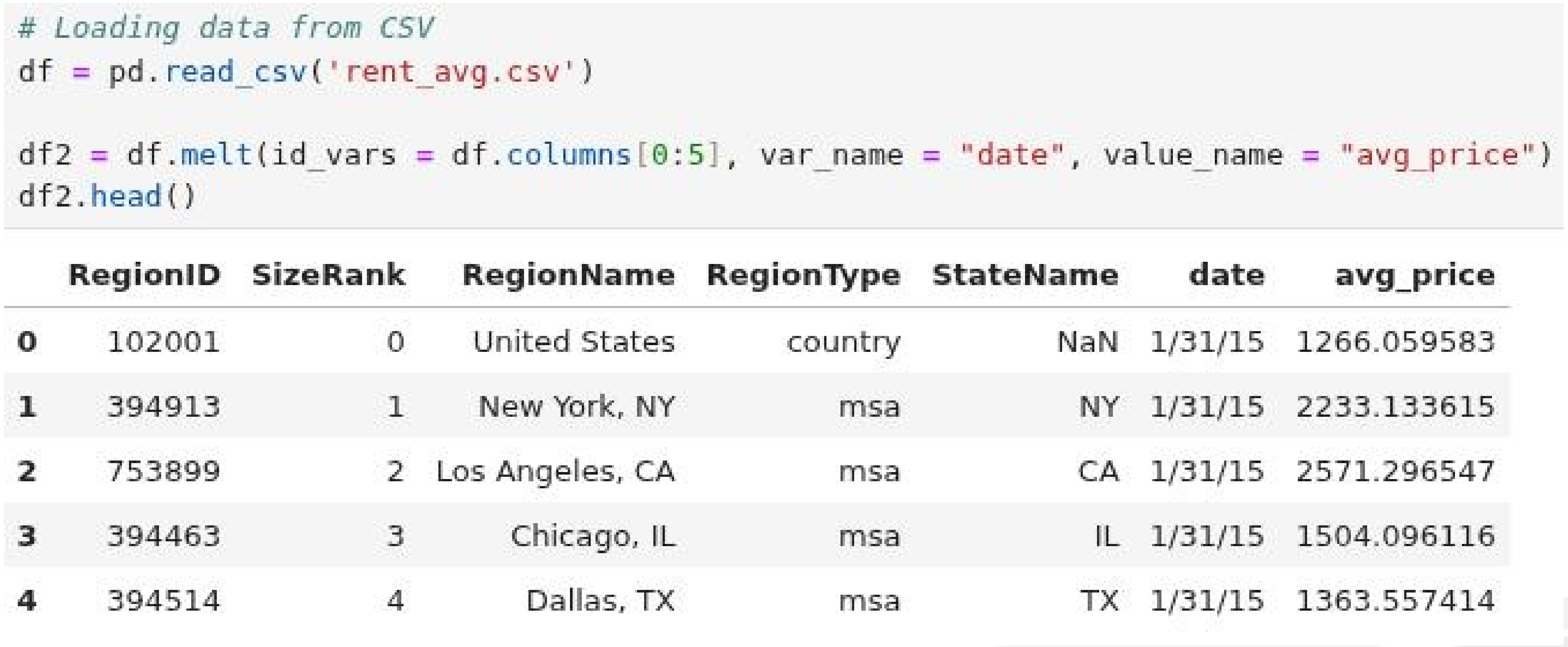
	key	variable	value
0	foo	A	1
1	bar	A	2
2	baz	A	3
3	foo	B	4
4	bar	B	5
5	baz	B	6



```python
df = pd.DataFrame({"key": ["foo", "bar", "baz"],
                   "A": [1, 2, 3],
                   "B": [4, 5, 6],
                   "C": [7, 8, 9]})

melted = pd.melt(df, value_vars=["key", "A", "B"])
melted
```

	variable	value
0	key	foo
1	key	bar
2	key	baz
3	A	1
4	A	2
5	A	3
6	B	4
7	B	5
8	B	6

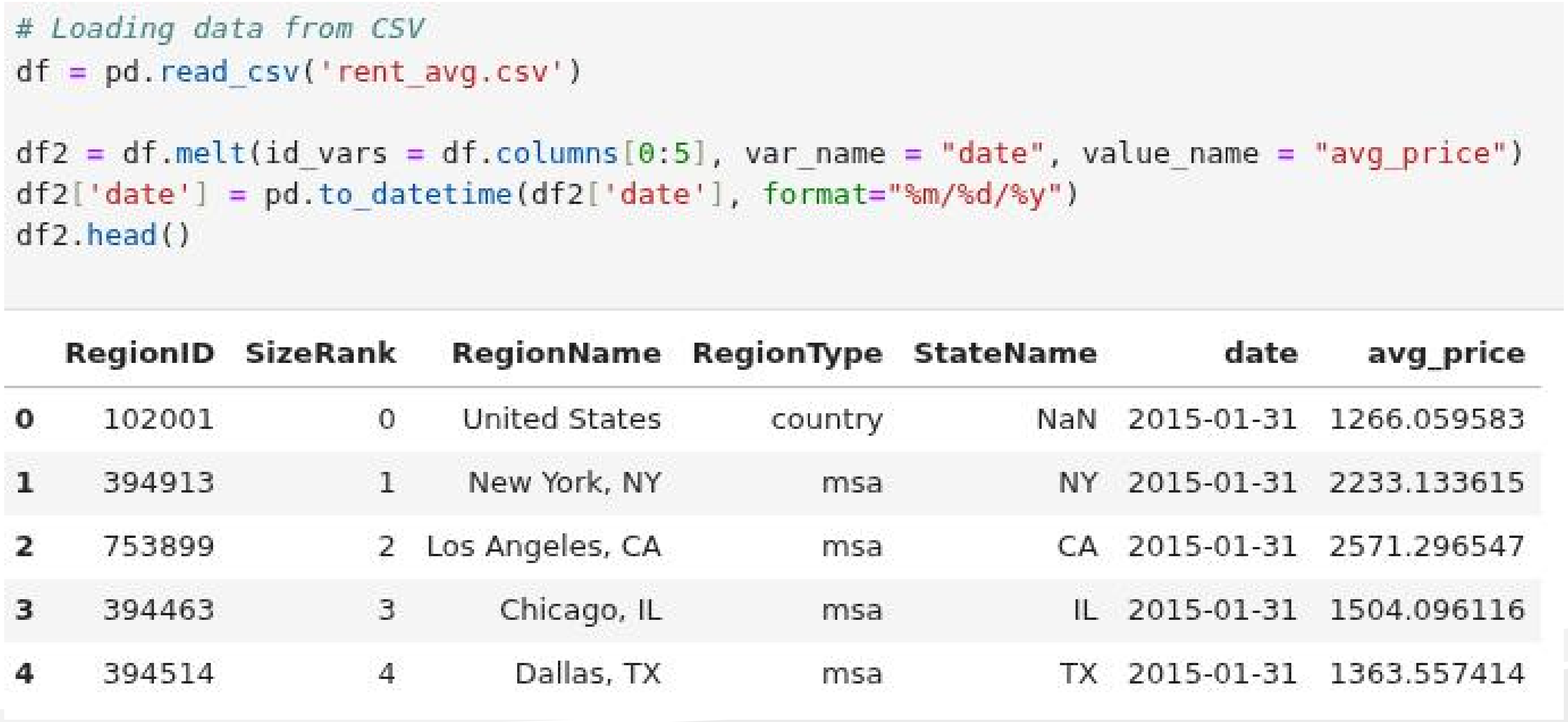
Melting



```
# Loading data from CSV
df = pd.read_csv('rent_avg.csv')

df2 = df.melt(id_vars = df.columns[0:5], var_name = "date", value_name = "avg_price")
df2.head()
```

	RegionID	SizeRank	RegionName	RegionType	StateName	date	avg_price
0	102001	0	United States	country	NaN	1/31/15	1266.059583
1	394913	1	New York, NY	msa	NY	1/31/15	2233.133615
2	753899	2	Los Angeles, CA	msa	CA	1/31/15	2571.296547
3	394463	3	Chicago, IL	msa	IL	1/31/15	1504.096116
4	394514	4	Dallas, TX	msa	TX	1/31/15	1363.557414

Convert to datetime



```
# Loading data from CSV
df = pd.read_csv('rent_avg.csv')

df2 = df.melt(id_vars = df.columns[0:5], var_name = "date", value_name = "avg_price")
df2['date'] = pd.to_datetime(df2['date'], format="%m/%d/%y")
df2.head()
```

	RegionID	SizeRank	RegionName	RegionType	StateName	date	avg_price
0	102001	0	United States	country	NaN	2015-01-31	1266.059583
1	394913	1	New York, NY	msa	NY	2015-01-31	2233.133615
2	753899	2	Los Angeles, CA	msa	CA	2015-01-31	2571.296547
3	394463	3	Chicago, IL	msa	IL	2015-01-31	1504.096116
4	394514	4	Dallas, TX	msa	TX	2015-01-31	1363.557414