# Data Cleaning and Preparation

Part 3

## Data Transformation

Part 2

#### Discretization and Binning

- Continuous data is often discretized or otherwise separated into "bins" for analysis.
- Suppose you have data about a group of people in a study, and you want to group them into discrete age buckets:

```
In [73]: ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

- Let's divide these into bins of 18 to 25, 26 to 35, 36 to 60, and finally 61 and older.
- To do so, you have to use cut, a function in pandas:

- The object pandas returns is a special Categorical object.
- The output you see describes the bins computed by pandas.cut.
- You can treat it like an array of strings indicating the bin name; internally it contains a categories array specifying the distinct category names along with a labeling for the ages data in the codes attribute:

- Consistent with mathematical notation for intervals, a parenthesis means that the side is open, while the square bracket means it is closed (inclusive).
- You can change which side is closed by passing right=False:

 You can also pass your own bin names by passing a list or array to the labels option:

- If you pass an integer number of bins to cut instead of explicit bin edges, it will compute equal-length bins based on the minimum and maximum values in the data.
- Consider the case of some uniformly distributed data chopped into fourths:

```
In [84]: pd.cut(data, 4, precision=2)
Out[84]: [(0.34, 0.55], (0.34, 0.55], (0.76, 0.97], (0.76, 0.97], (0.34, 0.55], ..., (0.34, 0.55], (0.34, 0.55], (0.55, 0.76], (0.34, 0.55], (0.12, 0.34]]
Length: 20
Categories (4, interval[float64]): [(0.12, 0.34] < (0.34, 0.55] < (0.55, 0.76] < (0.76, 0.97]]</pre>
```

- A closely related function, qcut, bins the data based on sample quantiles.
- Depending on the distribution of the data, using cut will not usually result in each bin having the same number of data points.
- Since qcut uses sample quantiles instead, by definition you will obtain roughly equal-size bins:

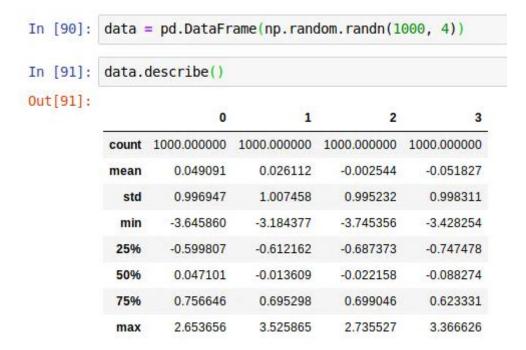
```
In [85]: data = np.random.randn(1000) # Normally distributed
In [86]: cats = pd.qcut(data, 4) # Cut into quartiles
In [87]: cats
Out[87]: [(-0.0265, 0.62], (0.62, 3.928], (-0.68, -0.0265], (0.62, 3.928], (-0.0265, 0.62], ..., (-0.68, -0.0265], (-0.68, -0.026
         5], (-2.94999999999997, -0.68], (0.62, 3.928], (-0.68, -0.0265]]
         Length: 1000
         Categories (4, interval[float64]): [(-2.94999999999997, -0.68] < (-0.68, -0.0265] < (-0.0265, 0.62] < (0.62, 3.928]]
In [88]: pd.value counts(cats)
Out[88]: (0.62, 3.928]
                                         250
         (-0.0265, 0.62]
                                         250
         (-0.68, -0.0265]
                                         250
         (-2.94999999999997, -0.68]
                                         250
         dtype: int64
```

• Similar to cut you can pass your own quantiles (numbers between 0 and 1, inclusive):

```
In [89]: pd.qcut(data, [0, 0.1, 0.5, 0.9, 1.])
Out[89]: [(-0.0265, 1.286], (-0.0265, 1.286], (-1.187, -0.0265], (-0.0265, 1.286], (-0.0265, 1.286], ..., (-1.187, -0.0265], (-1.187, -0.0265], (-2.94999999999997, -1.187], (-0.0265, 1.286], (-1.187, -0.0265]]
Length: 1000
Categories (4, interval[float64]): [(-2.94999999999997, -1.187] < (-1.187, -0.0265] < (-0.0265, 1.286] < (1.286, 3.928]]</pre>
```

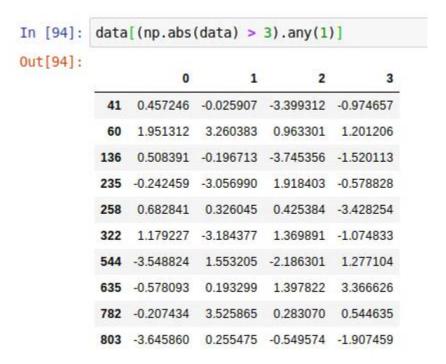
#### Detecting and Filtering Outliers

- Filtering or transforming outliers is largely a matter of applying array operations.
- Consider a DataFrame with some normally distributed data:

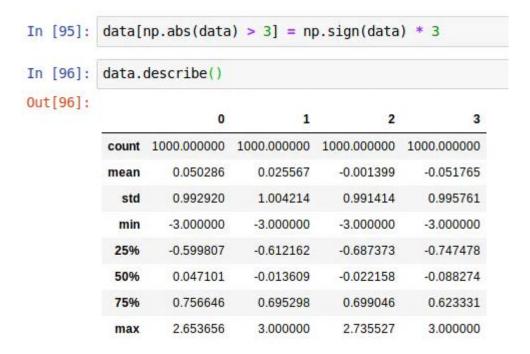


• Suppose you wanted to find values in one of the columns exceeding 3 in absolute value:

 To select all rows having a value exceeding 3 or −3, you can use the any method on a boolean DataFrame:



- Values can be set based on these criteria.
- Here is code to cap values outside the interval -3 to 3:



#### Permutation and Random Sampling

- Permuting (randomly reordering) a Series or the rows in a DataFrame is easy to do using the numpy.random.permutation function.
- Calling permutation with the length of the axis you want to permute produces an array of integers indicating the new ordering:

```
In [97]: df = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)))
In [98]: sampler = np.random.permutation(5)
In [99]: sampler
Out[99]: array([3, 1, 4, 2, 0])
```

• That array can then be used in iloc-based indexing or the equivalent take function:

```
In [100]: df
Out[100]:
          3 12 13 14 15
          4 16 17 18 19
In [101]: df.take(sampler)
Out[101]:
             0 1 2 3
         3 12 13 14 15
         0 0 1 2 3
```

• To select a random subset without replacement, you can use the sample method on Series and DataFrame:

• To generate a sample with replacement (to allow repeat choices), pass replace=True to sample:

### Computing Indicator/Dummy Variables

- Another type of transformation for statistical modeling or machine learning applications is converting a categorical variable into a "dummy" or "indicator" matrix.
- If a column in a DataFrame has  $\Bbbk$  distinct values, you would derive a matrix or DataFrame with  $\Bbbk$  columns containing all 1s and 0s.
- pandas has a get dummies function for doing this, though devising one yourself is not difficult.

- In some cases, you may want to add a prefix to the columns in the indicator DataFrame, which can then be merged with the other data.
- get dummies has a prefix argument for doing this:

- If a row in a DataFrame belongs to multiple categories, things are a bit more complicated.
- Let's look at the MovieLens 1M dataset.

geines	titue	movic_id	
Animation Children's Comedy	Toy Story (1995)	1	0
Adventure Children's Fantasy	Jumanji (1995)	2	1
Comedy Romance	Grumpier Old Men (1995)	3	2
Comedy Drama	Waiting to Exhale (1995)	4	3
Comedy	Father of the Bride Part II (1995)	5	4
Action Crime Thriller	Heat (1995)	6	5
Comedy Romance	Sabrina (1995)	7	6
Adventure Children's	Tom and Huck (1995)	8	7
Action	Sudden Death (1995)	9	8
Action Adventure Thriller	GoldenEye (1995)	10	9

- Adding indicator variables for each genre requires a little bit of wrangling.
- First, we extract the list of unique genres in the dataset:

 One way to construct the indicator DataFrame is to start with a DataFrame of all zeros:

200	<pre>dummies = pd.DataFrame(zero_matrix, columns=genres)</pre>																
333	dummies[:5]																
	Animation	Children's	Comedy	Adventure	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror	Sci- Fi	Documentary	War	Musical	Mystery	Film- Noir
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	2 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	3 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	4 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

- Now, iterate through each movie and set entries in each row of dummies to 1.
- To do this, we use the dummies.columns to compute the column indices for each genre:

```
In [121]: gen = movies.genres[0]
In [122]: gen.split('|')
Out[122]: ['Animation', "Children's", 'Comedy']
In [123]: dummies.columns.get_indexer(gen.split('|'))
Out[123]: array([0, 1, 2])
```

• Then, we can use .iloc to set values based on these indices:

```
In [124]: for i, gen in enumerate(movies.genres):
    indices = dummies.columns.get_indexer(gen.split('|'))
    dummies.iloc[i, indices] = 1
```

• Then, as before, you can combine this with movies:

```
In [125]: movies windic = movies.join(dummies.add prefix('Genre '))
In [126]: movies windic.iloc[0]
Out[126]: movie id
                                           Toy Story (1995)
          title
                                Animation|Children's|Comedy
          genres
          Genre Animation
          Genre Children's
          Genre Comedy
          Genre Adventure
          Genre Fantasy
          Genre Romance
          Genre Drama
                                           . . .
          Genre Crime
          Genre Thriller
          Genre Horror
          Genre Sci-Fi
          Genre Documentary
          Genre War
          Genre Musical
          Genre Mystery
          Genre Film-Noir
          Genre Western
          Name: 0, Length: 21, dtype: object
```

• A useful recipe for statistical applications is to combine get dummies with a discretization function like cut:

```
In [127]: np.random.seed(12345)
In [128]: values = np.random.rand(10)
In [129]: values
Out[129]: array([0.9296, 0.3164, 0.1839, 0.2046, 0.5677, 0.5955, 0.9645, 0.6532,
                 0.7489, 0.6536])
In [130]: bins = [0, 0.2, 0.4, 0.6, 0.8, 1]
In [131]: pd.get dummies(pd.cut(values, bins))
Out[131]:
             (0.0, 0.2] (0.2, 0.4] (0.4, 0.6] (0.6, 0.8]
                                                             (0.8, 1.0]
```