Data Cleaning and Preparation

In [1]:
```
import numpy as np
import pandas as pd
PREVIOUS_MAX_ROWS = pd.options.display.max_rows
pd.options.display.max_rows = 20
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
np.set_printoptions(precision=4, suppress=True)
```

Handling Missing Data

In [2]:
```
string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])
string_data
```

Out[2]:
```
0     aardvark
1    artichoke
2          NaN
3      avocado
dtype: object
```

In [3]:
```
string_data.isnull()
```

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

In [4]:
```
string_data[0] = None
string_data.isnull()
```

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

Filtering Out Missing Data

In [7]:
```
from numpy import nan as NA
data = pd.Series([1, NA, 3.5, NA, 7])
data
```

Out[7]:
```
0    1.0
1    NaN
2    3.5
3    NaN
4    7.0
dtype: float64
```

In [8]:
```
data.dropna()
```

Out[8]:
```
0    1.0
2    3.5
4    7.0
dtype: float64
```

In [9]:
```
data[data.notnull()]
```

Out[9]:
```
0    1.0
2    3.5
4    7.0
dtype: float64
```

In [10]:
```
data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],
                     [NA, NA, NA], [NA, 6.5, 3.]])
cleaned = data.dropna()
data
```

Out[10]:

	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

In [11]:
```
cleaned
```

Out[11]:

	0	1	2
0	1.0	6.5	3.0

In [12]:
```
data.dropna(how='all')
```

Out[12]:

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

In [13]:
```
data[4] = NA
data
```

Out[13]:

	0	1	2	4
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

In [14]:
```
data.dropna(axis=1, how='all')
```

Out[14]:

	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

In [15]:
```
df = pd.DataFrame(np.random.randn(7, 3))
df
```

Out[15]:

	0	1	2
0	-0.204708	0.478943	-0.519439
1	-0.555730	1.965781	1.393406
2	0.092908	0.281746	0.769023
3	1.246435	1.007189	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

In [17]:
```
df.iloc[:4, 1] = NA
df
```

Out[17]:

	0	1	2
0	-0.204708	NaN	-0.519439
1	-0.555730	NaN	1.393406
2	0.092908	NaN	0.769023
3	1.246435	NaN	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

In [18]:
```
df.iloc[:2, 2] = NA
df
```

Out[18]:

	0	1	2
0	-0.204708	NaN	NaN
1	-0.555730	NaN	NaN
2	0.092908	NaN	0.769023
3	1.246435	NaN	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

In [19]:
```
df.dropna()
```

Out[19]:

	0	1	2
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

In [20]:
```
df.dropna(thresh=2)
```

Out[20]:

	0	1	2
2	0.092908	NaN	0.769023
3	1.246435	NaN	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

Filling In Missing Data

In [21]:
```
df.fillna(0)
```

Out[21]:

	0	1	2
0	-0.204708	0.000000	0.000000
1	-0.555730	0.000000	0.000000
2	0.092908	0.000000	0.769023
3	1.246435	0.000000	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

In [22]:
```
df.fillna({1: 0.5, 2: 0})
```

Out[22]:

	0	1	2
0	-0.204708	0.500000	0.000000
1	-0.555730	0.500000	0.000000
2	0.092908	0.500000	0.769023
3	1.246435	0.500000	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

In [23]:
```
_ = df.fillna(0, inplace=True)
df
```

Out[23]:

	0	1	2
0	-0.204708	0.000000	0.000000
1	-0.555730	0.000000	0.000000
2	0.092908	0.000000	0.769023
3	1.246435	0.000000	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

In [24]:
```
df = pd.DataFrame(np.random.randn(6, 3))
df.iloc[2:, 1] = NA
df.iloc[4:, 2] = NA
df
```

Out[24]:

	0	1	2
0	0.476985	3.248944	-1.021228
1	-0.577087	0.124121	0.302614
2	0.523772	NaN	1.343810
3	-0.713544	NaN	-2.370232
4	-1.860761	NaN	NaN
5	-1.265934	NaN	NaN

In [25]:
```
df.fillna(method='ffill')
```

Out[25]:

	0	1	2
0	0.476985	3.248944	-1.021228
1	-0.577087	0.124121	0.302614
2	0.523772	0.124121	1.343810
3	-0.713544	0.124121	-2.370232
4	-1.860761	0.124121	-2.370232
5	-1.265934	0.124121	-2.370232

In [26]:
```
df.fillna(method='ffill', limit=2)
```

Out[26]:

	0	1	2
0	0.476985	3.248944	-1.021228
1	-0.577087	0.124121	0.302614
2	0.523772	0.124121	1.343810
3	-0.713544	0.124121	-2.370232
4	-1.860761	NaN	-2.370232
5	-1.265934	NaN	-2.370232

In [27]:
```
data = pd.Series([1., NA, 3.5, NA, 7])
data.fillna(data.mean())
```

Out[27]:
```
0    1.000000
1    3.833333
2    3.500000
3    3.833333
4    7.000000
dtype: float64
```

Data Transformation

Removing Duplicates

In [28]:
```
data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
                     'k2': [1, 1, 2, 3, 3, 4, 4]})
data
```

Out[28]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

In [29]:
```
data.duplicated()
```

Out[29]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

In [30]:
```
data.drop_duplicates()
```

Out[30]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

In [31]:
```
data['v1'] = range(7)
data
```

Out[31]:

	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
5	two	4	5
6	two	4	6

In [32]:
```
data.drop_duplicates(['k1'])
```

Out[32]:

	k1	k2	v1
0	one	1	0
1	two	1	1

In [33]:
```
data.drop_duplicates(['k1', 'k2'], keep='last')
```

Out[33]:

	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
6	two	4	6

Transforming Data Using a Function or Mapping

In [34]:
```
data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',
                              'Pastrami', 'corned beef', 'Bacon',
                              'pastrami', 'honey ham', 'nova lox'],
                     'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
data
```

Out[34]:

	food	ounces
0	bacon	4.0
1	pulled pork	3.0
2	bacon	12.0
3	Pastrami	6.0
4	corned beef	7.5
5	Bacon	8.0
6	pastrami	3.0
7	honey ham	5.0
8	nova lox	6.0

In [35]:
```
meat_to_animal = {
  'bacon': 'pig',
  'pulled pork': 'pig',
  'pastrami': 'cow',
  'corned beef': 'cow',
  'honey ham': 'pig',
  'nova lox': 'salmon'
}
```

In [36]:
```
lowercased = data['food'].str.lower()
lowercased
```

Out[36]:
```
0          bacon
1    pulled pork
2          bacon
3       pastrami
4    corned beef
5          bacon
6       pastrami
7      honey ham
8       nova lox
Name: food, dtype: object
```

In [37]:
```
data['animal'] = lowercased.map(meat_to_animal)
data
```

Out[37]:

	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	Pastrami	6.0	cow
4	corned beef	7.5	cow
5	Bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

In [38]:
```
data['food'].map(lambda x: meat_to_animal[x.lower()])
```

Out[38]:
```
0       pig
1       pig
2       pig
3       cow
4       cow
5       pig
6       cow
7       pig
8    salmon
Name: food, dtype: object
```

Replacing Values

In [39]:
```
data = pd.Series([1., -999., 2., -999., -1000., 3.])
data
```

Out[39]:
```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

In [40]:
```
data.replace(-999, np.nan)
```

Out[40]:
```
0       1.0
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
```

In [41]:
```
data.replace([-999, -1000], np.nan)
```

Out[41]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

In [42]:
```
data.replace([-999, -1000], [np.nan, 0])
```

Out[42]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

In [43]:
```
data.replace({-999: np.nan, -1000: 0})
```

Out[43]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

Renaming Axis Indexes

In [45]:
```
data = pd.DataFrame(np.arange(12).reshape((3, 4)),
                    index=['Ohio', 'Colorado', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
data
```

Out[45]:

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

In [46]:
```
transform = lambda x: x[:4].upper()
data.index.map(transform)
```

Out[46]:
```
Index(['OHIO', 'COLO', 'NEW '], dtype='object')
```

In [47]:
```
data.index = data.index.map(transform)
data
```

Out[47]:

	one	two	three	four
OHIO	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

In [48]:
```
data.rename(index=str.title, columns=str.upper)
```

Out[48]:

	ONE	TWO	THREE	FOUR
Ohio	0	1	2	3
Colo	4	5	6	7
New	8	9	10	11

In [49]:
```
data.rename(index={'OHIO': 'INDIANA'},
            columns={'three': 'peekaboo'})
```

Out[49]:

	one	two	peekaboo	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

In [50]:
```
data.rename(index={'OHIO': 'INDIANA'}, inplace=True)
data
```

Out[50]:

	one	two	three	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

Discretization and Binning

In [51]:
```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

In [52]:
```
bins = [18, 25, 35, 60, 100]
cats = pd.cut(ages, bins)
cats
```

Out[52]:
```
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35,
60], (35, 60], (25, 35]]
Length: 12
Categories (4, interval[int64, right]): [(18, 25] < (25, 35] < (35, 60] < (60, 10
0]]
```

In [53]:
```
cats.codes
```

Out[53]:
```
array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)
```

In [54]:
```
cats.categories
```

Out[54]:
```
IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]], dtype='interval[int64, ri
ght]')
```

In [55]:
```
pd.value_counts(cats)
```

Out[55]:
```
(18, 25]     5
(25, 35]     3
(35, 60]     3
(60, 100]    1
dtype: int64
```

In [56]:
```
pd.cut(ages, [18, 26, 36, 61, 100], right=False)
```

Out[56]:
```
[[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ..., [26, 36), [61, 100), [36,
61), [36, 61), [26, 36)]
Length: 12
Categories (4, interval[int64, left]): [[18, 26) < [26, 36) < [36, 61) < [61, 10
0)]
```

In [57]:
```
group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
pd.cut(ages, bins, labels=group_names)
```

Out[57]:
```
['Youth', 'Youth', 'Youth', 'YoungAdult', 'Youth', ..., 'YoungAdult', 'Senior', 'M
iddleAged', 'MiddleAged', 'YoungAdult']
Length: 12
Categories (4, object): ['Youth' < 'YoungAdult' < 'MiddleAged' < 'Senior']
```

In [58]:
```
data = np.random.rand(20)
pd.cut(data, 4, precision=2)
```

Out[58]:
```
[(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, right]): [(0.12, 0.34] < (0.34, 0.55] < (0.55, 0.
76] < (0.76, 0.97]]
```

In [59]:
```
data
```

Out[59]:
```
array([0.4896, 0.3773, 0.8486, 0.9111, 0.3838, 0.3155, 0.5684, 0.1878,
       0.1258, 0.6876, 0.7996, 0.5735, 0.9732, 0.6341, 0.8884, 0.4954,
       0.3516, 0.7142, 0.5039, 0.2256])
```

In [60]:
```
data = np.random.randn(1000)  # Normally distributed
cats = pd.qcut(data, 4)  # Cut into quartiles
cats
```

Out[60]:
```
[(-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, -0.0265], (-2.9499999999999997, -0.68], (0.62, 3.92
8], (-0.68, -0.0265]]
Length: 1000
Categories (4, interval[float64, right]): [(-2.9499999999999997, -0.68] < (-0.68,
-0.0265] < (-0.0265, 0.62] < (0.62, 3.928]]
```

In [61]:
```
pd.value_counts(cats)
```

Out[61]:
```
(-2.9499999999999997, -0.68]    250
(-0.68, -0.0265]                250
(-0.0265, 0.62]                 250
(0.62, 3.928]                   250
dtype: int64
```

In [62]:
```
pd.qcut(data, [0, 0.1, 0.5, 0.9, 1.])
```

Out[62]:
```
[(-0.0265, 1.286], (-0.0265, 1.286], (-1.187, -0.0265], (-0.0265, 1.286], (-0.026
5, 1.286], ..., (-1.187, -0.0265], (-1.187, -0.0265], (-2.9499999999999997, -1.18
7], (-0.0265, 1.286], (-1.187, -0.0265]]
Length: 1000
Categories (4, interval[float64, right]): [(-2.9499999999999997, -1.187] < (-1.18
7, -0.0265] < (-0.0265, 1.286] < (1.286, 3.928]]
```

Detecting and Filtering Outliers

In [63]:
```
data = pd.DataFrame(np.random.randn(1000, 4))
data
```

Out[63]:

	0	1	2	3
0	-0.799318	0.777233	-0.612905	0.316447
1	0.838295	-1.034423	0.434304	-2.213133
2	0.758040	0.553933	0.339231	-0.688756
3	-0.815526	-0.332420	2.406483	-1.361428
4	-0.669619	0.781199	-0.395813	-0.180737
...	...	...	...	...
995	-0.856979	-0.446678	1.229042	-1.558031
996	-0.289339	-0.232531	0.409304	-0.813190
997	0.023646	0.232781	-0.345727	1.519174
998	1.060646	-1.456358	1.128420	0.032166
999	-0.485783	1.181563	-2.314042	-0.865834

1000 rows × 4 columns

In [64]:
```
data.describe()
```

Out[64]:

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.049091	0.026112	-0.002544	-0.051827
std	0.996947	1.007458	0.995232	0.998311
min	-3.645860	-3.184377	-3.745356	-3.428254
25%	-0.599807	-0.612162	-0.687373	-0.747478
50%	0.047101	-0.013609	-0.022158	-0.088274
75%	0.756646	0.695298	0.699046	0.623331
max	2.653656	3.525865	2.735527	3.366626

In [65]:
```
col = data[2]
col[np.abs(col) > 3]
```

Out[65]:
```
41    -3.399312
136   -3.745356
Name: 2, dtype: float64
```

In [66]:
```
data[(np.abs(data) > 3).any(1)]
```

Out[66]:

	0	1	2	3
41	0.457246	-0.025907	-3.399312	-0.974657
60	1.951312	3.260383	0.963301	1.201206
136	0.508391	-0.196713	-3.745356	-1.520113
235	-0.242459	-3.056990	1.918403	-0.578828
258	0.682841	0.326045	0.425384	-3.428254
322	1.179227	-3.184377	1.369891	-1.074833
544	-3.548824	1.553205	-2.186301	1.277104
635	-0.578093	0.193299	1.397822	3.366626
782	-0.207434	3.525865	0.283070	0.544635
803	-3.645860	0.255475	-0.549574	-1.907459

In [68]:
```
data[np.abs(data) > 3] = np.sign(data) * 3
data.describe()
```

Out[68]:

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.050286	0.025567	-0.001399	-0.051765
std	0.992920	1.004214	0.991414	0.995761
min	-3.000000	-3.000000	-3.000000	-3.000000
25%	-0.599807	-0.612162	-0.687373	-0.747478
50%	0.047101	-0.013609	-0.022158	-0.088274
75%	0.756646	0.695298	0.699046	0.623331
max	2.653656	3.000000	2.735527	3.000000

In [67]:
```
np.sign(data).head()
```

Out[67]:

	0	1	2	3
0	-1.0	1.0	-1.0	1.0
1	1.0	-1.0	1.0	-1.0
2	1.0	1.0	1.0	-1.0
3	-1.0	-1.0	1.0	-1.0
4	-1.0	1.0	-1.0	-1.0

Permutation and Random Sampling

In [69]:
```
df = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)))
df
```

Out[69]:

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

In [70]:
```
sampler = np.random.permutation(5)
sampler
```

Out[70]:
```
array([3, 1, 4, 2, 0])
```

In [71]:
```
df
df.take(sampler)
```

Out[71]:

	0	1	2	3
3	12	13	14	15
1	4	5	6	7
4	16	17	18	19
2	8	9	10	11
0	0	1	2	3

In [72]:
```
df.sample(n=3)
```

Out[72]:

	0	1	2	3
3	12	13	14	15
4	16	17	18	19
2	8	9	10	11

In [73]:
```
choices = pd.Series([5, 7, -1, 6, 4])
draws = choices.sample(n=10, replace=True)
draws
```

Out[73]:
```
4    4
1    7
4    4
2   -1
0    5
3    6
1    7
4    4
0    5
4    4
dtype: int64
```

Computing Indicator/Dummy Variables

In [74]:
```
df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                   'data1': range(6)})
```

```
df
```

Out[74]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	b	5

In [75]:
```
pd.get_dummies(df['key'])
```

Out[75]:

	a	b	c
0	0	1	0
1	0	1	0
2	1	0	0
3	0	0	1
4	1	0	0
5	0	1	0

In [76]:
```
dummies = pd.get_dummies(df['key'], prefix='key')
df_with_dummy = df[['data1']].join(dummies)
df_with_dummy
```

Out[76]:

	data1	key_a	key_b	key_c
0	0	0	1	0
1	1	0	1	0
2	2	1	0	0
3	3	0	0	1
4	4	1	0	0
5	5	0	1	0

In [78]:
```
mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('datasets/movielens/movies.dat', sep='::',
                       header=None, names=mnames, engine='python')
movies[:10]
```

Out[78]:

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

In [80]:
```
all_genres = []
for x in movies.genres:
    all_genres.extend(x.split('|'))
all_genres[0:10]
```

Out[80]:
```
['Animation',
 "Children's",
 'Comedy',
 'Adventure',
 "Children's",
 'Fantasy',
 'Comedy',
 'Romance',
 'Comedy',
 'Drama']
```

In [81]:
```
genres = pd.unique(all_genres)
```

In [82]:
```
genres
```

Out[82]:
```
array(['Animation', "Children's", 'Comedy', 'Adventure', 'Fantasy',
       'Romance', 'Drama', 'Action', 'Crime', 'Thriller', 'Horror',
       'Sci-Fi', 'Documentary', 'War', 'Musical', 'Mystery', 'Film-Noir',
       'Western'], dtype=object)
```

In [85]:
```
zero_matrix = np.zeros((len(movies), len(genres)))
zero_matrix[:5]
```

Out[85]:
```
array([[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., 0., 0.,
        0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0.]])
```

In [86]:
```
dummies = pd.DataFrame(zero_matrix, columns=genres)
dummies[:5]
```

Out[86]:

	Animation	Children's	Comedy	Adventure	Fantasy	Romance	Drama	Action	Crime	Thriller
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	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
3	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

In [87]:
```
gen = movies.genres[0]
```

In [88]:
```
gen.split('|')
```

Out[88]:
```
['Animation', "Children's", 'Comedy']
```

In [89]:
```
dummies.columns.get_indexer(gen.split('|'))
```

Out[89]:
```
array([0, 1, 2], dtype=int64)
```

In [90]:
```
for i, gen in enumerate(movies.genres):
    indices = dummies.columns.get_indexer(gen.split('|'))
    dummies.iloc[i, indices] = 1
```

In [91]:
```
movies_windic = movies.join(dummies.add_prefix('Genre_'))
movies_windic.iloc[0]
```

Out[91]:
```
movie_id                                    1
title                        Toy Story (1995)
genres             Animation|Children's|Comedy
Genre_Animation                            1.0
Genre_Children's                           1.0
                                          ...
Genre_War                                  0.0
Genre_Musical                              0.0
Genre_Mystery                              0.0
Genre_Film-Noir                            0.0
Genre_Western                              0.0
Name: 0, Length: 21, dtype: object
```

In [92]:
```
np.random.seed(12345)
values = np.random.rand(10)
values
```

Out[92]:
```
array([0.9296, 0.3164, 0.1839, 0.2046, 0.5677, 0.5955, 0.9645, 0.6532,
       0.7489, 0.6536])
```

In [93]:
```
bins = [0, 0.2, 0.4, 0.6, 0.8, 1]
pd.get_dummies(pd.cut(values, bins))
```

Out[93]:

	(0.0, 0.2]	(0.2, 0.4]	(0.4, 0.6]	(0.6, 0.8]	(0.8, 1.0]
0	0	0	0	0	1
1	0	1	0	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	0	1	0	0
6	0	0	0	0	1
7	0	0	0	1	0
8	0	0	0	1	0
9	0	0	0	1	0

String Manipulation

String Object Methods

In [94]:
```
val = 'a,b,  guido'
val.split(',')
```

Out[94]:
```
['a', 'b', '  guido']
```

In [95]:
```
pieces = [x.strip() for x in val.split(',')]
pieces
```

Out[95]:
```
['a', 'b', 'guido']
```

In [96]:
```
first, second, third = pieces
first + '::' + second + '::' + third
```

Out[96]:
```
'a::b::guido'
```

In [97]:
```
'::'.join(pieces)
```

Out[97]:
```
'a::b::guido'
```

In [98]:
```
'guido' in val
```

Out[98]:
```
True
```

In [99]:
```
val.index(',')
```

Out[99]:
```
1
```

In [100…
```
val.find(':')
```

Out[100]:
```
-1
```

In [101…
```
val.index(':')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_15416\2927268062.py in <module>
----> 1 val.index(':')

ValueError: substring not found
```

In [102…
```
val.count(',')
```

Out[102]:
```
2
```

In [103…
```
val.replace(',', '::')
```

Out[103]:
```
'a::b::  guido'
```

In [105…
```
val.replace(',', '')
```

Out[105]:
```
'ab  guido'
```

In [106…
```
val.endswith('o')
```

Out[106]:
```
True
```

In [107…
```
val.endswith('r')
```

Out[107]:
```
False
```

In [108…
```
val.startswith('a')
```

Out[108]:
```
True
```

In [109…
```
val.startswith('b')
```

Out[109]:
```
False
```

In [113…
```
val.join('111')
```

Out[113]:
```
'1a,b,  guido1a,b,  guido1'
```

In [114…
```
val.index('g') #devolve erro se não encontrar
```

Out[114]:
```
6
```

In [115…
```
val.find('n') #devolve -1 se não encontrar
```

Out[115]:
```
-1
```

In [116…
```
val.rfind(',') #devolve -1 se não encontrar
```

Out[116]:
```
3
```

In [117…
```
val.replace(',',';')
```

Out[117]:
```
'a;b;  guido'
```

In [122…
```
val.strip()
```

Out[122]:
```
'a,b,  guido'
```

In [124…
```
value = ' abcd '
value.strip()
```

Out[124]:
```
'abcd'
```

In [125…
```
value.rstrip()
```

Out[125]:
```
' abcd'
```

In [126…
```
value.lstrip()
```

Out[126]:
```
'abcd '
```

In [127…
```
value.upper()
```

Out[127]:
```
' ABCD '
```

In [128…
```
value.upper().lower()
```

Out[128]:
```
' abcd '
```

In [129…
```
value.casefold()
```

Out[129]:
```
' abcd '
```

In [132…
```
value.ljust(10,'-')
```

Out[132]:
```
' abcd ----'
```

In [133…
```
value.rjust(10,'-')
```

Out[133]:
```
'---- abcd '
```

Regular Expressions

In [134…
```
import re
text = "foo    bar\t baz  \tqux"
text
```

Out[134]:
```
'foo    bar\t baz  \tqux'
```

In [135…
```
re.split('\s+', text)
```

Out[135]:
```
['foo', 'bar', 'baz', 'qux']
```

In [137…
```
regex = re.compile('\s+')
regex.split(text)
```

Out[137]:
```
['foo', 'bar', 'baz', 'qux']
```

In [138…
```
regex.findall(text)
```

Out[138]:
```
['    ', '\t ', '  \t']
```

In [139…
```
text = """Dave dave@google.com
Steve steve@gmail.com
Rob rob@gmail.com
Ryan ryan@yahoo.com
"""
pattern = r'[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}'

# re.IGNORECASE makes the regex case-insensitive
regex = re.compile(pattern, flags=re.IGNORECASE)
```

In [140…
```
regex.findall(text)
```

Out[140]:
```
['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.com']
```

In [141…
```
m = regex.search(text)
m
```

Out[141]:
```
<re.Match object; span=(5, 20), match='dave@google.com'>
```

In [142…
```
text[m.start():m.end()]
```

Out[142]:
```
'dave@google.com'
```

In [143…
```
print(regex.match(text))
```

```
None
```

In [144…
```
print(regex.sub('REDACTED', text))
```

```
Dave REDACTED
Steve REDACTED
Rob REDACTED
Ryan REDACTED
```

In [145…
```
pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'
regex = re.compile(pattern, flags=re.IGNORECASE)
```

In [146…
```
m = regex.match('wesm@bright.net')
m.groups()
```

Out[146]:
```
('wesm', 'bright', 'net')
```

In [147…
```
regex.findall(text)
```

Out[147]:
```
[('dave', 'google', 'com'),
 ('steve', 'gmail', 'com'),
 ('rob', 'gmail', 'com'),
 ('ryan', 'yahoo', 'com')]
```

In [148…
```
print(regex.sub(r'Username: \1, Domain: \2, Suffix: \3', text))
```

```
Dave Username: dave, Domain: google, Suffix: com
Steve Username: steve, Domain: gmail, Suffix: com
Rob Username: rob, Domain: gmail, Suffix: com
Ryan Username: ryan, Domain: yahoo, Suffix: com
```

Vectorized String Functions in pandas

In [149…
```
data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',
        'Rob': 'rob@gmail.com', 'Wes': np.nan}
data = pd.Series(data)
data
```

Out[149]:
```
Dave     dave@google.com
Steve    steve@gmail.com
Rob        rob@gmail.com
Wes                  NaN
dtype: object
```

In [150…
```
data.isnull()
```

Out[150]:
```
Dave     False
Steve    False
Rob      False
Wes       True
dtype: bool
```

In [151…
```
data.str.contains('gmail')
```

Out[151]:
```
Dave     False
Steve     True
Rob       True
Wes        NaN
dtype: object
```

In [152…
```
pattern
```

Out[152]:
```
'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\\.([A-Z]{2,4})'
```

In [153…
```
data.str.findall(pattern, flags=re.IGNORECASE)
```

Out[153]:
```
Dave     [(dave, google, com)]
Steve    [(steve, gmail, com)]
Rob        [(rob, gmail, com)]
Wes                        NaN
dtype: object
```

In [154…
```
matches = data.str.match(pattern, flags=re.IGNORECASE)
matches
```

Out[154]:
```
Dave     True
Steve    True
Rob      True
Wes       NaN
dtype: object
```

In [159…
```
data.str.get(0)
```

Out[159]:
```
Dave       d
Steve      s
Rob        r
Wes      NaN
dtype: object
```

In [157…
```
data.str[0]
```

Out[157]:
```
Dave       d
Steve      s
Rob        r
Wes      NaN
dtype: object
```

In [160…
```
data.str[:5]
```

Out[160]:
```
Dave     dave@
Steve    steve
Rob      rob@g
Wes        NaN
dtype: object
```

In [161…
```
pd.options.display.max_rows = PREVIOUS_MAX_ROWS
```

Conclusion