pandas_notes

September 10, 2024

1 Importing pandas

[1]:
```
import pandas as pd
import numpy as np
```

2 Using Series

[2]:
```
g7_pop = pd.Series([35.467, 63.951, 80.940, 60.665, 127.061, 64.511, 318.523])
```

[3]:
```
g7_pop.name = 'G7 Population in millions'

g7_pop.index =[
    'Canada', 'France', 'Germany', 'Italy', 'Japan', 'UK', 'USA'
]
```

[4]:
```
g7_pop
```

[4]:
```
Canada      35.467
France      63.951
Germany     80.940
Italy       60.665
Japan      127.061
UK          64.511
USA        318.523
Name: G7 Population in millions, dtype: float64
```

[5]:
```
g7_pop['Canada']
```

[5]:
```
35.467
```

[6]:
```
g7_pop.iloc[-1]
```

[6]:
```
318.523
```

3 Using DF

[7]:
```
df = pd.DataFrame({
    'Population': [35.467, 63.951, 80.94 , 60.665, 127.061, 64.511, 318.523],
    'GDP': [
        1785387,
        2833687,
        3874437,
        2167744,
        4602367,
        2950039,
        17348075
    ],
    'Surface Area': [
        9984670,
        640679,
        357114,
        301336,
        377930,
        242495,
        9525067
    ],
    'HDI': [
        0.913,
        0.888,
        0.916,
        0.873,
        0.891,
        0.907,
        0.915
    ],
    'Continent': [
        'America',
        'Europe',
        'Europe',
        'Europe',
        'Asia',
        'Europe',
        'America'
    ]
}, columns=['Population', 'GDP', 'Surface Area', 'HDI', 'Continent'])
```

[8]:
```
df.index = [
    'Canada',
    'France',
    'Germany',
    'Italy',
```

```
    'Japan',
    'United Kingdom',
    'United States',
]
```

[9]:
```
df
```

[9]:

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
United Kingdom	64.511	2950039	242495	0.907	Europe
United States	318.523	17348075	9525067	0.915	America

[10]:
```
df.columns
```

[10]:
```
Index(['Population', 'GDP', 'Surface Area', 'HDI', 'Continent'], dtype='object')
```

[11]:
```
df.index
```

[11]:
```
Index(['Canada', 'France', 'Germany', 'Italy', 'Japan', 'United Kingdom',
       'United States'],
      dtype='object')
```

[12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7 entries, Canada to United States
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Population    7 non-null      float64
 1   GDP           7 non-null      int64  
 2   Surface Area  7 non-null      int64  
 3   HDI           7 non-null      float64
 4   Continent     7 non-null      object 
dtypes: float64(2), int64(2), object(1)
memory usage: 336.0+ bytes
```

[13]:
```
df.size
```

[13]:
```
35
```

[14]:
```
df.shape # (rows, columns)
```

[14]:
```
(7, 5)
```

[15]:
```
df.describe() # Summary of statistics
```

[15]:
```
       Population           GDP  Surface Area       HDI
count    7.000000  7.000000e+00  7.000000e+00  7.000000
mean   107.302571  5.080248e+06  3.061327e+06  0.900429
std     97.249970  5.494020e+06  4.576187e+06  0.016592
min     35.467000  1.785387e+06  2.424950e+05  0.873000
25%     62.308000  2.500716e+06  3.292250e+05  0.889500
50%     64.511000  2.950039e+06  3.779300e+05  0.907000
75%    104.000500  4.238402e+06  5.082873e+06  0.914000
max    318.523000  1.734808e+07  9.984670e+06  0.916000
```

[16]:
```
df.dtypes
```

[16]:
```
Population      float64
GDP               int64
Surface Area      int64
HDI             float64
Continent        object
dtype: object
```

[17]:
```
df.dtypes.value_counts()
```

[17]:
```
float64    2
int64      2
object     1
Name: count, dtype: int64
```

4 DF operations

[18]:
```
df.loc['Canada']
```

[18]:
```
Population       35.467
GDP             1785387
Surface Area    9984670
HDI               0.913
Continent       America
Name: Canada, dtype: object
```

[19]:
```
df.iloc[0]
```

[19]:
```
Population       35.467
GDP             1785387
Surface Area    9984670
HDI               0.913
Continent       America
Name: Canada, dtype: object
```

[20]:
```
df[['Population', 'GDP']]
```

[20]:

	Population	GDP
Canada	35.467	1785387
France	63.951	2833687
Germany	80.940	3874437
Italy	60.665	2167744
Japan	127.061	4602367
United Kingdom	64.511	2950039
United States	318.523	17348075

[21]:
```
df.loc['France' : 'Italy', 'Population'].to_frame()
```

[21]:

	Population
France	63.951
Germany	80.940
Italy	60.665

[22]:
```
df.loc[df['Population'] > 70]
```

[22]:

	Population	GDP	Surface Area	HDI	Continent
Germany	80.940	3874437	357114	0.916	Europe
Japan	127.061	4602367	377930	0.891	Asia
United States	318.523	17348075	9525067	0.915	America

[23]:
```
df.loc[df['Population'] > 70, ['Population']]
```

[23]:

	Population
Germany	80.940
Japan	127.061
United States	318.523

[24]:
```
df['Population']+=10
```

[25]:
```
df
```

[25]:

	Population	GDP	Surface Area	HDI	Continent
Canada	45.467	1785387	9984670	0.913	America
France	73.951	2833687	640679	0.888	Europe
Germany	90.940	3874437	357114	0.916	Europe
Italy	70.665	2167744	301336	0.873	Europe
Japan	137.061	4602367	377930	0.891	Asia
United Kingdom	74.511	2950039	242495	0.907	Europe
United States	328.523	17348075	9525067	0.915	America

[26]:
```
df = df.rename(
    columns={
        'HDI': 'IDH',
```

```
        'GDP' : 'PIB',
        'Population': 'Poblacion',
        'Continent': 'Continente',
        'Surface Area': 'Area'
    }, index={
        'United States': 'USA',
        'United Kingdom': 'UK',
        'Canada': 'CA',
        'Argentina': 'AR',
        'France':'FR',
    })
```

[27]:
```
df.rename(index=str.upper)
```

[27]:

	Poblacion	PIB	Area	IDH	Continente
CA	45.467	1785387	9984670	0.913	America
FR	73.951	2833687	640679	0.888	Europe
GERMANY	90.940	3874437	357114	0.916	Europe
ITALY	70.665	2167744	301336	0.873	Europe
JAPAN	137.061	4602367	377930	0.891	Asia
UK	74.511	2950039	242495	0.907	Europe
USA	328.523	17348075	9525067	0.915	America

[28]:
```
df['Language'] = 'English' # Add Column
df
```

[28]:

	Poblacion	PIB	Area	IDH	Continente	Language
CA	45.467	1785387	9984670	0.913	America	English
FR	73.951	2833687	640679	0.888	Europe	English
Germany	90.940	3874437	357114	0.916	Europe	English
Italy	70.665	2167744	301336	0.873	Europe	English
Japan	137.061	4602367	377930	0.891	Asia	English
UK	74.511	2950039	242495	0.907	Europe	English
USA	328.523	17348075	9525067	0.915	America	English

[29]:
```
df.drop(columns='Language', inplace=True) # Delete Column
df
```

[29]:

	Poblacion	PIB	Area	IDH	Continente
CA	45.467	1785387	9984670	0.913	America
FR	73.951	2833687	640679	0.888	Europe
Germany	90.940	3874437	357114	0.916	Europe
Italy	70.665	2167744	301336	0.873	Europe
Japan	137.061	4602367	377930	0.891	Asia
UK	74.511	2950039	242495	0.907	Europe
USA	328.523	17348075	9525067	0.915	America

[30]:
```
df.loc['China'] = pd.Series({'Poblacion': 1_400_000_000, 'Continente': 'Asia'})␣
↪# Add row
```

[31]:
```
df.drop('China', inplace=True) # Delete row
```

[32]:
```
df
```

[32]:

	Poblacion	PIB	Area	IDH	Continente
CA	45.467	1785387.0	9984670.0	0.913	America
FR	73.951	2833687.0	640679.0	0.888	Europe
Germany	90.940	3874437.0	357114.0	0.916	Europe
Italy	70.665	2167744.0	301336.0	0.873	Europe
Japan	137.061	4602367.0	377930.0	0.891	Asia
UK	74.511	2950039.0	242495.0	0.907	Europe
USA	328.523	17348075.0	9525067.0	0.915	America

[33]:
```
langs = pd.Series(
    ['French', 'German', 'Italian'],
    index=['FR', 'Germany', 'Italy'],
    name='Language'
)

df['Language'] = langs
```

[34]:
```
df
```

[34]:

	Poblacion	PIB	Area	IDH	Continente	Language
CA	45.467	1785387.0	9984670.0	0.913	America	NaN
FR	73.951	2833687.0	640679.0	0.888	Europe	French
Germany	90.940	3874437.0	357114.0	0.916	Europe	German
Italy	70.665	2167744.0	301336.0	0.873	Europe	Italian
Japan	137.061	4602367.0	377930.0	0.891	Asia	NaN
UK	74.511	2950039.0	242495.0	0.907	Europe	NaN
USA	328.523	17348075.0	9525067.0	0.915	America	NaN

[35]:
```
df.drop(columns='Language', inplace=True)
```

[36]:
```
df
```

[36]:

	Poblacion	PIB	Area	IDH	Continente
CA	45.467	1785387.0	9984670.0	0.913	America
FR	73.951	2833687.0	640679.0	0.888	Europe
Germany	90.940	3874437.0	357114.0	0.916	Europe
Italy	70.665	2167744.0	301336.0	0.873	Europe
Japan	137.061	4602367.0	377930.0	0.891	Asia
UK	74.511	2950039.0	242495.0	0.907	Europe
USA	328.523	17348075.0	9525067.0	0.915	America

[37]:
```
df['IDH Per Capita'] = round(df['IDH'] / df['Poblacion'],6)
df
```

[37]:

	Poblacion	PIB	Area	IDH	Continente	IDH Per Capita
CA	45.467	1785387.0	9984670.0	0.913	America	0.020080
FR	73.951	2833687.0	640679.0	0.888	Europe	0.012008
Germany	90.940	3874437.0	357114.0	0.916	Europe	0.010073
Italy	70.665	2167744.0	301336.0	0.873	Europe	0.012354
Japan	137.061	4602367.0	377930.0	0.891	Asia	0.006501
UK	74.511	2950039.0	242495.0	0.907	Europe	0.012173
USA	328.523	17348075.0	9525067.0	0.915	America	0.002785

[38]:
```
population = df['Poblacion']
```

[39]:
```
population.min(), population.max()
```

[39]:
```
(45.467, 328.523)
```

[40]:
```
population.sum()
```

[40]:
```
821.118
```

[41]:
```
population.sum() / len(population)
```

[41]:
```
117.30257142857144
```

[42]:
```
population.mean()
```

[42]:
```
117.30257142857144
```

[43]:
```
population.std()
```

[43]:
```
97.24996987121581
```

[44]:
```
population.median()
```

[44]:
```
74.511
```

[45]:
```
population.var()
```

[45]:
```
9457.556639952383
```

[46]:
```
population.describe().to_frame()
```

[46]:

	Poblacion
count	7.000000
mean	117.302571
std	97.249970
min	45.467000

```
25%     72.308000
50%     74.511000
75%    114.000500
max    328.523000
```

[47]:
```
population.quantile(.25)
```

[47]:
```
72.30799999999999
```

[48]:
```
population.quantile([.2, .4, .6, .8, 1])
```

[48]:
```
0.2     71.3222
0.4     74.1750
0.6     84.3684
0.8    127.8368
1.0    328.5230
Name: Poblacion, dtype: float64
```

5 Data cleaning

5.1 nan and inf

[49]:
```
falsy_values = (0, False, None, '', [], {})
```

For Python, all the values above are considered "falsy":

[50]:
```
any(falsy_values)
```

[50]:
```
False
```

5.1.1 How they work

Numpy has a special "nullable" value for numbers which is np.nan. It's NaN: "Not a number"

[51]:
```
np.nan
```

[51]:
```
nan
```

The np.nan value is kind of a virus. Everything that it touches becomes np.nan:

[52]:
```
3 + np.nan
```

[52]:
```
nan
```

[53]:
```
a = np.array([1, 2, 3, np.nan, np.nan, 4])
```

[54]:
```
a.sum()
```

[54]:
```
nan
```

[55]:
```
a.mean()
```

[55]:
```
nan
```

This is better than regular None values, which in the previous examples would have raised an exception:

[56]:
```
3 + None
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[56], line 1
----> 1 3 + None

TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'
```

For a numeric array, the None value is replaced by np.nan:

[147]:
```
a = np.array([1, 2, 3, np.nan, None, 4], dtype='float')
a
```

[147]:
```
array([ 1.,  2.,  3., nan, nan,  4.])
```

Numpy also supports an “Infinite” type which also behaves as a virus::

[151]:
```
np.inf
```

[151]:
```
inf
```

[152]:
```
3 + np.inf
```

[152]:
```
inf
```

[153]:
```
np.inf / 3
```

[153]:
```
inf
```

[154]:
```
np.inf / np.inf
```

[154]:
```
nan
```

[156]:
```
b = np.array([1, 2, 3, np.inf, np.nan, 4], dtype=float)

b.sum()
```

[156]:
```
nan
```

5.1.2 Checking for them

There are two functions: np.isnan and np.isinf that will perform the desired checks:

[ ]:
```
np.isnan(np.nan)
```

[ ]:
```
np.isinf(np.inf)
```

Whenever you're trying to perform an operation with a Numpy array and you know there might be missing values, you'll need to filter them out before proceeding, to avoid nan propagation. We'll use a combination of the previous np.isnan + boolean arrays for this purpose:

[ ]:
```
a = np.array([1, 2, 3, np.nan, np.nan, 4])

a[~np.isnan(a)]
```

Which is equivalent to:

[ ]:
```
a[np.isfinite(a)]
```

And with that result, all the operation can be now performed:

[ ]:
```
a[np.isfinite(a)].sum()
```

[ ]:
```
a[np.isfinite(a)].mean()
```

5.2 Process of cleaning

5.2.1 Deleting the null values

[ ]:
```
df = pd.read_csv('./data/btc-market-price-checkpoint.csv')
```

```
   2017-04-02 00:00:00  1099.169125
0  2017-04-03 00:00:00  1141.813000
1  2017-04-04 00:00:00  1141.600363
2  2017-04-05 00:00:00  1133.079314
3  2017-04-06 00:00:00  1196.307937
4  2017-04-07 00:00:00  1190.454250
```

[ ]:
```
df.isnull().sum()
```

```
2017-04-02 00:00:00    0
1099.169125            0
dtype: int64
```

If there were some null values...

[62]:
```
df[df.notnull()] # Selecting
```

[62]:
```
      2017-04-02 00:00:00  1099.169125
0     2017-04-03 00:00:00  1141.813000
1     2017-04-04 00:00:00  1141.600363
```

```
2    2017-04-05 00:00:00  1133.079314
3    2017-04-06 00:00:00  1196.307937
4    2017-04-07 00:00:00  1190.454250
..                   …          …
359  2018-03-28 00:00:00  7960.380000
360  2018-03-29 00:00:00  7172.280000
361  2018-03-30 00:00:00  6882.531667
362  2018-03-31 00:00:00  6935.480000
363  2018-04-01 00:00:00  6794.105000

[364 rows x 2 columns]
```

[65]:
```
df.dropna(inplace=True) # Selecting and dropping null
```

[66]:
```
df.dropna(how='all') # only drop fully nan rows
```

[66]:
```
     2017-04-02 00:00:00  1099.169125
0    2017-04-03 00:00:00  1141.813000
1    2017-04-04 00:00:00  1141.600363
2    2017-04-05 00:00:00  1133.079314
3    2017-04-06 00:00:00  1196.307937
4    2017-04-07 00:00:00  1190.454250
..                   …          …
359  2018-03-28 00:00:00  7960.380000
360  2018-03-29 00:00:00  7172.280000
361  2018-03-30 00:00:00  6882.531667
362  2018-03-31 00:00:00  6935.480000
363  2018-04-01 00:00:00  6794.105000

[364 rows x 2 columns]
```

[ ]:
```
df.dropna(how='any') # Default, any row with nan
```

[69]:
```
df.dropna(thresh=3) # Treshold for deleting row (3 or more nan values = deleted)
```

[69]:
```
Empty DataFrame
Columns: [2017-04-02 00:00:00, 1099.169125]
Index: []
```

[68]:
```
df.dropna(thresh=3, axis='columns') # The same, but for colums
```

[68]:
```
     2017-04-02 00:00:00  1099.169125
0    2017-04-03 00:00:00  1141.813000
1    2017-04-04 00:00:00  1141.600363
2    2017-04-05 00:00:00  1133.079314
3    2017-04-06 00:00:00  1196.307937
4    2017-04-07 00:00:00  1190.454250
..                   …          …
```

```
359  2018-03-28 00:00:00  7960.380000
360  2018-03-29 00:00:00  7172.280000
361  2018-03-30 00:00:00  6882.531667
362  2018-03-31 00:00:00  6935.480000
363  2018-04-01 00:00:00  6794.105000

[364 rows x 2 columns]
```

5.2.2 Filling the null values

[ ]:
```
df.fillna(df.mean()) # Specify the value to fill
```

[ ]:
```
df.fillna(method='ffill') # Foward fill, fills with the above ~(nan) value
```

[ ]:
```
df.fillna(method='bfill') # Backwards fill, fills with the below ~(nan) value
```

[ ]:
```
df.fillna(method='ffill', axis=0) # axis=0 = column | axis=1 = row
```

5.2.3 Detecting and fixing unique values

[ ]:
```
df.isnull().sum()
```

```
2017-04-02 00:00:00    0
1099.169125            0
dtype: int64
```

[73]:
```
df = pd.DataFrame({
    'Sex': ['M', 'F', 'F', 'D', '?'],
    'Age': [29, 30, 24, 290, 25],
})
df
```

[73]:
```
  Sex  Age
0   M   29
1   F   30
2   F   24
3   D  290
4   ?   25
```

[74]:
```
df['Sex'].unique()
```

[74]:
```
array(['M', 'F', 'D', '?'], dtype=object)
```

[75]:
```
df['Sex'].value_counts()
```

[75]:
```
Sex
F    2
M    1
```

```
D    1
?    1
Name: count, dtype: int64
```

[82]:
```
df.replace({'Sex': {'D': 'F', 'N': 'M'}}, inplace=True) # 'Wrong-Value':
 ↪'Correct-Value'

# or

df['Sex'] = df['Sex'].replace({'D': 'F', 'N': 'M'})
```

[83]:
```
df
```

[83]:
```
  Sex  Age
0   M   29
1   F   30
2   F   24
3   F  290
4   ?   25
```

For multiple columns...

[91]:
```
df['Age'].value_counts()
```

[91]:
```
Age
29     1
30     1
24     1
290    1
25     1
Name: count, dtype: int64
```

[93]:
```
df['Sex'].value_counts()
```

[93]:
```
Sex
F    3
M    1
?    1
Name: count, dtype: int64
```

[95]:
```
df.replace({
    'Sex': {
        'D': 'F',
        'N': 'M',
        '?' : np.nan
    },
    'Age': {
        290: 29
```

```
    }
})
```

[95]:
```
   Sex  Age
0    M   29
1    F   30
2    F   24
3    F   29
4  NaN   25
```

[102]:
```
df.loc[df['Age'] > 100, 'Age'] = df.loc[df['Age'] > 100, 'Age'] / 10
```

[103]:
```
df
```

[103]:
```
  Sex  Age
0   M   29
1   F   30
2   F   24
3   F   29
4   ?   25
```

5.2.4 Cleaning duplicates

For Series

[104]:
```
ambassadors = pd.Series([
    'France',
    'United Kingdom',
    'United Kingdom',
    'Italy',
    'Germany',
    'Germany',
    'Germany',
], index=[
    'Gérard Araud',
    'Kim Darroch',
    'Peter Westmacott',
    'Armando Varricchio',
    'Peter Wittig',
    'Peter Ammon',
    'Klaus Scharioth '
])
```

[105]:
```
ambassadors
```

[105]:
```
Gérard Araud                 France
Kim Darroch           United Kingdom
Peter Westmacott      United Kingdom
Armando Varricchio             Italy
```

```
Peter Wittig                Germany
Peter Ammon                 Germany
Klaus Scharioth             Germany
dtype: object
```

[112]:
```
ambassadors.duplicated() # Top-down
```

[112]:
```
Gérard Araud          False
Kim Darroch           False
Peter Westmacott       True
Armando Varricchio    False
Peter Wittig          False
Peter Ammon            True
Klaus Scharioth        True
dtype: bool
```

[110]:
```
ambassadors.duplicated(keep='last') # Inverted
```

[110]:
```
Gérard Araud          False
Kim Darroch            True
Peter Westmacott      False
Armando Varricchio    False
Peter Wittig           True
Peter Ammon            True
Klaus Scharioth       False
dtype: bool
```

[113]:
```
ambassadors.duplicated(keep=False) # All duplicates
```

[113]:
```
Gérard Araud          False
Kim Darroch            True
Peter Westmacott       True
Armando Varricchio    False
Peter Wittig           True
Peter Ammon            True
Klaus Scharioth        True
dtype: bool
```

[117]:
```
ambassadors.drop_duplicates() # Drop TD duplicates
```

[117]:
```
Gérard Araud                 France
Kim Darroch           United Kingdom
Armando Varricchio             Italy
Peter Wittig                 Germany
dtype: object
```

[118]:
```
ambassadors.drop_duplicates(keep='last') # Drop INV duplicates
```

[118]:
```
Gérard Araud                   France
Peter Westmacott       United Kingdom
Armando Varricchio              Italy
Klaus Scharioth               Germany
dtype: object
```

[119]:
```
ambassadors.drop_duplicates(keep=False) # Drop All duplicates
```

[119]:
```
Gérard Araud          France
Armando Varricchio     Italy
dtype: object
```

For Df

[123]:
```
players = pd.DataFrame({
    'Name': [
        'Kobe Bryant',
        'LeBron James',
        'Kobe Bryant',
        'Carmelo Anthony',
        'Kobe Bryant',
    ],
    'Pos': [
        'SG',
        'SF',
        'SG',
        'SF',
        'SF'
    ]
})
players
```

[123]:

	Name	Pos
0	Kobe Bryant	SG
1	LeBron James	SF
2	Kobe Bryant	SG
3	Carmelo Anthony	SF
4	Kobe Bryant	SF

[121]:
```
players.duplicated()
```

[121]:
```
0    False
1    False
2     True
3    False
4    False
dtype: bool
```

"Duplicated" means "all the column values should be duplicates". We can customize this with the

subset parameter:

[124]:
```
players.duplicated(subset=['Name'])
```

[124]:
```
0    False
1    False
2     True
3    False
4     True
dtype: bool
```

[125]:
```
players.drop_duplicates(subset=['Name'], keep='last')
```

[125]:
```
              Name Pos
1     LeBron James  SF
3  Carmelo Anthony  SF
4      Kobe Bryant  SF
```

[153]:
```
df = pd.DataFrame({
    'Data': [
        '1987_M_US _1',
        '1990?_M_UK_1',
        '1992_F_US_2',
        '1970?_M_   IT_1',
        '1985_F_I  T_2'
]})

df
```

[153]:
```
              Data
0     1987_M_US _1
1     1990?_M_UK_1
2      1992_F_US_2
3  1970?_M_   IT_1
4    1985_F_I  T_2
```

[154]:
```
df['Data'].str.split('_', expand=True)
```

[154]:
```
       0  1      2  3
0   1987  M    US   1
1  1990?  M     UK  1
2   1992  F     US  2
3  1970?  M     IT  1
4   1985  F   I  T  2
```

[156]:
```
df = df['Data'].str.split('_', expand=True)
```

[157]:
```
df
```

[157]:

	0	1	2	3
0	1987	M	US	1
1	1990?	M	UK	1
2	1992	F	US	2
3	1970?	M	IT	1
4	1985	F	I T	2

[159]:
```
df.columns = ['Year', 'Sex', 'Country', 'No Children']
```

[160]:
```
df
```

[160]:

	Year	Sex	Country	No Children
0	1987	M	US	1
1	1990?	M	UK	1
2	1992	F	US	2
3	1970?	M	IT	1
4	1985	F	I T	2

[164]:
```
df['Year'] = df['Year'].str.replace(r'(?P<year>\d{4})\?', lambda m: m.
↪group('year'), regex=True)
```

[165]:
```
df
```

[165]:

	Year	Sex	Country	No Children
0	1987	M	US	1
1	1990	M	UK	1
2	1992	F	US	2
3	1970	M	IT	1
4	1985	F	I T	2