Lecture 18: Data transformations

Imputation and normalization and standardization, oh my!

Contents

- Transformations
  - Log scale
  - Stationarity, percent change, first-differencing
  - Normalization
  - Standardization
- Missing values
  - Dropping
  - Imputing

Log rule

Logging turns exponential values linear

$\log(x^y) = y * \log(x)$

```
In [2]: log(10**2)
Out[2]: 4.605170185988092

In [3]: 2 * log(10)
Out[3]: 4.605170185988092
```

Log rule

Logging turns exponential values linear

$\log(x^y) = y * \log(x)$

```
In [2]: log(10**2)
Out[2]: 4.605170185988092

In [3]: 2 * log(10)
Out[3]: 4.605170185988092
```

Data can't have zeros or negatives if you are logging!

```
In [4]: log(1)
Out[4]: 0.0

In [5]: log(0)
__main__:1: RuntimeWarning: divide by zero encountered in log
Out[5]: -inf

In [6]: log(-1)
__main__:1: RuntimeWarning: invalid value encountered in log
Out[6]: nan
```

Log rule

Logging turns exponential values linear

$\log(x^y) = y * \log(x)$

```
In [2]: log(10**2)
Out[2]: 4.605170185988092

In [3]: 2 * log(10)
Out[3]: 4.605170185988092
```

Data can't have zeros or negatives if you are logging!

```
In [4]: log(1)
Out[4]: 0.0

In [5]: log(0)
__main__:1: RuntimeWarning: divide by zero encountered in log
Out[5]: -inf

In [6]: log(-1)
__main__:1: RuntimeWarning: invalid value encountered in log
Out[6]: nan
```

Inverse Hyperbolic Sine (HIS) Transformation

$$\ln(\,x + \sqrt{x^2 + 1}\,)$$

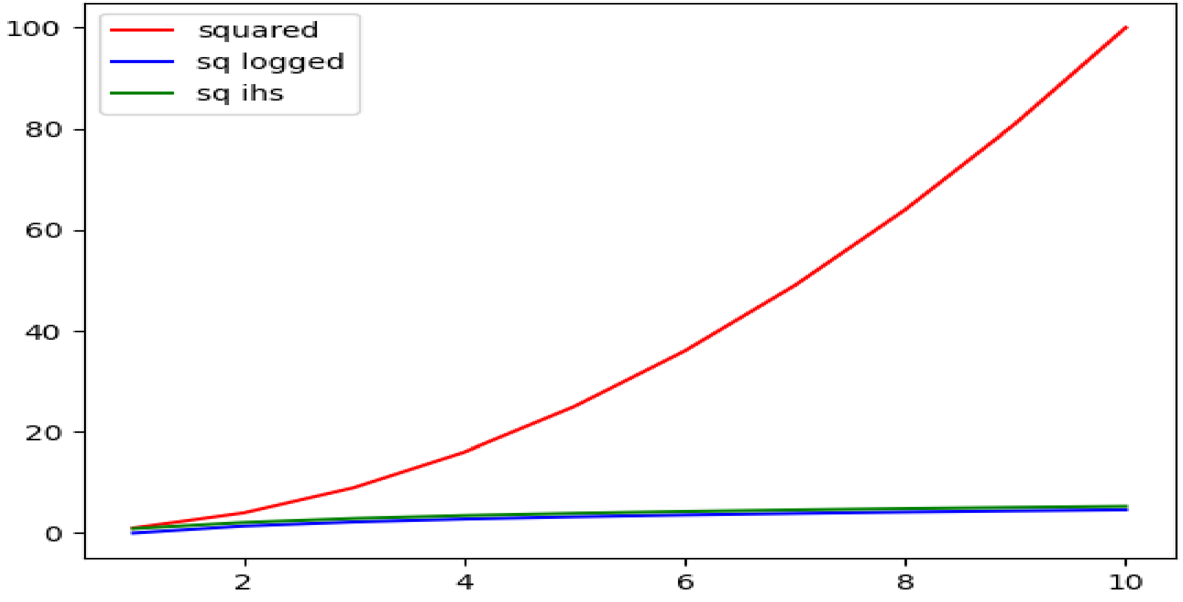
- Closely approximates the log transformation
- Handles zeroes and negatives
- See *Burbridge, Magee, and Robb, Journal of the American Statistical Association, 1988*

```
def ihs(x):
    return log(x + sqrt(x**2 + 1))
```

Logged vs un-logged

```
linear  = range(1, 11)
squared = [v**2 for v in linear]
logged  = [log(s) for s in squared]
ihsed = [ihs(s) for s in squared]
```

Logged vs un-logged



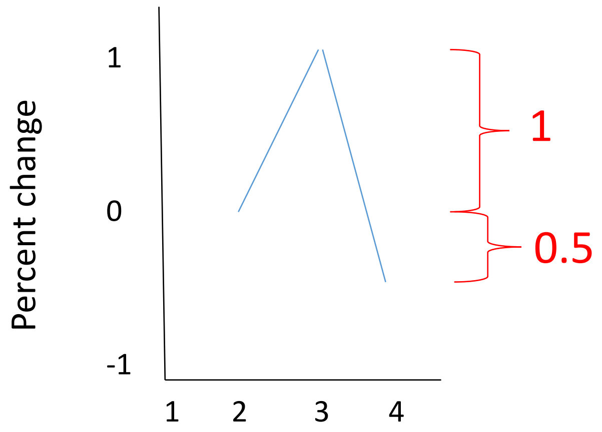

```
linear  = range(1, 11)
squared = [v**2 for v in linear]
logged  = [log(s) for s in squared]
ihsed = [ihs(s) for s in squared]
```

```
fig, ax = plt.subplots()
ax.plot(linear, squared, 'r-', label='squared')
ax.plot(linear, logged,  'b-', label='logged')
ax.plot(linear, ihsed, 'g-', label='ihs')
ax.legend()
```

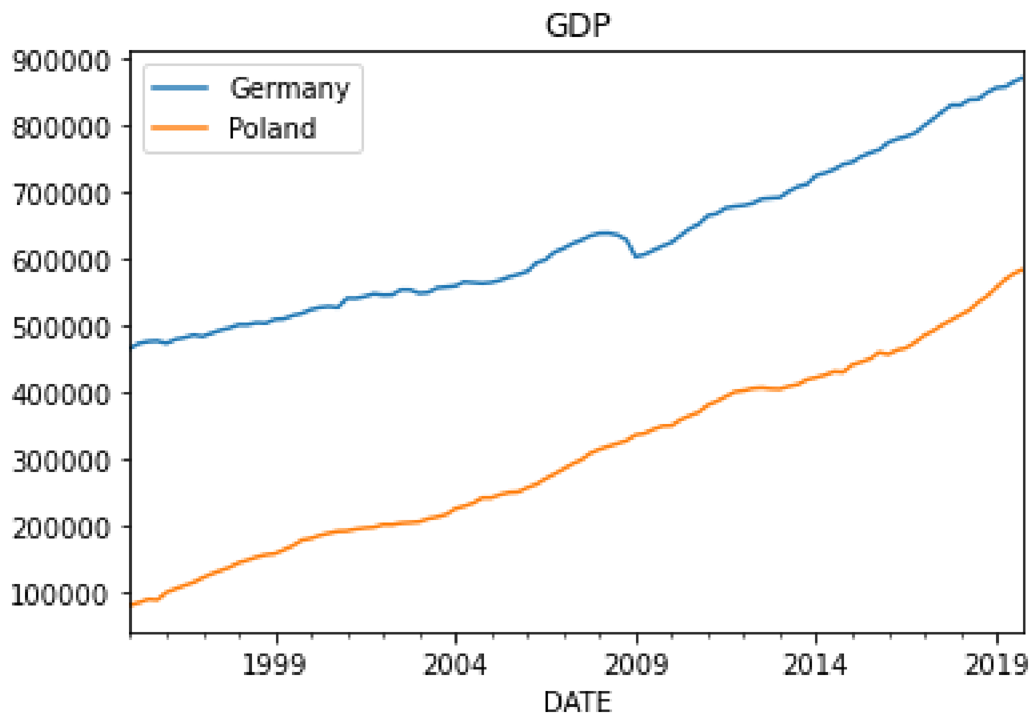
Percent change?

Series: 2, 2, 4, 2, 4

Percent change: NA -> 2 = NA
2 -> 2 = 0%
2 -> 4 = 100%
4 -> 2 = -50%



Stationarity



Stationarity

```
42    model = smf.ols('GDPGermany ~ GDPPoland ', data=df)
43    result = model.fit()
```

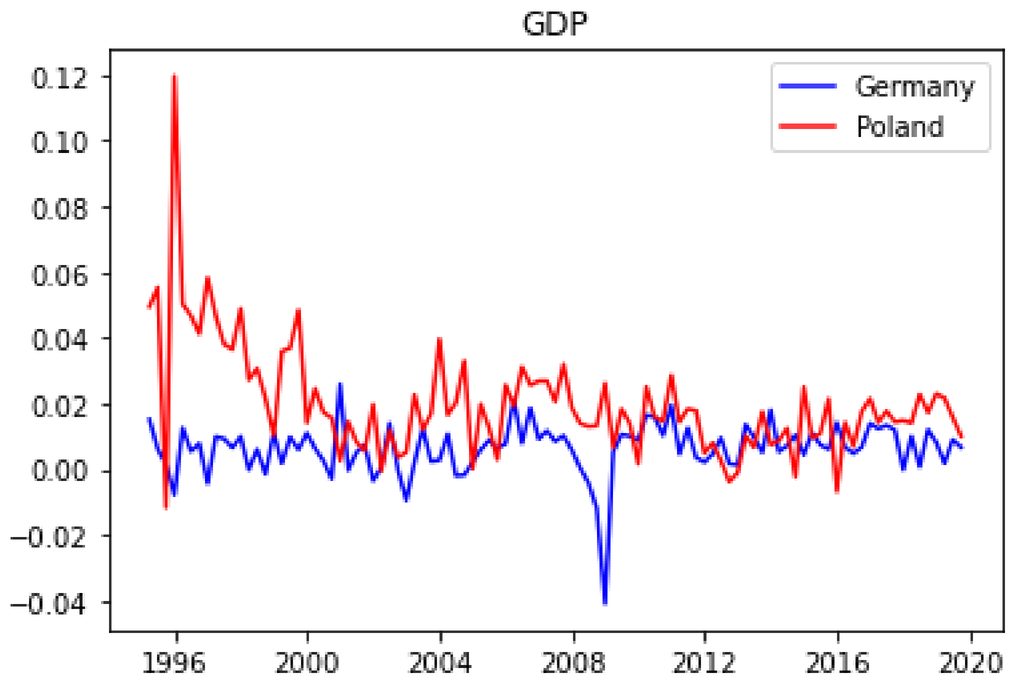
```
                            OLS Regression Results
==============================================================================
Dep. Variable:             GDPGermany   R-squared:                       0.973
Model:                            OLS   Adj. R-squared:                  0.973
Method:                 Least Squares   F-statistic:                     3594.
Date:                Wed, 26 May 2021   Prob (F-statistic):           4.85e-79
Time:                        10:48:41   Log-Likelihood:                -1125.6
No. Observations:                 100   AIC:                             2255.
Df Residuals:                      98   BIC:                             2260.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   3.762e+05   4625.478     81.339      0.000    3.67e+05    3.85e+05
GDPPoland      0.8259      0.014     59.949      0.000       0.799       0.853
==============================================================================
Omnibus:                        8.858   Durbin-Watson:                   0.088
Prob(Omnibus):                  0.012   Jarque-Bera (JB):                8.733
Skew:                          -0.705   Prob(JB):                       0.0127
Kurtosis:                       3.326   Cond. No.                     8.22e+05
==============================================================================
```

Log first-difference

```
df['Germany_GDP_lfd'] = log(df['GDPGermany']) - log(df['GDPGermany'].shift())
df['Poland_GDP_lfd']  = log(df['GDPPoland'])  - log(df['GDPPoland'].shift())
```

Log first-difference

```
df['Germany_GDP_lfd'] = log(df['GDPGermany']) - log(df['GDPGermany'].shift())
df['Poland_GDP_lfd']  = log(df['GDPPoland'])  - log(df['GDPPoland'].shift())
```



Log first-difference

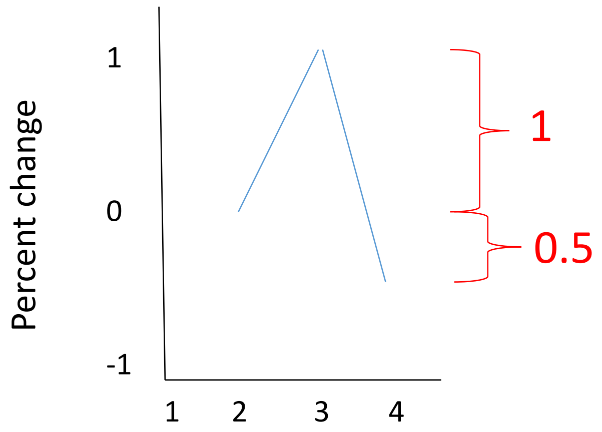
```
                            OLS Regression Results
==============================================================================
Dep. Variable:        Germany_GDP_lfd   R-squared:                       0.011
Model:                            OLS   Adj. R-squared:                  0.001
Method:                 Least Squares   F-statistic:                     1.127
Date:                Wed, 26 May 2021   Prob (F-statistic):              0.291
Time:                        10:59:00   Log-Likelihood:                 338.64
No. Observations:                  99   AIC:                            -673.3
Df Residuals:                      97   BIC:                            -668.1
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept         0.0073      0.001      5.926      0.000       0.005       0.010
Poland_GDP_lfd   -0.0498      0.047     -1.062      0.291      -0.143       0.043
==============================================================================
Omnibus:                       68.088   Durbin-Watson:                   1.564
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              558.385
Skew:                          -2.028   Prob(JB):                    5.60e-122
Kurtosis:                      13.905   Cond. No.                         58.4
==============================================================================
```

Percent change?

Series: 2, 2, 4, 2, 4

Percent change:

NA -> 2 = NA
2 -> 2 = 0%
2 -> 4 = 100%
4 -> 2 = -50%



Percent change?

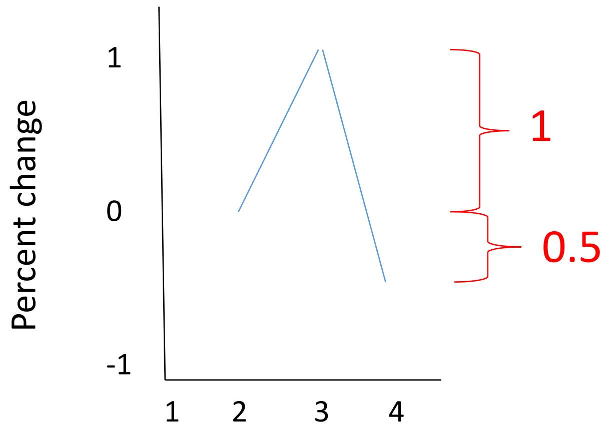
Series: 2, 2, 4, 2, 4

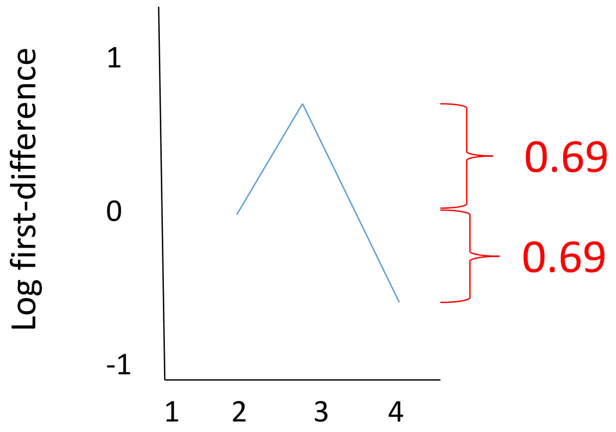
Percent change:

NA -> 2 = NA
2 -> 2 = 0%
2 -> 4 = 100%
4 -> 2 = -50%

Log first-diff:

log(NA) – log(2) = NA
log(2) – log(2) = 0
log(2) – log(4) = 0.693
log(4) – log(2) = -0.693





Percent change vs log first-difference

	Germany_GDP_pctchange	Germany_GDP_lfd
0	NaN	NaN
1	0.015303	0.015187
2	0.006200	0.006181
3	0.001477	0.001475
4	-0.007745	-0.007775
5	0.012676	0.012597
6	0.005471	0.005456
7	0.007891	0.007860
8	-0.004282	-0.004291
9	0.009955	0.009906

Scaling

Normalization: rescale data to [0, 1]

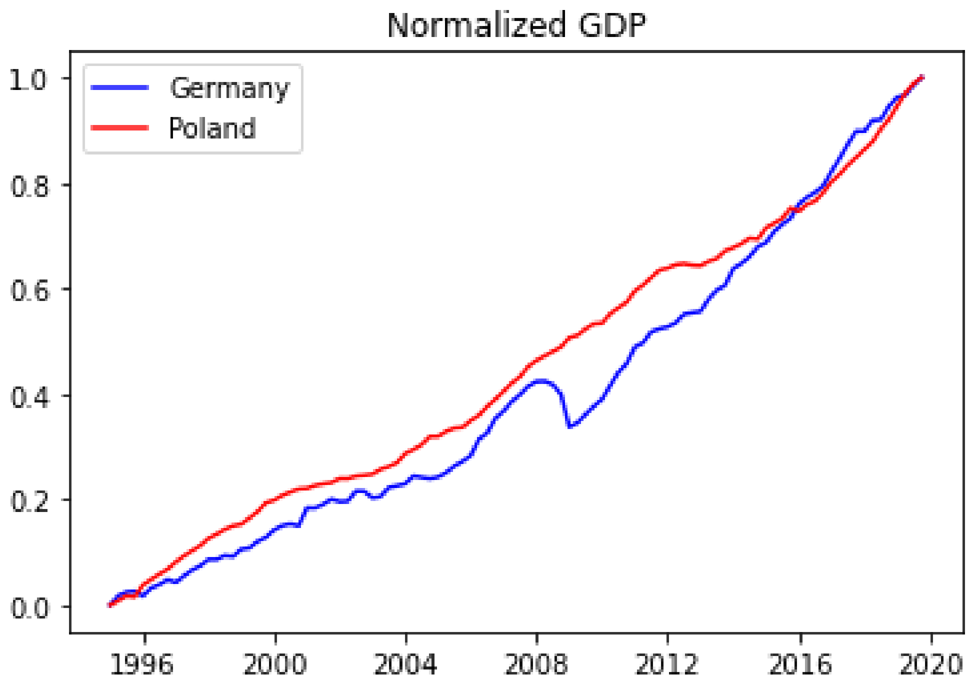
$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Standardization: rescale data to mean 0 and standard deviation 1

$$X_{std} = \frac{X - \mu}{\sigma}$$

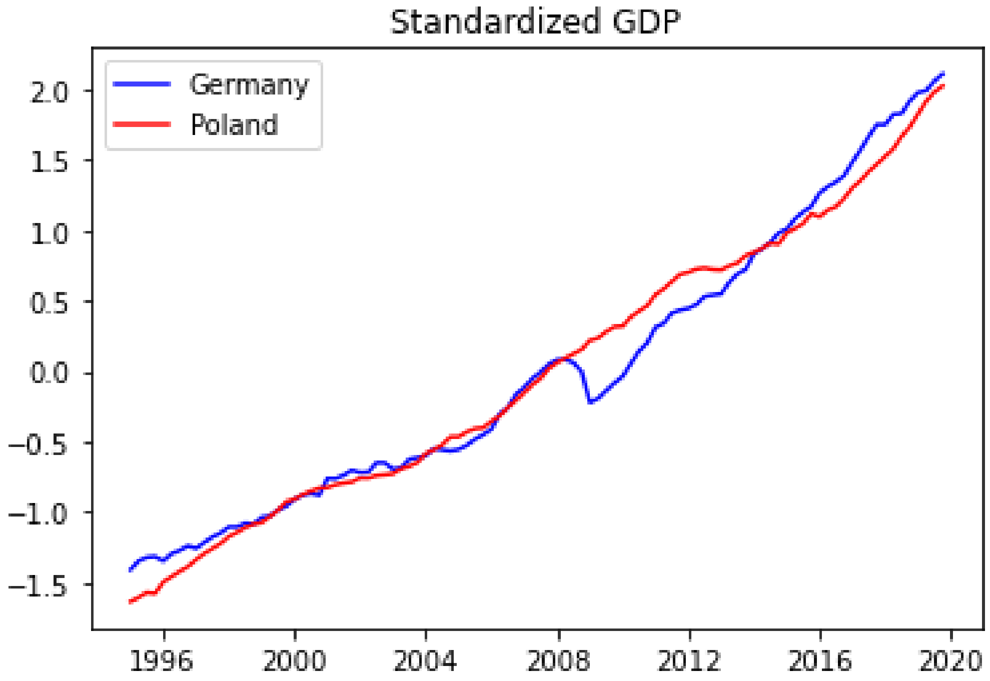
Normalization

```
df['GDPGermany_norm'] = ((df['GDPGermany'] - df['GDPGermany'].min()) /
                         (df['GDPGermany'].max() - df['GDPGermany'].min()))
df['GDPPoland_norm'] = ((df['GDPPoland'] - df['GDPPoland'].min()) /
                         (df['GDPPoland'].max() - df['GDPPoland'].min()))
```



Standardization

```
df['GDPGermany_std'] = (df['GDPGermany'] - df['GDPGermany'].mean()) /df['GDPGermany'].std()
df['GDPPoland_std'] = (df['GDPPoland'] - df['GDPPoland'].mean()) /df['GDPPoland'].std()
```



Standardization vs Normalization

```
In [57]: df['GDPGermany_norm'].std()
Out[57]: 0.28451997608044993

In [58]: df['GDPGermany_std'].std()
Out[58]: 0.9999999999999998
```

```
In [59]: df['GDPGermany_norm'].max()
Out[59]: 1.0

In [60]: df['GDPGermany_std'].max()
Out[60]: 2.105693737402705 6
```

```
In [61]: df['GDPGermany_norm'].mean()
Out[61]: 0.40088806820142886

In [62]: df['GDPGermany_std'].mean()
Out[62]: -2.0872192862952942e-16
```

```
In [63]: df['GDPGermany_norm'].min()
Out[63]: 0.0

In [64]: df['GDPGermany_std'].min()
Out[64]: -1.408997968171047 4
```

Missing values

To impute or to drop?

Missing values

```
df = pd.DataFrame({'one':[1, 2, NaN, 4, 5, NaN],
                   'two':[2, 3, 4, 5, 6, NaN]})
```

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

Missing values

```python
df = pd.DataFrame({'one':[1, 2, NaN, 4, 5, NaN],
                   'two':[2, 3, 4, 5, 6, NaN]})
```

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [68]: df.isnull()
Out[68]:
     one    two
0  False  False
1  False  False
2   True  False
3  False  False
4  False  False
5   True   True
```

Missing values

```
df = pd.DataFrame({'one':[1, 2, NaN, 4, 5, NaN],
                   'two':[2, 3, 4, 5, 6, NaN]})
```

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [68]: df.isnull()
Out[68]:
     one    two
0  False  False
1  False  False
2   True  False
3  False  False
4  False  False
5   True   True
```

```
In [69]: df.isnull().sum(axis=0)
Out[69]:
one     2
two     1
dtype: int64
```

```
In [70]: df.isnull().sum(axis=1)
Out[70]:
0    0
1    0
2    1
3    0
4    0
5    2
dtype: int64
```

Missing values: dropping

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [71]: df.dropna()
Out[71]:
   one  two
0  1.0  2.0
1  2.0  3.0
3  4.0  5.0
4  5.0  6.0
```

```
In [72]: df.dropna(axis=1)
Out[72]:
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4, 5]
```

Missing values: dropping

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [73]: df.dropna(how='all')
Out[73]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
```

Missing values: dropping

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [73]: df.dropna(how='all')
Out[73]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
```

```
In [74]: df.dropna(subset=['one', 'two'])
Out[74]:
   one  two
0  1.0  2.0
1  2.0  3.0
3  4.0  5.0
4  5.0  6.0
```

Missing values: dropping

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [76]: df.dropna(thresh=1)
Out[76]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
```

```
In [73]: df.dropna(how='all')
Out[73]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
```

```
In [74]: df.dropna(subset=['one', 'two'])
Out[74]:
   one  two
0  1.0  2.0
1  2.0  3.0
3  4.0  5.0
4  5.0  6.0
```

Missing values: imputing

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [77]: df.fillna(100)
Out[77]:
     one    two
0    1.0    2.0
1    2.0    3.0
2  100.0    4.0
3    4.0    5.0
4    5.0    6.0
5  100.0  100.0
```

Missing values: imputing

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [77]: df.fillna(100)
Out[77]:
     one    two
0    1.0    2.0
1    2.0    3.0
2  100.0    4.0
3    4.0    5.0
4    5.0    6.0
5  100.0  100.0
```

```
In [79]: df.fillna(df.mean())
Out[79]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  3.0  4.0
3  4.0  5.0
4  5.0  6.0
5  3.0  4.0
```

Missing values: imputing

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [77]: df.fillna(100)
Out[77]:
     one    two
0    1.0    2.0
1    2.0    3.0
2  100.0    4.0
3    4.0    5.0
4    5.0    6.0
5  100.0  100.0
```

```
In [79]: df.fillna(df.mean())
Out[79]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  3.0  4.0
3  4.0  5.0
4  5.0  6.0
5  3.0  4.0
```

```
In [80]: df['one'].fillna(df['two'])
Out[80]:
0    1.0
1    2.0
2    4.0
3    4.0
4    5.0
5    NaN
Name: one, dtype: float64
```

Missing values: imputing

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [81]: df.ffill()
Out[81]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  2.0  4.0
3  4.0  5.0
4  5.0  6.0
5  5.0  6.0
```

```
In [82]: df.bfill()
Out[82]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  4.0  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

Missing values: imputing

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [83]: df.interpolate(method='linear')
Out[83]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  3.0  4.0
3  4.0  5.0
4  5.0  6.0
5  5.0  6.0
```

Missing values: imputing

```
In [67]: df
Out[67]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  NaN  4.0
3  4.0  5.0
4  5.0  6.0
5  NaN  NaN
```

```
In [83]: df.interpolate(method='linear')
Out[83]:
   one  two
0  1.0  2.0
1  2.0  3.0
2  3.0  4.0
3  4.0  5.0
4  5.0  6.0
5  5.0  6.0
```

?