```
df = pd.DataFrame({'AAA' : [4,5,6,7], 'BBB' : [10,20,30,40],'CCC' : [100,50,-30,-50]})

# An if-then on one column
# 如果满足 df.AAA > 5 的行，设置 BBB 列为 -1
df.loc[df.AAA > 5, 'BBB'] = -1

# An if-then with assignment to 2 columns
df.loc[df.AAA >= 5, ['BBB', 'CCC']] = 555

# Add another line with different logic, to do the -else
df.loc[df.AAA < 5,['BBB','CCC']] = 2000

# Or use pandas where after you've set up a mask
df_mask = pd.DataFrame({'AAA': [True] * 4, 'BBB': [False] * 4, 'CCC': [True, False] * 2})
df.where(df_mask, -1000)

# if-then-else using numpy's where()
df['logic'] = df.where(df['AAA'] >= 5, 'high', 'low')

# Split a frame with a boolean criterion
dflow = df[df.AAA < 5]
dhigh = df[df.AAA >= 5]

# Select with multi-column criteria
df = pd.DataFrame({'AAA' : [4,5,6,7], 'BBB' : [10,20,30,40],'CCC' : [100,50,-30,-50]})

#
newseries = df.loc[(df['BBB'] < 25) & (df['CCC'] >= -40), 'AAA']
newseries = df.loc[(df['BBB'] > 25) | (df['CCC'] >= -40), 'AAA']
df.loc[(df['BBB'] > 25) | (df['CCC'] >= 75), 'AAA'] = 0.1

# Select rows with data closest to certain value using argsort
df = pd.DataFrame({'AAA' : [4,5,6,7], 'BBB' : [10,20,30,40],'CCC' : [100,50,-30,-50]})
aValue = 43.0
df.loc[(df['CCC'] - aValue).abs().argsort()]

# Dynamically reduce a list of criteria using a binary operators
df = pd.DataFrame({'AAA' : [4,5,6,7], 'BBB' : [10,20,30,40],'CCC' : [100,50,-30,-50]})
Crit1 = df.AAA <= 5.5
Crit2 = df.BBB == 10.0
Crit3 = df.CCC > -40.0
AllCrit = Crit1 & Crit2 & Crit3
df[AllCrit]
####################################################################################################
df[AllCrit]
AllCrit = functools.reduce(lambda x,y: x & y, CritList)

# Using both row labels and value conditionals
df = pd.DataFrame({'AAA' : [4,5,6,7], 'BBB' : [10,20,30,40],'CCC' : [100,50,-30,-50]})
df[(df.AAA <= 6) & (df.index.isin([0,2,4]))]

# Use loc for label-oriented slicing and iloc positional slicing
data = {'AAA' : [4,5,6,7], 'BBB' : [10,20,30,40],'CCC' : [100,50,-30,-50]}
df = pd.DataFrame(data=data,index=['foo','bar','boo','kar'])

# 1. Positional-oriented
df.loc['bar':'kar']

# 2. Label-oriented
df.loc['bar':'kar']

# 3. General
df.iloc[0:3]

# Ambiguity arises when an index consists of integers with a non-zero start or non-unit increment
# Note index starts at 1
df2 = pd.DataFrame(data=data,index=[1,2,3,4])

# Position-oriented
df2.iloc[1:3]

# Label-oriented
df2.loc[1:3]

# Using inverse operator (~) to take the complement of a mask
df = pd.DataFrame({'AAA' : [4,5,6,7], 'BBB' : [10,20,30,40], 'CCC' : [100,50,-30,-50]})
```

```
df[~((df.AAA <= 6) & (df.index.isin([0,2,4])))]

##############################################################################################
# Extend a panel frame by transposing, adding a new dimension, and transposing back to the original dimensions
df1, df2, df3 = pd.DataFrame(data, rng, cols), pd.DataFrame(data, rng, cols), pd.DataFrame(data, rng, cols)
pf = pd.Panel({'df1':df1,'df2':df2,'df3':df3})

# Efficiently and dynamically creating new columns using applymap
df = pd.DataFrame({'AAA' : [1,2,1,3], 'BBB' : [1,1,2,2], 'CCC' : [2,1,3,1]})
source_cols = df.columns # or some subset would work too
new_cols = [str(x) + "_cat" for x in source_cols]
categories = {1 : 'Alpha', 2 : 'Beta', 3 : 'Charlie' }
df[new_cols] = df[source_cols].applymap(categories.get)

##############################################################################################
# Keep other columns when using min() with groupby
df = pd.DataFrame({'AAA' : [1,1,1,2,2,2,3,3], 'BBB' : [2,1,3,4,5,1,2,3]})

# idxmin() to get the index of the mins
df.loc[df.groupby('AAA')['BBB'].idxmin()]
df.sort_values(by="BBB").groupby("AAA", as_index=False).first()

##############################################################################################


# Creating a multi-index from a labeled frame
df = pd.DataFrame({'row' : [0,1,2],'One_X' : [1.1,1.1,1.1],'One_Y' : [1.2,1.2,1.2],'Two_X' : [1.11,1.11,1.11],'Two_Y' : [1.22,1.22,1.22]})

# As Labelled Index
df = df.set_index('row')

# With Hierarchical Columns
df.columns = pd.MultiIndex.from_tuples([tuple(c.split('_')) for c in df.columns])
```

WORKING WITH TEXT DATA

```
# 设置一个pandas的Series对象
s = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA', 'dog', 'cat'])

# 通过 Series 对象的 str属性，操作 Series 对象中的每一个元素

# 将每一个字符串类型的元素变成小写
s.str.lower()

# 将每一个字符串类型的元素变成大写
s.str.upper()

# 获取每一个字符串类型元素的长度
s.str.len()

# 获取 pandas 的 Index 对象
idx = pd.Index([' jack', 'jill ', ' jesse ', 'frank'])

# 将 pandas 的 Index 对象左右两侧的空格去掉
idx.str.strip()

# 将 pandas 的 Index 对象左侧的空格去掉
idx.str.lstrip()

# 将 pandas 的 Index 对象右侧的空格去掉
idx.str.rstrip()

# 构建用于说明 df.coumns 也具有 str 属性的 DataFrame
df = pd.DataFrame(randn(3, 2), columns=[' Column A ', ' Column B '], index=range(3))

# 使用 df.columns 的 str 属性
df.columns.str.strip()

# 综合案例
df.columns.str.strip().str.lower().str.replace(' ', '_')
```

> Note: If you have a Series where lots of elements are repeated (i.e. the number of unique elements in the Series is a lot smaller than the length of

> the Series), it can be faster to convert the original Series to one of type category and then use .str. or .dt. on that. The performance difference comes from the fact that, for Series of type category, the string operations are done on the .categories and not on each element of the Series.

> Please note that a Series of type category with string .categories has some limitations in comparison of Series of type string (e.g. you can't add strings to each other: s + " " + s won't work if s is a Series of type category). Also, .str methods which operate on elements of type list are not available on such a Series.

```
# 将一个Series对象的元素变更为list对象
s2 = pd.Series(['a_b_c', 'c_d_e', np.nan, 'f_g_h'])
s2.str.split('_')

# 获取 Series 对象中 list对象中的元素
s2.str.split('_').str.get(1)

s2.str.split('_').str[1]

# 将 Series 对象扩展为一个 DataFrame， 使用 split 函数的 expand 参数
s2.str.split('_', expand = True)

# 限制设定的 split出来的个数，使用 split 函数的 n 参数
s2.str.split('_', expand = True, n = 1)

# 从相反的方向进行 split 可以使用 rsplit 函数
s2.str.rsplit('_', expand = True, n = 2)

# replace 和 findall 函数的案例
s3 = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', '', np.nan, 'CABA', 'dog', 'cat'])

s3.str.replace('^.a|dog', 'XX-XX ', case=False)

# 特殊符号的替换
dollars = pd.Series(['12', '-$10', '$10,000'])

# 简单的替换 $
dollars.str.replace('$', '')

# 如果要替换 -$
dollars.str.replace(r'-\$', '-')
```

> The replace method can also take a callable as replacement. It is called on every pat using re.sub(). The callable should expect one positional argument (a regex object) and return a string

```
# Reverse every lowercase alphabetic word
pat = r'[a-z]+'

repl = lambda m: m.group(0)[ : : -1]

pd.Series(['foo 123', 'bar baz', np.nan]).str.replace(pat, repl)

# Using regex groups
pat = r"(?P<one>\w+) (?P<two>\w+) (?P<three>\w+)"

repl = lambda m: m.group('two').swapcase()

pd.Series(['Foo Bar Baz', np.nan]).str.replace(pat, repl)
```

> The replace method also accepts a compiled regular expression object from re.compile() as a pattern. All flags should be included in the compiled regular expression object.

```
import re

regex_pat = re.compile(r'^.a|dog', flags=re.IGNORECASE)

s3.str.replace(regex_pat, 'XX-XX ')

# Including a flags argument when calling replace with a compiled regular expression object will raise a ValueError.
s3.str.replace(regex_pat, 'XX-XX ', flags=re.IGNORECASE)
```

> You can use [] notation to directly index by position locations. If you index past the end of the string, the result will be a NaN

```
s = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA', 'dog', 'cat'])

s.str[0]

s.str[1]
```

IO TOOLS (TEXT, CSV, HDF5, ...)

-Format Type-	-Data Description -	-Reader-	- Writer-
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	Stata	read_stata	to_stata
binary	Msgpack	read_msgpack	to_msgpack
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq