Change Values

pandas' methods to change the values of a pandas DataFrame or a pandas Series.

pandas.DataFrame.agg: Aggregate over Columns or Rows Using Multiple Operations

If you want to aggregate over columns or rows using one or more operations, try `pd.DataFrame.agg`.

```
from collections import Counter
import pandas as pd


def count_two(nums: list):
    return Counter(nums)[2]


df = pd.DataFrame({"coll": [1, 3, 5], "col2": [2, 4, 6]})
df.agg(["sum", count_two])
```

	coll	col2
sum	9	12
count_two	0	1

pandas.DataFrame.agg: Apply Different Aggregations to Different Columns

If you want to apply different aggregations to different columns, insert a dictionary of column and aggregation methods to the `pd.DataFrame.agg` method.

```
df = pd.DataFrame({"a": [1, 2, 3, 4], "b": [2, 3, 4, 5]})

df.agg({"a": ["sum", "mean"], "b": ["min", "max"]})
```

	a	b
sum	10.0	NaN
mean	2.5	NaN
min	NaN	2.0
max	NaN	5.0

pandas.DataFrame.pipe: Increase the Readability of your Code when Applying Multiple Functions to a DataFrame

If you want to increase the readability of your code when applying multiple functions to a DataFrame, use `pands.DataFrame.pipe` method.

```
from textblob import TextBlob

def remove_white_space(df: pd.DataFrame):
    df['text'] = df['text'].apply(lambda row: row.strip())
    return df

def get_sentiment(df: pd.DataFrame):
    df['sentiment'] = df['text'].apply(lambda row:
                                       TextBlob(row).sentiment[0])
    return df

df = pd.DataFrame({'text': ["It is a beautiful day today  ",
                            "  This movie is terrible"]})

df = (df.pipe(remove_white_space)
    .pipe(get_sentiment)
)

df
```

	text	sentiment
0	It is a beautiful day today	0.85
1	This movie is terrible	-1.00

pandas.Series.map: Change Values of a Pandas Series Using a Dictionary

If you want to change values of a pandas Series using a dictionary, use `pd.Series.map`.

```
s = pd.Series(["a", "b", "c"])

s.map({"a": 1, "b": 2, "c": 3})
```

```
0    1
1    2
2    3
dtype: int64
```

pandas.Series.str: Manipulate Text Data in a Pandas Series

If you are working the text data in a pandas Series, instead of creating your own functions, use `pandas.Series.str` to access common methods to process string.

The code below shows how to convert text to lower case then replace “e” with “a”.

```
fruits = pd.Series(['Orange', 'Apple', 'Grape'])
fruits
```

```
0    Orange
1     Apple
2     Grape
dtype: object
```

```
fruits.str.lower()
```

```
0    orange
1     apple
2     grape
dtype: object
```

```
fruits.str.lower().str.replace("e", "a")
```

```
0    oranga
1     appla
2     grapa
dtype: object
```

Find other useful string methods here.

set_categories in Pandas: Sort Categorical Column by a Specific Ordering

If you want to sort pandas DataFrame's categorical column by a specific ordering such as small, medium, large, use `df.col.cat.set_categories()` method.

```
df = pd.DataFrame(
    {"col1": ["large", "small", "mini", "medium", "mini"],
"col2": [1, 2, 3, 4, 5]}
)
ordered_sizes = "large", "medium", "small", "mini"

df.col1 = df.col1.astype("category")
df.col1.cat.set_categories(ordered_sizes, ordered=True,
inplace=True)
df.sort_values(by="col1")
```

	col1	col2
0	large	1
3	medium	4
1	small	2
2	mini	3
4	mini	5

parse_dates: Convert Columns into Datetime When Using Pandas to Read CSV Files

If there are datetime columns in your csv file, use the `parse_dates` parameter when reading csv file with pandas. This reduces one extra step to convert these columns from string to datetime after reading the file.

```
df = pd.read_csv("data1.csv", parse_dates=["date_column_1",
"date_column_2"])
```

```
df
```

	date_column_1	date_column_2	value
0	2021-02-10	2021-02-11	3
1	2021-02-12	2021-02-13	3

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date_column_1  2 non-null      datetime64[ns]
 1   date_column_2  2 non-null      datetime64[ns]
 2   value          2 non-null      int64
dtypes: datetime64[ns](2), int64(1)
memory usage: 176.0 bytes
```

Pandas.Series.isin: Filter Rows Only If Column Contains Values From Another List

When working with a pandas Dataframe, if you want to select the rows when a column contains values from another list, the fastest way is to use `isin`.

In the example below, row 2 is filtered out because 3 is not in the list.

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

	a	b
0	1	4
1	2	5
2	3	6

```
l = [1, 2, 6, 7]
df.a.isin(l)
```

```
0     True
1     True
2    False
Name: a, dtype: bool
```

```
df = df[df.a.isin(l)]
df
```

	a	b
0	1	4
1	2	5

Specify Suffixes When Using df.merge()

If you are merging 2 dataframes that have the same features using `df.merge()`, it might be confusing to know which dataframe `a_x` or `a_y` belongs to.

```
df1 = pd.DataFrame({"left_key": [1, 2, 3], "a": [4, 5, 6]})
df2 = pd.DataFrame({"right_key": [1, 2, 3], "a": [5, 6, 7]})
df1.merge(df2, left_on="left_key", right_on="right_key")
```

	left_key	a_x	right_key	a_y
0	1	4	1	5
1	2	5	2	6
2	3	6	3	7

A better way is to specify suffixes of the features in each Dataframe like below. Now `a_x` becomes `a_left` and `a_y` becomes `a_right`.

```
df1.merge(df2, left_on="left_key", right_on="right_key",
suffixes=("_left", "_right"))
```

	left_key	a_left	right_key	a_right
0	1	4	1	5
1	2	5	2	6
2	3	6	3	7

Try it if you want the names of your columns to be less confusing.

Highlight your pandas DataFrame

Have you ever wanted to highlight your pandas DataFrame to analyze it easier? For example, positive values will be highlighted as green and negative values will be highlighted as red.

That could be done with `df.style.apply(highlight_condition_func)`.

```
df = pd.DataFrame({"col1": [-5, -2, 1, 4], "col2": [2, 3, -1, 4]})
```

```
def highlight_number(row):
    return [
        "background-color: red; color: white"
        if cell <= 0
        else "background-color: green; color: white"
        for cell in row
    ]
```

```
df.style.apply(highlight_number)
```

	col1	col2
0	-5	2
1	-2	3
2	1	-1
3	4	4

Assign Values to Multiple New Columns

If you want to assign values to multiple new columns, instead of assigning them separately, you can do everything in one line of code with `df.assign`.

In the code below, I first created `col3` then use `col3` to create `col4`. Everything is in one line of code.

```
df = pd.DataFrame({"col1": [1, 2], "col2": [3, 4]})

df = df.assign(col3=lambda x: x.col1 * 100 + x.col2).assign(
    col4=lambda x: x.col2 * x.col3
)
df
```

	col1	col2	col3	col4
0	1	3	103	309
1	2	4	204	816

Reduce pandas.DataFrame's Memory

If you want to reduce the memory of your pandas DataFrame, start with changing the data type of a column. If your categorical variable has low cardinality, change the data type to category like below.

```
from sklearn.datasets import load_iris

X, y = load_iris(as_frame=True, return_X_y=True)
df = pd.concat([X, pd.DataFrame(y, columns=["target"])],
axis=1)
df.memory_usage()
```

```
Index                 128
sepal length (cm)    1200
sepal width (cm)     1200
petal length (cm)    1200
petal width (cm)     1200
target               1200
dtype: int64
```

```
df["target"] = df["target"].astype("category")
df.memory_usage()
```

```
Index                 128
sepal length (cm)    1200
sepal width (cm)     1200
petal length (cm)    1200
petal width (cm)     1200
target                282
dtype: int64
```

The memory is now is reduced to almost a fifth of what it was!

pandas.DataFrame.explode: Transform Each Element in an Iterable to a Row

When working with `pandas DataFrame`, if you want to transform each element in an iterable to a row, use `explode`.

```
df = pd.DataFrame({"a": [[1, 2], [4, 5]], "b": [11, 13]})
df
```

	a	b
0	[1, 2]	11
1	[4, 5]	13

```
df.explode("a")
```

	a	b
0	1	11
0	2	11
1	4	13
1	5	13

pandas.cut: Bin a DataFrame's values into Discrete Intervals

If you want to bin your Dataframe's values into discrete intervals, use `pd.cut`.

```
df = pd.DataFrame({"a": [1, 3, 7, 11, 14, 17]})

bins = [0, 5, 10, 15, 20]
df["binned"] = pd.cut(df["a"], bins=bins)

df
```

	a	binned
0	1	(0, 5]
1	3	(0, 5]
2	7	(5, 10]
3	11	(10, 15]
4	14	(10, 15]
5	17	(15, 20]

Forward Fill in pandas: Use the Previous Value to Fill the Current Missing Value

If you want to use the previous value in a column or a row to fill the current missing value in a pandas DataFrame, use `df.fillna(method='ffill')`. `ffill` stands for forward fill.

```
import numpy as np

df = pd.DataFrame({"a": [1, np.nan, 3], "b": [4, 5, np.nan],
"c": [1, 2, 3]})
df
```

	a	b	c
0	1.0	4.0	1
1	NaN	5.0	2
2	3.0	NaN	3

```
df = df.fillna(method="ffill")
df
```

	a	b	c
0	1.0	4.0	1
1	1.0	5.0	2
2	3.0	5.0	3

pandas.pivot_table: Turn Your DataFrame Into a Pivot Table

A pivot table is useful to summarize and analyze the patterns in your data. If you want to turn your DataFrame into a pivot table, use `pandas.pivot_table`.

```
df = pd.DataFrame(
    {
        "item": ["apple", "apple", "apple", "apple", "apple"],
        "size": ["small", "small", "large", "large", "large"],
        "location": ["Walmart", "Aldi", "Walmart", "Aldi",
"Aldi"],
        "price": [3, 2, 4, 3, 2.5],
    }
)

df
```

	item	size	location	price
0	apple	small	Walmart	3.0
1	apple	small	Aldi	2.0
2	apple	large	Walmart	4.0
3	apple	large	Aldi	3.0
4	apple	large	Aldi	2.5

```
pivot = pd.pivot_table(
    df, values="price", index=["item", "size"], columns=
["location"], aggfunc="mean"
)
pivot
```

	location	Aldi	Walmart
item	**size**		

	location	Aldi	Walmart
item	**size**		
apple	**large**	2.75	4.0
	small	2.00	3.0