Get Values

This section contains some methods to get specific values of a pandas DataFrame or a pandas Series.

DataFrame.columns.str.startswith: Find DataFrame's Columns that Start With a Pattern

To find pandas DataFrame whose columns start with a pattern, use `df.columns.str.startswith`.

```
import pandas as pd

df = pd.DataFrame({'pricel': [1, 2, 3],
                   'price2': [2, 3, 4],
                   'year': [2020, 2021, 2021]})

mask = df.columns.str.startswith('price')
df.loc[:, mask]
```

	pricel	price2
0	1	2
1	2	3
2	3	4

pandas.Series.dt: Access Datetime Properties of a pandas Series

The easiest way to access datetime properties of pandas Series values is to use `pandas.Series.dt`.

```
df = pd.DataFrame({"date": ["2021/05/13 15:00", "2022-6-20
14:00"], "values": [1, 3]})

df["date"] = pd.to_datetime(df["date"])

df["date"].dt.year
```

```
0    2021
1    2022
Name: date, dtype: int64
```

```
df["date"].dt.time
```

```
0    15:00:00
1    14:00:00
Name: date, dtype: object
```

pd.Series.between: Select Rows in a Pandas Series Containing Values Between 2 Numbers

To get the values that are smaller than the upper bound and larger than the lower bound, use the `pandas.Series.between` method.

In the code below, I obtained the values between 0 and 10 using `between`.

```
s = pd.Series([5, 2, 15, 13, 6, 10])

s[s.between(0, 10)]
```

```
0     5
1     2
4     6
5    10
dtype: int64
```

DataFrame rolling: Find The Average of The Previous n Datapoints Using pandas

If you want to find the average of the previous n data points (simple moving average) with pandas, use `df.rolling(time_period).mean()`.

The code below shows how to find the simple moving average of the previous 3 data-points.

```python
from datetime import date

df = pd.DataFrame(
    {
        "date": [
            date(2021, 1, 20),
            date(2021, 1, 21),
            date(2021, 1, 22),
            date(2021, 1, 23),
            date(2021, 1, 24),
        ],
        "value": [1, 2, 3, 4, 5],
    }
).set_index("date")

df
```

	value
date	
2021-01-20	1
2021-01-21	2
2021-01-22	3
2021-01-23	4
2021-01-24	5

```
df.rolling(3).mean()
```

	value
date	
2021-01-20	NaN
2021-01-21	NaN
2021-01-22	2.0
2021-01-23	3.0
2021-01-24	4.0

select_dtypes: Return a Subset of a DataFrame Including/Excluding Columns Based on Their dtype

You might want to apply different kinds of processing to categorical and numerical features. Instead of manually choosing categorical features or numerical features, you can automatically get them by using `df.select_dtypes('data_type')`.

In the example below, you can either include or exclude certain data types using `exclude`.

```
df = pd.DataFrame({"col1": ["a", "b", "c"], "col2": [1, 2, 3],
"col3": [0.1, 0.2, 0.3]})

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   col1    3 non-null      object 
 1   col2    3 non-null      int64  
 2   col3    3 non-null      float64
dtypes: float64(1), int64(1), object(1)
memory usage: 200.0+ bytes
```

```
df.select_dtypes(include=["int64", "float64"])
```

	col2	col3
0	1	0.1
1	2	0.2
2	3	0.3

pandas.Series.pct_change: Find The Percentage Change Between The Current and a Prior Element in a pandas Series

If you want to find the percentage change between the current and a prior element in a pandas Series, use the `pct_change` method.

In the example below, 35 is 75% larger than 20, and 10 is 71.4% smaller than 35.

```
df = pd.DataFrame({"a": [20, 35, 10], "b": [1, 2, 3]})
df
```

	a	b
0	20	1
1	35	2
2	10	3

```
df.a.pct_change()
```

```
0         NaN
1    0.750000
2   -0.714286
Name: a, dtype: float64
```

DataFrame.diff and DataFrame.shift: Take the Difference Between Rows Within a Column in pandas

If you want to get the difference between rows within a column, use `DataFrame.diff()`.

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

	a	b
0	NaN	NaN
1	1.0	1.0
2	1.0	1.0
3	1.0	2.0

This will leave the first index null. You can shift the rows up to match the first difference with the first index using `DataFrame.shift(-1)`.

```
shift = diff.shift(-1)
shift
```

	a	b
0	1.0	1.0
1	1.0	1.0
2	1.0	2.0
3	NaN	NaN

```
processed_df = shift.dropna()
processed_df
```

	a	**b**
0	1.0	1.0
1	1.0	1.0
2	1.0	2.0

pandas.clip: Exclude Outliers

Outliers are unusual values in your dataset, and they can distort statistical analyses.

```
data = {"col0": [9, -3, 0, -1, 5]}
df = pd.DataFrame(data)
df
```

	col0
0	9
1	-3
2	0
3	-1
4	5

If you want to trim values that the outliers, one of the methods is to use `df.clip`.

Below is how to use the 0.5-quantile as the lower threshold and .95-quantile as the upper threshold

```
lower = df.col0.quantile(0.05)
upper = df.col0.quantile(0.95)

df.clip(lower=lower, upper=upper)
```

	col0
0	8.2
1	-2.6
2	0.0
3	-1.0
4	5.0

Get Rows within a Year Range

If you want to get all data starting in a particular year and exclude the previous years, simply use `df.loc['year':]` like below. This works when the index of your `pd.Dataframe` is `DatetimeIndex`.

```
from datetime import datetime

df = pd.DataFrame(
    {
        "date": [datetime(2018, 10, 1), datetime(2019, 10, 1),
datetime(2020, 10, 1)],
        "val": [1, 2, 3],
    }
).set_index("date")

df
```

	val
date	
2018-10-01	1
2019-10-01	2
2020-10-01	3

```
df.loc["2019":]
```

	val
date	
2019-10-01	2
2020-10-01	3

pandas.reindex: Replace the Values of the Missing Dates with 0

Have you ever got a time series with missing dates? This can cause a problem since many time series methods require a fixed frequency index.

To fix this issue, you can replace the values of the missing dates with 0 using `pd.date_range` and `pd.reindex`.

```
s = pd.Series([1, 2, 3], index=["2021-07-20", "2021-07-23",
"2021-07-25"])
s.index = pd.to_datetime(s.index)
s
```

```
2021-07-20    1
2021-07-23    2
2021-07-25    3
dtype: int64
```

```
# Get dates ranging from 2021/7/20 to 2021/7/25
new_index = pd.date_range("2021-07-20", "2021-07-25")

# Conform Series to new index
new_s = s.reindex(new_index, fill_value=0)
new_s
```

```
2021-07-20    1
2021-07-21    0
2021-07-22    0
2021-07-23    2
2021-07-24    0
2021-07-25    3
Freq: D, dtype: int64
```

Select DataFrame Rows Before or After a Specific Date

If you want to get the rows whose dates are before or after a specific date, use the comparison operator and a date string.

```
df = pd.DataFrame(
    {"date": pd.date_range(start="2021-7-19", end="2021-7-23"),
"value": list(range(5))}
)
df
```

	date	value
0	2021-07-19	0
1	2021-07-20	1
2	2021-07-21	2
3	2021-07-22	3
4	2021-07-23	4

```
filtered_df = df[df.date <= "2021-07-21"]
filtered_df
```

	date	value
0	2021-07-19	0
1	2021-07-20	1
2	2021-07-21	2

DataFrame.groupby.sample: Get a Random Sample of Items from Each Category in a Column

If you want to get a random sample of items from each category in a column, use `pandas.DataFrame.groupby.sample`.This method is useful when you want to get a subset of a DataFrame while keeping all categories in a column.

```
df = pd.DataFrame({"col1": ["a", "a", "b", "c", "c", "d"],
"col2": [4, 5, 6, 7, 8, 9]})
df.groupby("col1").sample(n=1)
```

	col1	col2
1	a	5
2	b	6
3	c	7
5	d	9

To get 2 items from each category, use `n=2`.

```
df = pd.DataFrame(
    {
        "col1": ["a", "a", "b", "b", "b", "c", "c", "d", "d"],
        "col2": [4, 5, 6, 7, 8, 9, 10, 11, 12],
    }
)
df.groupby("col1").sample(n=2)
```

	col1	col2
0	a	4
1	a	5
4	b	8
2	b	6

	col1	col2
5	c	9
6	c	10
8	d	12
7	d	11

pandas.Categorical: Turn a List of Strings into a Categorical Variable

If you want to create a categorical variable, use `pandas.Categorical`. This variable takes on a limited number of possible values and can be ordered. In the code below, I use `pd.Categorical` to create a list of ordered categories.

```
import pandas as pd

size = pd.Categorical(['M', 'S', 'M', 'L'], ordered=True,
categories=['S', 'M', 'L'])
size
```

```
['M', 'S', 'M', 'L']
Categories (3, object): ['S' < 'M' < 'L']
```

Note that the parameters `categories = ['S', 'M', 'L']` and `ordered=True` tell pandas that `'S' < 'M' < 'L'`. This means we can get the smallest value in the list:

```
size.min()
```

```
'S'
```

Or sort the DataFrame by the column that contains categorical variables:

```
df = pd.DataFrame({'size': size, 'val': [5, 4, 3, 6]})

df.sort_values(by='size')
```

	size	val
1	S	4
0	M	5
2	M	3
3	L	6