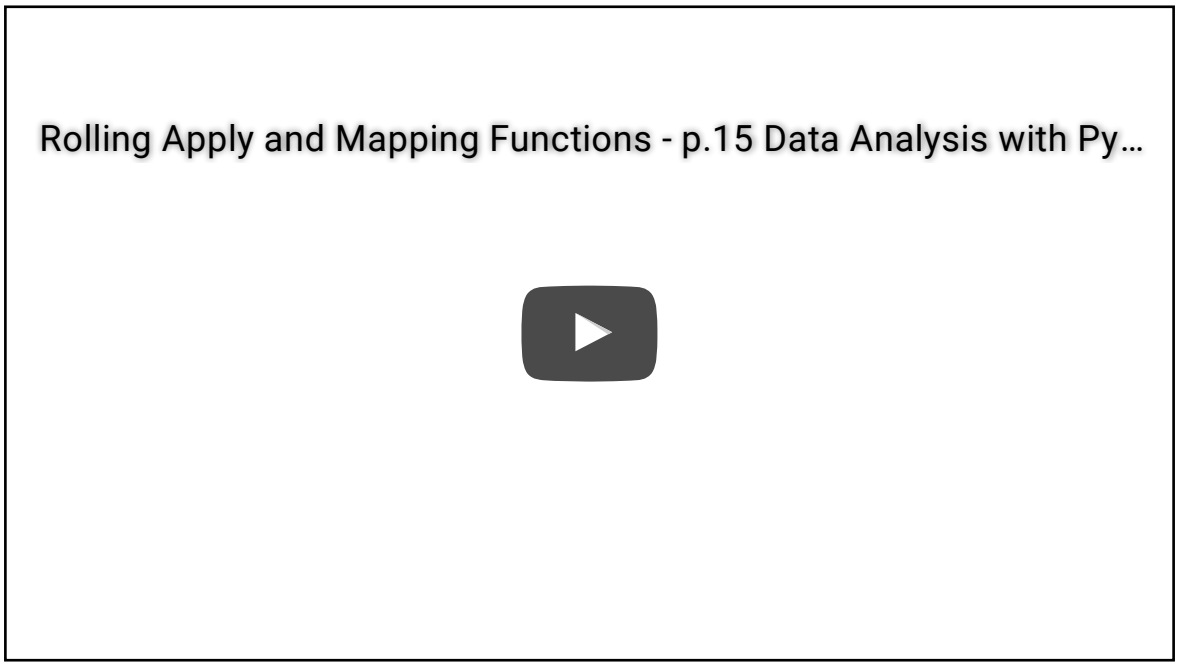
Rolling Apply and Mapping Functions - p.15 Data Analysis with Python and Pandas Tutorial

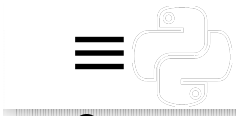


This data analysis with Python and Pandas tutorial is going to cover two topics. First, within the context of machine learning, we need a way to create "labels" for our data. Second, we're going to cover mapping functions and the rolling apply capability with Pandas.

Creating labels is essential for the supervised machine learning process, as it is used to "teach" or train the machine correct answers that are associated with features.

Mapping functions to a Pandas Dataframe is useful, to write custom formulas that you wish to apply to the entire dataframe, a certain column, or to create a new column. If you recall, a while back, we made new columns by doing something like df['Column2'] = df['Column1']*1.5, and so on. If you wanted to create far more logically intense operations, however, you would want to write a function. We will show how to do that.

Since mapping functions is one of the two major ways that users can dramatically customize what Pandas can do, we might as well cover the second major way, which is with rolling_apply. This allows us to do a moving window application of a function. We will just write a moving average function, but you could do just




```
import Quandl
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import numpy as np
from statistics import mean
style.use('fivethirtyeight')

housing_data = pd.read_pickle('HPI.pickle')
housing_data = housing_data.pct_change()
```

First, we're going to lead in the dataset, and then convert all columns to percent change. This will help us to normalize all of the data.

Next:

```
housing_data.replace([np.inf, -np.inf], np.nan, inplace=True)
housing_data['US_HPI_future'] = housing_data['United States'].shift(-1)
```

Here, we replace the infinity values with nan values first. Next, we create a new column, which contains the future HPI. We can do this with a new method: .shift(). This method will shift the column in question. Shifting by -1 means we're shifting down, so the value for the next point is moved back. This is our quick way of having the current value, and the next period's value on the same row for easy comparison.

Next up, we will have some NaN data from both the percent change application and the shift, so we need to do:

```
housing_data.dropna(inplace=True)
```

Now, we want to do some sort of label creation. There are actually a couple of ways that we could do this, but, since this is a tutorial, let's cover function mapping. There is a one-liner that we could do to achieve this same result, but I don't tend to buy the hype on crazy one-liners. To map a function, it's quite simple. You just simply build a function with reasonable logic, like so:

```
def create_labels(cur_hpi, fut_hpi):
    if fut_hpi > cur_hpi:
        return 1
    else:
        return 0
```

Here, we're obviously passing the current HPI and the future HPI columns. If the future HPI is higher than the current, this means prices went up, and we are going to return a 1. This is going to be our label. If the future HPI is not greater than the current, then we return a simple 0. To map this function, we can do something like:

```
housing_data['label'] = list(map(create_labels,housing_data['United States'], hou
```

This might look like a confusing one-liner, but it doesn't need to be. It breaks down to:

```
new_column = list(map( function_to_map, parameter1, parameter2, ... ))
```

That's all there is to it, and you can continue adding more and more parameters.

```
Date
1990-03-31  0.003628  0.062548 -0.000033  0.005578  0.007252  0.000961
1990-04-30  0.006277  0.095081 -0.002126  0.005257  0.005569 -0.000318
1990-05-31  0.007421  0.112105  0.001513  0.005635  0.002409  0.004512
1990-06-30  0.004930  0.100642  0.004353  0.006238  0.003569  0.007884
1990-07-31  0.000436  0.067064  0.003322  0.006173  0.004351  0.004374

                CT        DE        FL        GA  ...          WV        WI  \
Date                                                ...
1990-03-31 -0.009234  0.002786 -0.001259 -0.007290  ...    0.013441  0.015638
1990-04-30 -0.010818  0.000074  0.002675 -0.002477  ...    0.015765  0.015926
1990-05-31 -0.010963 -0.000692  0.004656  0.002808  ...    0.017085  0.012106
1990-06-30 -0.007302 -0.001542  0.003710  0.002857  ...    0.016638  0.010545
1990-07-31 -0.003439 -0.004680  0.003116  0.002276  ...    0.011129  0.009425

                WY  United States       M30  Unemployment Rate       GDP  \
Date
1990-03-31  0.009831       0.004019  0.090909           0.035714 -0.234375
1990-04-30  0.016868       0.004957  0.119048          -0.068966  4.265306
1990-05-31  0.026130       0.005260  0.117021           0.000000 -1.092539
1990-06-30  0.029359       0.005118 -0.304762           0.074074  3.115183
1990-07-31  0.023640       0.003516 -0.164384          -0.103448  0.441476

              sp500  US_HPI_future  label
Date
1990-03-31  0.030790       0.004957      1
1990-04-30 -0.001070       0.005260      1
1990-05-31  0.045054       0.005118      0
1990-06-30  0.036200       0.003516      0
1990-07-31 -0.001226       0.000395      0

[5 rows x 57 columns]
```

Next, let's show a custom way to apply a moving-window function. We're going to just do a simple moving average example:

```
def moving_average(values):
    ma = mean(values)
    return ma
```

There's our function, notice that we just pass the "values" parameter. We do not need to code any sort of "window" or "time-frame" handling, Pandas will handle that for us.

Now, you can use rolling_apply:

```
housing_data['ma_apply_example'] = pd.rolling_apply(housing_data['M30'], 10, mov
```

```
print(housing_data.tail())
```

```
                AL        AK        AZ        AR        CA        CO  \
```

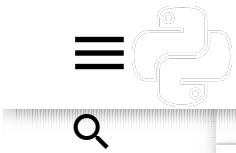
```
2011-09-30 -0.011103 -0.007609  0.003190  0.000505 -0.006537 -0.004569
2011-10-31 -0.013189 -0.007754  0.000541  0.001059  0.005390  0.009231
2011-11-30 -0.008055 -0.006551  0.005119 -0.000856 -0.003570 -0.010812

                 CT        DE        FL        GA        ...         \
Date                                                     ...
2011-07-31 -0.002806 -0.001084 -0.001531 -0.003036        ...
2011-08-31 -0.010243 -0.002133  0.001438 -0.006488        ...
2011-09-30 -0.012240 -0.004171  0.002307 -0.013116        ...
2011-10-31 -0.013075 -0.006204 -0.001566 -0.021542        ...
2011-11-30 -0.012776 -0.008252 -0.006211 -0.022371        ...

                 WI        WY  United States       M30  Unemployment Rate  \
Date
2011-07-31 -0.002068  0.001897      -0.000756 -0.008130           0.000000
2011-08-31 -0.006729 -0.002080      -0.005243  0.057377           0.000000
2011-09-30 -0.011075 -0.006769      -0.007180  0.031008          -0.100000
2011-10-31 -0.015025 -0.008818      -0.008293  0.007519          -0.111111
2011-11-30 -0.014445 -0.006293      -0.008541  0.014925          -0.250000

                GDP     sp500  US_HPI_future  label  ma_apply_example
Date
2011-07-31  0.024865  0.031137      -0.005243      0         -0.003390
2011-08-31  0.022862 -0.111461      -0.007180      0         -0.000015
2011-09-30 -0.039361 -0.010247      -0.008293      0          0.004432
2011-10-31  0.018059  0.030206      -0.008541      0          0.013176
2011-11-30  0.000562  0.016886      -0.009340      0          0.015728

[5 rows x 58 columns]
```


The next tutorial:

Scikit Learn Incorporation - P.16 Data Analysis With Python And Pandas Tutorial

Data Analysis with Python and Pandas Tutorial Introduction

Pandas Basics - p.2 Data Analysis with Python and Pandas Tutorial



Home +=1 Store Community Log in

Sign up

Concatenating and Appending dataframes - p.5 Data Analysis with Python and Pandas Tutorial

Joining and Merging Dataframes - p.6 Data Analysis with Python and Pandas Tutorial

Pickling - p.7 Data Analysis with Python and Pandas Tutorial

Percent Change and Correlation Tables - p.8 Data Analysis with Python and Pandas Tutorial

Resampling - p.9 Data Analysis with Python and Pandas Tutorial

Handling Missing Data - p.10 Data Analysis with Python and Pandas Tutorial

Rolling statistics - p.11 Data Analysis with Python and Pandas Tutorial

Applying Comparison Operators to DataFrame - p.12 Data Analysis with Python and Pandas Tutorial

Joining 30 year mortgage rate - p.13 Data Analysis with Python and Pandas Tutorial

Adding other economic indicators - p.14 Data Analysis with Python and Pandas Tutorial

Rolling Apply and Mapping Functions - p.15 Data Analysis with Python and Pandas Tutorial

You are currently here.

Scikit Learn Incorporation - p.16 Data Analysis with Python and Pandas Tutorial

You've reached the end!

Contact: Harrison@pythonprogramming.net.

Support this Website!
Hire me
Facebook
Twitter
Google+

Legal stuff:

Terms and Conditions
Privacy Policy