Rishab Vaishya

Class - CSE 8392 Special Topics (Advanced Application Programming)

Quest 3 - Pandas 1

SMU ID – 47505527

Example 1 - Option & Settings

https://pandas.pydata.org/pandas-docs/stable/options.html

The API is composed of 5 relevant functions, available directly from the pandas namespace:

1. get_option() / set_option() - get/set the value of a single option.
2. reset_option() - reset one or more options to their default value.
3. describe_option() - print the descriptions of one or more options.
4. option_context() - execute a codeblock with a set of options that revert to prior settings after execution.

gets the value of the opion: max_rows to display of a table

In [187]:

```
import pandas as pd

pd.get_option("display.max_rows")
```

Out[187]:

```
60
```

sets the value of the opion: max_rows to display of a table to 101

In [188]:

```
pd.set_option("display.max_rows",101)

# displays updated value
pd.get_option("display.max_rows")
```

Out[188]:

```
101
```

resets the value of the opion: max_rows to display of a table to default

In [189]:

```
pd.reset_option("display.max_rows")

# displays updated value
pd.get_option("display.max_rows")
```

Out[189]:

```
60
```

Example 2 - Cleaning / Filling Missing Data

https://pandas.pydata.org/pandas-docs/stable/missing_data.html

we are gonna use the

1. replace() which lets use replace any value with desired value
2. fillna() function whic lets you fill all the NaN values with our desired value

In [190]:

```
import numpy  as np

# set up a dataframe
df = pd.DataFrame(np.random.randint(-1,10,size=(5, 10)), columns=list('abcdefghij'))
df
```

Out[190]:

	a	b	c	d	e	f	g	h	i	j
0	3	5	4	2	3	8	7	6	3	8
1	0	4	3	3	1	9	0	5	9	8
2	7	5	0	1	3	2	7	4	0	0
3	9	3	1	-1	3	2	7	5	1	5
4	-1	0	0	7	2	4	4	-1	5	1

replacing value -1 by NaN, to make some NaN values in table, to fill in later

In [191]:

```
df = df.replace(-1, np.nan)
df
```

Out[191]:

	a	b	c	d	e	f	g	h	i	j
0	3.0	5	4	2.0	3	8	7	6.0	3	8
1	0.0	4	3	3.0	1	9	0	5.0	9	8
2	7.0	5	0	1.0	3	2	7	4.0	0	0
3	9.0	3	1	NaN	3	2	7	5.0	1	5
4	NaN	0	0	7.0	2	4	4	NaN	5	1

filling Nan values with -1

In [192]:

```
df.fillna(-1)
```

Out[192]:

	a	b	c	d	e	f	g	h	i	j
0	3.0	5	4	2.0	3	8	7	6.0	3	8
1	0.0	4	3	3.0	1	9	0	5.0	9	8
2	7.0	5	0	1.0	3	2	7	4.0	0	0
3	9.0	3	1	-1.0	3	2	7	5.0	1	5
4	-1.0	0	0	7.0	2	4	4	-1.0	5	1

filling Nan values with 'Missing' string value

In [193]:

```
df.fillna('Missing')
```

Out[193]:

	a	b	c	d	e	f	g	h	i	j
0	3	5	4	2	3	8	7	6	3	8
1	0	4	3	3	1	9	0	5	9	8
2	7	5	0	1	3	2	7	4	0	0
3	9	3	1	Missing	3	2	7	5	1	5
4	Missing	0	0	7	2	4	4	Missing	5	1

filling Nan values with average/mean value of it's column

In [194]:

```
df.fillna(df.mean())
```

Out[194]:

	a	b	c	d	e	f	g	h	i	j
0	3.00	5	4	2.00	3	8	7	6.0	3	8
1	0.00	4	3	3.00	1	9	0	5.0	9	8
2	7.00	5	0	1.00	3	2	7	4.0	0	0
3	9.00	3	1	3.25	3	2	7	5.0	1	5
4	4.75	0	0	7.00	2	4	4	5.0	5	1

filling Nan values with max value of it's column

In [195]:

```
df.fillna(df.max())
```

Out[195]:

	a	b	c	d	e	f	g	h	i	j
0	3.0	5	4	2.0	3	8	7	6.0	3	8
1	0.0	4	3	3.0	1	9	0	5.0	9	8
2	7.0	5	0	1.0	3	2	7	4.0	0	0
3	9.0	3	1	7.0	3	2	7	5.0	1	5
4	9.0	0	0	7.0	2	4	4	6.0	5	1

Example 3 - Time series / Date functionality

https://pandas.pydata.org/pandas-docs/stable/timeseries.html

The following table shows the type of time-related classes pandas can handle and how to create them.

In [196]:

```
df = pd.DataFrame([["Timestamp",'Represents a single timestamp','to_datetime, Timestamp'],
          ["DatetimeIndex",'Index of Timestamp','to_datetime, date_range, bdate_range, DatetimeIndex
'],
          ["Period",'Represents a single time span','Period'],
```

```
        ["PeriodIndex",'Index of Period','period_range, PeriodIndex']],columns=list(["Class",'Rema
rks','How to create']))
df
```

Out[196]:

	Class	Remarks	How to create
0	Timestamp	Represents a single timestamp	to_datetime, Timestamp
1	DatetimeIndex	Index of Timestamp	to_datetime, date_range, bdate_range, Datetime...
2	Period	Represents a single time span	Period
3	PeriodIndex	Index of Period	period_range, PeriodIndex

Create a range of dates starting from midnight of 9/1/2018 & an interval of 1 hour & 72 values

In [197]:

```
timeRange = pd.date_range('9/1/2018', periods=72, freq='H')
# show only first 10
timeRange[:10]
```

Out[197]:

```
DatetimeIndex(['2018-09-01 00:00:00', '2018-09-01 01:00:00',
               '2018-09-01 02:00:00', '2018-09-01 03:00:00',
               '2018-09-01 04:00:00', '2018-09-01 05:00:00',
               '2018-09-01 06:00:00', '2018-09-01 07:00:00',
               '2018-09-01 08:00:00', '2018-09-01 09:00:00'],
              dtype='datetime64[ns]', freq='H')
```

Index pandas objects (df) with dates

In [198]:

```
df = pd.Series(np.random.randn(len(timeRange)), index=timeRange)
df.head()
```

Out[198]:

```
2018-09-01 00:00:00   -1.315265
2018-09-01 01:00:00   -0.524184
2018-09-01 02:00:00   -0.331979
2018-09-01 03:00:00   -0.864254
2018-09-01 04:00:00   -1.536040
Freq: H, dtype: float64
```

Change frequency and fill gaps

In [199]:

```
# to 45 minute frequency and forward fill
converted = df.asfreq('45Min', method='pad')
converted.head()
```

Out[199]:

```
2018-09-01 00:00:00   -1.315265
2018-09-01 00:45:00   -1.315265
2018-09-01 01:30:00   -0.524184
2018-09-01 02:15:00   -0.331979
2018-09-01 03:00:00   -0.864254
Freq: 45T, dtype: float64
```

Resample the series to a daily frequency

In [200]:

```
df.resample('D').mean()
```

Out[200]:

```
2018-09-01    0.065329
2018-09-02    0.110969
2018-09-03   -0.043462
Freq: D, dtype: float64
```

Example 4 - Styling

https://pandas.pydata.org/pandas-docs/stable/style.html

Pass your style functions into one of the following methods:

1. Styler.applymap: elementwise
2. Styler.apply: column-/row-/table-wise

In [201]:

```
# set up a dataframe
np.random.seed(24)
df = pd.DataFrame({'A': np.linspace(1, 10, 10)})
df = pd.concat([df, pd.DataFrame(np.random.randn(10, 4), columns=list('BCDE'))],
               axis=1)
df.iloc[0, 2] = np.nan
```

In [202]:

```
# origina style
df.style
```

Out[202]:

	A	B	C	D	E
0	1	1.32921	nan	-0.31628	-0.99081
1	2	-1.07082	-1.43871	0.564417	0.295722
2	3	-1.6264	0.219565	0.678805	1.88927
3	4	0.961538	0.104011	-0.481165	0.850229
4	5	1.45342	1.05774	0.165562	0.515018
5	6	-1.33694	0.562861	1.39285	-0.063328
6	7	0.121668	1.2076	-0.00204021	1.6278
7	8	0.354493	1.03753	-0.385684	0.519818
8	9	1.68658	-1.32596	1.42898	-2.08935
9	10	-0.12982	0.631523	-0.586538	0.29072

Let's write a simple style function that will color negative numbers red and positive numbers black

In [203]:

```
def color_negative_red(val):
    """
    Takes a scalar and returns a string with
    the css property `'color: red'` for negative
    strings, black otherwise.
    """
    color = 'red' if val < 0 else 'black'
    return 'color: %s' % color
```

In this case, the cell's style depends only on it's own value. That means we should use the Styler.applymap method which works

elementwise.

In [204]:

```
df.style.applymap(color_negative_red)
```

Out[204]:

	A	B	C	D	E
0	1	1.32921	nan	-0.31628	-0.99081
1	2	-1.07082	-1.43871	0.564417	0.295722
2	3	-1.6264	0.219565	0.678805	1.88927
3	4	0.961538	0.104011	-0.481165	0.850229
4	5	1.45342	1.05774	0.165562	0.515018
5	6	-1.33694	0.562861	1.39285	-0.063328
6	7	0.121668	1.2076	-0.00204021	1.6278
7	8	0.354493	1.03753	-0.385684	0.519818
8	9	1.68658	-1.32596	1.42898	-2.08935
9	10	-0.12982	0.631523	-0.586538	0.29072

Now suppose you wanted to highlight the maximum value in each column

In [205]:

```
def highlight_max(s):
    '''
    highlight the maximum in a Series yellow.
    '''
    is_max = s == s.max()
    return ['background-color: yellow' if v else '' for v in is_max]
```

We can't use .applymap anymore since that operated elementwise. Instead, we'll turn to .apply which operates columnwise (or rowwise using the axis keyword). Later on we'll see that something like highlight_max is already defined on Styler so you wouldn't need to write this yourself.

In [206]:

```
df.style.apply(highlight_max)
```

Out[206]:

	A	B	C	D	E
0	1	1.32921	nan	-0.31628	-0.99081
1	2	-1.07082	-1.43871	0.564417	0.295722
2	3	-1.6264	0.219565	0.678805	1.88927
3	4	0.961538	0.104011	-0.481165	0.850229
4	5	1.45342	1.05774	0.165562	0.515018
5	6	-1.33694	0.562861	1.39285	-0.063328
6	7	0.121668	1.2076	-0.00204021	1.6278
7	8	0.354493	1.03753	-0.385684	0.519818
8	9	1.68658	-1.32596	1.42898	-2.08935
9	10	-0.12982	0.631523	-0.586538	0.29072

It's encourage you to use method chains to build up a style piecewise, before finally rending at the end of the chain.

In [207]:

```
df.style.\
    applymap(color_negative_red).\
    apply(highlight_max)
```

Out[207]:

	A	B	C	D	E
0	1	1.32921	nan	-0.31628	-0.99081
1	2	-1.07082	-1.43871	0.564417	0.295722
2	3	-1.6264	0.219565	0.678805	1.88927
3	4	0.961538	0.104011	-0.481165	0.850229
4	5	1.45342	1.05774	0.165562	0.515018
5	6	-1.33694	0.562861	1.39285	-0.063328
6	7	0.121668	1.2076	-0.00204021	1.6278
7	8	0.354493	1.03753	-0.385684	0.519818
8	9	1.68658	-1.32596	1.42898	-2.08935
9	10	-0.12982	0.631523	-0.586538	0.29072

Example 5 - Comparing with SQL

https://pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html

Importing up a dummy table from github for us to use as an example

In [208]:

```
url = 'https://raw.github.com/pandas-dev/pandas/master/pandas/tests/data/tips.csv'
tips = pd.read_csv(url)
tips.head()
```

Out[208]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

5.1 select with Where

how it's done in SQL :

> SELECT *
> FROM tips
> WHERE time = 'Dinner'
> LIMIT 5;

the panda equivalent for it

In [209]:

```
tips[tips['time'] == 'Dinner'].head(5)
```

Out[209]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

5.2 'Update' clause

how it's done in SQL :

> UPDATE tips
> SET tip = tip*2
> WHERE tip < 2;

the panda equivalent for multiplyin the tip by 2 for all tips less than 2

In [210]:

```
tips.loc[tips['tip'] < 2, 'tip'] *= 2
tips.head()
```

Out[210]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	2.02	Female	No	Sun	Dinner	2
1	10.34	3.32	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

5.3 'Delete' clause

how it's done in SQL :

> DELETE FROM tips
> WHERE tip ==2.02;

In pandas we select the rows that should remain, instead of deleting them

In [211]:

```
tips = tips.loc[tips['tip'] != 2.02]
tips.head()
```

Out[211]:

	total_bill	tip	sex	smoker	day	time	size
1	10.34	3.32	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2

	total_bill	tip	sex	smoker	day	time	size
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4

5.4 Full Join

show all records from both tables

```
SELECT *
FROM df1
FULL OUTER JOIN df2
ON df1.key = df2.key;
```

we are are creating 2 tables df1 & df2 onto which we'll perform full outer join

In [212]:

```
df1 = pd.DataFrame({'key': ['A', 'B', 'C', 'D'],'value': np.random.randn(4)})
df1.head()
```

Out[212]:

	key	value
0	A	1.264103
1	B	0.290035
2	C	-1.970288
3	D	0.803906

In [213]:

```
df2 = pd.DataFrame({'key': ['B', 'D', 'D', 'E'],'value': np.random.randn(4)})
df2.head()
```

Out[213]:

	key	value
0	B	1.030550
1	D	0.118098
2	D	-0.021853
3	E	0.046841

In [214]:

```
pd.merge(df1, df2, on='key', how='outer')
```

Out[214]:

	key	value_x	value_y
0	A	1.264103	NaN
1	B	0.290035	1.030550
2	C	-1.970288	NaN
3	D	0.803906	0.118098
4	D	0.803906	-0.021853
5	E	NaN	0.046841