Pandas Tips and Tricks

This post includes some useful tips for how to use Pandas for efficiently preprocessing and feature engineering from large datasets.

Pandas Ufuncs and why they are so much better than apply command

Pandas has an **apply** function which let you apply just about any function on all the values in a column. Note that **apply** is just a little bit faster than a **python for loop**! That's why it is most recommended using pandas builtin **ufuncs** for applying preprocessing tasks on columns (if a suitable **ufunc** is available for your task)**. Ufuncs,** are special functions (based on numpy library) implemented in C and that's why they are highly efficient. Among the useful ufuncs we will mention are: **.diff, .shift, .cumsum, .cumcount, .str commands** (works on strings)**, .dt commands** (works on dates) and many more.

Example data set—summer activities

I will demonstrate the pandas tricks on a made up data set with different people names, their summer activities and their corresponding timestamps. A person can make multiple activities in various timestamps.

	name	activity	timestamp
1	Chandler Bing	party	2017-08-04 08:00:00
6	Harry Kane	football	2017-08-04 13:00:00
0	John Doe	beach	2017-08-04 14:00:00
3	Joey Tribbiani	party	2017-08-04 10:00:00
2	Monica Geller	travel	2017-08-04 07:00:00

Randomly generated data with summer activities

Let's say our goal is to predict, based on the given data set, who is the most fun person in the data set :).

1. String commands

For string manipulations it is most recommended to use the Pandas **stringcommands** (which are **Ufuncs).**

For example, you can split a column which includes the full name of a person into two columns with the first and last name using **.str.split** and expand=True.

	name
4	Ross Geller
2	Monica Geller
6	Harry Kane
4	Ross Geller
3	Joey Tribbiani

Name column before split

```
df['name'] = df.name.str.split(" ", expand=True)
```

	0	1
4	Ross	Geller
2	Monica	Geller
6	Harry	Kane
4	Ross	Geller
3	Joey	Tribbiani

Name column after split

In addition you can clean any string column efficiently using **.str.replace** and a suitable **regex**.

2. Group by and value_counts

Groupby is a very powerful pandas method. You can group by one column and count the values of another column per this column value using **value_counts**.
Using **groupby** and **value_counts** we can count the number of activities each person did.

```
df.groupby('name')['activity'].value_counts()
```

```
name            activity
Chandler Bing   party       3
Harry Kane      party       2
                football    1
Joey Tribbiani  party       2
                football    1
John Doe        beach       3
Monica Geller   travel      3
Phoebe Buffay   travel      2
                football    1
Ross Geller     party       2
                travel      1
dtype: int64
```

Group by person name and value counts for activities

This is multi index, a valuable trick in pandas dataframe which allows us to have a few levels of index hierarchy in our dataframe. In this case the person name is the level 0 of the index and the activity is on level 1.

3. Unstack

We can also create features for the summer activities counts per person, by applying unstack on the above code. **Unstack** switches the rows to columns to get the activity counts as features. By doing **unstack** we are transforming the last level of the index to the columns. All the activities values will now be the columns of a the dataframe and when a person has not done a certain activity this feature will get Nan value. **Fillna** fills all these missing values (activities which were not visited by the person) with 0.

```
df.groupby('name')['activity'].value_counts().unstack().fillna(0)
```

activity	beach	football	party	travel
name				
Chandler Bing	0	0	3	0
Harry Kane	0	1	2	0
Joey Tribbiani	0	1	2	0
John Doe	3	0	0	0
Monica Geller	0	0	0	3
Phoebe Buffay	0	1	0	2
Ross Geller	0	0	2	1

Activity count in columns

3. groupby, diff, shift, and loc + A great tip for efficiency

Knowing the time differences between person activities can be quite interesting for predicting who is the most fun person. How long did a person hang out in a party? how long did he/she hang out at the the beach? This might be useful for us as a feature, depends on the activity.

The most straight forward way to calculate the time differences would be to **groupby** the person name and them calculate the difference on the timestamp field using **diff**():

```
df = df.sort_values(by=['name','timestamp'])
df['time_diff'] = df.groupby('name')['timestamp'].diff()
```

	activity	name	timestamp	time_diff
5	travel	Phoebe Buffay	2017-08-04 12:00:00	02:00:00
3	football	Phoebe Buffay	2017-08-04 15:00:00	03:00:00
4	party	Ross Geller	2017-08-04 09:00:00	NaT
4	party	Ross Geller	2017-08-04 11:00:00	02:00:00
2	travel	Ross Geller	2017-08-04 14:00:00	03:00:00

Calculating the time difference between person activities to get the duration of each activity

If you have a lot of data and you want to save some time (this can be about 10 times faster depends on your data size) you can skip the **groupby** and just do the **diff** after sorting the data and then deleting the first row of each person which is not relevant.

```
df = df.sort_values(by=['name','timestamp'])
df['time_diff'] = df['timestamp'].diff()
df.loc[df.name != df.name.shift(), 'time_diff'] = None
```

BTW — the useful **.Shift** command shift all the column down per one space, so we can see on which row this column is changing by doing this: df.name!=df.name.shift().

And **.loc** command is the most recommended way to set values for a column for specific indices.

To change the time_diff to seconds units:

```
df['time_diff'] = df.time_diff.dt.total_seconds()
```

To get the duration per row:

```
df['row_duration'] = df.time_diff.shift(-1)
```

	activity	name	timestamp	time_diff	money_spent	money_spend_so_far	activity_change	activity_num	row_duration
1	party	Chandler Bing	2017-08-04 08:00:00	NaT	51	51	True	1	05:00:00
1	party	Chandler Bing	2017-08-04 13:00:00	05:00:00	60	111	False	1	02:00:00
1	party	Chandler Bing	2017-08-04 15:00:00	02:00:00	59	170	False	1	NaT
4	party	Harry Kane	2017-08-04 07:00:00	NaT	68	68	True	1	04:00:00
6	party	Harry Kane	2017-08-04 11:00:00	04:00:00	90	158	False	1	02:00:00
6	football	Harry Kane	2017-08-04 13:00:00	02:00:00	80	238	True	2	NaT
0	beach	Jhon Doe	2017-08-04 07:00:00	NaT	63	63	True	1	05:00:00
0	beach	Jhon Doe	2017-08-04 12:00:00	05:00:00	61	124	False	1	02:00:00
0	beach	Jhon Doe	2017-08-04 14:00:00	02:00:00	65	189	False	1	NaT
3	football	Joey Tribbiani	2017-08-04 08:00:00	NaT	84	84	True	1	01:00:00
6	party	Joey Tribbiani	2017-08-04 09:00:00	01:00:00	54	138	True	2	01:00:00
3	party	Joey Tribbiani	2017-08-04 10:00:00	01:00:00	67	205	False	2	NaT
2	travel	Monica Geller	2017-08-04 07:00:00	NaT	90	90	True	1	01:00:00
5	travel	Monica Geller	2017-08-04 08:00:00	01:00:00	96	186	False	1	01:00:00
2	travel	Monica Geller	2017-08-04 09:00:00	01:00:00	74	260	False	1	NaT
5	travel	Phoebe Buffay	2017-08-04 10:00:00	NaT	52	52	True	1	02:00:00
5	travel	Phoebe Buffay	2017-08-04 12:00:00	02:00:00	84	136	False	1	03:00:00
3	football	Phoebe Buffay	2017-08-04 15:00:00	03:00:00	58	194	True	2	NaT
4	party	Ross Geller	2017-08-04 09:00:00	NaT	96	96	True	1	02:00:00
4	party	Ross Geller	2017-08-04 11:00:00	02:00:00	81	177	False	1	03:00:00
2	travel	Ross Geller	2017-08-04 14:00:00	03:00:00	60	237	True	2	NaT

Added duration per row

4. Cumcount and Cumsum

This are two really cool Ufuncs which can help you with many things. Cumcount create a cumulative count. For example we can take only the second activity for each person by grouping by the person name and then applying cumcount. This will just give a count for the activities by their order. Than we can take only the second activity for each person by doing ==1 (or the third activity by doing ==2) and applying the indices on the original sorted dataframe.

```
df = df.sort_values(by=['name','timestamp'])
df2 = df[df.groupby('name').cumcount()==1]
```

	activity	name	timestamp	time_diff
1	party	Chandler Bing	2017-08-04 13:00:00	05:00:00
6	party	Harry Kane	2017-08-04 11:00:00	04:00:00
0	beach	Jhon Doe	2017-08-04 12:00:00	05:00:00
6	party	Joey Tribbiani	2017-08-04 09:00:00	01:00:00
5	travel	Monica Geller	2017-08-04 08:00:00	01:00:00

The second activity of each person

```
df = df.sort_values(by=['name','timestamp'])
df2 = df[df.groupby('name').cumcount()==2]
```

	activity	name	timestamp	time_diff
1	party	Chandler Bing	2017-08-04 15:00:00	02:00:00
6	football	Harry Kane	2017-08-04 13:00:00	02:00:00
0	beach	Jhon Doe	2017-08-04 14:00:00	02:00:00
3	party	Joey Tribbiani	2017-08-04 10:00:00	01:00:00
2	travel	Monica Geller	2017-08-04 09:00:00	01:00:00

The third activity of each person

Cumsum is just a cummulative summary of a numeric cell. For example you can add the money the person spend in each activity as an additional cell and then summarize the money spent by a person at each time of the day using:

```
df = df.sort_values(by=['name','timestamp'])
df['money_spent_so_far'] = df.groupby('name')['money_spent'].cumsum()
```

	activity	name	timestamp	time_diff	money_spent	money_spend_so_far
5	travel	Phoebe Buffay	2017-08-04 10:00:00	NaT	52	52
5	travel	Phoebe Buffay	2017-08-04 12:00:00	02:00:00	84	136
3	football	Phoebe Buffay	2017-08-04 15:00:00	03:00:00	58	194
4	party	Ross Geller	2017-08-04 09:00:00	NaT	96	96
4	party	Ross Geller	2017-08-04 11:00:00	02:00:00	81	177
2	travel	Ross Geller	2017-08-04 14:00:00	03:00:00	60	237

Money spent so far

5. groupby, max, min for measuring the duration of activities

In section 3 we wanted to know how much time each person spent in each activity. But we overlooked that sometimes we get multiple records for an acitivity which is actually the continuance of the same activities. So to get the actual activity duration we should measure the time from the first consecutive activity appearance to the last. For that we need to mark the change in activities and mark each row with the activity number. We would do this using the **.shift** command and the **.cumsum** command we saw before. A new activity is when the activity changes **or** the person changes.

```
df['activity_change'] = (df.activity!=df.activity.shift()) |
(df.name!=df.name.shift())
```

Then we will calculate the activity number for each row by grouping per user and applying the glorious **.cumsum**:

```
df['activity_num'] = df.groupby('name')['activity_change'].cumsum()
```

	activity	name	timestamp	time_diff	money_spent	money_spend_so_far	activity_change	activity_num
5	travel	Phoebe Buffay	2017-08-04 10:00:00	NaT	52	52	True	1
5	travel	Phoebe Buffay	2017-08-04 12:00:00	02:00:00	84	136	False	1
3	football	Phoebe Buffay	2017-08-04 15:00:00	03:00:00	58	194	True	2
4	party	Ross Geller	2017-08-04 09:00:00	NaT	96	96	True	1
4	party	Ross Geller	2017-08-04 11:00:00	02:00:00	81	177	False	1
2	travel	Ross Geller	2017-08-04 14:00:00	03:00:00	60	237	True	2

Add activity num for the activities which continues between rows

Now we can calculate the duration of each activity as follows by grouping per name and activity num (and activity—which doesn't really change the grouping but we need it to have the activity name) and calculating the sum of activity duration per row:

```
activity_duration =
df.groupby(['name','activity_num','activity'])['activity_duration'].sum
()
```

```
name            activity_num  activity
Chandler Bing   1             party      07:00:00
Harry Kane      1             party      06:00:00
                2             football         NaT
Joey Tribbiani  1             football   01:00:00
                2             party      01:00:00
John Doe        1             beach      07:00:00
Monica Geller   1             travel     02:00:00
Phoebe Buffay   1             travel     05:00:00
                2             football         NaT
Ross Geller     1             party      05:00:00
                2             travel           NaT
Name: row_duration, dtype: timedelta64[ns]
```

activity duration

This will return the activity duration in some kind of timedelta type. You could get the session activity duration in seconds using .dt.total_seconds:

```
activity_duration = activity_duration.dt.total_seconds()
```

Then you can the maximal/minimal activity duration for each person (or median or mean) using a command like this:

```
activity_duration =
activity_duration.reset_index().groupby('name').max()
```

	activity_num	activity	row_duration
name			
Chandler Bing	1	party	07:00:00
Harry Kane	2	party	06:00:00
Joey Tribbiani	2	party	01:00:00
John Doe	1	beach	07:00:00
Monica Geller	1	travel	02:00:00
Phoebe Buffay	2	travel	05:00:00
Ross Geller	2	travel	05:00:00

Maximal activity duration per user

Summary

This was a short Pandas tour using a summer activities made-up dataset. Hope you've learned and enjoy it. Good luck with your next Pandas project and enjoy the summer!