Python pandas quick guide

Shiu-Tang Li

May 12, 2016

Contents

1 Dataframe initialization / outputs

1.1 Load csv files into dataframe.

```
import pandas
data_frame = pandas.read_csv("C:/Users/Shiu-Tang Li/...csv",
                             encoding = "ISO-8859-1")
# encoding: to deal with unicodes
```

1.2 Initialize a dataframe

```
import pandas as pd
df1 = pd.DataFrame = ({'c1':['1','2','3','4'], 'c2':['5','6','7','8']})
df2 = pd.DataFrame({ 'c1' : 2,
                     'c2' : np.array([0] * 100,dtype='int32'),
                     'c3' : 'hello'})
```

```
import pandas as pd
list_of_dicts = [{'column1':3,'column2':4}, {'column1':7,'column2':2}, {'column1
    ':6,'column2':8}]
data_frame = pd.DataFrame(list_of_dicts,index=['index1','index2'])
# construct data frame from list of dictionaries
```

1.3 Create a new column

```
data_frame['new_column'] = List OR Series
# will get warning message
```

1.4 Output a dataframe to csv

```
import pandas
data_frame.to_csv("C:/Users/Shiu-Tang Li/...csv")
```

Remark. May load .csv as list of lists instead of data frames.

2 Take a quick glance of a dataframe

2.1 Print the data frame

```
print(data_frame.head(5))
# print first 5 rows
print(data_frame)
# print the data frame, dimension information is also attached
```

2.2 Get the description of numerical columns

```
print(data_frame.describe())
```

2.3 Get the dimension

```
dim = data_frame.shape
number_of_rows = dim[0]
number_of_columns = dim[1]
```

2.4 Get the data type / get the filtered data by data type

```
types = data_frame.dtypes
# types is a Series labeled by column names, showing the data type for each
    column.
integer_index = types[types == int64].index
# types[types == int64] is a filtered Series with integer values.
print(data_frame[integer_index])
# this gives us the filtered data frame with only int64 values.
```

2.5 Get the unique elements

```
print(data_frame['column_name'].unique())
# will return a list showing distinct elements in the column
print(data_frame['column_name'].value_counts())
# will return a table showing the counts in the column
```

3 Select data from a dataframe

3.1 Get column names

```
print(data_frame.columns)
# data_frame.columns is a list of strings
first_column = data_frame.columns[0]
# print the fist column, which is a string
```

3.2 Select a specific column

```
column = data_frame['column_name']
# column is a [Series] object, contains row index + values, both are lists
column_values = column.values
column_index = column.index
```

3.3 Select the sub-dataframe of a few columns

```
data_frame2 = data_frame[['column_name1', 'column_name2']]
data_frame2 = data_frame[data_frame.column[0:2]]
# select the first two columns in two different ways
data_frame2 = data_frame['column_name_x':'column_name_y']
# select the columns between the two columns
```

3.4 Select rows with restrictions on columns

```
data_frame[data_frame['column_name'] == some_values]
```

3.5 Select rows with row index

```
data_frame.iloc[i]
#i: row index
data_frame.iloc[0:3]
# select the rows with indices 0,1,2
```

Remark. The difference between loc and iloc: If the index of the dataframe is 3, 7, 0, 2, ..., iloc[0] will select the third row (true integer index), loc will select the 1st row (index by locations).

3.6 Select row index with max values in a specific column

```
data_frame['column_name'].idxmax()
# returns the 1st row index that has max
```

3.7 Select given entry

```
# Approach 1:  i: true row index
data_frame.iloc[i]['column_name']
# Approach 2:  i: location
data_frame['column_name'].values[i]
# Approach 3:  i: true row index
data_frame.ix[i,'column_name']
```

3.8 Iterate rows

```
for i, row in data_frame.iterrows():
# i: row indices; row: each row
```

4 Revise data in a dataframe

4.1 Revise data in a particular entry

```
# i: true row index
# Approach 1 (will get warning message):
data_frame.ix[i, 'column_name'] = new_value
# Approach 2 (will get warning message):
data_frame['column_name'][i] = new_value
# Approach 3:
data_frame.set_value(i, 'column_name', new_value)
# Approach 4:
data_frame.at[i, 'column_name'] =  new_value
```

4.2 Reindex rows

```
data_frame.index = [index1, index2, ...]
# replace indices with a new list
data_frame = data_frame.set_index(['column_name'])
# indexed by a particular column
data_frame = data_frame.reset_index(drop=True)
# reindexed from 0. drop=False: make a dataframe column with the old index
    values.
```

4.3 Reindex one row

```
old_indices = data_frame.index
new_indices  = old_indices.values
for i,item in enumerate(new_indices):
    if item == 'old_index_to_be_changed':
        new_indices[i] = 'new_index'
data_frame.index = new_indices
```

4.4 Rename columns

```
data_frame.rename(columns={'column_name1':'new_column_name1','column_name2':'
    new_column_name2'}, inplace=True)
# rename a few columns: will get warning message
data_frame.columns = ['column_name1','column_name2',...]
# rename all columns
```

4.5 Drop columns / rows

```
data_frame.drop('column_name', axis=1, inplace=True)
# drop a column
data_frame.drop(['column_name1','column_name2',...], axis=1, inplace=True)
# drop a few columns
data_frame.drop('row_index1', axis=0, inplace=True)
# drop a row
```

```
data_frame.drop(['row_index1','row_index2',...], axis=0, inplace=True)
# drop a few rows
```

4.6 Find / drop / fill missing values

```
import pandas as pd
is_null = pd.isnull(data_frame["column_name"])
# will return a true / false Series. Null value = NaN.
new_data_frame = data_frame.dropna()
# drop all rows with missing values
new_data_frame = data_frame.dropna(axis = 1)
# drop all columns with missing values
new_data_frame = data_frame.dropna(subset=["column1", "column2"])
# drop all rows with missing values in the two columns
data_frame["column"] = data_frame["column"].fillna(data_frame["column"].median()
    )
data_frame["column"] = data_frame["column"].fillna(data_frame["column"].mean())
data_frame["column"] = data_frame["column"].fillna(something)
# fill NaN values with column median / mean / or sth else
```

4.7 Data frame transpose

```
data_frame.T
```

4.8 Change types of a column

```
data_frame['column_name'] = data_frame['column_name'].astype(float)
# changing types to float
```

4.9 Merge data frames

```
import pandas as pd
list1 = [{'c1':4,'c2':3}, {'c1':2,'c2': 3}]
list2 = [{'c1':6,'c2':9}, {'c1':8,'c2':10}]
df1 = pd.DataFrame(list1 ,index=[0,1])
df2 = pd.DataFrame(list2 ,index=['two','three'])
df = pd.concat([df1,df2])
# combine two data frames with common columns, increase rows. index could
    contain keys of different data types
```

```
import pandas as pd
df1 = pd.DataFrame({'key': ['a', 'b'], 'c1': [1, 2], 'c2': [8, 2]})
df2 = pd.DataFrame({'key': ['a', 'b'], 'c3': [3, 5]})
merged_df = pd.merge(df1, df2, on='key')
# combine two data frames with the same data in the column 'key'. Merged data
    frame will contain 4 columns.
```

5 Search key words in a dataframe

5.1 Exact match in target column

```
for i, row in movies.iterrows():
    if re.search ("Keywords", row["column"])!= None:
        print(i)
```

6 Perform operations on a dataframe

6.1 Sort dataframe

```
data_frame.sort("column_name", inplace=True, ascending=False)
# sort by column, in decreasing order
data_frame.sort_index(inplace=True, ascending=False)
# sort by index, in decreasing order
```

6.2 Rearrange dataframe - pivot table

```
import numpy as np
new_data_frame = data_frame.pivot_table(index="column1", values="column2",
    aggfunc=np.mean)
# For different column1 values, find the result of aggfunction applied to
    column2.
new_data_frame = data_frame.pivot_table(index="column1", values="any_column",
    aggfunc='count')
# count the frequency for column1.
```

Remark. Other choices of aggfunc: 'max', 'min', np.std, np.sum, np.median.

6.3 Grouping

```
groups = data_frame.groupby("column1")
table = groups.aggregate(np.mean)["column2"]
# classify rows into different groups based on 'column1', for each group apply
    np.mean() on all values in "column2".
```

6.4 'Apply' function

	c1	c2	c3	apply2
r1	-	-	-	ap2
r2	-	-	-	ap2
r3	-	-	-	ap2
apply1	ap1	ap1	ap1	

```
apply1 = data_frame.apply(some_function) OR
apply1 = data_frame.apply(lambda x: some_function(x))
# apply function to columns, to get a Sereis object labeled column names
apply2 = data_frame.apply(some_function, axis=1) OR
apply2 = data_frame.apply(lambda x: some_function(x), axis=1)
# apply function to rows, to get a Sereis object labeled row indices
```

7 Others

Series.tolist() converts Series to lists.
numpy.nan: missing value