Chapter 1 – Import Data, Select Cases and Variables, Save DataFrame

---

Tutorials for Sarbanes-Oxley Paper Data

Dan Neely (from University of Milwaukee-Wisconsin) and I just had the following article published at the *Journal of Business Ethics*:
Saxton, G. D., & Neely, D. G. (2018). The Relationship Between Sarbanes–Oxley Policies and Donor Advisories in Nonprofit Organizations. *Journal of Business Ethics.*
This page contains tutorials on how to download the IRS 990 e-file data that was used for the control variables in our study.

Tutorials

- IRS 990 e-File Data (part 1) – Set up AWS CLI credentials and grab index files
- IRS 990 e-File Data (part 2) – Load Index Data and Insert into MongoDB
- IRS 990 e-File Data (part 3) – Load Index Files into PANDAS
- IRS 990 e-File Data (part 4) – Download IRS 990 Filings into MongoDB
- IRS 990 e-File Data (part 5) – Download IRS 990 Filings

and Schedules into MongoDB
- IRS 990 e-File Data (part 6) – Download IRS 990, 990PF, and 990EZ Filings into MongoDB
- IRS 990 e-File Data (part 7) – Generate Data Codebook
- IRS 990 e-File Data (part 8) – Data Wrangling and Export to Stata

I hope you have found this helpful. If so, please spread the word, and happy coding!

Python PANDAS Code Bytes

This page contains brief (generally one-liner) blocks of code for working with Python and PANDAS for data analytics. I created it as a handy reference for PANDAS commands I tended to forget when I was learning. I hope it proves useful to you, too! I also have a page with longer data analytics tutorials.

Table of Contents

Jupyter Notebook Settings

Set width of columns for display:

```
pd.set_option('display.max_columns', None)
```

Set cell width:

```
pd.set_option('max_colwidth', 150)
```

Working with Python Lists

Break list into chunks of 4 (needed for some APIs, for example)

```
mylist_chunked = [mylist[i:i+4] for i in xrange(0, len(mylist), 4)]
```

Finding duplicates in a list:

```
import collections
mylist = ['twitter', 'facebook','instagram', 'twitter']
[item for item, count in collections.Counter(mylist).items() if count > 1]
```

Remove list element:

```
mylist.remove('revenue')
```

Working with Python Dictionaries

Delete a key from a dictionary:

```
del dict[key]['key_to_delete']
```

Create sub-dictionary (from sub-set of keys):

```
subdict = {k: dict[k] for k in ('key1', 'key2')}
```

Analysis

Cross-tabulation:

```
pd.crosstab(df['var1'], df['var2'])
```

Generating New Variables, Arrays, etc.

Create list from dataframe column:

```
screen_names = df[‘screen_name’].tolist()
```

Create list of unique column values in DF:

```
tickers_in_df = pd.unique(df.ticker.ravel()).tolist()
```

Convert string variable to float:

```
df[‘count_V2’] = df[‘count’].convert_objects(convert_numeric=True)
```

Convert float column to int (only works if there are no missing values):

```
df_2014[‘rt_count’] = df_2014[‘rt_count’].astype(int)
```

Convert column to string:

```
df.index = df.index.astype(str)
```

Create new variable as a slice of an existing one:

```
df[‘cusip8’] = df[‘cusip’].apply(lambda x: x[:8])
```

Replace a word *within* a column with another word:

```
df[‘8-K’] = df[‘8-K’].str.replace(‘Item’, ‘Section’)
```

Fill in missing values for one column with zero:

```
df_2014[‘rt_count’] = df_2014[‘rt_count’].fillna(0)
```

Get new list of unique items in a list:

```
screen_names_unique = list(set(screen_names))
```

Create dummy variable based on whether another column contains specific text (values will be 'True' and 'False'):

```
df[‘retweeted_status_dummy’] =
df[‘retweeted_status’].str.contains(‘THIS’, na=False)
```

Then convert to float (will convert 'True' and 'False' categories of above variable into '1' and '0', respectively):

```
df[‘retweeted_status_dummy’] =
df[‘retweeted_status_dummy’].astype(float)
```

Replace values (here, replace 'None' with '0'):

```
df[‘reply_message’] =
df[‘in_reply_to_screen_name’].replace([None], [‘0’])
```

Replace values (here, replace np.nan values with '0'):

```
df.ix[df.Number_of_retweets.isnull(),
‘Number_of_retweets’] = 0
```

Switch values of '0' and '1':

```
df[‘reply_message’] = df[‘reply_message’].replace([1],
[99]).replace([0],[1]).replace([99],[0])
```

Create binary variable from count variable (if old var=0, assign value of 0; otherwise 1):

```
df[‘urls’] = np.where(df[‘entities_urls_count’]==0, 0, 1)
```

Change each unicode element in a list to string:

```
tickers_in_df = [x.encode('UTF8') for x in tickers_in_df]
```

Change column values to upper case:

```
df['Name'] = df['Name'].apply(lambda x: x.upper())
```

Change column values to upper case:

```
sec['8-K Item'] = sec['8-K Item'].str.upper()
```

Find number of unique values:

```
len(pd.unique(compustat_2013.cusip.ravel()))
```

Add leading zeros to string variable (as many as needed to reach 10 characters):

```
df['cik_x'] = df['cik_x'].apply(lambda x: x.zfill(10))
```

Convert column to string and add leading zeros:

```
df['CIK_10']  =  df['CIK'].astype(str).apply(lambda  x: x.zfill(10))
```

Get a list of values from a DF column:

```
values_list = df['8-K'].value_counts().keys().tolist()
```

Find number of cases with fewer than the mean value:

```
sum(df['followers_count'] < df['followers_count'].mean())
```

I/O

Read in a pickled dataframe:

```
ticker_master = pd.read_pickle('valid_tickers_317.pkl')
```

Read in JSON file:

```
f = open('valid_tickers_list_317.json', 'r')
valid_tickers = simplejson.load(f)
```

Read in JSON file -- method 2:

```
with open('my_list.json', 'r') as fp:
    my_list = json.load(fp)
```

Save a list as JSON file:

```
f = open('all_valid_screen_names.json', 'w')
simplejson.dump(valid_twitter_accounts, f)
f.close()
```

Save a list as JSON file -- method 2:

import json

```
with open('my_list.json', 'w') as fp:
    json.dump(my_list, fp)
```

Read in Excel file:

```
twitter_accounts = pd.read_excel('Twitter Account-level
Database – non-excluded tickers and accounts only.xlsx',
'accounts', header=0)
```

Write Excel file:

```
df.to_excel('df.xls', sheet_name='Sheet1′)
```

Looping

Looping over rows (note how to select a slice of rows):

```
for index, row in df[:10].iterrows():
        print index, row['content']
```

Loop over rows and update existing variable:

```
for index, row in df.iterrows():
        df['count'] = count
```

Loop over rows and create new variable, method 1:

```
for index, row in df.iterrows():
        #df.ix[df.index==index, 'count'] = count #LONGER VERSION
       df.ix[index, 'count'] = count #SHORTER VERSION
```

Loop over rows and create new variable, method 2:

```
for index, row in df.iterrows():
       df.loc[index,'count'] = count
```

Time Series

Weekdays:

```
weekdays_only = by_day2[by_day2['weekday'] < 5 ]
weekday_count = df.groupby(ts.index.weekday).apply(f)
by_day3['weekday'] = by_day['date'].apply(lambda x: x.weekday())
```

Add missing days (with zeros) for every day in a dataframe:

```
df_filled = df.unstack().fillna(0).stack()
```

Change specific columns

```
df_filled.loc[df_filled['Number of Firm Tweets'] == 0, ['Number of Firm Followers (start)', 'Number of Lists for Firm (start)']] = np.nan
```

Set column to datetime:

```
df['created_at'] = pd.to_datetime(df['created_at'])
```

Convert datetime column to date:

```
df['date'] = [t.date() for t in df['datetime']]
```

Generate lagged variable with multi-index DF:

```
df['Mentions [t-1]'] = df['Mentions'].unstack().shift(1).stack()
```

Generate variable aggregated over 3-day window lagged one day:

```
df['Mentions [sum of t-1:t-3]'] = pd.rolling_sum(df['Mentions'].unstack().shift(), window=3).stack()
```

Select date range for DF:

```
df_2014 = df['2014-1-1':'2014-12-31']
```

Indexing and Sorting

Set the index:

```
df = df.set_index(['created_at'])
```

Reset Index:

```
df.reset_index(inplace=True)
```

Set Multi-Index:

```
df = df.set_index(['ticker', 'day'])
```

Sort dataframe:

```
df.sort(['ticker', 'day'], inplace=True)
```

Name Existing Multi-index columns:

```
df.index.names = ['day', 'ticker']
```

With multi-index df -- get unique items per index level:

```
len(firm_day_counts_firm_tweets.index.levels[0])
len(firm_day_counts_firm_tweets.index.levels[1])
```

Swap levels on multi-index dataframe:

```
df_swapped = df.swaplevel('ticker', 'day')
```

Get minimum date in DF:

```
df.index.min()
```

Complex conditional row selection during loop:

```
if (row[‘Form Type’] == ‘8-K’) and (row[‘8-K Item’]==”) and (row[‘8-K Item – V2’]==”) or (row[‘8-K Item’]==’Item that’) or (row[‘8-K Item’]==’Item fo’):
```

Missing Data

Interpolation with backfill and forward fill (n.b. - does not respect multi-index):

```
df[‘F_x)’]=df[‘F’].interpolate(method=’linear’).bfill().ffill()
```

Find rows with empty column:

```
missing = df[df[‘ticker’].isnull()]
```

Fill missing values in one column with values from another column:

```
sec.loc[sec[‘8-K  Item_v2′].isnull(),’8-K  Item_v2’]  = sec[‘8-K Item’]
```

Custom Functions

Function for generating a series of new one-day lag variables in a dataframe:

```
def lag_one_day(df):
    #df_cols = df.columns
     df_cols = [u’Number of Firm Mentions’, u’Number of Firm Tweets’]
    for i in df_cols:
        col = str(i)+str(‘[t-1]’)
        if ‘[t-1]’ not in str(i):
            df[col] = df[i].unstack().shift(1).stack()
```

Function for generating a series of new dataframe variables that aggregate over a multi-day period:

```
def lag_one_day(df):
    #df_cols = df.columns
     df_cols = [u'Number of Firm Mentions', u'Number of Firm Tweets']
    for i in df_cols:
        col = str(i)+str('[sum t-1:t-3]')
        if '[t-1]' not in str(i):
                                                df[col]   = pd.rolling_sum(df[i].unstack().shift(),
            window=3).stack()
```

DataFrame Manipulations

Subset of DF -- based on a condition in a column:

```
df[df['reply_message'] == 1]
```

Subset of DF -- specific columns:

```
df[['in_reply_to_screen_name',  'in_reply_to_status_id', 'reply_message']].head()
```

Drop a column:

```
df = df.drop('entities_urls',1)
```

Intersection of two lists:

```
pd.Series(np.intersect1d(pd.Series(tickers_in_df), pd.Series(valid_tickers)))
```

Difference between two lists (all different elements from either list):

```
set(tickers_in_df).symmetric_difference(valid_tickers_with_twitter)
```

Difference between two lists (elements from list1 missing from list2):

```
set(list1) – set(list2)
```

Create DF based on whether column value is in a list:

```
df = df[df.ticker.isin(mylist)]
```

Creat an empty dataframe:

```
columns = [‘ticker’, ‘month’, ‘degree’]

df = pd.DataFrame(columns=columns)
```

Add row (row 0) to empty dataframe:

```
df.loc[0]  =  pd.Series({‘ticker’:’Z’,  ‘month’:’12’,
‘degree’: ”})
```

Change cell column values for a specific row (index=16458):

```
sec.ix[16458, “8-K Item – V2”] = ‘Item 2.01’
```

Create dataframe from list/dictionary:

```
months = [‘2014-1’, ‘2014-2’, ‘2104-3’]

data = {‘month’: months}
df = pd.DataFrame(data, columns=[‘month’, ‘degree’])
```

Add rows to a dataframe:

```
df = df.append(df_month)
```

Create dataframe from list of column names:

```
d = {‘Variable Name’: list(df.columns.values)}
variables = DataFrame(d)
```

Create dataframe by deleting all rows with null values in one column:

```
df2 = df[df[‘keep’].notnull()]
```

Rename a single column:

```
df.rename(columns={'cusip':'cusip_COMPUSTAT'},
inplace=True)
```

Create dataframe based on column value appearing in a list:

```
missing = df[df['cusip'].isin(mylist)]
```

Look for duplicates in a dataframe:

```
counts = merged.groupby('cusip').size()
df2 = pd.DataFrame(counts, columns = ['size'])
df2 = df2[df2.size>1]
df2
```

Create version of dataframe with conditions on two variables (for removing a duplicate firm):

```
merged = merged[~((merged['fyear']==2012) &
(merged['gvkey']=='176103'))]
```

Select partial dataframe -- complex conditions:

```
sec[(sec['Form Type'] == '8-K') & (sec['8-K Item']==") &
(sec['8-K Item – V2']==") | (sec['8-K Item']=='Item fo') &
(sec['8-K Item – V2']==")| (sec['8-K Item']=='Item that')
& (sec['8-K Item – V2']==")]
```

Merge two dataframes:

```
merged = pd.merge(fb, org_data, left_on='feed_id',
right_on='Org_ID')
```

Deep copy of DF:

```
df2 = df.copy(deep=True)
```

Get a slice of dataframe (here, the two rows with the given indexes):

```
df.loc[11516:11517]
```

Custom Twitter Variables

Time on Twitter:

```
import datetime
df['start']=np.datetime64('2016-04-01')
df['created_at'] = pd.to_datetime(df['created_at'])
df['time_on_twitter'] = df['start'] - df['created_at']
df['time_on_twitter_days']                              =
df['time_on_twitter'].astype('timedelta64[D]')
df['time_on_twitter_days']                              =
df.time_on_twitter_days.astype('int')
```

Working with MongoDB in PANDAS

Show first 2 documents:

```
for user in users.find()[:2]:
    print user, '\n'
```

Get frequency counts:

```
from bson.son import SON
pipeline = [ {"$group": {"_id": "$FormType", "count":
{"$sum": 1}}} ]
list(file_list.aggregate(pipeline))
```

Loop over random sample of 100 with filtering:

```
for file in file_list.aggregate([
    { '$match': {'FormType': { '$in': ['990PF']}} },
    { '$sample': {'size': 100} }
    ]):
    print file
```

I hope you have found this helpful. If so, please spread the word, and happy coding!

Using Your Twitter API Key

Below is an embedded version of an iPython notebook I have made publicly available on nbviewer. To download a copy of the code, click on the icon with three horizontal lines at the top right of the notebook (just below this paragraph) and select "Download Notebook." I hope you find it helpful. If so, please share, and happy coding!

Analyzing Big Data with Python PANDAS

This is a series of iPython notebooks for analyzing Big Data – specifically Twitter data – using Python's powerful PANDAS (Python Data Analysis) library. Through these tutorials I'll walk you through how to analyze your raw social media data using a typical social science approach.
The target audience is those who are interested in covering key steps involved in taking a social media dataset and moving it through the stages needed to deliver a valuable research product. I'll show you how to import your data, aggregate tweets by organization and by time, how to analyze hashtags, how to create new variables, how to produce a summary statistics table for publication, how to analyze audience reaction (e.g., # of retweets) and, finally, how to run a logistic regression to test your hypotheses. Collectively, these tutorials cover essential steps needed to move from the data collection to the research product stage.
Prerequisites
I've put these tutorials in a GitHub repository called PANDAS. For these tutorials I am assuming you have already downloaded some data and are now ready to begin examining it. In the

first notebook I will show you how to set up your ipython working environment and import the Twitter data we have downloaded. If you are new to Python, you may wish to go through a series of tutorials I have created in order.

If you want to skip the data download and just use the sample data, but don't yet have Python set up on your computer, you may wish to go through the tutorial "Setting up Your Computer to Use My Python Code".

Also note that we are using the iPython notebook interactive computing framework for running the code in this tutorial. If you're unfamiliar with this see this tutorial "Four Ways to Run your Code".

For a more general set of PANDAS notebook tutorials, I'd recommend this cookbook by Julia Evans. I also have a growing list of "recipes" that contains frequently used PANDAS commands.

As you may know from my other tutorials, I am a big fan of the free Anaconda version of Python 2.7. It contains all of the prerequisites you need and will save you a lot of headaches getting your system set up.

Chapters:

At the GitHub site you'll find the following chapters in the tutorial set:

Chapter 1 – Import Data, Select Cases and Variables, Save DataFrame.ipynb

Chapter 2 – Aggregating and Analyzing Data by Twitter Account.ipynb

Chapter 3 – Analyzing Twitter Data by Time Period.ipynb

Chapter 4 – Analyzing Hashtags.ipynb

Chapter 5 – Generating New Variables.ipynb

Chapter 6 – Producing a Summary Statistics Table for Publication.ipynb

Chapter 7 – Analyzing Audience Reaction on Twitter.ipynb

Chapter 8 – Running, Interpreting, and Outputting Logistic Regression.ipynb

I hope you find these tutorials helpful; please acknowledge the source in your own research papers if you've found them

useful:

Saxton, Gregory D. (2015). *Analyzing Big Data with Python.* Buffalo, NY: http://social-metrics.org

Also, please share and spread the word to help build a vibrant community of PANDAS users.

Happy coding!

Producing a Summary Statistics Table in iPython using PANDAS

Below is an embedded version of an iPython notebook I have made publicly available on nbviewer. To download a copy of the code, click on the icon with three horizontal lines at the top right of the notebook (just below this paragraph) and select "Download Notebook." I hope you find it helpful. If so, please share, and happy coding!

---

iPython Notebook and PANDAS Cookbook

More and more of my research involves some degree of 'Big Data' – typically datasets with a million or so tweets. Getting these data prepped for analysis can involve massive amounts of data manipulation – anything from aggregating data to the daily or organizational level, to merging in additional variables, to generating data required for social network

analysis. For all such steps I now almost exclusively use Python's PANDAS library ('Python Data Analysis Library'). In conjunction with the iPython Notebook interactive computing framework and NetworkX, you will have a powerful set of analysis tools at your disposal.
Given that I am now doing almost all of my dataset manipulation – and much of the analysis – in PANDAS, and how new I am to the framework, I created this page mostly as a handy reference for all those PANDAS commands I tend to forget or find particularly useful. But if it proves helpful to any others, great!

iPython Notebook Settings

Set width of columns for display:

```
pd.set_option('display.max_colwidth', 200)
```

Set cell width:

```
pd.set_option('max_colwidth',200)
```

Analysis

Cross-tab (can be saved as dataframe):

```
pd.crosstab(df['8-K filing_info'], df['8-K Item'])
```

Generating New Variables, Arrays, etc.

Create list from dataframe column:

```
screen_names = ticker_master['screen_name'].tolist()
```

Create list of unique column values in DF:

```
tickers_in_df = pd.unique(df.ticker.ravel()).tolist()
```

Convert string variable to float:

```
df['count_V2'] =
df['count'].convert_objects(convert_numeric=True)
```

Convert float column to int (only works if there are no missing values):

```
df_2014['rt_count'] = df_2014['rt_count'].astype(int)
```

Convert column to string:

```
df.index = df.index.astype(str)
```

Create new variable as a slice of an existing one:

```
df['cusip8'] = df['cusip'].apply(lambda x: x[:8])
```

Replace a word *within* a column with another word:

```
df['8-K'] = df['8-K'].str.replace('Item', 'Section')
```

Fill in missing values for one column with zero:

```
df_2014['rt_count'] = df_2014['rt_count'].fillna(0)
```

Get new list of unique items in a list:

```
screen_names_unique = list(set(screen_names))
```

Create dummy variable based on whether another column contains specific text (values will be 'True' and 'False'):

```
df['retweeted_status_dummy'] =
df['retweeted_status'].str.contains('THIS', na=False)
```

Then convert to float (will convert 'True' and 'False' categories of above variable into '1' and '0', respectively):

```
df['retweeted_status_dummy'] =
```

```
df['retweeted_status_dummy'].astype(float)
```

Replace values (here, replace 'None' with '0'):

```
df['reply_message']                                    =
df['in_reply_to_screen_name'].replace([None], ['0'])
```

Replace values (here, replace np.nan values with '0'):

```
df.ix[df.Number_of_retweets.isnull(),
'Number_of_retweets'] = 0
```

Switch values of '0' and '1':

```
df['reply_message']  =  df['reply_message'].replace([1],
[99]).replace([0],[1]).replace([99],[0])
```

Create binary variable from count variable (if old var=0, assign value of 0; otherwise 1):

```
df['urls'] = np.where(df['entities_urls_count']==0, 0, 1)
```

Change each unicode element in a list to string:

```
tickers_in_df = [x.encode('UTF8') for x in tickers_in_df]
```

Change column values to upper case:

```
df['Name'] = df['Name'].apply(lambda x: x.upper())
```

Change column values to upper case:

```
sec['8-K Item'] = sec['8-K Item'].str.upper()
```

Find number of unique values:

```
len(pd.unique(compustat_2013.cusip.ravel()))
```

Add leading zeros to string variable (as many as needed to reach 10 characters):

```
df['cik_x'] = df['cik_x'].apply(lambda x: x.zfill(10))
```

Convert column to string and add leading zeros:

```
df[‘CIK_10’] = df[‘CIK’].astype(str).apply(lambda x: x.zfill(10))
```

Get a list of values from a DF column:

```
values_list = df[‘8-K’].value_counts().keys().tolist()
```

Find number of cases with fewer than the mean value:

```
sum(df[‘followers_count’] < df['followers_count'].mean())
```

I/O

Read in a pickled dataframe:

```
ticker_master = pd.read_pickle(‘valid_tickers_317.pkl’)
```

Read in JSON file:

```
f = open(‘valid_tickers_list_317.json’, ‘r’)
valid_tickers = simplejson.load(f)
```

Read in JSON file – method 2:

```
with open(‘my_list.json’, ‘r’) as fp:
    my_list = json.load(fp)
```

Save a list as JSON file:

```
f = open(‘all_valid_screen_names.json’, ‘w’)
simplejson.dump(valid_twitter_accounts, f)
f.close()
```

Save a list as JSON file – method 2:

import json

```
with open(‘my_list.json’, ‘w’) as fp:
    json.dump(my_list, fp)
```

Read in Excel file:

```
twitter_accounts = pd.read_excel('Twitter Account-level Database – non-excluded tickers and accounts only.xlsx', 'accounts', header=0)
```

Write Excel file:

```
df.to_excel('df.xls', sheet_name='Sheet1')
```

Looping

Looping over rows (note how to select a slice of rows):

```
for index, row in df[:10].iterrows():
        print index, row['content']
```

Loop over rows and update existing variable:

```
for index, row in df.iterrows():
        df['count'] = count
```

Loop over rows and create new variable, method 1:

```
for index, row in df.iterrows():
         #df.ix[df.index==index, 'count'] = count #LONGER VERSION
        df.ix[index, 'count'] = count #SHORTER VERSION
```

Loop over rows and create new variable, method 2:

```
for index, row in df.iterrows():
        df.loc[index,'count'] = count
```

Time Series

Weekdays:

```
1]
```

```
df2
```

Create version of dataframe with conditions on two variables (for removing a duplicate firm):

```
merged = merged[~((merged['fyear']==2012) & (merged['gvkey']=='176103')) ]
```

Select partial dataframe – complex conditions:

```
sec[(sec['Form Type'] == '8-K') & (sec['8-K Item']==") & (sec['8-K Item – V2']==") | (sec['8-K Item']=='Item fo') & (sec['8-K Item – V2']==")| (sec['8-K Item']=='Item that') & (sec['8-K Item – V2']==")]
```

Merge two dataframes:

```
merged = pd.merge(fb, org_data, left_on='feed_id', right_on='Org_ID')
```

Deep copy of DF:

```
df2 = df.copy(deep=True)
```

Get a slice of dataframe (here, the two rows with the given indexes):

```
df.loc[11516:11517]
```

Custom Twitter Variables

Time on Twitter:

```
import datetime
```

```
df['start']=np.datetime64('2016-04-01')
```

```
df['created_at'] = pd.to_datetime(df['created_at'])
```

```
df['time_on_twitter'] = df['start'] – df['created_at']
```

```
df[‘time_on_twitter_days’] =
df[‘time_on_twitter’].astype(‘timedelta64[D]’)
```

```
df[‘time_on_twitter_days’] =
df.time_on_twitter_days.astype(‘int’)
```