CAS Python Session II

Stephen J. Mildenhall

Created 2020-07-21 22:01:50.845762

1 CAS 2020 Python Workshop: Session II Pandas

1.1 Session Descriptions

Welcome to CAS Python Workshop

No	Date	Lead	Contents
1	July 15	BF	Python programming basics variables, types, lists, dictionaries, functions, dates, strings, dir, help Simulated transactional data, computing Earned Premium (see 5)
2	**July 22**	**SM**	**Pandas 1: DataFrame creation and basic data manipulation; make a triangle, make development factors, make an exhibit from the CAS Loss Reserve Database**
3	July 29	BF	Pandas 2: data io with external sources: Excel, CSV, markdown, HTML, web; advanced data manipulation: querying, merging, indexes, stack, unstack, pivot-table, tidydata Prem and loss simulated data…
4	Aug 5	SM	Pandas 3: Visualization and Reporting plotting plus matplotlib, geopandas, jinja, COVID data, NY Auto data
5	Aug 12	SM	Simulation modeling, pandas, numpy, scipy.stats Cat model Creating data for session 1
6	Aug 19	BF	Linear regression, lm, glm, sklearn Triangles analysis

1.2 Session II Agenda: `pandas`

- Recall from Session I: lists, dictionaries, functions
- Two handy Python user-defined functions
- `pandas` Introduction
- Creating DataFrames and Accessing Elements
- Extracting information from DataFrames
- Plotting: Bar Chart, Scatter Plot, Histograms
- Web data access
- Grouping and Aggregation
- Stacking and Pivoting
- Triangles…

1.2.1 Reference: Functions We Will Discuss

- `DataFrame`, `Series`
- `head`, `tail`
- `unique`, `value_counts`

- `read_csv`
- `loc`, `slices`, `xs`
- `query`
- `pivot`, `stack` and `unstack`
- `pivot_table`
- `groupby` (.groups, .get_group, as_index)
- `sum`, `mean`, `std` etc.
- `aggregate`
- `plot`

1.3 Recall from Session 1: lists and indexing

```
a = [1,2,3,4,6]
```

```
a[1], a[3:], a[-2], a[-2:], a[::-1]
```

1.3.1 Custom functions

```
def myfunction(x):
    return x * 10
```

```
myfunction(20)
```

1.3.2 Dictionaries and comprehensions

Count letters in a sentence with a dictionary comprehension. Remember dictionaries are {key: value} pairs.

```
s = "jack and jill went up the hill to fetch a pail of water "
```

```
{ i : s.count(i) for i in set(s) }
```

1.3.3 Custom functions, default arguments

```
def letter(s, omit=''):
    return { i : s.count(i) for i in s if i not in omit }
```

```
letter(s), letter(s, ' aeiou')
```

1.3.4 Exact same function counts words(!!!)

`split` breaks a string into words.

```
print(s.split())
```

```
letter(s.split())
```

1.4 Handy Utility Functions

- `dir`: what can a function do?
- `sdir`: better version of `dir`
- `all_doc`: all the documentation on a function

```
dir(str)
```

1.4.1 (a) What Can a Function Do?

There is no distinction between a variable, data and a function. All equal citizens to Python.

```
def sdir(x, colwidth=80):
    """
    Directory of useful elements, wrapped
    """
    from textwrap import fill

    # all the work is in this line:
    l = [i for i in dir(x) if i[0] != '_']

    # frills to printout nicely
    mx = max(map(len, l))
    mx += 2
    fs = f'{{:<{mx:d}s}}'
    l = [fs.format(i) for i in l if i[0] != '_']
    print(fill('\t'.join(l), colwidth))
```

```
sdir(str)
```

1.4.2 (b) Get all the Help

`?function` or `help(function)` shows the help on a function. Custom functions can have help: the string immediately after the declaration.

```
def all_doc(obj):
    """
    print the documentation on every public callable method of obj
    """

    s = f'{str(obj)} (type={type(obj)}) Documentation'
    print(f'{s}\n{"="*len(s)}\n\ntype={type(obj)})\n\nMETHODS\n=======\n')

    # iterate over methods
    for x in dir(obj):
        if x[0] != '_':
            # get the method
            method = getattr(obj, x)
            if callable(method):
                # if it is callable, i.e. a function, show help
                # help lives in obj.__doc__
                print(f'{x}\n{"~"*len(x)}\n{method.__doc__}\n')
```

```
all_doc(bytes)
```

1.5 The Zen of Python

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

1.6 Module 1: Introduction

- Libraries for today: `numpy` (`np`), `pandas` (`pd`), `matplotlib`, `matplotlib.pyplot` (`plt`)
- `np.random`, `rand`, `lognormal`, `choice`, `poisson`
- Select from list `np.random.choice(list('ABCDE'), 10)`
- Select from list, non-uniform prior `np.random.choice(list('ABCDE'), 10, p=[.4,.3,.2,.05,.05])`
- Dictionaries and dictionary comprehension: count letters in a sentence, `f = {i: s.count(i) for i in set(s)}`; apply to random selection using `''.join()` or convert to `list`
- Start by importing, very important!

```
import numpy as np

r_letters = np.random.choice(list('ABCDE'), 10)

r_unif = np.random.rand(10)

r_lognorm = np.random.lognormal(10, .2, 10)

r_letters, r_unif, r_lognorm
```

1.6.1 Exercise

- Simulate random letters from ABCDEF…
- Summarize by letter and check you get distribution you expect, convert sample to list using `list(...)`
- Add a prior distribution
- Summarize again and check you get distribution you expect

1.6.2 solutions to exercise

```
letters = list(np.random.choice(list('ABCDE'), 500, p=[.4,.3,.2,.05,.05]))
n = len(letters)

freq = { i: letters.count(i) / n for i in 'ABCDE'}

freq
```

1.7 Module 2: Create a DataFrame and Access Elements

Finally, Pandas: your spreadsheet in Python.

1.7.1 Creating a DataFrame

- Create from dictionary: keys become column names.
- Create from list of lists
- Allows mixed data types. workd
- Nice Jupyter Lab output.
- Row and column indexes in bold
- Again, start with import!

```
import pandas as pd

df  = pd.DataFrame({'a': range(100, 110),
                    'b' : np.random.choice(list('ABCDEF'), 10),
                    'c' : np.random.rand(10),
                    'd': pd.to_datetime('2020/07/05')+pd.to_timedelta(np.arange(10), unit='D')
                    })
df
```

1.7.2 Accessing Data within a DataFrame

- Access column as item and attribute
- Access row or element using `loc` for row, both
- Access with logic: `df.c < .25`, `query`
- Slicing with `loc`, `df.loc[1:4, 'a':'c']` **includes endpoints**; no well defined notion of the *one before* the end
- Integer indexing `iloc`
- `query`
- `display` vs. `print`; intermediate results vs. final result

```
df['a'], df.a
```

1.7.3 Accessing Data within a DataFrame: Row Index

```
df.loc[3]
```

1.7.4 Accessing Data Within a DataFrame: Row and Column Index

```
df.loc[3, 'd']
```

1.7.5 Accessing Data Within a DataFrame: Range of Rows

```
df.loc[:3]
```

1.7.6 Accessing Data Within a DataFrame: Range of Rows

```
df.iloc[::2]
```

1.7.7 Accessing Data Within a DataFrame: Logic

```
df.a < 105
```

1.7.8 Accessing Data Within a DataFrame: Logic

```
df.loc[df.a < 105]
```

1.7.9 Accessing Data Within a DataFrame: The Query Operator

- Very powerful, very fast
- SQL like
- Access elements with @

```
df.query(' .4 < c < .8 ')
```

1.7.10 Add Data

- Create new columns with math, from old columns
- Create new row
- Can't create on the fly like tidyverse

```
df['E'] = df.a / df.c
```

```
df.loc[100, :] = (110, 'Z', .11223344, pd.to_datetime('2020/11/03'), np.nan)
```

```
display(df)
```

1.7.11 Exercise

Add a column F equal to E * c, check it equals a

Remember everything is case sensitive!

1.7.12 Solution

```
df['F'] = df.E * df.c
```

```
df.F == df.a
```

. . . wait, what?

1.7.13 Sorting

- `sort_values` and `sort_index`: return a new object; `ascending=False` for descending order

```
df.sort_values('c')
```

1.7.14 Exercise

- Create function to take a string, make lower case, break it into words, and create a DataFrame with columns `word` and `freq` counting word frequency
- Reconsider your approach if you go beyond five lines of code...
- *Optionally* make case independent, default arguments `case=False` argument
- Optionally sort output by descending freq
- Extra credit: strip out punctuation

1.7.15 Solution

```
def word_count(s):
    """
    always document here!
    """
    word_list = s.lower().split()
    df = pd.DataFrame([[i, word_list.count(i)] for i in set(word_list)], columns=['word', 'freq'])
    return df

def word_count_ex(s, excluded_chars='",\';:()[]!?@#$%&=\\.'):
    """"
    word counter with excluded characters
    """
    for i in excluded_chars:
        s = s.replace(i, ' ')
    # which is kinda yucky
    word_list = s.lower().split()
    df = pd.DataFrame([[i, word_list.count(i)] for i in set(word_list)], columns=['word', 'freq'])
    df['letters'] = df.word.str.len()
    df = df.sort_values('letters', ascending=False)
    return df
```

- Apply to `In[xx]`

1.8 Module 3: Requests (interlude) and Graphics

1.8.1 Read Longer Document

- read longer document, *The Declaration of Independence* (`di`)
- `requests` library for Internet calls
- create data frame of word count etc., using previous function

```
import requests
```

```
r = requests.get('http://www.mynl.com/RPM/di.txt')
```

```
di = r.text
```

```
print(di[:100])

df = word_count_ex(di)

df
```

1.8.2 Exercise

Sort `df` by descending frequency

1.8.3 Solution

Often helpful to only show `head` or `tail`

```
df.sort_values('freq', ascending=False).head(10)
```

1.8.4 Exercise

Show five most common words that occur at least 10 times and that have five or more letters, sorted descending order by frequency.

Note use of \ for line continuation; nothing can appear after it!

Indentation after first `df` is free-form.

1.8.5 Solution

```
df.query(' freq >= 10 and letters >= 5 '). \
    sort_values('freq', ascending=False). \
    head(5)
```

1.8.6 Graphics! The Bar Chart

- Bar chart of word freq
- `bar` for vertical and `barh` for horizontal
- Subset to longer words using Exercise
- Breakdown `set_index` statement

```
bit = df.query(' freq >= 5 and letters >= 5 '). \
    sort_values('freq', ascending=False). \
    head(10). \
    set_index('word')

display(bit)
bit.plot(kind='bar', rot=315)
```

1.8.7 Just Plot Frequency, Not Letters

```
bit['freq'].plot(kind='barh')
```

1.8.8 [x] vs [[x]] is the Same as R

- `bit['freq']` returns a Pandas `Series` object
- `bit[['freq']]` returns a Pandas `DataFrame` object

```
display(bit['freq'])
display(bit[['freq']])
```

1.8.9 Scatter Plot

- Scatter plot: frequency vs. number of letters

```
df.plot(kind='scatter', x='letters', y='freq', marker='o', alpha=0.4)
```

1.8.10 Exercise

- Jitter number of letters and re-plot, i.e., add a new column equal to the number of letters plus a small random number
- Explore alpha, different markers, e.g., `'x'`, change marker size `s=2`

1.8.11 Solutions

`lw` sets the line width

```
df['letters_j'] = df.letters + np.random.rand(len(df)) * .8 - 0.4
df.plot(kind='scatter', x='letters_j', y='freq', marker='x', alpha=0.4)
df.plot(kind='scatter', x='letters_j', y='freq', marker='x', s=10, lw=.25, alpha=0.8)
```

1.8.12 Nicer Plots and Plot Decorations

```
%config InlineBackend.figure_format = 'svg'

ax = df.plot(kind='scatter', x='letters_j', y='freq', marker='x', s=10, lw=1)
ax.grid()
ax.set(title='My Title', xlabel='(Jitterd) Number of Letters', ylabel='Word Frequency')
```

1.8.13 Extended Exercise

- Create data frame with [100+] claims, loss=lognormal(10,1), kind=randomly selected from A-E, open=random 0,1
- 30% chance claim closed (`choice` or `np.random.binomial(1, 0.3, n)`)‘
- Name index claim_index
- Create new column log_loss using np.log()
- `df = pd.DataFrame({'loss': something, ... })`
- Extra credit: make the mean vary by kind

1.8.14 Solution to extended exercise

```
n = 1000
df = pd.DataFrame({'loss': np.random.lognormal(10,1,n),
                   'kind': np.random.choice(list('ABCDE'), n),
                   'open': np.random.binomial(1, 0.3, n)})
df.loc[df.kind=="A", "loss"] *= 1.95
df.loc[df.kind=="B", "loss"] *= 1.45
df.loc[df.kind=="D", "loss"] *= 0.95
df.loc[df.kind=="E", "loss"] *= 0.65
df['log_loss'] = np.log(df.loss)
df.head(10)
```

1.9 Module 4: Grouping and More Charting

1.9.1 Histograms

- Histogram of claims
- `ec` = edge color, puts nice border around bars
- `bins` determines number of bins or bin boundaries

```
df.log_loss.hist(bins=50, ec='white', lw=0.5)
```

1.9.2 Grouping

- Grouping: `group_by` breaks DataFrame into groups
- Apply a function
- `agg` to summarize
- Summary functions include mean, std etc.

```
df.groupby('kind').mean()
```

1.9.3 Exercise

Are the claim relativities correct?

1.9.4 Solution

```
g = df.groupby('kind').mean()

display(g / g.loc['C'])

print('\n\nBetter\n')
g.loss / g.loc['C', 'loss']
```

1.9.5 Grouping and Aggregating

- Very flexible processing for applying different functions

```
stat_fns =  [np.size, np.mean, np.max]

df.groupby('kind').agg(stat_fns)
```

1.9.6 Flexible Application of Aggregation Functions

```
df.groupby('kind').agg({'loss': stat_fns, 'log_loss': stat_fns[1:] } )
```

1.9.7 Grouping by Two Variables

- First make the open claims more interesting
- Only apply to the loss variable

```
df.loc[df.open==1, 'loss'] *= 0.9

g = df.groupby(['kind', 'open'])['loss'].agg([np.size, np.mean, np.max, np.std])

g
```

1.9.8 What is the Group By Object?

- Pull off name and group variables separately

```
for g, x in df.groupby(['kind', 'open']):
    display(g)
    display(x.head())
```

1.9.9 Hence We Can Play Games Like

```
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 8})

f, axs = plt.subplots(2, 5, sharex=True, sharey=False, constrained_layout=True,
    figsize=(8, 3))
axi = iter(axs.flat)

for g, x in df.groupby(['open', 'kind']):
    ax = x.log_loss.hist(bins=20, ax=next(axi))
    ax.set(title=f"{g[1]}: {'Open' if g[0] else 'Closed'}")
```

1.9.10 New Index and Data Transformation

- Go back to our `g` double grouped data frame
- Usual tidyverse spread (unstack) and gather (stack)
- Stack / unstack from shelving and unshelving books

```
g = df.groupby(['kind', 'open'])['loss'].agg([np.size, np.mean, np.max, np.std])
```

```
display(g)
```

```
g.unstack(1)
```

1.9.11 Stack Different Dimension

- `df.T` for transpose also available

```
g.unstack(1).stack(0)
```

1.9.12 Access the Indices

- Note the names for the levels
- Row index is an example of a `MultiIndex`

```
print(g.columns)
```

```
g.index
```

1.9.13 Exercise

Determine the maximum and minimum claim size by kind and open/closed indicator. Display by kind as rows.

1.9.14 Solution

```
df.groupby(['kind', 'open'])['loss'].agg([np.min, np.max]).unstack(1)
```

1.9.15 Stylin'

```
df.groupby(['kind', 'open'])['loss'].agg([np.min, np.max]).unstack(1).\
    style.format('{:,.1f}')
```

1.9.16 More Stylin'

```
df.groupby(['kind', 'open'])['loss'].agg([np.min, np.max]).\
    unstack(1).style.format('{:,.1f}').\
    background_gradient(subset=[('amax', 0)], cmap='viridis_r').\
    bar(color='#FFA07A', vmin=0, subset=[('amin', 0)], align='zero').\
    set_caption('An Over-Produced DataFrame')
```

1.10 Module 5: The CAS Loss Reserve Database

- Read CAS loss reserve database (extract)
- Automatically read csv file from URL
- Add some helpful columns
- Summarize

```
df = pd.read_csv(r'http://www.mynl.com/RPM/masterdata.csv')
df['LR'] = df.UltIncLoss / df['EarnedPrem']
df.loc[:, 'PdLR'] = df.PaidLoss / df.loc[:, 'EarnedPrem']
 # some company names for future use
sfm = 'State Farm Mut Grp'
amg = 'American Modern Ins Grp Inc'
eix = 'Erie Ins Exchange Grp'
fmg = 'Federated Mut Grp'
wbi = 'West Bend Mut Ins Grp'
vnl = 'Vanliner Ins Co'
df.head().T
```

1.10.1 What Does the DataFrame Contain?

- 10 years development for 10 accident years 1988-97
- Six lines of business
- Variety of companies

```
print(df.columns)
print('\n\n')
for c in ['AY', 'DY', 'Lag', 'Line']:
    print(c, df[c].unique(), '\n')

for c in ['AY', 'DY', 'Lag', 'Line']:
    print(c, df[c].value_counts(), '\n')
```

1.10.2 Summarizing The Data

- If **not** analyzing triangles need `Lag==10` subset to avoid double counting!
- Let's give more meaningful index

```
dfl = df.query(' Lag == 10 ').copy()

dfl = dfl.set_index(['GRName', 'AY', 'Lag', 'Line'], drop=True)
```

```
dfl = dfl.drop('GRCode', axis=1)
```

```
dfl.head()
```

1.10.3 Accessing Chunks

- `sfm` defined earlier to be State Farm Mut Grp

```
dfl.xs([sfm, 'Comm Auto'], axis=0, level=[0,3])
```

1.10.4 Group by with MultiIndex

```
dfl.groupby(level=[1,3])[['UltIncLoss', 'EarnedPrem']].sum().unstack(1)
```

1.10.5 Exercise

- Compute weighted average ultimate loss ratio by line by year

1.10.6 Solution

```
s = dfl.groupby(level=[1,3])[['UltIncLoss', 'EarnedPrem']].sum()
s['LR'] = s.UltIncLoss / s.EarnedPrem
s = s['LR'].unstack(1)
s.style.format('{:.1%}')
```

1.11 Module 6: Make A Triangle!

- Standard `pivot_table` functionality
- Extol virtues of zero-based arrays, `lag` starts at 0

```
bigCos =[sfm, amg, eix, fmg, wbi, vnl]
df['Lag'] -= 1
bit = df.query(f' GRName in @bigCos ').\
    pivot_table(index=['GRName', 'Line', 'AY'], columns='Lag', values='PaidLoss')
bit.tail(20)
```

1.11.1 Link Ratios

- Use integer indexing…

```
trg = bit.xs((vnl, 'Comm Auto'))
display(trg)
link = trg.iloc[:, 1:] / trg.iloc[:, :-1]
link
```

- Index-awareness is *usually* helpful!

1.11.2 Link Ratios… Correctly

- `to_numpy()` or `values` converts into an array, drops index information
- Pick up column index from denominator—retains zero base

```
link = trg.iloc[:, 1:].to_numpy() / trg.iloc[:, :-1]
link
```

1.11.3 Trim-Up to an Historical Triangle

- Compute all the triangles at once…
- Drop 1997 year

```
bit = df.query(f' GRName in @bigCos and Lag + AY <= 1997 ').pivot_table(index=['GRName', 'Line', 'AY'],
link = bit.iloc[:, 1:].to_numpy() / bit.iloc[:, :-1]
link.drop(1997, axis=0, level=2).head(19)
link.tail(20)
```

1.12 THE END