Python Programming for Data Science

Week 41, Monday

More Pandas

More Pandas

Pandas popquiz

What do we know about pandas so far:

- What is the difference between a pandas Series and a numpy array?
- What is the difference between a pandas Series and a pandas DataFrame?
- How do I read data from a file into a pandas dataframe?

pandas - Indexing into a dataframe

General - also supports slicing, etc (see later slide):

Get by label:	`df.loc[row_label, col_label]`
Get by index:	`df.iloc[row_index, col_index]`

Faster - for lookup of single values:

Get by label:	`df.at[row_label, col_label]`
Get by index:	`df.iat[row_index, col_index]`

pandas - Other ways to index into a dataframe

Indexing directly into a dataframe:

Get column by label:	`df[col_label]`	→Series
Slice rows by index range:	`df[start:end]`	→DataFrame

This can be a bit confusing. Recommendation: stick with `loc`, `iloc`, `at`, `iat`

pandas - `.loc`: selecting by labels

The `.loc` attribute is the primary access method

- It selects **by label!**
- Ranges are allowed: all labels between **and including** both endpoints are included

```
np_array =
np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array, index=
['a', 'b', 'c'],
                  columns=['col1',
'col2'])
print(df)


# note that both endpoints are
included
print(df.loc['a':'b'])
```

```
output



   col1  col2
a     0     1
b     2     3
c     4     5

   col1  col2
a     0     1
b     2     3
```

pandas - `.loc`: selecting by labels (2)

You can select along both rows and columns with `.loc`

```
np_array =
np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array, index=
['a', 'b', 'c'],
                  columns=['col1',
'col2'])
print(df)



print(df.loc['a':'b', :'col1'])
```

```
output


   col1  col2
a     0     1
b     2     3
c     4     5

   col1
a     0
b     2
```

Q: What would happen if I removed the ':' before 'col1'?
A: It returns a series

pandas dataframes: selecting with a boolean array (1)

```
np_array =
np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array,
                  index=['a',
'b', 'c'],
                  columns=
['col1','col2'])
print(df)

# A boolean array is sometimes
# called a mask
mask = (df > 2)
print(mask)

# You can index with a mask
# to operate on the entries of
# interest - e.g. assign to
them
df[mask] = 0

print(df)
```

output

```
   col1  col2
a     0     1
b     2     3
c     4     5

    col1   col2
a  False  False
b  False   True
c   True   True


   col1  col2
a     0     1
b     2     0
c     0     0
```

pandas dataframes: selecting with a boolean array (2)

We can also create a mask for the rows or columns

```
np_array =
np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array,
                  index=['a',
'b', 'c'],
                  columns=
['col1','col2'])
print(df)

# Check which rows in the first
col
# have a value larger than 2
row_mask = df.iloc[:,0] > 2

print(row_mask)


# Now, select only those rows
for
# which the mask is true
new_df = df.loc[row_mask,:]
print(new_df)
```

```
output



   col1  col2
a     0     1
b     2     3
c     4     5


a    False
b    False
c     True
Name: col1, dtype: bool


   col1  col2
c     4     5
```

pandas - DataFrame loc/iloc - Exercise

Last week, we read the british-english wordlist into a pandas dataframe:

```
df = pd.read_table('british-english',
keep_default_na=False, header=None)
df.columns = ['words']
```

Let's look a bit more at this dataframe

1. Print out the first word in the 'words' column?
2. Figure out how to select all rows starting with the letter "A"

If you need the british-english file, you can download it from: **https://wouterboomsma.github.io/ppds2023/data/british-english**.

pandas - DataFrame loc/iloc - Exercise - solution

1. Print out the first word in the 'words' column?

```
print(df.loc[0, 'words'])
```

or

```
print(df.at[0, 'words'])
```

2. Figure out how to select all rows starting with the letter A

```
df.loc[df['words'].str.startswith('A')]
```

or

```
df.loc[df['words'].str[0] == 'A']
```

`df['words']` is a series

pandas - DataFrames - Adding columns

Add a new column by assigning to the relevant label

```
np_array = np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array, index=['a', 'b', 'c'], columns=
['col1', 'col2'])
df.loc[:,'col3'] = df.loc[:,'col1']   # Adding 'col3' - copy
of `col1`
```

...or use the `.assign()` method

```
df = df.assign(col4 = df.loc[:,'col1'])
```

...or `.insert()` (inserts at a specific location)

```
df.insert(0, 'col0', df.loc[:,'col1'])
```

pandas - DataFrames - Adding columns (2)

If columns don't have the expected index, missing elements will be set to NaN

```
np_array =
np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array, index=
['a', 'b', 'c'], columns=['col1',
'col2'])
print(df)

# Copying col1, but skipping first
row
df.loc[:,'col3'] =
df.loc['b':,'col1']
print(df)
```

```
output

   col1  col2
a     0     1
b     2     3
c     4     5

   col1  col2  col3
a     0     1   NaN
b     2     3   2.0
c     4     5   4.0
```

pandas - DataFrames - Adding rows

Add a new row by assigning to the relevant label

```
np_array =
np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array,
                  index=['a',
'b', 'c'],
                  columns=
['col1', 'col2'])
# Adding row labeled 'd' - copy
of `a`
df.loc['d',:] = df.loc['a',:]
print(df)
```

```
   col1  col2
a   0.0   1.0
b   2.0   3.0
c   4.0   5.0
d   0.0   1.0
```

pandas - DataFrames - data alignment

When operating with two dataframes, pandas aligns on both row and column labels

```
np_array =
np.arange(9).reshape((3,3))
df1 = pd.DataFrame(np_array)
df2 = pd.DataFrame(np_array[:2,:2])
print(df1)



print(df2)



print(df1+df2)
```

output

```
   0  1  2
0  0  1  2
1  3  4  5
2  6  7  8

   0  1
0  0  1
1  3  4

     0    1   2
0  0.0  2.0 NaN
1  6.0  8.0 NaN
2  NaN  NaN NaN
```

pandas - DataFrames - descriptive statistics (1)

Long list of standard operations: `.mean()`, `.std()`, `.var()`, `.min()`, `.max()`, `.sumsum()`, `.sumprod()`, ...
These functions generally take an `axis` argument, and a `skipna` argument (which default to True)

```
np_array =
np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array,
                  index=['a',
'b', 'c'],
                  columns=
['col1', 'col2'])
print(df)



print(df.mean(axis=0))
```

```
   col1  col2
a     0     1
b     2     3
c     4     5

col1    2.0
col2    3.0
dtype: float64
```

pandas - DataFrames - descriptive statistics (2)

Changing the `axis`

```
np_array =
np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array,
                  index=['a',
'b', 'c'],
                  columns=
['col1', 'col2'])
print(df)



print(df.mean(axis=0))


print(df.mean(axis=1))
```

```
   col1  col2
a     0     1
b     2     3
c     4     5

col1    2.0
col2    3.0
dtype: float64

a    0.5
b    2.5
c    4.5
dtype: float64
```

Missing values (1)

As we've seen, pandas will insert NaN for missing values
Unlike in numpy, pandas deals gracefully with missing values

```
np_array =
np.arange(9).reshape((3,3))
df1 = pd.DataFrame(np_array)
df2 = pd.DataFrame(np_array[:2,:2])
print(df1+df2)



print((df1+df2).sum(axis=0))



print((df1+df2).sum(axis=0,
min_count=1))
```

```
output



     0    1   2
0  0.0  2.0 NaN
1  6.0  8.0 NaN
2  NaN  NaN NaN

0     6.0
1    10.0
2     0.0
dtype: float64

0     6.0
1    10.0
2     NaN
dtype: float64
```

Use `min_count` to specify the minimum number of non-NaN values.

Missing values (2)

You can detect missing values using `.isna()` and `.notna()`

```
np_array =
np.arange(9).reshape((3,3))
df1 = pd.DataFrame(np_array)
df2 =
pd.DataFrame(np_array[:2,:2])
print(df1+df2)



print((df1+df2).isna())
```

```
output



     0    1   2
0  0.0  2.0 NaN
1  6.0  8.0 NaN
2  NaN  NaN NaN

       0      1     2
0  False  False  True
1  False  False  True
2   True   True  True
```

Missing values (3)

You can replace missing values using `.fillna()`

```
np_array =
np.arange(9).reshape((3,3))
df1 = pd.DataFrame(np_array)
df2 =
pd.DataFrame(np_array[:2,:2])
print(df1+df2)



print((df1+df2).fillna(0.0))
```

```
output


     0    1   2
0  0.0  2.0 NaN
1  6.0  8.0 NaN
2  NaN  NaN NaN

     0    1    2
0  0.0  2.0  0.0
1  6.0  8.0  0.0
2  0.0  0.0  0.0
```

Missing values (4)

You can also:

- Drop missing values using `.dropna()`
- Replace missing by interpolation `.interpolate()`
- ...

pandas - sorting labels

`.sort_index()` sorts either by row or column labels:

```
np_array =
np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array,
                  index=['a', 'b',
'c'],
                  columns=['col1',
'col2'])
print(df.sort_index(axis=0,
ascending=False))
```

```
output
```

	col1	col2
c	4	5
b	2	3
a	0	1

There is an `inplace` option which changes the original instead of returning a new data frame

pandas - sorting values

`.sort_values(by=`*label*`)` sorts either by row or column values:

```
np_array =
np.arange(6).reshape((3,2))
df = pd.DataFrame(np_array,
                  index=['a', 'b',
'c'],
                  columns=['col1',
'col2'])
print(df.sort_values(by='col2',
ascending=False))
```

```
output
```

	col1	col2
c	4	5
b	2	3
a	0	1

There is an `inplace` option which changes the original instead of returning a new data frame
You can sort by multiple labels by providing a list to by=

pandas - DataFrames - merging (1)

`.merge()` combines two dataframes based on a shared column label

```
df1 = pd.DataFrame({'id' : [1, 2, 3],
                    'gender' : pd.Categorical(['male',
'female', 'female'])})
df2 = pd.DataFrame({'id' : [1, 2, 3],
                    'name' : pd.Categorical(['Bob', 'Alice',
'Anna'])})
pd.merge(df1, df2, on='id')
```

```
output
   gender  id   name
0    male   1    Bob
1  female   2  Alice
2  female   3   Anna
```

You can also merge on the index labels

```
df1 = pd.DataFrame({'gender' : pd.Categorical(['male',
'female', 'female'])})
df2 = pd.DataFrame({'name' : pd.Categorical(['Bob', 'Alice',
'Anna'])})
pd.merge(df1, df2, left_index=True, right_index=True)
```

pandas - DataFrames - merging (2)

If the indices don't match exactly, you can tell `merge()` how to deal with this, using the `how` argument:

Merge type	Description
'inner'	use intersection of key sets
'outer'	use union of key sets
'left'	use key set from 1st dataframe
'right	use key set from 2nd dataframe

pandas - DataFrames - merging (3)

Examples:

```
df1 = pd.DataFrame({'id' : [1, 2, 4],
                    'gender' :
pd.Categorical(['male',
'female',
'female'])})
df2 = pd.DataFrame({'id' : [1, 2, 3],
                    'name' :
pd.Categorical(['Bob',
'Alice',
'Anna'])})
print(pd.merge(df1, df2, on='id'))


print(pd.merge(df1, df2, on='id',
how='left'))
```

output

```
   id  gender   name
0   1    male    Bob
1   2  female  Alice

   id  gender   name
0   1    male    Bob
1   2  female  Alice
2   4  female    NaN
```

pandas - DataFrame merging - Exercise

1. In addition to the british, also download **https://wouterboomsma.github.io/ppds2022/data/amer english**. Read the british-english file into one dataframe and the american-english file into another.
2. Use pandas' merge functionality to merge the two dataframes based on their word entries, using british as the first argument and american as the second. We want to find words in the british list that are not in the american (just like handin4). The type of merge should therefore be ('left'), so that it uses the british words as a reference. If you give `merge` an `indicator=True` argument, it will add a column to the output called `"_merge"`. Check that this is indeed the case.
3. Now create a new dataframe containing only the rows for which the `"_merge"` columns is equal to `"left_only"`. Count them, and see if it matches the result you got in handin 4.

pandas - DataFrame merging - Exercise - solution

1. Read the british-english file into one dataframe and the american-english file into another.

```
df1 = pd.read_table('british-english',
keep_default_na=False, header=None)
df2 = pd.read_table('american-english',
keep_default_na=False, header=None)
```

2. Use pandas' merge functionality...

```
merged_df = pd.merge(df1, df2, on=0, how='left',
indicator=True)
```

3. Now create a new dataframe containing only the rows for which the "_merge" columns is equal to "left_only"

```
mask = (merged_df['_merge'] == 'left_only')
new_df = merged_df[mask]
print(len(new_df))
```

pandas - DataFrames - concatenating

Concatenating "appends" data frames along a specified axis

```
np_array =
np.arange(6).reshape((3,2))
df1 = pd.DataFrame(np_array, index=
['a', 'b', 'c'],
                   columns=['col1',
'col2'])
print(df1)

df2 = pd.DataFrame(np_array[:2, :1],
                   index=['a', 'b'],
                   columns=['col3',
])
print(df2)


print(pd.concat([df1, df2], axis=1))
```

output

```
   col1  col2
a     0     1
b     2     3
c     4     5

   col3
a     0
b     2

   col1  col2  col3
a     0     1   0.0
b     2     3   2.0
c     4     5   NaN
```

pandas - DataFrames - grouping

The `.groupby()` groups by value

```
df = pd.DataFrame({'name' : pd.Categorical(['Bob', 'Alice',
'Anna']),
                   'sex' : pd.Categorical(['male', 'female',
'female']),
                   'height': [170, 180, 165]})
sex_grps = df.groupby('sex')
print(sex_grps)
```

```
output
<pandas.core.groupby.DataFrameGroupBy object at 0x11861bd30>
```

You can also group by a dynamically created series:

```
df.groupby(df['name'].str.len()) # Group by name length
```

What can you do with these groups?

pandas - DataFrames - grouping (2)

1. The `.groups` attribute provides some information

```
print(sex_grps.groups)
```

output

```
{'female': Int64Index([1, 2], dtype='int64'), 'male': Int64Index([0], dtype='int64')}
```

2. You an also iterate over the groups

```
for name,grp in sex_grps:
    print(name)
    print(grp)
```

output

```
female
      sex  height   name
1  female     180  Alice
2  female     165   Anna
male
     sex  height name
0   male     170  Bob
```

pandas - DataFrames - grouping (3)

...or you can calculate summary statistics

```
print(sex_grps.agg({'height':np.mean})) # Returns a
DataFrame
```

or

```
print(sex_grps['height'].mean()) # Returns a Series
```

Result:

```
output

        height
sex
female   172.5
male     170.0
```

pandas - DataFrame grouping - Exercise

1. Download: **https://wouterboomsma.github.io/ppds2022/data/britis english**, and read it into a dataframe
2. Get pandas to group by first letter - and use this to count the words for each letter

pandas - DataFrame grouping - Exercise - solution

1. Download: **https://wouterboomsma.github.io/ppds2022/data/britis english**, and read it into a dataframe

```
df = pd.read_table('british-english',
keep_default_na=False, header=None)
```

2. Get pandas to group by first letter - and use this to count the words for each letter

```
df.groupby(df[0].str[0]).count()
```

pandas - working with time data

Pandas has been designed for use with time series data
You can create a time range using `pd.date_range()`, and use this as index in a `Series`

```
idx = pd.date_range('26/9/2018 08:00', periods=3,
freq='H')
series = pd.Series(np.arange(len(idx)), index=idx)
print(series)
```

```
output

2018-09-26 08:00:00    0
2018-09-26 09:00:00    1
2018-09-26 10:00:00    2
Freq: H, dtype: int64
```

pandas - working with time data (2)

We can change the frequency using `.asfreq`

```
print(series.asfreq('30Min'))
```

```
output
2018-09-26 08:00:00    0.0
2018-09-26 08:30:00    NaN
2018-09-26 09:00:00    1.0
2018-09-26 09:30:00    NaN
2018-09-26 10:00:00    2.0
```

Use the `method` option to specify how to fill new values. Alternatively:

```
print(series.asfreq('30Min').interpolate())
```

```
output
2018-09-26 08:00:00    0.0
2018-09-26 08:30:00    0.5
2018-09-26 09:00:00    1.0
...
```

pandas - working with time data (2)

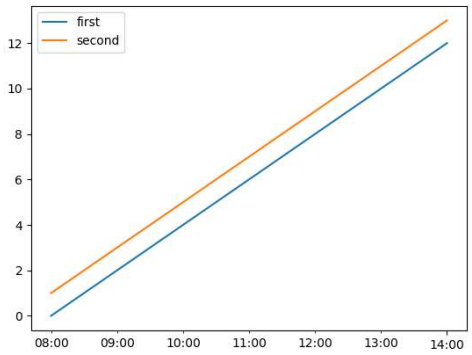
Too much to cover here, but you can:

- Using indexing and slicing on the time index
- Index with partial date strings
- Use BusinessDay as time unit - and define custom versions
- Automatically skip holidays
- Resampling time (time-based groupby)
- ...

pandas - plotting

It is easy to plot from DataFrames through matplotlib

```
idx = pd.date_range('26/9/2018 08:00',
                    periods=7, freq='H')
df = pd.DataFrame(np.arange(14).reshape([7,2]),
                  columns=['first', 'second'],
                  index=idx)
df.plot()
```



output

	first	second
2018-09-26 08:00:00	0	1
2018-09-26 09:00:00	2	3
2018-09-26 10:00:00	4	5
2018-09-26 11:00:00	6	7
2018-09-26 12:00:00	8	9
2018-09-26 13:00:00	10	11
2018-09-26 14:00:00	12	13

Note how the data frame labels are automatically used (axes and legend).
Other plot types: `df.plot.bar`, `df.plot.scatter`, etc

pandas - DataFrame plotting - Exercise

1. Create a histogram of the word counts from the previous exercise.
2. Bonus: Try to get meaningful labels on the x-axis and in the legend.

pandas - DataFrame plotting - Exercise - solution

1. Create a histogram of the word counts from the previous exercise.

```
# From previous exercise
df = pd.read_table('british-english',
keep_default_na=False, header=None)
result = df.groupby(df.loc[:,0].str[0]).count()

# This exercise:
result = result.rename_axis(index='letter')  # rename
index axis
result.columns = ['word count']   # Give column a more
informative name
result.plot.bar()
```

Speaker notes