Pandas II

10.4.24

When is Pandas useful?

- Tabular, structured data
- Mixed data-types (text, num)
- Biggest advantage for categorical data

DataFrames = table with style

Building dataframes:

pd.DataFrame(.....)

- Dictionaries: elements must be lists
- NumPy arrays
- Series
- Anything that can become a NumPy array

pd.read_csv(....)
pd.read_excel(....)
pd.read_json(....)
pd.read_clipboard(...)

This converts whatever you've copied (as in ctrl+c) into a DataFrame

Structuring DataFrames from arrays

By default **pd.DataFrame** labels the columns/rows 0, 1, 2 etc.

Assign values in call

columns=

index= (rows)

Change later:

data.columns

data.index

```
zz=pd.DataFrame([[0,1],[2,3]],\
columns=['a','b'],index=['x','y'])
print(zz)
```

```
   a  b
x  0  1
y  2  3
```

Essential Functions

- Comparison
- Sorting
- Aggregating
- Censoring/Replacing
- Stats

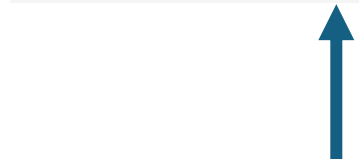
Grocery DataFrame

	Aisle	Sold	Remainder
Spam	Can	53	9
Plum	Fruit	41	71
Carrot	Veg	23	82
Beet	Veg	13	29
Bean	Can	23	8
Kale	Veg	20	10

```
data=pd.DataFrame([['Can',53,9],['Fruit',41,71],['Veg',23,82],['Veg',13,29],\
                   ['Can',23,8],['Veg',20,10]],index=['Spam','Plum','Carrot',\
                    'Beet','Bean','Kale'],columns=['Aisle','Sold','Remainder']);
```

Let's find where sales exceed inventory

```
shortage=data['Sold']>data['Remainder'];
data.index[shortage]
```



index = row name

```
['Spam', 'Bean', 'Kale']
```

	Aisle	Sold	Remainder
Spam	Can	53	9
Plum	Fruit	41	71
Carrot	Veg	23	82
Beet	Veg	13	29
Bean	Can	23	8
Kale	Veg	20	10

Let's find top sellers

sort values (mutable operation):

```
data.sort_values(by='Sold')
print(data)
```

	Aisle	Sold	Remainder
Spam	Can	53	9
Plum	Fruit	41	71
Carrot	Veg	23	82
Beet	Veg	13	29
Bean	Can	23	8
Kale	Veg	20	10

data.idxmax() and **data.idxmin()** return the labels for min/max of each column:

idxmax() →

Aisle	Carrot
Sold	Spam
Remainder	Carrot

Practice Together: Time to order more spam

- We need to order stock for the next day
- To be safe, we want to stock 150% of the food sold today.

- Create a new column (‘Buy’) that displays the amount needed to buy, i.e.

  ceil(max(0,1.5*sold-remaining))

- Use np.maximum(A,b) to perform elementwise max, np.ceil() or create your own fct. with np.vectorize

Applying fcts to dataframe

data.apply(fun,axis=...,result_type=...)

applies a function over each row (axis=1) or column (axis=0) of a dataframe

optionally: result_type='expand' if you want to add new rows/columns (fct. should output a series)

```
def toOrder(tmpData):
    toOrder=np.ceil(1.5*tmpData['Sold']\
                    -tmpData['Remainder']);
    return int(max(0,toOrder))
data['Buy']=data.apply(toOrder,axis=1)
print(data)
```

	Aisle	Sold	Remainder	Buy
Spam	Can	53	9	71
Plum	Fruit	41	71	0
Carrot	Veg	23	82	0
Beet	Veg	13	29	0
Bean	Can	23	8	27
Kale	Veg	20	10	20

Pipe: Pipelines

- Pipelines: chaining a bunch of functions together
- data.pipe(fun, args)

- Fun should take data as its first argument. Args denote any additional arguments to supply.

data.pipe(fun1,args1)**.pipe**(fun2,args2)**.pipe**(fun3,args3).

Querying

- **data.query**(‘expression’)
  - Expression is a string that is evaluated to filter values
  - Refer to environmental variables with @var_name

```
data.query('Aisle=="Veg"')
```

	Aisle	Sold	Remainder	Buy
Carrot	Veg	23	82	0
Beet	Veg	13	29	0
Kale	Veg	20	10	20

```
data.query('Buy>0')
```

	Aisle	Sold	Remainder	Buy
Spam	Can	53	9	71
Bean	Can	23	8	27
Kale	Veg	20	10	20

Querying

- Query is column-based
- To query using row labels, use transpose:

```
data.iloc[:,1:].T.query('Spam>Plum').T
```

	Sold	Buy
Spam	53	71
Plum	41	0
Carrot	23	0
Beet	13	0
Bean	23	27
Kale	20	20

Aggregate Functions

- Summary measures across data

data.aggregate('func_names',axis=...) or **df.agg**(....)

```
data.aggregate(['sum','max'])
```

See what it did there...

	Aisle	Sold	Remainder	Buy
sum	CanFruitVegVegCan	153	199	98
max	Veg	53	82	71

Practice Together

- You can use a dictionary for column-specific aggregates

- For Aisle return the mode, otherwise return max and sum:

Grouping

- **data.groupby**('label').operation. Performs aggregate operations over each group-case

```
data.groupby('Aisle').sum()
```

Aisle	Sold	Remainder	Buy
Can	76	17	98
Fruit	41	71	0
Veg	56	121	20

```
data.groupby('Aisle')['Sold'].sum()
```

```
Aisle
Can      76
Fruit    41
Veg      56
Name: Sold, dtype: int64
```

Censoring/Replacing

- **df.drop**(label,axis=): drop a row/column
- **df**[col]**.str.replace**(..,...): use str.replace on a column
- **df.clip**(lower=..,upper=...): clip data to upper/lower bounds
- **df.replace**(old_Val, new_Val): replace a value
- **df.fillna**(Value): replace na with a value

- **df.mask**(cond, val): where cond is **true**, replace with val
- **df.where**(cond,val): where cond is **false**, replace with val

Statistics Methods

- data.describe(): All the summary measures

```
data.describe()
```

	Sold	Remainder	Buy
count	6.000000	6.000000	6.000000
mean	28.833333	34.833333	19.666667
std	15.025534	33.379135	27.746471
min	13.000000	8.000000	0.000000
25%	20.750000	9.250000	0.000000
50%	23.000000	19.500000	10.000000
75%	36.500000	60.500000	25.250000
max	53.000000	82.000000	71.000000

Statistics Methods

- pd.crosstab(dat1,dat2):

```
pd.crosstab(data['Aisle'],data['Buy']>0)
```

Buy	False	True
Aisle		
Can	0	2
Fruit	1	0
Veg	2	1

- data.sample(n)

```
data.sample(3,replace=False)
```

	Aisle	Sold	Remainder	Buy
Kale	Veg	20	10	20
Spam	Can	53	9	71
Bean	Can	23	8	27

Fin