Feb 16th, 2022

Date

Financial Deivatives

Pandas

Series: 1D, one column

Dataframe: multiple dimensions, 2D, collection of series

np.arange(1, 6, 1) → create values from 1 to 6 (exclude 6) with jump of 1 → 1 2 3 4 5.

pd.Series(np.arange(12, 14, 1), index = [1, 2])

0- NaN
1- 12.0 ← assign indexes to array 12, 13
2- 13.0
3- NaN → when there's NaN, column values become float.

= pd.DataFrame([[10, 11], [20, 21]], index = ['R1', 'R2'], columns = ['a', 'b'])

df.index = ['r1', 'r2'] → to change index labels later on.

df.columns ← Tells about all the names of columns df.index ← same as columns

df.values ← all values in DF df.info() ← detailed summary df.describe() ← mean, std, quartiles

df['c1']
df.c1 ← same but maybe c1 is a builtin variable, will give error then.

df[['c1', 'c2']] ← gives values of both col.

df[['r1', 'r2']] ← gives error cuz no slicing done

df[['r1':'r2']] ← gives error cuz slicing done so [] not needed

df['r1':'r2'] ← gives rows r1+r2

df[1] ← gives 0th index row i.e. r1

df.iloc[0] ← integer location, 0th for rows, 1st for columns

df.iloc[2, 4] ← give 2nd row 4th column

df.iloc[[0, 1]] ← gives 0th + 1st row

Date Feb 17th, 2022

```
sp500 = pd.read_csv('sp500.csv')
sp500
↓
Symbol Name ..... Price (per share) ... Price/
```

S&P = Standard and Poor

SP 500 = Standard + Poor US companies in PSX.

- Price / Earnings per share

↓

$$EPS = \frac{\text{Net income}}{\text{\# of outstanding shares}}$$

→ shares held by market shares not with company

10 ⟸

means investors are ying $10 to get earning of $1

- MSFT
  - stocks issue by firm 100,000
  - bought back by firm 20,000
  - outstanding stocks held by investors in market 80,000

	A	B
P/E =	$100	$50

↳ investors are willing to pay twice as for A because they see Prosperity in future. minimize risk, maximise return

10 $ → 12 $

12 − 10 = 2

↓

Return

hat's Return? $10/share

you sell your $10 share for $12, return is $2.

- Book Value: value of shares in books of a firm. Books are the documents which contain balance sheets, income stmts etc.

Market value (Price) > Book Value (always)

Shareholder's Equity (SHE) book value/share

0, 23, 7

Date

Intrinsic Value

```
sp500 = pd.read_csv('sp500.csv', index_col='Symbol',
        usecols=[0, 2, 3, 7])
sp500.head(5)  ← only first 5.
```

```
sp500.iloc[sp500.index.get_loc('ABT')]
    ↕ equal
sp500.loc['ABT']
```

↓ when we don't know integer index of row but know its name.

```
sp500.loc['ABT', 'Price']
          ↓       ↓
          r       c
```

```
sp500['Price'] < 100  ← gives T/F on rows in Series
```

```
sp500[sp500['Price'] < 100][['Price', 'Book Value']]
```

{ $100/share truly worth
  $80/share being trade

- undepriced so its good to buy undepriced shares because it will eventually converge to $
- happens so company can buy their shares back.

Date 23rd February - 2022.

- df[0] ← giving KeyError cuz rows can either be extracted by iloc or slicing.
- df.iloc[[0,1],[2,3]]
- df.iloc[[0,1]][['C','D']] ← gives first 2 rows of C, D columns.
  (1st rows, 2nd col)
- subframe

	C	D
0	3	4
1	7	8

df

	A	B	C	D
0				
1				
2				
3				

df - subframe → first alignment is performed

	A	B	C	D
0	NaN	NaN	0.	0.
1	.	.	0.	0.
2	.	.	NaN	NaN
3	.	.	"	"

df = df.iloc[0] ← remove all rows from 0th row values

df = df.iloc[[0]] ← removes only where match comes, otherwise NaN.

w.r.t (df - dataset)

DIY

- df = df['A'] ←

	A	B	C	D	0	1	2	3
0	NaN							
1								
2								
3								

NaN

but works fine with df.sub(df['A'], axis=0) → subtraction col. wise

axis = 1 ← row wise

with this it gives all NaNs + A B C D 0 1 2 3 0 1 2 3

this tells us it subtracts row wise

Date 24th Feb, 2022.

- array = np.array(12).reshape(3,4)
  (0 to 11; 3 rows, 4 columns)

- frame = pd.DataFrame(np.array(12).reshape(4,3), columns = list('bde'), index = ['Pes', 'Sob', 'Khi'])

	b	d	e
Pes	0	1	2
Sob	3	4	5
Chr	6	7	8
Khi	9	10	11

- series = frame.iloc[0]

b	0
d	1
e	2

- frame - series

Broadcasting over rows when indexes from series are matched against columns in dataframe. Default alignment

	b	d	e
Pes	0	0	0
Sob	3	3	3
Chr	6	6	6
Khi	9	9	9

- series2 = pd.Series(range(3), index = ['b', 'e', 'f'])

b	0
e	1
f	2

~~series2 - frame~~

frame - series2

	b	d	e	f
Pes	0.0	NaN	1.0	NaN
Sob	3.0	.	4.0	.
Chr	6.0	.	7.0	.
Khi	9.0	.	10.0	.

- series3 = frame['d']

P	1
S	4
L	7
K	10

frame - series3

	P	S	L	K	b	d	e
P	NaN	NaN	NaN	NaN	NaN	NaN	NaN
S	.	.	.	.	.	.	.
L	.	.	.	.	.	.	.
K	.	.	.	.	.	.	.

	b	d	e
P	-1	0	1
J	-1	0	1
L	-1	0	1
K	-1	0	1

Date

frame.sub(series3, axis='index') → now subtraction is based on matching row indexes

$$\text{return } r_t = \frac{P_t}{P_{t-1}} - 1$$

(P_t: price; t: month, (time) year)

$$r_{\text{Feb. 2022}} = \frac{53}{53} - 1 = 0.06 = 6\%$$

$1 investment has grown to $1.06.

Measures of Central Tendency

Mean, Median, Mode

- influenced by outliers

Measures of Dispersion

Standard deviation, variance, range

mean could be same for 2 datasets but std different

$$r_{\text{Feb, 2022}} = \frac{P_{\text{Feb, 2022}}}{P_{\text{Jan, 2022}}} - 1 = \frac{102}{100} - 1 = 1.02 - 1 = 0.02 \text{ (net return)}$$

profit 2% → return

100 is the amount you paid
102 is the amount that you can sell
so 2% is the profit / net return.

don't use this term

$12\sqrt{60}$

Stock ~~Data~~ analysts calculate return of a company of past 60 months.

Date March 2nd, 2022.

MSFT past returns: 2%, 4%, -3%, 9%, 1%, -5%.

how do we select an appropriate value of return? take avera

let's say return = 3%.

exp. return — why measure of central tendency? because -5% can represent the whole dataset, too far from 9%. vice versa

but this is not an accurate measure so we con

The spread (std).

Return = 3%.

let's say std. = 2%. This means 1% 5%

low risk

Tells us about Risk

if std. = 20%. -17% 23%

high risk

this is why we need past returns. The ↑ spreader, ↑ risk

- we use median or mode when we have outliers and we don't want to remove outliers. Then it will have no affect on return. But for risk analysis you have to calculate std which requires mean so you may remove outlier. If you remove outlier you don't have to do m

T-Bills: 3% for 3 months ⇐ Tbills will mature after 3 months

Treasury bills

government issues them

risk free investment cuz you're expecting 3% & will get 3%.

reliable source cuz they have authority to print money

→ debt instrument

debt → has to be pa back no matt how much f a male or non

Date

Pandas

- read_csv('msft.csv', index_col=0)
- pd.set_option('precision', 2) → gives float values to 2 decimal places.

Datasets' columns:
- Open → when stock market opens
- High → highest rate in the day
- Low → lowest " " " "
- Close → closing rate at the end of day
- Volume → no. of stocks traded in the day
- Adjusted close → close price (not exactly).

- only January's data

```
msft01 = msft['2012-01':'2012-02']
```

↳ if only some cols

```
msft01 = msft['2012-01':'2012-02'][['Adj Close']]
msft02 =      "02       03    [[   "   ]]
```

- pd.concat([msft01.head(3), msft02.head(3)])
  we get 6 rows

same style

- aapl01 = aapl['2012-01':'2012-02'][['Adj Close']]

windings = pd.concat([msft01.head(), aapl01.head()])
↓
we get 6 rows, 2012-01-03 two times, depiction of indexes

- pandas does row-wise concatenation

Date

- withidxs.loc['2012-01-03'] → gives 2 rows ; apple, ms.
  multi level indexes
- closes = pd.concat(" , " , keys = ['MSFT','AAPL']
- closes.loc['MSFT'][:2] ← gives first 2 rows of MSFT.
  Date
  2012-01-03 24.22
  2012-01-04 25.14
- closes.loc['MSFT'].loc['2012-01-03'] ← gives 24.42.

March 03, 2022.

msftAV = msft[['Adj Close', 'Volume']]
aaplAV = aapl "

rowwise appending / concatenation

pd.concat([msftAV.head(), aaplAV.head()]) → 10 rows msft followed by aapl.

aaplA = aapl['Adj Close']

pd.concat([msftAV.head(), aaplA.head()])
gives ajeeb values cuz msftAV is dataframe aaplA is Series

so do: aapl = aapl[['Adj Close']]

now pd.concat([msftAV " , aaplA]) gives okay results.

- alignment not based on row indexes : 2 values of 2012-01-03 one for MSFT one for AAPL

msft['Adj Close'] → Series
msft[['Adj Close']] → Dataframe

Date

- perform concat only for common columns: default = "outer"
  pd.concat([msftAV.head(), aaplA.head()], join="inner")
  ↓
  gives only Adj Close column

- if columnwise concatenation? specify
  axis = 'index' = 0 (row wise)
  axis = 'column' = 1 (column wise)

pd.concat([msftAV.head(), aaplA.head()], axis='column', keys=['MSFT', 'AAPL'])

- pd.concat([msftAV.head(), aaplAV.head(3)], axis=1, keys=['MSFT', 'AAPL'])

- pd.concat([msftAV.head(), aaplAV.head()], ignore_index=True)

axis=1 { when we do column wise indexing/concat and ignore_index then it ignore the names of columns. But its not helpful.

• merge default settings = col wise merging

• col wise concat & merge are not much different

Date March 9th, 2022.

msftA = msft[['Adj Close']]

• msftAR = msftA.reset_index()
msftVR = msftV.reset_index()
msftAVR = pd.merge(msftAR.head(), msftVR.head())

→ Date Adj Close

→ date, volume

(5 rows)

	Date	Adj Close	Volume
0			
1			
2			
3			
4			

• msftVR2_4 = msftVR[2:4]
pd.merge(msftAR.head(), msftVR2_4)

	Date	Adj Close	Volume
only matching columns row mentioned	0		
	1		

• pd.merge(..., how='outer') → gives all rows, matched & unmatch

msft.insert(0, 'Symbol', 'MSFT') → inserting a new column at index 0 with heading Symbol and all values of MSFT

combined = pd.concat([msft.head(), aapl.head()]).sort_index()

↓ sorts rows on the basis of Date.

tmp = combined.reset_index() → resets Date index and makes Date a regular column

Date

make it index ↑ make columns from Symbol ↑

closes = S4p.pivot(index = 'Date', columns = 'Symbol', values = 'Adj Close')

↓ mention values of AC for 'Symbol'

Symbol	AAPL	MSFT
Date		
01-03	...	...
01-04	...	...
01-05	...	...
01-06	...	...
01-09	...	...

- unstack → pivot
  stack → unpivot

stackedcloses = closes.stack() →

Date	Symbol	Adj Close values
01-03	AAPL	...
	MSFT	...
01-04	AAPL	...
	MSFT	...

↓ Series

- stackedcloses.loc['2012-01-03', 'AAPL'] → gives Adj Close value of AAPL on 01-03 date
  == → cuz stackedcloses is Series not df
  stackedcloses['2012-01-03', 'AAPL']
- unstackedcloses = stackedcloses.unstack() . == pivot

Melting

- melted = pd.melt(s4p, id_vars = ['Date', 'Symbol'])

  not melted, stays same become id combination

  each to rows of Open, Close, Vol, Adj Close, High, Low = 60 rows total

Date	Symbol	variable	value (in scientific notation)
01-02		Open	
		Close	
		Vol	
		Adj Close	
		High	
		Low	

Date

condition ↓

fil = (melted['Date'] == '2012-01-03') & (melted['Symbol'] == 'MSFT')

melted[fil] → gives only values of MSFT at 01-03.

- Grouping

- grouped = stg.groupby('Symbol') → gives groupby obj (df).
- type(grouped.groups) → types of dicts
- grouped.groups → gives dict containing keys AAPL + MSFT and their values have indexes that they have in df.
- grouped.ngroups → 2 (AAPL, MSFT)
- grouped.size() →

Symbol	
AAPL	249
MSFT	249

- grouped.get_group('MSFT') → gives values of only MSFT key.