Creating a DataFrame

INTERMEDIATE PYTHON FOR FINANCE

Kennedy Behrman
Data Engineer, Author, Founder

Pandas

```
import pandas as pd
```

```
print(pd)
```

```
<module 'pandas' from '.../pandas/__init__.py'>
```

Pandas DataFrame

```
pd.DataFrame()
```

Pandas DataFrame

	Col 1	Col 2	Col 3
0	v1	a	00
1	v2	b	01
2	v3	c	13.02

From dict

```
data = {'Bank Code': ['BA', 'AAD', 'BA'],
        'Account#': ['ajfdk2', '1234nmk', 'mm3d90'],
        'Balance':[1222.00, 390789.11, 13.02]}
```

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

From dict

```
data = {'Bank Code': ['BA', 'AAD', 'BA'],
        'Account#': ['ajfdk2', '1234nmk', 'mm3d90'],
        'Balance':[1222.00, 390789.11, 13.02]}
```

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

	Bank Code	Account#	Balance
0	BA	ajfdk2	1222.00
1	AAD	1234nmk	390789.11
1	BA	mm3d90	13.02

From list of dicts

```
data = [{'Bank Code': 'BA', 'Account#': 'ajfdk2',  'Balance': 1222.00},
        {'Bank Code': 'AAD', 'Account#': '1234nmk', 'Balance': 390789.11},
        {'Bank Code': 'BA', 'Account#': 'mm3d90', 'Balance': 13.02}]
df = pd.DataFrame(data=data)
```

From list of dicts

```
data = [{'Bank Code': 'BA', 'Account#': 'ajfdk2',  'Balance': 1222.00},
        {'Bank Code': 'AAD', 'Account#': '1234nmk', 'Balance': 390789.11},
        {'Bank Code': 'BA', 'Account#': 'mm3d90', 'Balance': 13.02}]
df = pd.DataFrame(data=data)
```

	Bank Code	Account#	Balance
0	BA	ajfdk2	1222.00
1	AAD	1234nmk	390789.11
1	BA	mm3d90	13.02

From list of lists

```
data = [['BA',  'ajfdk2',  1222.00],
        ['AAD', '1234nmk', 390789.11],
        ['BA',  'mm3d90',  13.02]]
df = pd.DataFrame(data=data)
```

From list of lists

```
data = [['BA',  'ajfdk2',  1222.00],
        ['AAD', '1234nmk', 390789.11],
        ['BA',  'mm3d90',  13.02]]
df = pd.DataFrame(data=data)
```

	0	1	2
0	BA	ajfdk2	1222.00
1	AAD	1234nmk	390789.11
1	BA	mm3d90	13.02

From list of lists with column names

```
data = [['BA',  'ajfdk2',  1222.00],
        ['AAD', '1234nmk', 390789.11],
        ['BA',  'mm3d90',  13.02]]
columns = ['Bank Code', 'Account#', 'Balance']
df = pd.DataFrame(data=data, columns=columns)
```

	Bank Code	Account#	Balance
0	BA	ajfdk2	1222.00
1	AAD	1234nmk	390789.11
1	BA	mm3d90	13.02

From list of lists with column names

```
data = [['BA',  'ajfdk2',  1222.00],
        ['AAD', '1234nmk', 390789.11],
        ['BA',  'mm3d90',  13.02]]
columns = ['Bank Code', 'Account#', 'Balance']
df = pd.DataFrame(data=data, columns=columns)
```

	Bank Code	Account#	Balance
0	BA	ajfdk2	1222.00
1	AAD	1234nmk	390789.11
2	BA	mm3d90	13.02

Reading data

- Excel `pd.read_excel`
- JSON `pd.read_json`
- HTML `pd.read_html`
- Pickle `pd.read_pickle`
- Sql `pd.read_sql`
- Csv `pd.read_csv`

CSV

Comma separated values

```
client id,trans type, amount
14343,buy,23.0
0574,sell,2000
7093,dividend,2234
```

Reading a csv file

```
df = pd.read_csv('/data/daily/transactions.csv')
```

Reading a csv file

```
df = pd.read_csv('/data/daily/transactions.csv')
```

client id	trans type	amount
14343	buy	23.0
0574	sell	2000
7093	dividend	2234

Non-comma csv

```
client id|trans type| amount
14343|buy|23.0
0574|sell|2000
7093|dividend|2234
```

Non-comma csv

```
df = pd.read_csv('/data/daily/transactions.csv', sep='|')
```

Non-comma csv

```
df = pd.read_csv('/data/daily/transactions.csv', sep='|')
```

client id	trans type	amount
14343	buy	23.0
0574	sell	2000
7093	dividend	2234

Let's practice!

INTERMEDIATE PYTHON FOR FINANCE

Accessing Data

INTERMEDIATE PYTHON FOR FINANCE

Kennedy Behrman
Data Engineer, Author, Founder

Account Balance

Introducing lesson data

	Bank Code	Account#	Balance
a	BA	ajfdk2	1222.00
b	AAD	1234nmk	390789.11
c	BA	mm3d90	13.02

```
accounts
```

Access column using brackets

```
accounts['Balance']
```

Access column using brackets

```
accounts['Balance']
```

a	1222.00
b	390789.11
c	13.02

Name: Balance, dtype: float6

Access column using dot-syntax

```
accounts.Balance
```

	Balance
a	1222.00
b	390789.11
c	13.02

Access multiple columns

```
accounts[['Bank Code', 'Account#']]
```

Access multiple columns

```
accounts[['Bank Code', 'Account#']]
```

	Bank Code	Account#
a	BA	ajfdk2
b	AAD	1234nmk
c	BA	mm3d90

Access rows using brackets

```
accounts[0:2]
```

Access rows using brackets

```
accounts[0:2]
```

	Bank Code	Account#	Balance
a	BA	ajfdk2	1222.00
b	AAD	1234nmk	390789.11

Access rows using brackets

```
accounts[[True, False, True]]
```

Access rows using brackets

```
accounts[[True, False, True]]
```

	Bank Code	Account#	Balance
a	BA	ajfdk2	1222.00
c	BA	mm3d90	13.02

loc and iloc

- `loc` access by name
- `iloc` access by position

loc

```
accounts.loc['b']
```

Bank Code	AAD
Account#	1234nmk
Balance	390789

Name: b, dtype: object

loc

```
accounts.loc[['a','c']]
```

	Bank Code	Account#	Balance
a	BA	ajfdk2	1222.00
c	BA	mm3d90	13.02

```
df.loc[[True, False, True]]
```

	Bank Code	Account#	Balance
a	BA	ajfdk2	1222.00
c	BA	mm3d90	13.02

Columns with loc

```
accounts.loc['a':'c','Balance']
```

```
accounts.loc['a':'c', ['Balance','Account#']]
```

```
accounts.loc['a':'c',[True,False,True]]
```

```
accounts.loc['a':'c','Bank Code':'Balance']
```

Columns with loc

```
accounts.loc['a':'c',['Bank Code', 'Balance']]
```

Columns with loc

```
accounts.loc['a':'c',['Bank Code', 'Balance']]
```

	Bank Code	Balance
a	BA	1222.00
b	AAD	390789.11
c	BA	13.02

iloc

```
accounts.iloc[0:2, [0,2]]
```

iloc

```
accounts.iloc[0:2, [0,2]]
```

iloc

```
accounts.iloc[0:2, [0,2]]
```

	Bank Code	Balance
a	BA	1222.00
b	AAD	390789.11

Setting a single value

	Bank Code	Account#	Balance
a	BA	ajfdk2	1222.00
b	AAD	1234nmk	390789.11
c	BA	mm3d90	13.02

```
accounts.loc['a', 'Balance'] = 0
```

Setting a single value

	Bank Code	Account#	Balance
a	BA	ajfdk2	0.00
b	AAD	1234nmk	390789.11
c	BA	mm3d90	13.02

```
accounts.loc['a', 'Balance'] = 0
```

Setting multiple values

	Bank Code	Account#	Balance
a	BA	ajfdk2	1222.00
b	AAD	1234nmk	390789.11
c	BA	mm3d90	13.02

```
accounts.iloc[:2, 1:] = 'NA'
```

Setting multiple columns

	Bank Code	Account#	Balance
a	BA	NA	NA
b	AAD	NA	NA
c	BA	mm3d90	13.02

```
accounts.iloc[:2, 1:] = 'NA'
```

Let's practice!

INTERMEDIATE PYTHON FOR FINANCE

Aggregating and summarizing

INTERMEDIATE PYTHON FOR FINANCE

Kennedy Behrman
Data Engineer, Author, Founder

DataFrame methods

- `.count()`
- `.min()`
- `.max()`
- `.first()`
- `.last()`
- `.sum()`
- `.prod()`
- `.mean()`
- `.median()`
- `.std()`
- `.var()`

Axis

Rows

- default
- `axis=0`
- `axis='rows'`

Columns

- `axis=1`
- `axis='columns'`

Count

	AAD	GDDL	IMA
2020-10-03	300.22	75.32	39.90
2020-10-04	301.49	79.99	44.99
2020-10-05	300.00	80.00	45.33
2020-10-07	302.90	82.92	49.00

```
df.count()
```

```
AAD      4
GDDL     4
IMA      4
dtype: int64
```

Sum

	AAD	GDDL	IMA
2020-10-03	300.22	75.32	39.90
2020-10-04	301.49	79.99	44.99
2020-10-05	300.00	80.00	45.33
2020-10-07	302.90	82.92	49.00

```
df.sum(axis=1)
```

```
2020-10-03    415.44
2020-10-04    426.47
2020-10-05    425.33
2020-10-07    434.82
dtype: float64
```

Product

	AAD	GDDL	IMA
2020-10-03	300.22	75.32	39.90
2020-10-04	301.49	79.99	44.99
2020-10-05	300.00	80.00	45.33
2020-10-07	302.90	82.92	49.00

```
df.prod(axis='columns')
```

```
2020-10-03    9.022416e+05
2020-10-04    1.084987e+06
2020-10-05    1.087920e+06
2020-10-07    1.230707e+06
dtype: float64
```

Mean

	AAD	GDDL	IMA
2020-10-03	300.22	75.32	39.90
2020-10-04	301.49	79.99	44.99
2020-10-05	300.00	80.00	45.33
2020-10-07	302.90	82.92	49.00

```
df.mean()
```

```
AAD      301.1525
GDDL      79.5575
IMA       44.8050
dtype: float64
```

Median

	AAD	GDDL	IMA
2020-10-03	300.22	75.32	39.90
2020-10-04	301.49	79.99	44.99
2020-10-05	300.00	80.00	45.33
2020-10-07	302.90	82.92	49.00

```
df.median()
```

```
AAD      300.855
GDDL      79.995
IMA       45.160
dtype: float64
```

Standard deviation

	AAD	GDDL	IMA
2020-10-03	300.22	75.32	39.90
2020-10-04	301.49	79.99	44.99
2020-10-05	300.00	80.00	45.33
2020-10-07	302.90	82.92	49.00

```
df.std()
```

```
AAD     1.337345
GDDL    3.143548
IMA     3.740183
dtype: float64
```

Variance

	AAD	GDDL	IMA
2020-10-03	300.22	75.32	39.90
2020-10-04	301.49	79.99	44.99
2020-10-05	300.00	80.00	45.33
2020-10-07	302.90	82.92	49.00

```
df.var()
```

```
AAD       1.788492
GDDL      9.881892
IMA      13.988967
dtype: float64
```

Columns and rows

	AAD	GDDL	IMA
2020-10-03	300.22	75.32	39.90
2020-10-04	301.49	79.99	44.99
2020-10-05	300.00	80.00	45.33
2020-10-07	302.90	82.92	49.00

```
df.loc[:,'AAD'].max()
```

```
302.9
```

```
df.iloc[0].min()
```

```
39.9
```

Let's practice!

INTERMEDIATE PYTHON FOR FINANCE

Extending and manipulating data

INTERMEDIATE PYTHON FOR FINANCE

Kennedy Behrman
Data Engineer, Author, Founder

PCE

Personal consumption expenditures (PCE)

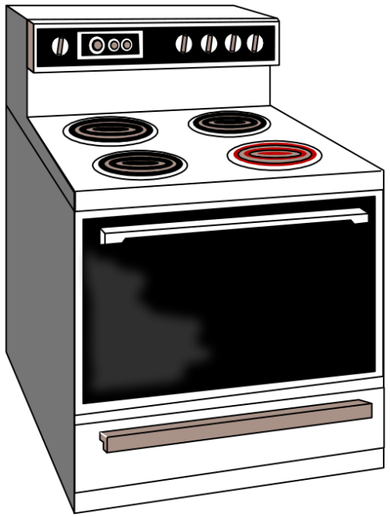
PCE =

PCE

Personal consumption expenditures (PCE)

PCE = PCDG

Durable goods



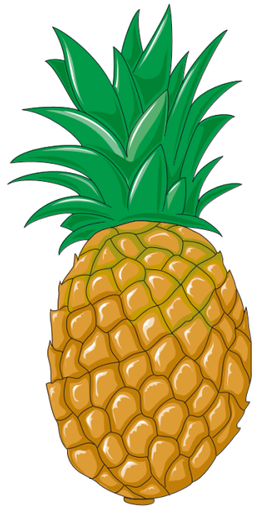
[1] By cactus cowboy [2] Open Clipart, CC0, https://commons.wikimedia.org/w/index.php?curid=64953673

PCE

Personal consumption expenditures (PCE)

PCE = PCDG + PCNDG

Non-durable goods



[1] By Smart Servier [2] https://smart.servier.com/, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=74765623

PCE

Personal consumption expenditures (PCE)

PCE = PCDG + PCNDG + PCESV

Services

[1] By Clip Art by Vector Toons [2] Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=65937611

PCE - adding and removing columns

DATE	PCDGA
1929-01-01	9.829
1930-01-01	7.661
1931-01-01	5.911
1932-01-01	3.959

PCE - adding and removing columns

```
pce['PCND'] = [[33.941,
                30.503,
                25.798000000000002,
                20.169]
```

PCE - adding and removing columns

```
pce
```

DATE	PCDG	PCND
1929-01-01	9.829	33.941
1930-01-01	7.661	30.503
1931-01-01	5.911	25.798
1932-01-01	3.959	20.169

PCE - adding and removing columns

```
pce
```

DATE	PCDG	PCND
1929-01-01	9.829	33.941
1930-01-01	7.661	30.503
1931-01-01	5.911	25.798
1932-01-01	3.959	20.169

```
pcesv
```

PCESV	
0	33.613
1	31.972
2	28.963
3	24.587

PCE - adding and removing columns

```
pce['PCESV'] = pcesv
```

```
pce
```

PCE - adding and removing columns

```
pce['PCESV'] = pcesv
```

```
pce
```

DATE	PCDG	PCND	PCESV
1929-01-01	9.829	33.941	33.613
1930-01-01	7.661	30.503	31.972
1931-01-01	5.911	25.798	28.963
1932-01-01	3.959	20.169	24.587

PCE - adding and removing columns

```
pce['PCE'] = pce['PCDG'] + pce['PCND'] + pce['PCESV']
```

PCE - adding and removing columns

```
pce['PCE'] = pce['PCDG'] + pce['PCND'] + pce['PCESV']
```

DATE	PCDG	PCND	PCESV	PCE
1929-01-01	9.829	33.941	33.613	77.383
1930-01-01	7.661	30.503	31.972	70.136
1931-01-01	5.911	25.798	28.963	60.672
1932-01-01	3.959	20.169	24.587	48.715

PCE - adding and removing columns

```
pce.drop(columns=['PCDG', 'PCND', 'PCESV'],
         axis=1,
         inplace=True)
```

PCE - adding and removing columns

```
pce.drop(columns=['PCDG', 'PCND', 'PCESV'],
         axis=1,
         inplace=True)
```

DATE	PCE
1929-01-01	77.383
1930-01-01	70.136
1931-01-01	60.672
1932-01-01	48.715

PCE - adding and removing rows

```
new_row
```

PCE - adding and removing rows

```
new_row
```

```
pce.append(new_row)
```

DATE	PCE
1933-01-01	45.945

PCE - adding and removing rows

```
new_row
```

DATE	PCE
1933-01-01	45.945

```
pce.append(new_row)
```

DATE	PCE
1929-01-01	77.383
1930-01-01	70.136
1931-01-01	60.672
1932-01-01	48.715
1933-01-01	45.945

PCE - adding and removing rows

Adding multiple rows

```
new_rows = [ row1, row2, row3
]
for row in new_rows:
    pce = pce.append(row)
```

PCE - adding and removing rows

Adding multiple rows

```
for row in new_rows:
    pce = pce.append(row)
```

DATE	PCE
1929-01-01	77.383
1930-01-01	70.136
1931-01-01	60.672
1932-01-01	48.715
1933-01-01	45.945
1934-01-01	51.461
1935-01-01	55.933

PCE - adding and removing rows

```
pce.drop(['1934-01-01',
          '1935-01-01',
          '1936-01-01',
          '1937-01-01',
          '1938-01-01',
          '1939-01-01'],
          inplace=True)
```

PCE - adding and removing rows

```
pce.drop(['1934-01-01',
          '1935-01-01',
          '1936-01-01',
          '1937-01-01',
          '1938-01-01',
          '1939-01-01'],
          inplace=True)
```

DATE	PCE
1929-01-01	77.383
1930-01-01	70.136
1931-01-01	60.672
1932-01-01	48.715
1933-01-01	45.945

PCE - adding and removing rows

```
all_rows = [row1, row2, row3, pce]
```

```
pd.concat(all_rows)
```

PCE - adding and removing rows

```
all_rows = [row1, row2, row3, pce]
```

```
pd.concat(all_rows)
```

DATE	PCE
1929-01-01	77.383
1930-01-01	70.136
1931-01-01	60.672
1932-01-01	48.715
1933-01-01	45.945
1934-01-01	51.461
1935-01-01	55.933

PCE - operations on DataFrames

```
ec = 0.88
pce * ec
```

PCE - operations on DataFrames

```
ec = 0.88
pce * ec
```

DATE	PCE
1934-01-01	45.28568
1935-01-01	49.22104
1936-01-01	54.72544
1937-01-01	58.81832

PCE - map

```
def convert_to_euro(x):
    return x * 0.88

pce['EURO'] = pce['PCE'].map(convert_to_euro)
```

PCE - map

```
def convert_to_euro(x):
    return x * 0.88

pce['EURO'] = pce['PCE'].map(convert_to_euro)
```

DATE	PCE	EURO
1934-01-01	51.461	45.28568
1935-01-01	55.933	49.22104
1936-01-01	62.188	54.72544

Gross Domestic Product (GDP)

- GDP = PCE + GE + GPDI + NE
- PCE: Personal Consumption Expenditures
- GE: Government Expenditures
- GPDI: Gross Private Domestic Investment
- NE: Net Exports

GDP - apply

map - Elements in a column (series)

apply - Across rows or columns

GDP - apply

	GCE	GPDI	NE	PCE
DATE				
1929-01-01	9.622	17.170	0.383	77.383
1930-01-01	10.273	11.428	0.323	70.136
1931-01-01	10.169	6.549	0.001	60.672
1932-01-01	8.946	1.819	0.043	48.715

GDP - apply

```
gdp.apply(np.sum, axis=1)
```

GDP - apply

```
gdp['GDP'] = gdp.apply(np.sum, axis=1)
```

	GCE	GPDI	NE	PCE	GDP
DATE					
1929-01-01	9.622	17.170	0.383	77.383	104.558
1930-01-01	10.273	11.428	0.323	70.136	92.160
1931-01-01	10.169	6.549	0.001	60.672	77.391
1932-01-01	8.946	1.819	0.043	48.715	59.523

Let's practice!

INTERMEDIATE PYTHON FOR FINANCE