Queen Mary
University of London

QHP4701
Introduction to Data Science Programming

Data handling with Pandas Library

Lecturer: Nikesh Bajaj, PhD
School of Physical and Chemical Sciences
http://nikeshbajaj.in

Lecture Outline

Introduction to Pandas

- Reading a CSV file and viewing Data
- Creating DataFrame and writing to CSV
- Selecting and Indexing
- Creating columns and Assigning values
- Aggregation of Data
- Aggregation of Data by group
- Concatenation and Renaming
- More on Pandas

Ref: Python Data Science Handbook, 2nd Edition, Chapter 3

Link: https://jakevdp.github.io/PythonDataScienceHandbook/

Pandas: Handling Data

- Pandas is a python library to handle tabulated data. It convert a tabulated data in an easy-to-use structure to manipulated and analyse.
- Among others, Anaconda comes with Pandas.
- To use Pandas, first we need to import it as

```
import  pandas as pd
```

Read a CSV file

- Let’s start with reading a file first.

```
pd.read_csv(file_path)
```

```
File_path has to be a string as full path (absolute or relative path)
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

file_path =‘../dataset_file.csv’
D = pd.read_csv(file_path)
```

Pandas Object: DataFrame

- After reading a file, check type and options.

```
D = pd.read_csv(file_path)

type(D)

D.<TAB>
```

Pandas read a file and save as a DataFrame object, which has many advantages over numpy array.

Pandas DataFrame can be converted to numpy array using

X = np.array(D)

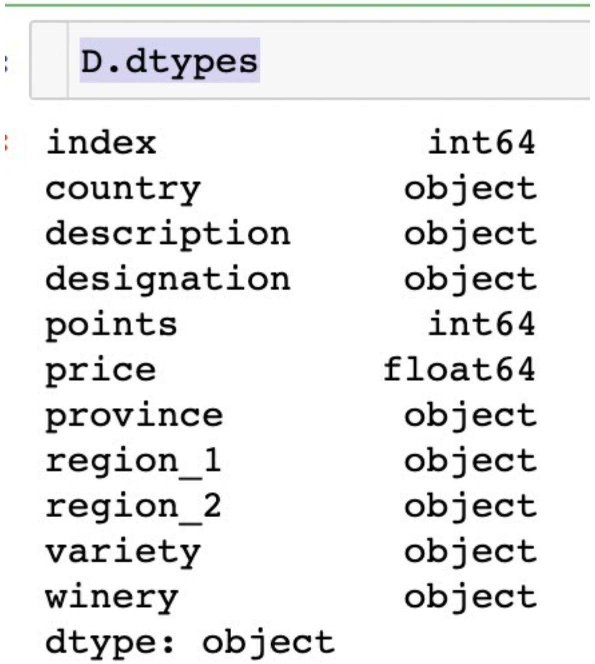
X = D.to_numpy()

Pandas Object: DataFrame

- Pandas DataFrame can have columns with different data-types

types

D.dtypes.



```
D.dtypes
```

```
index            int64
country         object
description     object
designation     object
points           int64
price          float64
province        object
region_1        object
region_2        object
variety         object
winery          object
dtype: object
```

shape

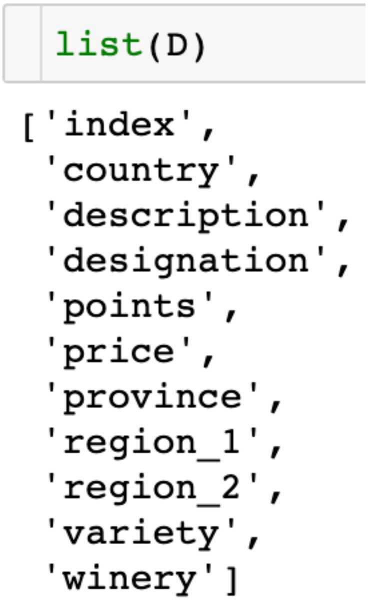
D.shape

(150930, 11)

name of columns

list(D)

Or *D.columns*



```
list(D)
```

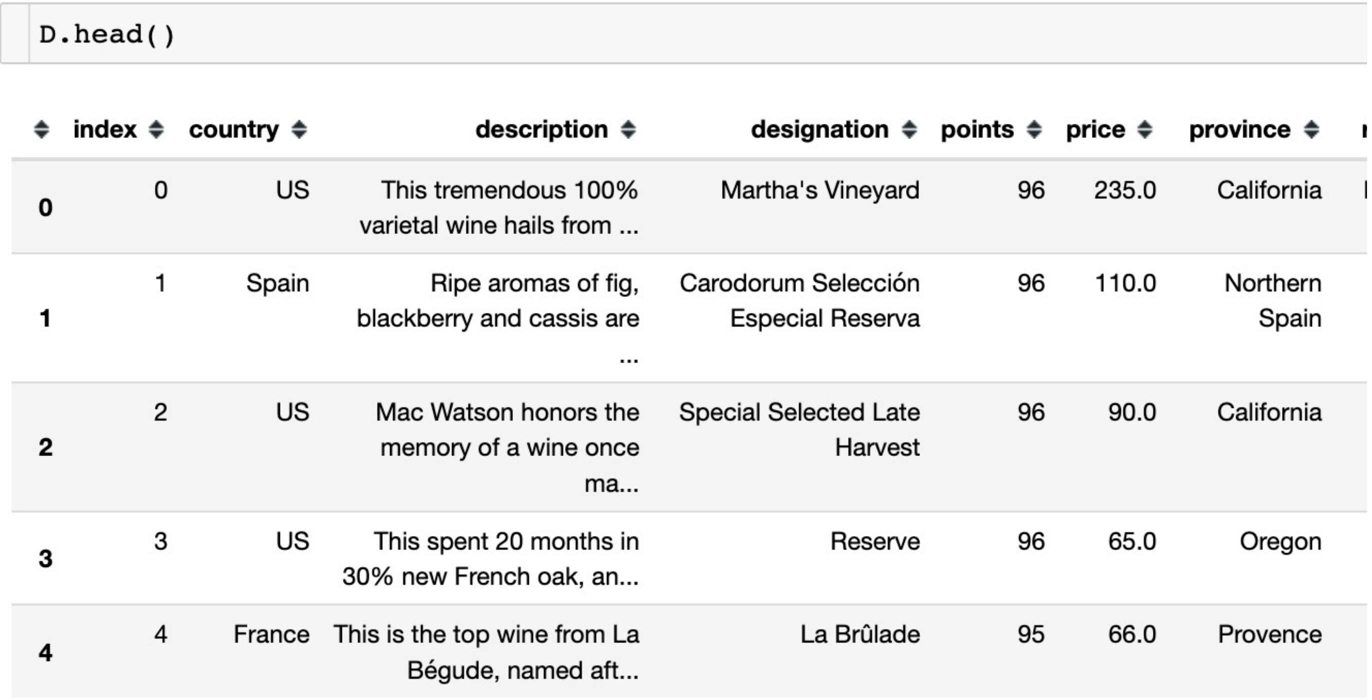
```
['index',
 'country',
 'description',
 'designation',
 'points',
 'price',
 'province',
 'region_1',
 'region_2',
 'variety',
 'winery']
```

View Data

- Pandas display DataFrame as Table like – more readable

Viewing first few rows or a particular column

- display(D)
- D.head()
- D.head(10)
- D['country']



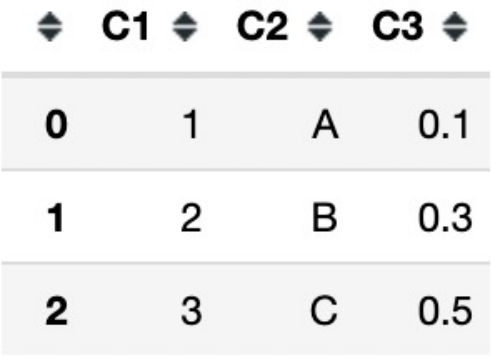
```
D.head()
```

	index	country	description	designation	points	price	province
0	0	US	This tremendous 100% varietal wine hails from ...	Martha's Vineyard	96	235.0	California
1	1	Spain	Ripe aromas of fig, blackberry and cassis are ...	Carodorum Selección Especial Reserva	96	110.0	Northern Spain
2	2	US	Mac Watson honors the memory of a wine once ma...	Special Selected Late Harvest	96	90.0	California
3	3	US	This spent 20 months in 30% new French oak, an...	Reserve	96	65.0	Oregon
4	4	France	This is the top wine from La Bégude, named aft...	La Brûlade	95	66.0	Provence

Pandas Object: DataFrame

- Creating a DataFrame
- Using Dict, Lists, Arrays

```
df = pd.DataFrame({'C1': [1,2,3],
                   'C2': ['A','B','C'],
                   'C3':[0.1, 0.3, 0.5]})
```



	C1	C2	C3
0	1	A	0.1
1	2	B	0.3
2	3	C	0.5

```
C = [[1,'A', 0.1],
     [2,'B', 0.3],
     [3,'C', 0.5]]
df = pd.DataFrame(C, , columns=['C1', 'C2','C3'])
```

Lecture Outline

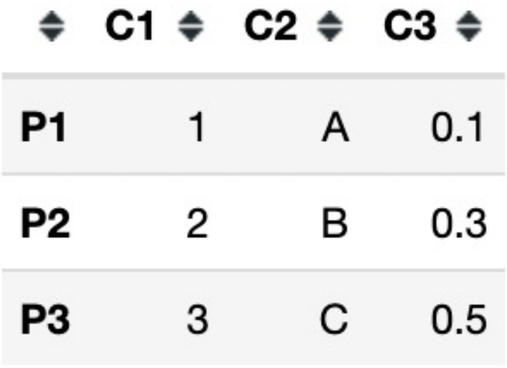
Introduction to Pandas

- Reading a CSV file and viewing Data
- Creating DataFrame and writing to CSV
- Selecting and Indexing
- Creating columns and Assigning values
- Aggregation of Data
- Aggregation of Data by group
- Concatenation and Renaming
- More on Pandas

Pandas Object: DataFrame

- Creating a DataFrame
- Index names

```
df = pd.DataFrame({'C1': [1,2,3],
                   'C2': ['A','B','C'],
                   'C3':[0.1, 0.3, 0.5]}
                   index=[‘P1’, ‘P2’, ‘P3'])
```



	C1	C2	C3
P1	1	A	0.1
P2	2	B	0.3
P3	3	C	0.5

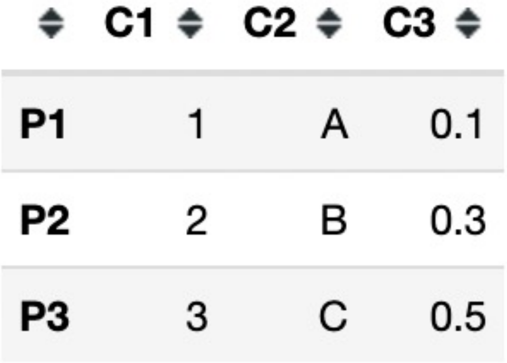
```
C = [[1,'A', 0.1],
     [2,'B', 0.3],
     [3,'C', 0.5]]
df = pd.DataFrame(C, , columns=['C1', 'C2','C3’],
index=[‘P1’, ‘P2’, ‘P3'])
```

Writing DataFrame to csv

- To write a DataFrame to a CSV file pd.to_csv() is used

- file_path = ‘C:/Users/my_path_to_data/data.csv’

- df.to_csv(file_path)



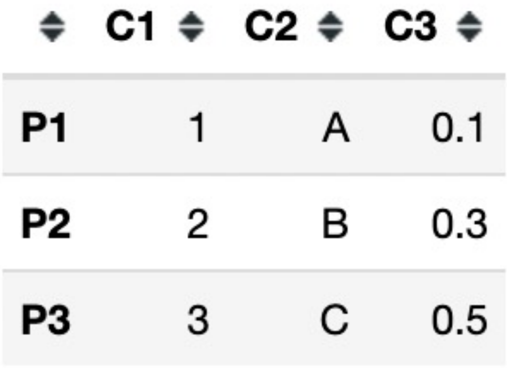
	C1	C2	C3
P1	1	A	0.1
P2	2	B	0.3
P3	3	C	0.5

Lecture Outline

Introduction to Pandas

- Reading a CSV file and viewing Data
- Creating DataFrame and writing to CSV
- Selecting and Indexing
- Creating columns and Assigning values
- Aggregation of Data
- Aggregation of Data by group
- Concatenation and Renaming
- More on Pandas

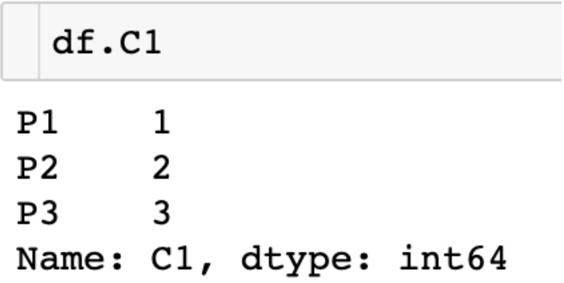
Selecting, Indexing



	C1	C2	C3
P1	1	A	0.1
P2	2	B	0.3
P3	3	C	0.5

- A column of DataFrame can be selected by using column name or index

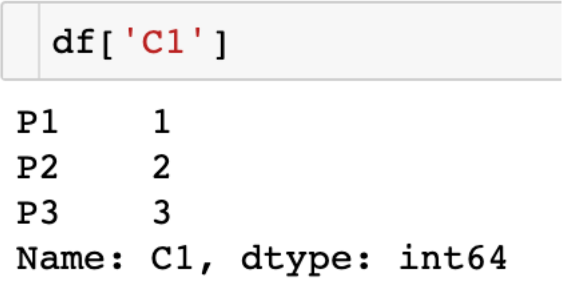
- df.C1



```
df.C1
```

```
P1    1
P2    2
P3    3
Name: C1, dtype: int64
```

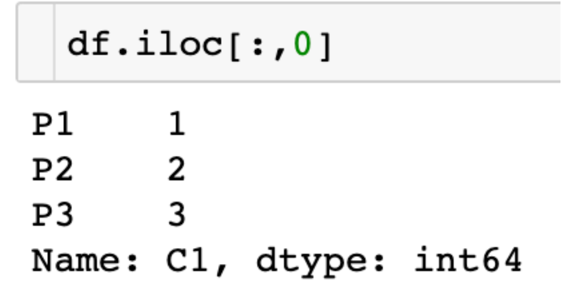
- df['C1']



```
df['C1']
```

```
P1    1
P2    2
P3    3
Name: C1, dtype: int64
```

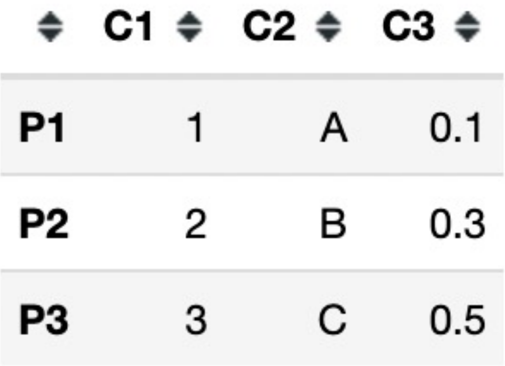
df.iloc[:, 0]



```
df.iloc[:,0]
```

```
P1    1
P2    2
P3    3
Name: C1, dtype: int64
```

Selecting, Indexing

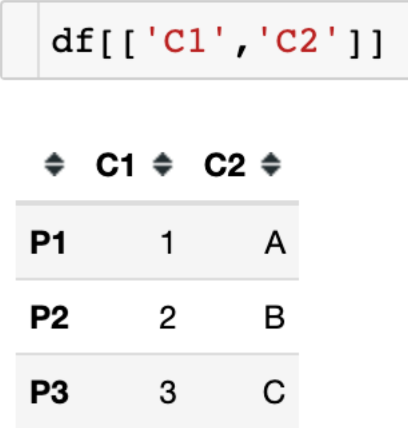


	C1	C2	C3
P1	1	A	0.1
P2	2	B	0.3
P3	3	C	0.5

- Multiple columns of DataFrame can be selected by using column name or index

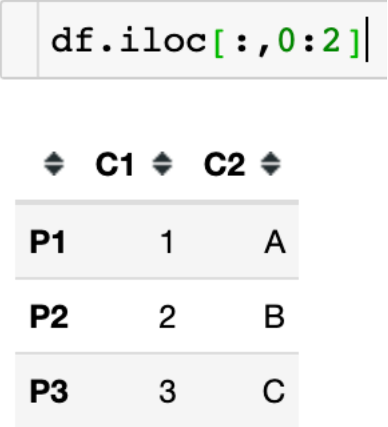
- df[[‘C1’,’C2’]]

df.iloc[:, 0:2]



```
df[['C1','C2']]
```

	C1	C2
P1	1	A
P2	2	B
P3	3	C



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

	C1	C2
P1	1	A
P2	2	B
P3	3	C

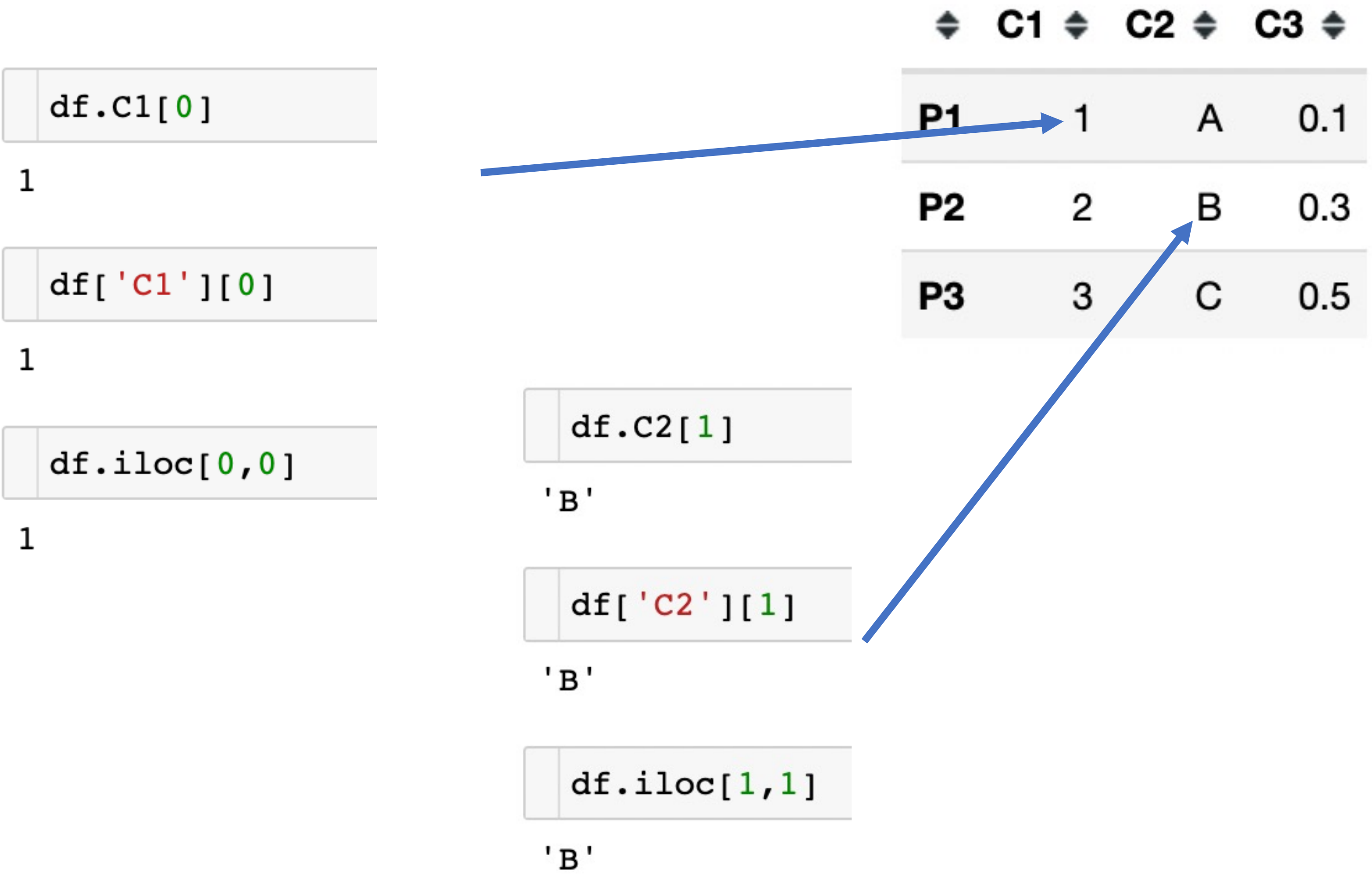
Selecting, Indexing

- An element in a column of DataFrame can be selected using python indexing

- df.C1[0]

- df[‘C1’][0]

- df.iloc[:, 0]



```
df.C1[0]
```

```
1
```

```
df['C1'][0]
```

```
1
```

```
df.iloc[0,0]
```

```
1
```

```
df.C2[1]
```

```
'B'
```

```
df['C2'][1]
```

```
'B'
```

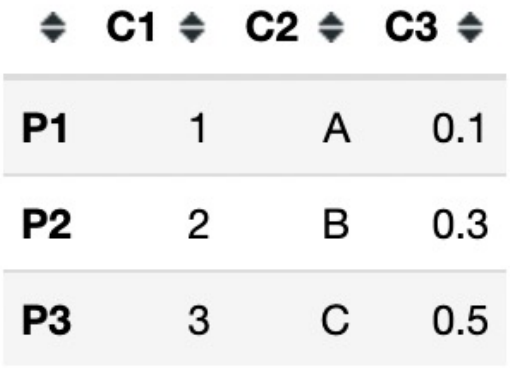
```
df.iloc[1,1]
```

```
'B'
```

	C1	C2	C3
P1	1	A	0.1
P2	2	B	0.3
P3	3	C	0.5

Selecting, Indexing

- In Pandas using .iloc, indexing can be done as numpy

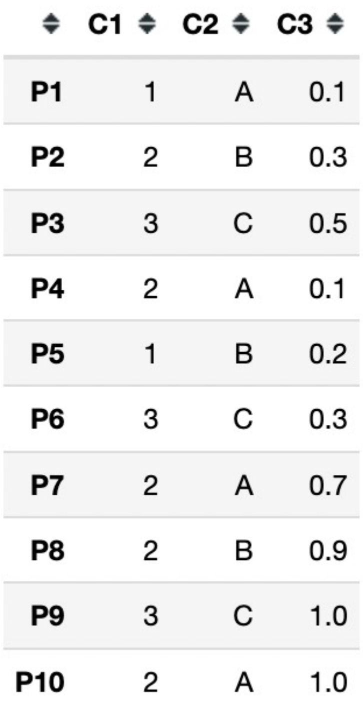


⬍	C1 ⬍	C2 ⬍	C3 ⬍
P1	1	A	0.1
P2	2	B	0.3
P3	3	C	0.5

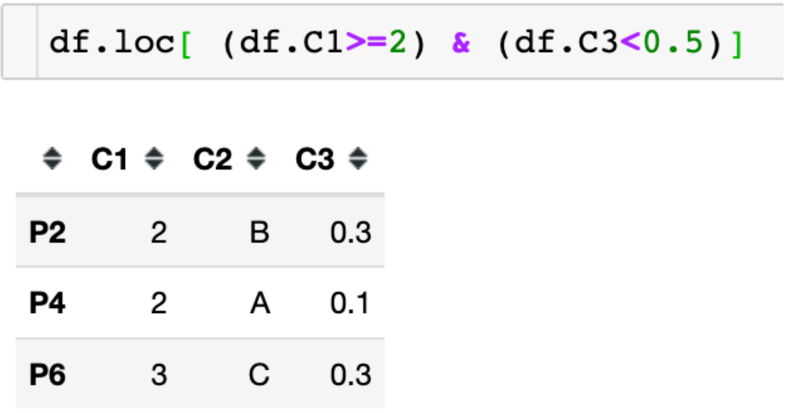
- df.iloc[:, 0]
- df.iloc[::2]
- df.iloc[0]

Selecting, Indexing

- Selecting values based on conditions (sub-table) using .loc
  - Multiple conditions using & and | operators
  - df.loc[df.C1>=2]
  - df.loc[df.C3<0.5]
  - df.loc[(df.C3<0.5) & (df.C1>=2)]

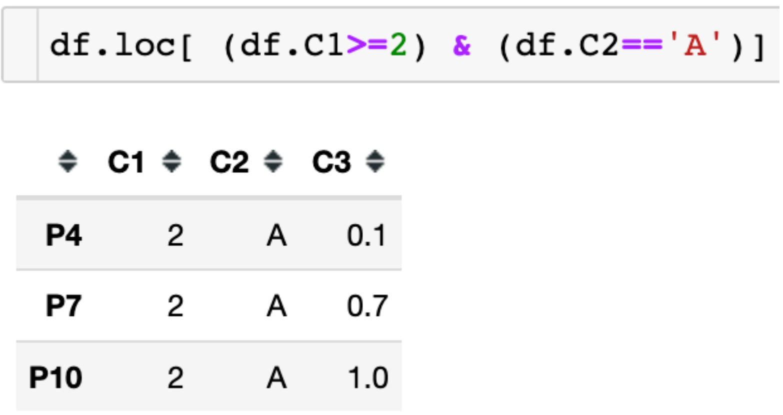


	C1	C2	C3
P1	1	A	0.1
P2	2	B	0.3
P3	3	C	0.5
P4	2	A	0.1
P5	1	B	0.2
P6	3	C	0.3
P7	2	A	0.7
P8	2	B	0.9
P9	3	C	1.0
P10	2	A	1.0



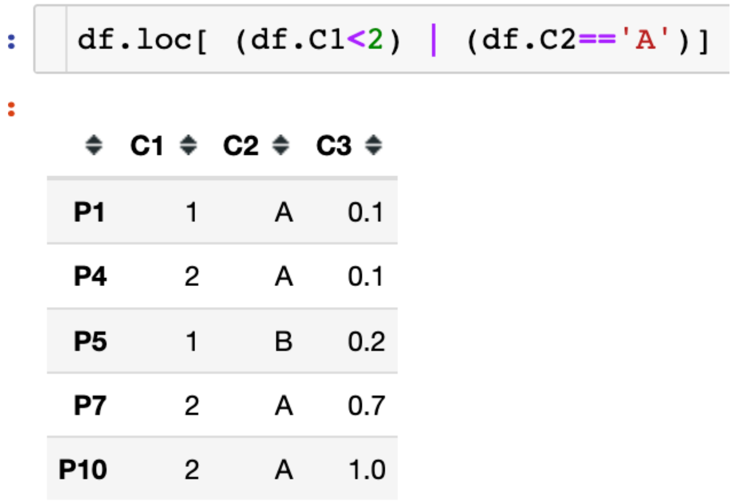
```
df.loc[ (df.C1>=2) & (df.C3<0.5)]
```

	C1	C2	C3
P2	2	B	0.3
P4	2	A	0.1
P6	3	C	0.3



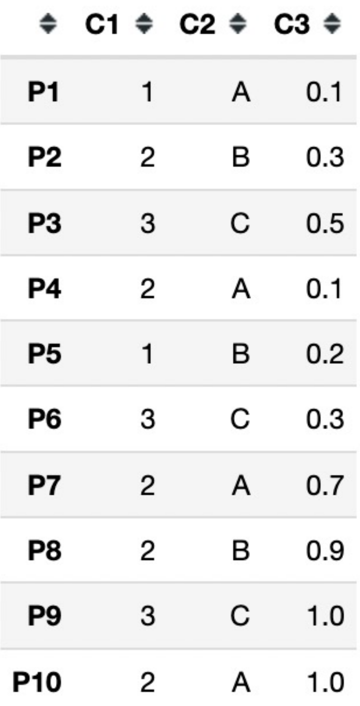
```
df.loc[ (df.C1>=2) & (df.C2=='A')]
```

	C1	C2	C3
P4	2	A	0.1
P7	2	A	0.7
P10	2	A	1.0



```
df.loc[ (df.C1<2) | (df.C2=='A')]
```

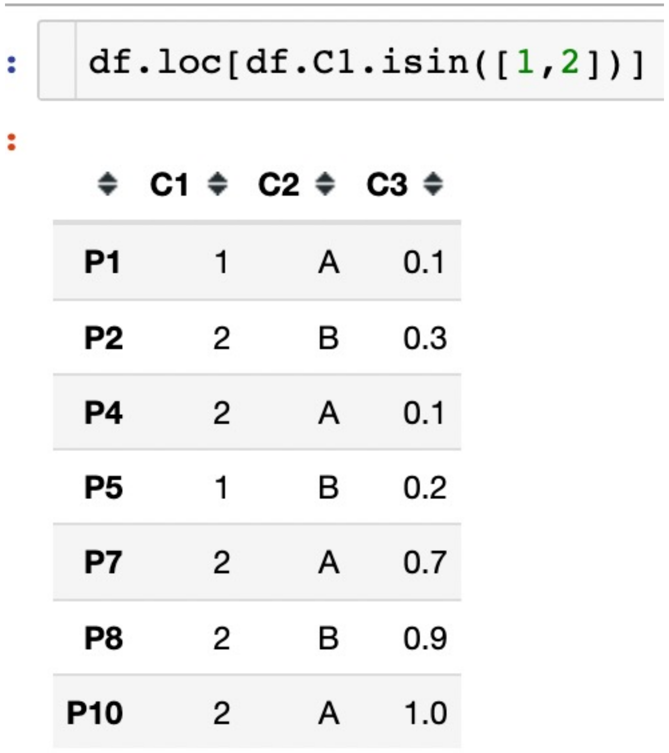
	C1	C2	C3
P1	1	A	0.1
P4	2	A	0.1
P5	1	B	0.2
P7	2	A	0.7
P10	2	A	1.0



	C1	C2	C3
P1	1	A	0.1
P2	2	B	0.3
P3	3	C	0.5
P4	2	A	0.1
P5	1	B	0.2
P6	3	C	0.3
P7	2	A	0.7
P8	2	B	0.9
P9	3	C	1.0
P10	2	A	1.0

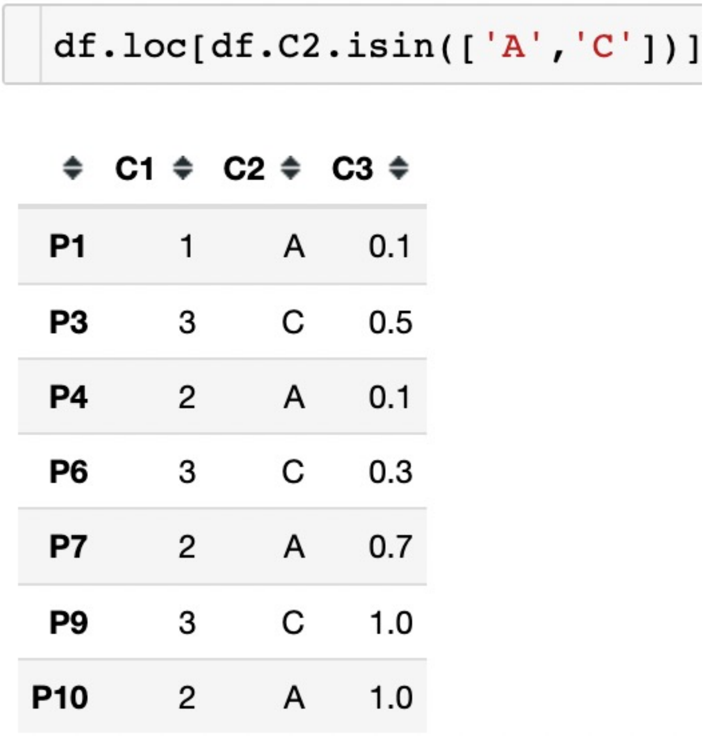
Selecting, Indexing

- Selecting values based on conditions (sub-table) using .loc
  - Multiple conditions using **isin**



```
df.loc[df.C1.isin([1,2])]
```

	C1	C2	C3
P1	1	A	0.1
P2	2	B	0.3
P4	2	A	0.1
P5	1	B	0.2
P7	2	A	0.7
P8	2	B	0.9
P10	2	A	1.0

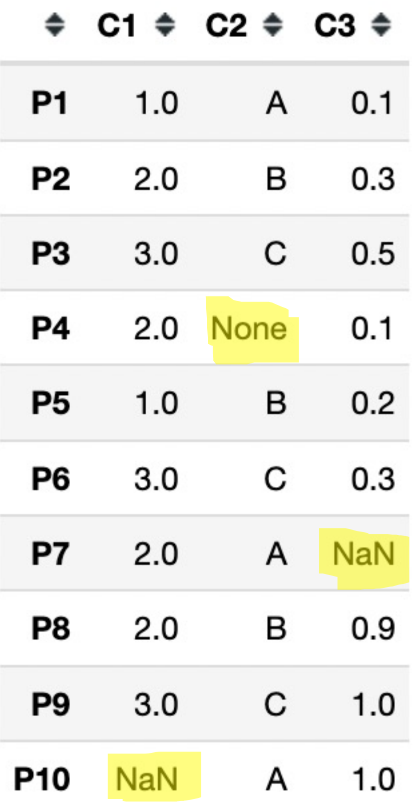


```
df.loc[df.C2.isin(['A','C'])]
```

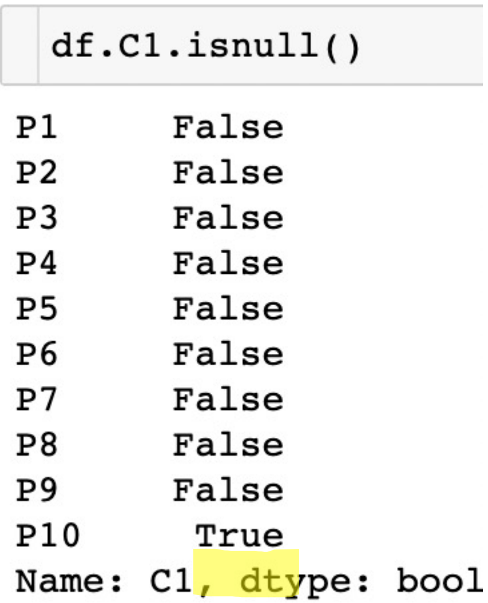
	C1	C2	C3
P1	1	A	0.1
P3	3	C	0.5
P4	2	A	0.1
P6	3	C	0.3
P7	2	A	0.7
P9	3	C	1.0
P10	2	A	1.0

Selecting, Indexing

- Selecting values based on conditions (sub-table) using .loc
  - Missing values using **isnull** and **notnull**

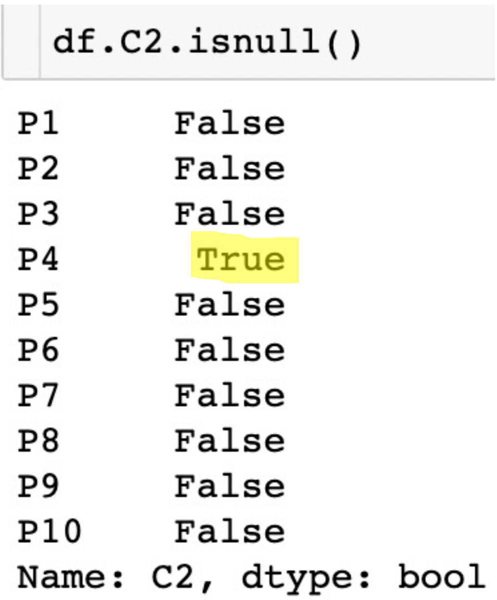


	C1	C2	C3
P1	1.0	A	0.1
P2	2.0	B	0.3
P3	3.0	C	0.5
P4	2.0	None	0.1
P5	1.0	B	0.2
P6	3.0	C	0.3
P7	2.0	A	NaN
P8	2.0	B	0.9
P9	3.0	C	1.0
P10	NaN	A	1.0



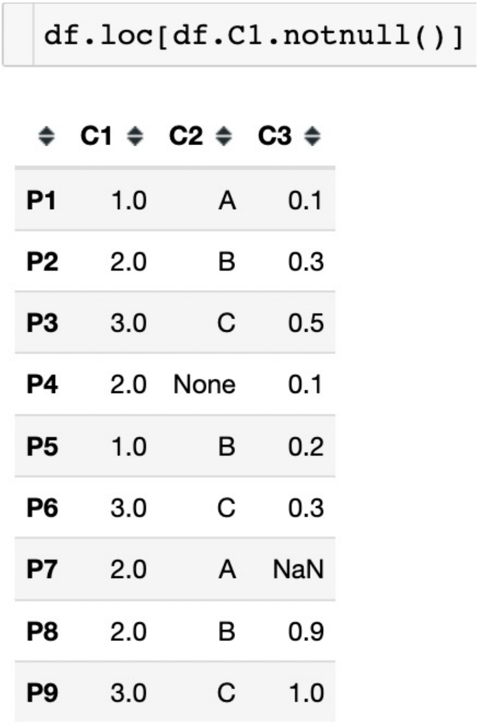
```
df.C1.isnull()
```

```
P1       False
P2       False
P3       False
P4       False
P5       False
P6       False
P7       False
P8       False
P9       False
P10       True
Name: C1, dtype: bool
```



```
df.C2.isnull()
```

```
P1       False
P2       False
P3       False
P4        True
P5       False
P6       False
P7       False
P8       False
P9       False
P10      False
Name: C2, dtype: bool
```



```
df.loc[df.C1.notnull()]
```

	C1	C2	C3
P1	1.0	A	0.1
P2	2.0	B	0.3
P3	3.0	C	0.5
P4	2.0	None	0.1
P5	1.0	B	0.2
P6	3.0	C	0.3
P7	2.0	A	NaN
P8	2.0	B	0.9
P9	3.0	C	1.0

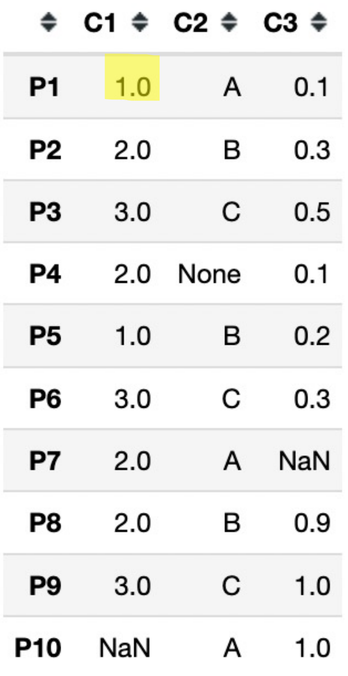
Lecture Outline

Introduction to Pandas

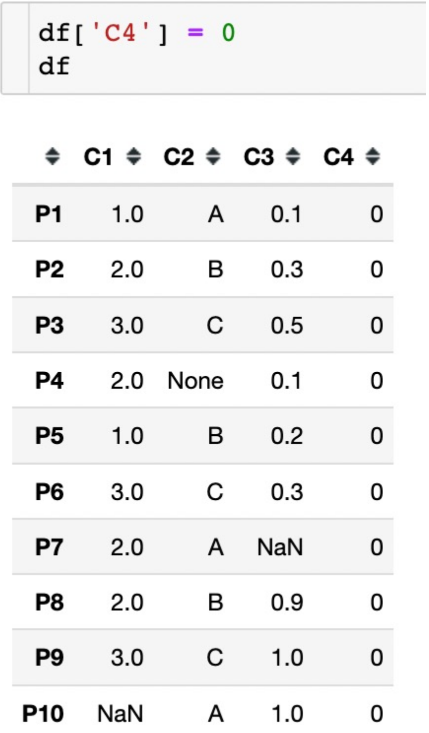
- Reading a CSV file and viewing Data
- Creating DataFrame and writing to CSV
- Selecting and Indexing
- Creating columns and Assigning values
- Aggregation of Data
- Aggregation of Data by group
- Concatenation and Renaming
- More on Pandas

Creating new column and Assigning

- To create a new column, dictionary type approach is used
  - df[‘C4’] = [1,2,1,2, …]
  - df[‘C4’] = 0
  - df[‘C5’] = None

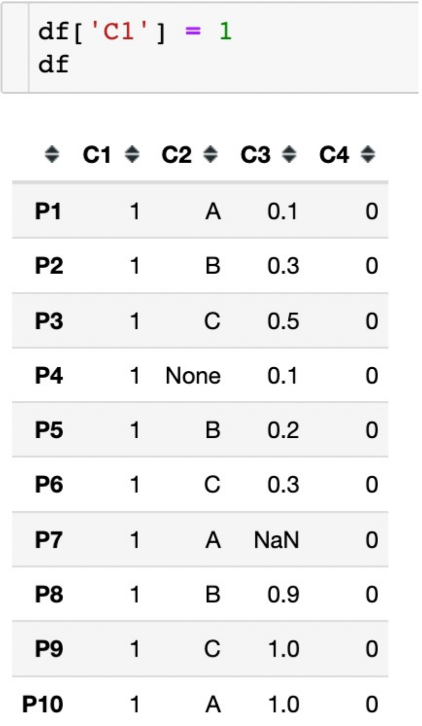


	C1	C2	C3
P1	1.0	A	0.1
P2	2.0	B	0.3
P3	3.0	C	0.5
P4	2.0	None	0.1
P5	1.0	B	0.2
P6	3.0	C	0.3
P7	2.0	A	NaN
P8	2.0	B	0.9
P9	3.0	C	1.0
P10	NaN	A	1.0



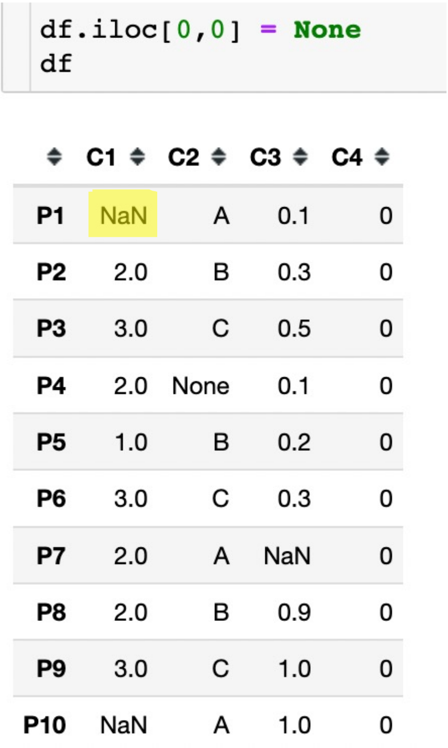
```
df['C4'] = 0
df
```

	C1	C2	C3	C4
P1	1.0	A	0.1	0
P2	2.0	B	0.3	0
P3	3.0	C	0.5	0
P4	2.0	None	0.1	0
P5	1.0	B	0.2	0
P6	3.0	C	0.3	0
P7	2.0	A	NaN	0
P8	2.0	B	0.9	0
P9	3.0	C	1.0	0
P10	NaN	A	1.0	0



```
df['C1'] = 1
df
```

	C1	C2	C3	C4
P1	1	A	0.1	0
P2	1	B	0.3	0
P3	1	C	0.5	0
P4	1	None	0.1	0
P5	1	B	0.2	0
P6	1	C	0.3	0
P7	1	A	NaN	0
P8	1	B	0.9	0
P9	1	C	1.0	0
P10	1	A	1.0	0



```
df.iloc[0,0] = None
df
```

	C1	C2	C3	C4
P1	NaN	A	0.1	0
P2	2.0	B	0.3	0
P3	3.0	C	0.5	0
P4	2.0	None	0.1	0
P5	1.0	B	0.2	0
P6	3.0	C	0.3	0
P7	2.0	A	NaN	0
P8	2.0	B	0.9	0
P9	3.0	C	1.0	0
P10	NaN	A	1.0	0

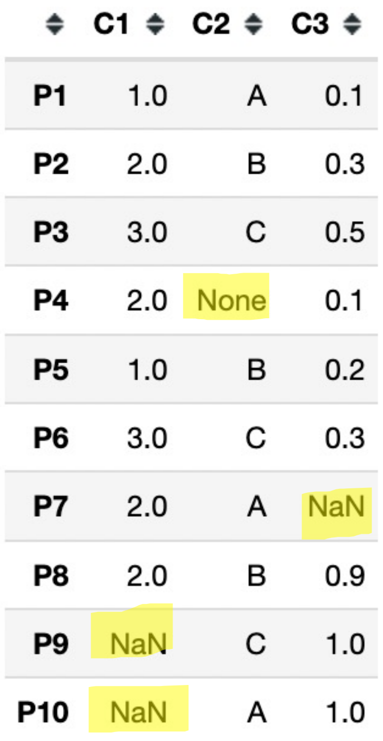
Lecture Outline

Introduction to Pandas

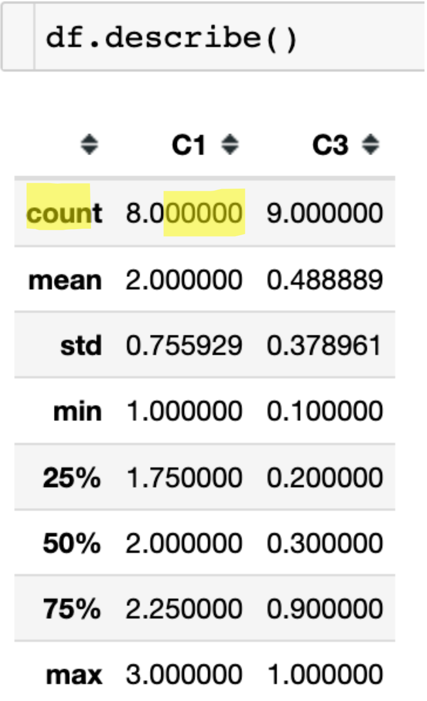
- Reading a CSV file and viewing Data
- Creating DataFrame and writing to CSV
- Selecting and Indexing
- Creating columns and Assigning values
- Aggregation of Data
- Aggregation of Data by group
- Concatenation and Renaming
- More on Pandas

Aggregation of Data

- Aggregation of data includes computing statistics such as minimum, maximum, mean, standard deviation and counts
- Pandas make is easy to compute such statistics using describe()
- It can be done for entire DataFrame or a column
- With entire DataFrame, It works only on numerical columns

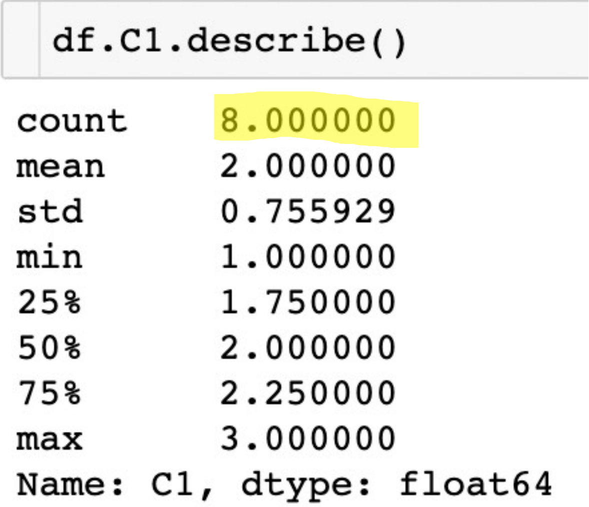


	C1	C2	C3
P1	1.0	A	0.1
P2	2.0	B	0.3
P3	3.0	C	0.5
P4	2.0	None	0.1
P5	1.0	B	0.2
P6	3.0	C	0.3
P7	2.0	A	NaN
P8	2.0	B	0.9
P9	NaN	C	1.0
P10	NaN	A	1.0



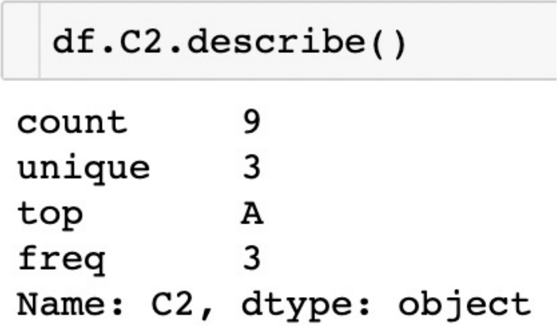
```
df.describe()
```

	C1	C3
count	8.000000	9.000000
mean	2.000000	0.488889
std	0.755929	0.378961
min	1.000000	0.100000
25%	1.750000	0.200000
50%	2.000000	0.300000
75%	2.250000	0.900000
max	3.000000	1.000000



```
df.C1.describe()
```

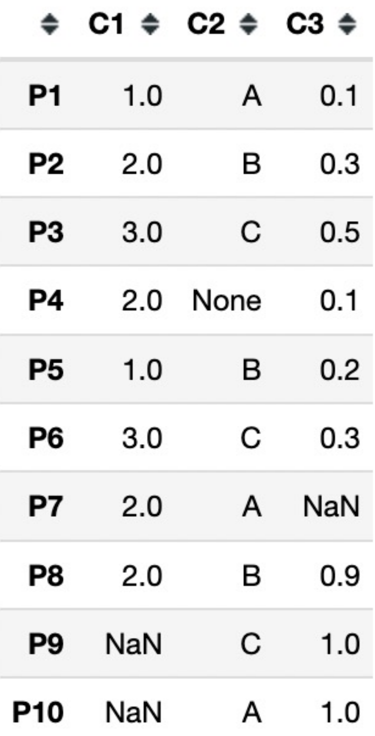
```
count     8.000000
mean      2.000000
std       0.755929
min       1.000000
25%       1.750000
50%       2.000000
75%       2.250000
max       3.000000
Name: C1, dtype: float64
```



```
df.C2.describe()
```

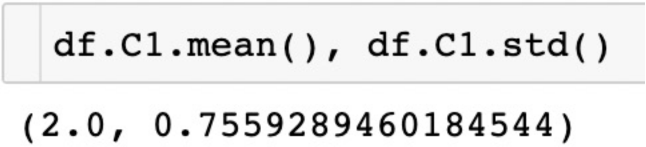
```
count      9
unique     3
top        A
freq       3
Name: C2, dtype: object
```

Aggregation of Data



	C1	C2	C3
P1	1.0	A	0.1
P2	2.0	B	0.3
P3	3.0	C	0.5
P4	2.0	None	0.1
P5	1.0	B	0.2
P6	3.0	C	0.3
P7	2.0	A	NaN
P8	2.0	B	0.9
P9	NaN	C	1.0
P10	NaN	A	1.0

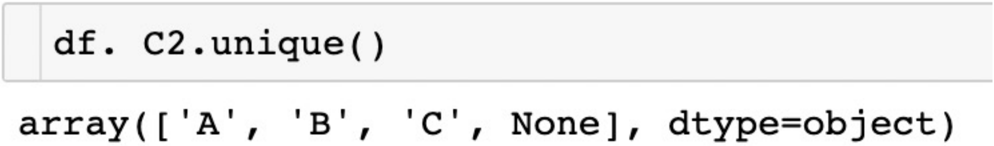
- A single aggregation metric can be computed from a column too.
- df.C1.mean(),   df.C1.std()



```
df.C1.mean(), df.C1.std()
```

```
(2.0, 0.7559289460184544)
```

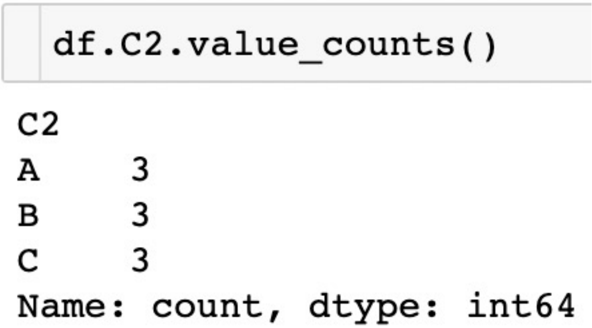
- df. C2.unique()



```
df. C2.unique()
```

```
array(['A', 'B', 'C', None], dtype=object)
```

- df.C2.value_counts()



```
df.C2.value_counts()
```

```
C2
A    3
B    3
C    3
Name: count, dtype: int64
```

Lecture Outline

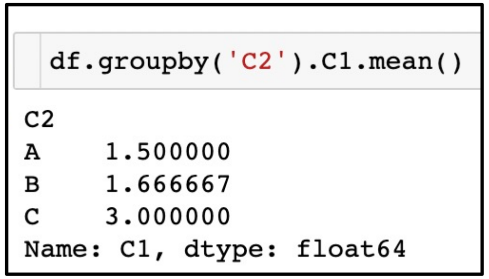
Introduction to Pandas

- Reading a CSV file and viewing Data
- Creating DataFrame and writing to CSV
- Selecting and Indexing
- Creating columns and Assigning values
- Aggregation of Data
- Aggregation of Data by group
- Concatenation and Renaming
- More on Pandas

Aggregation of Data by groups

- Aggregation of Data can be done group-wise using ‘groupby’.

  For example mean of C1 corresponds to names in C2

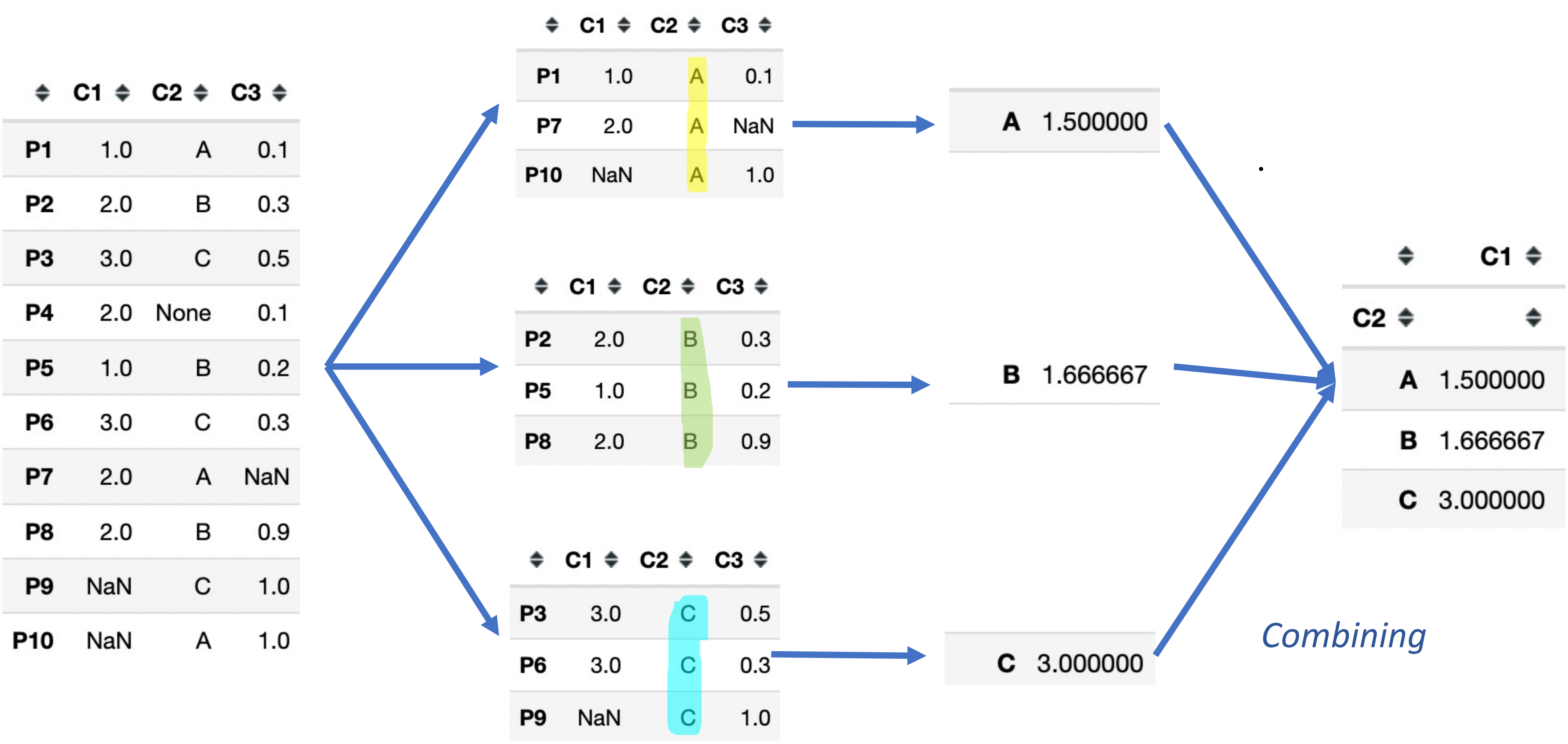


```
df.groupby('C2').C1.mean()
```

```
C2
A     1.500000
B     1.666667
C     3.000000
Name: C1, dtype: float64
```

Splitting

Aggregating (mean of column C2)



	C1	C2	C3
P1	1.0	A	0.1
P2	2.0	B	0.3
P3	3.0	C	0.5
P4	2.0	None	0.1
P5	1.0	B	0.2
P6	3.0	C	0.3
P7	2.0	A	NaN
P8	2.0	B	0.9
P9	NaN	C	1.0
P10	NaN	A	1.0

	C1	C2	C3
P1	1.0	A	0.1
P7	2.0	A	NaN
P10	NaN	A	1.0

	C1	C2	C3
P2	2.0	B	0.3
P5	1.0	B	0.2
P8	2.0	B	0.9

	C1	C2	C3
P3	3.0	C	0.5
P6	3.0	C	0.3
P9	NaN	C	1.0

A	1.500000

B	1.666667

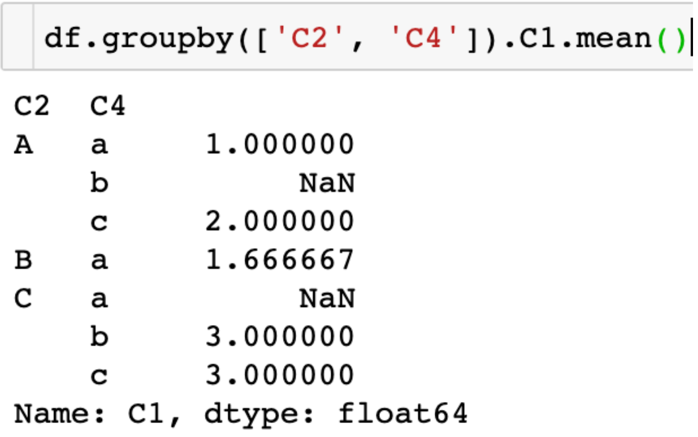
C	3.000000

	C1
C2	
A	1.500000
B	1.666667
C	3.000000

Combining

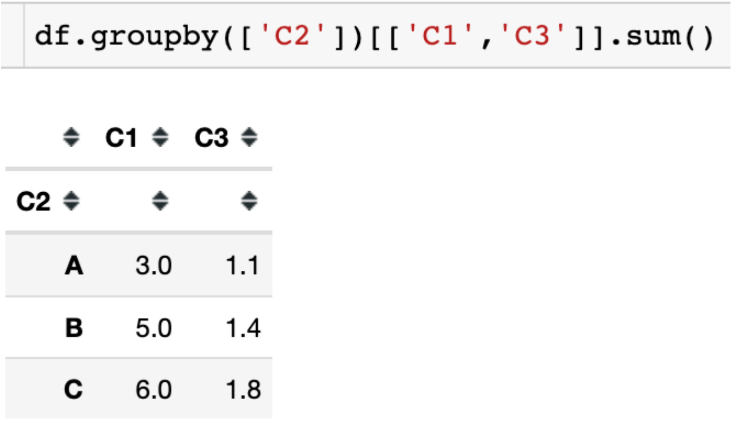
Aggregation of Data by groups

- Group by multiple columns (multi-index).



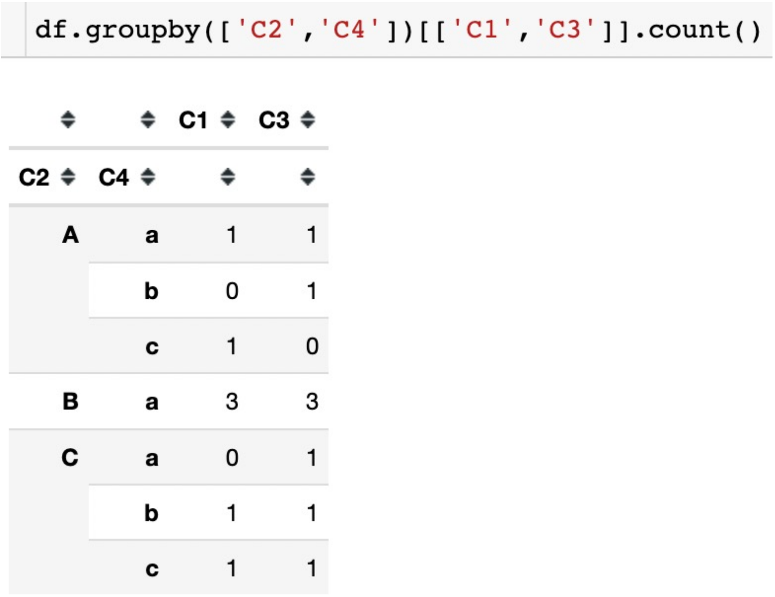
```
df.groupby(['C2', 'C4']).C1.mean()
```

```
C2  C4
A   a       1.000000
    b            NaN
    c       2.000000
B   a       1.666667
C   a            NaN
    b       3.000000
    c       3.000000
Name: C1, dtype: float64
```



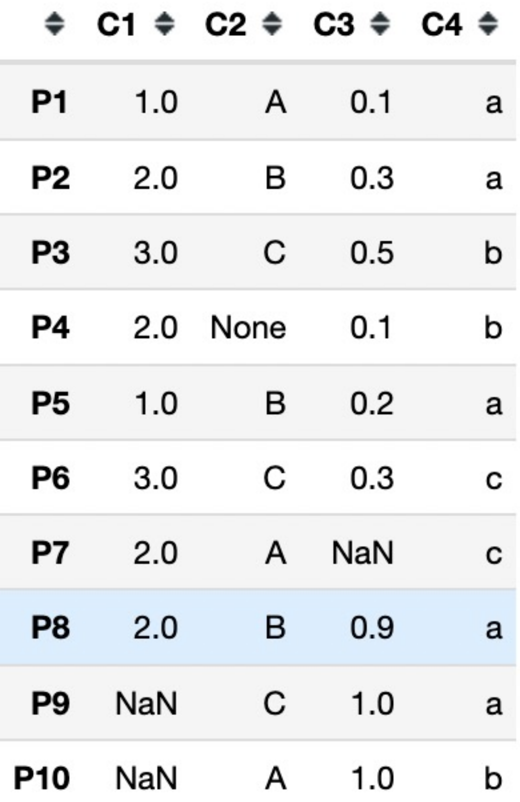
```
df.groupby(['C2'])[['C1','C3']].sum()
```

	C1	C3
C2		
A	3.0	1.1
B	5.0	1.4
C	6.0	1.8



```
df.groupby(['C2','C4'])[['C1','C3']].count()
```

		C1	C3
C2	**C4**		
A	**a**	1	1
	b	0	1
	c	1	0
B	**a**	3	3
C	**a**	0	1
	b	1	1
	c	1	1



	C1	C2	C3	C4
P1	1.0	A	0.1	a
P2	2.0	B	0.3	a
P3	3.0	C	0.5	b
P4	2.0	None	0.1	b
P5	1.0	B	0.2	a
P6	3.0	C	0.3	c
P7	2.0	A	NaN	c
P8	2.0	B	0.9	a
P9	NaN	C	1.0	a
P10	NaN	A	1.0	b

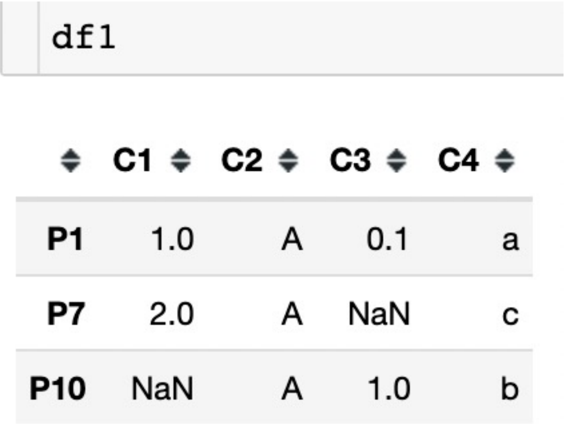
Lecture Outline

Introduction to Pandas

- Reading a CSV file and viewing Data
- Creating DataFrame and writing to CSV
- Selecting and Indexing
- Creating columns and Assigning values
- Aggregation of Data
- Aggregation of Data by group
- Concatenation and Renaming
- More on Pandas

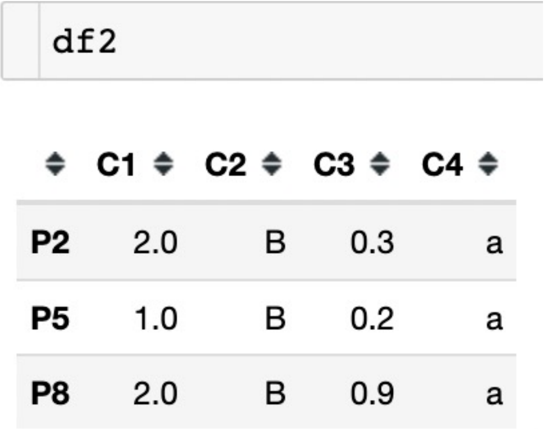
Concatenation of DataFrames

- Two dataFrames can be combined using pd.concat()



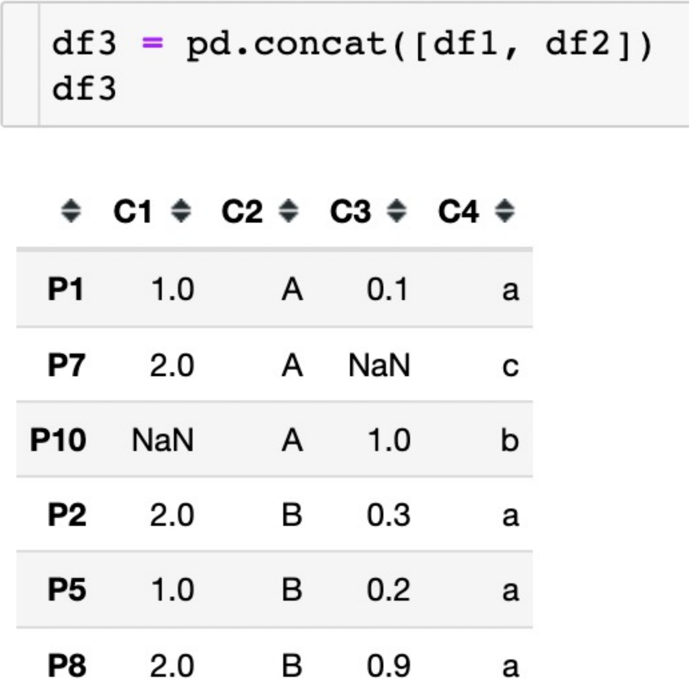
```
df1
```

⬍	C1 ⬍	C2 ⬍	C3 ⬍	C4 ⬍
P1	1.0	A	0.1	a
P7	2.0	A	NaN	c
P10	NaN	A	1.0	b



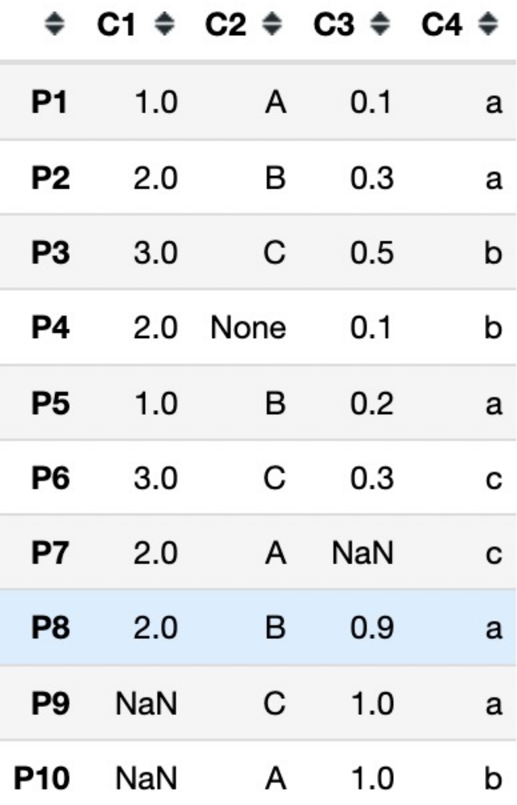
```
df2
```

⬍	C1 ⬍	C2 ⬍	C3 ⬍	C4 ⬍
P2	2.0	B	0.3	a
P5	1.0	B	0.2	a
P8	2.0	B	0.9	a



```
df3 = pd.concat([df1, df2])
df3
```

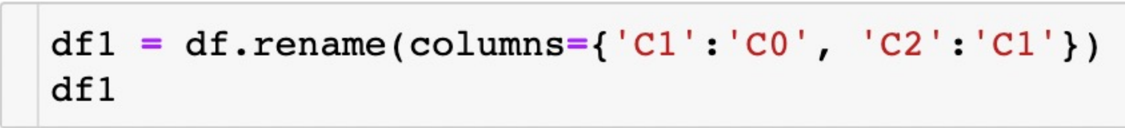
⬍	C1 ⬍	C2 ⬍	C3 ⬍	C4 ⬍
P1	1.0	A	0.1	a
P7	2.0	A	NaN	c
P10	NaN	A	1.0	b
P2	2.0	B	0.3	a
P5	1.0	B	0.2	a
P8	2.0	B	0.9	a



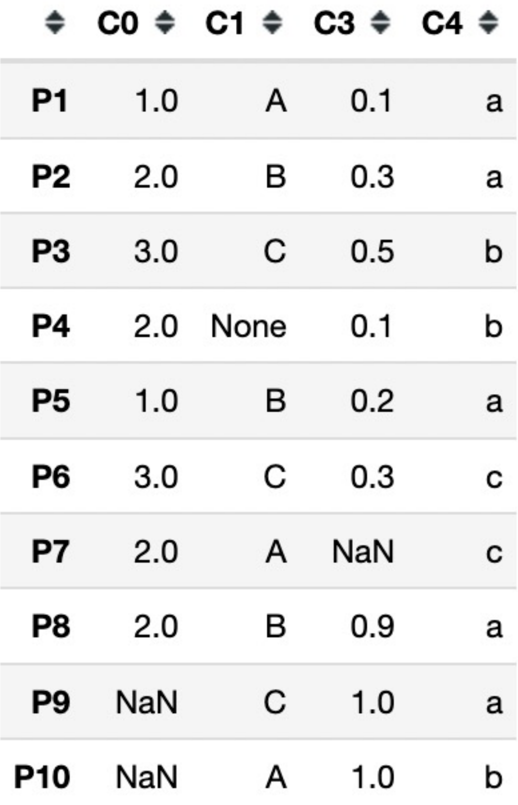
	C1	C2	C3	C4
P1	1.0	A	0.1	a
P2	2.0	B	0.3	a
P3	3.0	C	0.5	b
P4	2.0	None	0.1	b
P5	1.0	B	0.2	a
P6	3.0	C	0.3	c
P7	2.0	A	NaN	c
P8	2.0	B	0.9	a
P9	NaN	C	1.0	a
P10	NaN	A	1.0	b

Renaming columns

- Renaming columns



```
df1 = df.rename(columns={'C1':'C0', 'C2':'C1'})
df1
```



	C0	C1	C3	C4
P1	1.0	A	0.1	a
P2	2.0	B	0.3	a
P3	3.0	C	0.5	b
P4	2.0	None	0.1	b
P5	1.0	B	0.2	a
P6	3.0	C	0.3	c
P7	2.0	A	NaN	c
P8	2.0	B	0.9	a
P9	NaN	C	1.0	a
P10	NaN	A	1.0	b

Lecture Outline

Introduction to Pandas

- Reading a CSV file and viewing Data
- Creating DataFrame and writing to CSV
- Selecting and Indexing
- Creating columns and Assigning values
- Aggregation of Data
- Aggregation of Data by group
- Concatenation and Renaming
- More on Pandas

More on Pandas

- Filling nan values with ‘unknown’: df.fillna("Unknown")
- For applying function on a column: df.apply () and lambda operation
- Sort DataFrame by a column: df.sort_values(by='C2’)
- Multiple-aggregation in groupby: df.groupby(‘C2’).aggregate(['min', np.median, max])
- Pandas Series Object: pd.Series
- Removing rows with nan : df.dropna
- Plots in Pandas : df.C1.plot(), df.C3.plot(kind='bar')

- Next !!!
  - 4.2: Lab session on Visualisation and Pandas

Queen Mary
University of London