AI VIETNAM
All-in-One Course

Data Analysis Using Pandas

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

Outline

- Series
- DataFrame
- Data Loading
- Data Visualization
- Case studies

House price data

Feature	Label
area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

price = a $* area + b$

Advertising data

Features			Label
TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9

Model: $\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$

Sale = $w_1 * TV + w_2 * Radio + w_3 * Newspaper + b$

Boston House Price Data

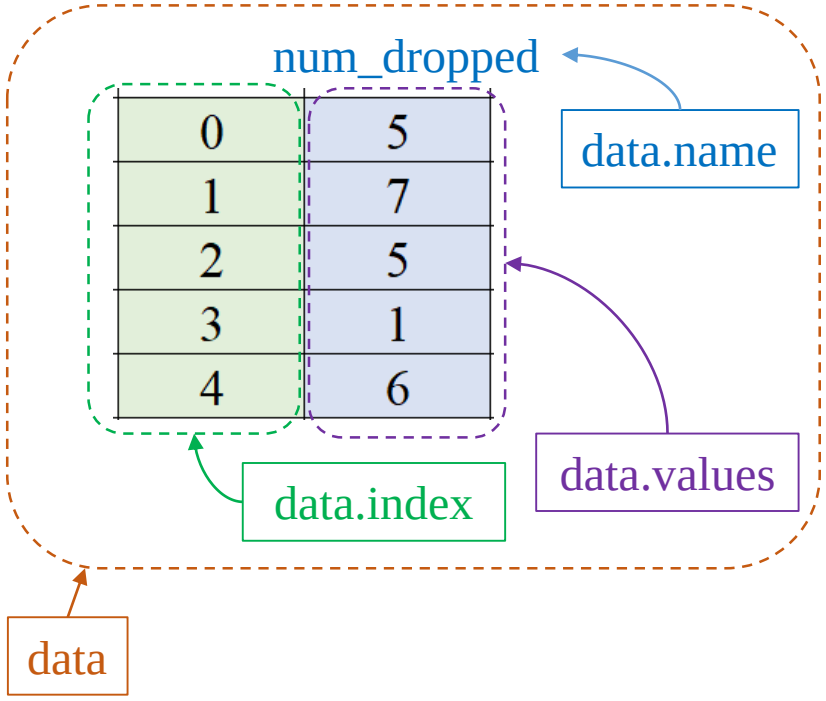
Features													Label
crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9

medv = $w_1 * x_1 + \cdots + w_{13} * x_{13} + b$

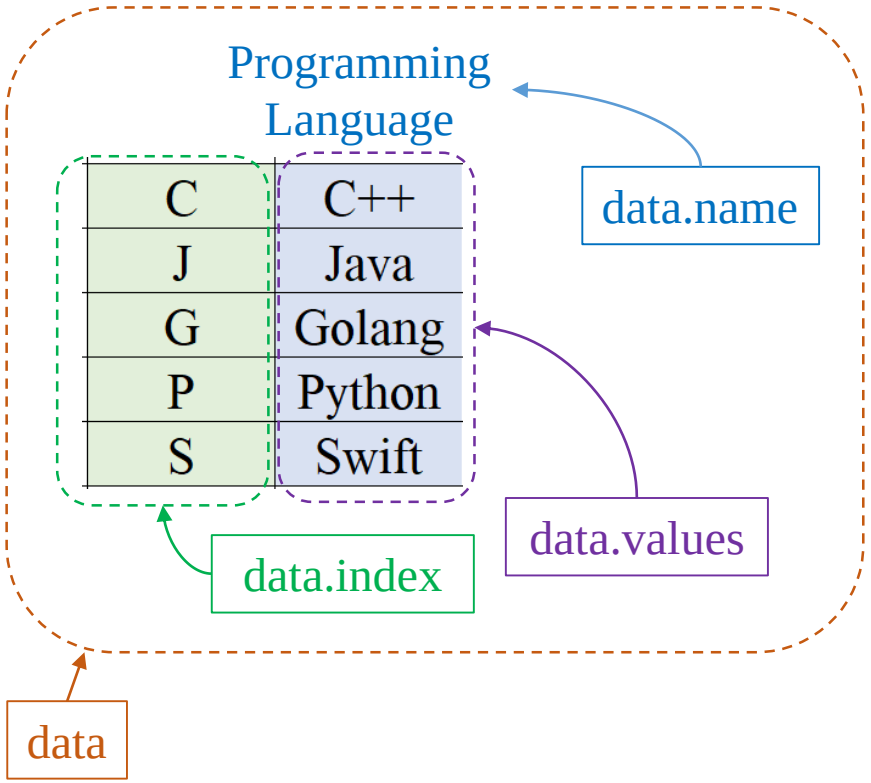
Series in Pandas

❖ Create a series

```python
import pandas as pd

data = pd.Series([5, 7, 5, 1, 6],
                 name='num_dropped')
```



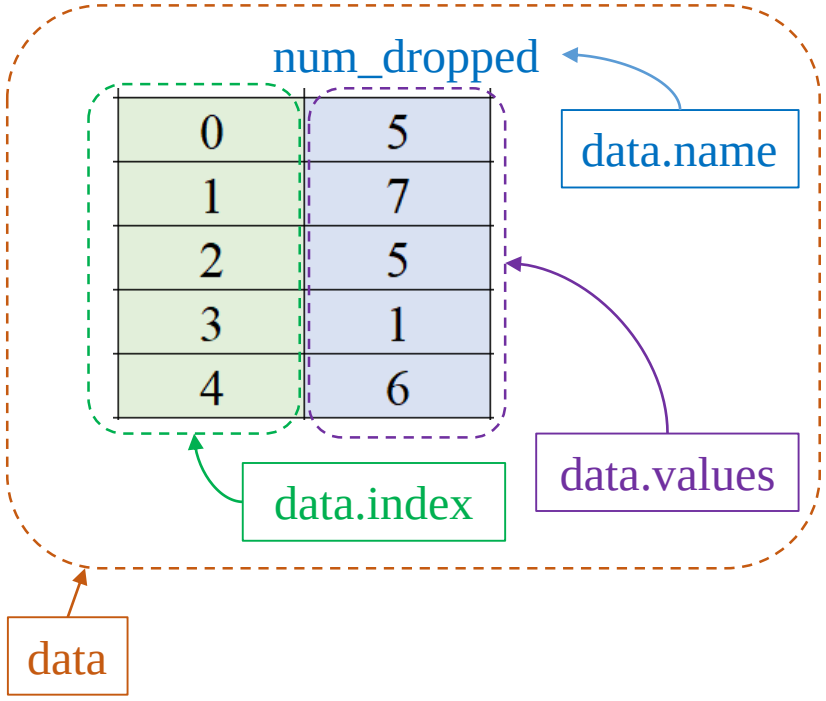

```python
import pandas as pd

data = pd.Series(['C++', 'Golang', 'Java', 'Python', 'Swift'],
                 index=list('CGJPS'),
                 name='Programming Language')
```



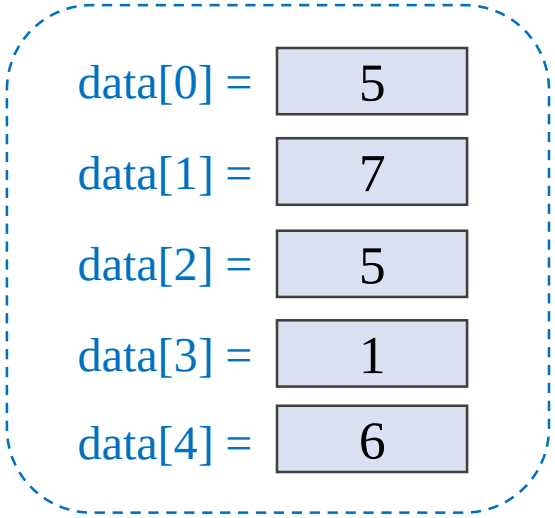
Series in Pandas

❖ Create a series

```
1  import pandas as pd
2
3  data = pd.Series([5, 7, 5, 1, 6],
4                   name='num_dropped')
```



❖ Get rows




```
for v in data:
    print(v)

#----------------

print(data[0])
print(data[1])
print(data[2])
print(data[3])
print(data[4])
```

```
print(data.loc[0])
print(data.loc[1])
print(data.loc[2])
print(data.loc[3])
print(data.loc[4])
```

result = data.loc[2:3]

result =

2	5
3	1

result = data[2:3]

result =

2	5

result = data[data.between(3, 6)]

result =

0	5
2	5
4	6

result = data[data>5]

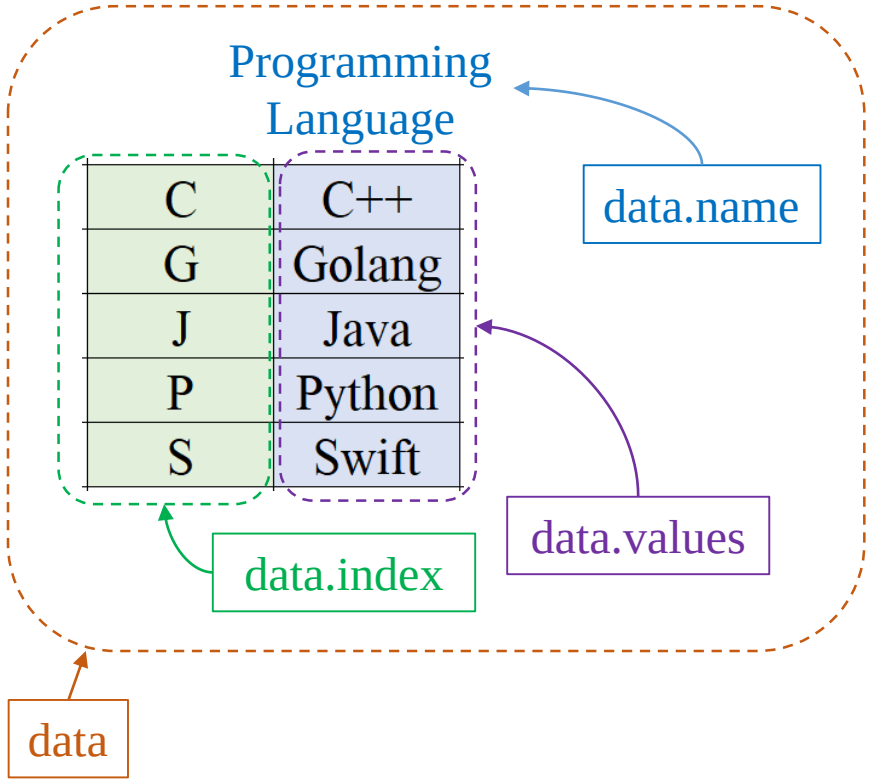
result =

1	7
4	6

Series in Pandas

❖ Create a series

```
1  import pandas as pd
2
3  data = pd.Series(['C++', 'Golang', 'Java', 'Python', 'Swift'],
4                   index=list('CGJPS'),
5                   name='Programming Language')
```



❖ Get rows

result = data['C':'P']

result =

C	C++
G	Golang
J	Java
P	Python

result = data[['C', 'P']]

result =

C	C++
P	Python

result = data[2:4]

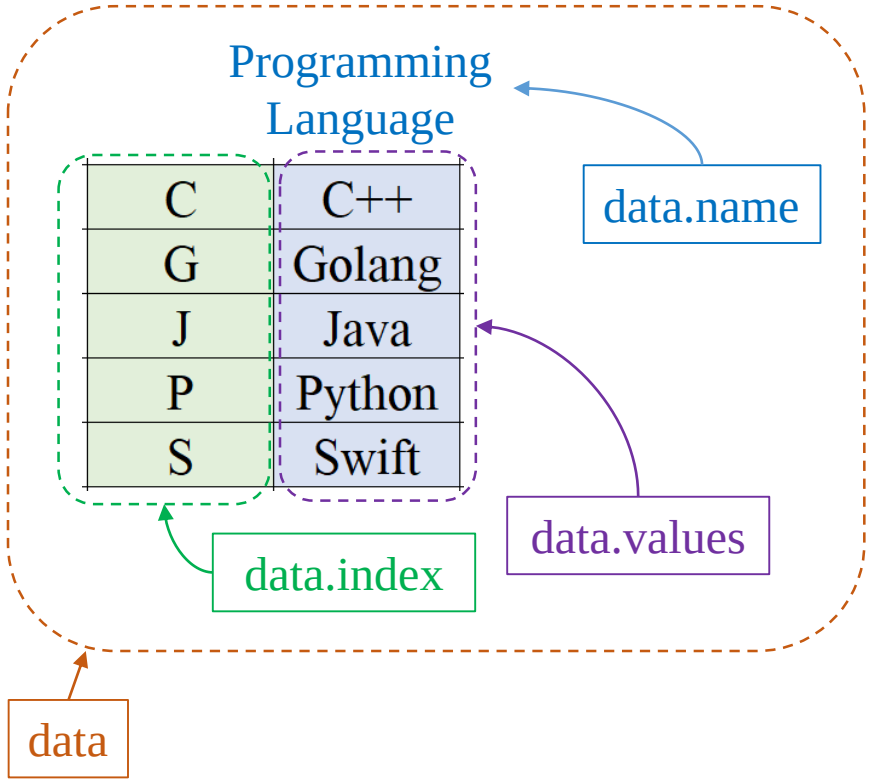
result =

J	Java
P	Python

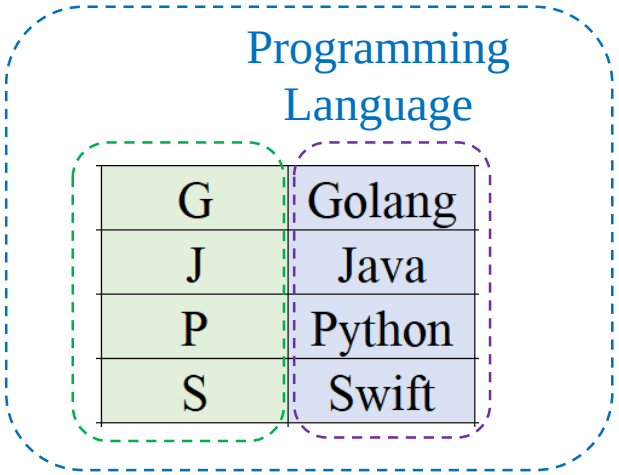
Series in Pandas

❖ Drop a row

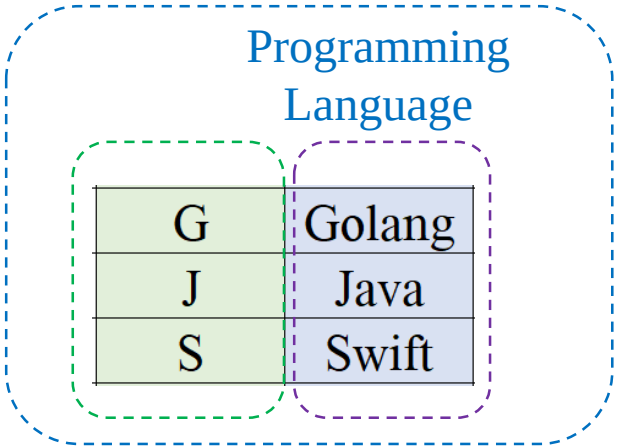
```
1  import pandas as pd
2
3  data = pd.Series(['C++', 'Golang', 'Java', 'Python', 'Swift'],
4                   index=list('CGJPS'),
5                   name='Programming Language')
```



data.drop('C')



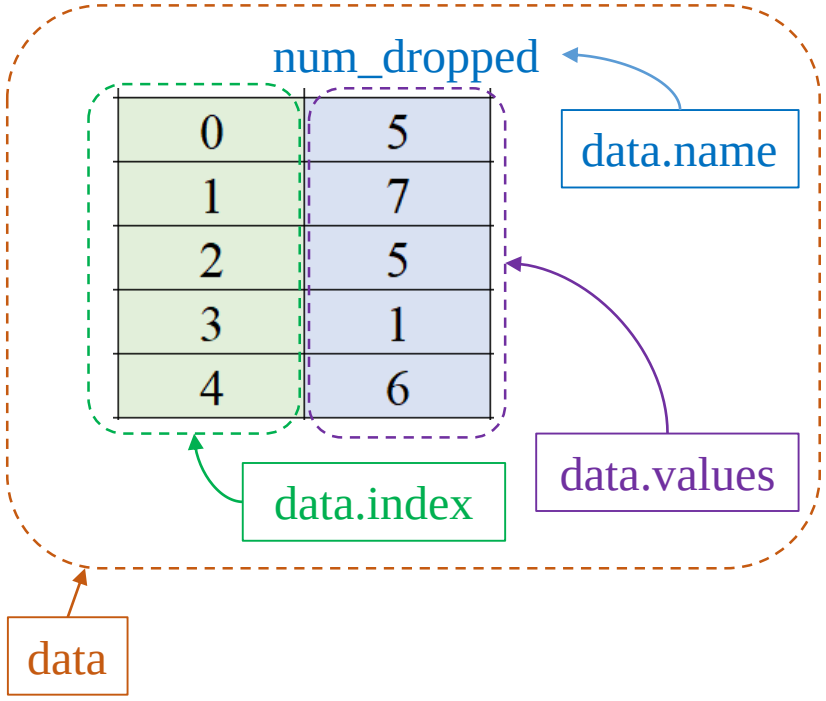
data.drop(['C', 'P'])

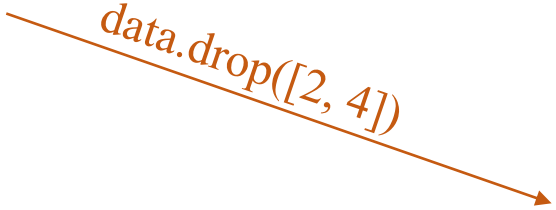


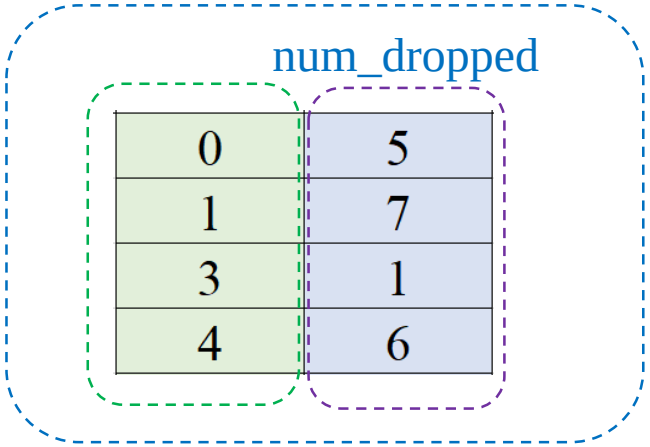
Series in Pandas

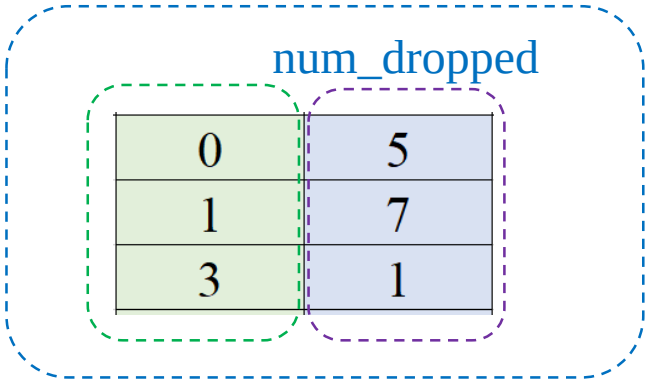
❖ Drop a row

```
1  import pandas as pd
2
3  data = pd.Series([5, 7, 5, 1, 6],
4                   name='num_dropped')
```









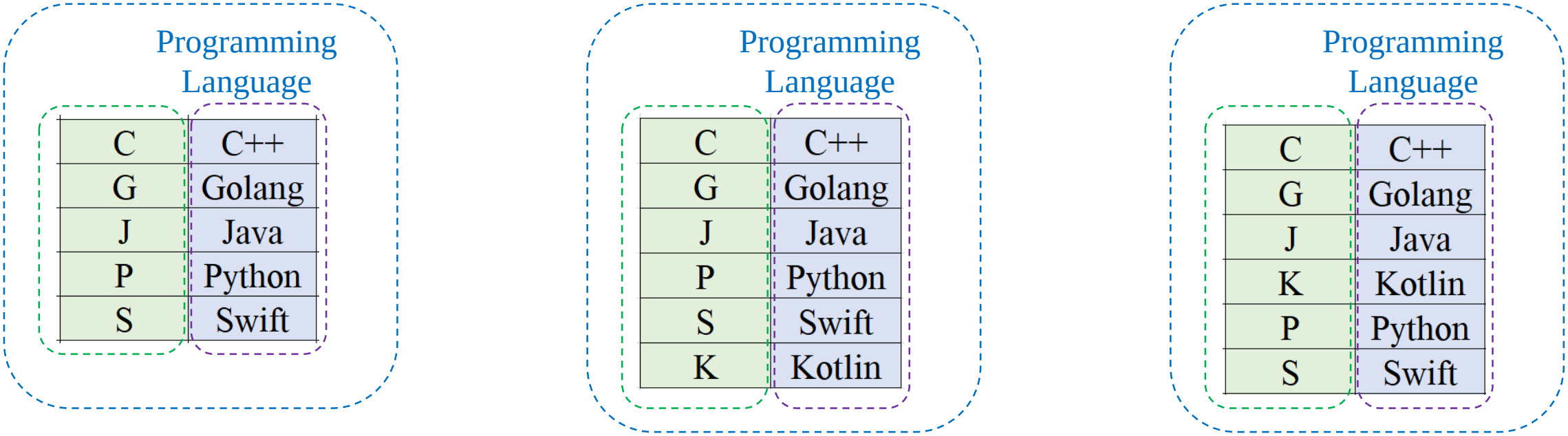
Series in Pandas

❖ Insert a row

```python
import pandas as pd

data = pd.Series(['C++', 'Golang', 'Java', 'Python', 'Swift'],
                 index=list('CGJPS'),
                 name='Programming Language')
```

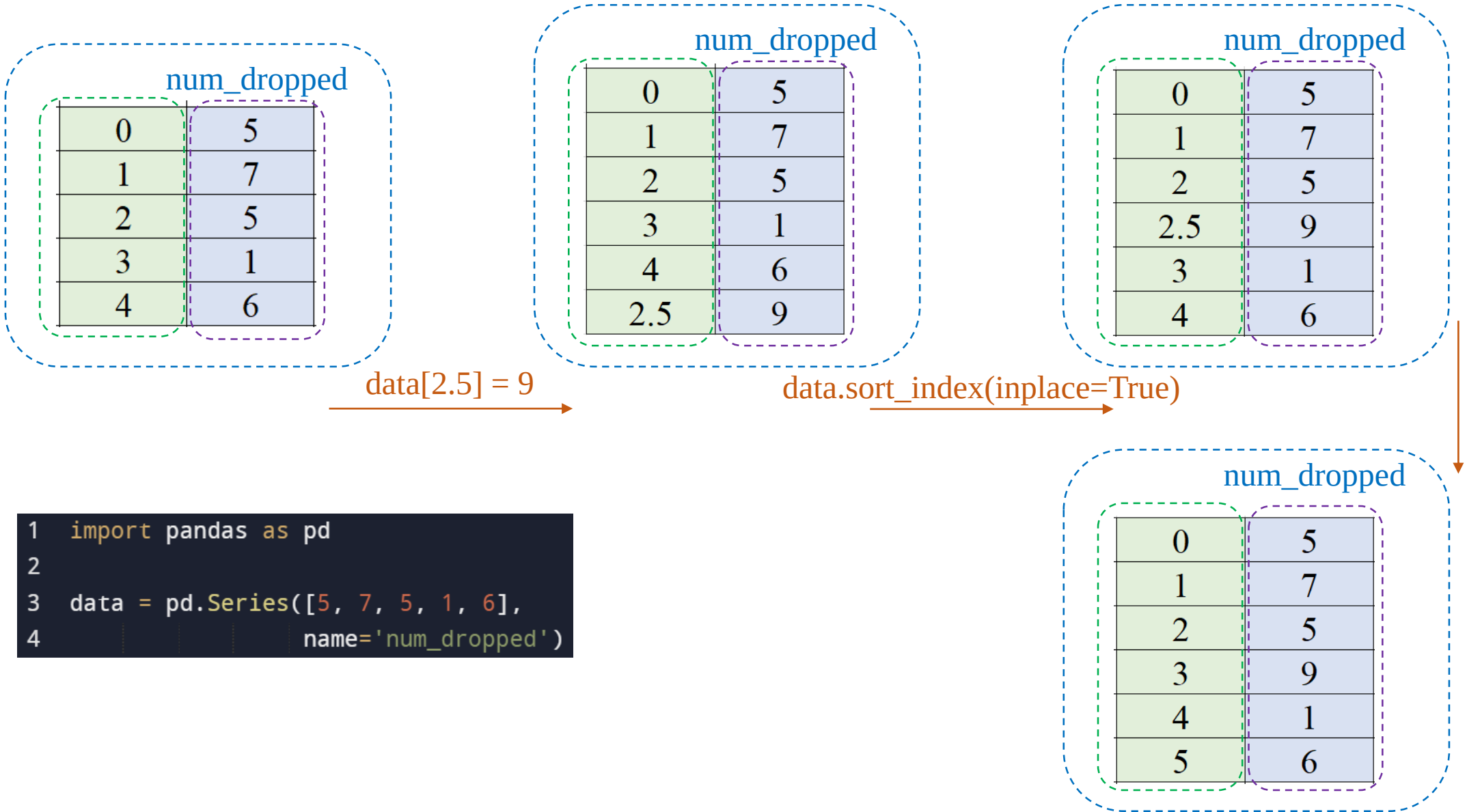
data['K'] = 'Kotlin '

data.sort_index(inplace=True)



Series in Pandas

Insert a row

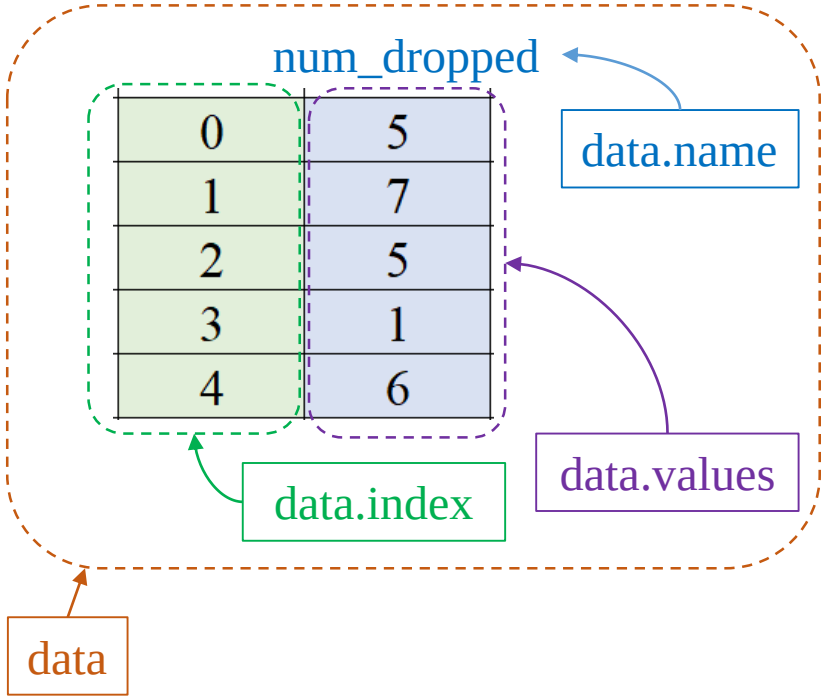



```python
import pandas as pd

data = pd.Series([5, 7, 5, 1, 6],
                 name='num_dropped')
```

Series in Pandas

❖ Common Pandas functions

```
1  import pandas as pd
2
3  data = pd.Series([5, 7, 5, 1, 6],
4                   name='num_dropped')
```



data.min() → 1

data.sum() → 24

data.max() → 7

data.mean() → 4.8

data.std() → 2.28

data.var() → 5.2

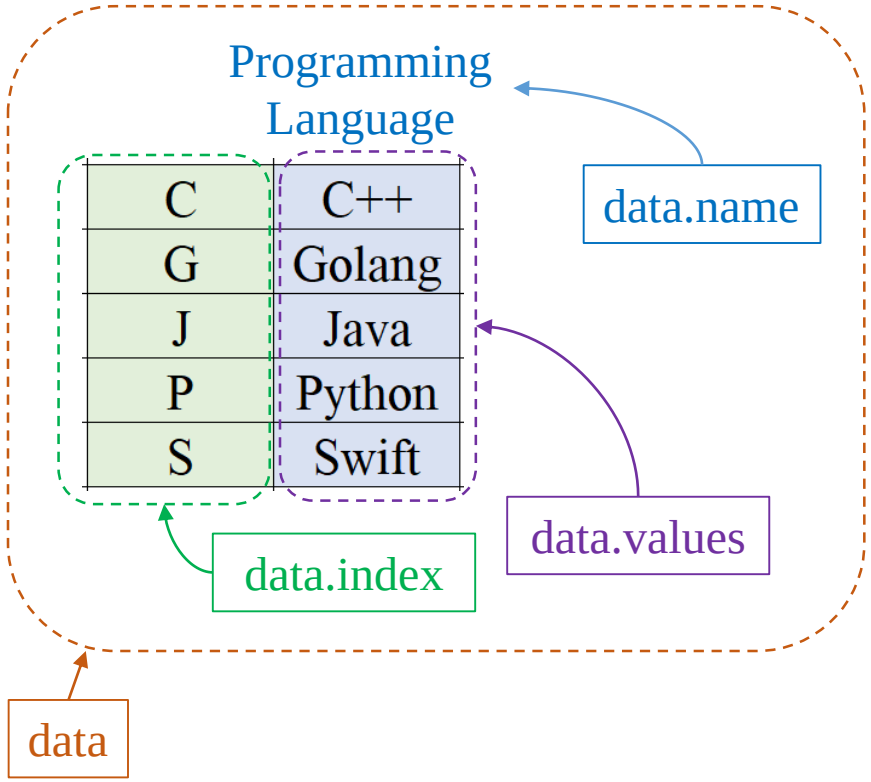
data.idxmax() → 1

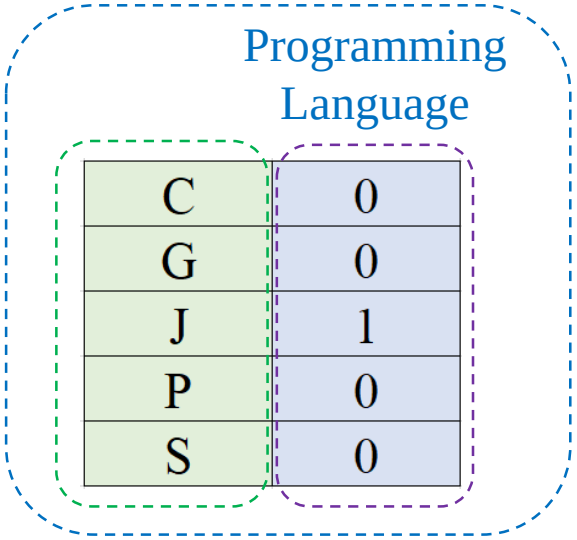
data.argmax() → 1

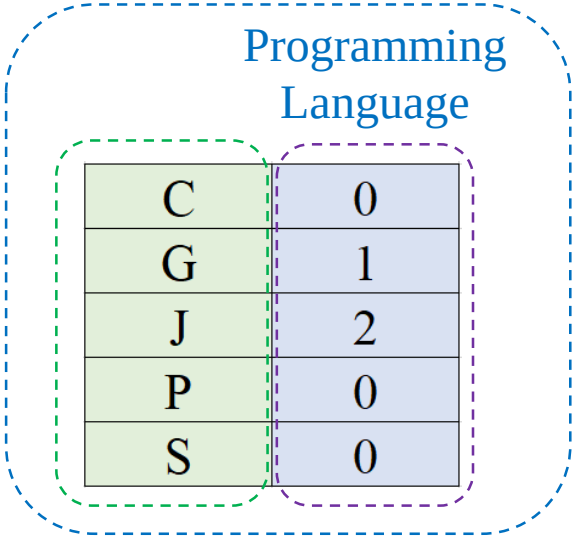
Series in Pandas

❖ Common Pandas functions

```
1  import pandas as pd
2
3  data = pd.Series(['C++', 'Golang', 'Java', 'Python', 'Swift'],
4                   index=list('CGJPS'),
5                   name='Programming Language')
```



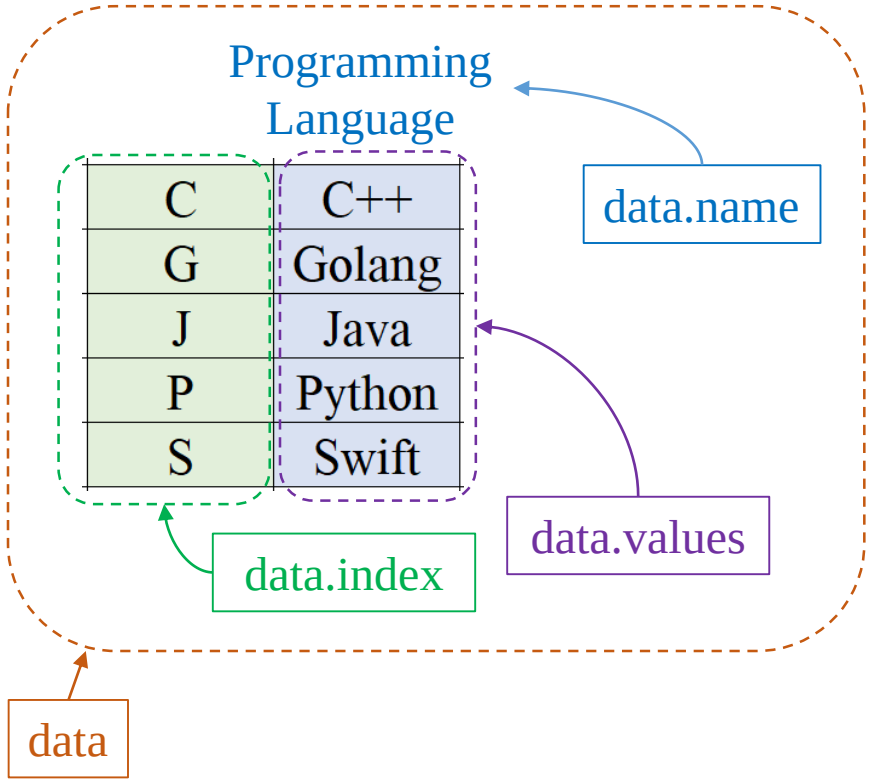


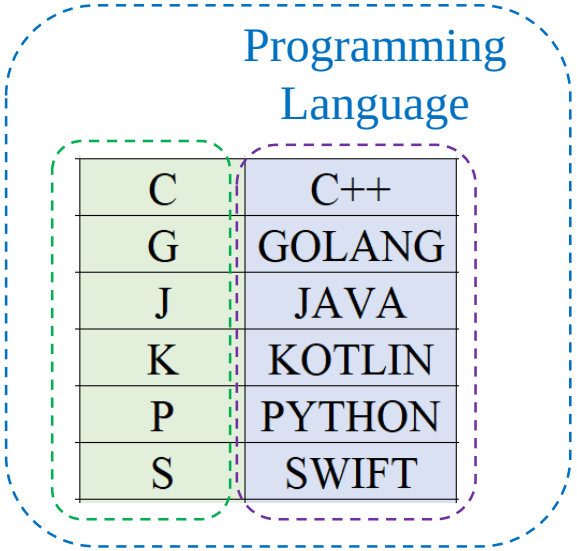


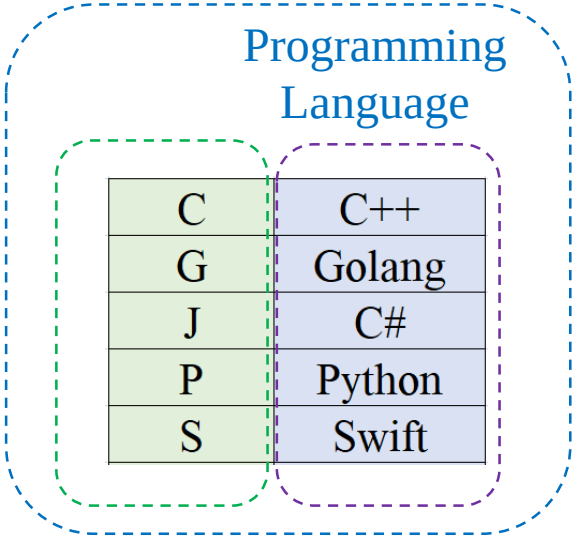
Series in Pandas

❖ Common Pandas functions

```
1  import pandas as pd
2
3  data = pd.Series(['C++', 'Golang', 'Java', 'Python', 'Swift'],
4                   index=list('CGJPS'),
5                   name='Programming Language')
```



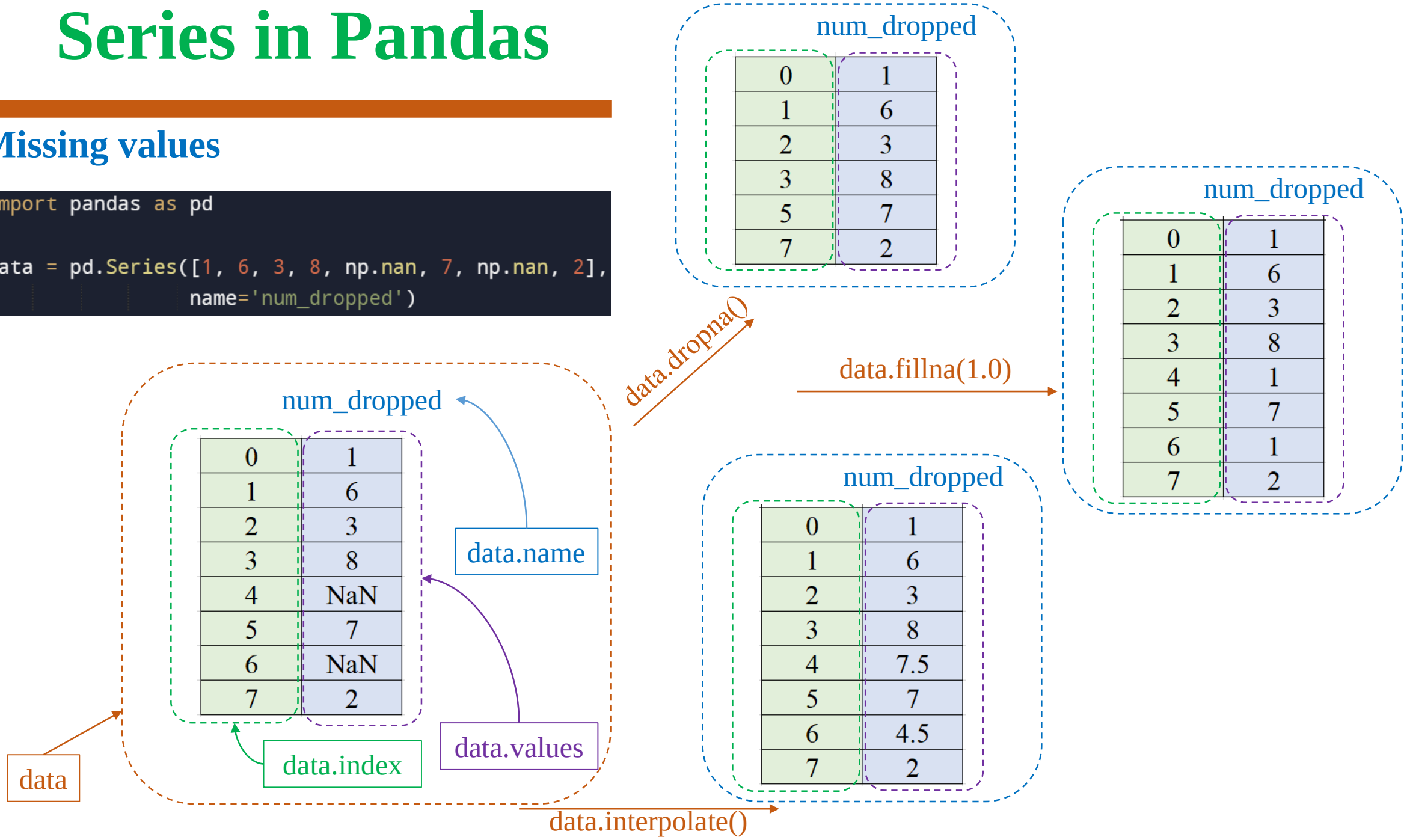




Series in Pandas

❖ Missing values

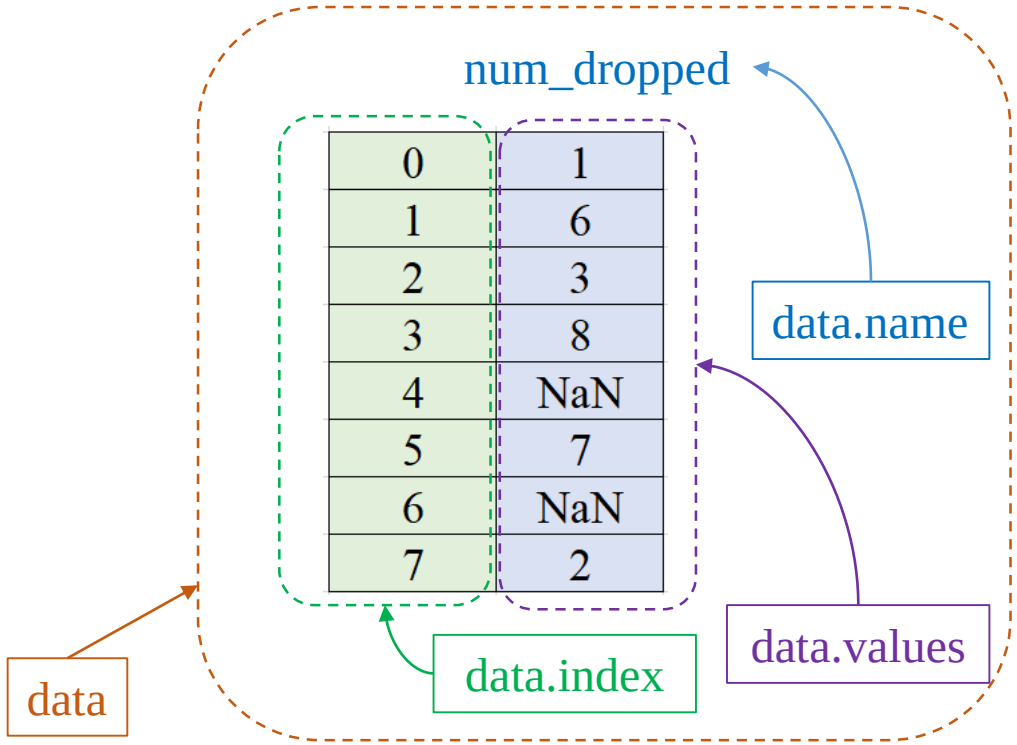
```
1  import pandas as pd
2
3  data = pd.Series([1, 6, 3, 8, np.nan, 7, np.nan, 2],
4                   name='num_dropped')
```



Series in Pandas

❖ Missing values

```
1  import pandas as pd
2
3  data = pd.Series([1, 6, 3, 8, np.nan, 7, np.nan, 2],
4                   name='num_dropped')
```



data.min() → 1

data.sum() → 27

data.max() → 8

data.mean() → 4.5

data.std() → 2.88

data.idxmax() → 3

data.argmax() → 3

Series in Pandas

Addition between two series

```python
import pandas as pd

center1 = pd.Series([12, 23, 31, 11, 9],
                    index=list(['C++', 'Golang',
                                'Java', 'Python',
                                'Swift']),
                    name='num_registered')

center2 = pd.Series([42, 18, 44, 49, 27],
                    index=list(['C++', 'Golang',
                                'Java', 'Python',
                                'Swift']),
                    name='num_registered')

total = center1 + center2
```

center1 →

num_registered

C++	12
Golang	23
Java	31
Python	11
Swift	9

center2 →

num_registered

C++	42
Golang	18
Java	44
Python	49
Swift	27

total →

num_registered

C++	54
Golang	41
Java	75
Python	60
Swift	36

Series in Pandas

Addition between two series

```python
import pandas as pd

center1 = pd.Series([12, 23, 31, 11, 9],
                    index=list(['C++', 'Golang',
                                'Java', 'Python',
                                'Swift']),
                    name='num_registered')

center2 = pd.Series([18, 44, 49, 27],
                    index=list(['Golang', 'Java',
                                'Python', 'Swift']),
                    name='num_registered')

total = center1 + center2
```

center1 →

num_registered

C++	12
Golang	23
Java	31
Python	11
Swift	9

center2 →

num_registered

Golang	18
Java	44
Python	49
Swift	27

total →

num_registered

C++	NaN
Golang	41
Java	75
Python	60
Swift	36

Series in Pandas

❖ Group values

```
1  import pandas as pd
2
3  data = pd.Series([4, 1, 0, 2, 2, 1],
4                   index=list(['C++', 'Golang',
5                               'Java', 'Golang',
6                               'Java', 'C++']),
7                   name='num_dropped')
```

data.groupby(level=0).sum()

num_dropped

C++	5
Golang	3
Java	2

data.groupby(data>3).sum()

num_dropped

FALSE	6
TRUE	4

num_dropped

C++	4
Golang	1
Java	0
Golang	2
Java	2
C++	1

data.name

data.values

data.index

data

data['C++']

C++	4
C++	1

data['C++'].sum() → 5

data['Golang']

Golang	1
Golang	2

data['Golang'].sum() → 3

data['Java']

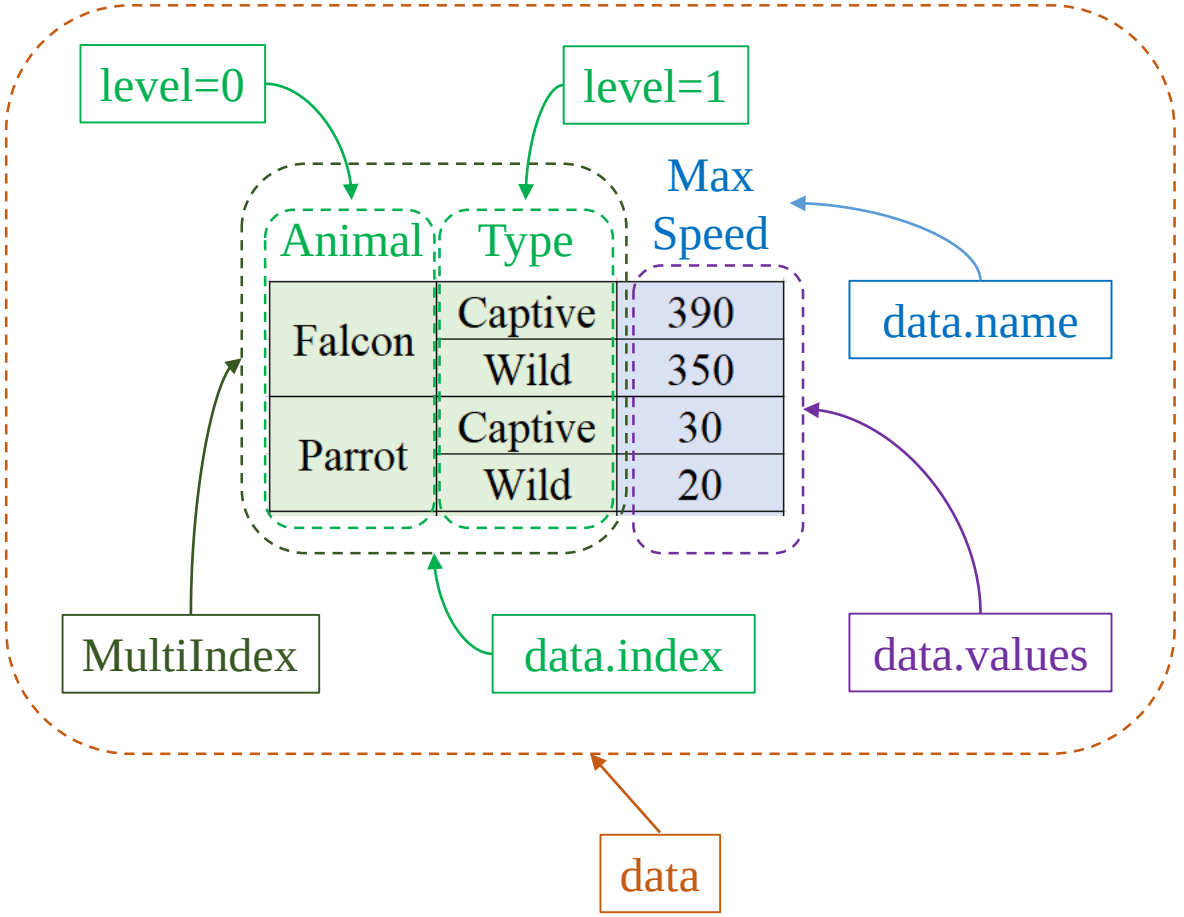
Java	0
Java	2

data['Java'].sum() → 2

Series in Pandas

❖ Multi-Index

```
import pandas as pd

arrays = [['Falcon', 'Falcon', 'Parrot', 'Parrot'],
          ['Captive', 'Wild', 'Captive', 'Wild']]
index = pd.MultiIndex.from_arrays(arrays,
                                  names=('Animal', 'Type'))
data = pd.Series([390., 350., 30., 20.],
                 index=index,
                 name="Max Speed")
```



data.groupby(level=0).sum()

Animal	Max Speed
Falcon	740
Parrot	50

data.groupby(level=1).sum()

Type	Max Speed
Captive	420
Wild	370

Data Frame

❖ Create a dataframe

```
1  import pandas as pd
2
3  df = pd.read_csv('advertising_simple.csv')
4  print(df)
5
6  df_t = df.T
7  print(df_t)
```

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

df.columns

df.index

df['Newspaper']
df.Newspaper

	0	1	2	3	4	5	6	7
TV	44	17	151	180	8	57	120	8
Radio	39	45	41	10	48	32	19	2
Newspaper	45	69	58	58	75	23	11	1
Sales	10	12	16	17	7	11	13	4

df_t.columns

df_t.index

Data Frame

❖ Sorting

```
1  import pandas as pd
2
3  df = pd.read_csv('advertising_simple.csv')
4  df.sort_values('Sales')
```

	TV	Radio	Newspaper	Sales
7	8	2	1	4
4	8	48	75	7
0	44	39	45	10
5	57	32	23	11
1	17	45	69	12
6	120	19	11	13
2	151	41	58	16
3	180	10	58	17

```
1  import pandas as pd
2
3  df = pd.read_csv('advertising_simple.csv')
4  df.sort_values(['Newspaper', 'Sales'])
```

	TV	Radio	Newspaper	Sales
7	8	2	1	4
6	120	19	11	13
5	57	32	23	11
0	44	39	45	10
2	151	41	58	16
3	180	10	58	17
1	17	45	69	12
4	8	48	75	7

AI VIETNAM
All-in-One Course

Data Frame

❖ Add a column

```
1  import pandas as pd
2
3  df = pd.read_csv('advertising_simple.csv')
4  df['ID'] = list(range(8))
5  df = df.set_index('ID')
```

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

df.columns

df.index

df['Newspaper']

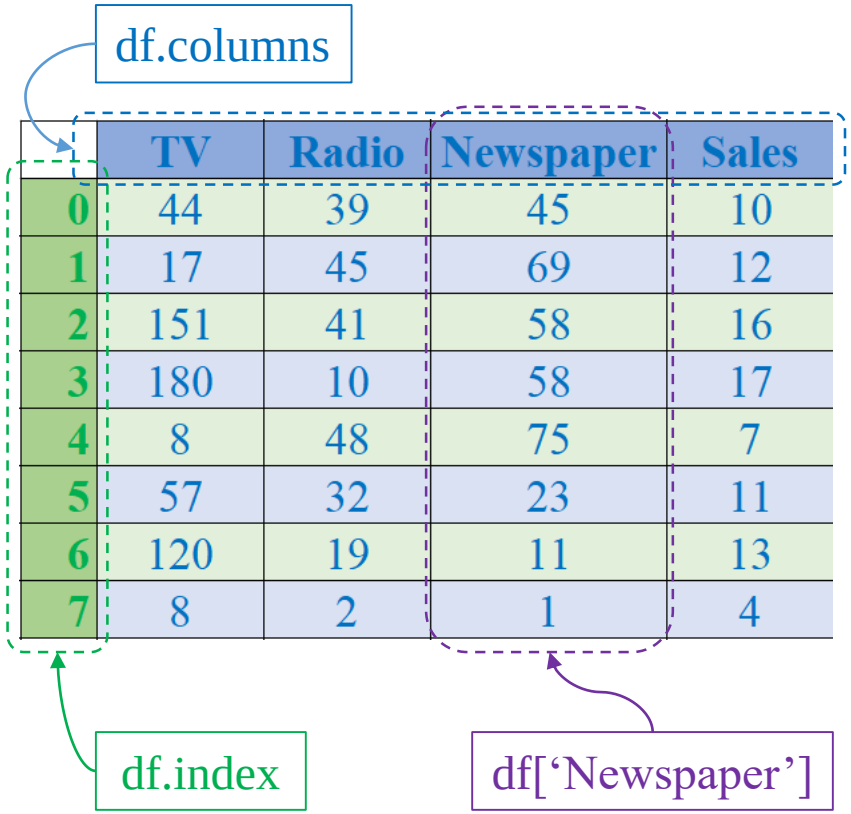
	TV	Radio	Newspaper	Sales	ID
0	44	39	45	10	2
1	17	45	69	12	3
2	151	41	58	16	4
3	180	10	58	17	5
4	8	48	75	7	6
5	57	32	23	11	7
6	120	19	11	13	8
7	8	2	1	4	9

ID	TV	Radio	Newspaper	Sales
2	44	39	45	10
3	17	45	69	12
4	151	41	58	16
5	180	10	58	17
6	8	48	75	7
7	57	32	23	11
8	120	19	11	13
9	8	2	1	4

Data Frame

❖ Delete a column

```
1  import pandas as pd
2
3  df = pd.read_csv('advertising_simple.csv')
```



	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

df.drop(1)

	TV	Radio	Newspaper	Sales
0	44	39	45	10
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

df.drop([1, 2, 3])

	TV	Radio	Newspaper	Sales
0	44	39	45	10
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

Data Frame

❖ Get values

```
1  import pandas as pd
2
3  df = pd.read_csv('advertising_simple.csv')
```

df.columns

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

df.index

df['Newspaper']

df.Radio
df['Radio']

	Radio
0	39
1	45
2	41
3	10
4	48
5	32
6	19
7	2

df[['Radio', 'Sales']]

	Radio	Sales
0	39	10
1	45	12
2	41	16
3	10	17
4	48	7
5	32	11
6	19	13
7	2	4

df[1:2]

	TV	Radio	Newspaper	Sales
1	17	45	69	12

df[4:]

	TV	Radio	Newspaper	Sales
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

df[1:4:2]

	TV	Radio	Newspaper	Sales
1	17	45	69	12
3	180	10	58	17

Data Frame

❖ Groupby

```
1  import pandas as pd
2
3  df = pd.read_csv('weatherHistory_simple.csv')
4  df.describe()
5
6  df.groupby('Precip Type').mean('Temperature (C)')
7  df['Fahrenheit'] = df['Temperature (C)']*9/5.0 + 32
```

	Temperature (C)
count	10
mean	6.25
std	7.63
min	-2.1
0.25	-0.92
0.5	6.35
0.75	13.52
max	14.7

Precip Type	Temperature (C)
rain	8.972222
snow	-2.038889

	Precip Type	Temperature (C)
0	rain	12.2
1	rain	13.8
2	rain	14.7
3	rain	13.8
4	rain	12.7
5	rain	0.5
6	snow	-0.4
7	snow	-1.1
8	snow	-1.6
9	snow	-2.1

	Precip Type	Temperature (C)	Fahrenheit
0	rain	12.2	53.96
1	rain	13.8	56.84
2	rain	14.7	58.46
3	rain	13.8	56.84
4	rain	12.7	54.86
5	rain	0.5	32.9
6	snow	-0.4	31.28
7	snow	-1.1	30.02
8	snow	-1.6	29.12
9	snow	-2.1	28.22

Data Frame

❖ loc and iloc

loc for indexing by labels

iloc for indexing by positional index

```
1  import pandas as pd
2
3  df = pd.read_csv('advertising_simple.csv')
4  df.set_index([['A', 'B', 'C', 'D',
5                 'F', 'G', 'H', 'I']],
6                inplace=True)
```

df.columns

	TV	Radio	Newspaper	Sales
A	44	39	45	10
B	17	45	69	12
C	151	41	58	16
D	180	10	58	17
E	8	48	75	7
F	57	32	23	11
G	120	19	11	13
H	8	2	1	4

df.loc['C', 'Radio']

df.iloc[2, 1]

df.index

df.loc['F':'H', 'Radio':'Sales']

df.loc['B', :]

	TV	Radio	Newspaper	Sales
A	44	39	45	10
B	17	45	69	12
C	151	41	58	16
D	180	10	58	17
E	8	48	75	7
F	57	32	23	11
G	120	19	11	13
H	8	2	1	4

df.loc[['E', 'G'], :]

Pandas: Select values

```
1  import pandas as pd
2
3  df = pd.read_csv('advertising_simple.csv')
```

df.columns

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

df.index

```
# select * from df where df.Newspaper<30
df.loc[df.Newspaper<30]
```

	TV	Radio	Newspaper	Sales
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

```
# select * from df where Sales>10
df.loc[df.Sales>10]
```

	TV	Radio	Newspaper	Sales
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
5	57	32	23	11
6	120	19	11	13

	TV	Radio	Newspaper	Sales
5	57	32	23	11
6	120	19	11	13

```
# select * from df where Sales>10 and Newspaper<30
df.loc[(df.Sales>10) & (df.Newspaper<30)]
```

```
# sql: select Radio from df
df['Radio']
```

	Radio
0	39
1	45
2	41
3	10
4	48
5	32
6	19
7	2

Data Frame

❖ Concatenate data

```
1  import pandas as pd
2
3  df1 = pd.read_csv('advertising_p1.csv')
4  df2 = pd.read_csv('advertising_p2.csv')
```

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17

df1

	TV	Radio	Newspaper	Sales
0	8	48	75	7
1	57	32	23	11
2	120	19	11	13
3	8	2	1	4

df2

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

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
0	8	48	75	7
1	57	32	23	11
2	120	19	11	13
3	8	2	1	4

df3

df4 = df3.reset_index(drop=True)

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

df4

Data Frame

❖ Concatenate data

```
1  import pandas as pd
2
3  df1 = pd.read_csv('advertising_p1.csv')
4  df2 = pd.read_csv('advertising_p2.csv')
```

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

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17

df1

	TV	Radio	Newspaper	Sales
0	8	48	75	7
1	57	32	23	11
2	120	19	11	13
3	8	2	1	4

df2

	TV	Radio	Newspaper	Sales
0	44	39	45	10
1	17	45	69	12
2	151	41	58	16
3	180	10	58	17
4	8	48	75	7
5	57	32	23	11
6	120	19	11	13
7	8	2	1	4

df3

Data Frame

❖ 1:1 relationship joins

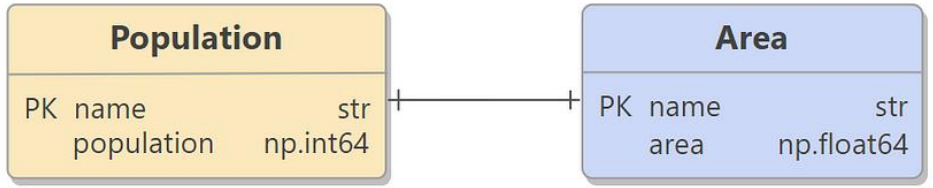
df

	name	population
10	Oslo	698660
20	Vienna	1911191

df1

	name	area
30	Vienna	414.8
40	Tokyo	2194.1

If the column you want to merge on is not in the index




```
df.merge(df1, on='name')
```

	name	population	area
0	Vienna	1911191	414.8

inner join

```
..., how='left')
```

	name	population	area
0	Oslo	698660.0	NaN
1	Vienna	1911191	414.8

left outer join

```
..., how='outer')
```

	name	population	area
0	Oslo	698660.0	NaN
1	Vienna	1911191	414.8
2	Tokyo	NaN	2194.1

full outer join

```
..., how='right')
```

	name	population	area
0	Vienna	1911191	414.8
1	Tokyo	NaN	2194.1

right outer join

https://betterprogramming.pub/pandas-illustrated-the-definitive-visual-guide-to-pandas-c31fa921a43

Data Frame

❖ 1:1 relationship joins

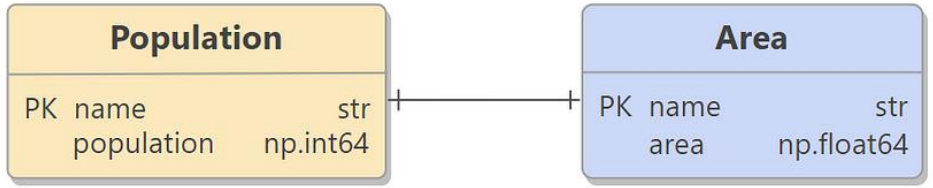
df

	population
Oslo	698660
Vienna	1911191

df1

	area
Vienna	414.8
Tokyo	2194.1

If the column is already in the index




```
df.join(df1, how='inner')
```

	population	area
Vienna	1911191	414.8

inner join

```
df.join(df1)
```

	population	area
Oslo	698660.0	NaN
Vienna	1911191	414.8

left outer join

```
df.join(df1, how='outer')
```

	population	area
Oslo	698660.0	NaN
Tokyo	NaN	2194.1
Vienna	1911191	414.8

full outer join

```
df.join(df1, how='right')
```

	population	area
Vienna	1911191	414.8
Tokyo	NaN	2194.1

right outer join

https://betterprogramming.pub/pandas-illustrated-the-definitive-visual-guide-to-pandas-c31fa921a43

Data Frame

❖ 1:n relationship joins

cities

	city	state_code
0	San Francisco	CA
1	Miami	FL
2	Washington	DC
3	Los Angeles	CA

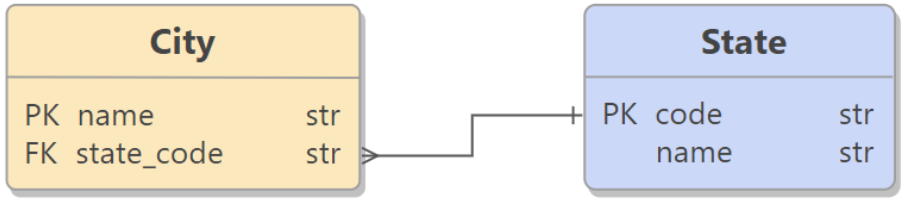
states

	code	state
0	CA	California
1	FL	Florida
2	TX	Texas

If the column you want to merge on is not in the index

```
df = cities.merge(states, left_on='state_code', right_on='code')
```

	city	state_code	code	state
0	Los Angeles	CA	CA	Florida
1	San Francisco	CA	CA	California
2	Miami	FL	FL	California




```
df.filter(['city', 'state'])
```

	city	state
0	San Francisco	California
1	Los Angeles	Florida
2	Miami	California

https://betterprogramming.pub/pandas-illustrated-the-definitive-visual-guide-to-pandas-c31fa921a43

Data Frame

❖ 1:n relationship joins

cities

	city	state_code
0	San Francisco	CA
1	Miami	FL
2	Washington	DC
3	Los Angeles	CA

states

	state
CA	California
FL	Florida
TX	Texas

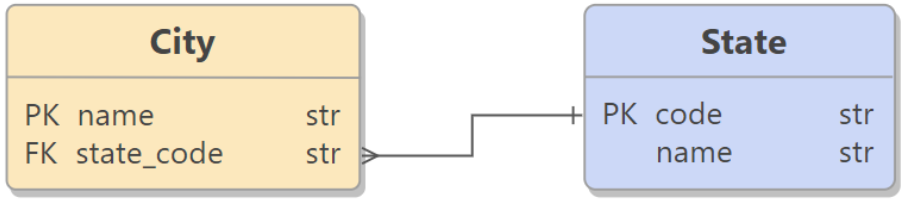
```
df = cities.join(states, on='state_code')
```

	city	state_code	state
0	San Francisco	CA	California
1	Miami	FL	Florida
2	Washington	DC	NaN
3	Los Angeles	CA	California

```
df.drop('state_code', axis=1)
```

	city	state
0	San Francisco	California
1	Miami	Florida
2	Washington	NaN
3	Los Angeles	California

If the column is already in the index



https://betterprogramming.pub/pandas-illustrated-the-definitive-visual-guide-to-pandas-c31fa921a43

Data Frame

Convert categories to numbers

```
1  import pandas as pd
2
3  df = pd.read_csv('weatherHistory_simple.csv')
4  df['Precip Type'] = pd.Categorical(df['Precip Type']).codes
```

	Precip Type	Temperature (C)
0	rain	12.2
1	rain	13.8
2	rain	14.7
3	rain	13.8
4	rain	12.7
5	rain	0.5
6	snow	-0.4
7	snow	-1.1
8	snow	-1.6
9	snow	-2.1

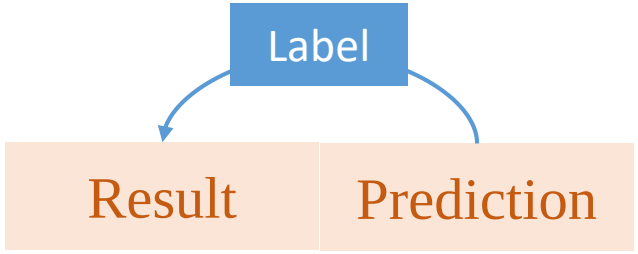
	Precip Type	Temperature (C)
0	0	12.2
1	0	13.8
2	0	14.7
3	0	13.8
4	0	12.7
5	0	0.5
6	1	-0.4
7	1	-1.1
8	1	-1.6
9	1	-2.1

Metrics

Confusion Matrix

		Actual Value: Positive	Actual Value: Negative
Predicted Value	Positive	TP (True Positive)	FP (False Positive)
Predicted Value	Negative	FN (False Negative)	TN (True Negative)

- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative
- False Positive (FP) : Observation is negative, but is predicted positive.

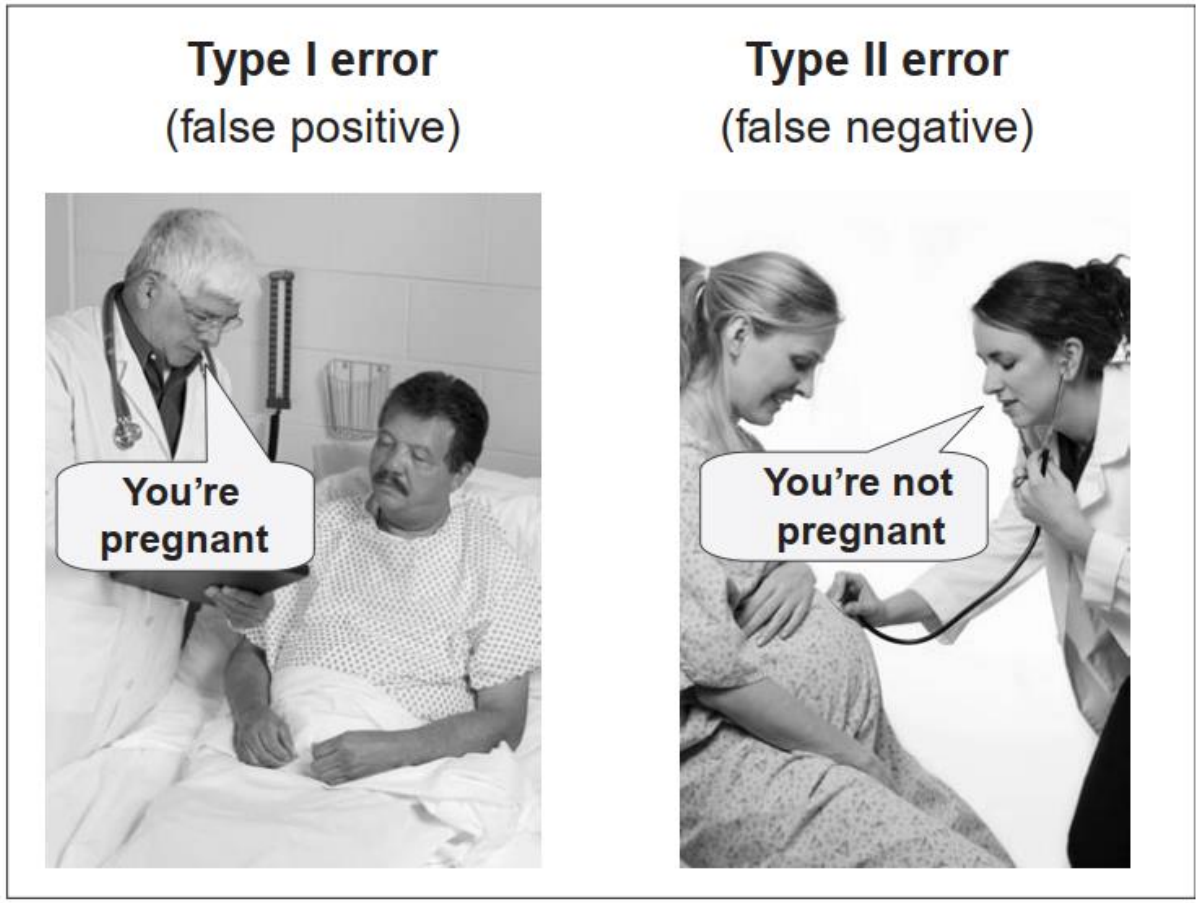


True Positive (TP): A correct detection.

False Positive (FP): A wrong detection.

False Negative (FN): A ground truth not detected

True Negative (TN): Does not apply.



The Essential Guide to Effect Sizes

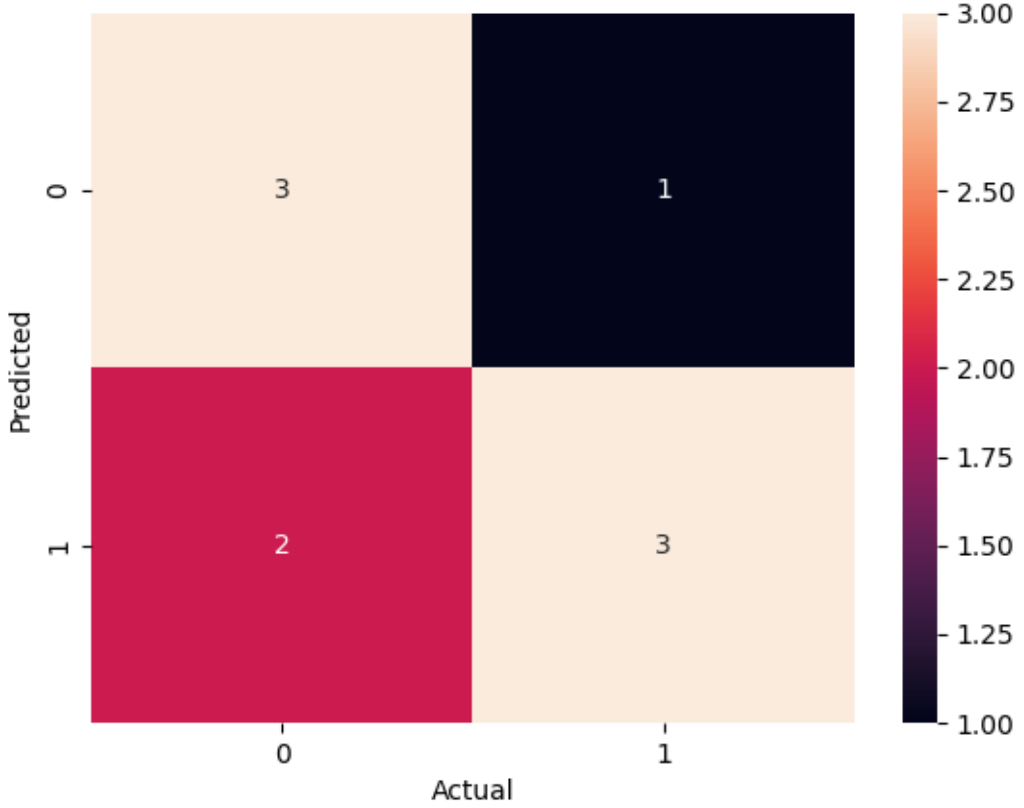
https://www.kdnuggets.com/2020/04/performance-evaluation-metrics-classification.html

Data Frame

❖ Visualize data

```
1  import pandas as pd
2
3  data = {'Actual Value':    [1, 0, 0, 1, 0, 1, 0, 0, 1],
4          'Predicted Value': [0, 1, 0, 1, 0, 1, 1, 0, 1]}
5  df = pd.DataFrame(data)
```

	Actual Value	Predicted Value
0	1	0
1	0	1
2	0	0
3	1	1
4	0	0
5	1	1
6	0	1
7	0	0
8	1	1




```
1  import pandas as pd
2  import seaborn as sn
3  import matplotlib.pyplot as plt
4
5  confusion_matrix = pd.crosstab(df['Predicted Value'],
6                                 df['Actual Value'],
7                                 rownames=['Predicted'],
8                                 colnames=['Actual'])
9
10 sn.heatmap(confusion_matrix, annot=True)
11 plt.show()
```

Data Frame

❖ Case study 1

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3  import seaborn as sns
4
5  data = pd.read_csv('iris.csv')
```

data.head()

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5	3.6	1.4	0.2	Setosa

data.describe()

	sepal_length	sepal_width	petal_length	petal_width
count	150	150	150	150
mean	5.843	3.057	3.758	1.199
std	0.828	0.435	1.765	0.762
min	4.3	2	1	0.1
0.25	5.1	2.8	1.6	0.3
0.5	5.8	3	4.35	1.3
0.75	6.4	3.3	5.1	1.8
max	7.9	4.4	6.9	2.5

data.info()

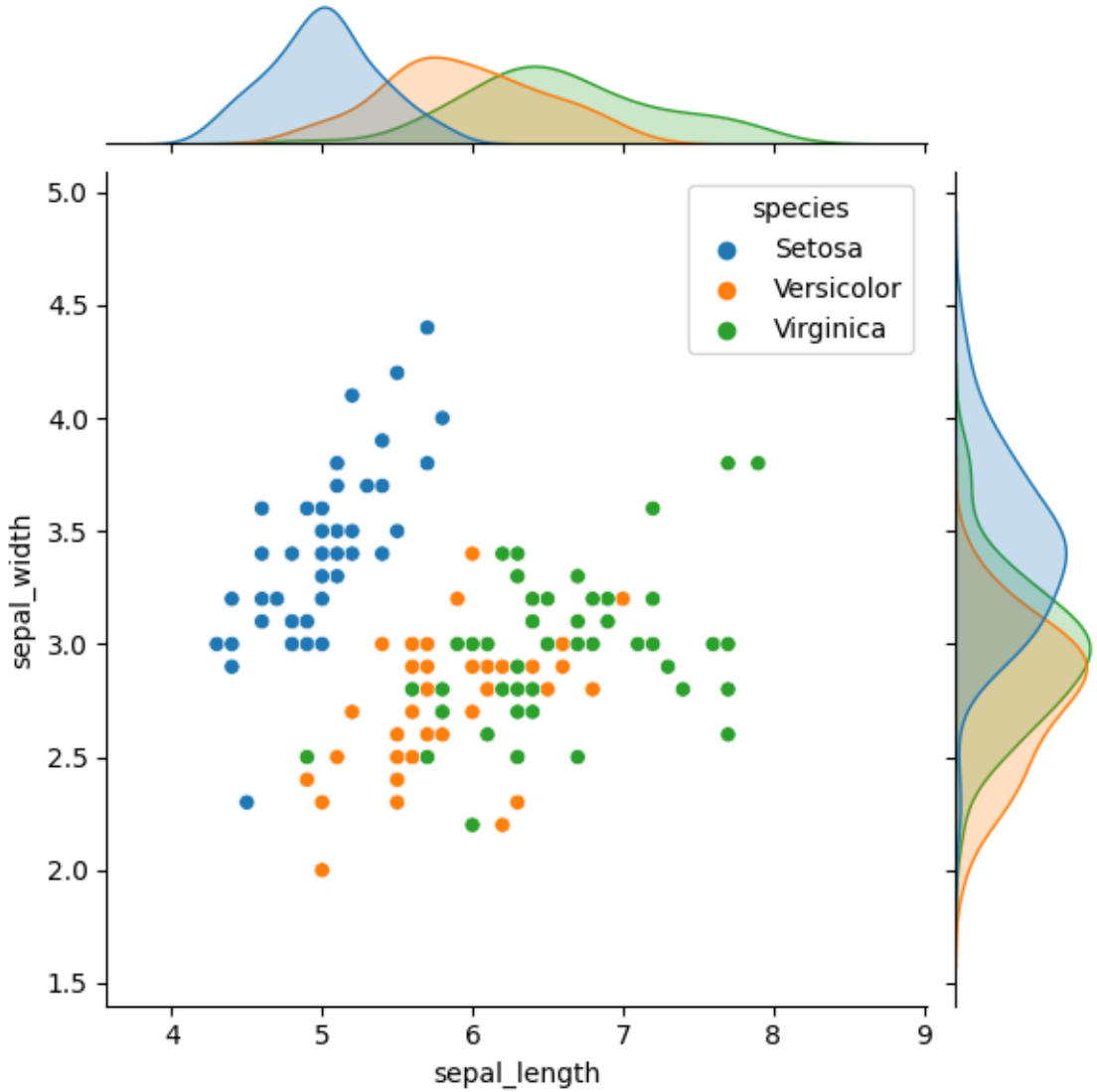
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   Species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

data["species"].value_counts()

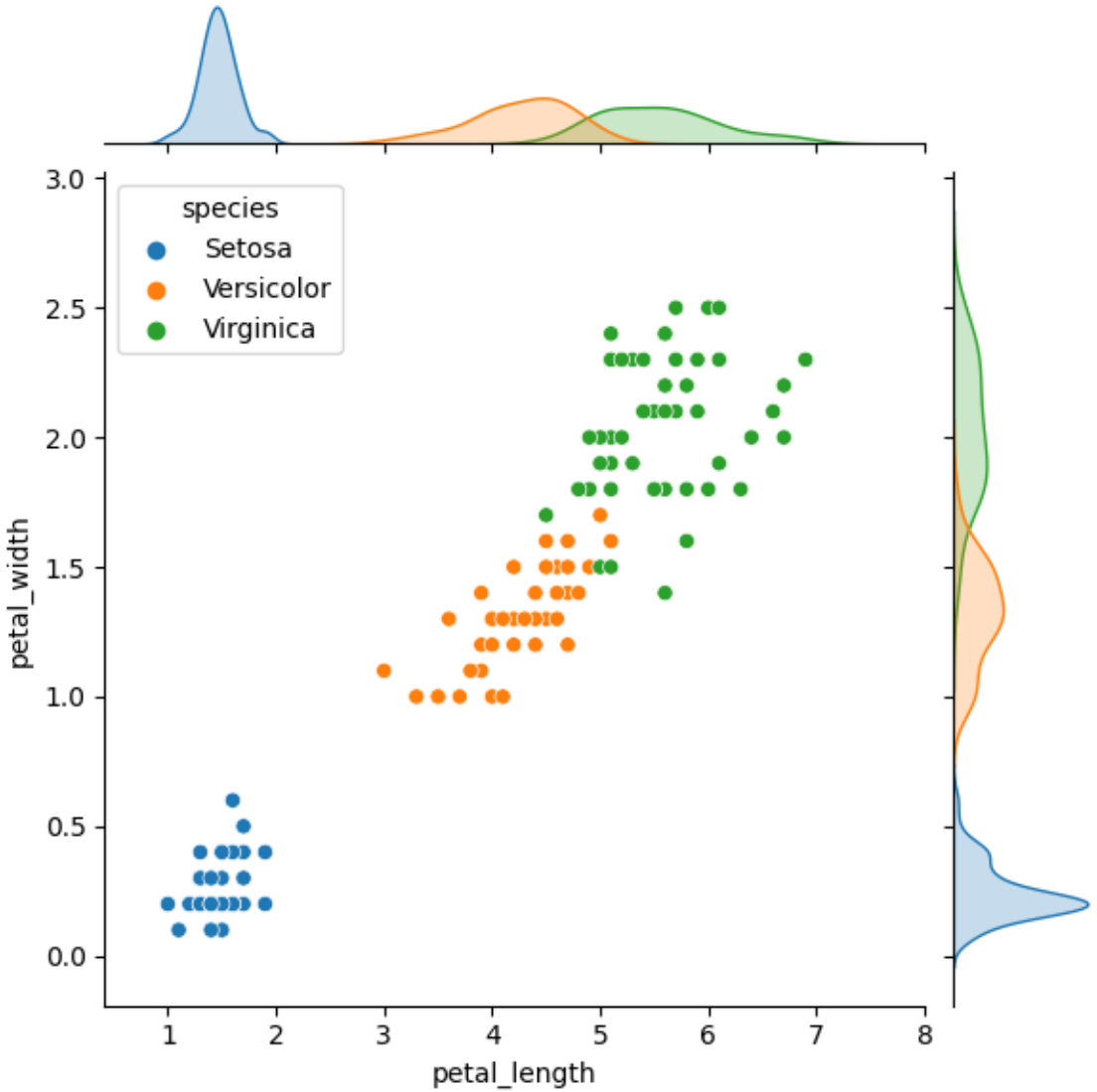
```
Setosa        50
Versicolor    50
Virginica     50
Name: species, dtype: int64
```

Case study 1

```
sns.jointplot(x="sepal_length", y="sepal_width",
              data=data, hue="species")
plt.show()
```

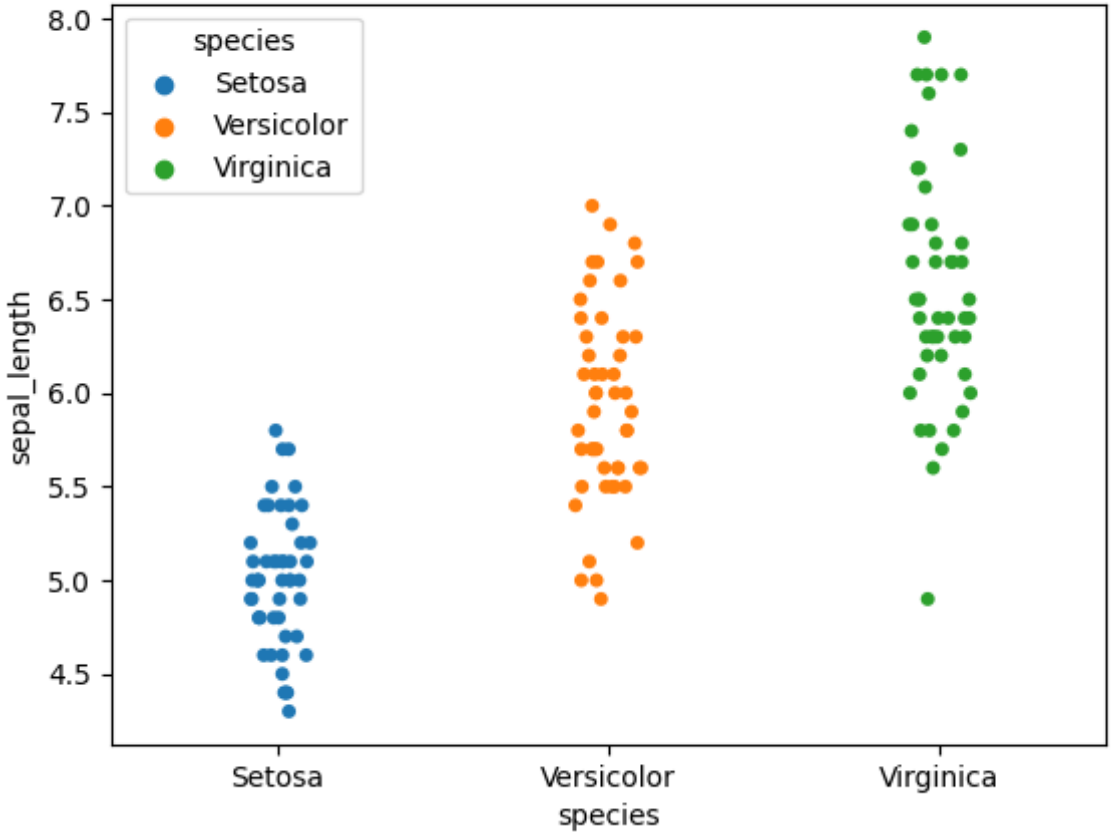



```
sns.jointplot(x="petal_length", y="sepal_width",
              data=data, hue="species")
plt.show()
```

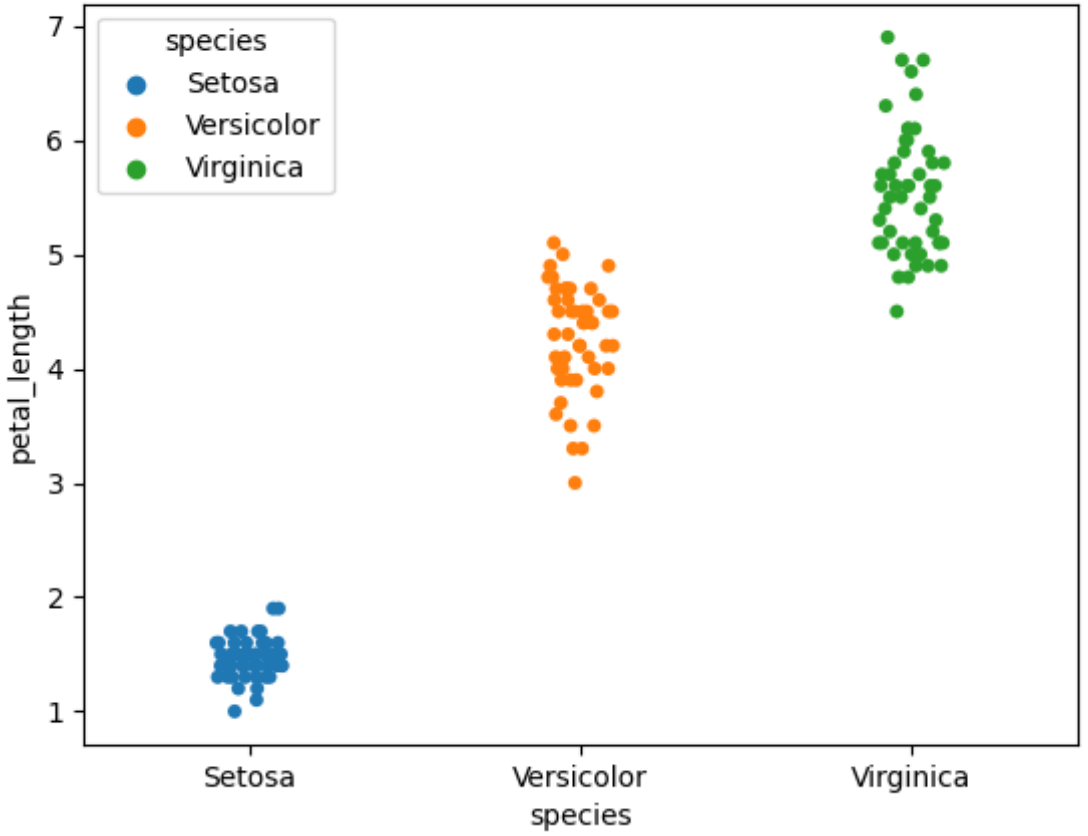


Case study 1

```
sns.stripplot(y='sepal_length', x='species',
              data=data, hue="species")
plt.show()
```

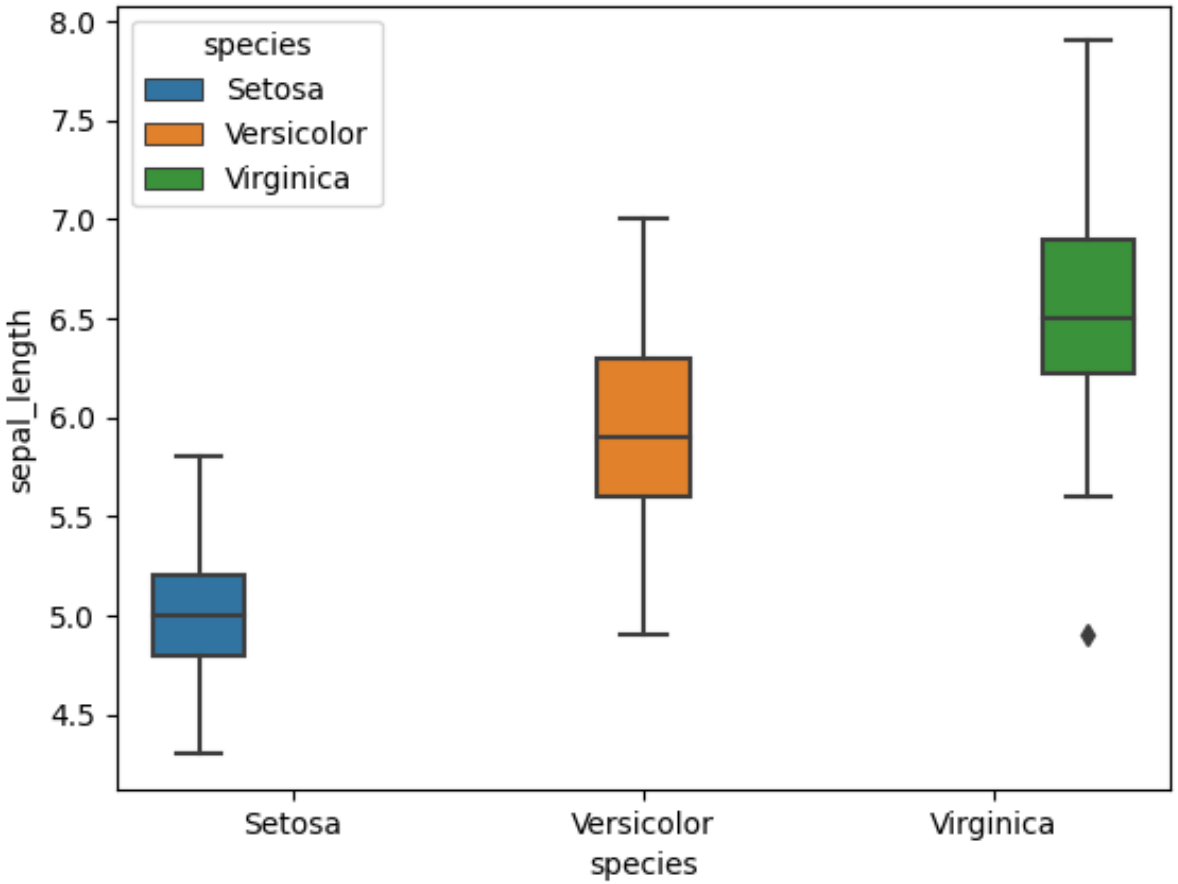


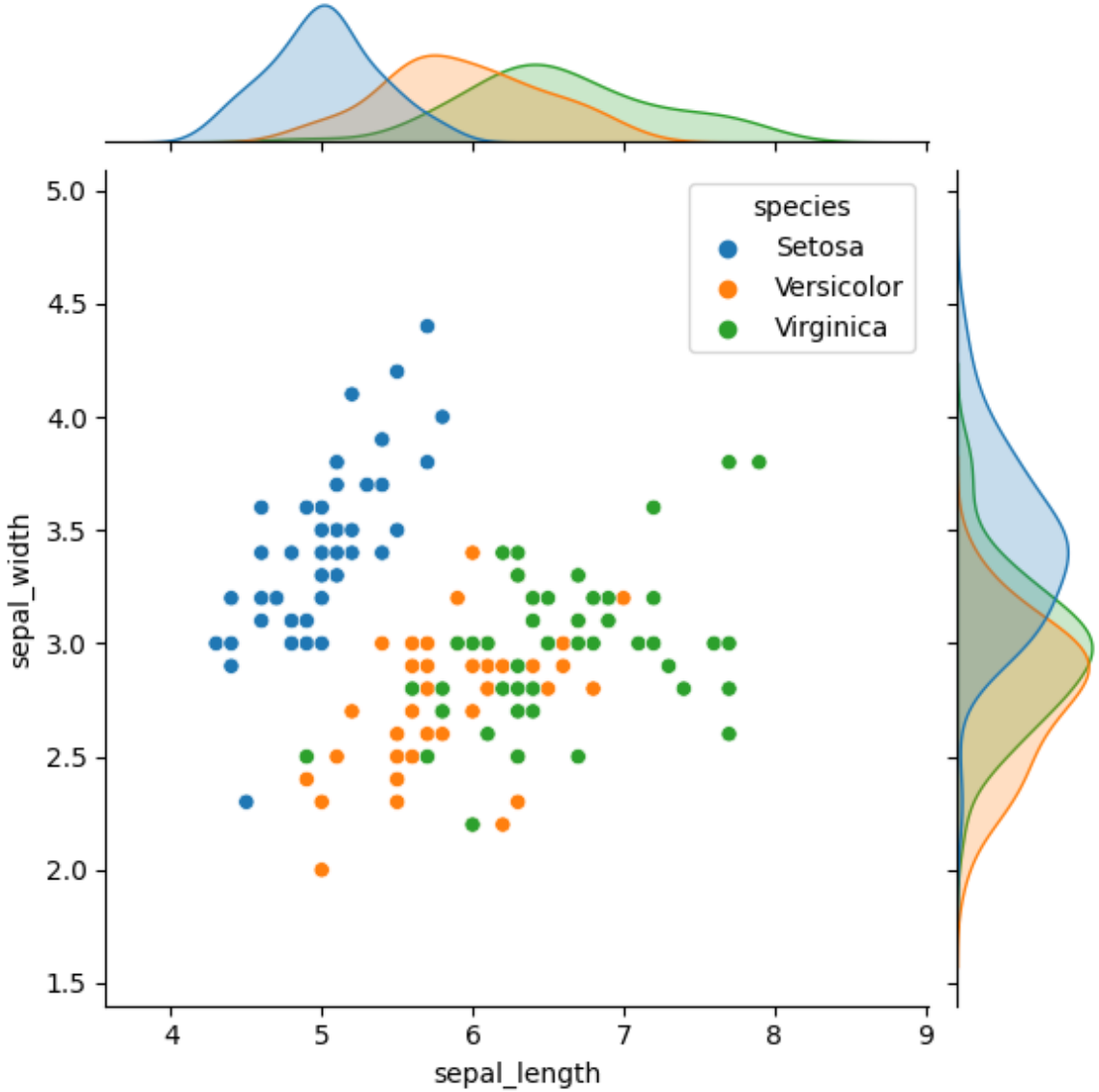

```
sns.stripplot(y='petal_length', x='species',
              data=data, hue="species")
plt.show()
```



Case study 1

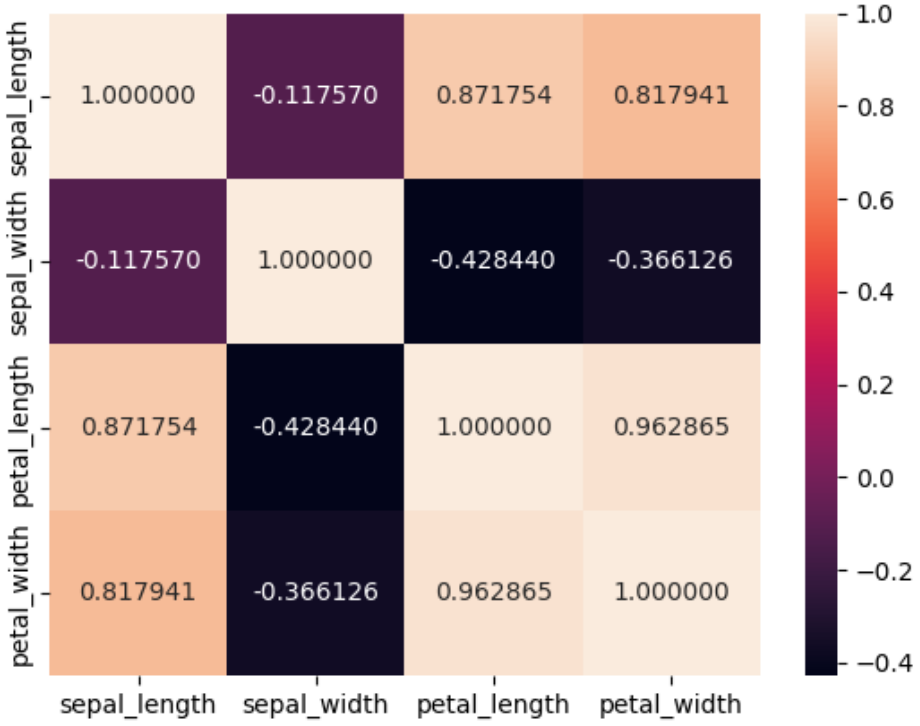
```
sns.boxplot(x="species", y="sepal_length",
            data=data, hue="species")
plt.show()
```

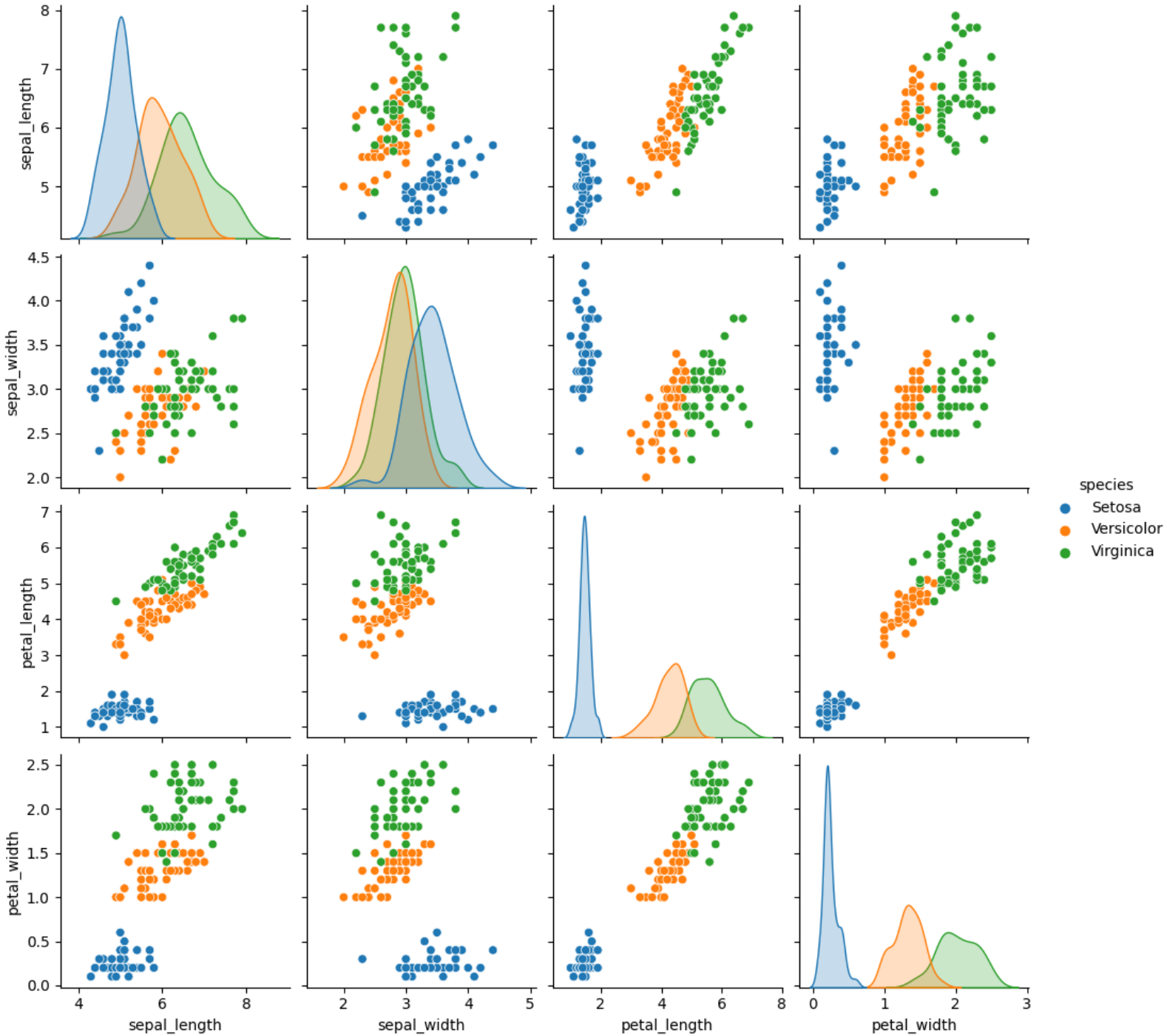




Case study 1

```
sns.pairplot(data=data,
                hue="species")
sns.heatmap(data.iloc[:,0:4].corr(),
                annot=True, fmt="f")
plt.show()
```

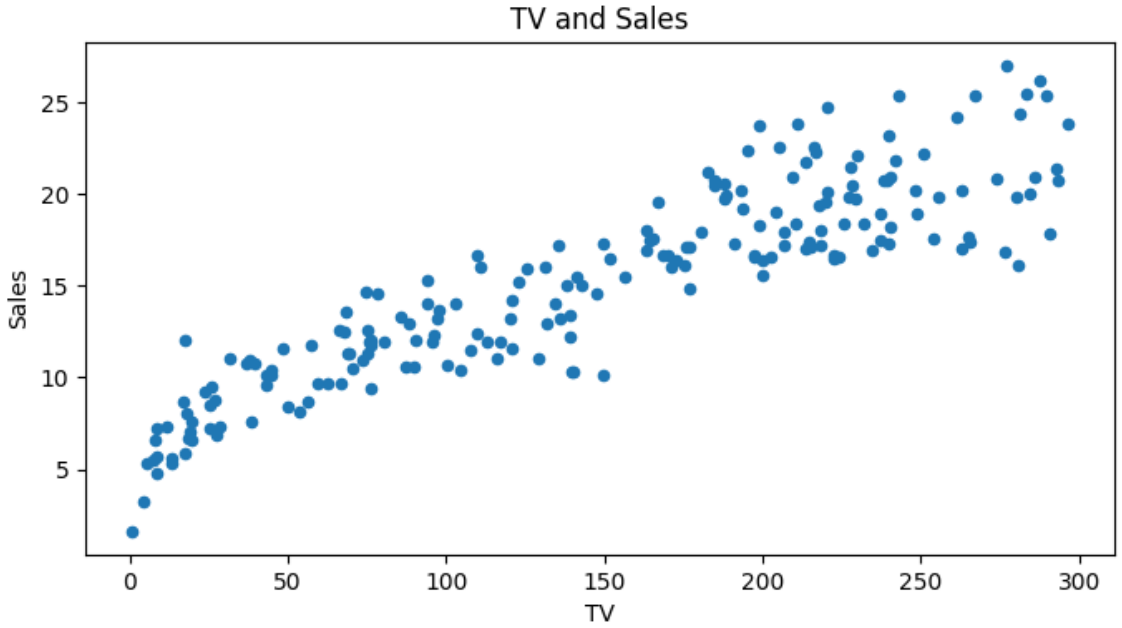


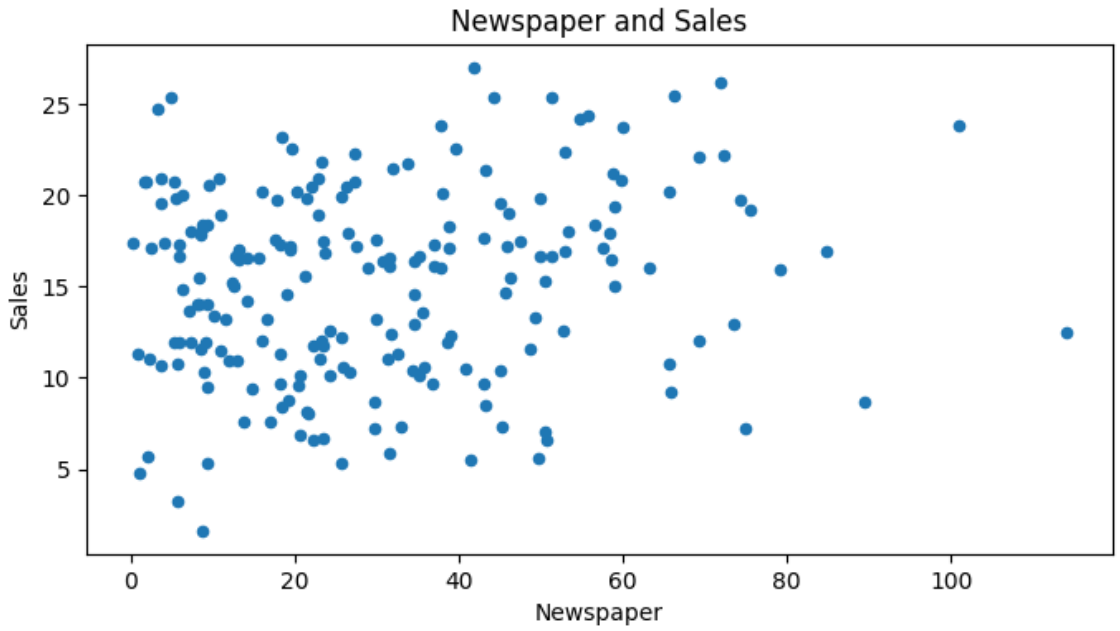


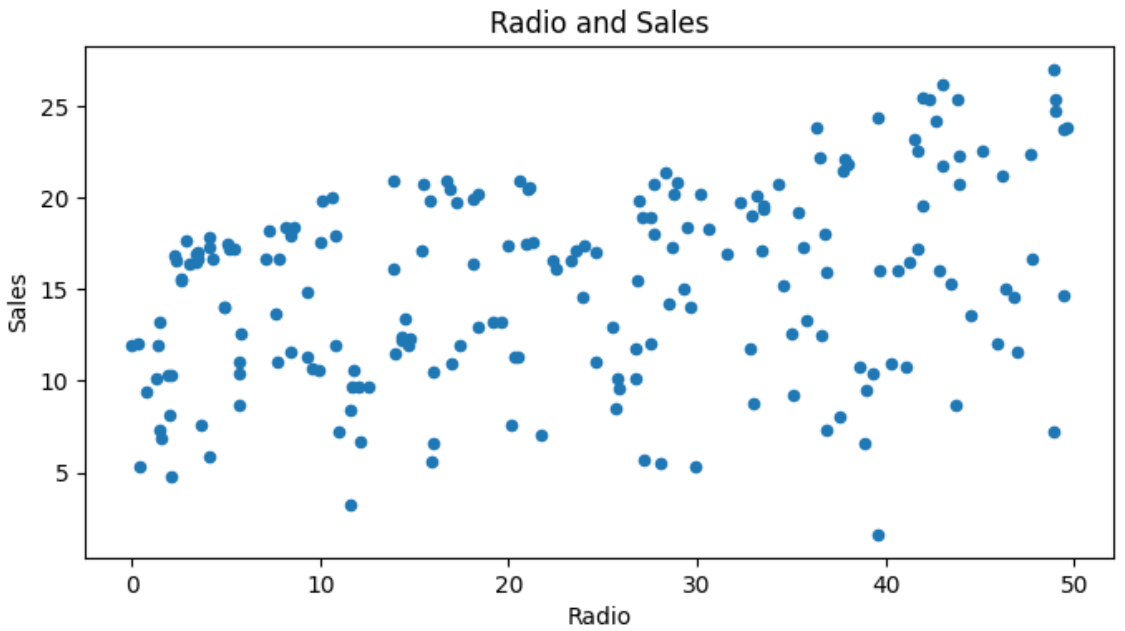
Case study 2

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9
...	...	...	...	...

```
1  import pandas as pd
2
3  df = pd.read_csv('advertising.csv')
4
5  df.plot(x='TV', y='Sales', kind='scatter')
6  df.plot(x='Radio', y='Sales', kind='scatter')
7  df.plot(x='Newspaper', y='Sales', kind='scatter')
```



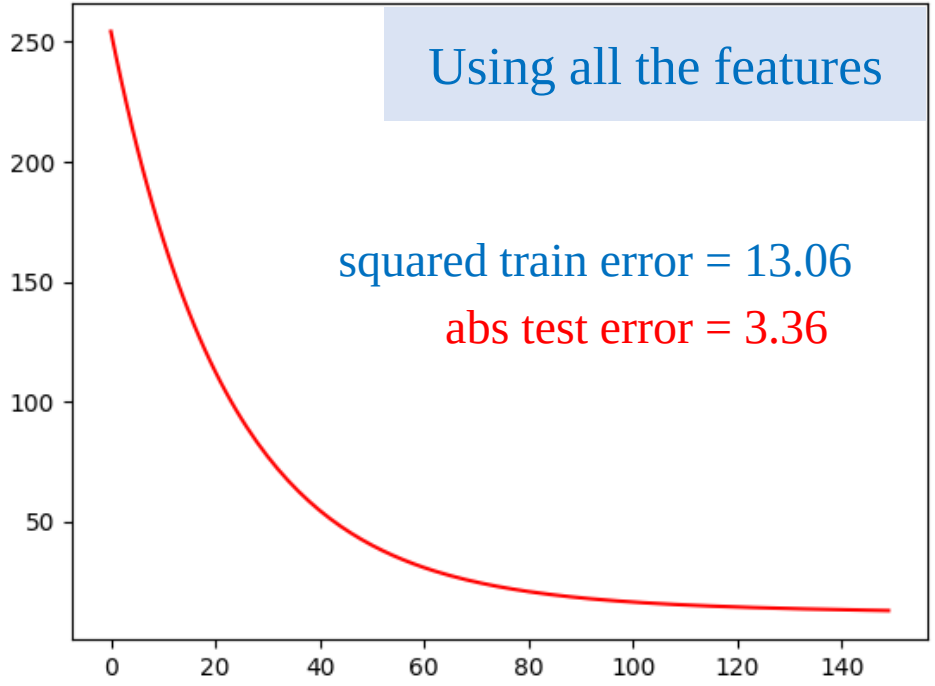




Case study 2

```python
import numpy as np

epochs = 150
lr = 0.01

# khởi tạo giá trị tham số
thetas = np.random.randn(F, 1)

for i in range(epochs):
    # tính output
    output = x_train.dot(thetas)

    # tính loss
    loss = (output - y_train)**2

    # tính đạo hàm cho loss
    loss_grd = 2*(output - y_train)/N

    # tính đạo hàm cho các tham số
    gradients = x_train.T.dot(loss_grd)

    # cập nhật tham số
    thetas = thetas - lr*gradients

# compute abs test error
output = x_test.dot(thetas)
test_loss = np.abs(output - y_test)
```

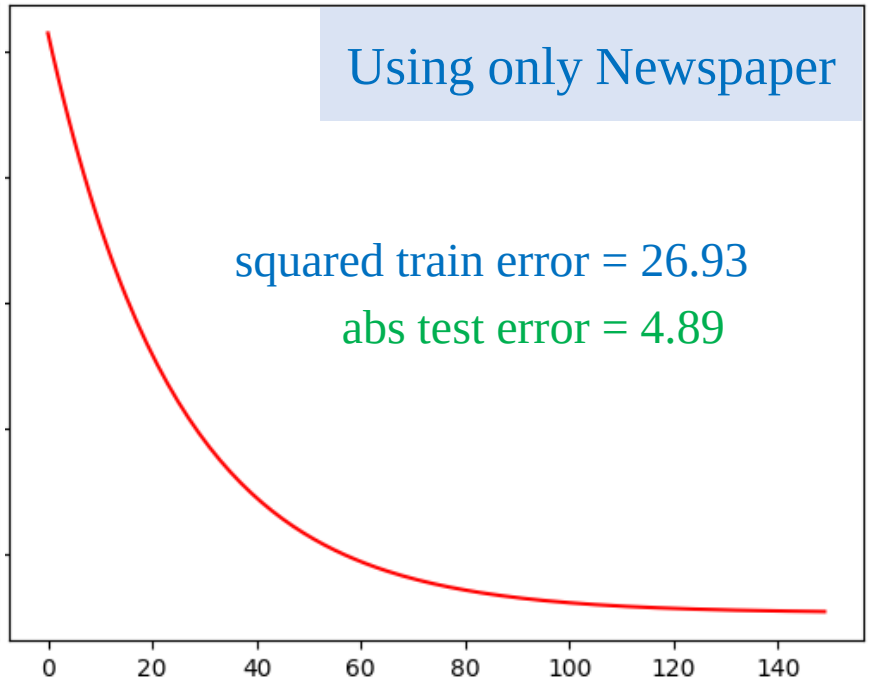
Using all the features



squared train error = 13.06

abs test error = 3.36

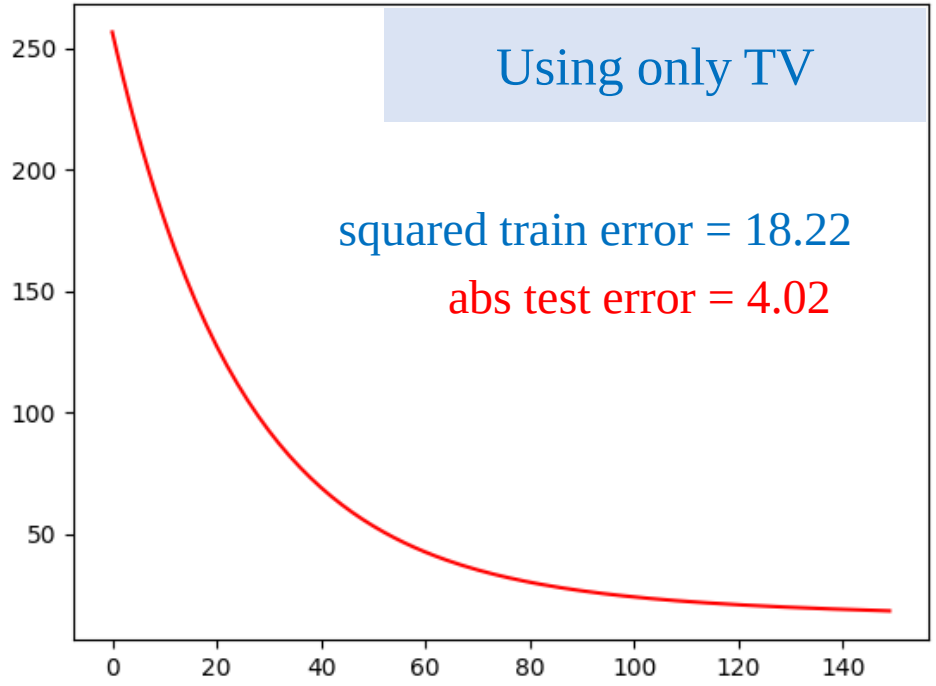
Using only Newspaper



squared train error = 26.93

abs test error = 4.89

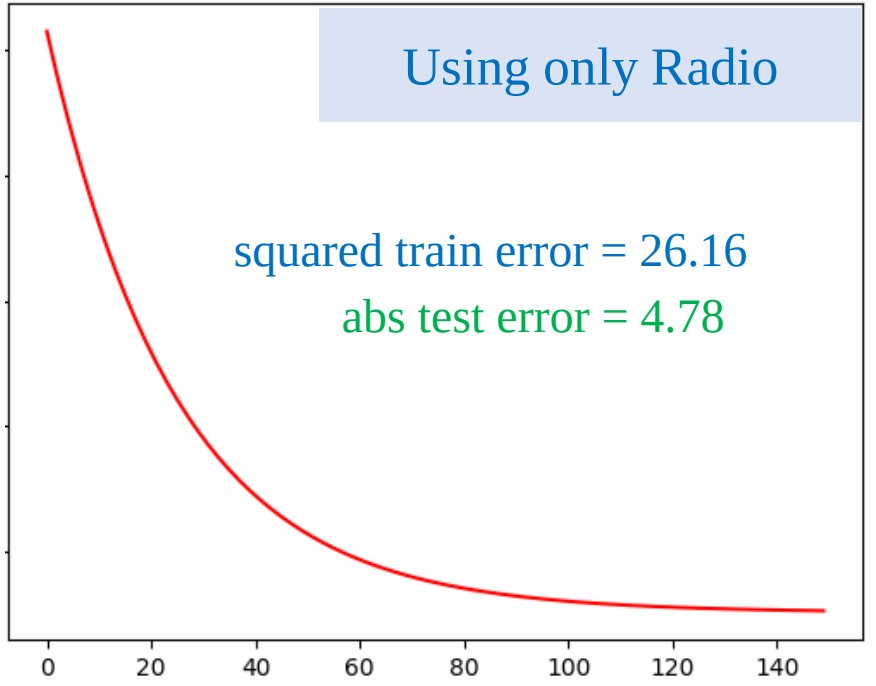
Using only TV



squared train error = 18.22

abs test error = 4.02

Using only Radio



squared train error = 26.16

abs test error = 4.78

Data Frame

❖ Case study 2

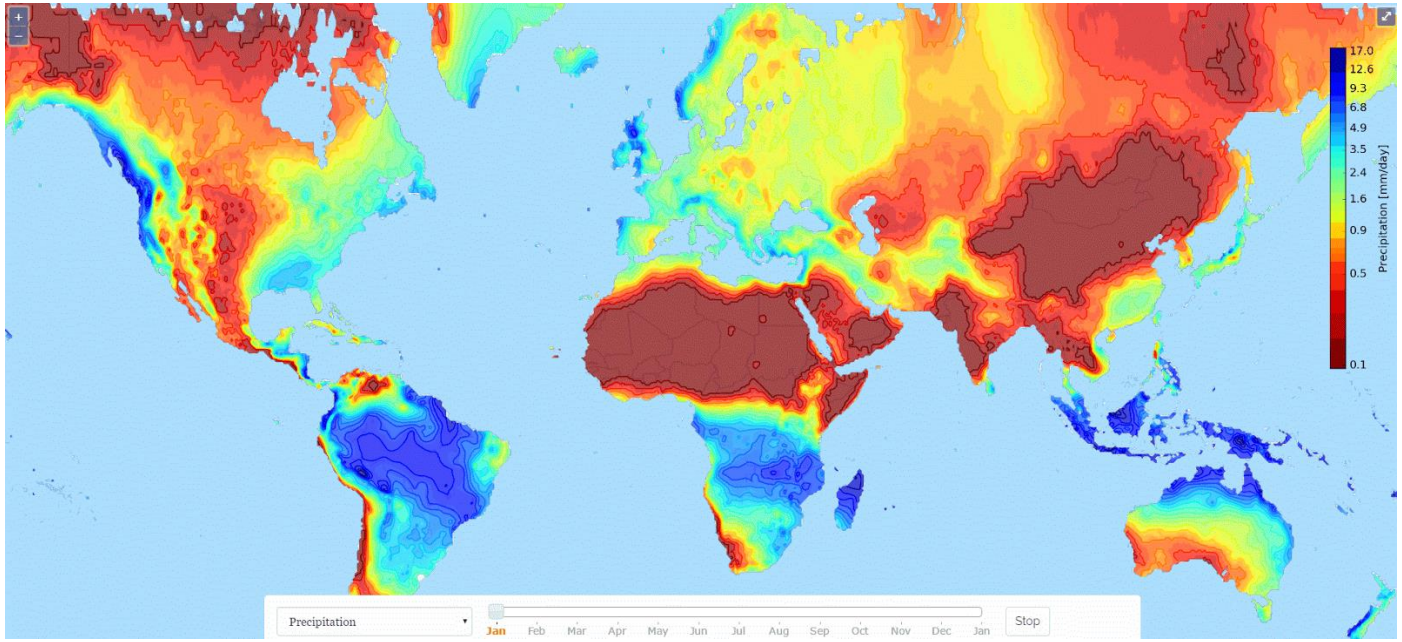
```python
import numpy as np
import pandas as pd

data = pd.read_csv('advertising.csv').to_numpy()
np.random.shuffle(data)

x = data[:,:3]
y = data[:,3:]

max_value = np.max(x)
min_value = np.min(x)
avg = np.mean(x)
x = (x-avg) / (max_value-min_value)
```

```python
import numpy as np

# tính output
output = x_test.dot(thetas)

# tính loss
loss = np.abs(output - y_test)
print('Test Loss: ', loss.mean())
```

```python
import numpy as np

n_iterations = 150
learning_rate = 0.01

# khởi tạo giá trị tham số
thetas = np.random.randn(4, 1)
losses = []

for i in range(n_iterations):
    # tính output
    output = x_train.dot(thetas)

    # tính loss
    loss = (output - y_train)**2

    # tính đạo hàm cho loss
    loss_grd = 2*(output - y_train)/N

    # tính đạo hàm cho các tham số
    gradients = x_train.T.dot(loss_grd)

    # cập nhật tham số
    thetas = thetas - learning_rate*gradients

    mean_loss = np.sum(loss)/N
    losses.append(mean_loss)
```

Data Frame

❖ Case study 3



Predict future temperature in weather forecasting

Data Frame

❖ Case study 3

Problem Statement: Given temperature from the previous 6 hours (including the current one), predict temperature of the next 1 hour.

Hour	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00
Condition								
Temperature	32	31	31	30	29	26	25	

Data Frame

Temperature forecasting

Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)
2006-04-01 00	Partly Cloudy	rain	9.472222222	7.388888889	0.89	14.1197	251	15.8263
2006-04-01 01	Partly Cloudy	rain	9.355555556	7.227777778	0.86	14.2646	259	15.8263
2006-04-01 02	Mostly Cloudy	rain	9.377777778	9.377777778	0.89	3.9284	204	14.9569
2006-04-01 03	Partly Cloudy	rain	8.288888889	5.944444444	0.83	14.1036	269	15.8263
2006-04-01 04	Mostly Cloudy	rain	8.755555556	6.977777778	0.83	11.0446	259	15.8263
2006-04-01 05	Partly Cloudy	rain	9.222222222	7.111111111	0.85	13.9587	258	14.9569
2006-04-01 06	Partly Cloudy	rain	7.733333333	5.522222222	0.95	12.3648	259	9.982
2006-04-01 07	Partly Cloudy	rain	8.772222222	6.527777778	0.89	14.1519	260	9.982
2006-04-01 08	Partly Cloudy	rain	10.82222222	10.82222222	0.82	11.3183	259	9.982
2006-04-01 09	Partly Cloudy	rain	13.77222222	13.77222222	0.72	12.5258	279	9.982
2006-04-01 10	Partly Cloudy	rain	16.01666667	16.01666667	0.67	17.5651	290	11.2056
2006-04-01 11	Partly Cloudy	rain	17.14444444	17.14444444	0.54	19.7869	316	11.4471
2006-04-01 12	Partly Cloudy	rain	17.8	17.8	0.55	21.9443	281	11.27
2006-04-01 13	Partly Cloudy	rain	17.33333333	17.33333333	0.51	20.6885	289	11.27
2006-04-01 14	Partly Cloudy	rain	18.87777778	18.87777778	0.47	15.3755	262	11.4471
2006-04-01 15	Partly Cloudy	rain	18.91111111	18.91111111	0.46	10.4006	288	11.27
2006-04-01 16	Partly Cloudy	rain	15.38888889	15.38888889	0.6	14.4095	251	11.27
2006-04-01 17	Mostly Cloudy	rain	15.55	15.55	0.63	11.1573	230	11.4471
2006-04-01 18	Mostly Cloudy	rain	14.25555556	14.25555556	0.69	8.5169	163	11.2056
2006-04-01 19	Mostly Cloudy	rain	13.14444444	13.14444444	0.7	7.6314	139	11.2056
2006-04-01 20	Mostly Cloudy	rain	11.55	11.55	0.77	7.3899	147	11.0285
2006-04-01 21	Mostly Cloudy	rain	11.18333333	11.18333333	0.76	4.9266	160	9.982
2006-04-01 22	Partly Cloudy	rain	10.11666667	10.11666667	0.79	6.6493	163	15.8263
2006-04-01 23	Mostly Cloudy	rain	10.2	10.2	0.77	3.9284	152	14.9569
2006-04-10 00	Partly Cloudy	rain	10.42222222	10.42222222	0.62	16.9855	150	15.8263
2006-04-10 01	Partly Cloudy	rain	9.911111111	7.566666667	0.66	17.2109	149	15.8263
2006-04-10 02	Mostly Cloudy	rain	11.18333333	11.18333333	0.8	10.8192	163	14.9569
2006-04-10 03	Partly Cloudy	rain	7.155555556	5.044444444	0.79	11.0768	180	15.8263
2006-04-10 04	Partly Cloudy	rain	6.111111111	4.816666667	0.82	6.6493	161	15.8263

Data Frame

❖ Case study 3

Time	Temperature (C)
2006-04-01 00:00:00.000 +0200	9.472222
2006-04-01 01:00:00.000 +0200	9.355556
2006-04-01 02:00:00.000 +0200	9.377778
2006-04-01 03:00:00.000 +0200	8.288889
2006-04-01 04:00:00.000 +0200	8.755556
2006-04-01 05:00:00.000 +0200	9.222222
2006-04-01 06:00:00.000 +0200	7.733333

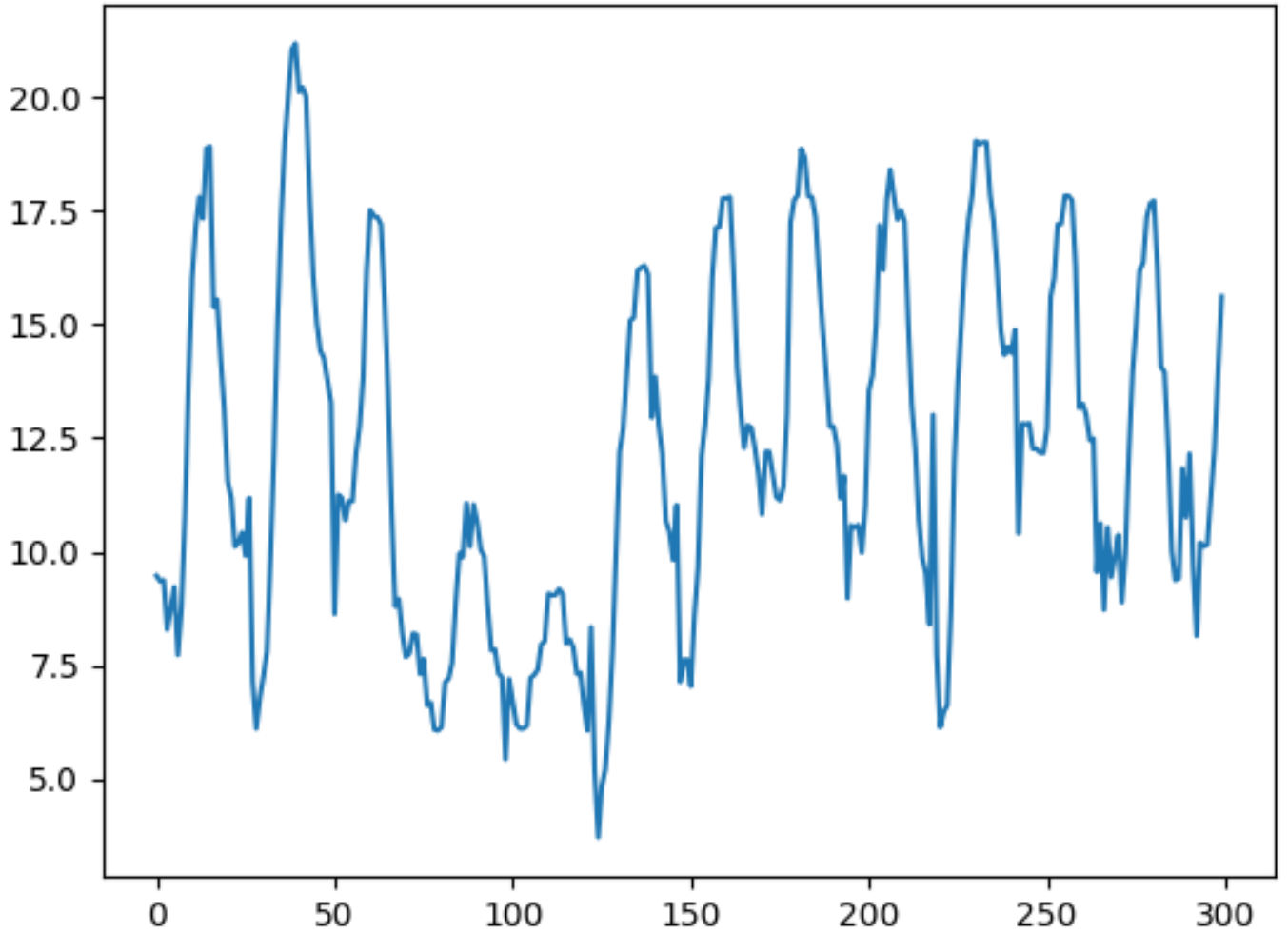
x

y

Temperature forecasting datatable

Data Frame

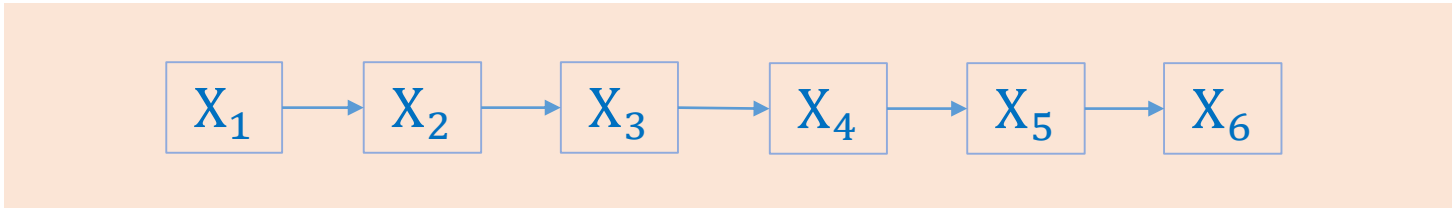
Temperature forecasting

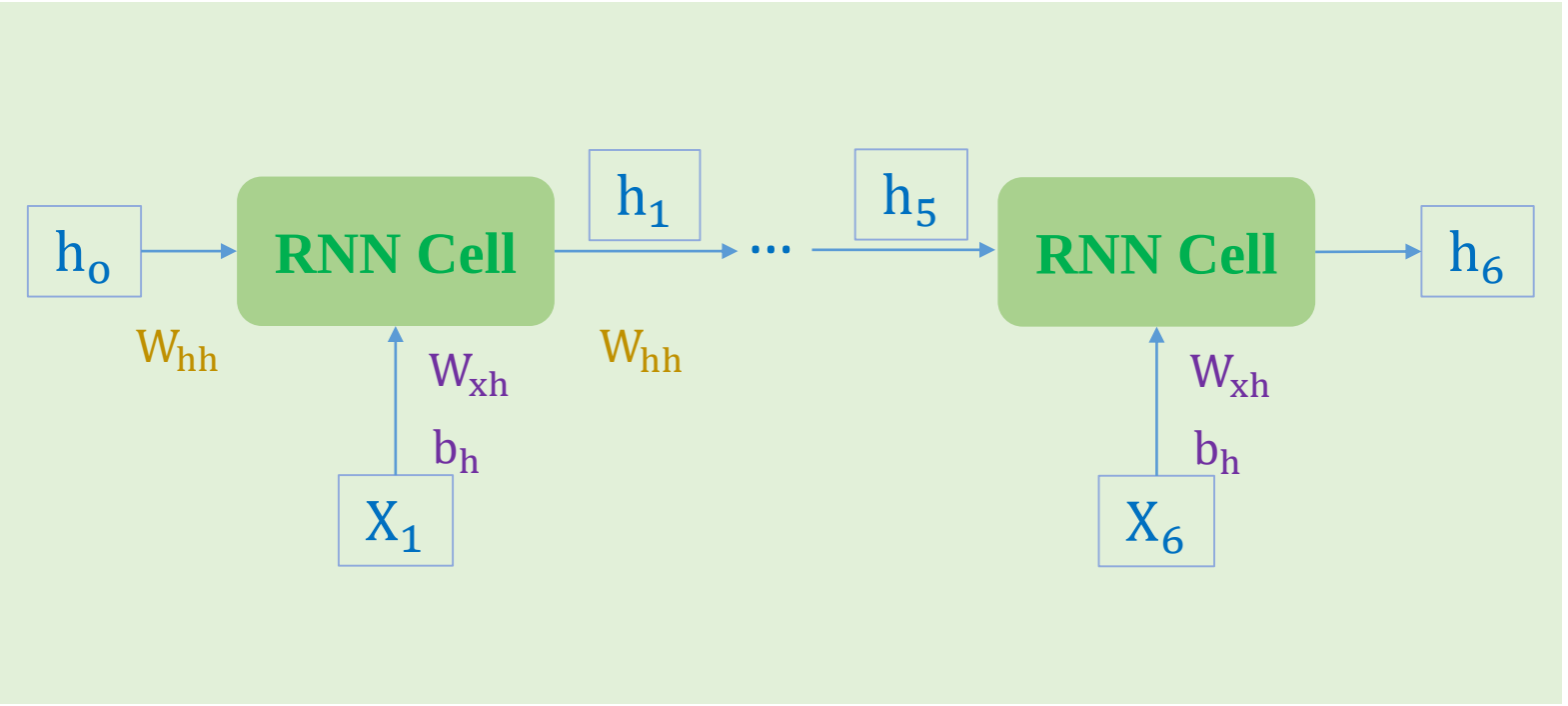


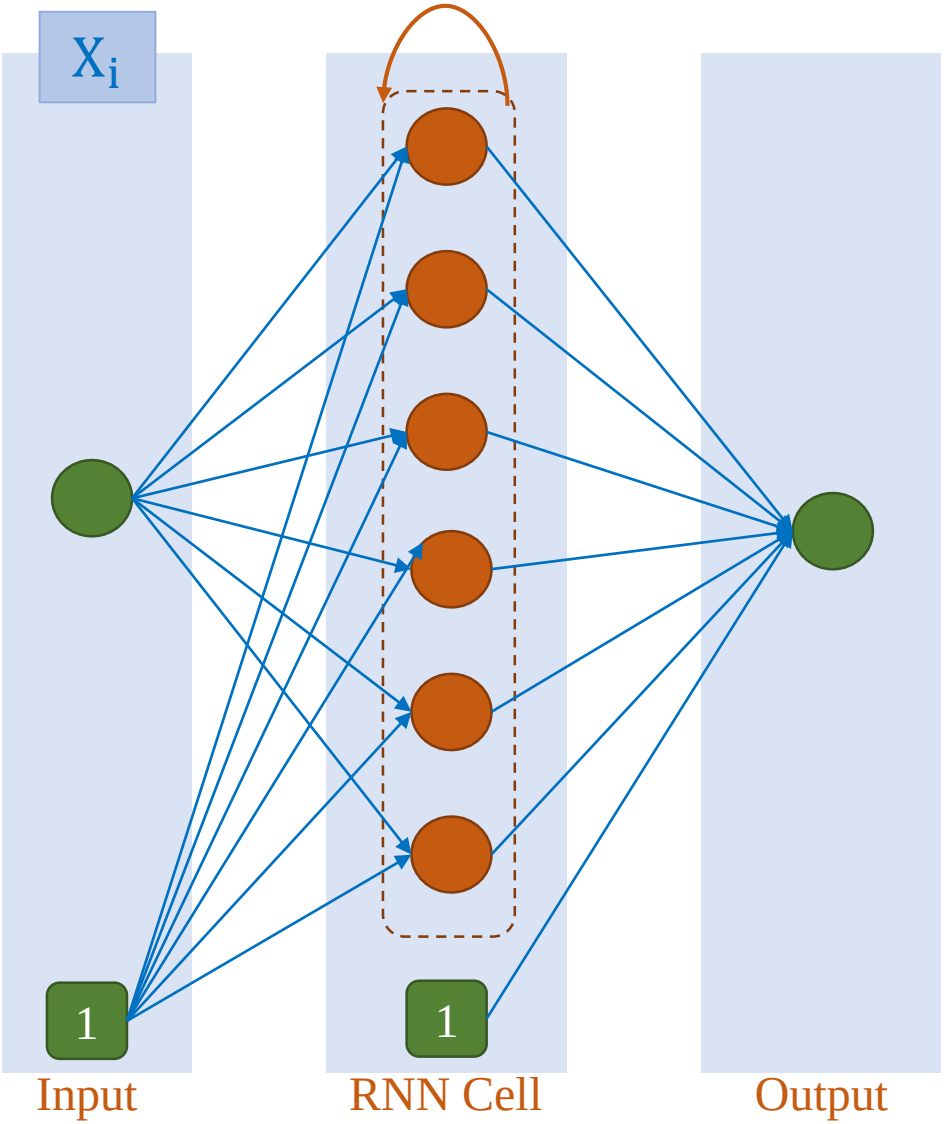
Date	Temperature (C)
2006-04-01 00	9.472222222
2006-04-01 01	9.355555556
2006-04-01 02	9.377777778
2006-04-01 03	8.288888889
2006-04-01 04	8.755555556
2006-04-01 05	9.222222222
2006-04-01 06	7.733333333
2006-04-01 07	8.772222222
2006-04-01 08	10.82222222
2006-04-01 09	13.77222222
2006-04-01 10	16.01666667
2006-04-01 11	17.14444444
2006-04-01 12	17.8
2006-04-01 13	17.33333333
2006-04-01 14	18.87777778
2006-04-01 15	18.91111111
2006-04-01 16	15.38888889
2006-04-01 17	15.55
2006-04-01 18	14.25555556
2006-04-01 19	13.14444444
2006-04-01 20	11.55
2006-04-01 21	11.18333333
2006-04-01 22	10.11666667
2006-04-01 23	10.2
2006-04-10 00	10.42222222
2006-04-10 01	9.911111111
2006-04-10 02	11.18333333
2006-04-10 03	7.155555556
2006-04-10 04	6.111111111

Data Frame

Temperature forecasting

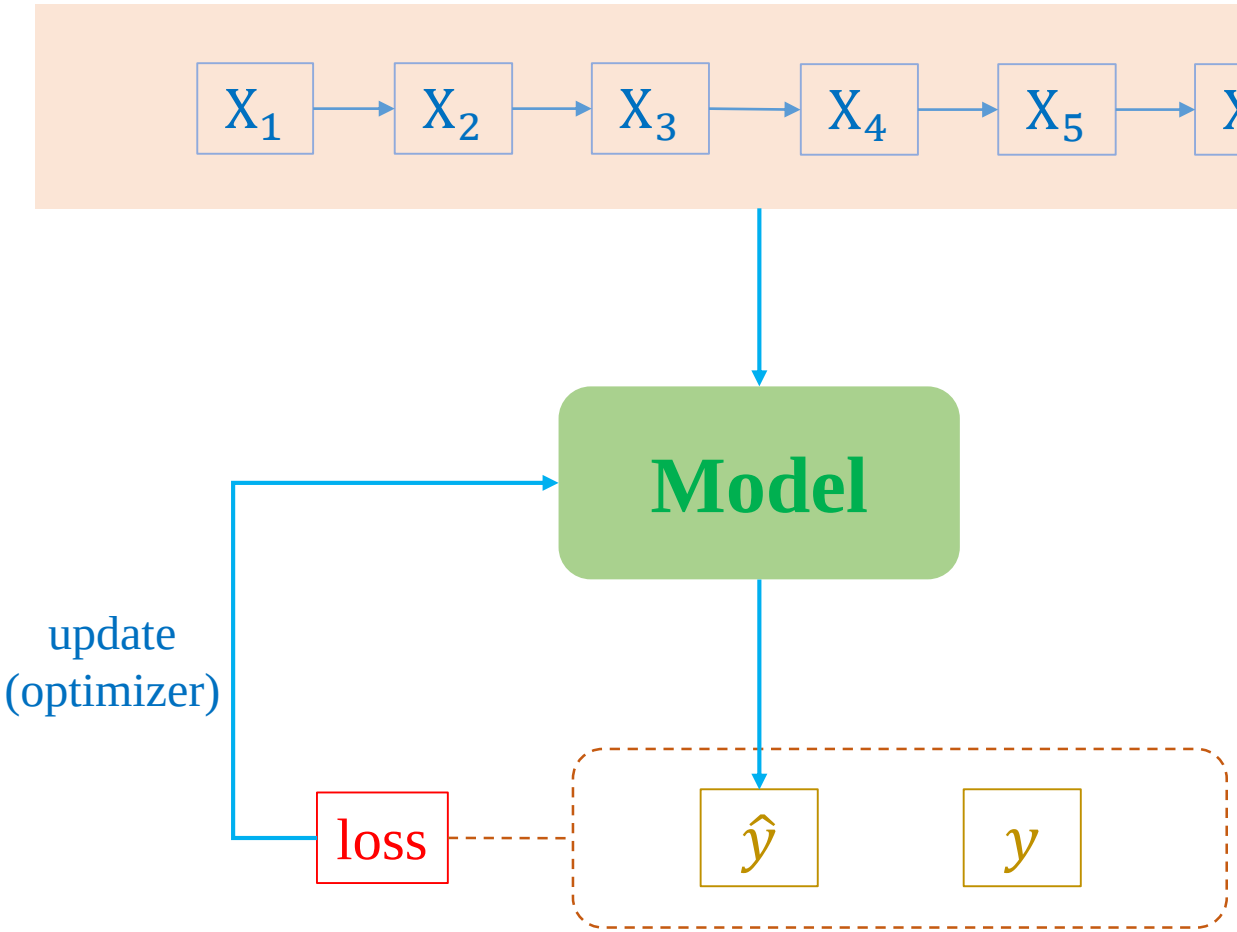


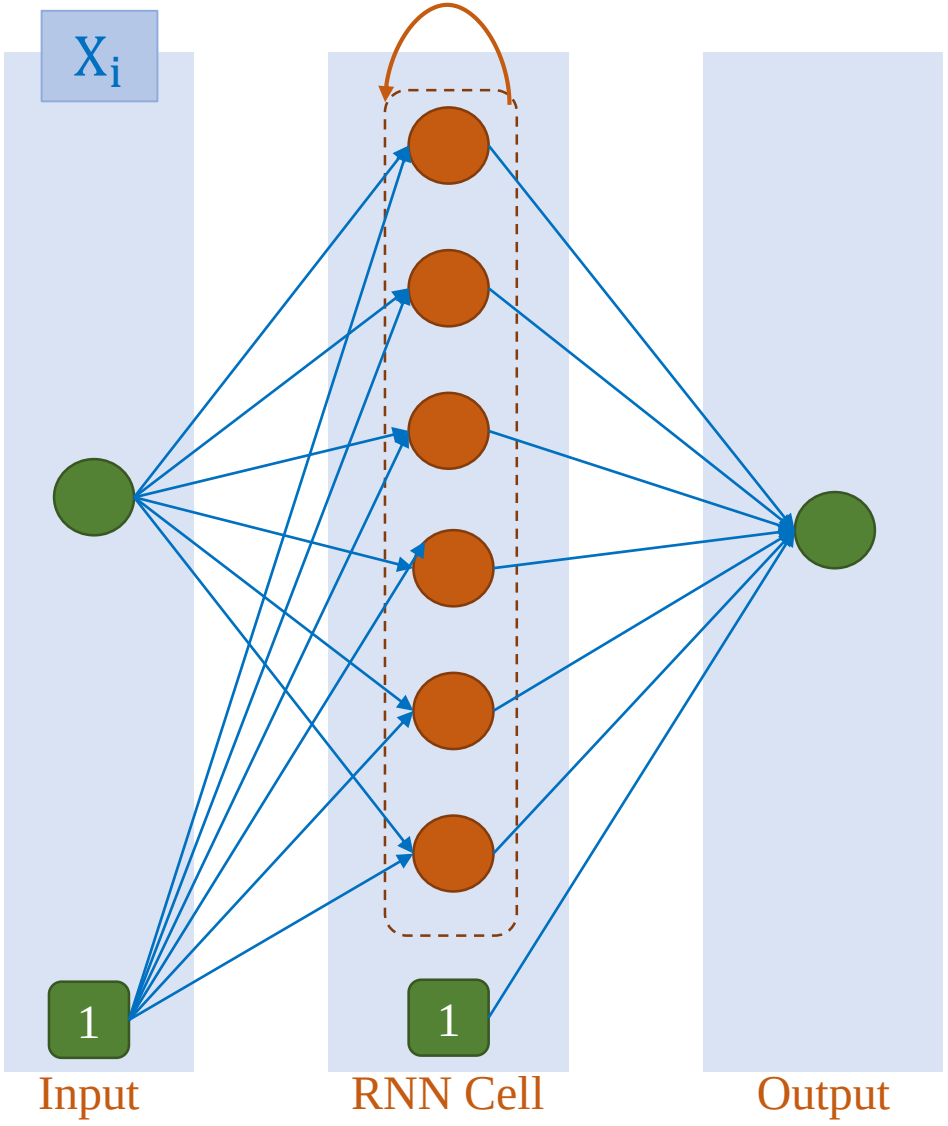




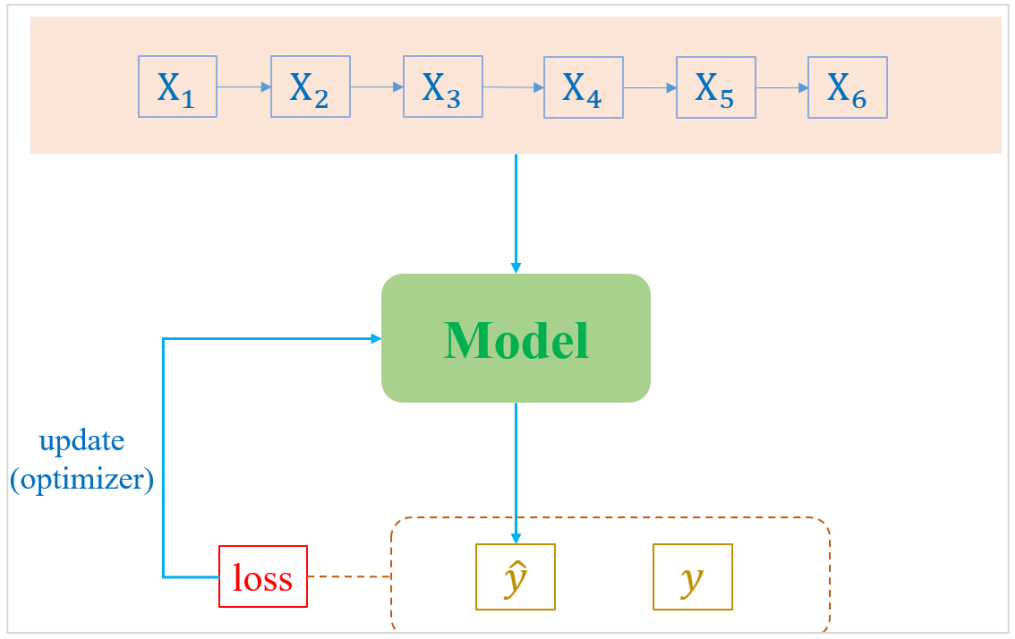
Data Frame

Temperature forecasting

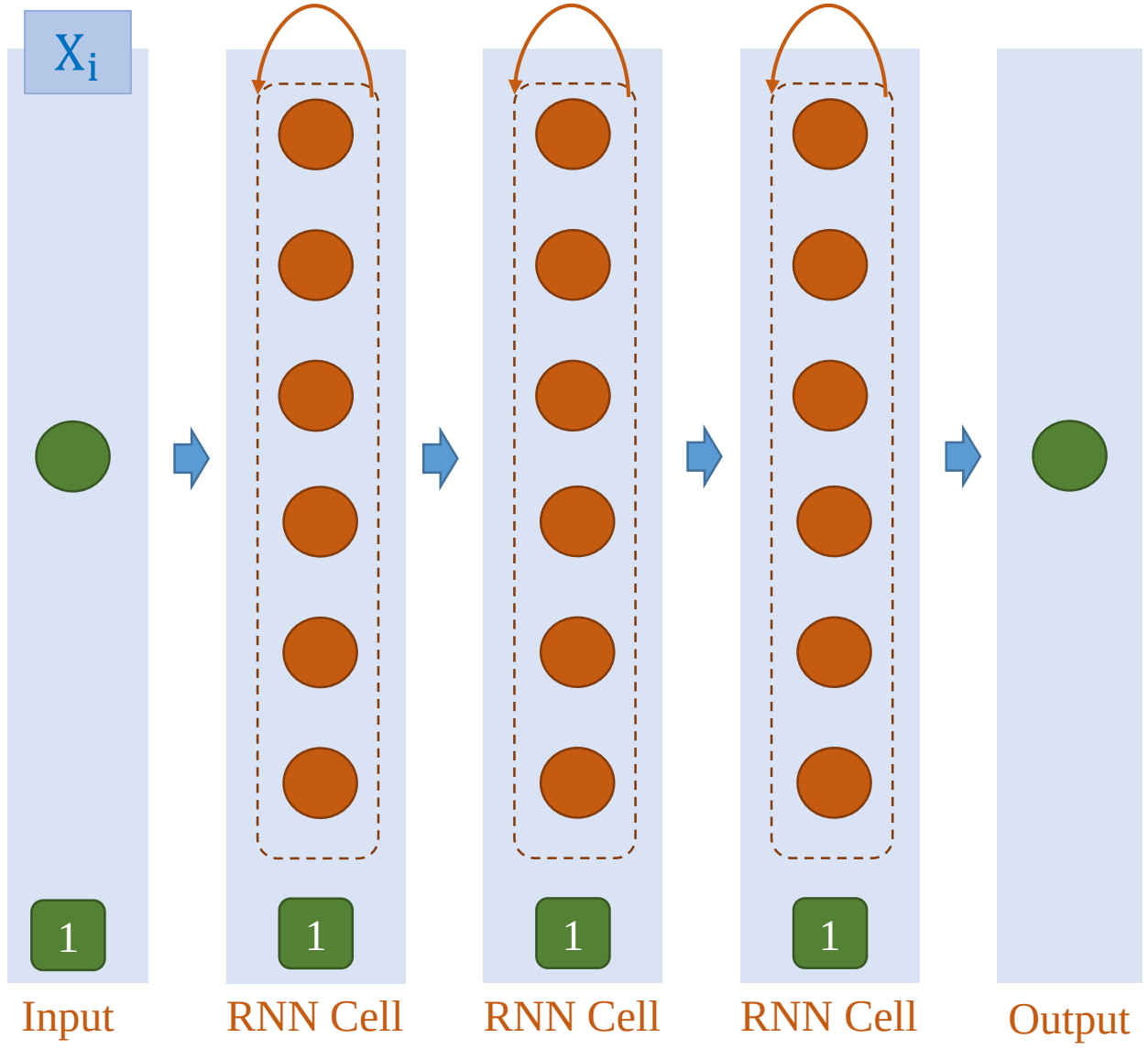




Data Frame




```
model = tf.keras.Sequential([
        # input
        tf.keras.Input(shape=(6,1)),

        tf.keras.layers.SimpleRNN(6, return_sequences=True),
        tf.keras.layers.SimpleRNN(6, return_sequences=True),
        tf.keras.layers.SimpleRNN(6),

        # Output Layer
        tf.keras.layers.Dense(1)])
```



Data Frame

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(6,1)),
        tf.keras.layers.Flatten(),

        # Dense
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(32, activation='relu'),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(6,1)),

        tf.keras.layers.SimpleRNN(16, return_sequences=True),
        tf.keras.layers.SimpleRNN(16, return_sequences=True),
        tf.keras.layers.SimpleRNN(16),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(6,1)),

        tf.keras.layers.LSTM(16, return_sequences=True),
        tf.keras.layers.LSTM(16, return_sequences=True),
        tf.keras.layers.LSTM(16),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(6,1)),

        layers.Bidirectional(layers.LSTM(16,
                                         return_sequences=True)),
        layers.Bidirectional(layers.LSTM(16,
                                         return_sequences=True)),
        layers.Bidirectional(layers.LSTM(16)),

        tf.keras.layers.Dense(1)])
```

Data Frame

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(20,1)),
        tf.keras.layers.Flatten(),

        # Dense
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(32, activation='relu'),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(20,1)),

        tf.keras.layers.SimpleRNN(16, return_sequences=True),
        tf.keras.layers.SimpleRNN(16, return_sequences=True),
        tf.keras.layers.SimpleRNN(16),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(20,1)),

        tf.keras.layers.LSTM(16, return_sequences=True),
        tf.keras.layers.LSTM(16, return_sequences=True),
        tf.keras.layers.LSTM(16),

        # Output Layer
        tf.keras.layers.Dense(1)])
```

```
model = tf.keras.Sequential([
        tf.keras.Input(shape=(20,1)),

        layers.Bidirectional(layers.LSTM(16,
                                         return_sequences=True)),
        layers.Bidirectional(layers.LSTM(16,
                                         return_sequences=True)),
        layers.Bidirectional(layers.LSTM(16)),

        tf.keras.layers.Dense(1)])
```

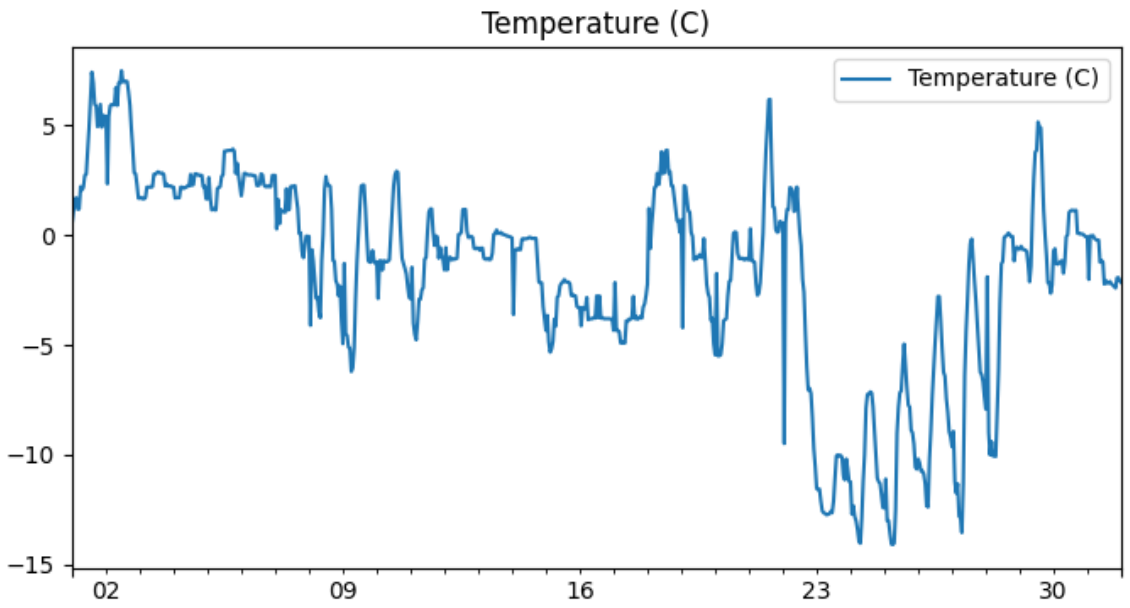
Data Frame

❖ Case study 3

```
import pandas as pd

df = pd.read_csv('weatherHistory_univariate.csv')
df = df.sort_values('Formatted Date')

df['Formatted Date'] = [pd.to_datetime(x)
                        for x in df['Formatted Date']]
```

	Formatted Date	Temperature (C)
2880	2006-01-01 00:00:00+01:00	0.577778
2881	2006-01-01 01:00:00+01:00	1.161111
2882	2006-01-01 02:00:00+01:00	1.666667
2883	2006-01-01 03:00:00+01:00	1.711111
2884	2006-01-01 04:00:00+01:00	1.183333
...	...	...
89728	2016-12-31 19:00:00+01:00	0.488889
89729	2016-12-31 20:00:00+01:00	0.072222
89730	2016-12-31 21:00:00+01:00	-0.233333
89731	2016-12-31 22:00:00+01:00	-0.472222
89732	2016-12-31 23:00:00+01:00	-0.677778

```
time1 = pd.to_datetime('2006-01-01 00:00:00+01:00',
                       format='%Y-%m-%d', utc=True)
time2 = pd.to_datetime('2006-01-31 23:59:59+00:00',
                       format='%Y-%m-%d', utc=True)

df_jan = df.loc[(df['Formatted Date']<time2) &
                (df['Formatted Date']>=time1)]
df_jan.plot(x='Formatted Date', y='Temperature (C)',
            kind='line',figsize=(8,4),
            title='Temperature (C)')
```



Data Frame

❖ Case study 3

```
df['date'] = [x.date() for x in df['Formatted Date']]
df['month'] = [x.month for x in df['Formatted Date']]
df['year'] = [x.year for x in df['Formatted Date']]
df['day'] = [x.day for x in df['Formatted Date']]
df['hour'] = [x.hour for x in df['Formatted Date']]
```

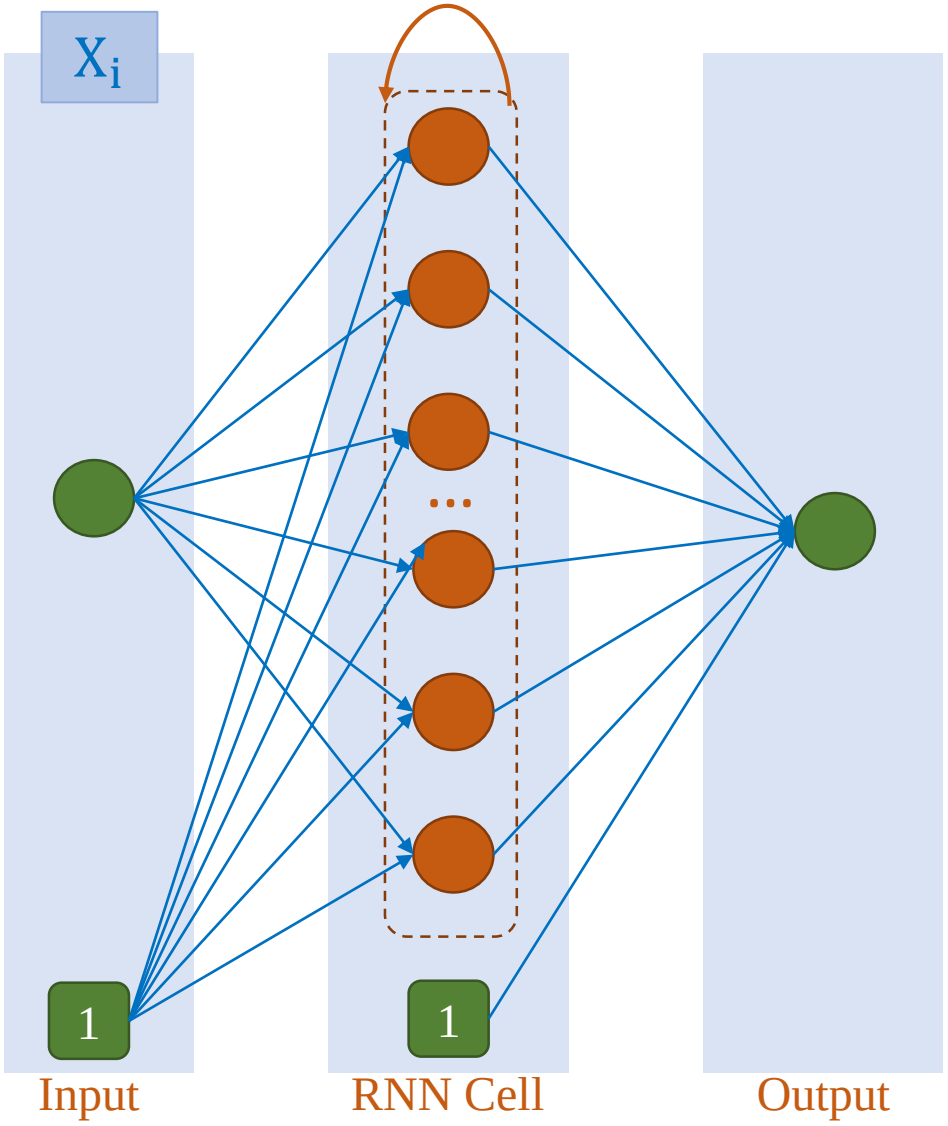
	Formatted Date	Temperature (C)	date	month	year	day	hour
2880	2006-01-01 00:00:00+01:00	0.577778	38718	1	2006	1	0
2881	2006-01-01 01:00:00+01:00	1.161111	38718	1	2006	1	1
2882	2006-01-01 02:00:00+01:00	1.666667	38718	1	2006	1	2
2883	2006-01-01 03:00:00+01:00	1.711111	38718	1	2006	1	3
2884	2006-01-01 04:00:00+01:00	1.183333	38718	1	2006	1	4
...	...	...	...	...	...	...	...
89728	2016-12-31 19:00:00+01:00	0.488889	42735	12	2016	31	19
89729	2016-12-31 20:00:00+01:00	0.072222	42735	12	2016	31	20
89730	2016-12-31 21:00:00+01:00	-0.233333	42735	12	2016	31	21
89731	2016-12-31 22:00:00+01:00	-0.472222	42735	12	2016	31	22
89732	2016-12-31 23:00:00+01:00	-0.677778	42735	12	2016	31	23

Data Frame

❖ Case study 3

```
import tensorflow as tf

normalize_layer = tf.keras.layers.Normalization(axis=-1)
normalize_layer.adapt(np.vstack((X_train, X_val, X_test)))

model = tf.keras.Sequential([tf.keras.Input(shape=(INPUT_SIZE, 1)),
                             normalize_layer,
                             tf.keras.layers.LSTM(32),
                             tf.keras.layers.Dense(LABEL_SIZE)])
```

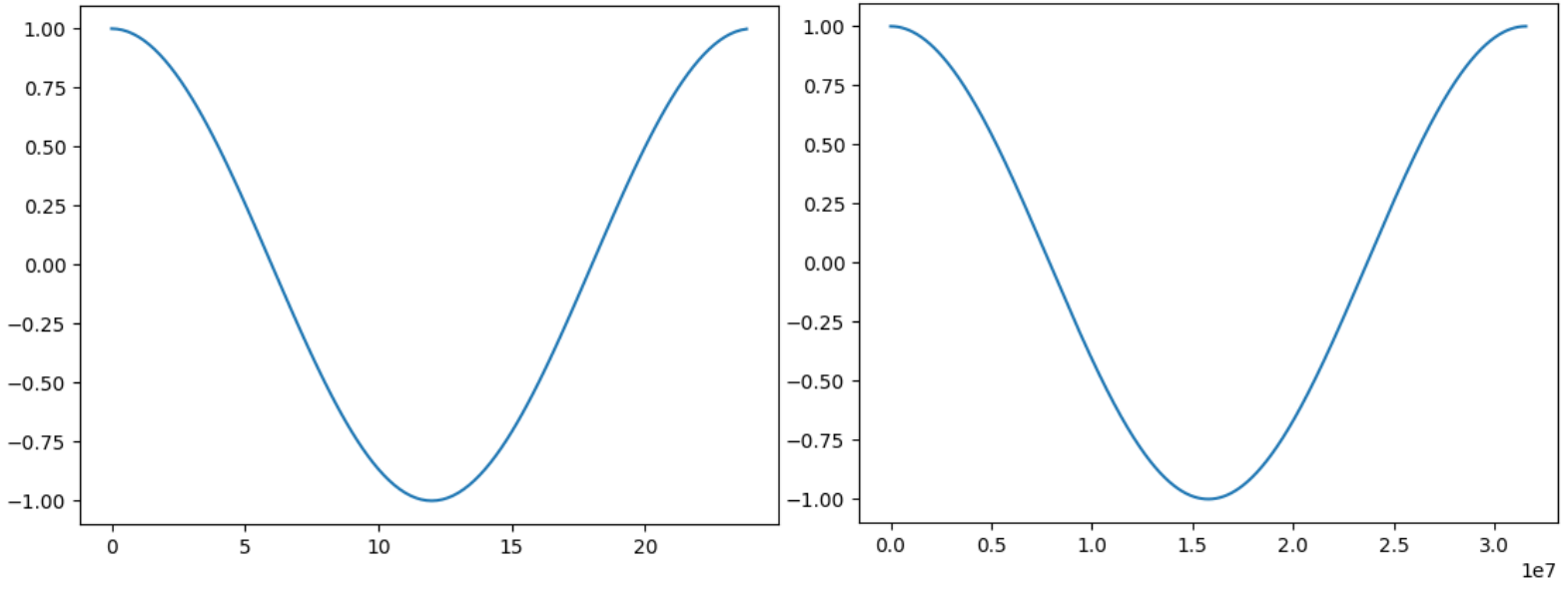
Formatted Date	Temperature (C)
2006-01-01 00:00:00+01:00	0.577778
2006-01-01 01:00:00+01:00	1.161111
2006-01-01 02:00:00+01:00	1.666667
2006-01-01 03:00:00+01:00	1.711111
2006-01-01 04:00:00+01:00	1.183333
...	...
2016-12-31 19:00:00+01:00	0.488889
2016-12-31 20:00:00+01:00	0.072222
2016-12-31 21:00:00+01:00	-0.233333
2016-12-31 22:00:00+01:00	-0.472222
2016-12-31 23:00:00+01:00	-0.677778

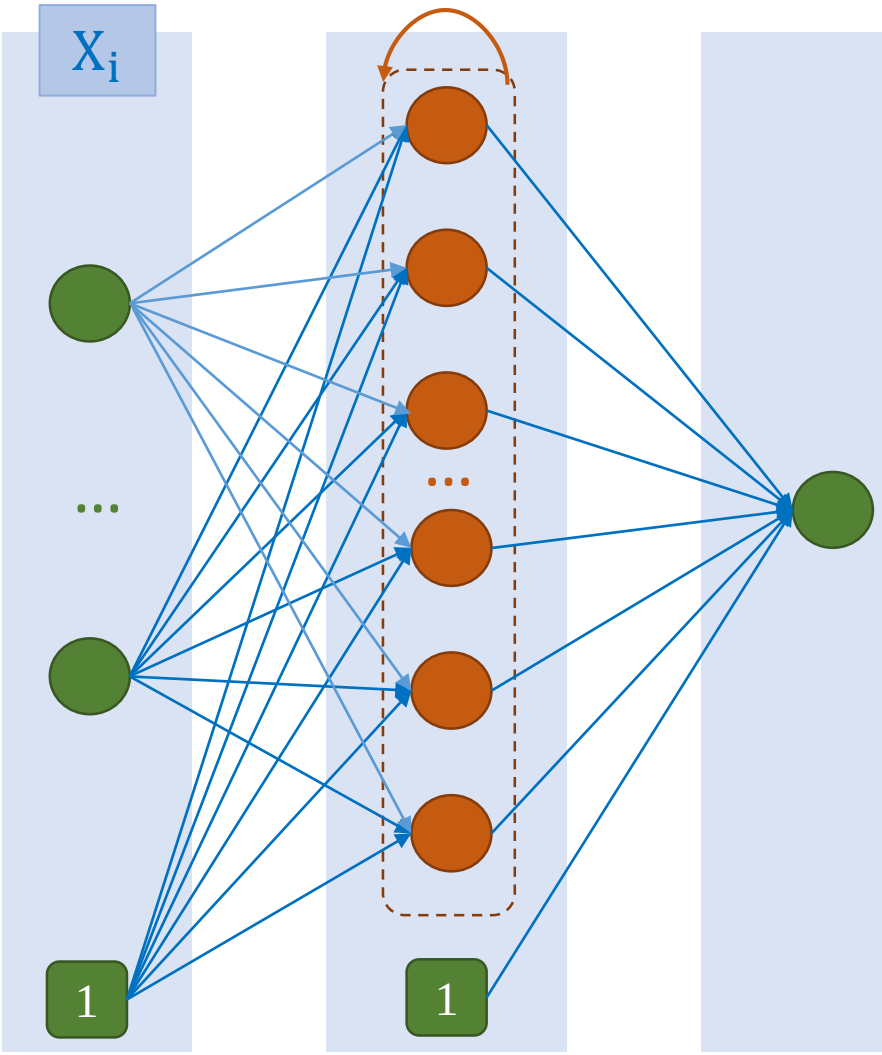


Case study 4

	Formatted Date	Temperature (C)	date	month	year	day	hour	day_cos	day_sin	timestamp	year_cos	year_sin
2880	2006-01-01 00:00:00+01:00	0.577778	38718	1	2006	1	0	1	0	1136070000	1	-0.000717
2881	2006-01-01 01:00:00+01:00	1.161111	38718	1	2006	1	1	0.9659258	0.25882	1136074000	1	-8.82E-15
2882	2006-01-01 02:00:00+01:00	1.666667	38718	1	2006	1	2	0.8660254	0.5	1136077000	1	0.0007168
2883	2006-01-01 03:00:00+01:00	1.711111	38718	1	2006	1	3	0.7071068	0.70711	1136081000	0.999999	0.0014335
2884	2006-01-01 04:00:00+01:00	1.183333	38718	1	2006	1	4	0.5	0.86603	1136084000	0.999998	0.0021503
2885	2006-01-01 05:00:00+01:00	1.205556	38718	1	2006	1	5	0.258819	0.96593	1136088000	0.999996	0.0028671
2886	2006-01-01 06:00:00+01:00	2.222222	38718	1	2006	1	6	6.1232E-17	1	1136092000	0.999994	0.0035838

```
df['day_cos'] = [np.cos(x * (2 * np.pi / 24)) for x in df['hour']]
df['day_sin'] = [np.sin(x * (2 * np.pi / 24)) for x in df['hour']]

df['timestamp'] = [x.timestamp() for x in df['Formatted Date']]
year = 365.25*24*60*60
df['year_cos'] = [np.cos((x) * (2 * np.pi / year)) for x in df['timestamp']]
df['year_sin'] = [np.sin((x) * (2 * np.pi / year)) for x in df['timestamp']]
```





Data Frame

Case study 4

	Formatted Date	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)
2880	2006-01-01 00:00:00+01:00	0.577778	-4.05	0.89	17.1143
2881	2006-01-01 01:00:00+01:00	1.161111	-3.238889	0.85	16.6152
2882	2006-01-01 02:00:00+01:00	1.666667	-3.155556	0.82	20.2538
2883	2006-01-01 03:00:00+01:00	1.711111	-2.194444	0.82	14.49
2884	2006-01-01 04:00:00+01:00	1.183333	-2.744444	0.86	13.9426
2885	2006-01-01 05:00:00+01:00	1.205556	-3.072222	0.85	15.9068
2886	2006-01-01 06:00:00+01:00	2.222222	-2.494444	0.79	20.5597

	Formatted Date	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	date	month	year	day	hour	day_cos	day_sin	timestamp
2880	2006-01-01 00:00:00+01:00	0.577778	-4.05	0.89	17.1143	38718	1	2006	1	0	1	0	1.136E+09
2881	2006-01-01 01:00:00+01:00	1.161111	-3.238889	0.85	16.6152	38718	1	2006	1	1	0.965926	0.2588	1.136E+09
2882	2006-01-01 02:00:00+01:00	1.666667	-3.155556	0.82	20.2538	38718	1	2006	1	2	0.866025	0.5	1.136E+09
2883	2006-01-01 03:00:00+01:00	1.711111	-2.194444	0.82	14.49	38718	1	2006	1	3	0.707107	0.7071	1.136E+09
2884	2006-01-01 04:00:00+01:00	1.183333	-2.744444	0.86	13.9426	38718	1	2006	1	4	0.5	0.866	1.136E+09
2885	2006-01-01 05:00:00+01:00	1.205556	-3.072222	0.85	15.9068	38718	1	2006	1	5	0.258819	0.9659	1.136E+09
2886	2006-01-01 06:00:00+01:00	2.222222	-2.494444	0.79	20.5597	38718	1	2006	1	6	6.12E-17	1	1.136E+09

Data Frame

❖ Case study 5

```
1  import pandas as pd
2
3  df = pd.read_csv('creditcard.csv')
4  df.groupby('Class').size()
```

Class	#samples
0	284315
1	492

False Negatives are more costly than False Positives

```
1  from tensorflow import keras
2
3  model = keras.Sequential([keras.layers.Dense(256, activation="relu",
4                                                  input_shape=(x_train.shape[-1],)),
5                            keras.layers.Dense(256, activation="relu"),
6                            keras.layers.Dropout(0.3),
7                            keras.layers.Dense(1, activation="sigmoid")])
8
9  counts = np.bincount(y_train[:, 0])
10 weight_for_0 = 1.0 / counts[0]
11 weight_for_1 = 1.0 / counts[1]
12 class_weight = {0: weight_for_0, 1: weight_for_1}
13
14 model.compile(optimizer=keras.optimizers.Adam(1e-2),
15               loss="binary_crossentropy", metrics=metrics)
16 model.fit(x_train, y_train, batch_size=1024, epochs=30,
17           validation_data=(x_val, y_val), class_weight=class_weight)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0	-1.3598	-0.0728	2.5363	1.3782	-0.3383	0.4624	0.2396	0.0987	0.3638	...	-0.0183	0.2778	-0.1105	0.0669	0.1285	-0.1891	0.1336	-0.0211	149.62	0
1	0	1.1919	0.2662	0.1665	0.4482	0.06	-0.0824	-0.0788	0.0851	-0.2554	...	-0.2258	-0.6387	0.1013	-0.3398	0.1672	0.1259	-0.009	0.0147	2.69	0
2	1	-1.3584	-1.3402	1.7732	0.3798	-0.5032	1.8005	0.7915	0.2477	-1.5147	...	0.248	0.7717	0.9094	-0.6893	-0.3276	-0.1391	-0.0554	-0.0598	378.66	0
3	1	-0.9663	-0.1852	1.793	-0.8633	-0.0103	1.2472	0.2376	0.3774	-1.387	...	-0.1083	0.0053	-0.1903	-1.1756	0.6474	-0.2219	0.0627	0.0615	123.5	0
4	2	-1.1582	0.8777	1.5487	0.403	-0.4072	0.0959	0.5929	-0.2705	0.8177	...	-0.0094	0.7983	-0.1375	0.1413	-0.206	0.5023	0.2194	0.2152	69.99	0
5	2	-0.426	0.9605	1.1411	-0.1683	0.421	-0.0297	0.4762	0.2603	-0.5687	...	-0.2083	-0.5598	-0.0264	-0.3714	-0.2328	0.1059	0.2538	0.0811	3.67	0
6	4	1.2297	0.141	0.0454	1.2026	0.1919	0.2727	-0.0052	0.0812	0.465	...	-0.1677	-0.2707	-0.1541	-0.7801	0.7501	-0.2572	0.0345	0.0052	4.99	0
7	7	-0.6443	1.418	1.0744	-0.4922	0.9489	0.4281	1.1206	-3.8079	0.6154	...	1.9435	-1.0155	0.0575	-0.6497	-0.4153	-0.0516	-1.2069	-1.0853	40.8	0
8	7	-0.8943	0.2862	-0.1132	-0.2715	2.6696	3.7218	0.3701	0.8511	-0.392	...	-0.0734	-0.2681	-0.2042	1.0116	0.3732	-0.3842	0.0117	0.1424	93.2	0
9	9	-0.3383	1.1196	1.0444	-0.2222	0.4994	-0.2468	0.6516	0.0695	-0.7367	...	-0.2469	-0.6338	-0.1208	-0.3851	-0.0697	0.0942	0.2462	0.0831	3.68	0

Metrics

Result	Prediction
True Positive (TP): A correct detection.	
False Positive (FP): A wrong detection.	
False Negative (FN): A ground truth not detected	
True Negative (TN): Does not apply.	

❖ Precision

❖ Ability of a model to identify only the relevant objects

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{all\ detections}$$

❖ Recall

❖ Ability of a model to find all the relevant cases (all ground truth bounding boxes)

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{all\ ground\ truths}$$