데이터사이언스응용
(Capstone design)

김응희

ehkim@sunmoon.ac.kr

Week 02

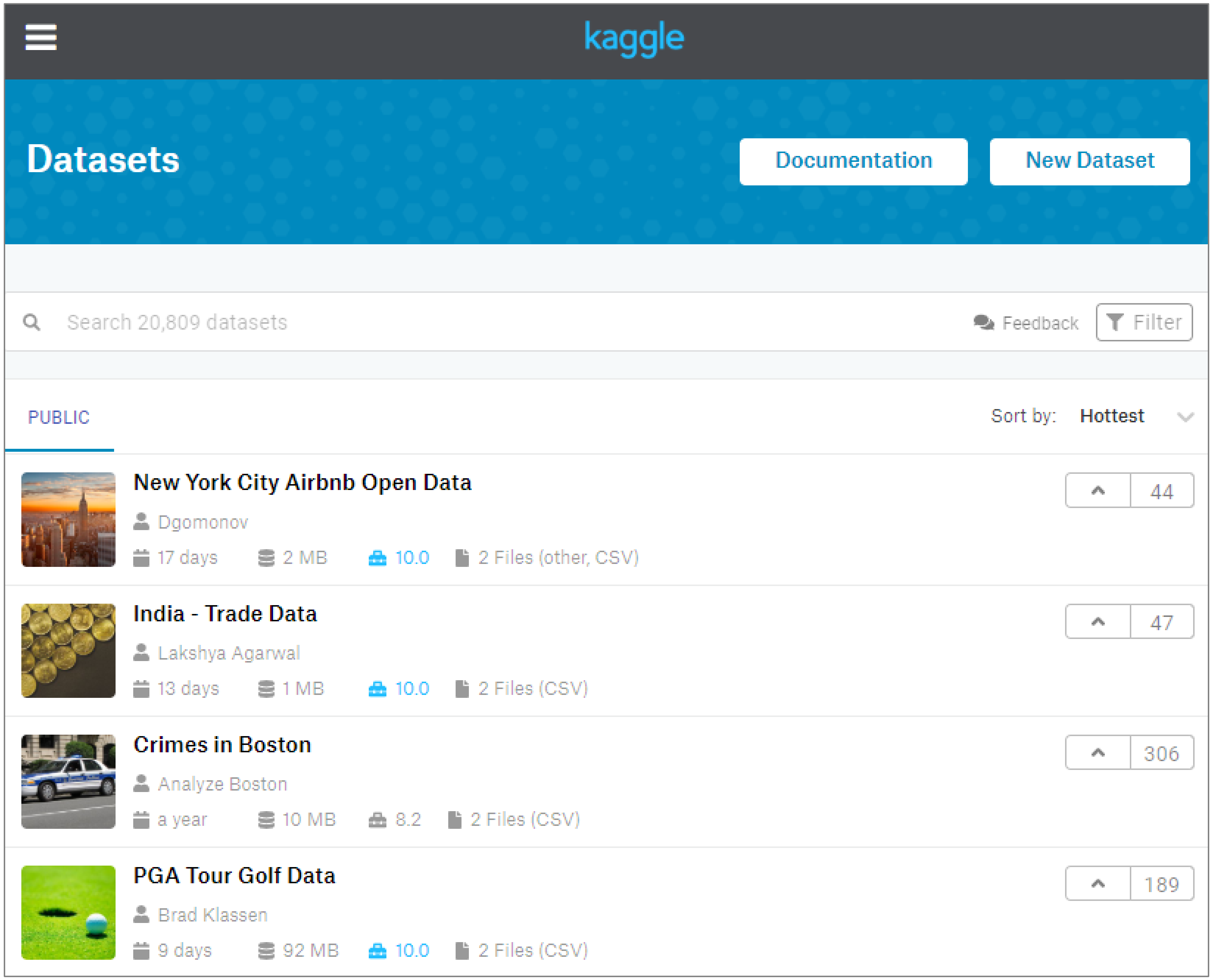
Working with Real Data

- The best way to learn about machine learning
  - Experiment with real-world data

Category	Repository
Popular open data repositories	UCI Irvine Machine Learning Repository http://archive.ics.uci.edu/ml
	Kaggle datasets https://www.kaggle.com/datasets
	Amazon's AWS datasets https://registry.opendata.aws/
	공공데이터포털 https://www.data.go.kr/
Meta portals	http://dataportals.org
	http://opendatamonitor.eu
	http://quandl.com
Others	https://homl.info/9
	https://reddit.com/r/datasets

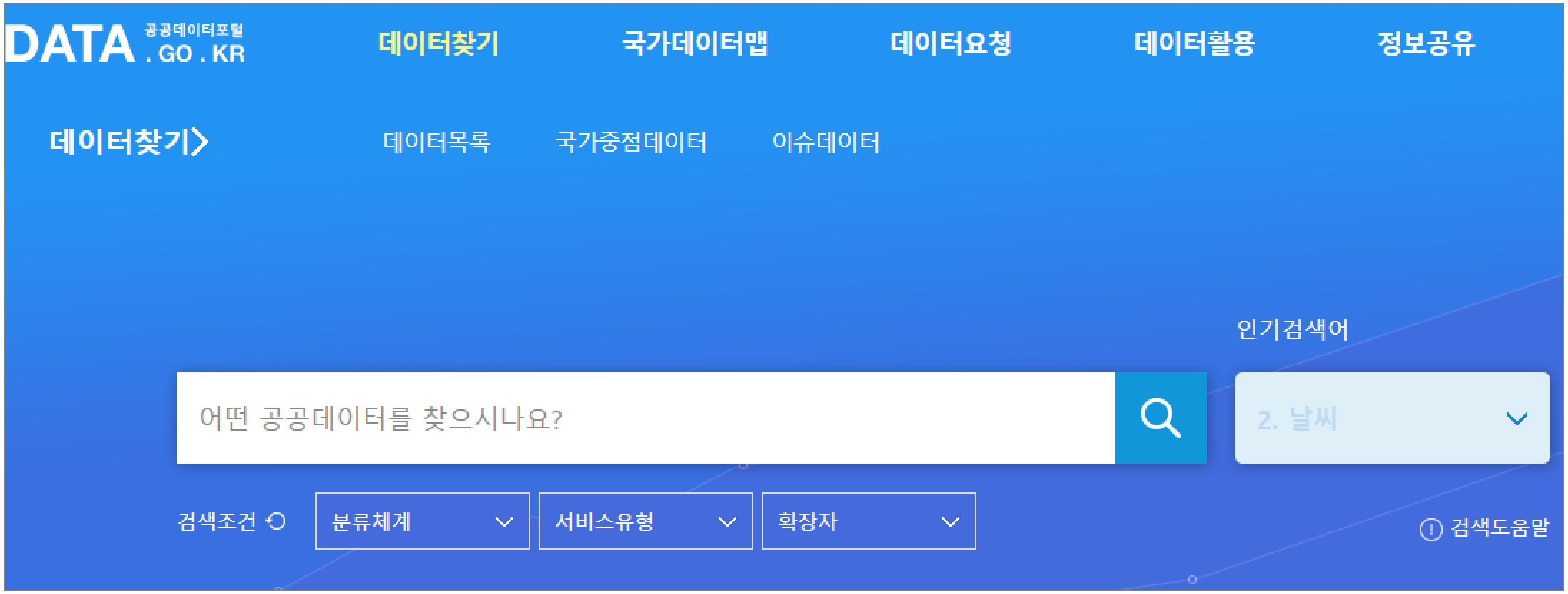
Kaggle dataset

https://www.kaggle.com/datasets



공공데이터포털

https://www.data.go.kr/



End-to-End Machine Learning Project

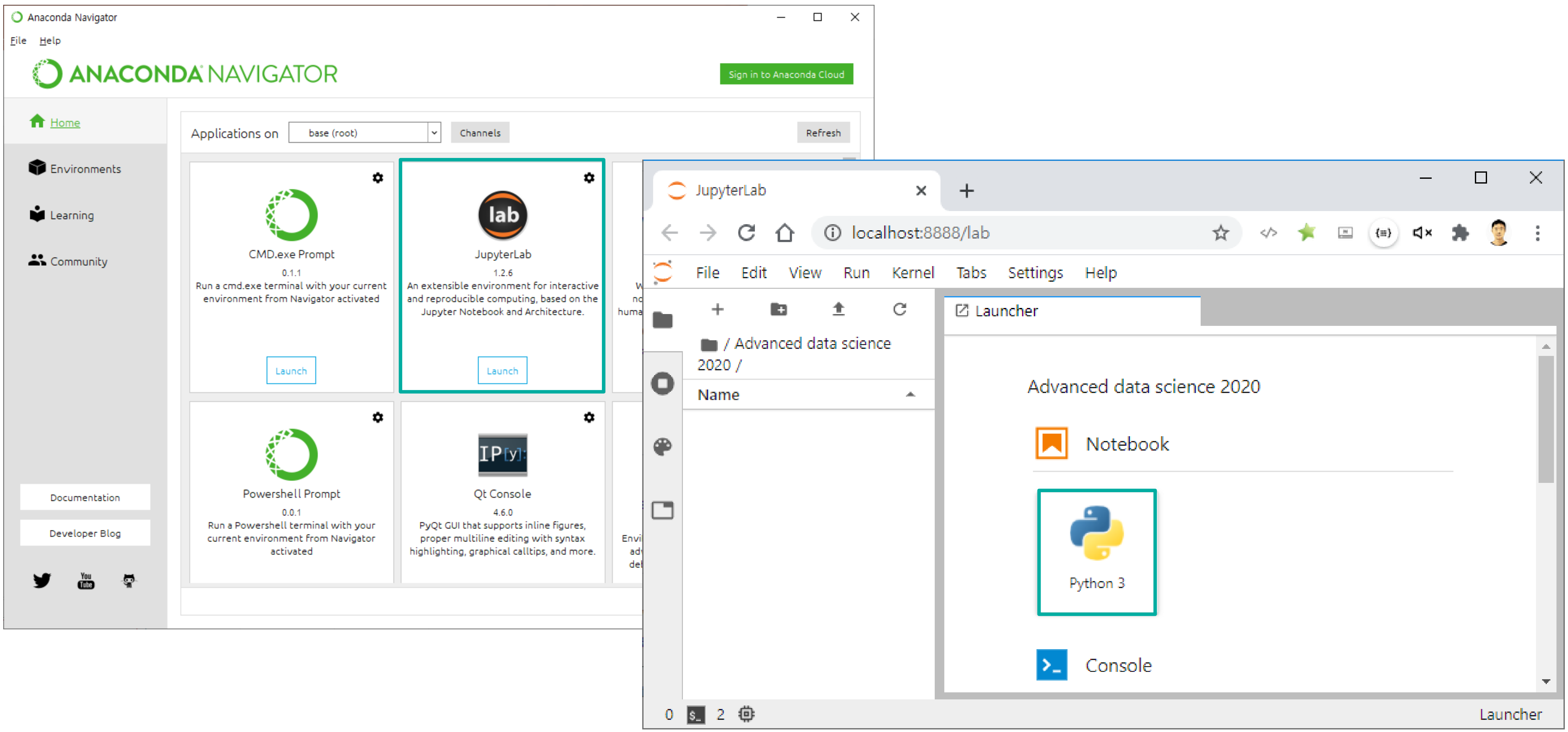
0. Look at the big picture.
1. Get the data.
2. Discover and visualize the data to gain insights.
3. Prepare the data for Machine Learning algorithms.
4. Select a model and train it.
5. Fine-tune your model.
6. Present your solution
7. Launch, monitor, and maintain your system.

End-to-End Machine Learning Project

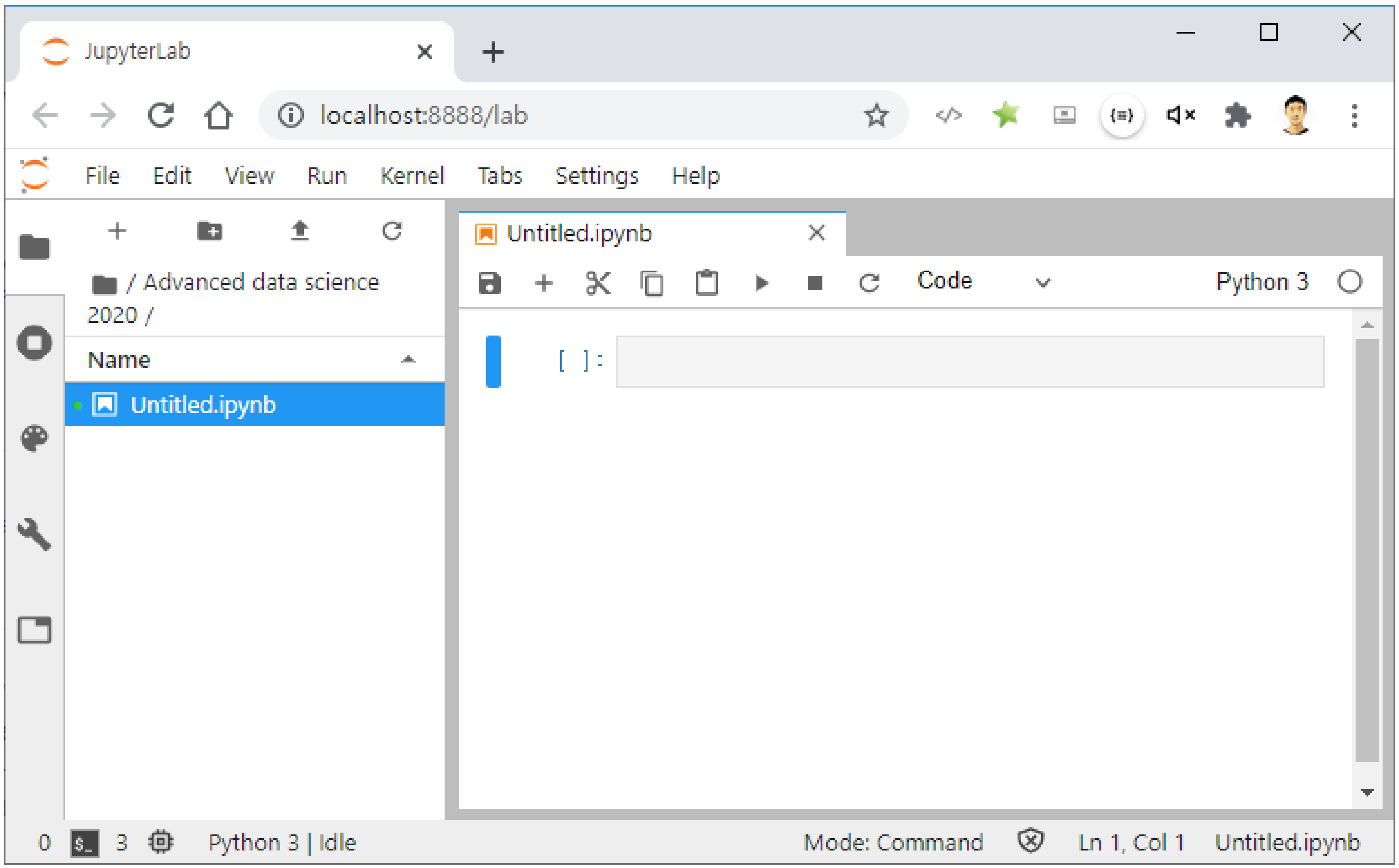
No.	Action	Package/library
0	Look at the big picture	–
1	Get the data	tarfile, urllib, pandas
2	Discover and visualize the data to gain insights	pandas, matplotlib
3	Prepare the data for Machine Learning algorithms	pandas, scikit-learn, numpy
4	Select a model and train it	
5	Fine-tune your model	
6	Present your solution	
7	Launch, monitor and maintain your system	joblib, flask

Get ready

- Run Anaconda Navigator > JupyterLab > New Notebook



Get ready

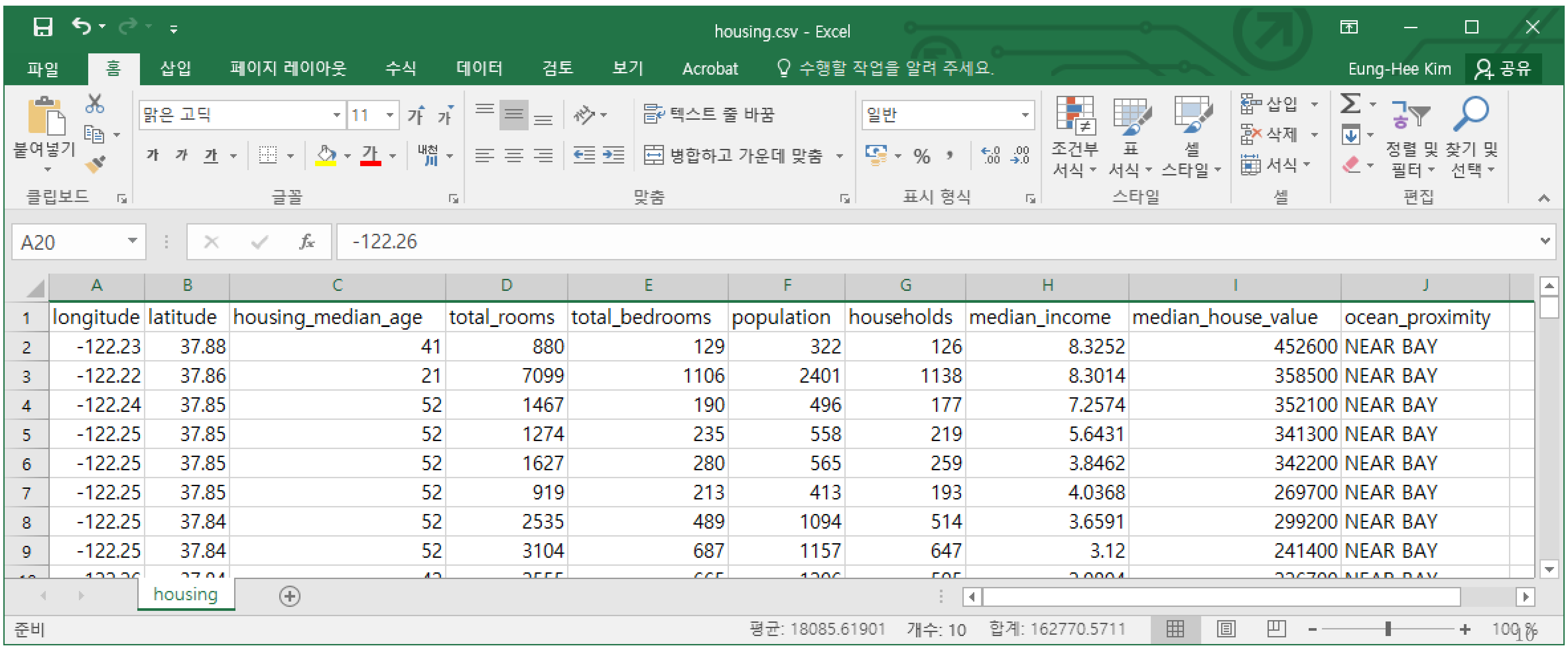

JupyterLab
localhost:8888/lab
File
Edit
View
Run
Kernel
Tabs
Settings
Help
/ Advanced data science 2020 /
Name
Untitled.ipynb
Untitled.ipynb
Code
Python 3
[ ]:
0
3
Python 3 | Idle
Mode: Command
Ln 1, Col 1
Untitled.ipynb

End-to-End Machine Learning Project

No.	Action	Package/library
0	Look at the big picture	–
1	Get the data	tarfile, urllib, pandas
2	Discover and visualize the data to gain insights	pandas, matplotlib
3	Prepare the data for Machine Learning algorithms	pandas, scikit-learn, numpy
4	Select a model and train it	
5	Fine-tune your model	
6	Present your solution	
7	Launch, monitor and maintain your system	joblib, flask

You are a recently hired data scientist.

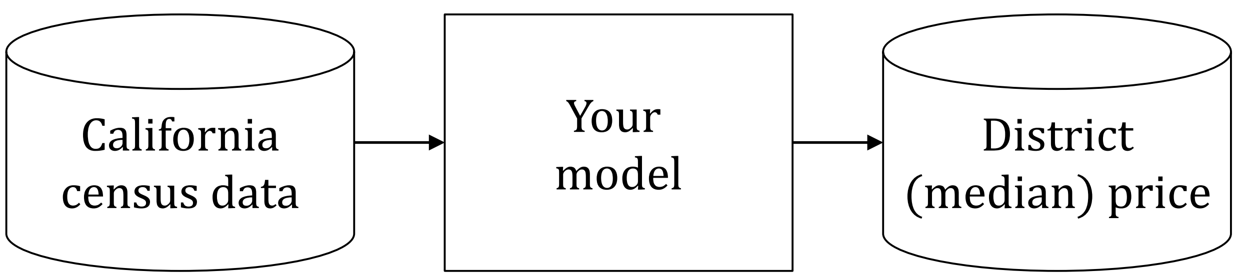
- 1st task
  - Goal: building a **model of housing prices** in California
  - Data: California census data
    - population, median income, …, median housing price for each block group (district)
      - 20,640 districts



	A	B	C	D	E	F	G	H	I	J
1	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
2	-122.23	37.88	41	880	129	322	126	8.3252	452600	NEAR BAY
3	-122.22	37.86	21	7099	1106	2401	1138	8.3014	358500	NEAR BAY
4	-122.24	37.85	52	1467	190	496	177	7.2574	352100	NEAR BAY
5	-122.25	37.85	52	1274	235	558	219	5.6431	341300	NEAR BAY
6	-122.25	37.85	52	1627	280	565	259	3.8462	342200	NEAR BAY
7	-122.25	37.85	52	919	213	413	193	4.0368	269700	NEAR BAY
8	-122.25	37.84	52	2535	489	1094	514	3.6591	299200	NEAR BAY
9	-122.25	37.84	52	3104	687	1157	647	3.12	241400	NEAR BAY

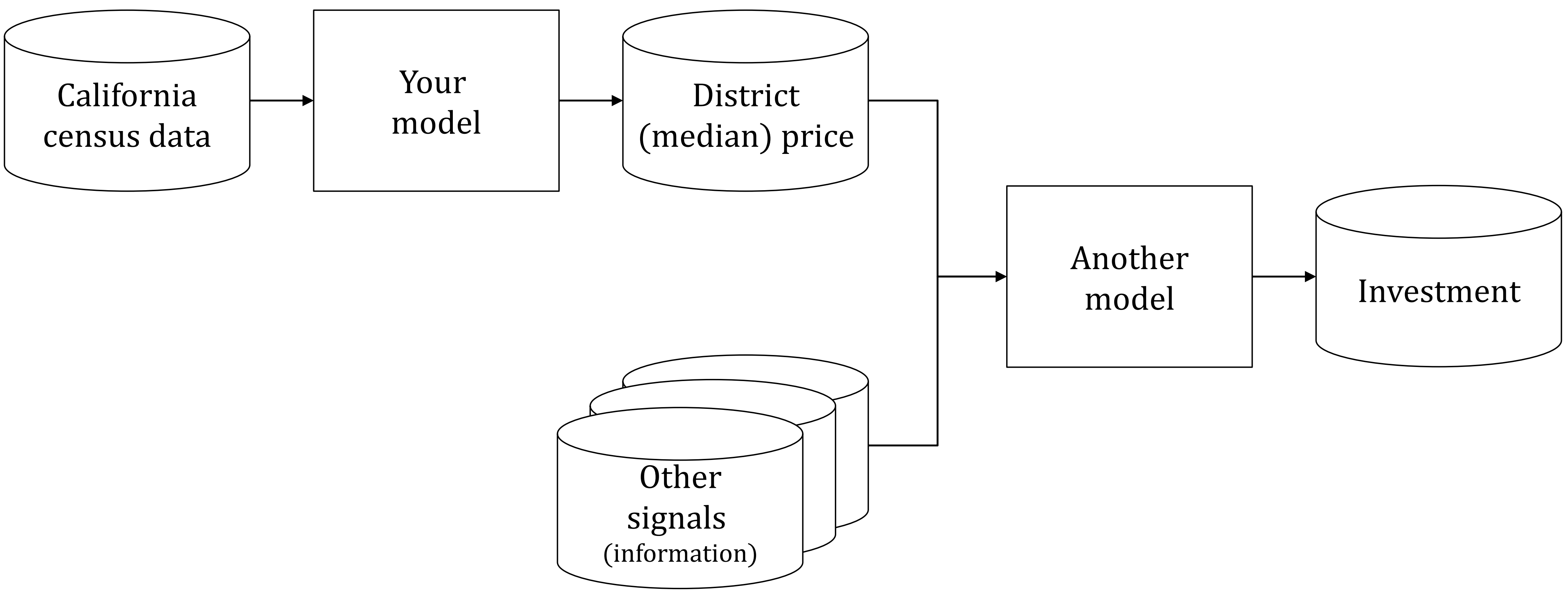
0.1 Frame the Problem

- Make it clear!
  - How does the company expect to use and benefit from your model?



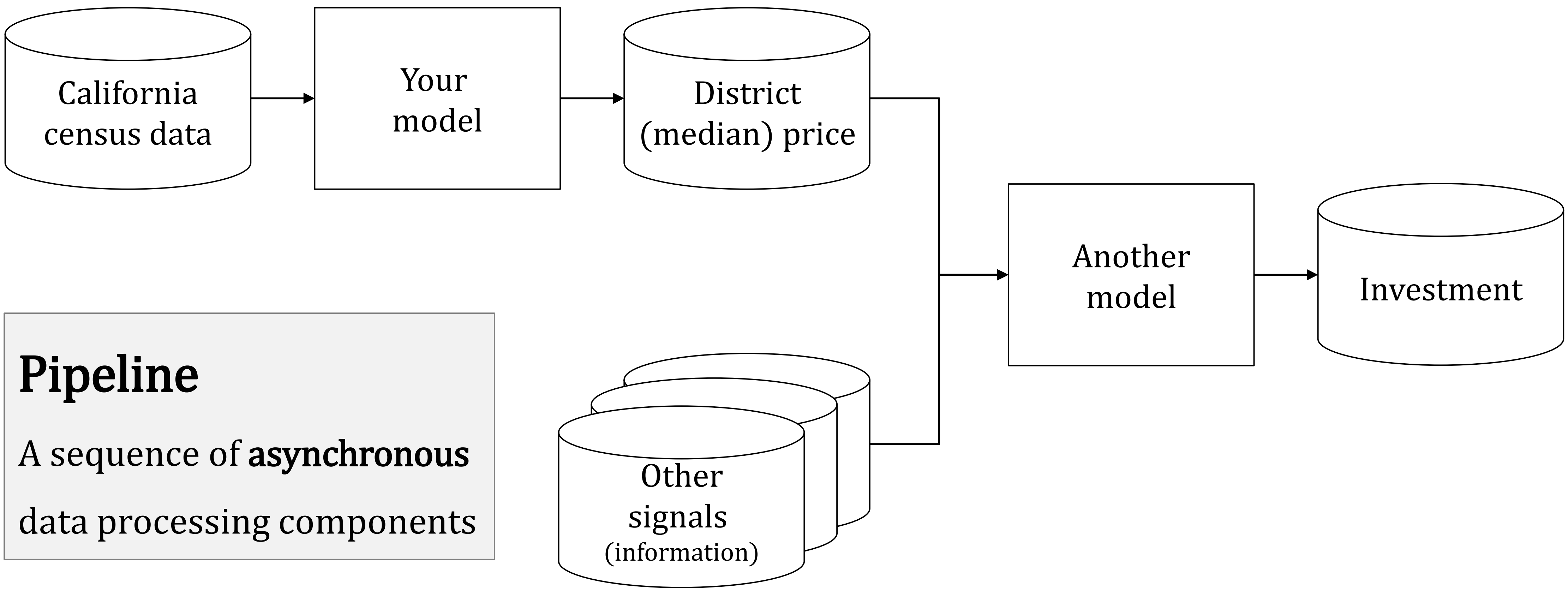
0.1 Frame the Problem

- Make it clear!
  - How does the company expect to use and benefit from your model?



0.1 Frame the Problem

- Make it clear!
  - How does the company expect to use and benefit from your model?



Pipeline

A sequence of **asynchronous** data processing components

0.1 Frame the Problem

Supervised Learning	Unsupervised Learning	Reinforcement Learning

Classification	Regression	Something else

Univariate regression	Multivariate regression

Batch learning	Online learning

0.2 Select a Performance Measure

- Common machine learning notations
  - m: the number of instances in the dataset
  - $\boldsymbol{x}^{(i)}$: a vector of all the feature values (excluding the label) of the i^{th} instance
  - $y^{(i)}$: the label (answer) of the i^{th} instance

0.2 Select a Performance Measure

- Common machine learning notations
  - m: the number of instances in the dataset
  - $\boldsymbol{x}^{(i)}$: a vector of all the feature values (excluding the label) of the i^{th} instance
  - $y^{(i)}$: the label (answer) of the i^{th} instance

$$\boldsymbol{x}^{(1)} = \begin{pmatrix} -122.23 \\ 37.88 \\ 41 \\ 880 \\ \dots \end{pmatrix}$$

$$y^{(1)} = 452{,}600$$

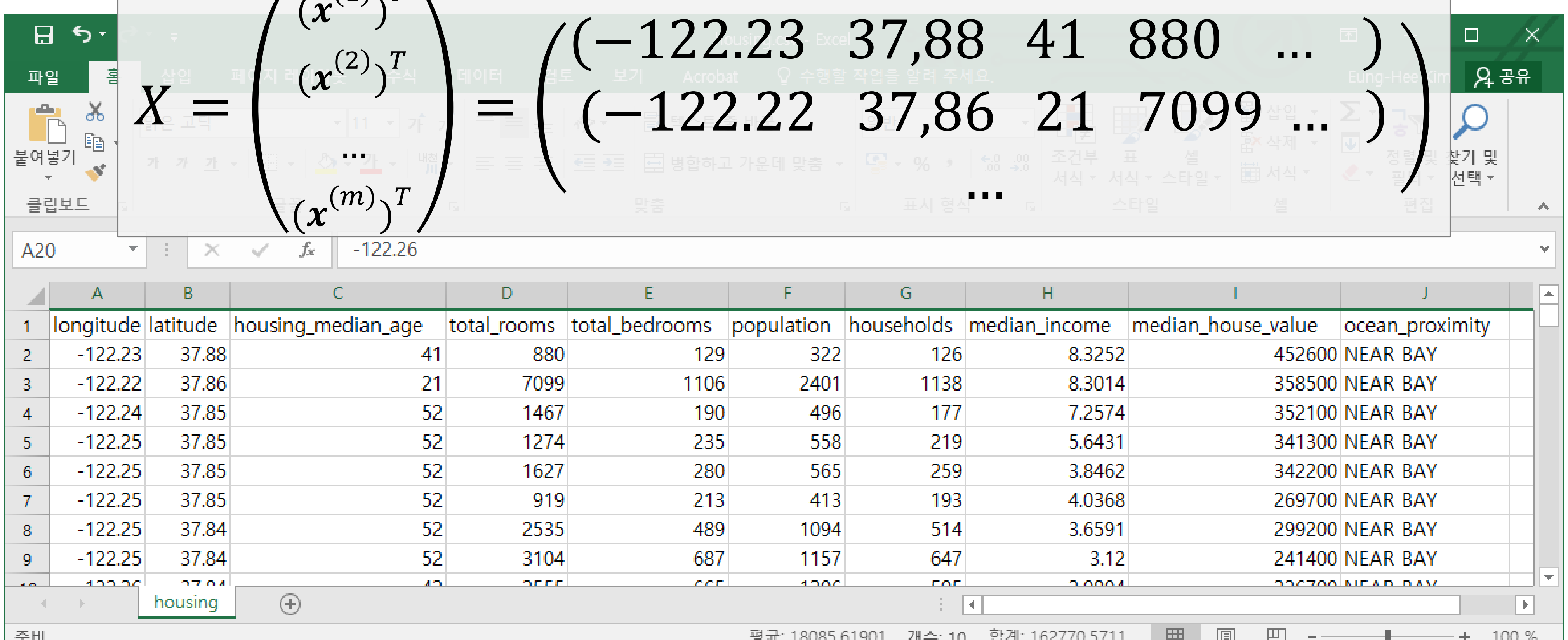
0.2 Select a Performance Measure

- Common machine learning notations
  - m: the number of instances in the dataset
  - $\boldsymbol{x}^{(i)}$: a vector of all the feature values (excluding the label) of the i^{th} instance
  - $y^{(i)}$: the label (answer) of the i^{th} instance
  - $\boldsymbol{X}$: a matrix containing all the feature values (excluding labels) of all instances
    - i^{th} row of $\boldsymbol{X}$ is the transpose of $\boldsymbol{x}^{(i)}$, denoted $(\boldsymbol{x}^{(i)})^T$

0.2 Select a Performance Measure

- Common machine learning notations
  - m: the number of instances in the dataset
  - $\boldsymbol{x}^{(i)}$: a vector of all the feature values (excluding the label) of the i^{th} instance
  - $y^{(i)}$: the label (answer) of the i^{th} instance
  - $\boldsymbol{X}$: a matrix containing all the feature values (excluding labels) of all instances
    - i^{th} row of $\boldsymbol{X}$ is the transpose of $\boldsymbol{x}^{(i)}$, denoted $(\boldsymbol{x}^{(i)})^T$

$$X = \begin{pmatrix} (\boldsymbol{x}^{(1)})^T \\ (\boldsymbol{x}^{(2)})^T \\ \cdots \\ (\boldsymbol{x}^{(m)})^T \end{pmatrix} = \begin{pmatrix} (-122.23 \quad 37{,}88 \quad 41 \quad 880 \quad \ldots) \\ (-122.22 \quad 37{,}86 \quad 21 \quad 7099 \quad \ldots) \\ \ldots \end{pmatrix}$$



	A	B	C	D	E	F	G	H	I	J
1	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
2	-122.23	37.88	41	880	129	322	126	8.3252	452600	NEAR BAY
3	-122.22	37.86	21	7099	1106	2401	1138	8.3014	358500	NEAR BAY
4	-122.24	37.85	52	1467	190	496	177	7.2574	352100	NEAR BAY
5	-122.25	37.85	52	1274	235	558	219	5.6431	341300	NEAR BAY
6	-122.25	37.85	52	1627	280	565	259	3.8462	342200	NEAR BAY
7	-122.25	37.85	52	919	213	413	193	4.0368	269700	NEAR BAY
8	-122.25	37.84	52	2535	489	1094	514	3.6591	299200	NEAR BAY
9	-122.25	37.84	52	3104	687	1157	647	3.12	241400	NEAR BAY

0.2 Select a Performance Measure

- Common machine learning notations
  - m: the number of instances in the dataset
  - $\boldsymbol{x}^{(i)}$: a vector of all the feature values (excluding the label) of the i^{th} instance
  - $y^{(i)}$: the label (answer) of the i^{th} instance
  - $\boldsymbol{X}$: a matrix containing all the feature values (excluding labels) of all instances
    - i^{th} row of $\boldsymbol{X}$ is the transpose of $\boldsymbol{x}^{(i)}$, denoted $(\boldsymbol{x}^{(i)})^T$
  - h: your hypothesis (prediction function/model)
    - $h\left(\boldsymbol{x}^{(i)}\right) = \hat{y}^{(i)}$: the predicted value by h for the i^{th} instance
    - $\hat{y}^{(i)} - y^{(i)}$: the prediction error

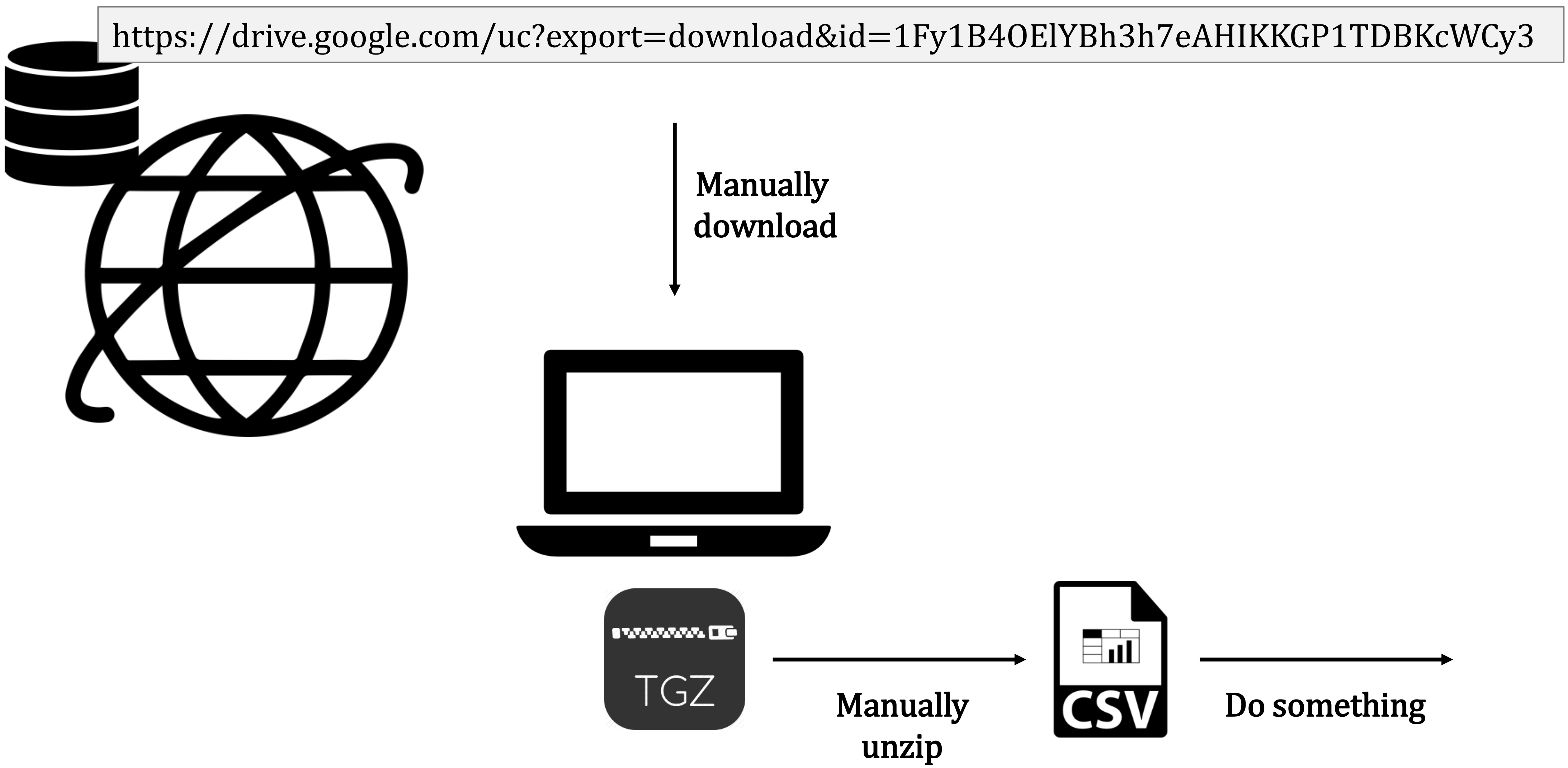
0.2 Select a Performance Measure

- Common machine learning notations
  - m: the number of instances in the dataset
  - $\boldsymbol{x}^{(i)}$: a vector of all the feature values (excluding the label) of the i^{th} instance
  - $y^{(i)}$: the label (answer) of the i^{th} instance
  - $\boldsymbol{X}$: a matrix containing all the feature values (excluding labels) of all instances
    - i^{th} row of $\boldsymbol{X}$ is the transpose of $\boldsymbol{x}^{(i)}$, denoted $(\boldsymbol{x}^{(i)})^T$
  - h: your hypothesis (prediction function/model)
    - $h\left(\boldsymbol{x}^{(i)}\right) = \hat{y}^{(i)}$: the predicted value by h for the i^{th} instance
    - $\hat{y}^{(i)} - y^{(i)}$: the prediction error
  - $\text{RMSE}(\boldsymbol{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(\boldsymbol{x}^{(i)}) - y^{(i)})^2}$
    - Root Mean Square Error
  - $\text{MAE}(\boldsymbol{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right|$
    - Mean Absolute Error

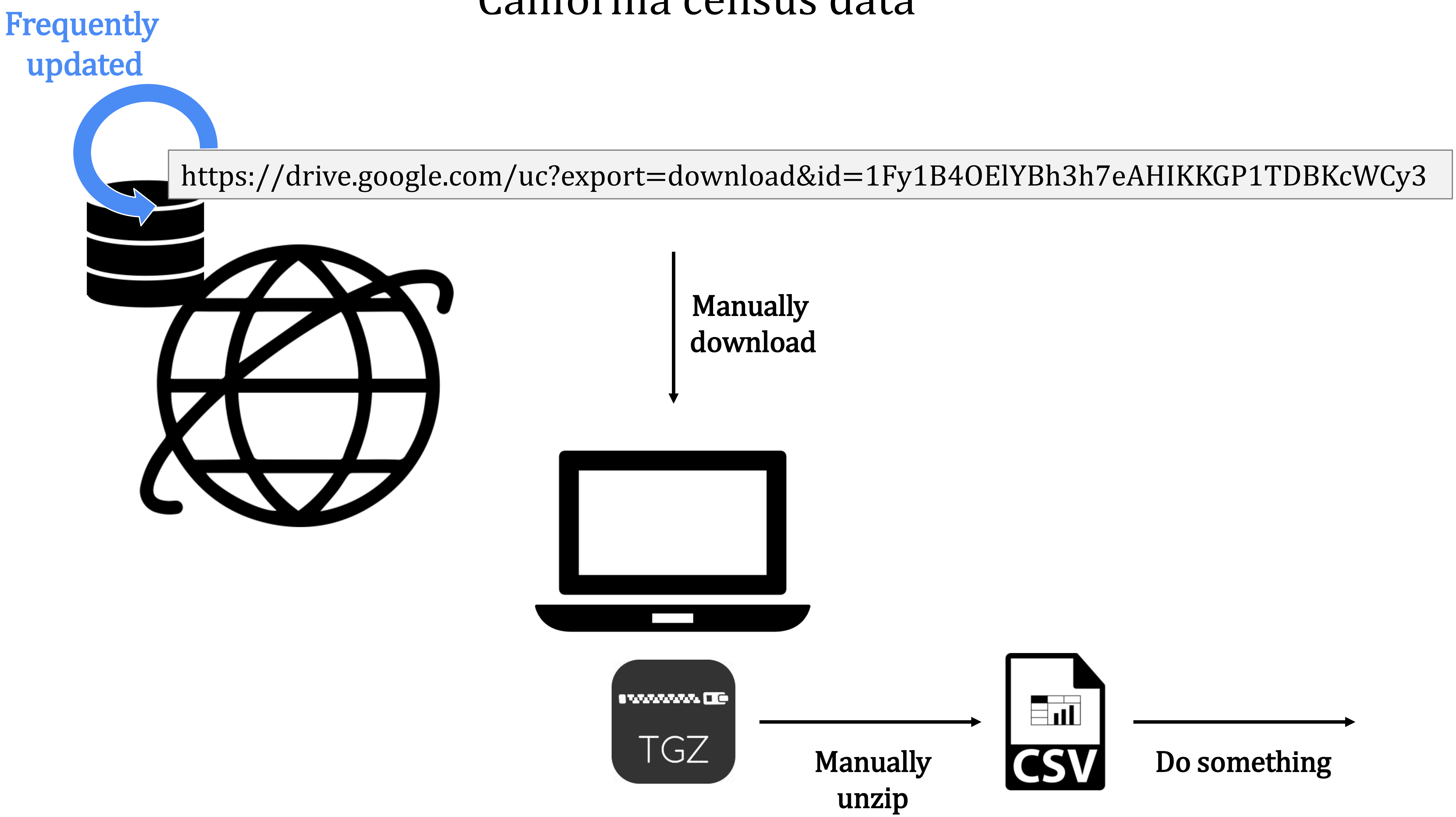
End-to-End Machine Learning Project

No.	Action	Package/library
0	Look at the big picture	–
1	Get the data	tarfile, urllib, pandas
2	Discover and visualize the data to gain insights	pandas, matplotlib
3	Prepare the data for Machine Learning algorithms	pandas, scikit-learn, numpy
4	Select a model and train it	
5	Fine-tune your model	
6	Present your solution	
7	Launch, monitor and maintain your system	joblib, flask

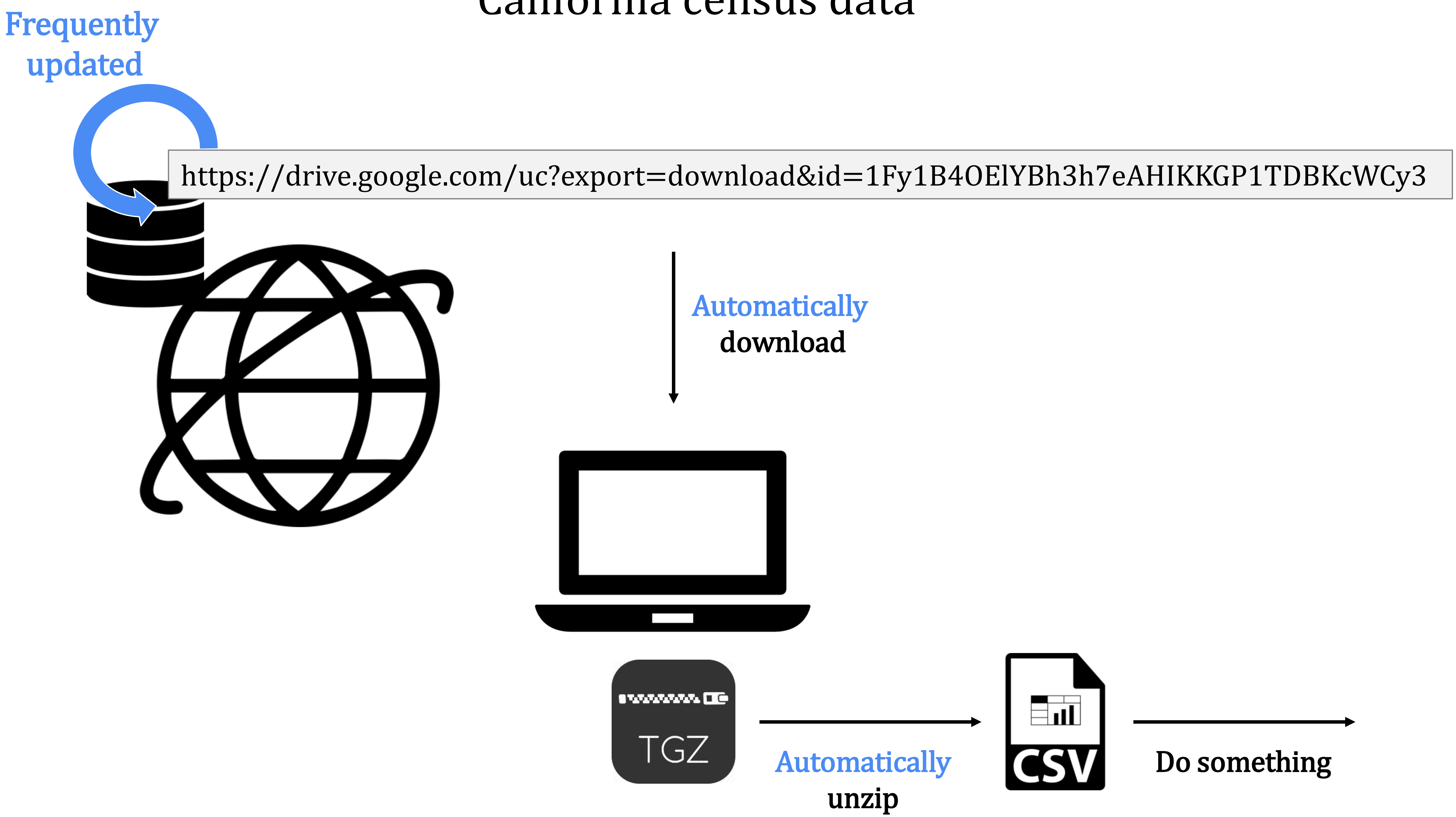
California census data



California census data



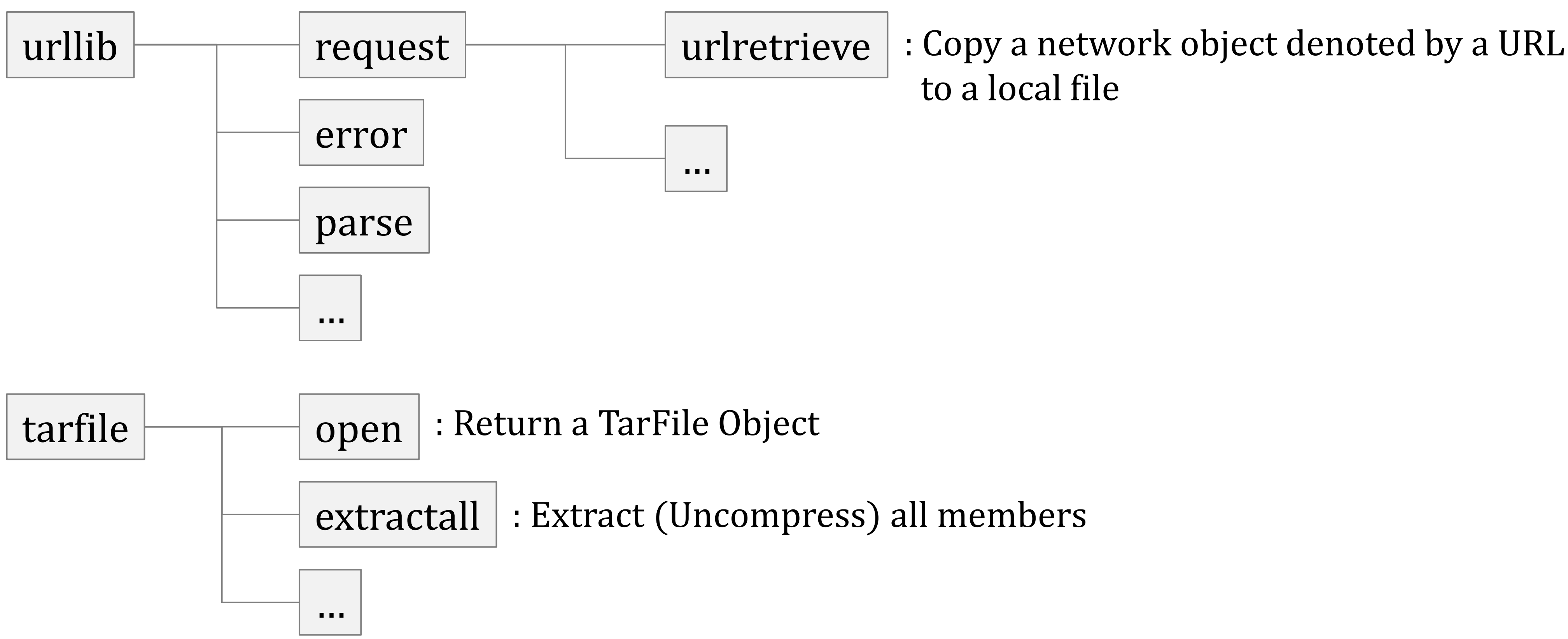
California census data



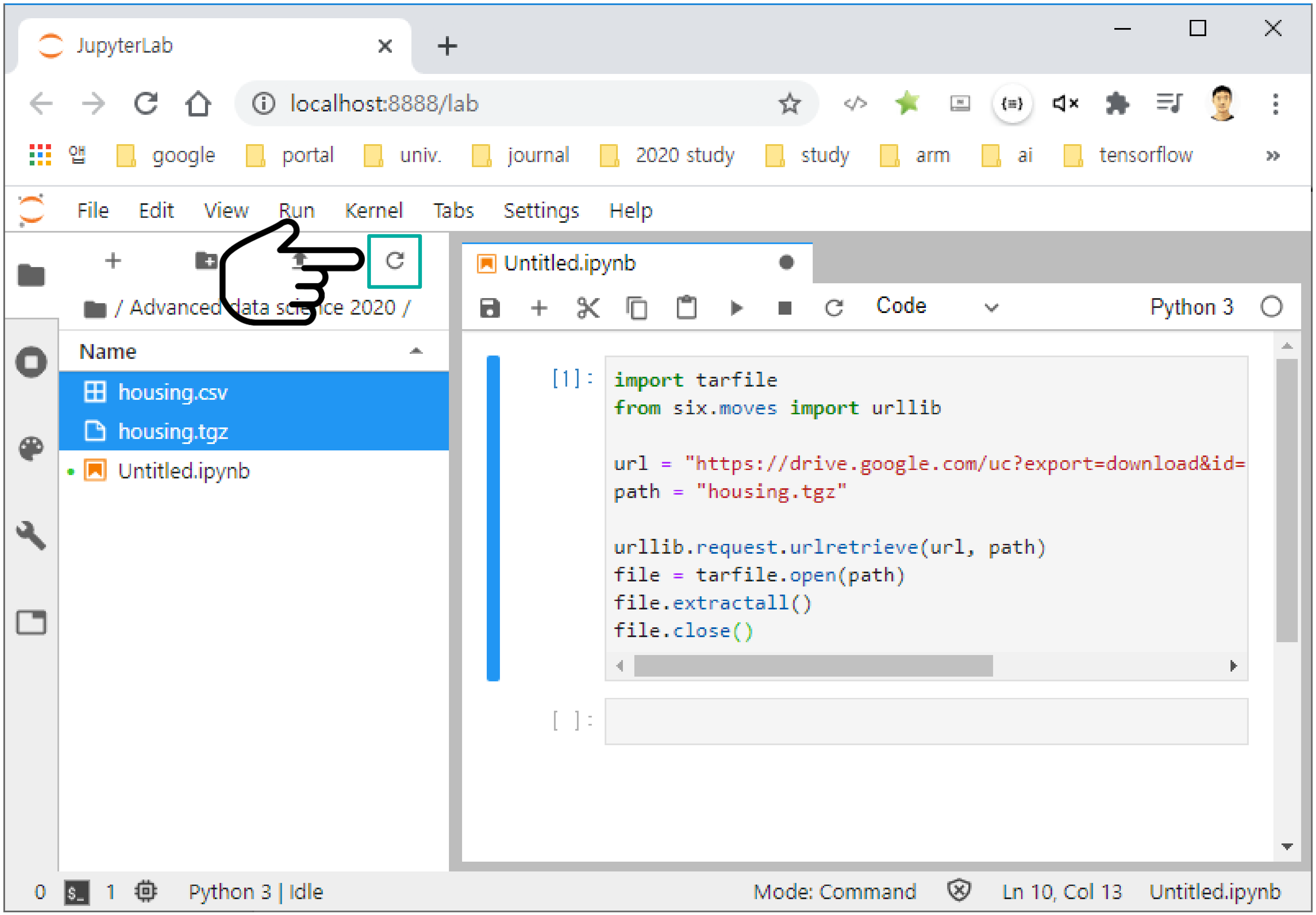
1.1 Download and uncompress data

[1]:

```python
import tarfile
from six.moves import urllib

url = "https://drive.google.com/uc?export=download&id=1Fy1B4OElYBh3h7eAHIKKGP1TDBKcWCy3"
path = "housing.tgz"

urllib.request.urlretrieve(url, path)
file = tarfile.open(path)
file.extractall()
file.close()
```



1.1 Download and uncompress data



1.2 Read data

```
import tarfile
from six.moves import urllib

url = "https://drive.google.com/uc?export=download&id=1Fy1B4OE
path = "housing.tgz"

urllib.request.urlretrieve(url, path)
file = tarfile.open(path)
file.extractall()
file.close()
```

```
import pandas as pd

data = pd.read_csv("housing.csv")
data.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
0	-122.23	37.88	41.0	880.0	129.0
1	-122.22	37.86	21.0	7099.0	1106.0
2	-122.24	37.85	52.0	1467.0	190.0
3	-122.25	37.85	52.0	1274.0	235.0
4	-122.25	37.85	52.0	1627.0	280.0

1.3 Take a Quick Look

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

1.3 Take a Quick Look

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Missing values

Categorical values

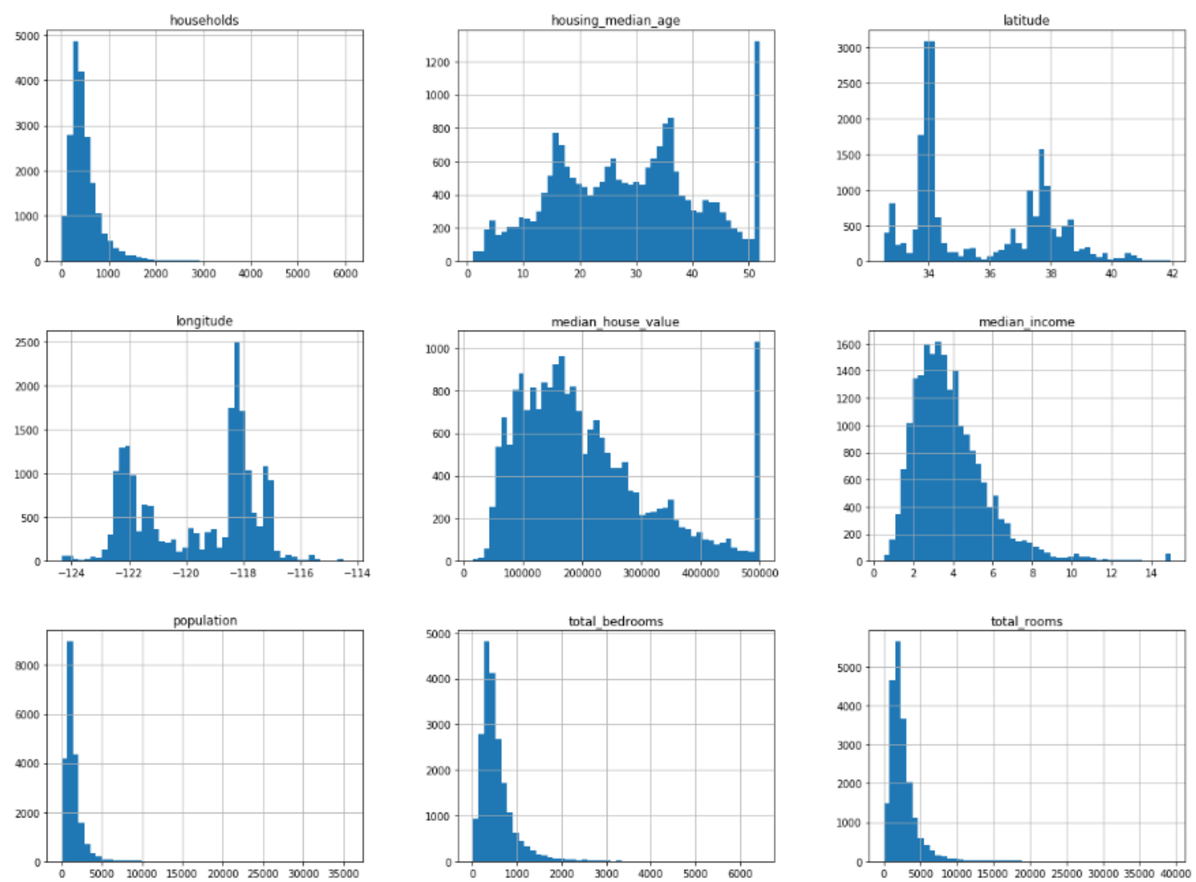
1.3 Take a Quick Look

```
data["ocean_proximity"].value_counts()
```

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

1.3 Take a Quick Look

```
import matplotlib.pyplot as plt

data.hist(bins=50, figsize=(20, 15))
plt.show()
```



Right shewed

Difference scales

Preprocessed values

End-to-End Machine Learning Project

No.	Action	Package/library
0	Look at the big picture	–
1	Get the data	tarfile, urllib, pandas
2	Discover and visualize the data to gain insights	pandas, matplotlib
3	Prepare the data for Machine Learning algorithms	pandas, scikit-learn, numpy
4	Select a model and train it	
5	Fine-tune your model	
6	Present your solution	
7	Launch, monitor and maintain your system	joblib, flask

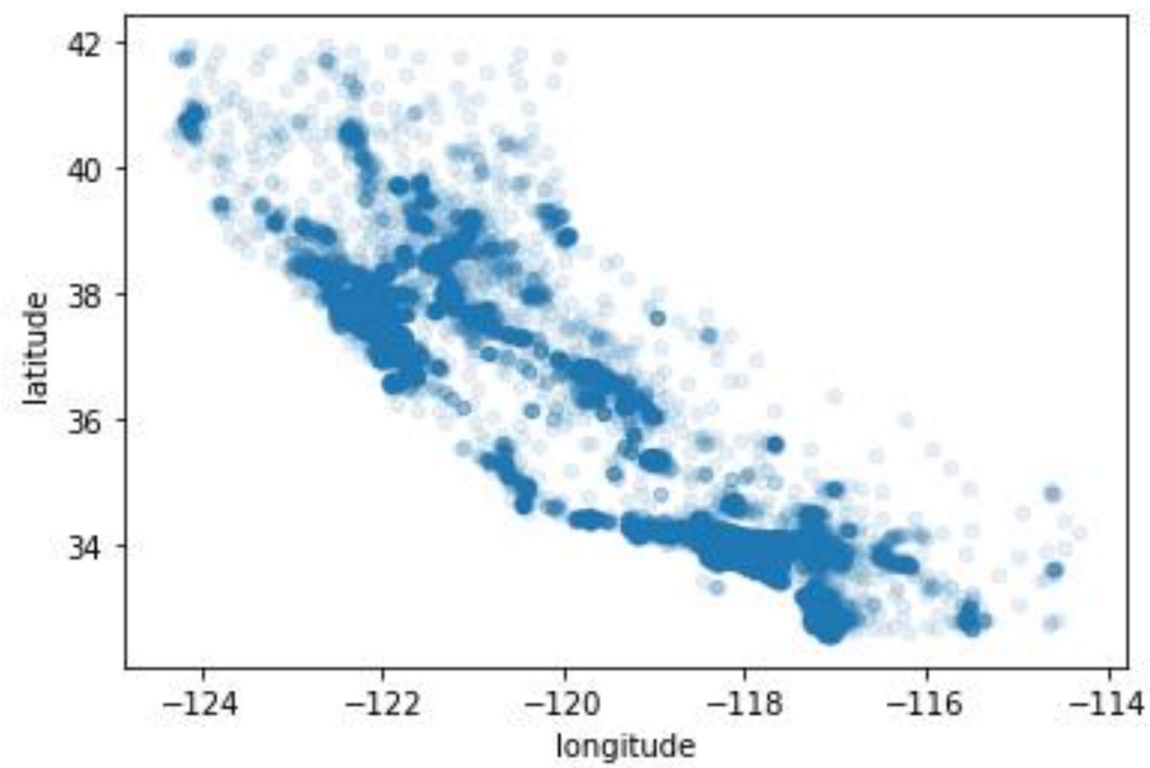
2.1 Visualize Geographical Data

data
(DataFrame)

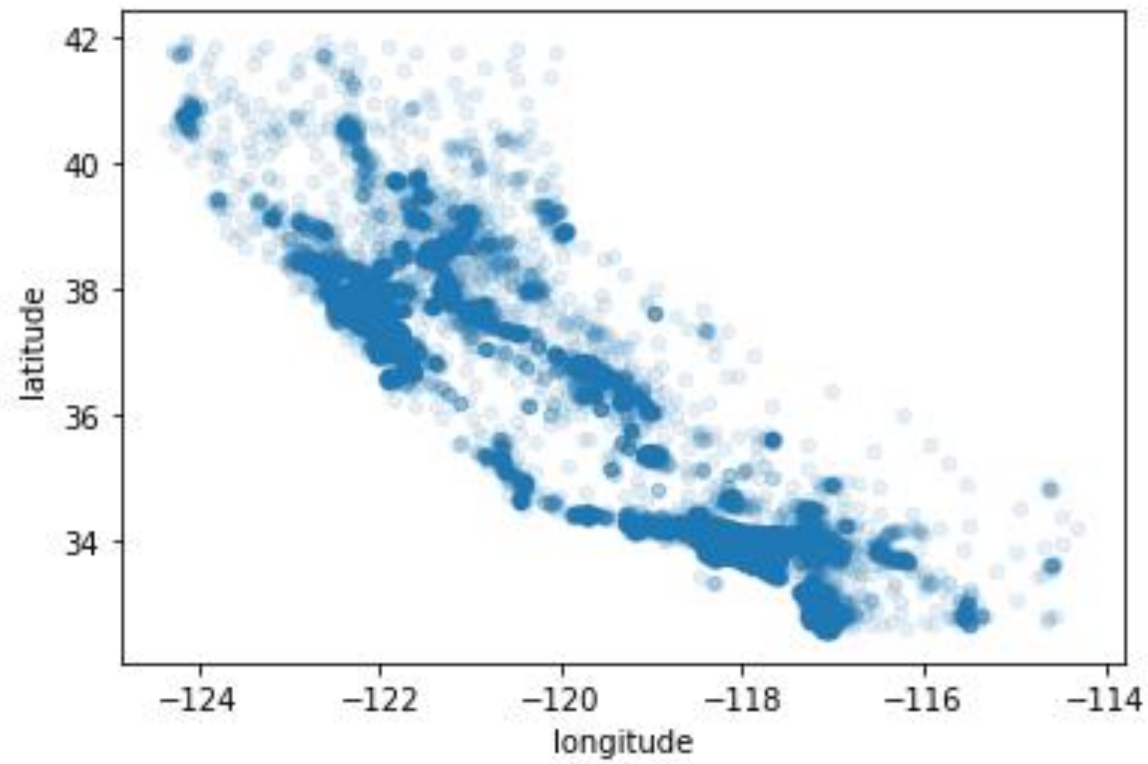
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
0	-122.23	37.88	41.0	880.0	129.0
1	-122.22	37.86	21.0	7099.0	1106.0
2	-122.24	37.85	52.0	1467.0	190.0
3	-122.25	37.85	52.0	1274.0	235.0
4	-122.25	37.85	52.0	1627.0	280.0

```
data.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

2.1 Visualize Geographical Data



2.1 Visualize Geographical Data



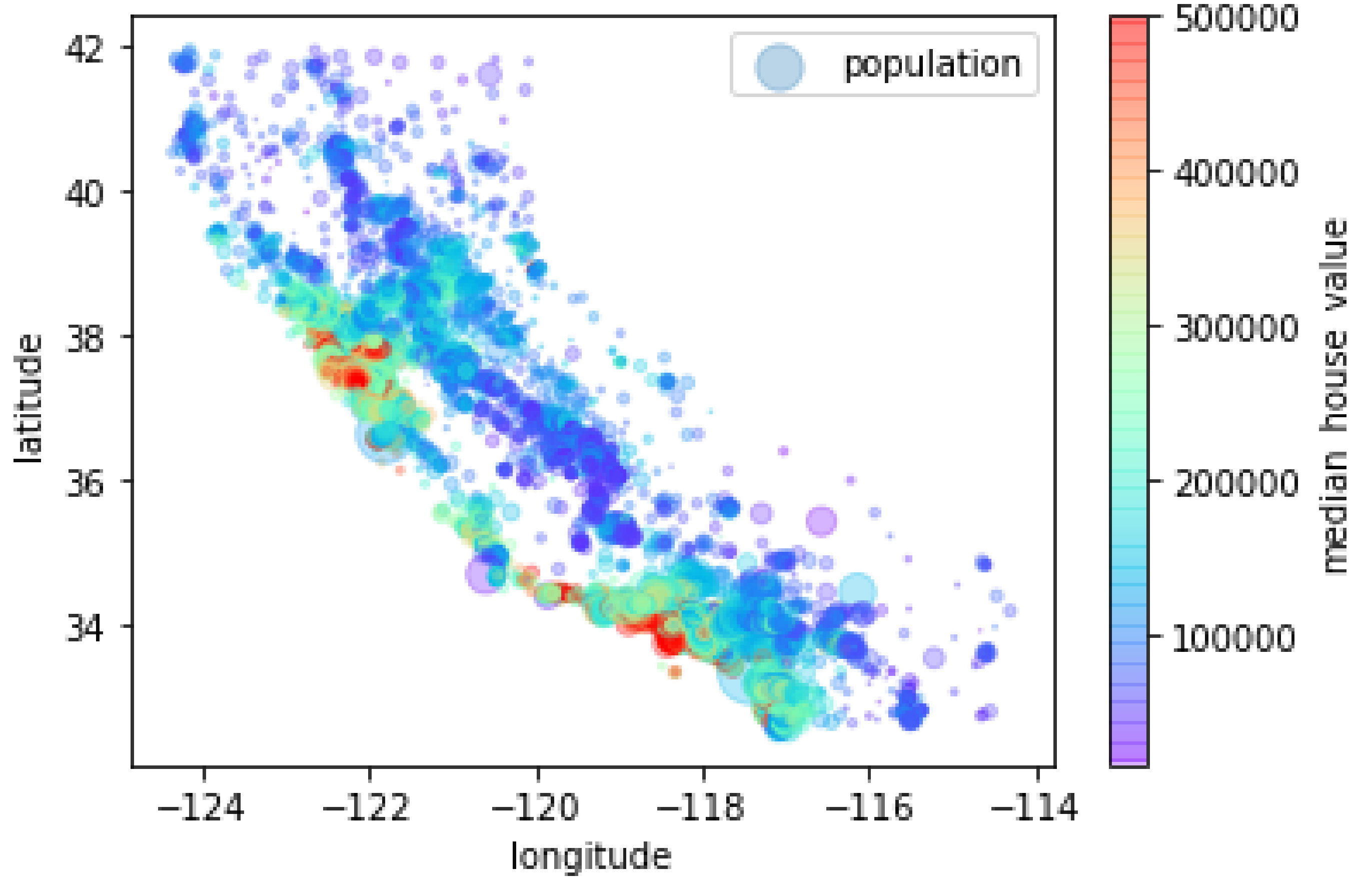
Size
population

Color
median house value

2.1 Visualize Geographical Data

```
data.plot(kind="scatter", x="longitude", y="latitude", alpha=0.3,
          s=data["population"]/100, label="population",
          c="median_house_value", cmap=plt.get_cmap("rainbow"),
          colorbar=True, sharex=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2976a0c31c8>
```



2.2 Look for Correlation

```
corr_matrix = data.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

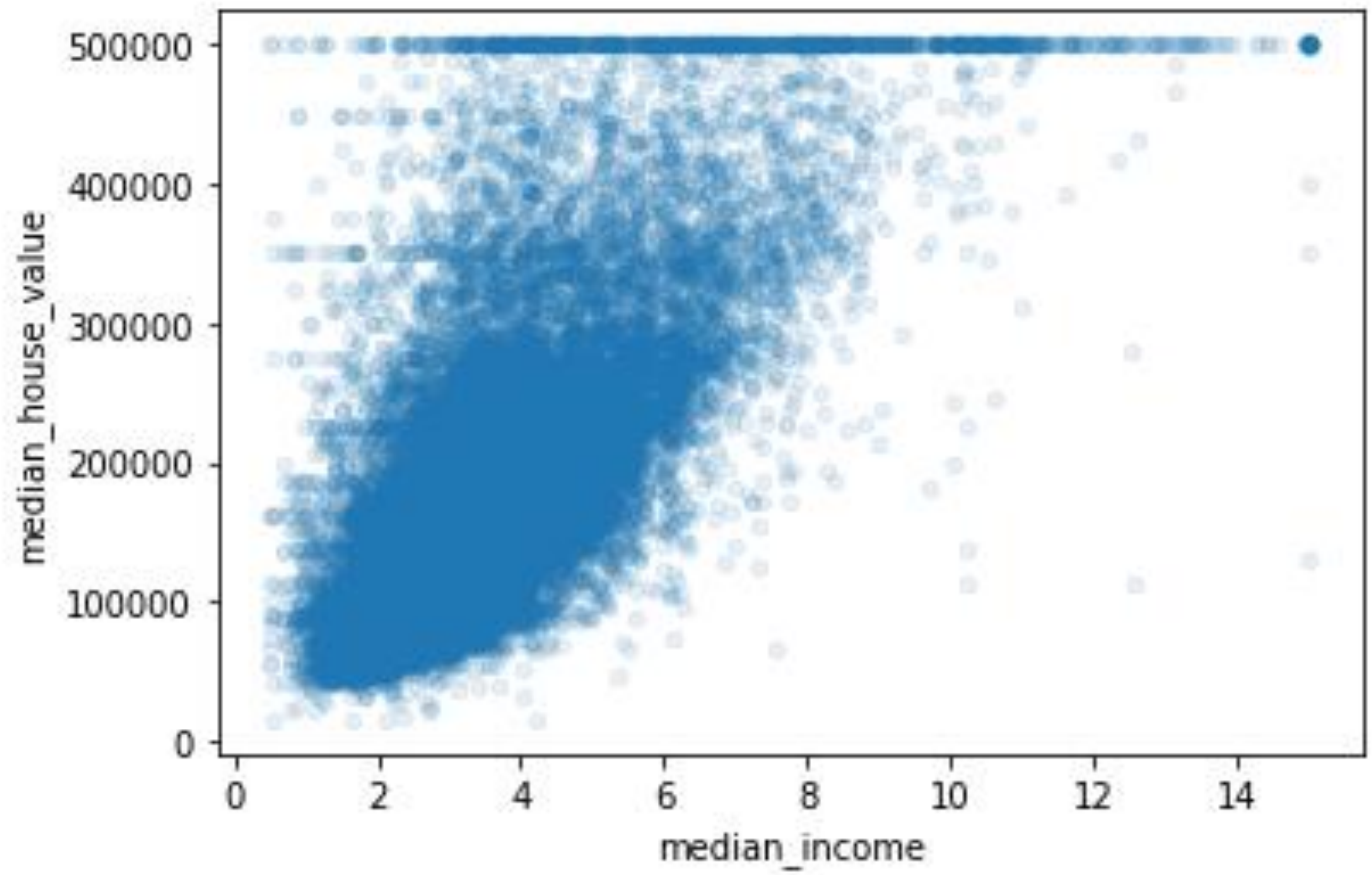
```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

- **Correlation**: how tightly the instances are clustered about a straight line.
  – Range: -1 and $+1$

2.2 Look for Correlation

```
data.plot(kind="scatter",
          x="median_income",
          y="median_house_value",
          alpha=0.1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fe07479470>
```



2.3 Feature extraction
(by manual combinations)

Attribute (Feature)
longitude
latitude
housing_median_age
total_rooms
total_bedrooms
population
households
median_income
median_house_value
ocean_proximity

2.3 Feature extraction
(by manual combinations)

Attribute (Feature)
longitude
latitude
housing_median_age
total_rooms
total_bedrooms
population
households
median_income
median_house_value
ocean_proximity
rooms_per_household= $\frac{total\_bedrooms}{households}$
bedrooms_per_room== $\frac{total\_bedrooms}{total\_rooms}$

2.3 Feature extraction
(by manual combinations)

Original data		
...	...	...

+

Extended data	
rooms_per_household	bedrooms_per_room

```
data["rooms_per_household"] = data["total_rooms"]/data["households"]
data["bedrooms_per_room"] = data["total_bedrooms"]/data["total_rooms"]
data.head()
```

ne	median_house_value	ocean_proximity	rooms_per_household	bedrooms_per_room
52	452600.0	NEAR BAY	6.984127	0.146591
14	358500.0	NEAR BAY	6.238137	0.155797
74	352100.0	NEAR BAY	8.288136	0.129516
31	341300.0	NEAR BAY	5.817352	0.184458
62	342200.0	NEAR BAY	6.281853	0.172096

2.3 Feature extraction
(by manual combinations)

```
corr_matrix = data.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
median_house_value      1.000000
median_income           0.688075
rooms_per_household     0.151948
total_rooms             0.134153
housing_median_age      0.105623
households              0.065843
total_bedrooms          0.049686
population             -0.024650
longitude              -0.045967
latitude               -0.144160
bedrooms_per_room      -0.255880
Name: median_house_value, dtype: float64
```

End-to-End Machine Learning Project

No.	Action	Package/library
0	Look at the big picture	–
1	Get the data	tarfile, urllib, pandas
2	Discover and visualize the data to gain insights	pandas, matplotlib
3	Prepare the data for Machine Learning algorithms	pandas, scikit-learn
4	Select a model and train it	
5	Fine-tune your model	
6	Present your solution	
7	Launch, monitor and maintain your system	joblib, flask

3.1 Handling Categorical Attributes (Feature values)

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
 10  rooms_per_household 20640 non-null  float64
 11  bedrooms_per_room   20433 non-null  float64
dtypes: float64(11), object(1)
memory usage: 1.9+ MB
```

Categorical values

Transformation #1

ocean_proximity
<1H OCEAN
NEAR OCEAN
INLAND
NEAR BAY
ISLAND

?

ocean_proximity
0
1
2
3
4

Transformation #2

ocean_proximity
<1H OCEAN
NEAR OCEAN
INLAND
NEAR BAY
ISLAND

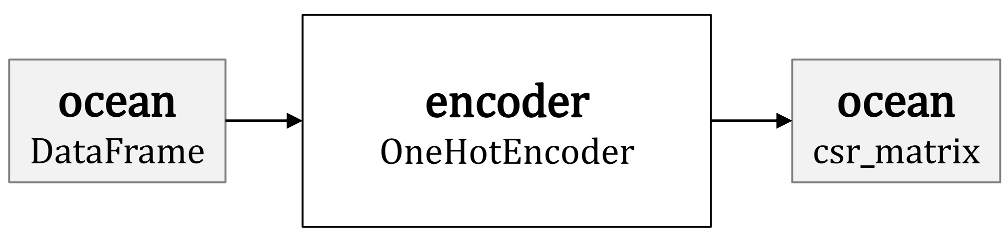
	ocean_proximity				
<1H OCEAN	1	0	0	0	0
NEAR OCEAN	0	1	0	0	0
INLAND	0	0	1	0	0
NEAR BAY	0	0	0	1	0
ISLAND	0	0	0	0	1

One hot encoding

3.1 Handling Categorical Attributes (Feature values)

```python
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()
ocean = data[["ocean_proximity"]]
ocean = encoder.fit_transform(ocean)
```

	ocean_proximity
0	NEAR BAY
1	NEAR BAY
2	NEAR BAY
3	NEAR BAY
4	NEAR BAY
...	...
20635	INLAND
20636	INLAND
20637	INLAND
20638	INLAND
20639	INLAND



0	0	0	1	0
0	0	0	1	0
0	0	0	1	0
0	0	0	1	0
0	0	0	1	0
		...		
0	1	0	0	0
0	1	0	0	0
0	1	0	0	0
0	1	0	0	0
0	1	0	0	0

3.1 Handling Categorical Attributes (Feature values)

```
print(ocean.toarray())
print(encoder.categories_)
```

```
[[0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 ...
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]]
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
      dtype=object)]
```

3.1 Handling Categorical Attributes (Feature values)

encoder
OneHotEncoder

<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN

+

ocean
csr_matrix

0	0	0	1	0
0	0	0	1	0
...				
0	1	0	0	0
0	1	0	0	0

ocean
DataFrame

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0	0.0
...	...	...	...	...	...
20635	0.0	1.0	0.0	0.0	0.0
20636	0.0	1.0	0.0	0.0	0.0
20637	0.0	1.0	0.0	0.0	0.0
20638	0.0	1.0	0.0	0.0	0.0
20639	0.0	1.0	0.0	0.0	0.0

3.1 Handling Categorical Attributes (Feature values)

```
ocean = pd.DataFrame(columns=encoder.categories_[0],
                     data=ocean.toarray())
ocean
```

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0	0.0
...	...	...	...	...	...
20635	0.0	1.0	0.0	0.0	0.0
20636	0.0	1.0	0.0	0.0	0.0
20637	0.0	1.0	0.0	0.0	0.0
20638	0.0	1.0	0.0	0.0	0.0
20639	0.0	1.0	0.0	0.0	0.0

3.1 Handling Categorical Attributes (Feature values)

data
DataFrame

	...	median_ house_value	ocean_ proximity	...
0	...	...	...	...
1	...	...	...	...
2	...	...	...	...
3	...	...	...	...
...	...	...	...	...

ocean
DataFrame

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	...	...	...	...	...
1	...	...	...	...	...
2	...	...	...	...	...
3	...	...	...	...	...
...	...	...	...	...	...

3.1 Handling Categorical Attributes (Feature values)

data
DataFrame

	...	median_ house_value	ocean_ proximity	...
0	...	...	...	...
1	...	...	...	...
2	...	...	...	...
3	...	...	...	...
...	...	...	...	...

ocean
DataFrame

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	...	...	...	...	...
1	...	...	...	...	...
2	...	...	...	...	...
3	...	...	...	...	...
...	...	...	...	...	...

data
DataFrame

	...	...
0	...	...
1	...	...
2	...	...
3	...	...
...	...	...

3.1 Handling Categorical Attributes (Feature values)

data
DataFrame

	...	median_ house_value	ocean_ proximity	...
0	...	...	...	...
1	...	...	...	...
2	...	...	...	...
3	...	...	...	...
...	...	...	...	...

ocean
DataFrame

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	...	...	...	...	...
1	...	...	...	...	...
2	...	...	...	...	...
3	...	...	...	...	...
...	...	...	...	...	...

data
DataFrame

	...	...	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	...	...	...	...	...	...	...
1	...	...	...	...	...	...	...
2	...	...	...	...	...	...	...
3	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

3.1 Handling Categorical Attributes (Feature values)

data
DataFrame

	...	median_ house_value	ocean_ proximity	...
0	...	...	...	...
1	...	...	...	...
2	...	...	...	...
3	...	...	...	...
...	...	...	...	...

ocean
DataFrame

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	...	...	...	...	...
1	...	...	...	...	...
2	...	...	...	...	...
3	...	...	...	...	...
...	...	...	...	...	...

	...	...	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN	median_ house_value
0	...	...	...	...	...	...	...	...
1	...	...	...	...	...	...	...	...
2	...	...	...	...	...	...	...	...
3	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

data
DataFrame

3.1 Handling Categorical Attributes (Feature values)

data
DataFrame

	...	median_ house_value	ocean_ proximity	...
0	...	...	...	...
1	...	...	...	...
2	...	...	...	...
3	...	...	...	...
...	...	...	...	...

ocean
DataFrame

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	...	...	...	...	...
1	...	...	...	...	...
2	...	...	...	...	...
3	...	...	...	...	...
...	...	...	...	...	...

	...	...	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN	median_ house_value
0	...	...	...	...	...	...	...	...
1	...	...	...	...	...	...	...	...
2	...	...	...	...	...	...	...	...
3	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

data
DataFrame

3.1 Handling Categorical Attributes (Feature values)

data
DataFrame

	...	median_ house_value	ocean_ proximity	...
0	...	...	...	...
1	...	...	...	...
2	...	...	...	...
3	...	...	...	...
...	...	...	...	...

ocean
DataFrame

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	...	...	...	...	...
1	...	...	...	...	...
2	...	...	...	...	...
3	...	...	...	...	...
...	...	...	...	...	...

Slice → Drop → Drop → Concatenate → Concatenate

	...	...	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN	median_ house_value
0	...	...	...	...	...	...	...	...
1	...	...	...	...	...	...	...	...
2	...	...	...	...	...	...	...	...
3	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

data
DataFrame

Slice → Drop → Drop → Concatenate → Concatenate

```
value = data[["median_house_value"]]
value
```

	median_house_value
0	452600.0
1	358500.0
2	352100.0
3	341300.0
4	342200.0
...	...
20635	78100.0
20636	77100.0
20637	92300.0
20638	84700.0
20639	89400.0

Slice → Drop → Drop → Concatenate → Concatenate

```
data.drop(["median_house_value", "ocean_proximity"], axis=1, inplace=True)
data
```

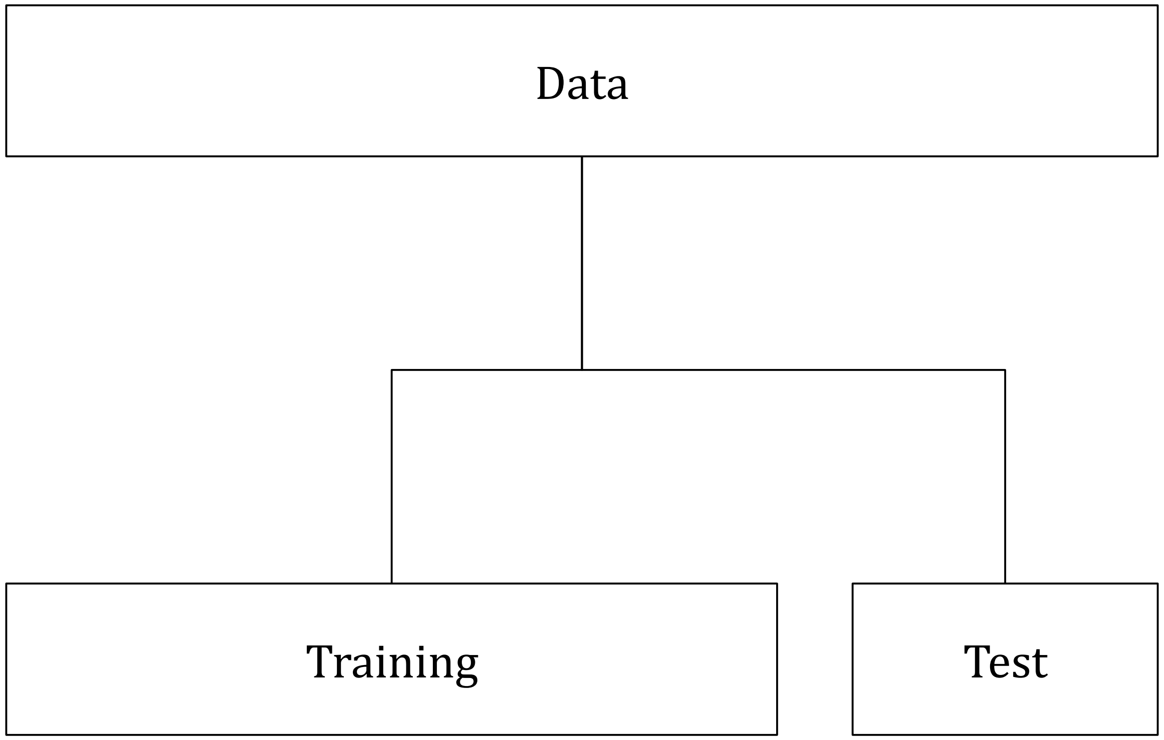
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0
...	...	...	...	...	...	...
20635	-121.09	39.48	25.0	1665.0	374.0	845.0
20636	-121.21	39.49	18.0	697.0	150.0	356.0
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0
20638	-121.32	39.43	18.0	1860.0	409.0	741.0
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0

Slice → Drop → Drop → Concatenate → Concatenate

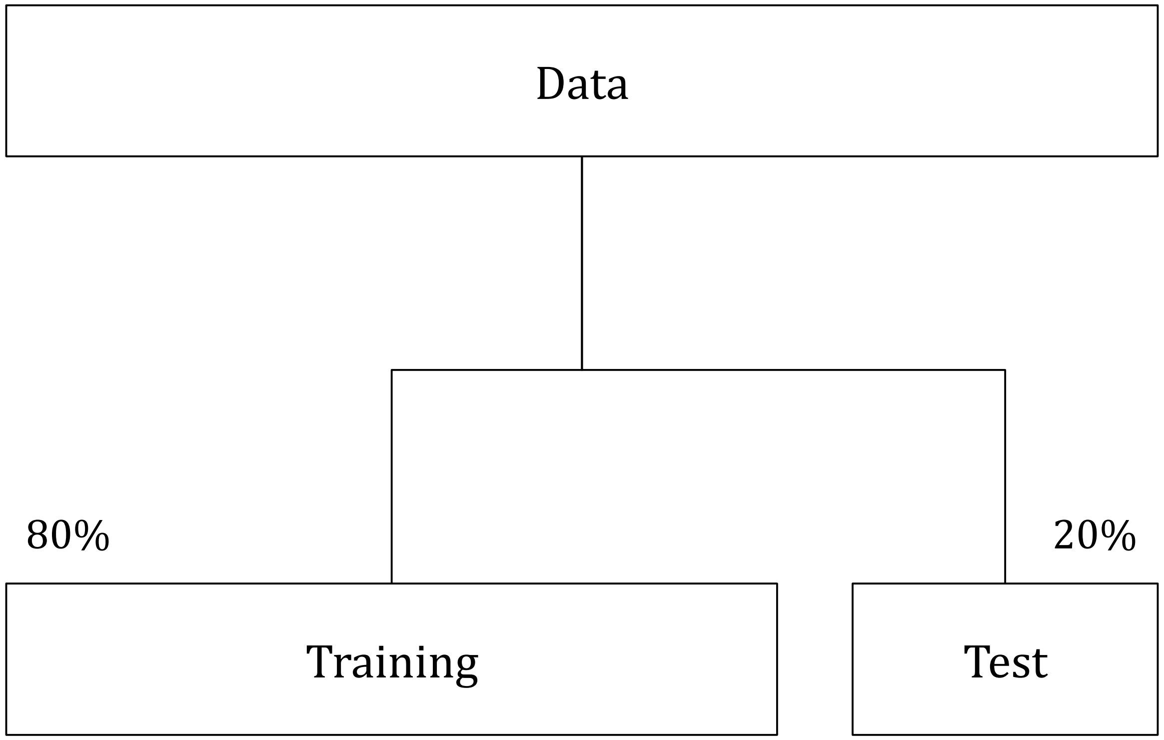
```
data = pd.concat([data, ocean, value], axis=1)
data
```

bedrooms_per_room	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN	median_house_value
0.146591	0.0	0.0	0.0	1.0	0.0	452600.0
0.155797	0.0	0.0	0.0	1.0	0.0	358500.0
0.129516	0.0	0.0	0.0	1.0	0.0	352100.0
0.184458	0.0	0.0	0.0	1.0	0.0	341300.0
0.172096	0.0	0.0	0.0	1.0	0.0	342200.0
...	...	...	...	...	...	...
0.224625	0.0	1.0	0.0	0.0	0.0	78100.0
0.215208	0.0	1.0	0.0	0.0	0.0	77100.0
0.215173	0.0	1.0	0.0	0.0	0.0	92300.0
0.219892	0.0	1.0	0.0	0.0	0.0	84700.0
0.221185	0.0	1.0	0.0	0.0	0.0	89400.0

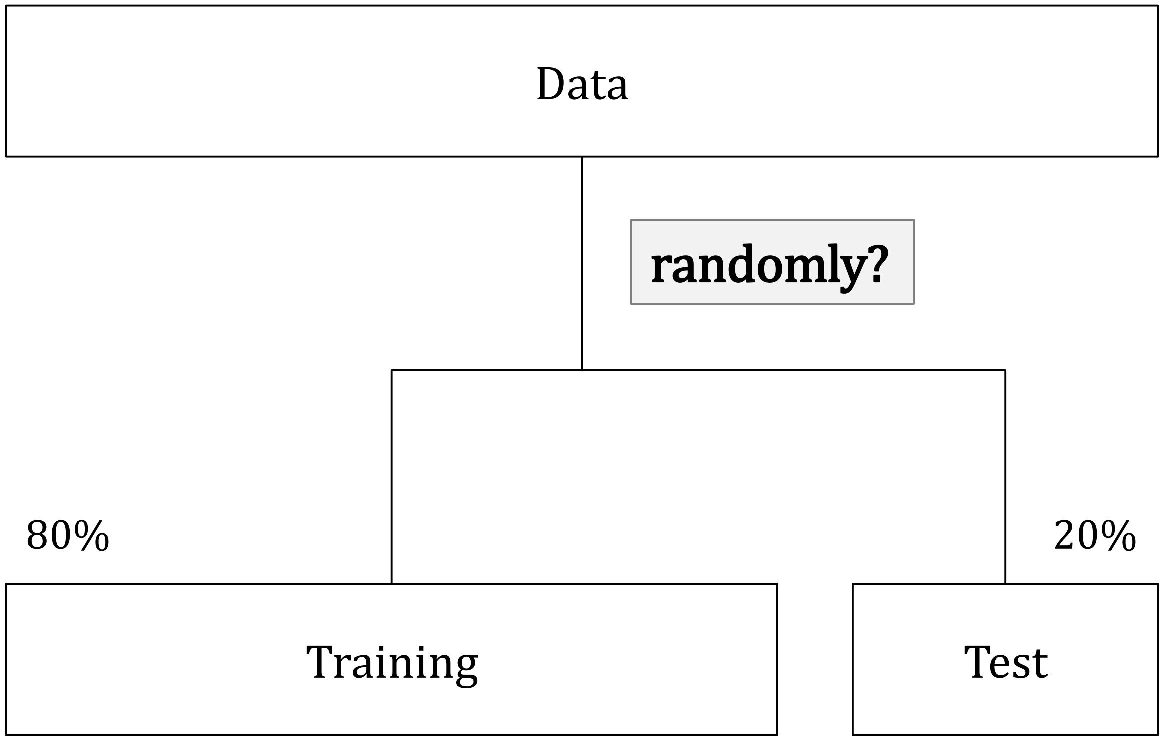
3.2 Create training and test set



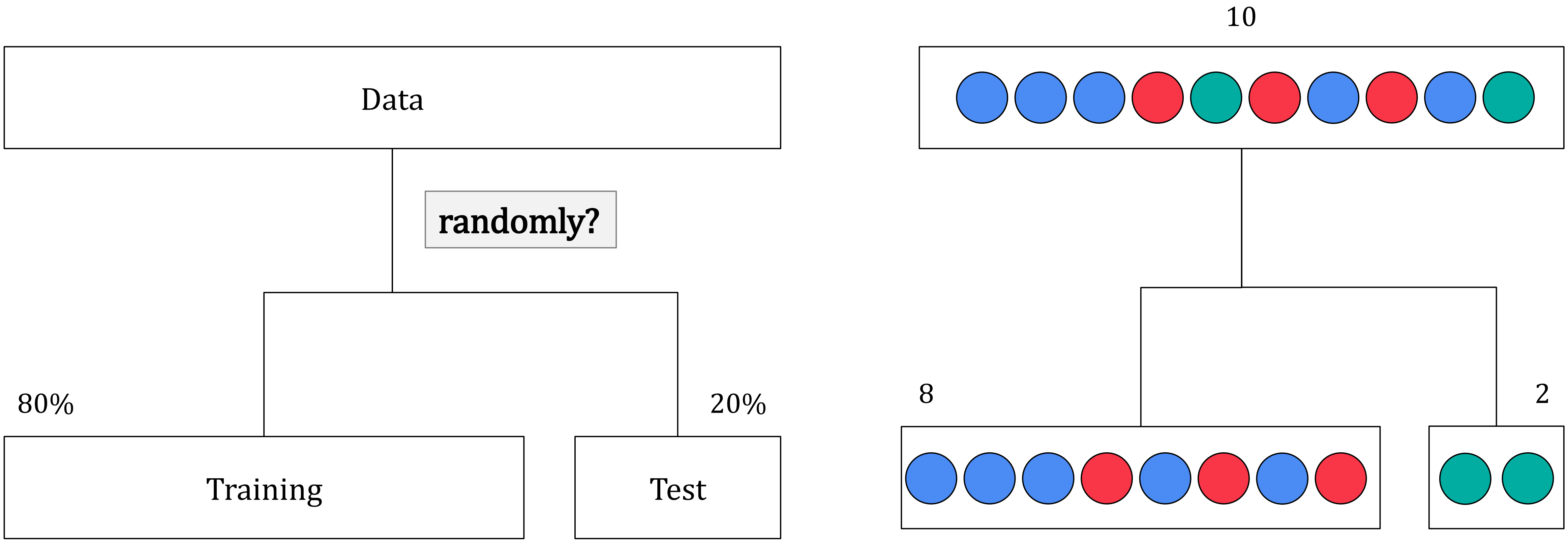
3.2 Create training and test set



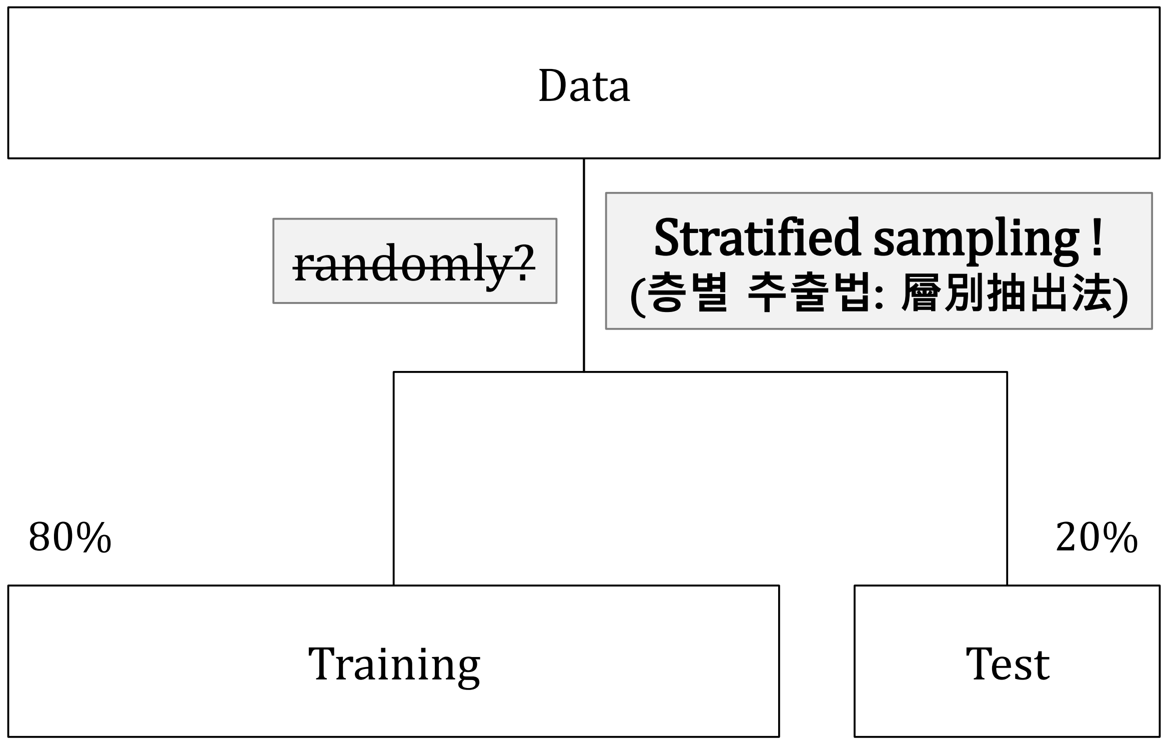
3.2 Create training and test set



3.2 Create training and test set



3.2 Create training and test set



3.2 Create training and test set

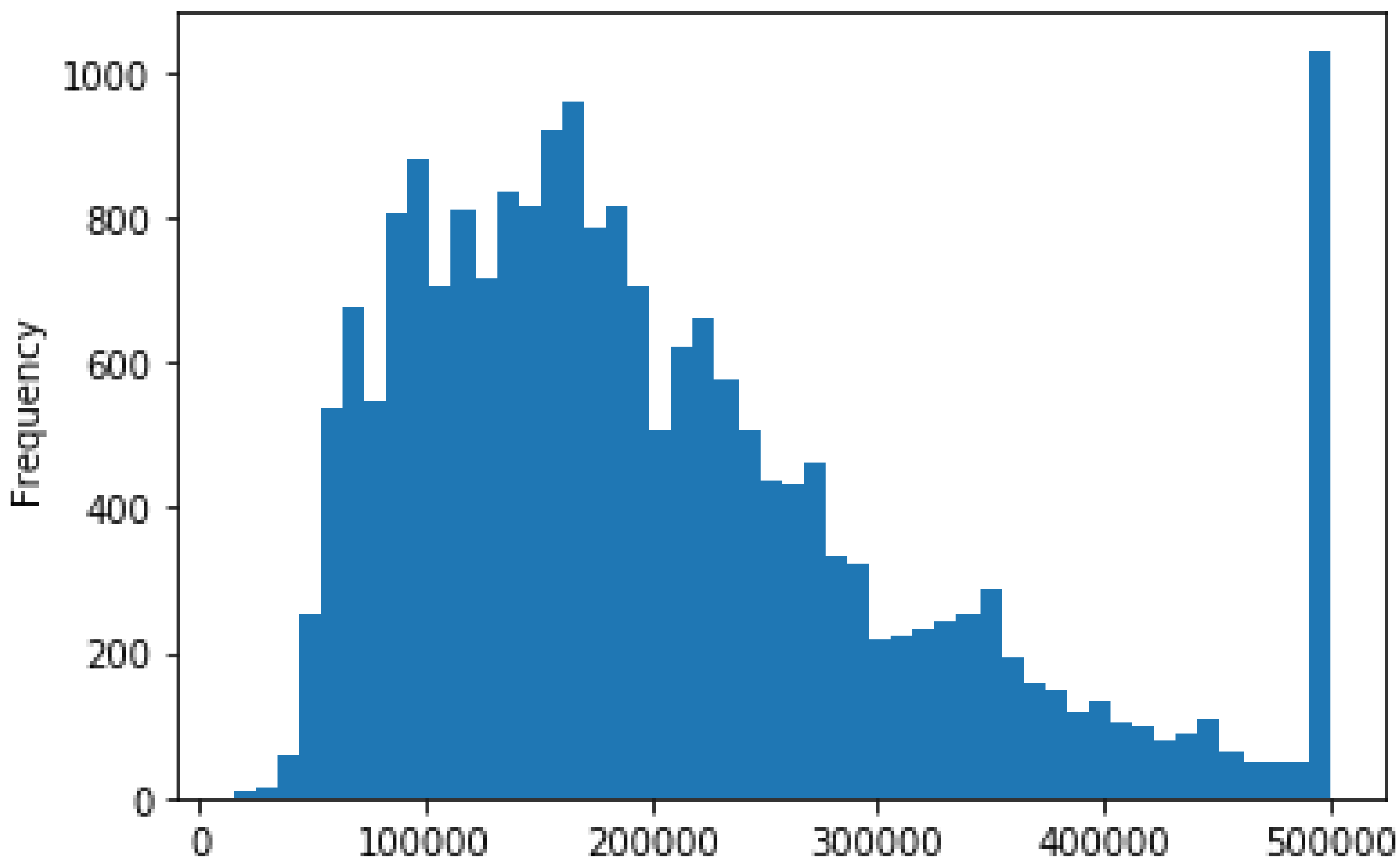
bedrooms_per_room	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN	median_house_value
0.146591	0.0	0.0	0.0	1.0	0.0	452600.0
0.155797	0.0	0.0	0.0	1.0	0.0	358500.0
0.129516	0.0	0.0	0.0	1.0	0.0	352100.0
0.184458	0.0	0.0	0.0	1.0	0.0	341300.0
0.172096	0.0	0.0	0.0	1.0	0.0	342200.0
...	...	...	...	...	...	...
0.224625	0.0	1.0	0.0	0.0	0.0	78100.0
0.215208	0.0	1.0	0.0	0.0	0.0	77100.0
0.215173	0.0	1.0	0.0	0.0	0.0	92300.0
0.219892	0.0	1.0	0.0	0.0	0.0	84700.0
0.221185	0.0	1.0	0.0	0.0	0.0	89400.0

Continuous
(not categorical)

3.2 Create training and test set

```
data["median_house_value"].plot(kind="hist", bins=50)
```

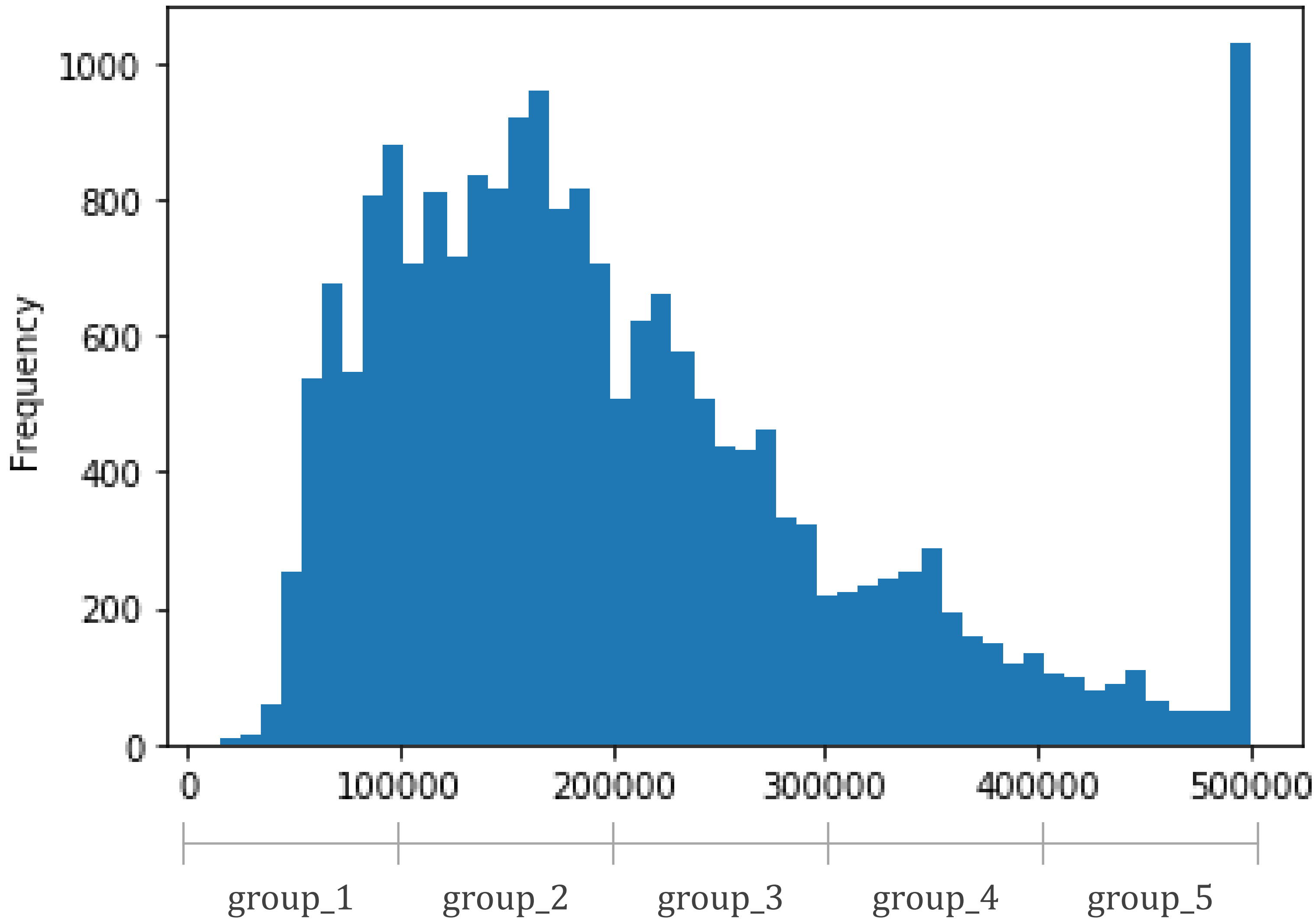
```
<matplotlib.axes._subplots.AxesSubplot at 0x29769c66608>
```



3.2 Create training and test set

```
data["median_house_value"].plot(kind="hist", bins=50)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29769c66608>
```

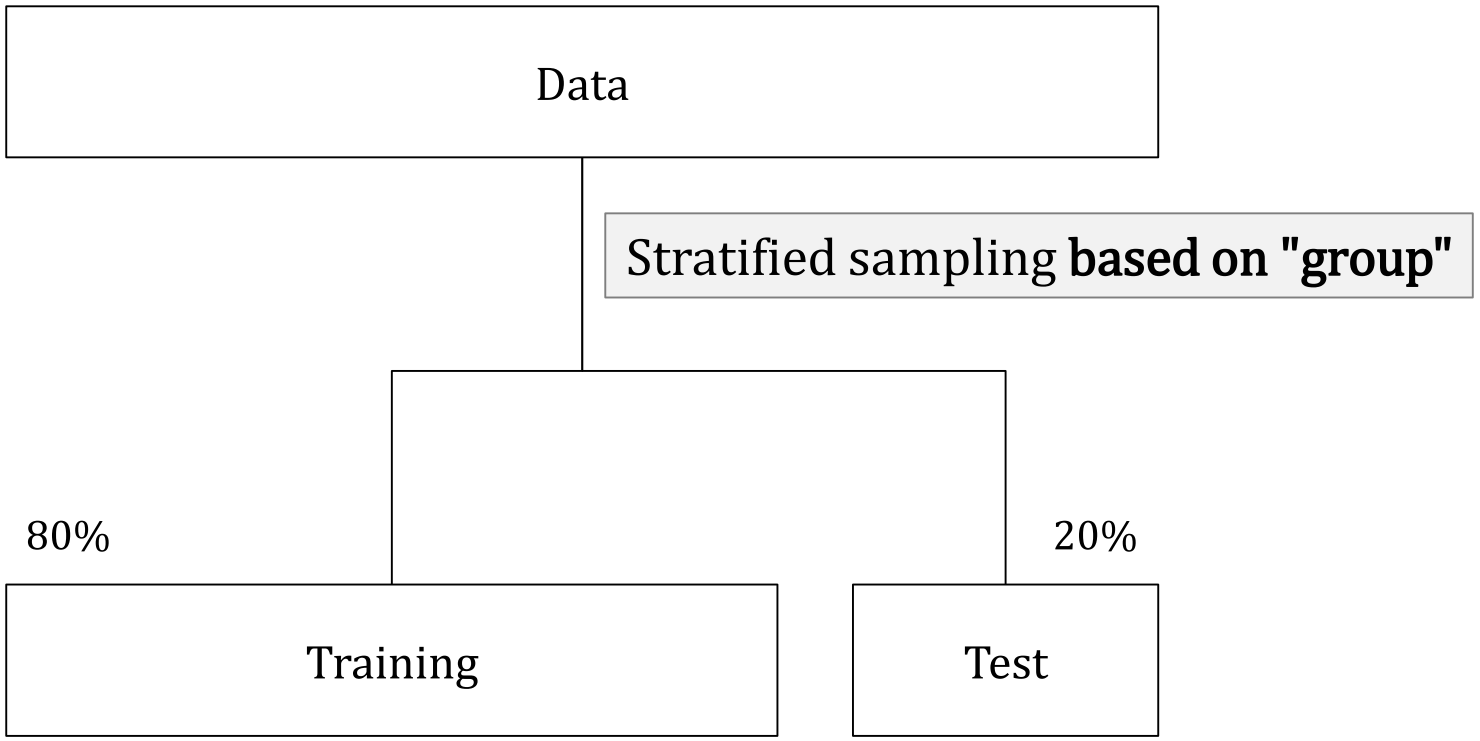


3.2 Create training and test set

```
data["group"] = pd.cut(x=data["median_house_value"],
                       bins=[0, 100000, 200000, 300000, 400000, 500001],
                       labels=["group_1", "group_2", "group_3", "group_4", "group_5"])
data
```

sehold	bedrooms_per_room	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN	median_house_value	group
984127	0.146591	0.0	0.0	0.0	1.0	0.0	452600.0	group_5
238137	0.155797	0.0	0.0	0.0	1.0	0.0	358500.0	group_4
288136	0.129516	0.0	0.0	0.0	1.0	0.0	352100.0	group_4
317352	0.184458	0.0	0.0	0.0	1.0	0.0	341300.0	group_4
281853	0.172096	0.0	0.0	0.0	1.0	0.0	342200.0	group_4
...	...	...	...	...	...	...	...	...
045455	0.224625	0.0	1.0	0.0	0.0	0.0	78100.0	group_1
114035	0.215208	0.0	1.0	0.0	0.0	0.0	77100.0	group_1
205543	0.215173	0.0	1.0	0.0	0.0	0.0	92300.0	group_1
329513	0.219892	0.0	1.0	0.0	0.0	0.0	84700.0	group_1
254717	0.221185	0.0	1.0	0.0	0.0	0.0	89400.0	group_1

3.2 Create training and test set



3.2 Create training and test set

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(data,
                               test_size=0.2,
                               stratify=data["group"],
                               random_state=0)
```

```
train["group"].value_counts() / len(train)
```

```
group_2    0.400799
group_3    0.236131
group_1    0.177204
group_4    0.101381
group_5    0.084484
Name: group, dtype: float64
```

```
test["group"].value_counts() / len(test)
```

```
group_2    0.400921
group_3    0.235950
group_1    0.177326
group_4    0.101260
group_5    0.084545
Name: group, dtype: float64
```

3.2 Create training and test set

- We don't need the "group" feature any longer → Drop it

```
train = train.drop("group", axis=1)
train
```

s_per_room	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN	median_house_value
0.193681	1.0	0.0	0.0	0.0	0.0	210200.0
0.291627	1.0	0.0	0.0	0.0	0.0	343800.0
0.236520	0.0	0.0	0.0	0.0	1.0	159400.0
0.186665	0.0	1.0	0.0	0.0	0.0	145000.0
0.222914	0.0	1.0	0.0	0.0	0.0	86700.0
...	...	...	...	...	...	...
0.222484	0.0	0.0	0.0	0.0	1.0	333800.0
0.177977	0.0	1.0	0.0	0.0	0.0	94900.0
0.274680	1.0	0.0	0.0	0.0	0.0	312500.0
0.120746	0.0	0.0	0.0	1.0	0.0	441400.0
0.922414	1.0	0.0	0.0	0.0	0.0	112500.0

3.2 Create training and test set

- We don't need the "group" feature any longer → Drop it

```
test = test.drop("group", axis=1)
test
```

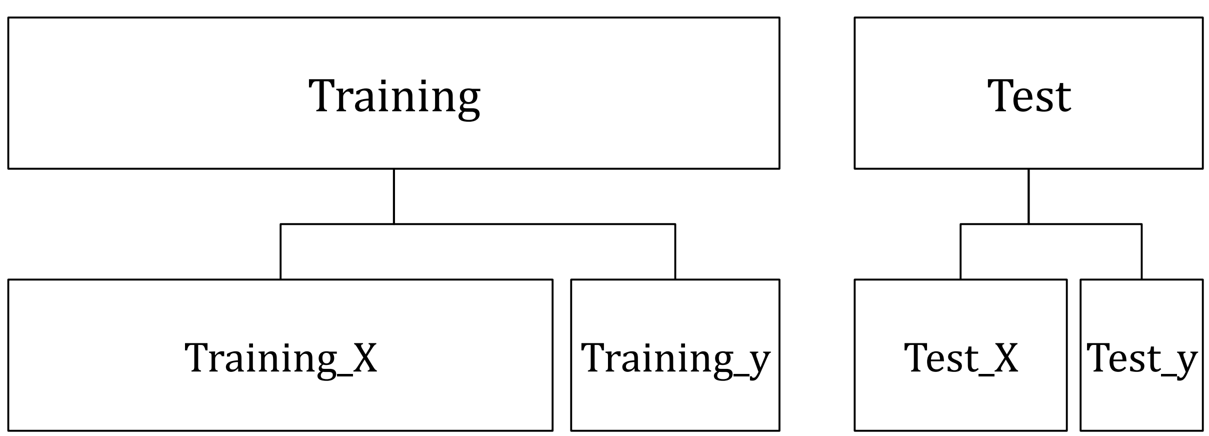
s_per_room	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN	median_house_value
0.187652	1.0	0.0	0.0	0.0	0.0	226800.0
0.196163	0.0	1.0	0.0	0.0	0.0	164000.0
NaN	1.0	0.0	0.0	0.0	0.0	122200.0
0.304348	0.0	1.0	0.0	0.0	0.0	110400.0
0.204733	0.0	1.0	0.0	0.0	0.0	150000.0
...	...	...	...	...	...	...
0.210106	0.0	1.0	0.0	0.0	0.0	39200.0
0.270597	0.0	1.0	0.0	0.0	0.0	141400.0
0.191644	0.0	1.0	0.0	0.0	0.0	87500.0
0.179072	1.0	0.0	0.0	0.0	0.0	223300.0
0.208522	0.0	1.0	0.0	0.0	0.0	106700.0

3.2 Create training and test set

- Common machine learning notations
  - $\boldsymbol{x}$: a vector (district) of all the feature values (excluding the label)
  - y: the label (answer) of $\boldsymbol{x}$

$$\boldsymbol{x} = \begin{pmatrix} -118.36 \\ 33.92 \\ 46 \\ 1{,}231 \\ \dots \end{pmatrix}$$

$$y = 226{,}800$$



3.2 Create training and test set

- For training data

```
train_y = train[["median_house_value"]]
train_y
```

	median_house_value
6229	210200.0
5406	343800.0
14033	159400.0
2236	145000.0
1898	86700.0
...	...
17097	333800.0
12834	94900.0
8836	312500.0
1580	441400.0
4552	112500.0

```
train.drop("median_house_value", axis=1, inplace=True)
train_X = train
train_X
```

_household	bedrooms_per_room	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
5.495177	0.193681	1.0	0.0	0.0	0.0	0.0
3.791971	0.291627	1.0	0.0	0.0	0.0	0.0
4.388013	0.236520	0.0	0.0	0.0	0.0	1.0
6.156938	0.186665	0.0	1.0	0.0	0.0	0.0
5.474950	0.222914	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...
5.005359	0.222484	0.0	0.0	0.0	0.0	1.0
5.628829	0.177977	0.0	1.0	0.0	0.0	0.0
3.879433	0.274680	1.0	0.0	0.0	0.0	0.0
8.106101	0.120746	0.0	0.0	0.0	1.0	0.0
1.260870	0.922414	1.0	0.0	0.0	0.0	0.0

3.2 Create training and test set

- For test data

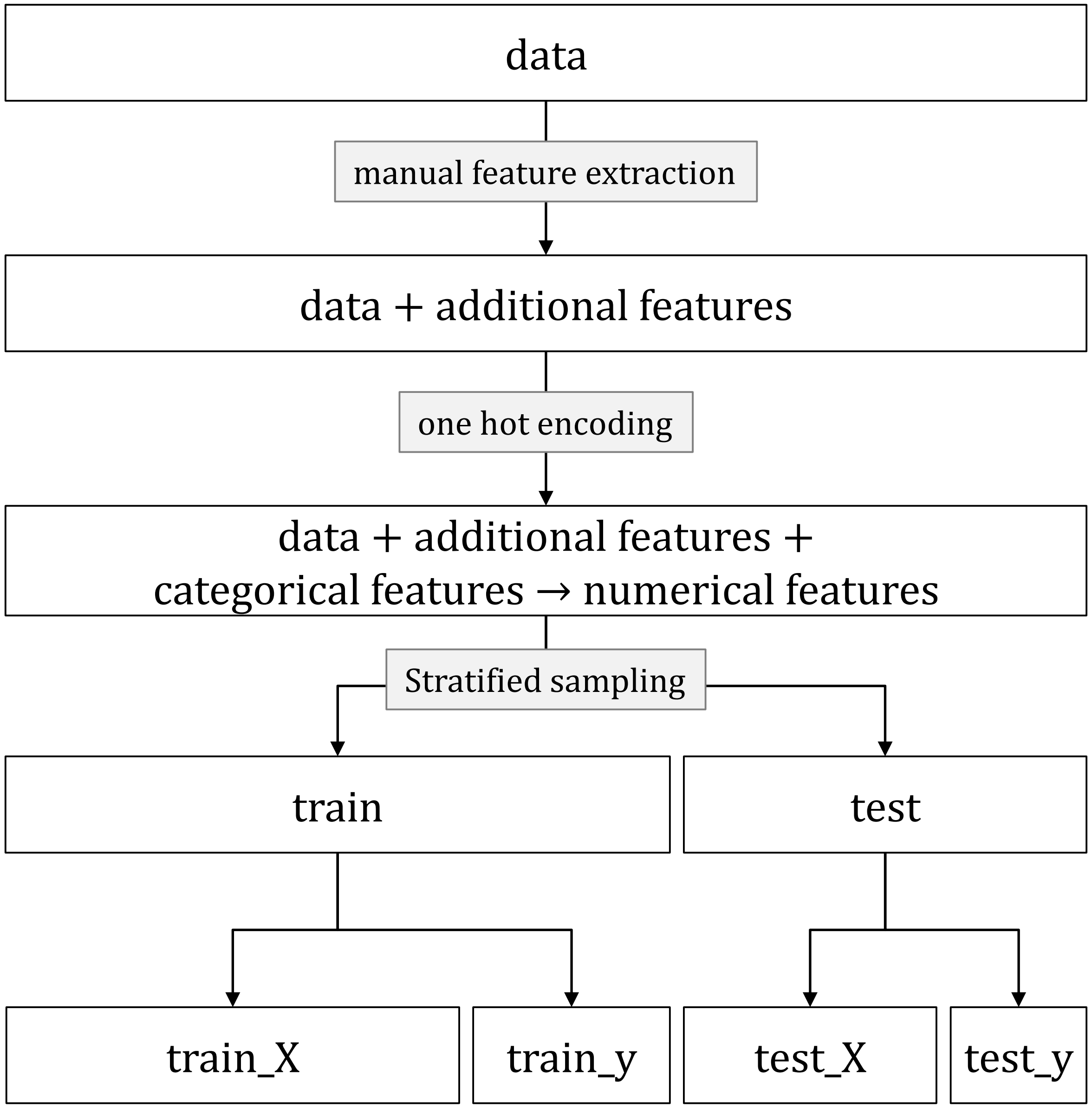
```
test_y = test[["median_house_value"]]
test_y
```

	median_house_value
8429	226800.0
20117	164000.0
4767	122200.0
16192	110400.0
12909	150000.0
...	...
2475	39200.0
12704	141400.0
1096	87500.0
17715	223300.0
13779	106700.0

```
test.drop("median_house_value", axis=1, inplace=True)
test_X = test
test_X
```

bedrooms_per_room	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0.187652	1.0	0.0	0.0	0.0	0.0
0.196163	0.0	1.0	0.0	0.0	0.0
NaN	1.0	0.0	0.0	0.0	0.0
0.304348	0.0	1.0	0.0	0.0	0.0
0.204733	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...
0.210106	0.0	1.0	0.0	0.0	0.0
0.270597	0.0	1.0	0.0	0.0	0.0
0.191644	0.0	1.0	0.0	0.0	0.0
0.179072	1.0	0.0	0.0	0.0	0.0
0.208522	0.0	1.0	0.0	0.0	0.0

Until now..


data
manual feature extraction
data + additional features
one hot encoding
data + additional features +
categorical features → numerical features
Stratified sampling
train
test
train_X
train_y
test_X
test_y

3.3 Data Cleaning

```
train_X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 6229 to 4552
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           16512 non-null  float64
 1   latitude            16512 non-null  float64
 2   housing_median_age  16512 non-null  float64
 3   total_rooms         16512 non-null  float64
 4   total_bedrooms      16349 non-null  float64
 5   population          16512 non-null  float64
 6   households          16512 non-null  float64
 7   median_income       16512 non-null  float64
 8   rooms_per_household 16512 non-null  float64
 9   bedrooms_per_room   16349 non-null  float64
 10  <1H OCEAN           16512 non-null  float64
 11  INLAND              16512 non-null  float64
 12  ISLAND              16512 non-null  float64
 13  NEAR BAY            16512 non-null  float64
 14  NEAR OCEAN          16512 non-null  float64
dtypes: float64(15)
memory usage: 2.0 MB
```

Missing values
(N/A: Not Available)

3.3 Data Cleaning

	...	total_bedrooms	...
0	...	...	...
1	...		...
2	...	...	...
3	...	...	...
...	...	...	...

1. Get rid of the corresponding districts

2. Get rid of the whole attribute (feature)

3. Set the values to some value

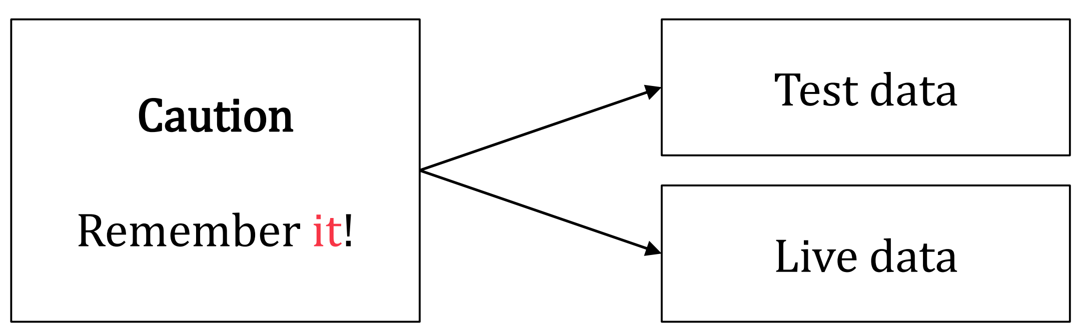
	...	total_bedrooms	...
0	...	...	...
~~1~~	~~...~~		~~...~~
2	...	...	...
3	...	...	...
...	...	...	...

	...	total_bedrooms	...
0	...	...	...
1	...		...
2	...	...	...
3	...	...	...
...	...	...	...

	...	total_bedrooms	...
0	...	...	...
1	...	some value	...
2	...	...	...
3	...	...	...
...	...	...	...

3.3 Data Cleaning

	...	total_bedrooms	...
0	...	...	...
1	...	zero/mean/**median**/etc.	...
2	...	...	...
3	...	...	...
...	...	...	...



3.3 Data Cleaning

- For training data

```
median = train_X["total_bedrooms"].median()
train_X["total_bedrooms"].fillna(median, inplace=True)
train_X["bedrooms_per_room"].fillna(train_X["total_bedrooms"]/train_X["total_rooms"], inplace=True)
train_X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 6229 to 4552
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   longitude            16512 non-null  float64
 1   latitude             16512 non-null  float64
 2   housing_median_age   16512 non-null  float64
 3   total_rooms          16512 non-null  float64
 4   total_bedrooms       16512 non-null  float64
 5   population           16512 non-null  float64
 6   households           16512 non-null  float64
 7   median_income        16512 non-null  float64
 8   rooms_per_household  16512 non-null  float64
 9   bedrooms_per_room    16512 non-null  float64
 10  <1H OCEAN            16512 non-null  float64
 11  INLAND               16512 non-null  float64
 12  ISLAND               16512 non-null  float64
 13  NEAR BAY             16512 non-null  float64
 14  NEAR OCEAN           16512 non-null  float64
dtypes: float64(15)
```

3.3 Data Cleaning

- For test data

```
test_X["total_bedrooms"].fillna(median, inplace=True)
test_X["bedrooms_per_room"].fillna(test_X["total_bedrooms"]/test_X["total_rooms"], inplace=True)
test_X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4128 entries, 8429 to 13779
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           4128 non-null   float64
 1   latitude            4128 non-null   float64
 2   housing_median_age  4128 non-null   float64
 3   total_rooms         4128 non-null   float64
 4   total_bedrooms      4128 non-null   float64
 5   population          4128 non-null   float64
 6   households          4128 non-null   float64
 7   median_income       4128 non-null   float64
 8   rooms_per_household 4128 non-null   float64
 9   bedrooms_per_room   4128 non-null   float64
 10  <1H OCEAN           4128 non-null   float64
 11  INLAND              4128 non-null   float64
 12  ISLAND              4128 non-null   float64
 13  NEAR BAY            4128 non-null   float64
 14  NEAR OCEAN          4128 non-null   float64
dtypes: float64(15)
```

End-to-End Machine Learning Project

	No.	Action	Package/library
Data preprocessing for machine learning	0	Look at the big picture	–
	1	Get the data	tarfile, urllib, pandas
	2	Discover and visualize the data to gain insights	pandas, matplotlib
	3	Prepare the data for Machine Learning algorithms	pandas, scikit-learn, numpy
	4	Select a model and train it	
	5	Fine-tune your model	
	6	Present your solution	
	7	Launch, monitor and maintain your system	joblib, flask

End-to-End Machine Learning Project

No.	Action	Package/library
0	Look at the big picture	–
1	Get the data	tarfile, urllib, pandas
2	Discover and visualize the data to gain insights	pandas, matplotlib
3	Prepare the data for Machine Learning algorithms	pandas, scikit-learn, numpy
4	Select a model and train it	
5	Fine-tune your model	
6	Present your solution	
7	Launch, monitor and maintain your system	joblib, flask

Three ML methods we gonna use today..

Linear regression

Decision tree regression

Random forest regression

4.1 Training and Evaluating on the Training Set

Linear regression

```
from sklearn.linear_model import LinearRegression

linear = LinearRegression()
linear.fit(train_X, train_y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_job
s=None, normalize=False)
```

```
from sklearn.metrics import mean_squared_error
import numpy as np

predictions = linear.predict(train_X)
mse = mean_squared_error(train_y, predictions)
rmse = np.sqrt(mse)
answer_mean = train_y["median_house_value"].mean()
print(str(rmse/answer_mean*100) + "%")
```

```
32.92999142716399%
```

4.1 Training and Evaluating on the Training Set

Decision tree regression

```
from sklearn.tree import DecisionTreeRegressor

tree = DecisionTreeRegressor()
tree.fit(train_X, train_y)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', m
ax_depth=None,
                      max_features=None, max_leaf_nodes
=None,
                      min_impurity_decrease=0.0, min_im
purity_split=None,
                      min_samples_leaf=1, min_samples_s
plit=2,
                      min_weight_fraction_leaf=0.0, pre
sort='deprecated',
                      random_state=None, splitter='bes
t')
```

```
predictions = tree.predict(train_X)
mse = mean_squared_error(train_y, predictions)
rmse = np.sqrt(mse)
answer_mean = train_y["median_house_value"].mean()
print(str(rmse/answer_mean*100) + "%")
```

```
0.0%
```

4.1 Training and Evaluating on the Training Set

Linear regression	**Decision tree regression**	**Random forest regression**
Under fitting	Over fitting	

4.2 Evaluating on the Test Set

```
predictions = linear.predict(test_X)
mse = mean_squared_error(test_y, predictions)
rmse = np.sqrt(mse)
answer_mean = test_y["median_house_value"].mean()
print(str(rmse/answer_mean*100) + "%")
```

```
33.387643759619735%
```

```
predictions = tree.predict(test_X)
mse = mean_squared_error(test_y, predictions)
rmse = np.sqrt(mse)
answer_mean = test_y["median_house_value"].mean()
print(str(rmse/answer_mean*100) + "%")
```

```
33.33372756965135%
```

4.3 Evaluating on the Training and Test Set using Random forest regression

Linear regression

Decision tree regression

Random forest regression

4.3 Evaluating on the Training and Test Set using Random forest regression

```
from sklearn.ensemble import RandomForestRegressor

forest = RandomForestRegressor(n_estimators=5, random_state=0)
forest.fit(train_X, train_y["median_house_value"].ravel())
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_node
s=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=5, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

4.3 Evaluating on the Training and Test Set using Random forest regression

```
predictions = forest.predict(train_X)
mse = mean_squared_error(train_y, predictions)
rmse = np.sqrt(mse)
answer_mean = train_y["median_house_value"].mean()
print(str(rmse/answer_mean*100) + "%")
```

```
12.12961140719765%
```

```
predictions = forest.predict(test_X)
mse = mean_squared_error(test_y, predictions)
rmse = np.sqrt(mse)
answer_mean = test_y["median_house_value"].mean()
print(str(rmse/answer_mean*100) + "%")
```

```
26.17531878445498%
```

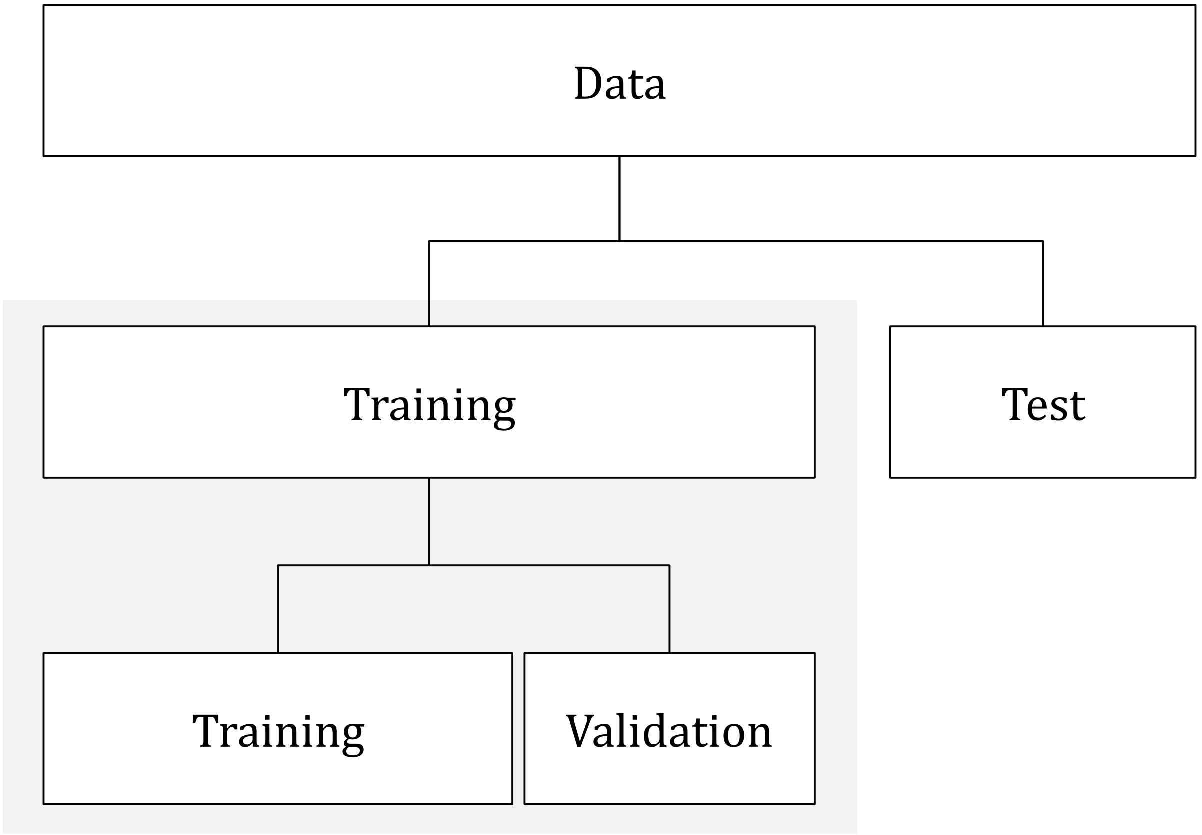
Comparison

	Linear regression	**Decision tree regression**	**Random forest regression**
RMSE on training set	32.9%	0%	12.1%
RMSE on test set	33.3%	33.3%	26.1%

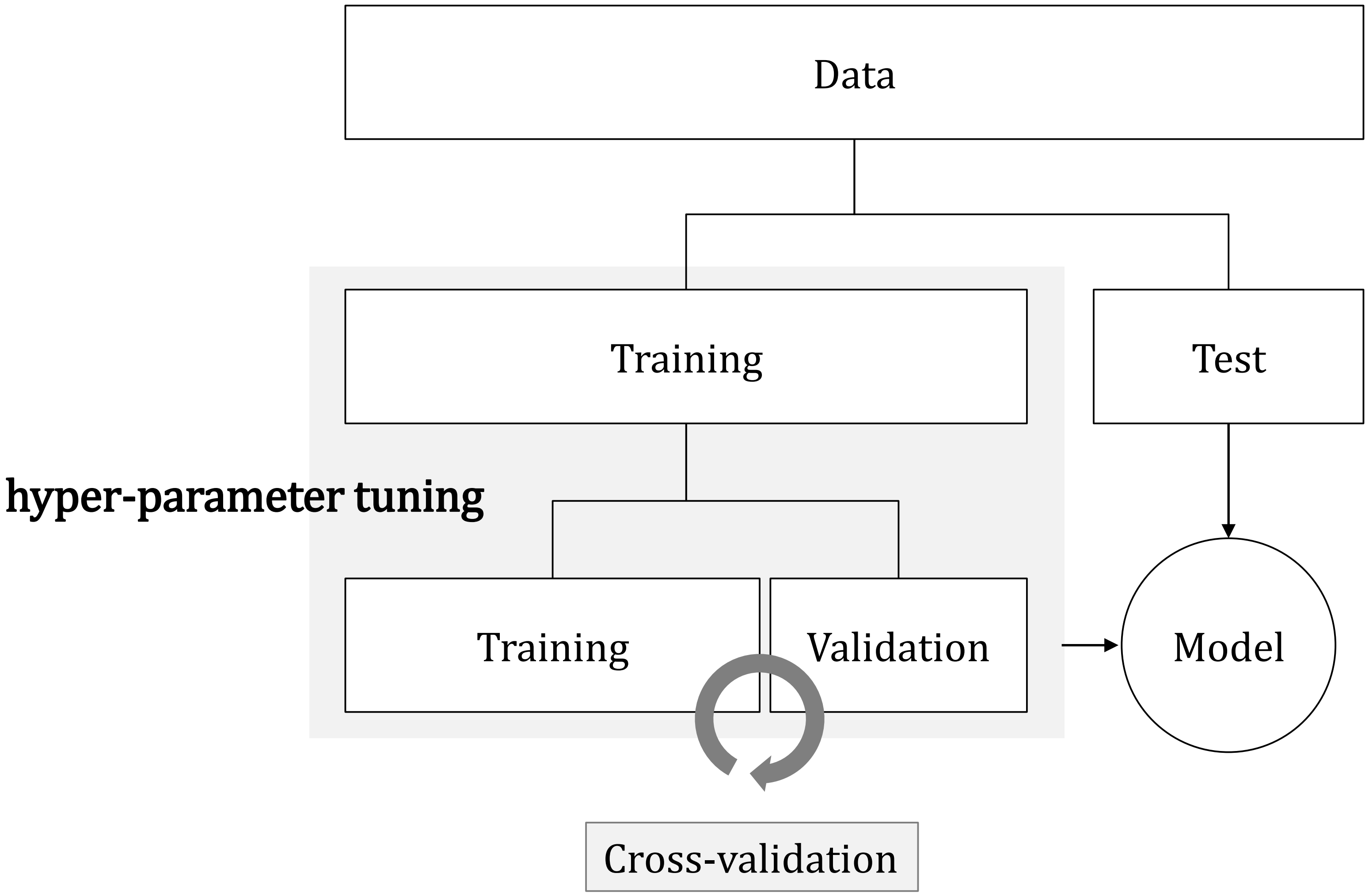
End-to-End Machine Learning Project

No.	Action	Package/library
0	Look at the big picture	–
1	Get the data	tarfile, urllib, pandas
2	Discover and visualize the data to gain insights	pandas, matplotlib
3	Prepare the data for Machine Learning algorithms	
4	Select a model and train it	
5	Fine-tune your model	pandas, scikit-learn, numpy
6	Present your solution	
7	Launch, monitor and maintain your system	joblib, flask

The ideal scenario



The ideal scenario



Grid Search with Cross Validation

```
from sklearn.model_selection import GridSearchCV

param = {"n_estimators": [5, 10, 30],
         "max_depth": [100, 200]}
forest = RandomForestRegressor(random_state=0)
search = GridSearchCV(forest, param,
                      cv=5, scoring="neg_mean_squared_error")
search.fit(train_X, train_y["median_house_value"].ravel())
```

Your fine–tuned model

```
search.best_params_
```

```
{'max_depth': 100, 'n_estimators': 30}
```

```
forest = RandomForestRegressor(max_depth=100,
                               n_estimators=30,
                               random_state=0)
forest.fit(train_X, train_y["median_house_value"].ravel())
```

Your fine–tuned model

```
predictions = forest.predict(train_X)
mse = mean_squared_error(train_y, predictions)
rmse = np.sqrt(mse)
answer_mean = train_y["median_house_value"].mean()
print(str(rmse/answer_mean*100) + "%")
```

```
9.421054753574424%
```

```
predictions = forest.predict(test_X)
mse = mean_squared_error(test_y, predictions)
rmse = np.sqrt(mse)
answer_mean = test_y["median_house_value"].mean()
print(str(rmse/answer_mean*100) + "%")
```

```
23.886510037964545%
```

Your fine–tune model

	Linear regression	Decision tree regression	Random forest regression	Fine-tuned model
RMSE on training set	32.9%	0%	12.1%	9.4%
RMSE on test set	33.3%	33.3%	26.1%	23.8%

End-to-End Machine Learning Project

No.	Action	Package/library
0	Look at the big picture	–
1	Get the data	tarfile, urllib, pandas
2	Discover and visualize the data to gain insights	pandas, matplotlib
3	Prepare the data for Machine Learning algorithms	
4	Select a model and train it	pandas, scikit-learn, numpy
5	Fine-tune your model	
6	Present your solution	
7	Launch, monitor and maintain your system	joblib, flask

Feature importance

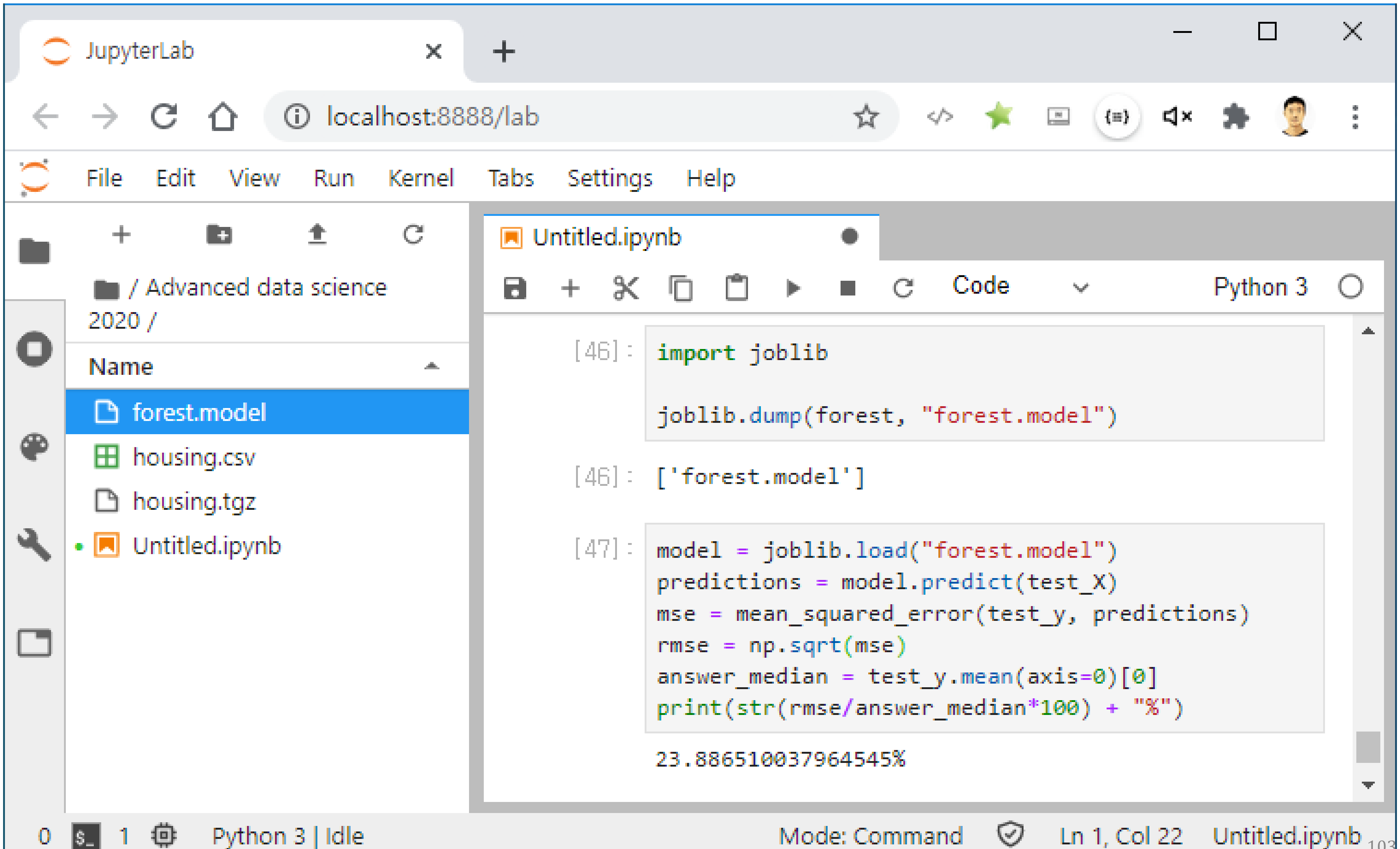
```
importances = forest.feature_importances_
features = train_X.columns.values

sorted(zip(importances, features), reverse=True)
```

```
[(0.47763306801959393, 'median_income'),
 (0.1466778103089264 3, 'INLAND'),
 (0.0967516915621464 1, 'longitude'),
 (0.08728313888933595, 'latitude'),
 (0.047588373717733196, 'housing_median_age'),
 (0.0324486860711676 7, 'bedrooms_per_room'),
 (0.028850969908527545, 'rooms_per_household'),
 (0.024088508967163785, 'population'),
 (0.0185580703269689 2, 'total_rooms'),
 (0.015667964018780612, 'total_bedrooms'),
 (0.0143135981636599 3, 'households'),
 (0.006327717669091209, 'NEAR OCEAN'),
 (0.0026947420610949524, '<1H OCEAN'),
 (0.0010376273846289 86, 'NEAR BAY'),
 (7.8032931180394e-05, 'ISLAND')]
```

End-to-End Machine Learning Project

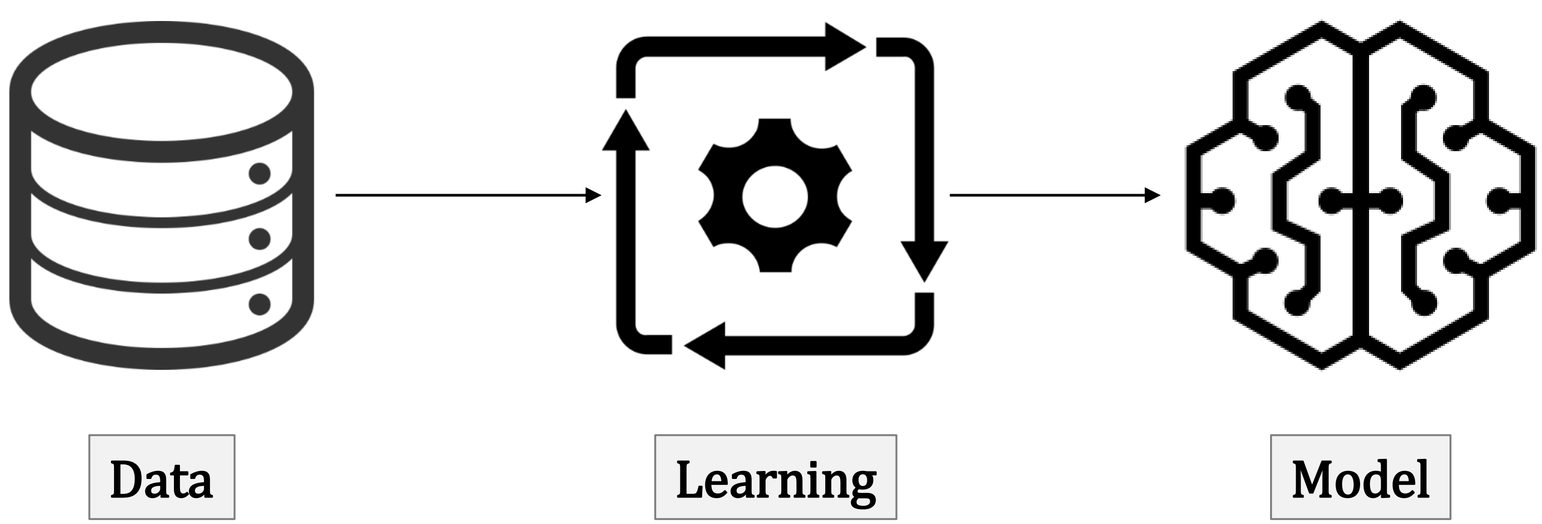
No.	Action	Package/library
0	Look at the big picture	–
1	Get the data	tarfile, urllib, pandas
2	Discover and visualize the data to gain insights	pandas, matplotlib
3	Prepare the data for Machine Learning algorithms	
4	Select a model and train it	pandas, scikit-learn, numpy
5	Fine-tune your model	
6	Present your solution	
7	Launch, monitor and maintain your system	joblib, flask

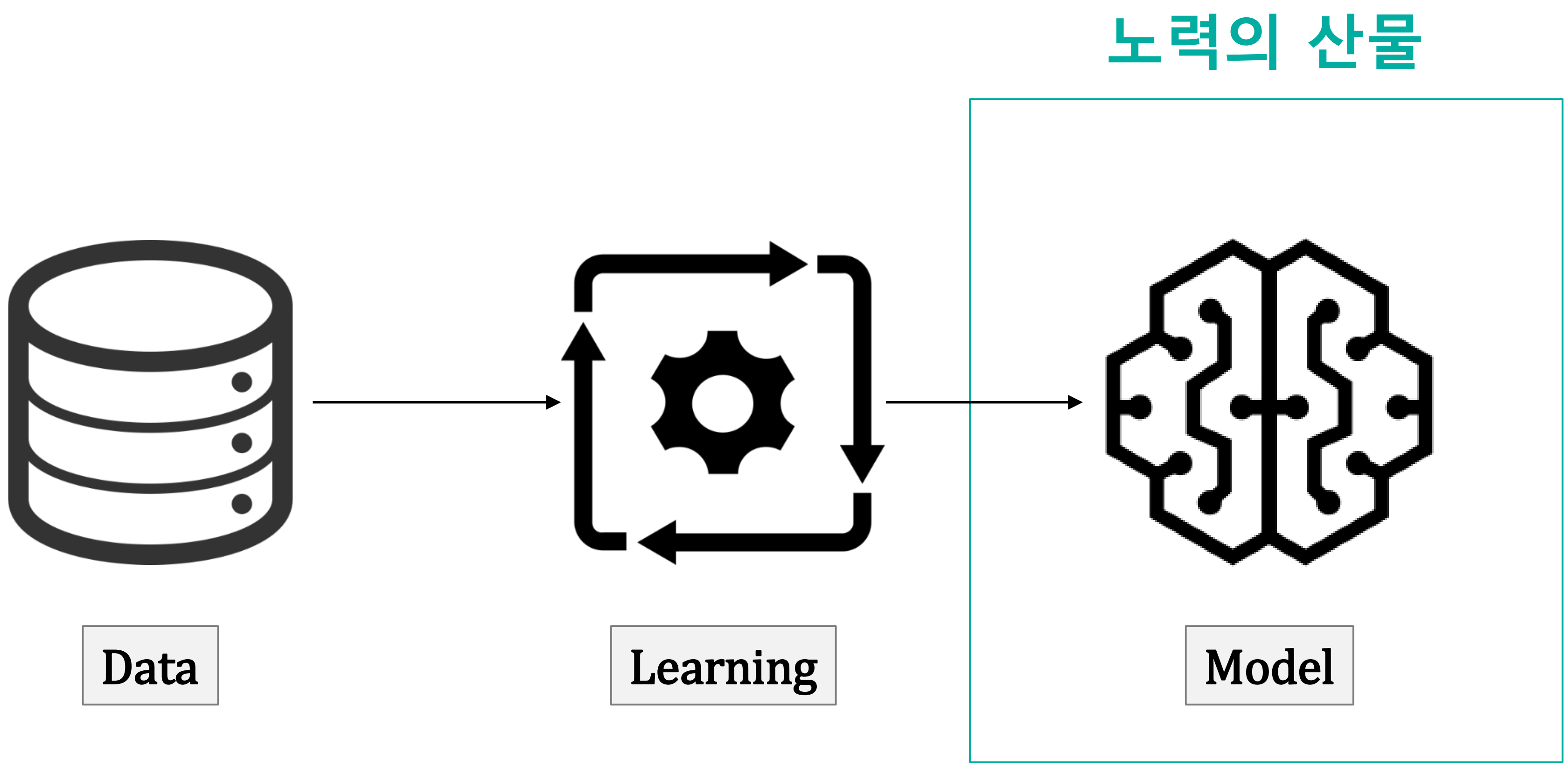
Save & load your model



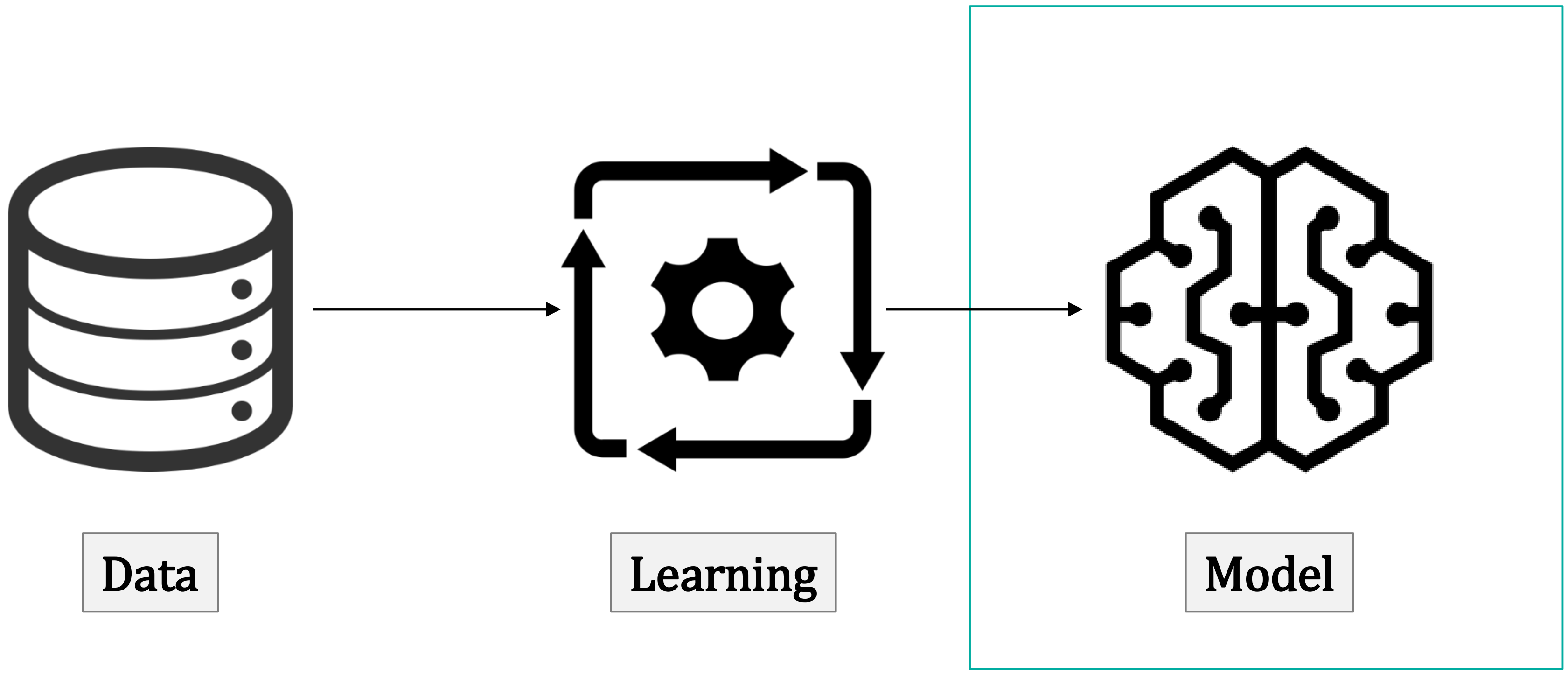
기계 학습 모델 구축 → 많은 자원 소모

- 데이터 수집 및 분석
- 적합한 기계 학습 방법론 선택
  - Linear regression, decision tree, random forest, perceptron, neural network, etc.
- 다양한 hyper parameter 설정
  - Num. of layers, activation function, learning rate, etc.




노력의 산물
Data
Learning
Model

노력의 산물



프로그래밍 언어의 장벽

지적 재산권(보안)

APIs & REST APIs

Application **P**rogramming **I**nterface

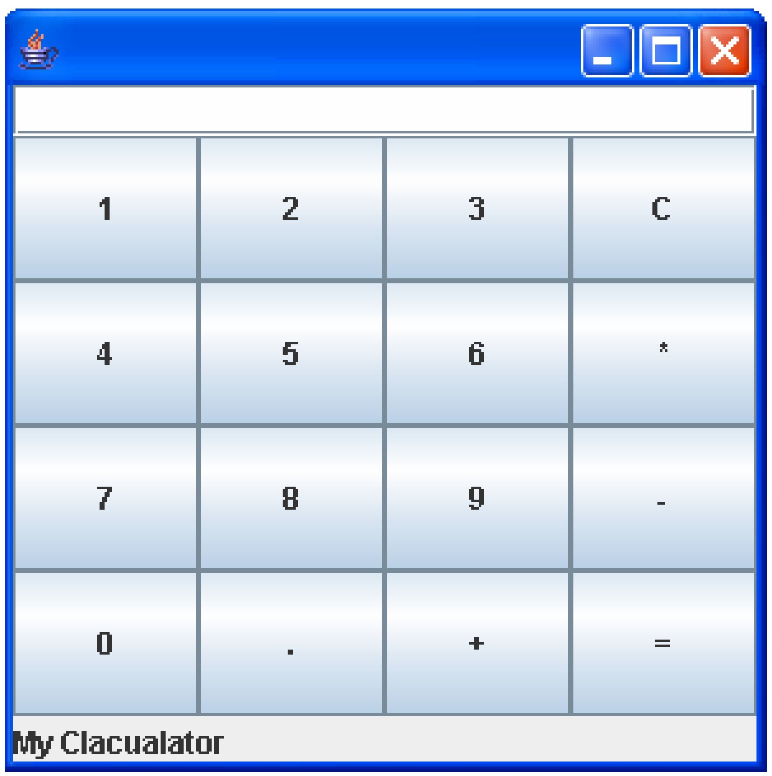
Application Programming Interface

- A way to let software components to talk to each other

API 경험이 있나요?

API 경험이 있나요?

- Swing (Java)



JFrame

+ EXIT_ON_CLOSE: int
– rootPane: JRootPane
...

+ JFrame( )
...
+ setTitle(String): void
+ setSize(int, int): void
+ setIconImage(Image): void
+ setVisible(boolean): void
...

```
public static void main(String args)
{
    JFrame frame = new JFrame();
    frame.setTitle("Cacluator");
    frame.setVisible(true);
}
```

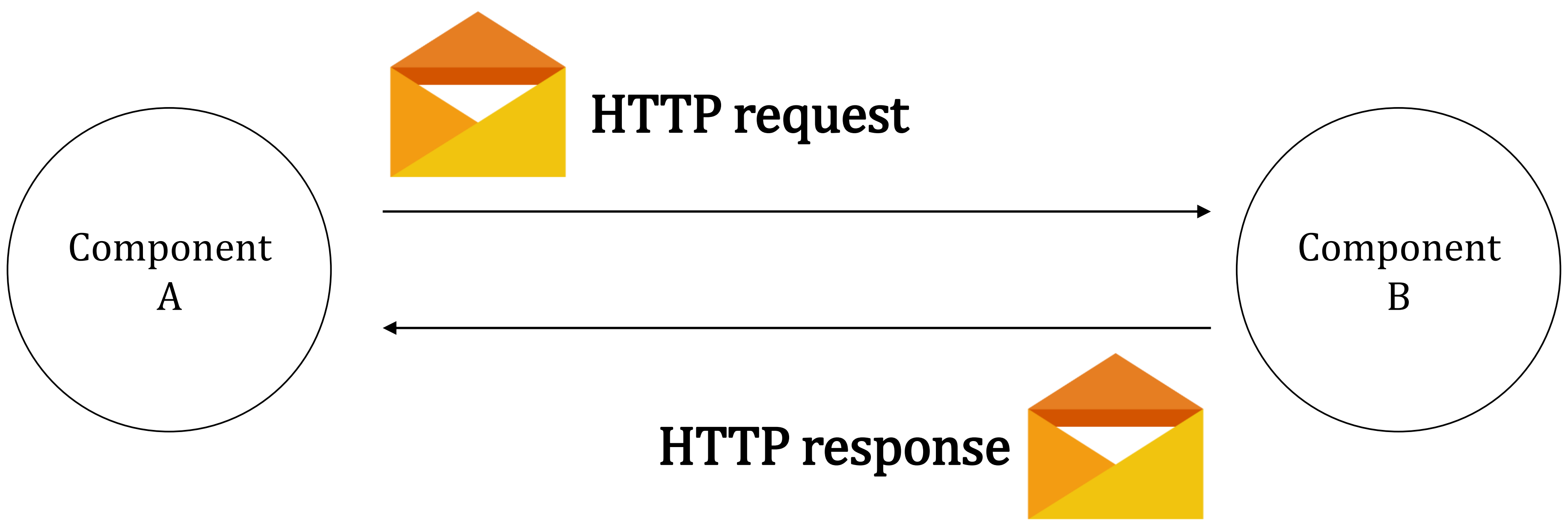
So an **API** could be anything in any form.

The only thing that it has to be is that

it has to be a **way to communicate with a software component**.

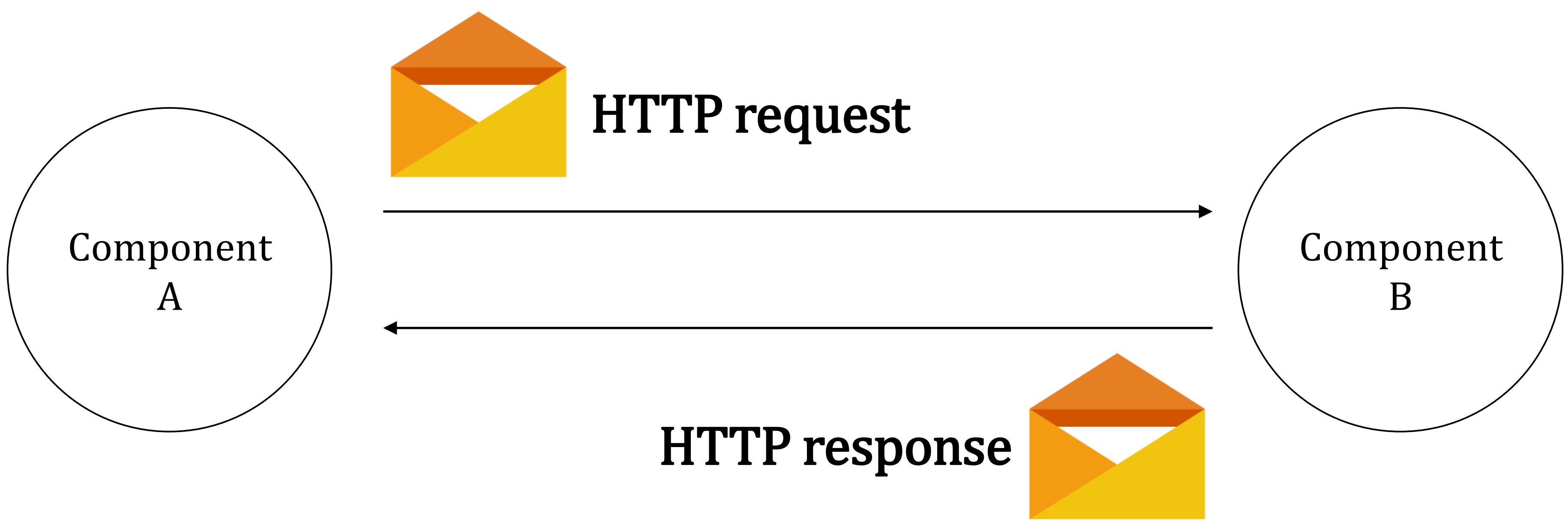
REST API

- HTTP 프로토콜 기반의 API



REST API

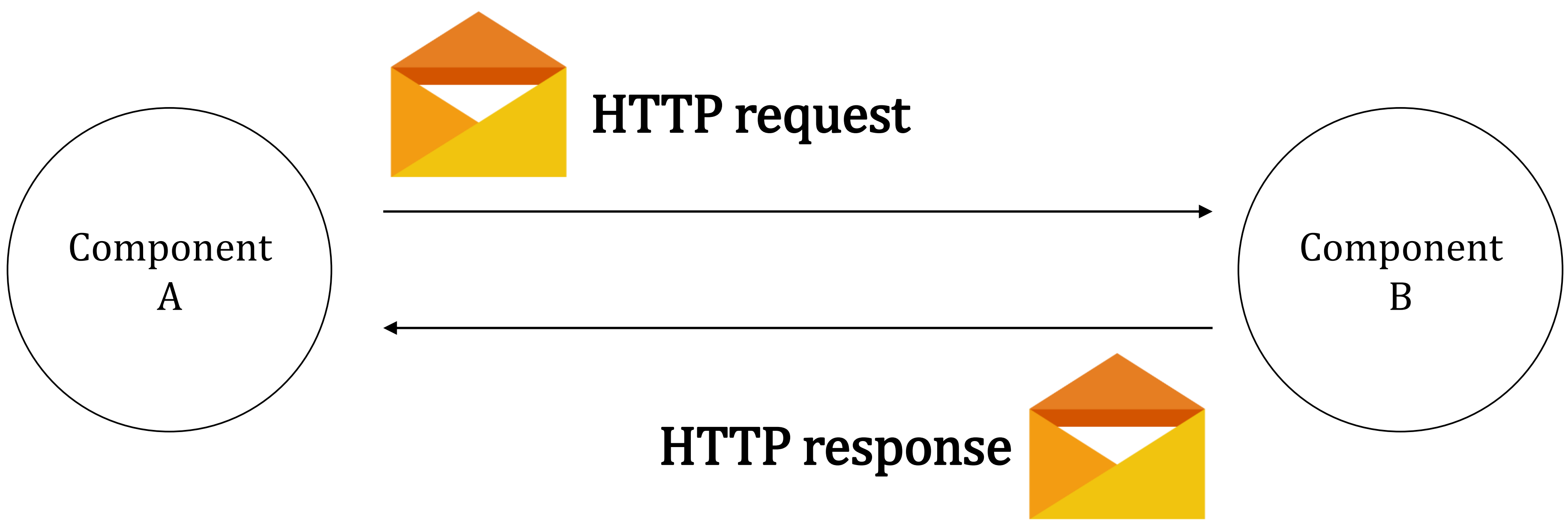
- HTTP 프로토콜 기반의 API



Client language	Server language
Python	Python
Python	Java
Java	Python
...	...

REST API

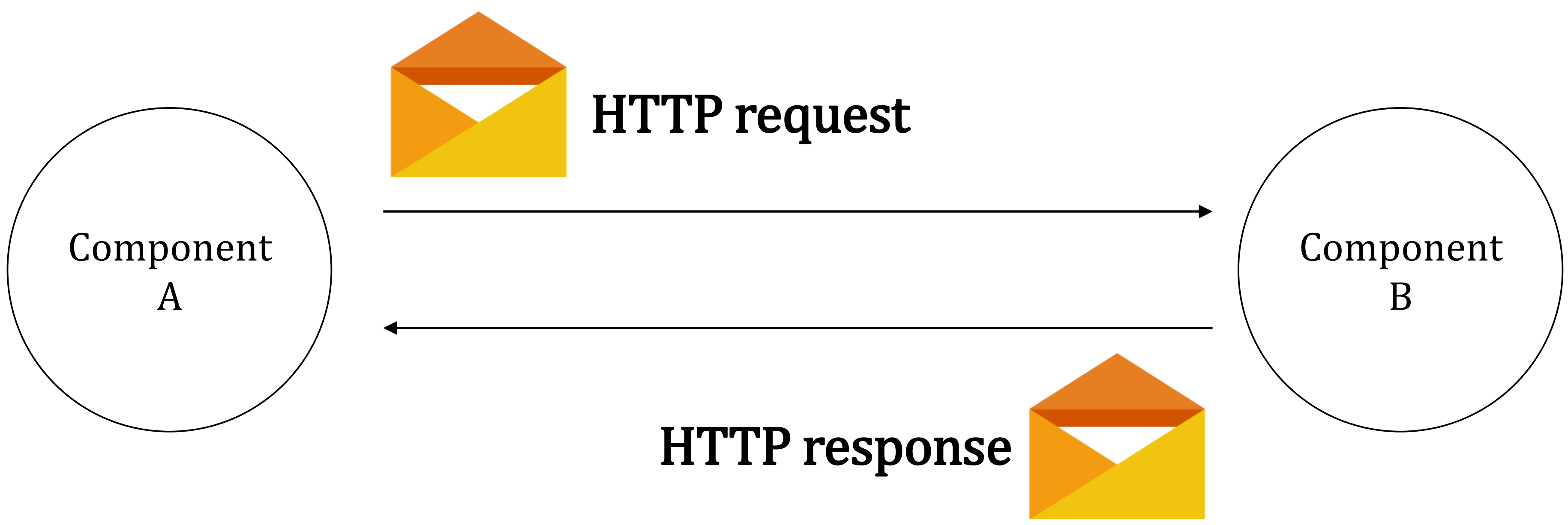
- HTTP 프로토콜 기반의 API



CRUD	HTTP verb
Create	POST
Read	GET
Update	PUT/PATCH
Delete	DELTE

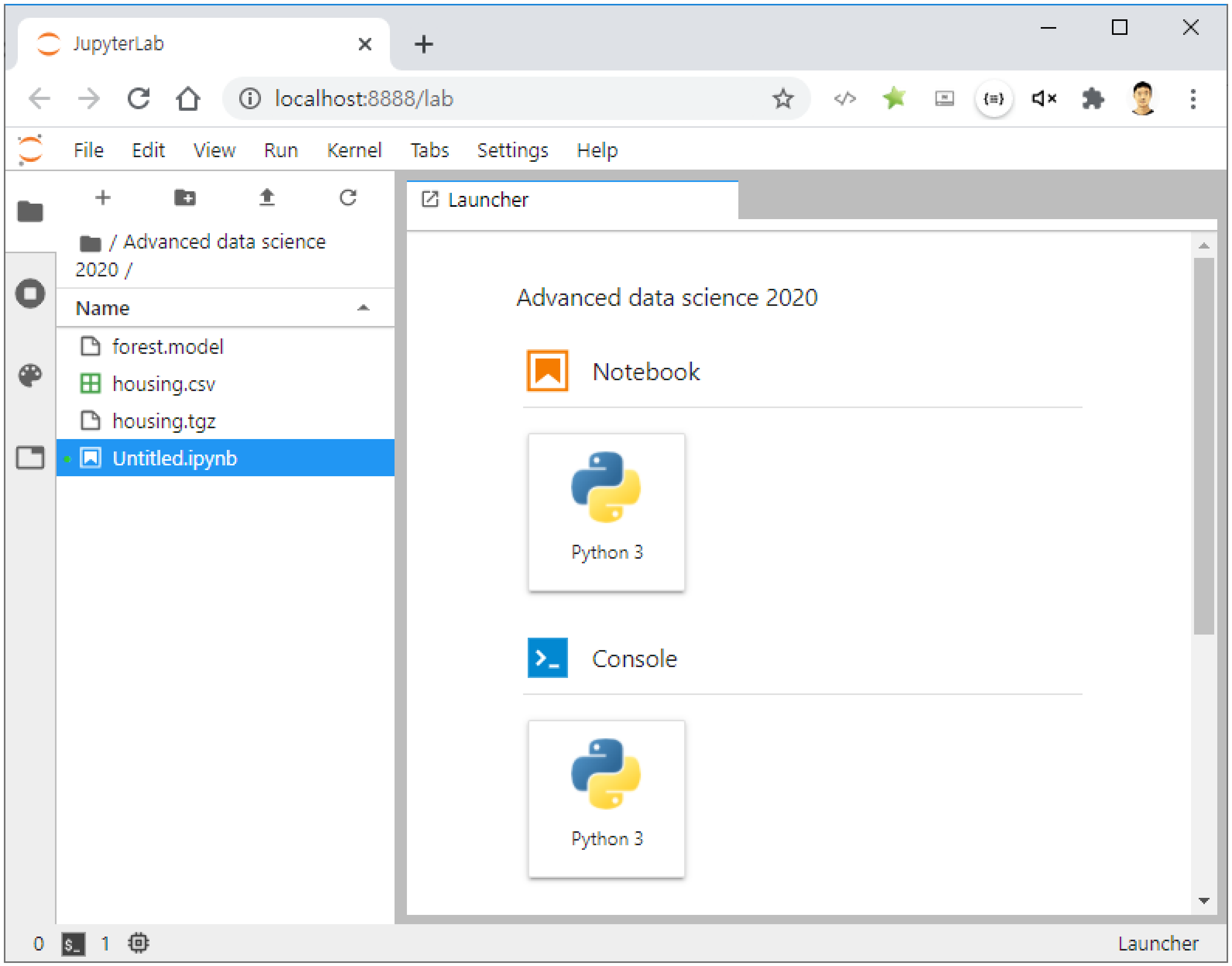
REST API

- HTTP 프로토콜 기반의 API

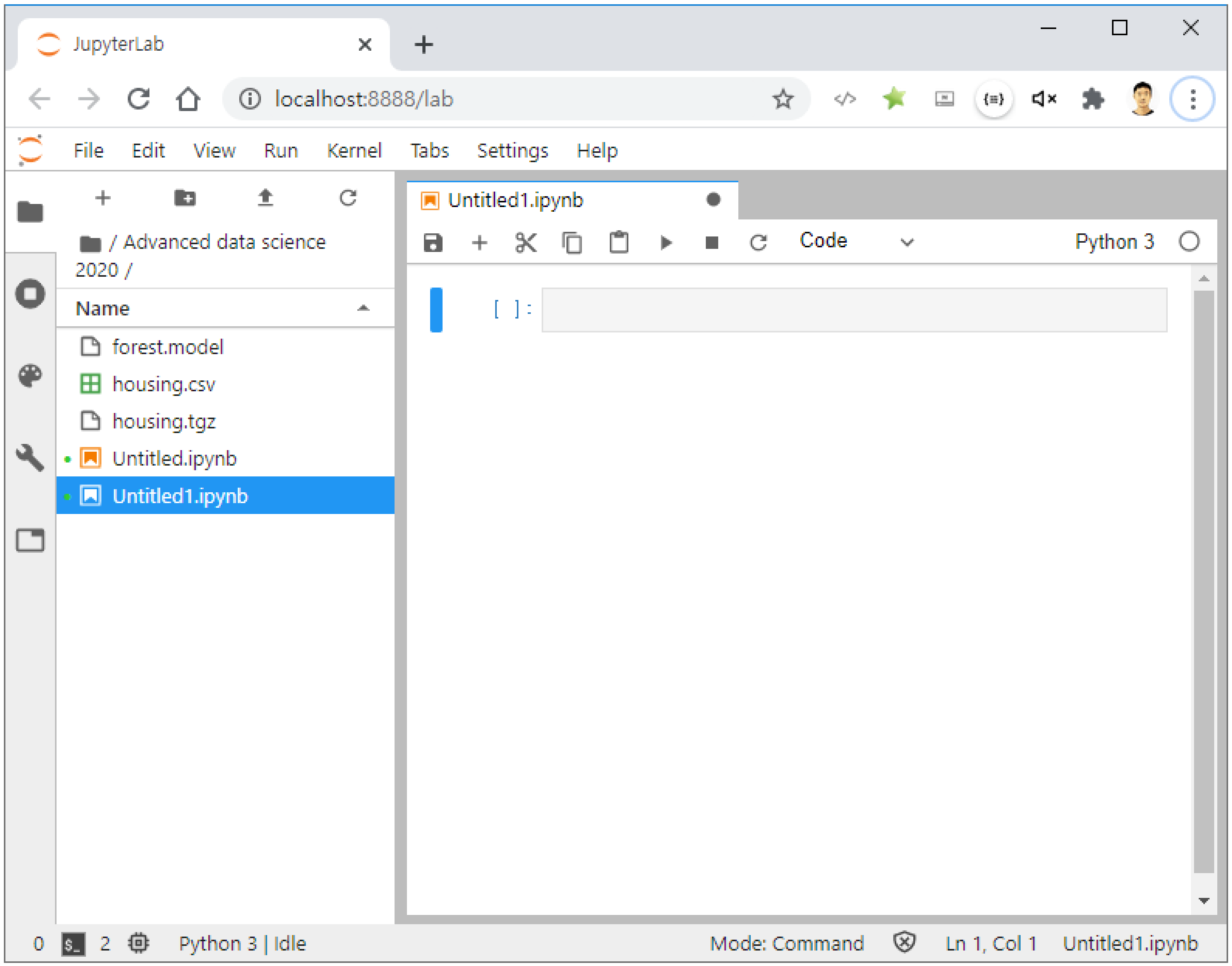


CRUD	HTTP verb
Create	POST
Read	**GET**
Update	PUT/PATCH
Delete	DELTE

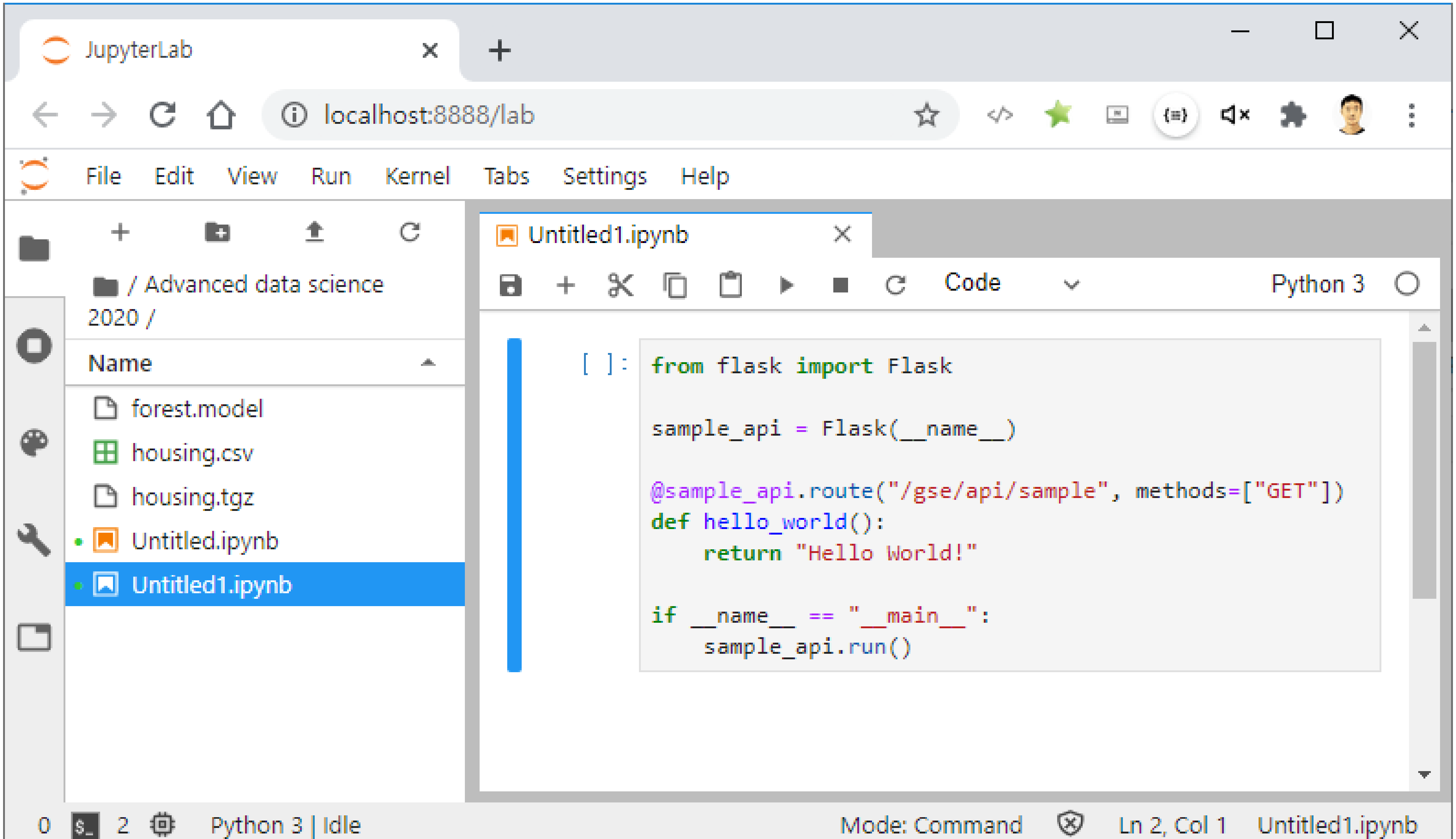
New notebook



New notebook



Sample api

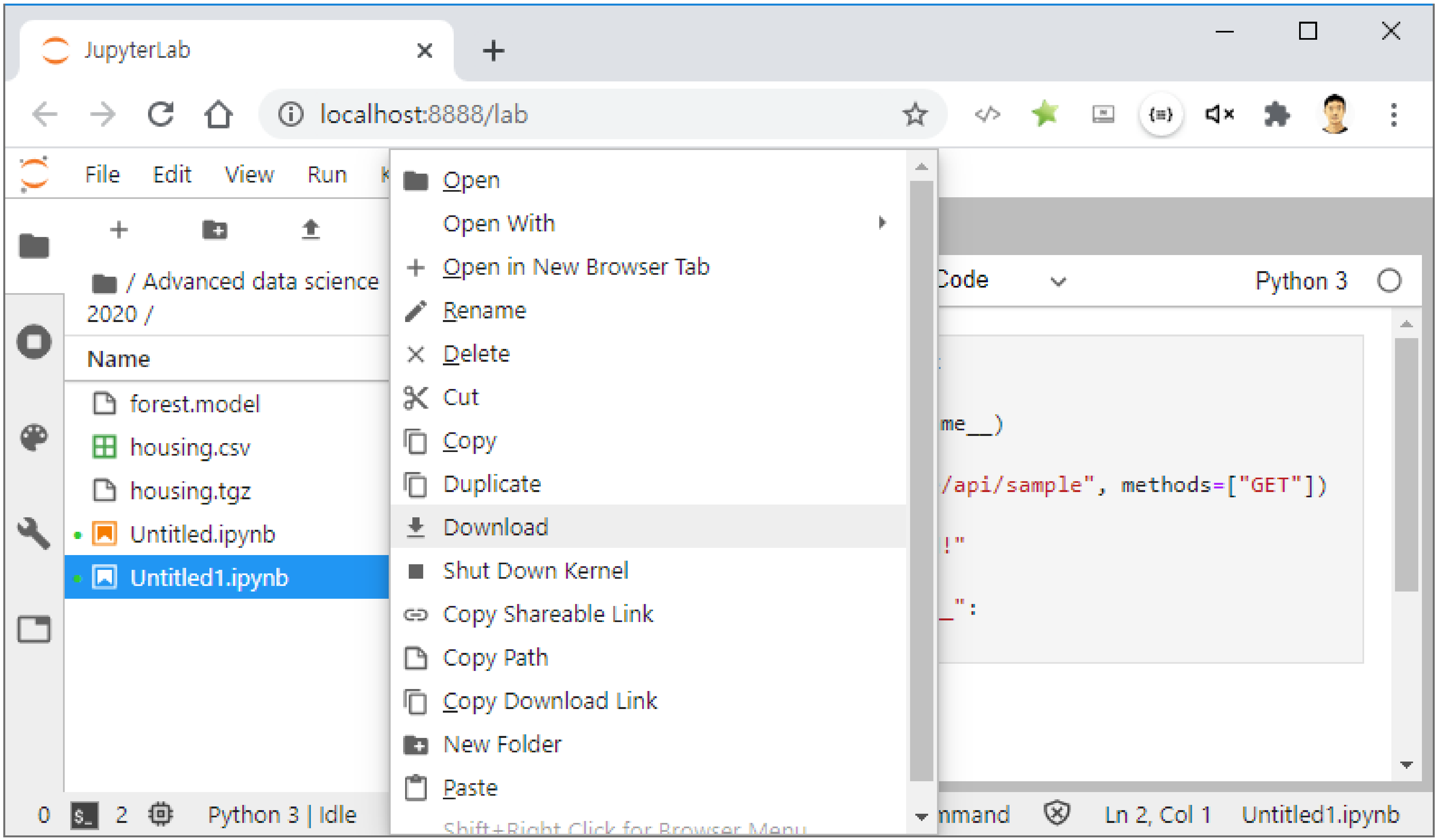

JupyterLab
localhost:8888/lab
File Edit View Run Kernel Tabs Settings Help
/ Advanced data science 2020 /
Name
forest.model
housing.csv
housing.tgz
Untitled.ipynb
Untitled1.ipynb
Untitled1.ipynb
Code
Python 3
[ ]:
from flask import Flask

sample_api = Flask(__name__)

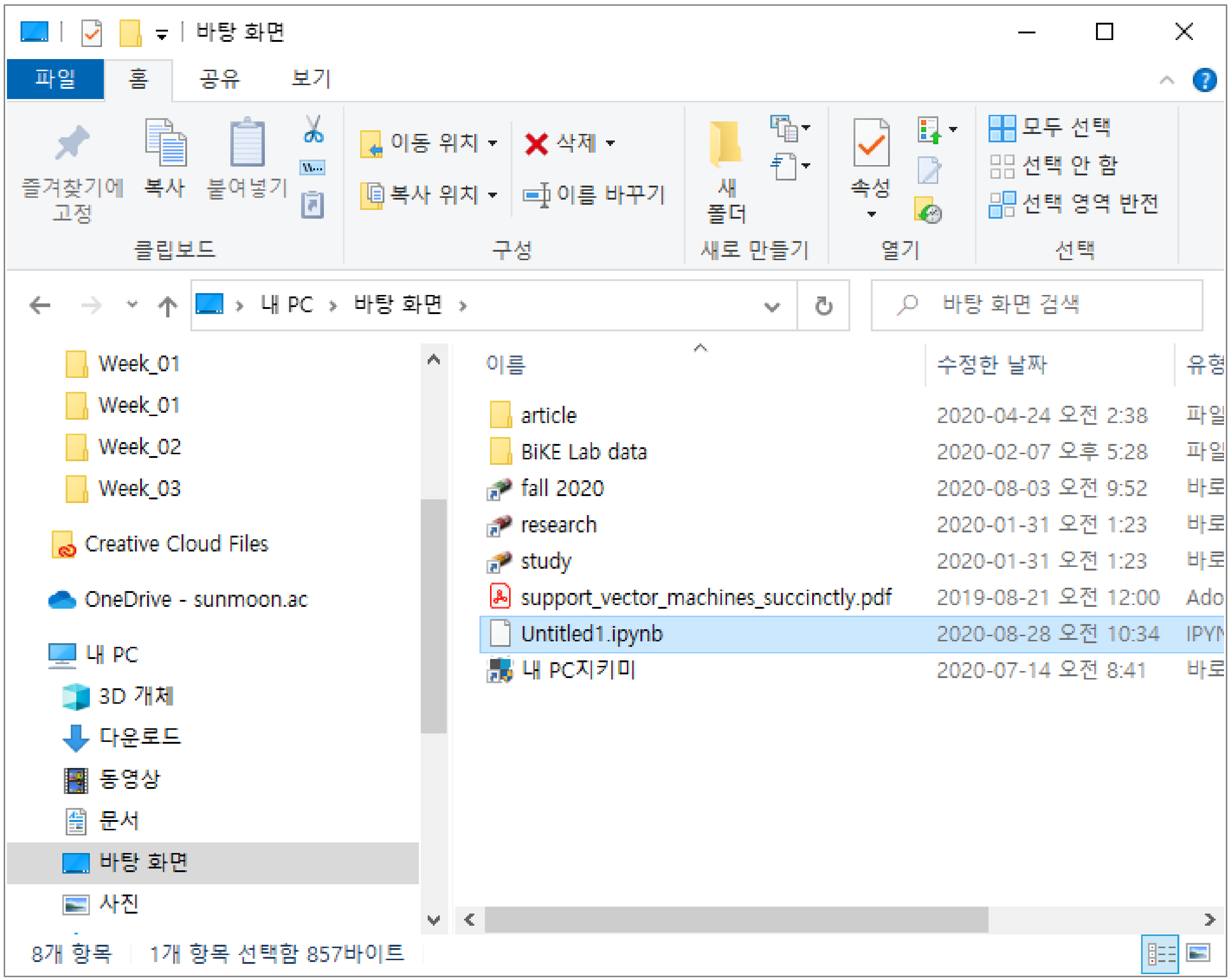
@sample_api.route("/gse/api/sample", methods=["GET"])
def hello_world():
    return "Hello World!"

if __name__ == "__main__":
    sample_api.run()
0 2 Python 3 | Idle
Mode: Command
Ln 2, Col 1
Untitled1.ipynb

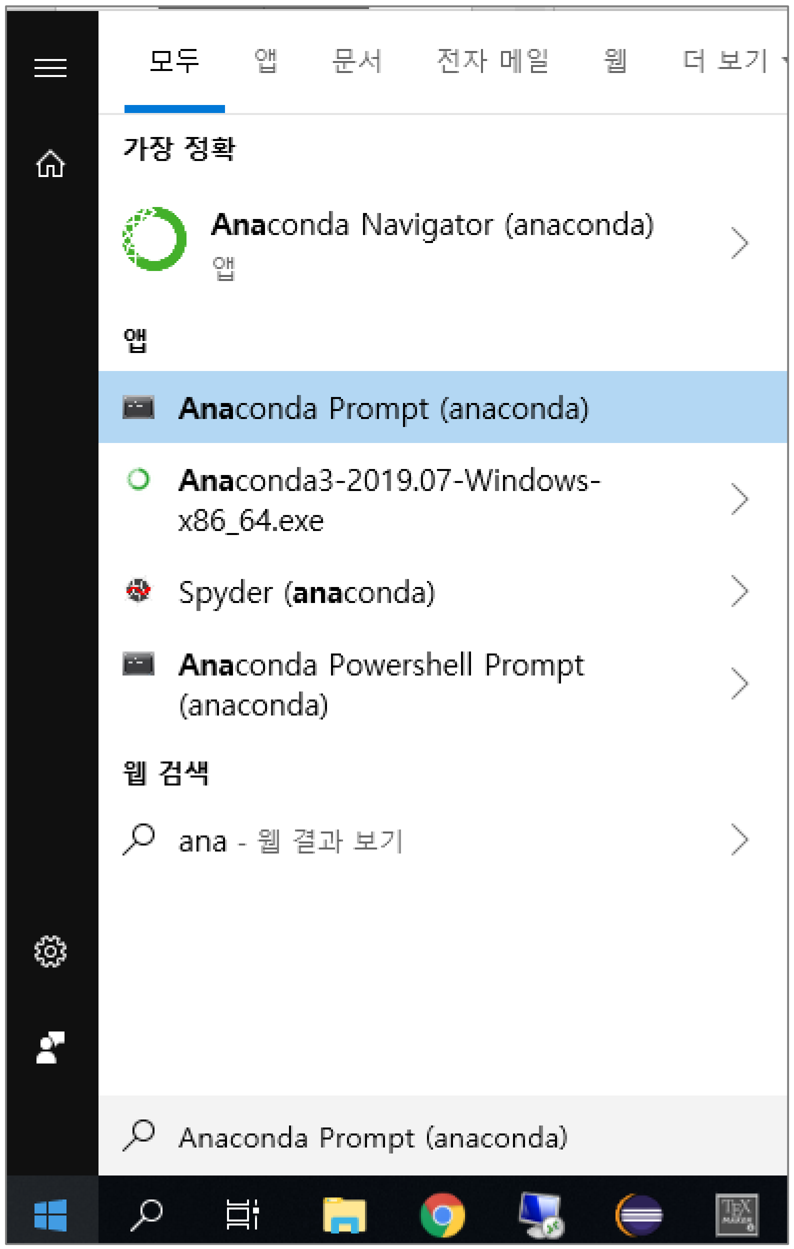
Download > Move to "Desktop"

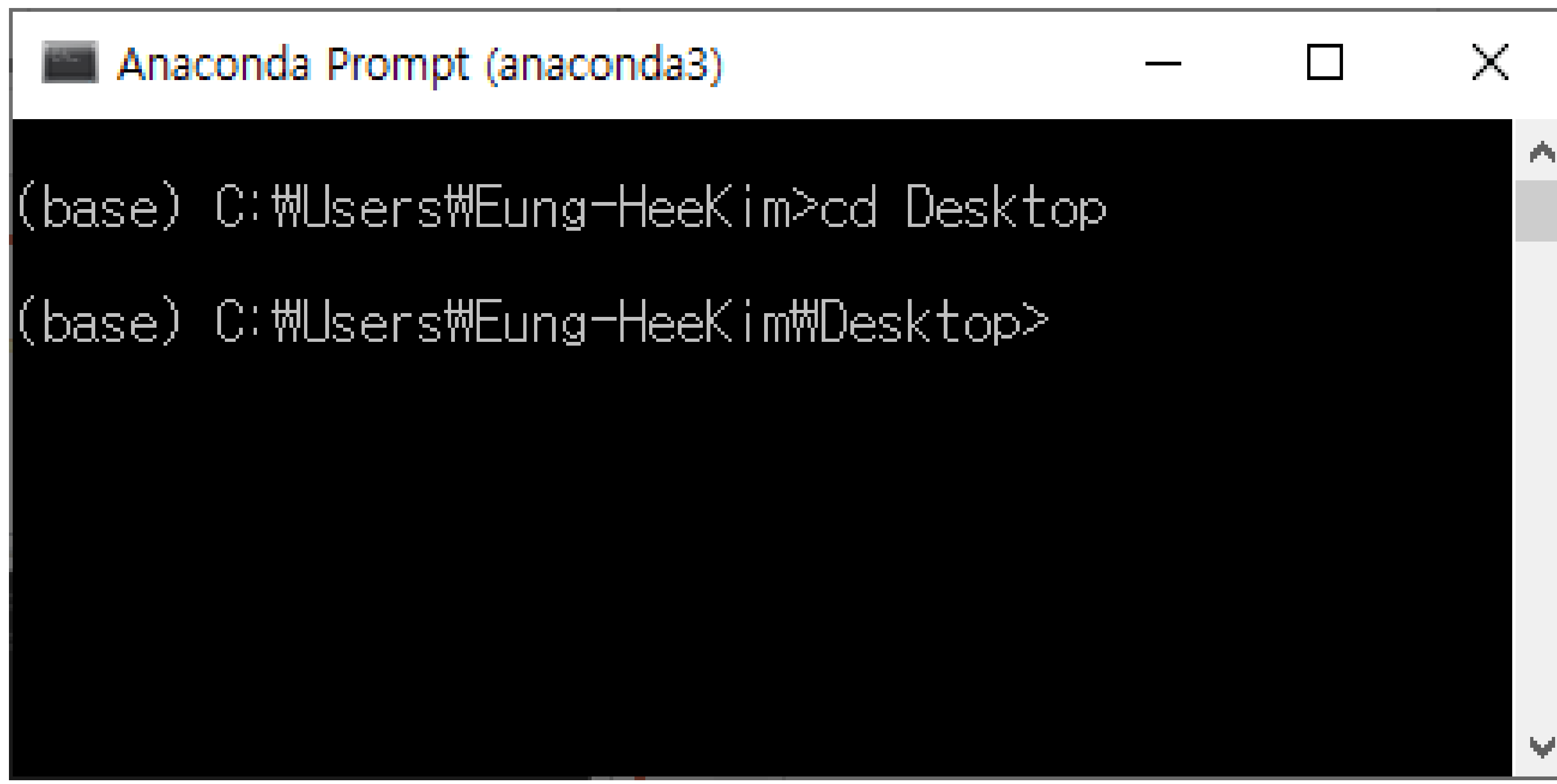


Download > Move to "Desktop"

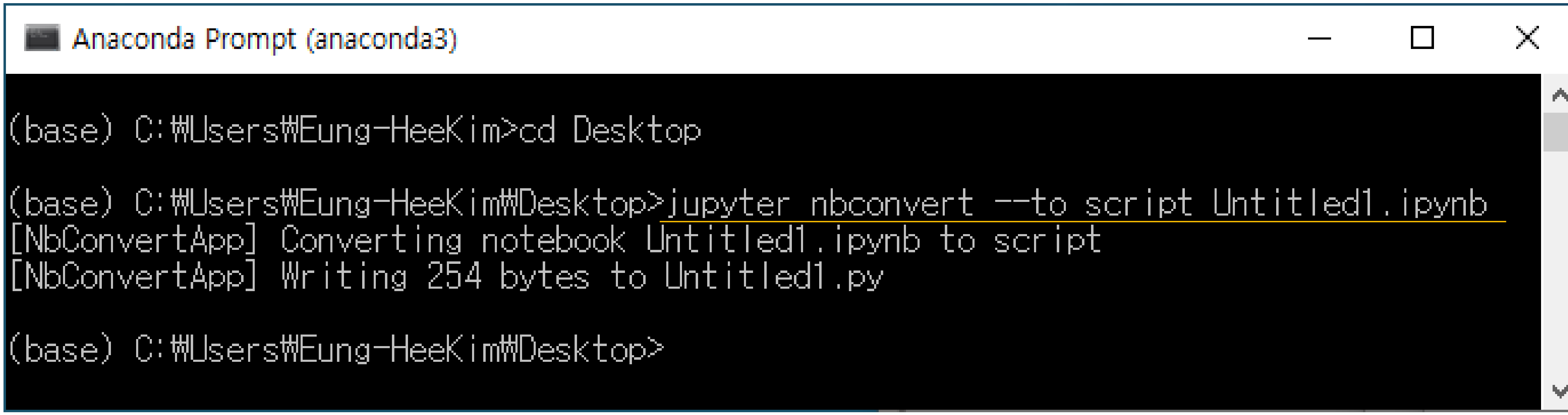


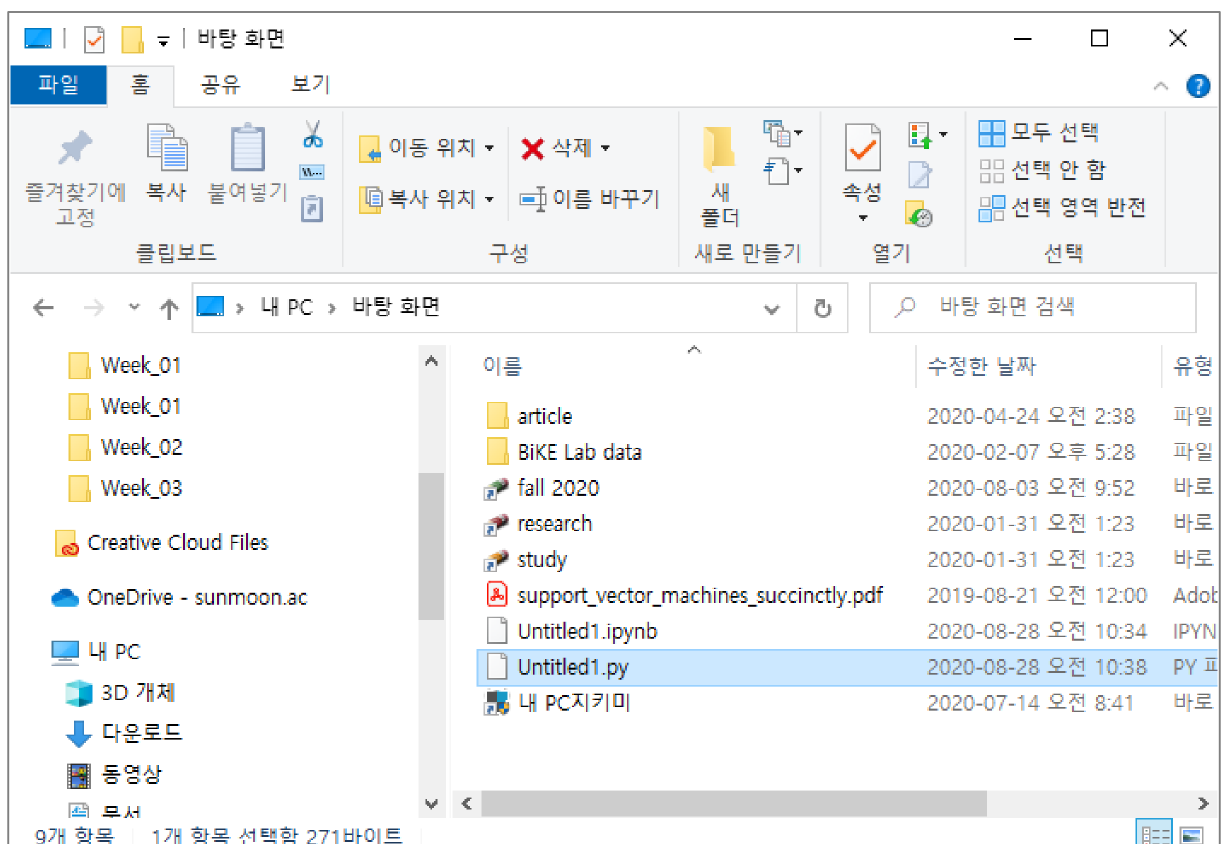
Anaconda prompt 실행 및 바탕화면으로 이동





Jupyter (.ipynb) to Python (.py)



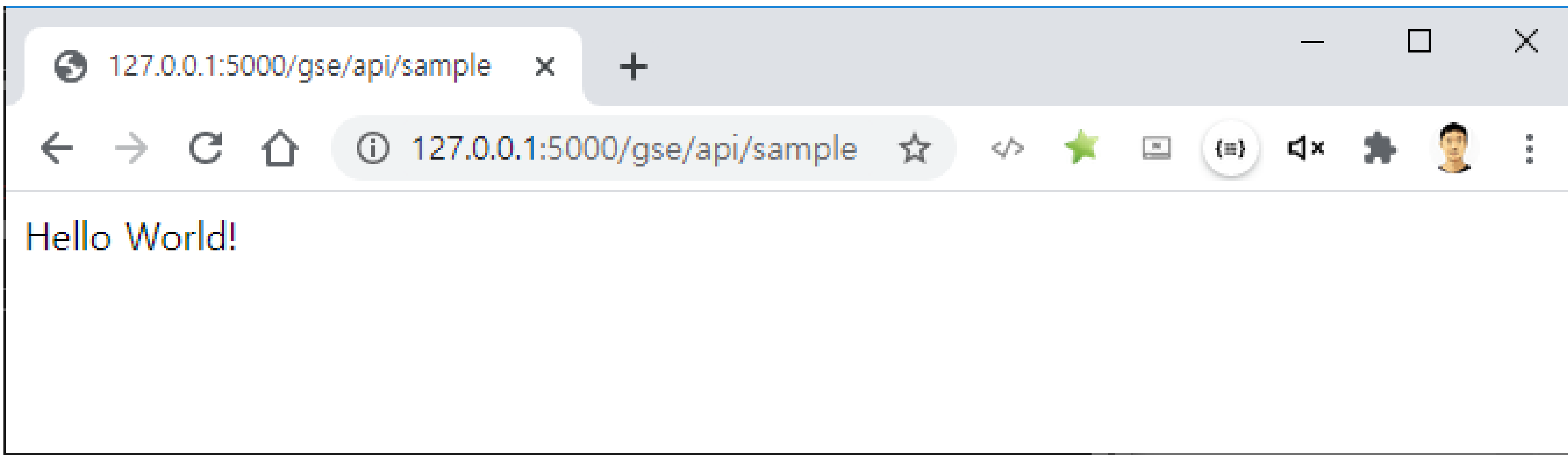


Untitled1.py 실행

Anaconda Prompt (anaconda3) - python Untitled1.py

```
(base) C:₩Users₩Eung-HeeKim>cd Desktop

(base) C:₩Users₩Eung-HeeKim₩Desktop>jupyter nbconvert --to script Untitled1.ipynb
[NbConvertApp] Converting notebook Untitled1.ipynb to script
[NbConvertApp] Writing 254 bytes to Untitled1.py

(base) C:₩Users₩Eung-HeeKim₩Desktop>python Untitled1.py
 * Serving Flask app "Untitled1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

REST API 호출


127.0.0.1:5000/gse/api/sample
127.0.0.1:5000/gse/api/sample
Hello World!

Untitled1.py 수정

*Untitled1.py - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```python
#!/usr/bin/env python
# coding: utf-8

# In[ ]:


from flask import Flask

sample_api = Flask(__name__)

@sample_api.route("/gse/api/sample", methods=["GET"])
def hello_world():
    answer = "<html><body><h1>Hi</h1></body></html>"
    return answer

if __name__ == "__main__":
    sample_api.run()
```

Ln 7, Col 24 | 100% | Windows (CRLF) | UTF-8

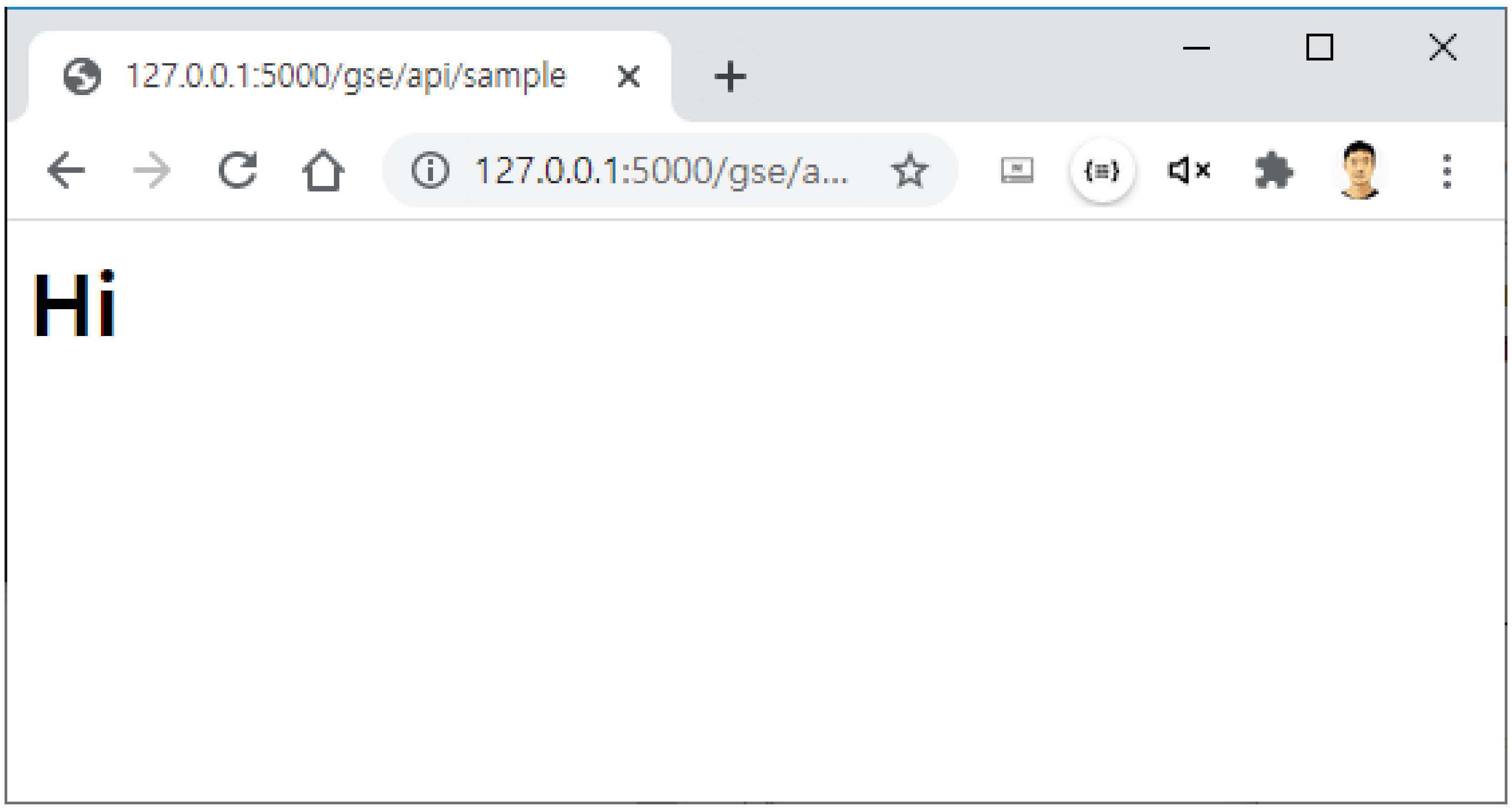
Untiled1.py 재실행

CTRL+C → python Untitled1.py

Anaconda Prompt (anaconda3) - python Untitled1.py

```
[NbConvertApp] Writing 254 bytes to Untitled1.py

(base) C:₩Users₩Eung-HeeKim₩Desktop>python Untitled1.py
 * Serving Flask app "Untitled1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

(base) C:₩Users₩Eung-HeeKim₩Desktop>python Untitled1.py
 * Serving Flask app "Untitled1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

REST API 재호출



매개변수(Parameter) 처리하기

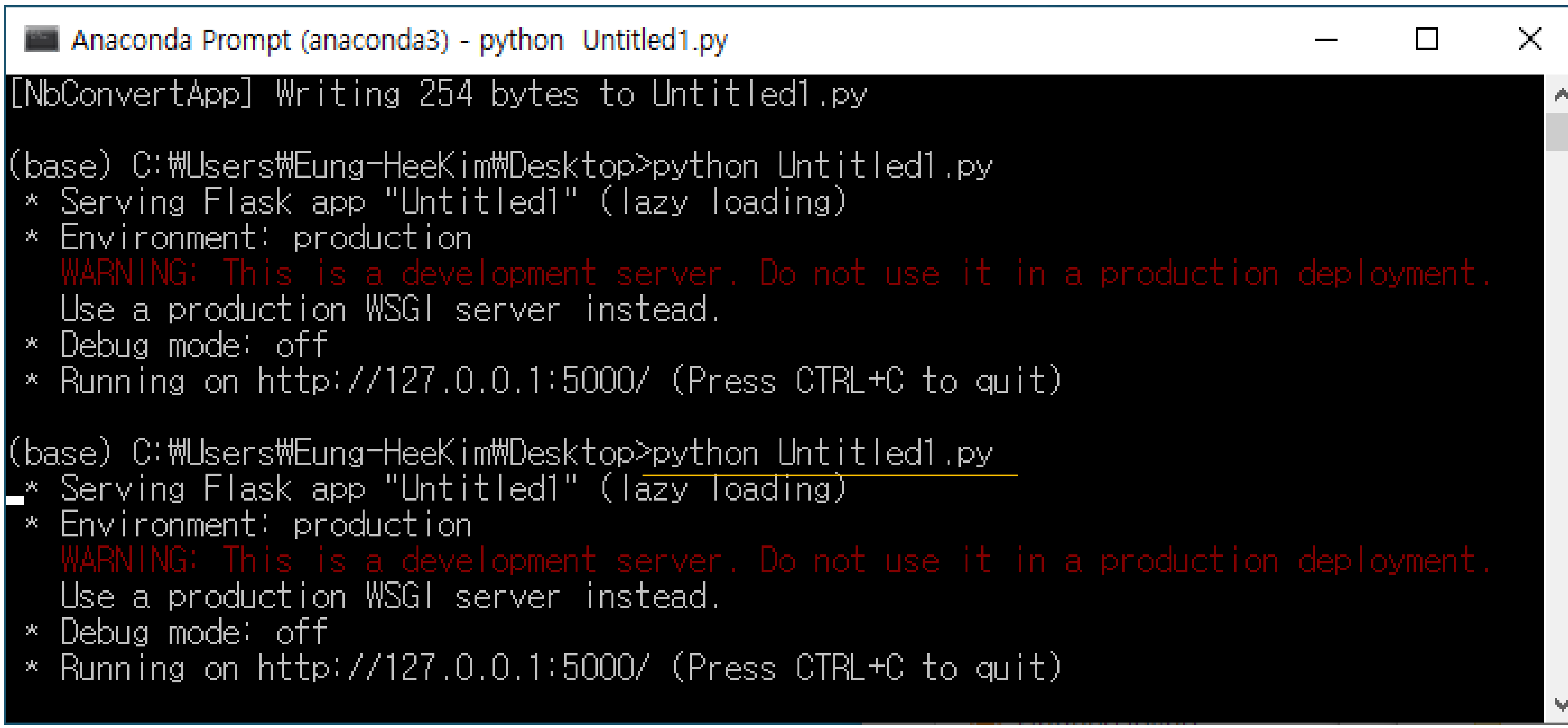
*Untitled1.py - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

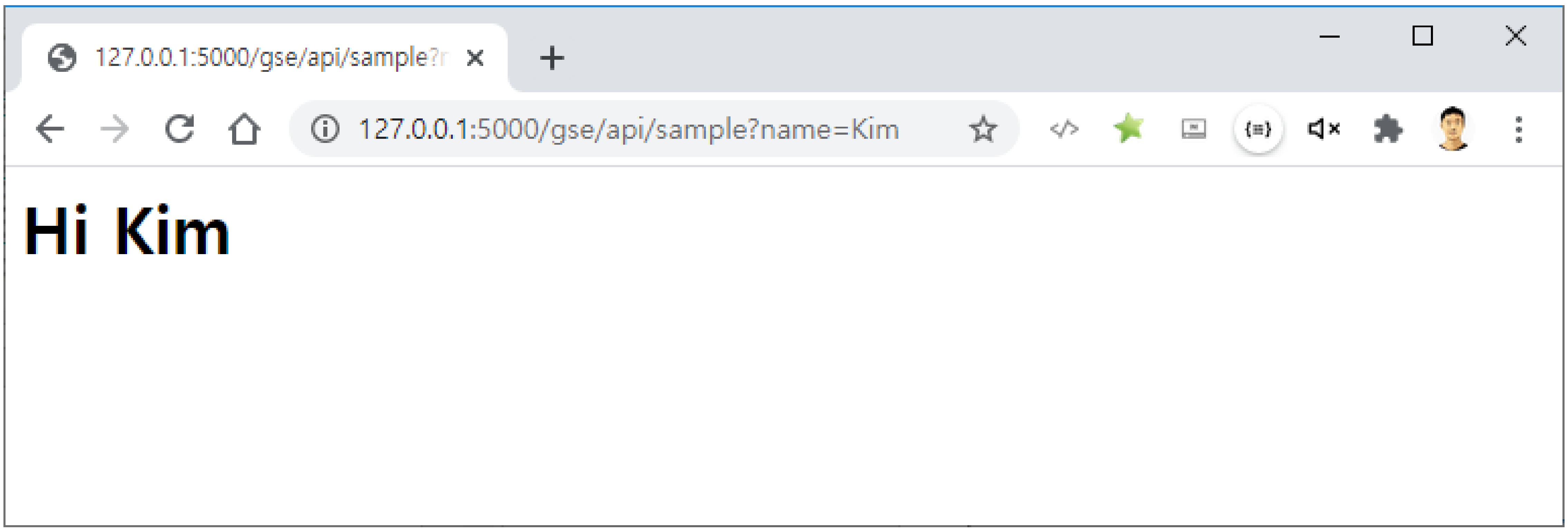
```python
# In[ ]:


from flask import Flask
from flask import request

sample_api = Flask(__name__)

@sample_api.route("/gse/api/sample", methods=["GET"])
def hello_world():
    answer = "<html><body><h1>Hi"
    if "name" in request.args:
        answer += " " + request.args["name"]
    answer += "</h1></body></html>"
    return answer

if __name__ == "__main__":
    sample_api.run()
```

Ln 23, Col 1 | 100% | Windows (CRLF) | UTF-8

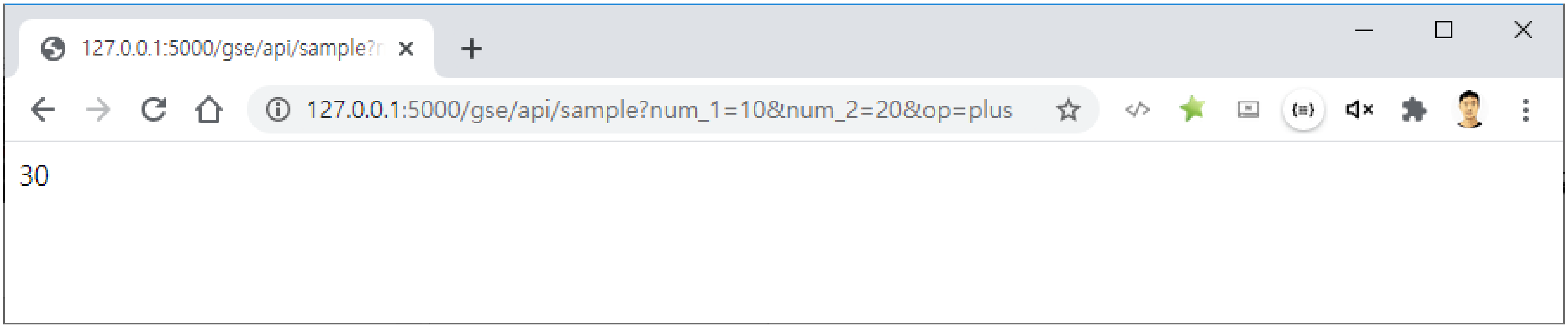
Untitled.py 재실행

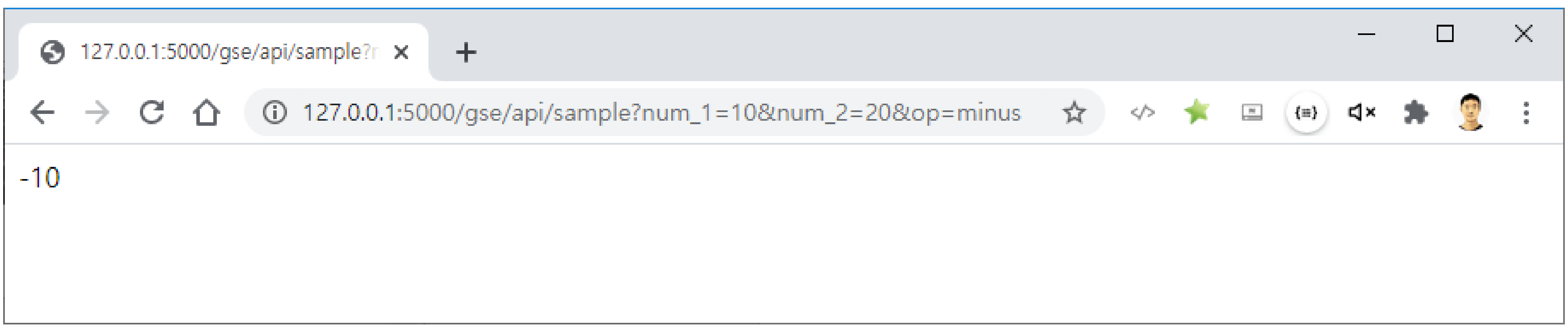


REST API 재호출


127.0.0.1:5000/gse/api/sample?r
127.0.0.1:5000/gse/api/sample?name=Kim
Hi Kim

덧셈 & 뺄셈 REST API 만들기

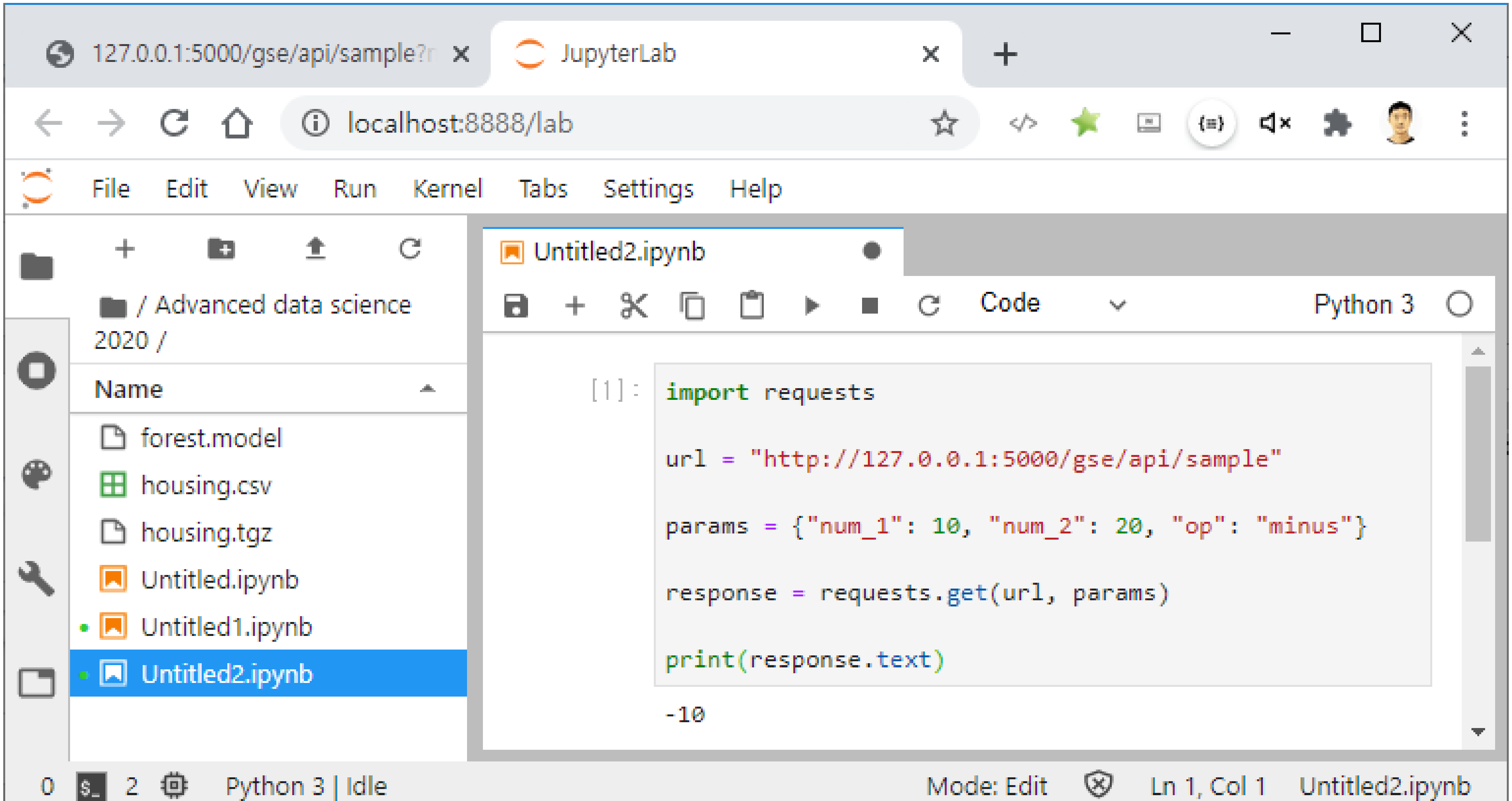




덧셈 & 뺄셈 REST API 만들기

```
@sample_api.route('/gse/api/sample/', methods=['GET'])
def hello_world( ):
    result = 0
    num_1 = int(request.args["num_1"])
    num_2 = int(request.args["num_2"])
    op = request.args["op"]
    if op == "plus":
        result = num_1 + num_2
    elif op == "minus":
        result = num_1 - num_2
    return str(result)
```

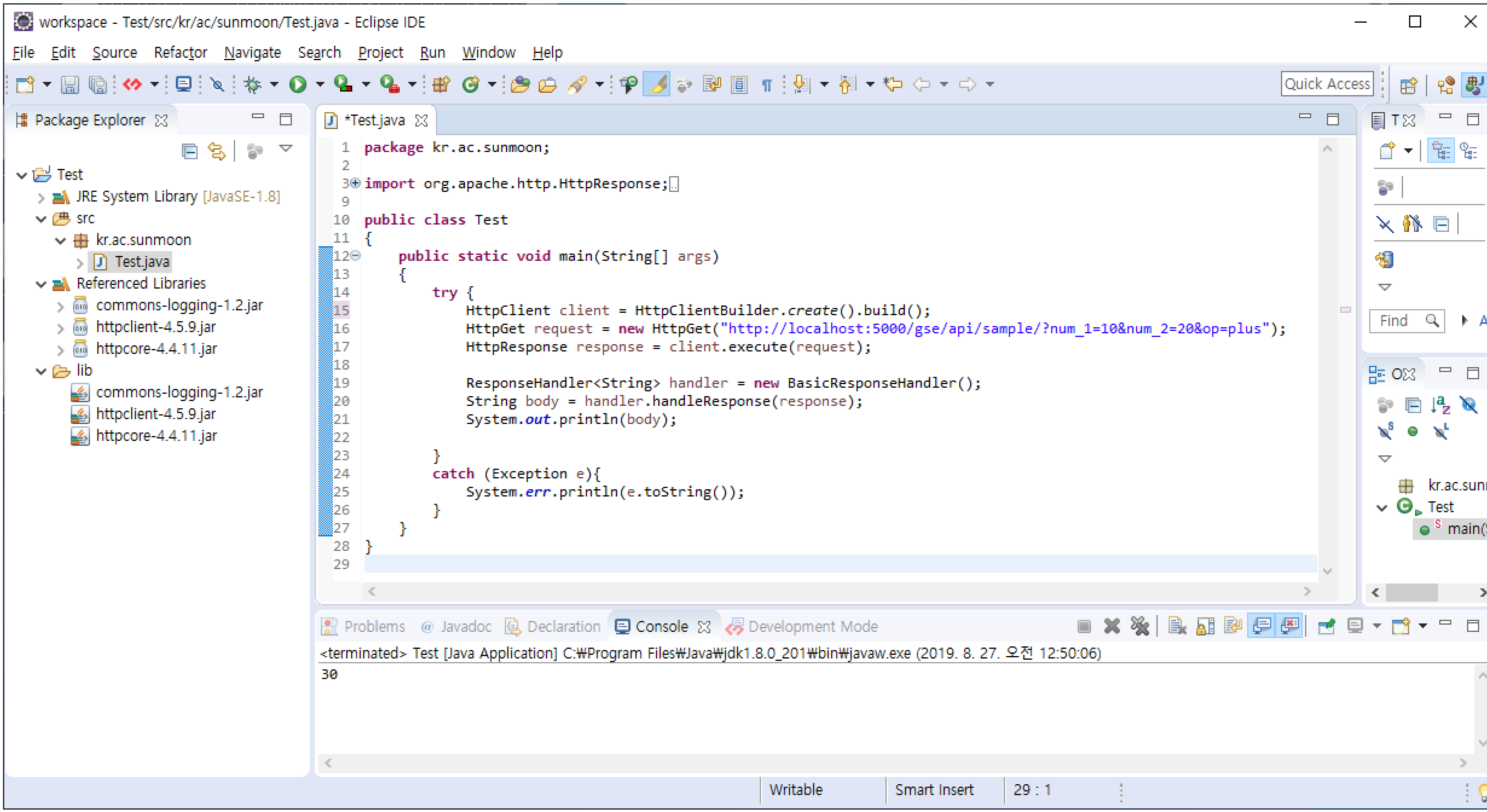
REST API 호출 in Python



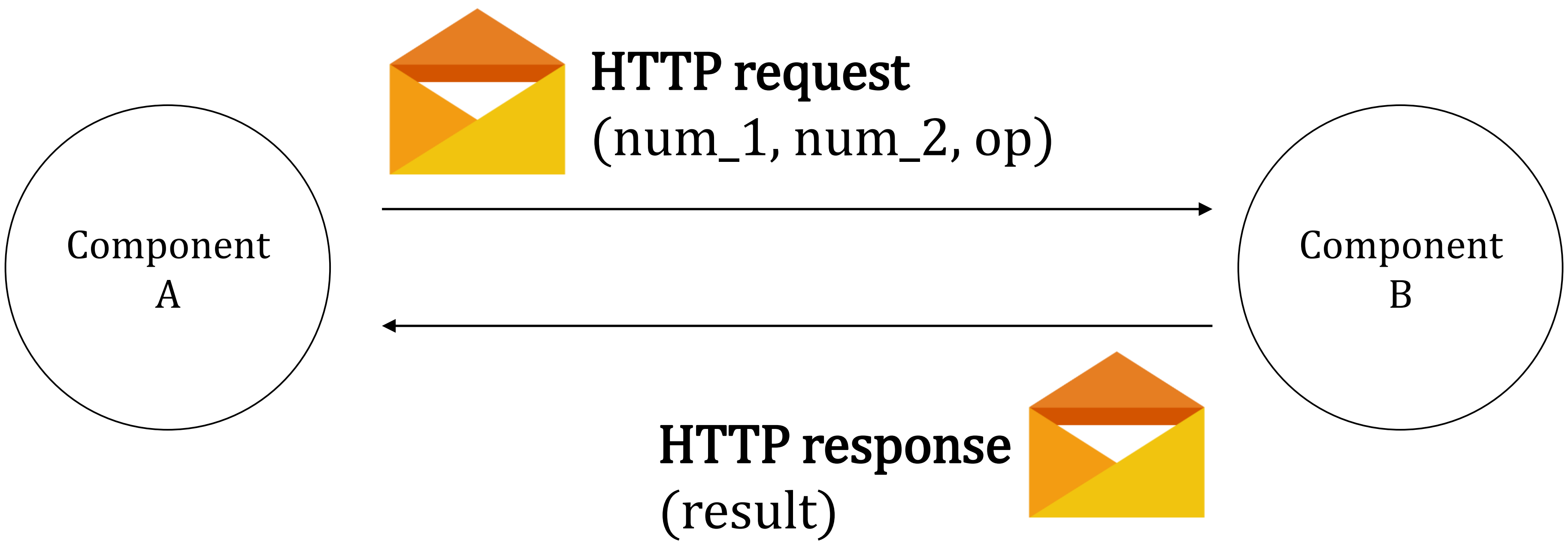
REST API 호출 in Java

- Eclipse 실행
- Java project 생성
- kr.ac.sunmoon 패키지 생성
- lib 폴더 생성
- e-강의동 > 데이터사이언스응용 > 02주차 강의자료
  - 3개의 jar 파일 다운로드 및 lib 폴더에 복사 → build path에 추가
  - Test.java 파일 다운로드 및 kr.ac.sunmoon 패키지에 복사 → 실행

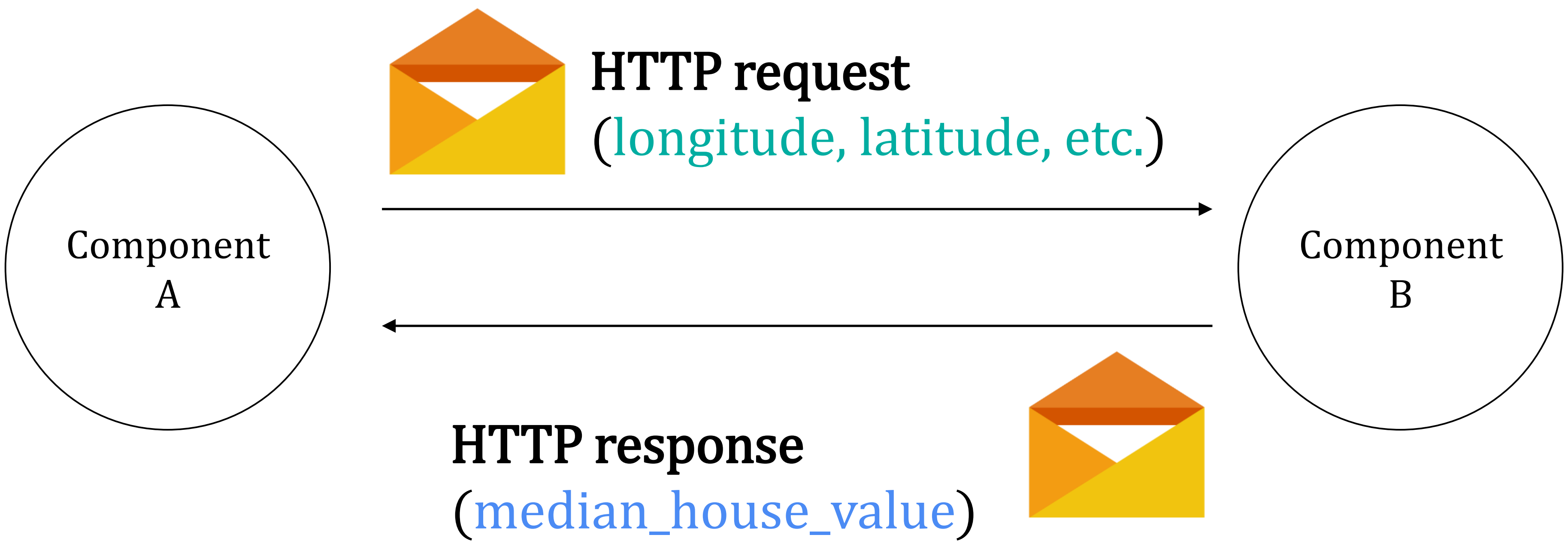
REST API 호출 in Java



What if..


HTTP request
(num_1, num_2, op)
Component
A
Component
B
HTTP response
(result)

What if..


HTTP request
(longitude, latitude, etc.)
Component
A
Component
B
HTTP response
(median_house_value)

Thank you