

# Machine Learning Pipeline

실습 코드 : <https://bit.ly/3MOtZr6>

---

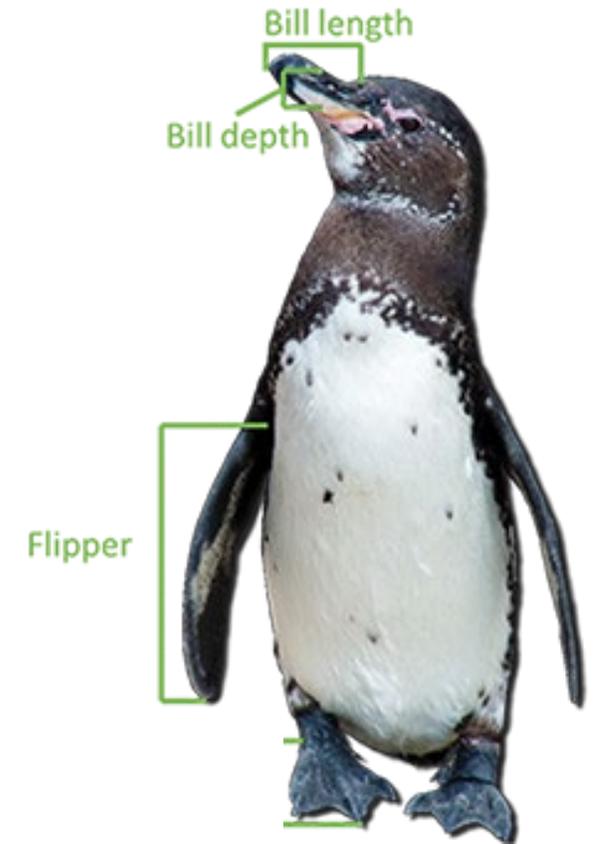
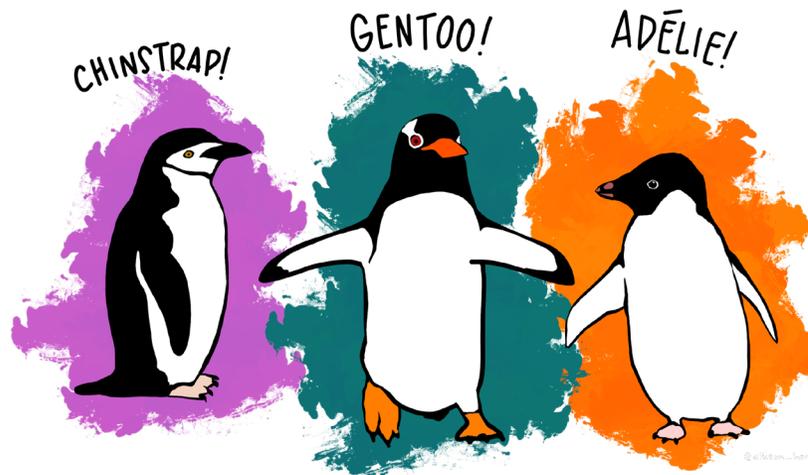
2021.05.24.

한국에너지기술연구원 계산과학연구실

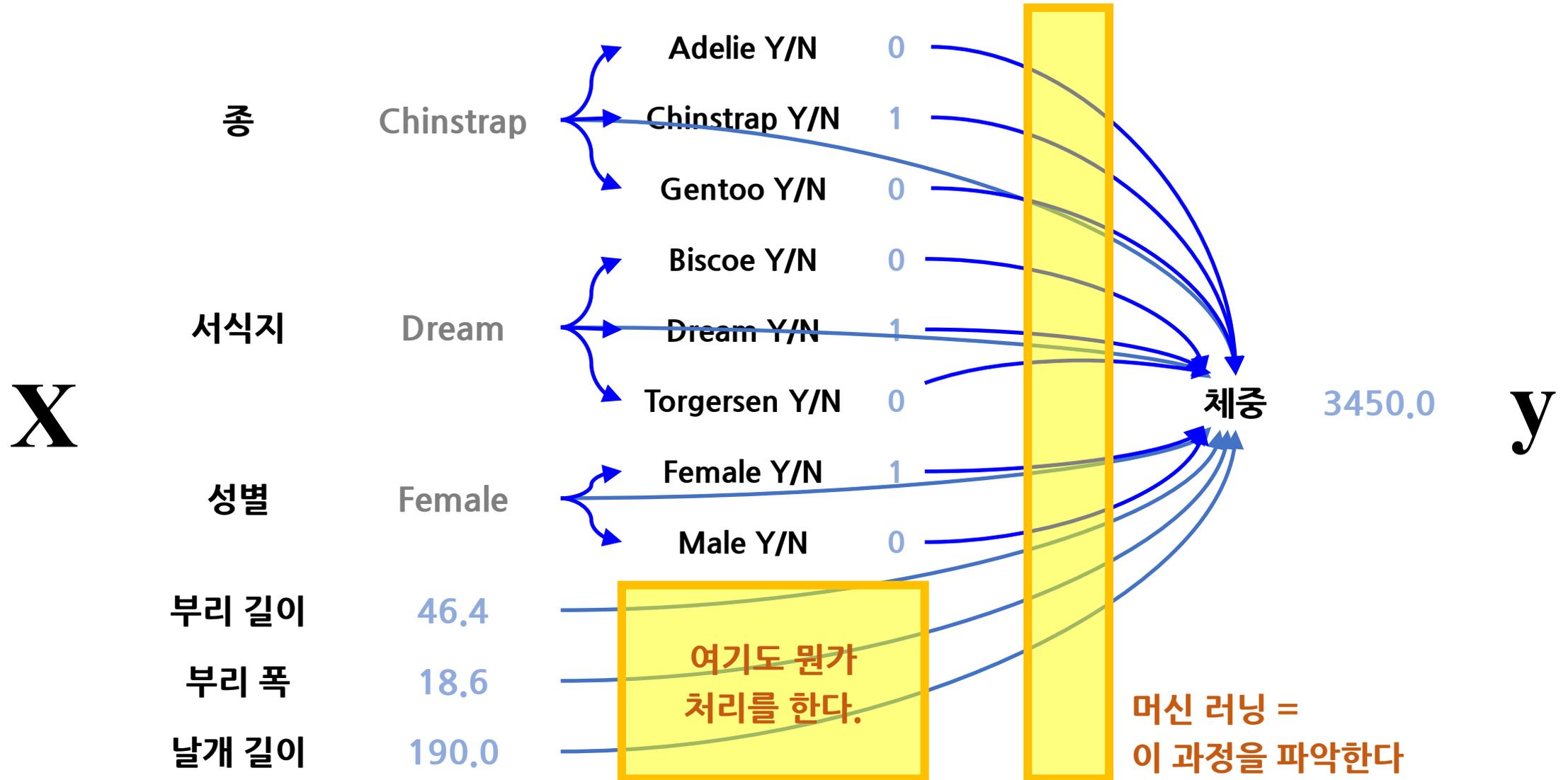
이제현

# 머신 러닝 예제

- 목표 : 펭귄 체중 예측
- X 인자 : 펭귄 서식지 3곳, 종 3가지, 성별, 신체 치수 3가지



# 머신 러닝 학습 과정



# 왜 독립 변수를 $X$ , 종속 변수를 $y$ 라고 부를까요?

- 행렬은 대문자, 벡터는 소문자

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	sex
0	Adelie	Torgersen	39.1	18.7	181.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	Female
4	Adelie	Torgersen	36.7	19.3	193.0	Female
5	Adelie	Torgersen	39.3	20.6	190.0	Male
6	Adelie	Torgersen	38.9	17.8	181.0	Female
7	Adelie	Torgersen	39.2	19.6	195.0	Male
12	Adelie	Torgersen	41.1	17.6	182.0	Female
13	Adelie	Torgersen	38.6	21.2	191.0	Male
14	Adelie	Torgersen	34.6	21.1	198.0	Male
15	Adelie	Torgersen	36.6	17.8	185.0	Female
16	Adelie	Torgersen	38.7	19.0	195.0	Female
17	Adelie	Torgersen	42.5	20.7	197.0	Male
18	Adelie	Torgersen	34.4	18.4	184.0	Female
19	Adelie	Torgersen	46.0	21.5	194.0	Male

$X$

	body_mass_g
0	3750.0
1	3800.0
2	3250.0
4	3450.0
5	3650.0
6	3625.0
7	4675.0
12	3700.0
13	3800.0
14	4400.0
15	3700.0
16	3450.0
17	4500.0
18	3325.0
19	4200.0

$y$

# train set과 test set 분할

- 8:2로 나누겠습니다.
  - 80%로 학습하고 20%로 평가합니다.

```
1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
4 print(f"전체 데이터 수 = {X.shape[0]} ")
5 print(f"train set 데이터 수 = {X_train.shape[0]} ")
6 print(f"test set 데이터 수 = {X_test.shape[0]} ")
```

전체 데이터 수 = 333

train set 데이터 수 = 266

test set 데이터 수 = 67

# X vs y 관계 외에도 학습할 게 좀 많습니다

- 머신 러닝 과정 = 숫자 계산
- 문자로 작성된 범주형 데이터도 숫자로 변환: one-hot encoding

train set  
원본

```
1 X_train.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	sex
170	Chinstrap	Dream	46.4	18.6	190.0	Female
239	Gentoo	Biscoe	48.7	15.1	222.0	Male
214	Chinstrap	Dream	45.7	17.0	195.0	Female
6	Adelie	Torgersen	38.9	17.8	181.0	Female
38	Adelie	Dream	37.6	19.3	181.0	Female

get\_dummies

train set  
one-hot  
encoded

```
1 X_train_d = pd.get_dummies(X_train)
2 X_train_d.head()
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	species_Adelie	species_Chinstrap	species_Gentoo	island_Biscoe	island_Dream	island_Torgersen	sex_Female	sex_Male
170	46.4	18.6	190.0	0	1	0	0	1	0	1	0
239	48.7	15.1	222.0	0	0	1	1	0	0	0	1
214	45.7	17.0	195.0	0	1	0	0	1	0	1	0
6	38.9	17.8	181.0	1	0	0	0	0	1	1	0
38	37.6	19.3	181.0	1	0	0	0	1	0	1	0

# X vs y 관계 외에도 학습할 게 좀 많습니다

- 어떤 데이터가 어떻게 인코딩 되었는지 외워 줘야 합니다.
- `get_dummies`는 사용하지 않는 게 좋습니다.

train set  
one-hot  
encoded

```
1 X_train_d = pd.get_dummies(X_train)
2 X_train_d.head()
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	species_Adelie	species_Chinstrap	species_Gentoo	island_Biscoe	island_Dream	island_Torgersen	sex_Female	sex_Male
170	46.4	18.6	190.0	0	1	0	0	1	0	1	0
239	48.7	15.1	222.0	0	0	1	1	0	0	0	1
214	45.7	17.0	195.0	0	1	0	0	1	0	1	0
6	38.9	17.8	181.0	1	0	0	0	0	1	1	0
38	37.6	19.3	181.0	1	0	0	0	1	0	1	0

test set  
one-hot  
encoded

```
1 X_test_d = pd.get_dummies(X_test)
2 X_test_d.head()
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	species_Adelie	species_Gentoo	island_Biscoe	island_Torgersen	sex_Female	sex_Male
62	37.6	17.0	185.0	1	0	1	0	1	0
60	35.7	16.9	185.0	1	0	1	0	1	0
283	54.3	15.7	231.0	0	1	1	0	0	1
107	38.2	20.0	190.0	1	0	1	0	0	1
65	41.6	18.0	192.0	1	0	1	0	0	1

# Categorical Data 변환

- OneHotEncoder를 사용해서 변환을 학습하고 적용합니다.

```
1 # 라이브러리 사용
2 from sklearn.preprocessing import OneHotEncoder
3
4 # 예제 데이터 1
5 data_sample1 = [["Apple"],
6                 ["Banana"],
7                 ["Cherry"]]
8
9 # 예제 데이터 1 OneHotEncoder 학습 & 변환
10 ohe = OneHotEncoder()
11 s_ohe1 = ohe.fit_transform(data_sample1)
12 print(s_ohe1.toarray())
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
1 # 예제 데이터 2
2 data_sample2 = [["Banana"]]
3
4 # 예제 데이터 2 OneHotEncoder 변환
5 s_ohe2 = ohe.transform(data_sample2)
6 print(s_ohe2.toarray())
```

```
[[0. 1. 0.]]
```

# Categorical Data 변환

- 학습 데이터, 평가 데이터도 변환합니다.
- 학습 데이터로부터 데이터 분포를 학습해서 평가 데이터에 적용합니다.

```
1 # 펭귄 데이터
2 cols_category = ["species", "island", "sex"]
3 X_train[cols_category].head(5)
```

	species	island	sex
97	Adelie	Dream	Male
133	Adelie	Dream	Male
159	Chinstrap	Dream	Male
185	Chinstrap	Dream	Male
274	Gentoo	Biscoe	Female



```
1 ohe = OneHotEncoder()
2 X_train_ohe = ohe.fit_transform(X_train[cols_category])
3 print(X_train_ohe.toarray()[:5])
```

```
[[1. 0. 0. 0. 1. 0. 0. 1.]
 [1. 0. 0. 0. 1. 0. 0. 1.]
 [0. 1. 0. 0. 1. 0. 0. 1.]
 [0. 1. 0. 0. 1. 0. 0. 1.]
 [0. 0. 1. 1. 0. 0. 1. 0.]]
```

```
1 X_test_ohe = ohe.transform(X_test[cols_category])
2 print(X_test_ohe.toarray()[:5])
```

```
[[1. 0. 0. 0. 1. 0. 0. 1.]
 [1. 0. 0. 1. 0. 0. 1. 0.]
 [0. 0. 1. 1. 0. 0. 0. 1.]
 [0. 0. 1. 1. 0. 0. 0. 1.]
 [0. 0. 1. 1. 0. 0. 1. 0.]]
```

# Numerical Data 변환

- 숫자 데이터도 변환할 수 있습니다.
- 이게 뭐 하는 걸까요? 그리고 왜 이러는 걸까요?

```
1 cols_numerical = ["bill_length_mm", "bill_depth_mm", "flipper_length_mm"]  
2 X_train[cols_numerical].head(5)
```

	bill_length_mm	bill_depth_mm	flipper_length_mm
97	40.3	18.5	196.0
133	37.5	18.5	199.0
159	51.3	18.2	197.0
185	51.0	18.8	203.0
274	46.5	14.4	217.0



```
1 from sklearn.preprocessing import StandardScaler  
2  
3 stdscaler = StandardScaler()  
4 X_train_std = stdscaler.fit_transform(X_train[cols_numerical])  
5 print(X_train_std[:5])
```

```
[[-0.62800518  0.63787293 -0.30961341]  
 [-1.12406866  0.63787293 -0.09345287]  
 [ 1.32081564  0.48062159 -0.2375599 ]  
 [ 1.26766598  0.79512428  0.19476119]  
 [ 0.4704211  -1.51122877  1.2035104  ]]
```

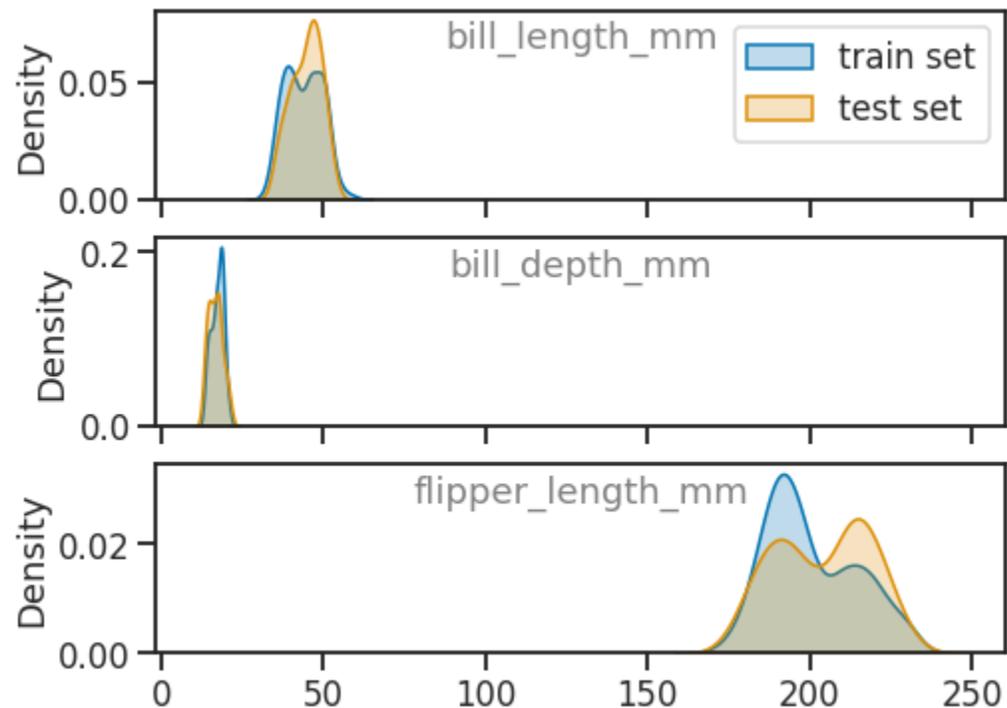
```
1 X_test_std = stdscaler.transform(X_test[cols_numerical])  
2 print(X_test_std[:5])
```

```
[[-0.75202105  0.42820447 -1.03014856]  
 [-0.73430449  0.21853602 -0.52577395]  
 [ 1.0727839  -0.62013782  0.91529634]  
 [ 0.80703561 -1.40639454  0.91529634]  
 [ 0.22238936 -1.45881165  1.05940337]]
```

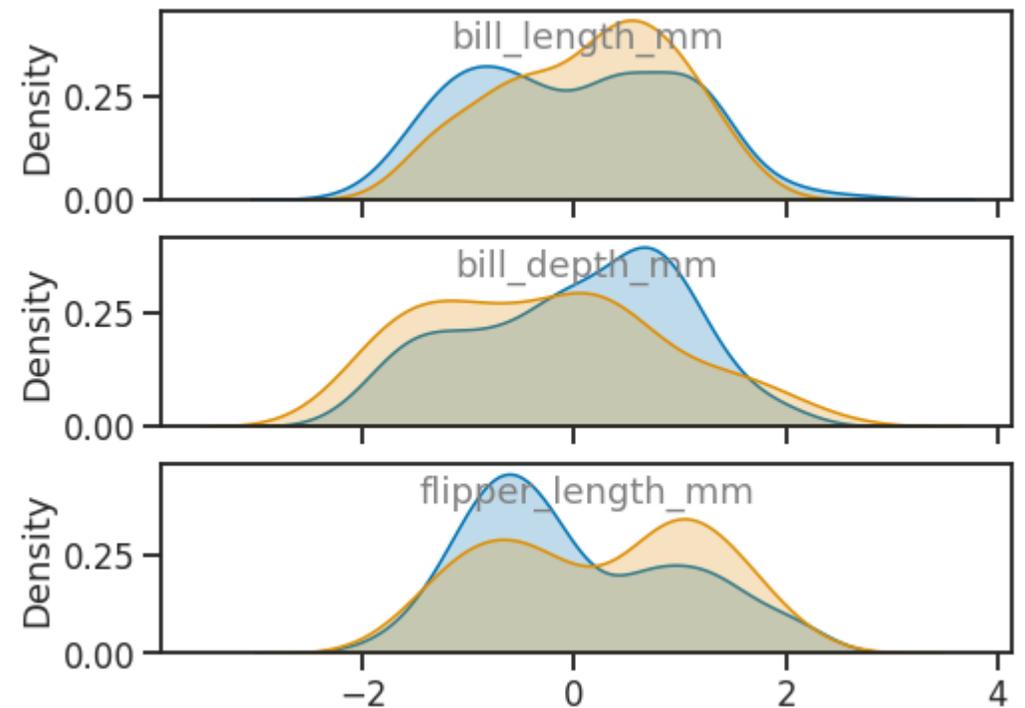
# Numerical Data 변환

- 데이터의 평균과 표준편차를 일치시킵니다.

Standard Scaler 적용 전



Standard Scaler 적용 후



# 머신 러닝 Machine Learning

- 선형 모델에서 인자간 기여도를 산출할 때,
- 신경망 모델에서 수렴을 돕기 위해 scaling이 필요합니다.

## 선형 모델 Linear Model

$$y_i = \beta_0 + \sum_{j=1}^p x_{ij} \beta_j + \varepsilon_i$$

number of features  $p$

response  $y_i$

global intercept  $\beta_0$

feature  $j$  of observation  $i$   $x_{ij}$

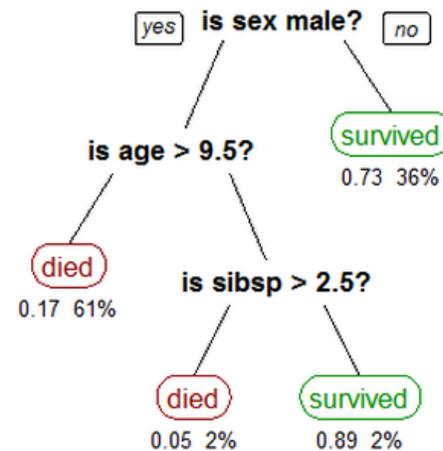
coefficient for feature  $j$   $\beta_j$

noise term  $\varepsilon_i$

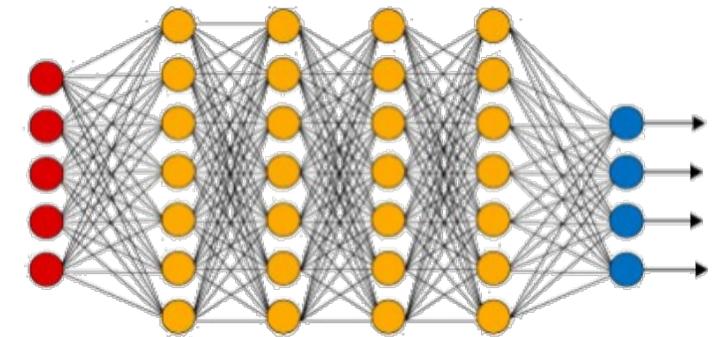
independence assumption  $\varepsilon_i \stackrel{iid}{\sim} \mathbf{N}(0, \sigma^2)$

noise level  $\sigma^2$

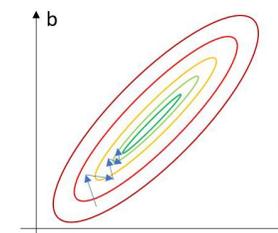
## 트리 모델 Tree Model



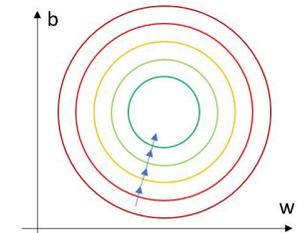
## 신경망 모델 Neural Network



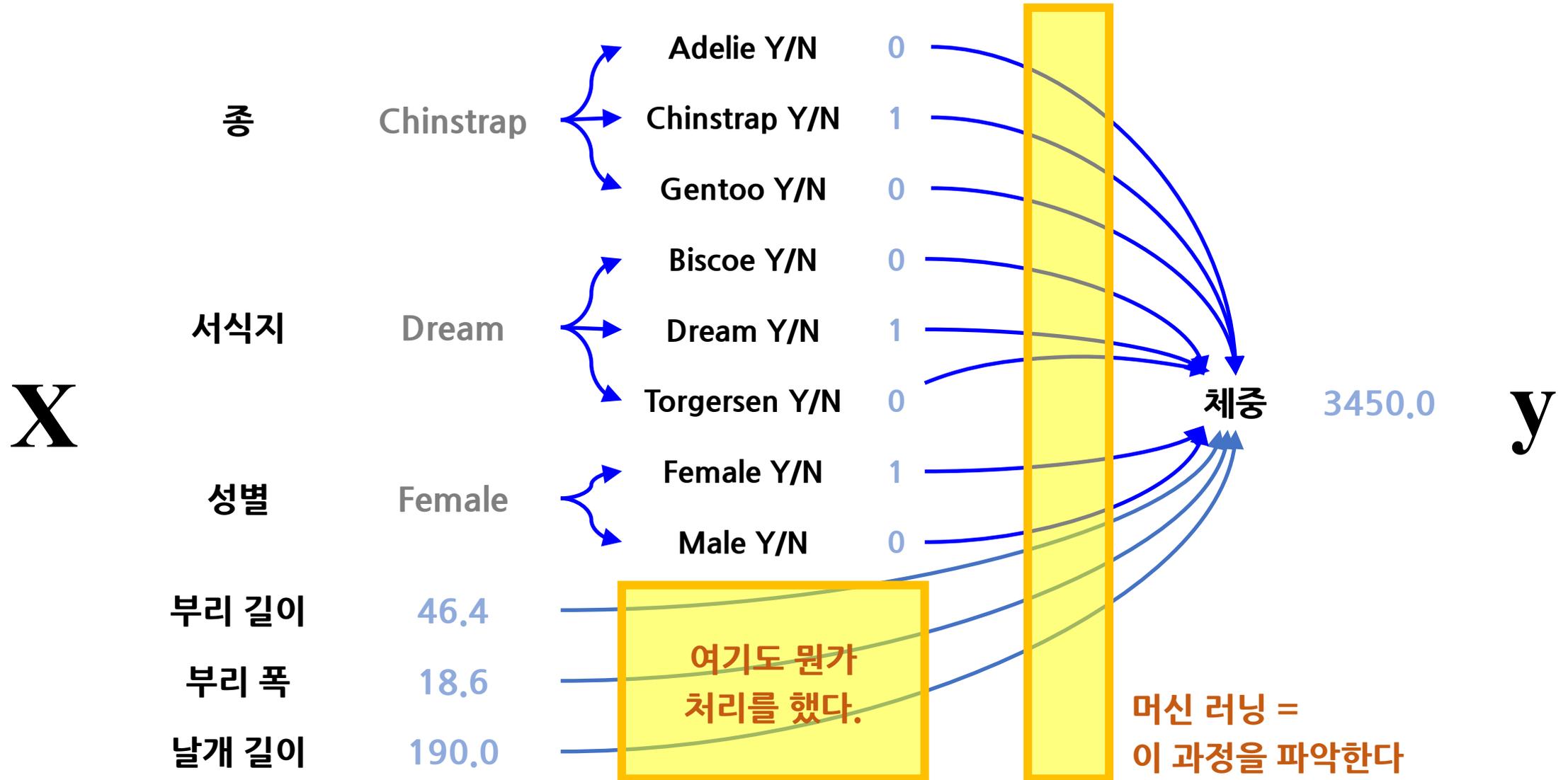
Unnormalized:



Normalized:



# 각각 처리한 두 데이터를 결합



# 각각 처리한 두 데이터를 결합

```
1 # categorical data  
2 X_train_ohc.toarray().shape
```

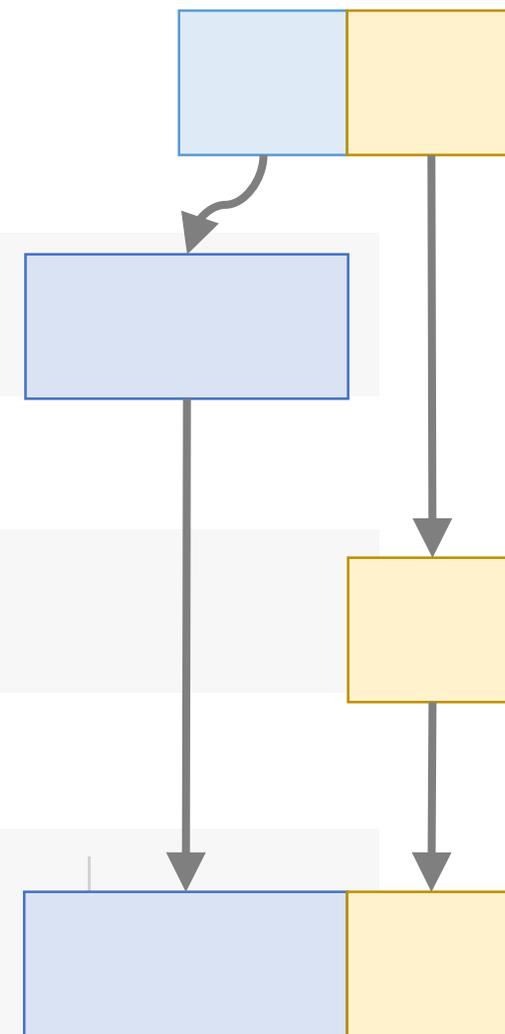
(266, 8)

```
1 # numerical data  
2 X_train_std.shape
```

(266, 3)

```
1 # 두 데이터 결합  
2  
3 import numpy as np  
4  
5 X_train_c = np.concatenate([X_train_ohc.toarray(), X_train_std], axis=1)  
6 X_test_c = np.concatenate([X_test_ohc.toarray(), X_test_std], axis=1)  
7 print(X_train_c.shape)
```

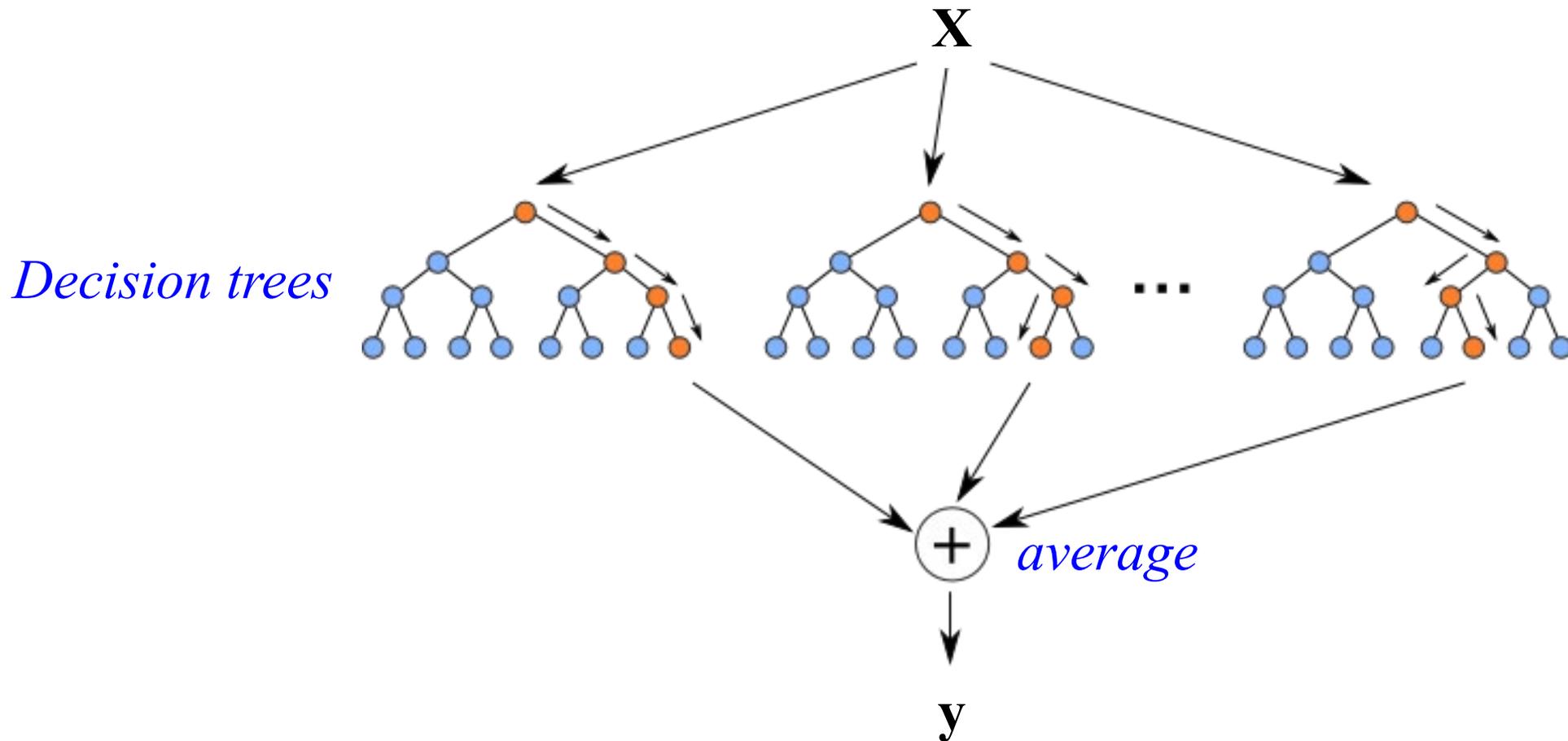
(266, 11)



```
# train set 결합  
# test set 결합
```

# 머신 러닝 모델 학습

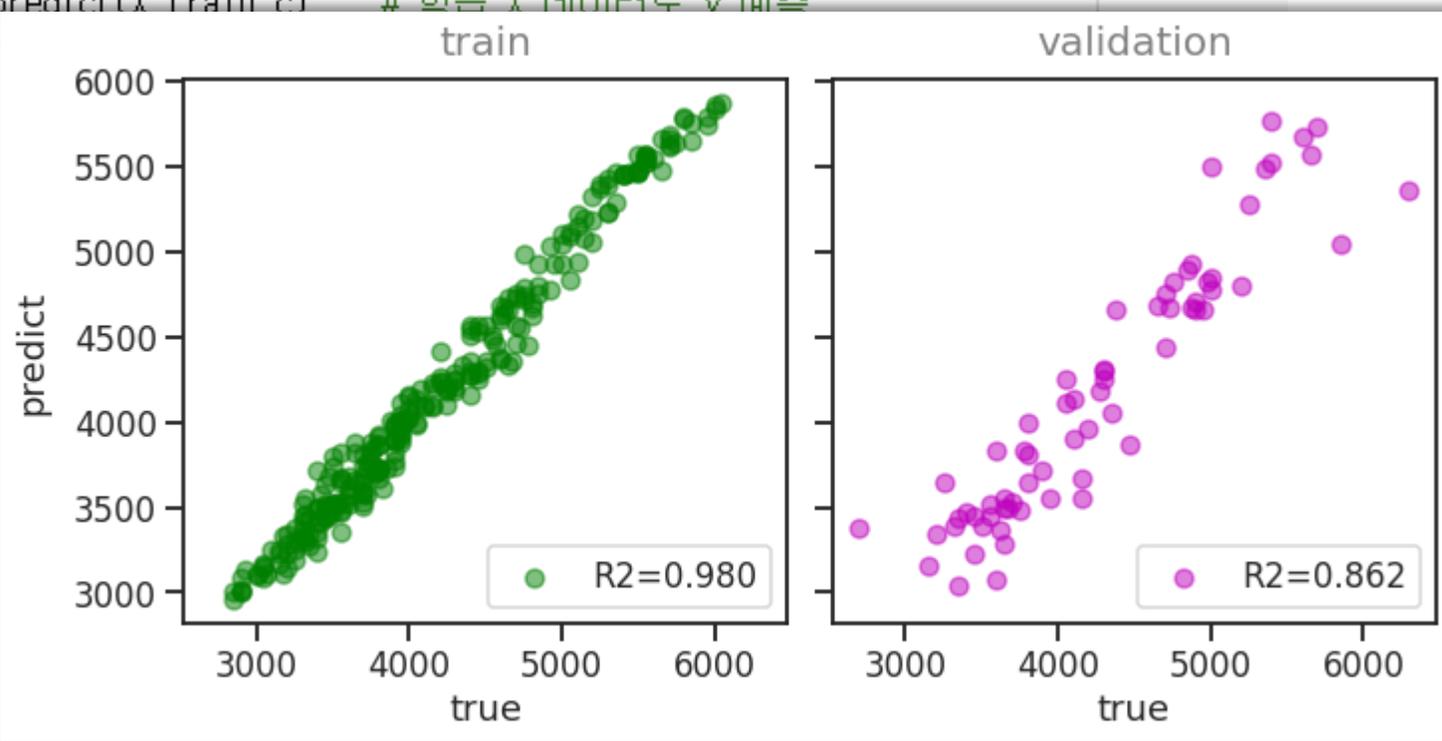
- Random Forest Regressor



# 머신 러닝 모델 학습

```
1 from sklearn.ensemble import RandomForestRegressor
2 from sklearn.metrics import r2_score
3
4 rf = RandomForestRegressor()           # 머신 러닝 모델 선택 : Random Forest Regressor
5 rf.fit(X_train_c, y_train)           # 머신 러닝 모델 학습
6
7 y_train_pred = rf.predict(X_train_c) # 학습 X 데이터로 y 예측
8 y_test_pred = rf.predict(X_test_c)
9
10 print(f"# R2_train = {r2_score(y_train, y_train_pred)}")
11 print(f"# R2_val = {r2_score(y_test, y_test_pred)}")
```

# R2\_train = 0.980  
# R2\_val = 0.862



# 인자 중요도 계산

- Permutation importance
  - 인자별로 데이터를 뒤섞어 성능 예측.
  - 성능이 떨어진 인자일수록 중요.

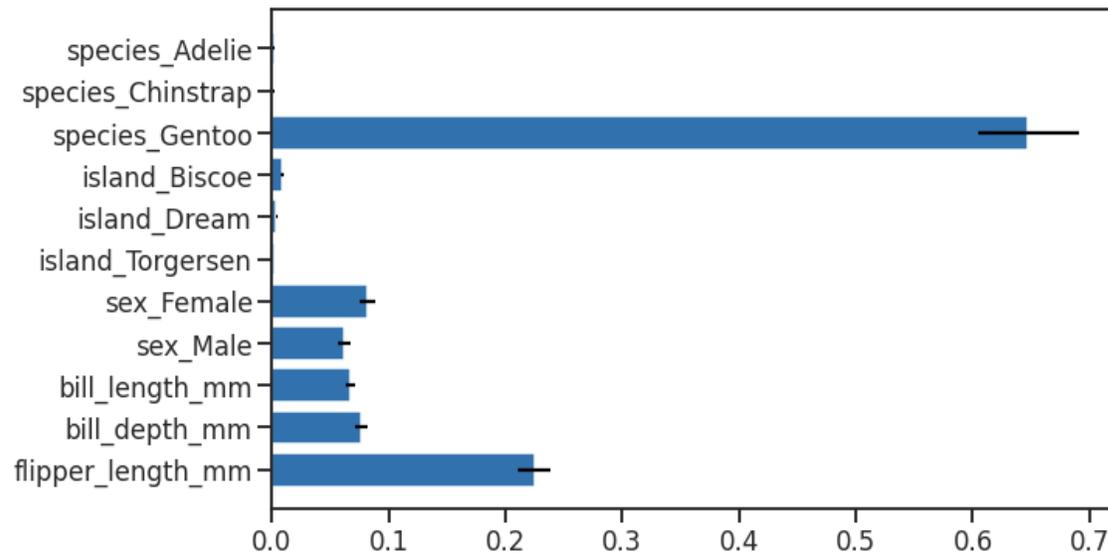
	RDSpend	Administration	MarketingSpend	Profit	state_California
1	165349.2	136897.8	471784.1	192261.83	0
2	162597.7	151377.59	443898.53	191792.06	1
3	153441.51	101145.55	407934.54	191050.39	1
...	...	...	...	...	...
48	0	135426.92	0	42559.73	1
49	542.05	51743.15	0	35673.41	0
50	0	116983.8	45173.06	14681.4	1

# 인자 중요도 계산

- Permutation importance: 뭔가 이상?

```
1 from sklearn.inspection import permutation_importance
2
3 pi = permutation_importance(rf, X_train_c, y_train, n_repeats=30)
```

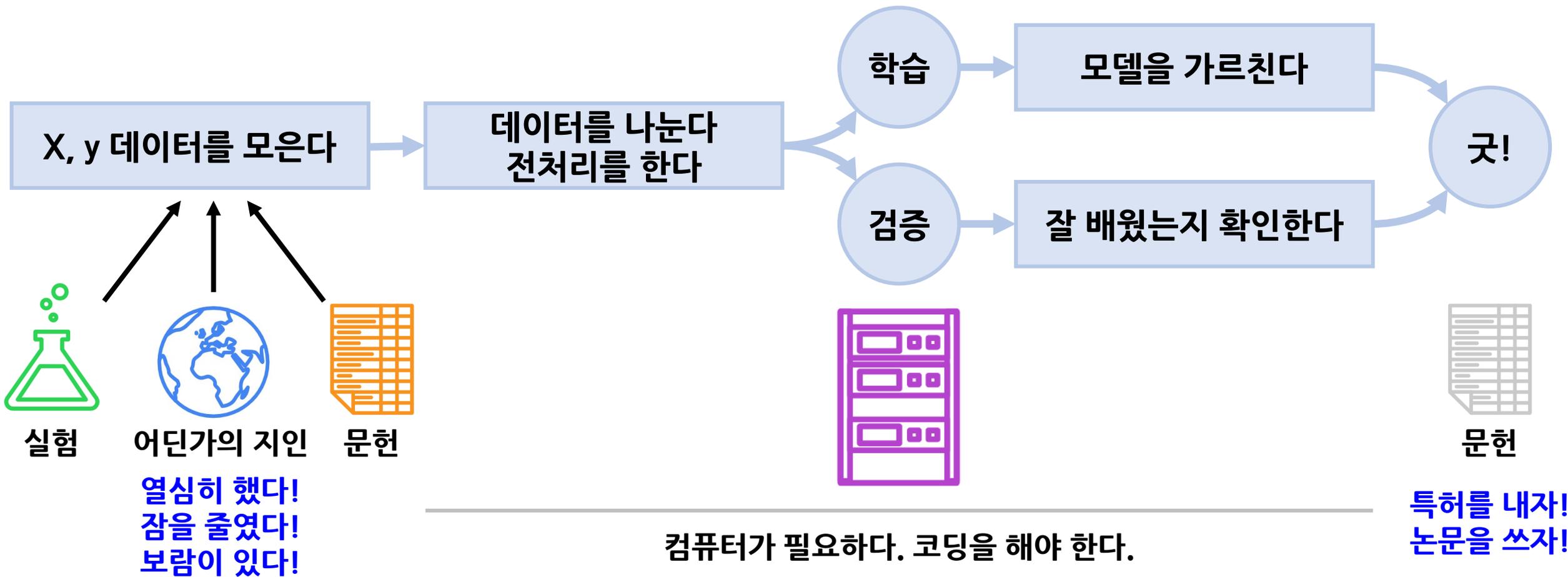
```
1 column_names = ["species_Adelie", "species_Chinstrap", "species_Gentoo", "island_Biscoe", "island_Dream", "is
2 column_names
3
4 fig, ax = plt.subplots(figsize=(10,5), constrained_layout=True)
5 ax.barh(column_names, pi.importances_mean, xerr=pi.importances_std)
6 ax.invert_yaxis()
```



**조금 지치지 않습니까?**

---

# 머신 러닝 활용





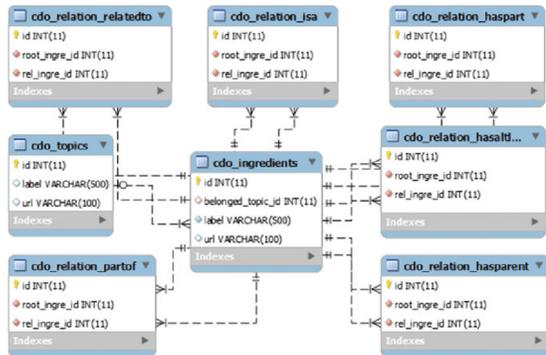
# 머신 러닝 활용

- 모델을 만든 후 지속 사용할 수 있는 환경이 중요

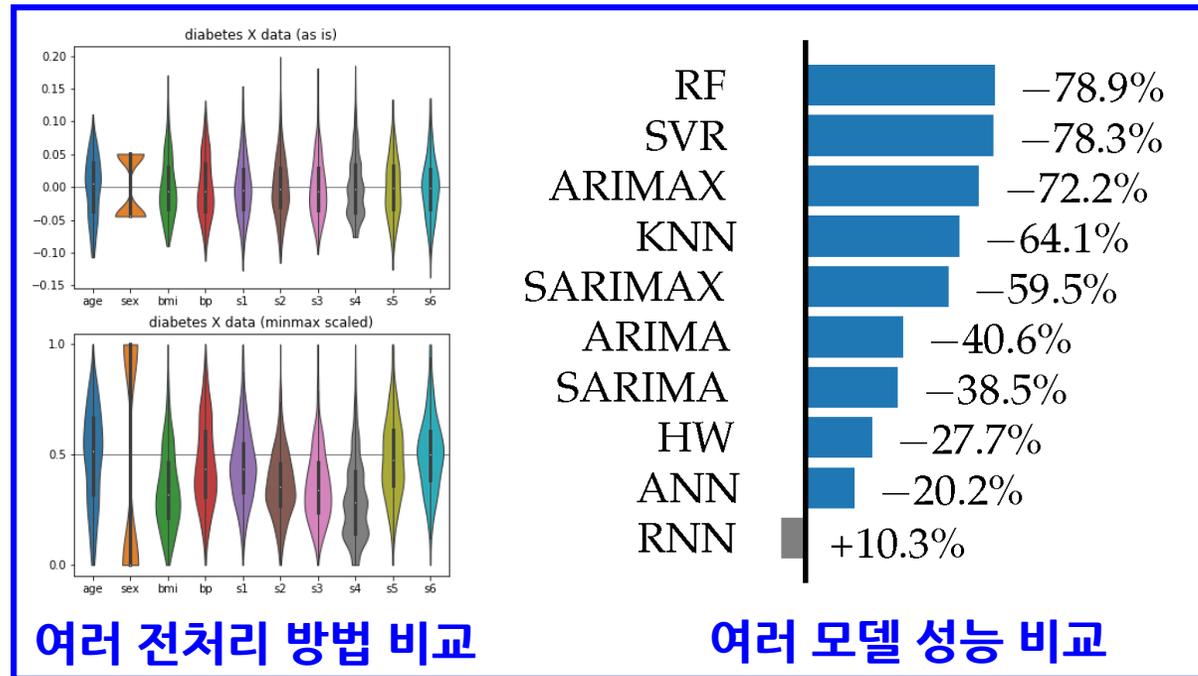


Data Pipeline

API Application Program Interface



데이터베이스



여러 전처리 방법 비교

여러 모델 성능 비교

trials & errors

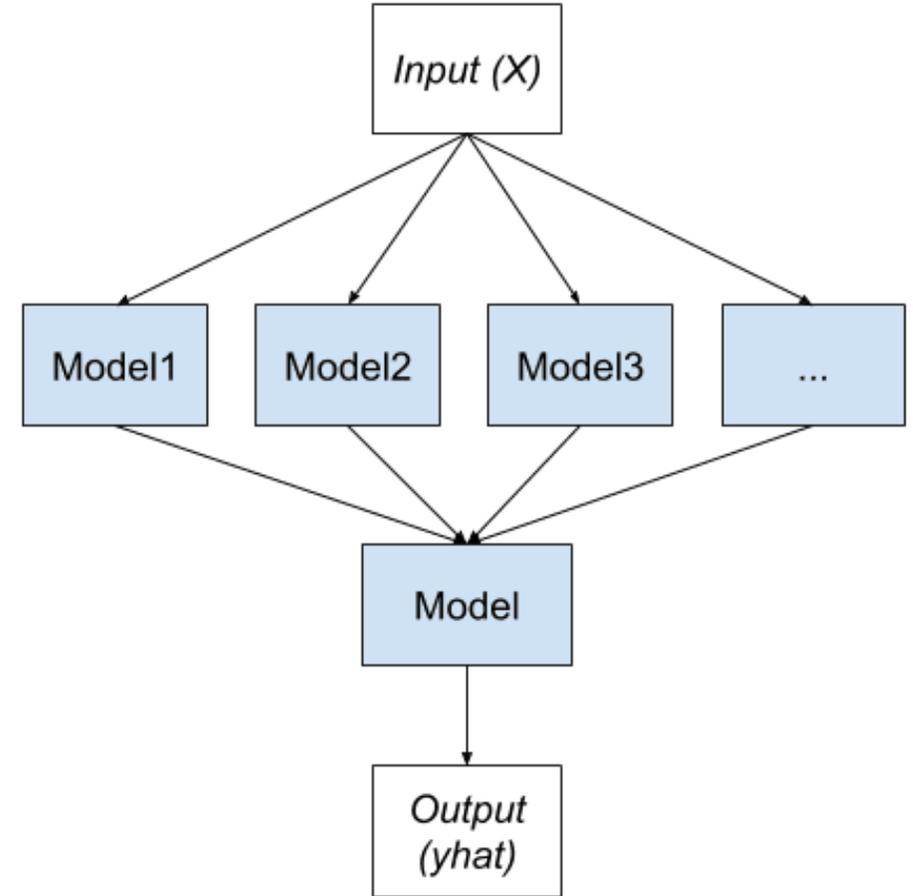
# 연결 단자 통일



# 머신 러닝 파이프라인



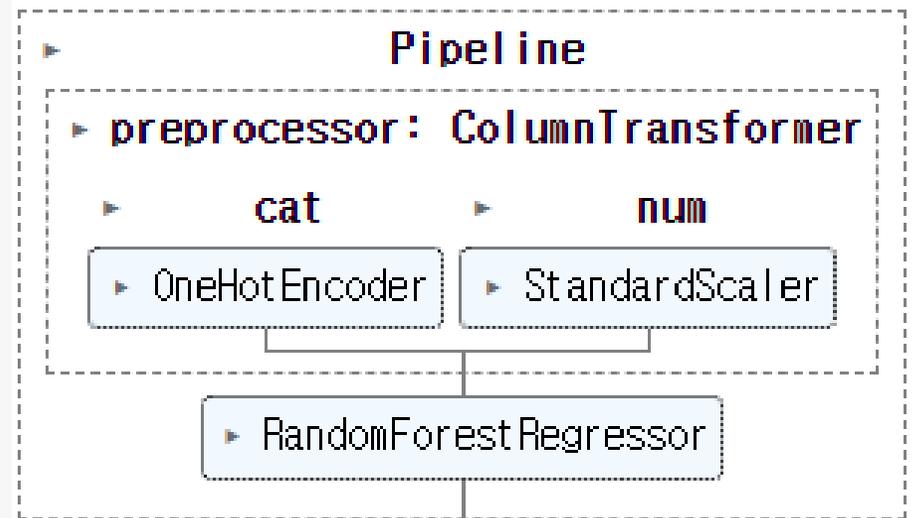
Stacking Ensemble



# 머신 러닝 파이프라인

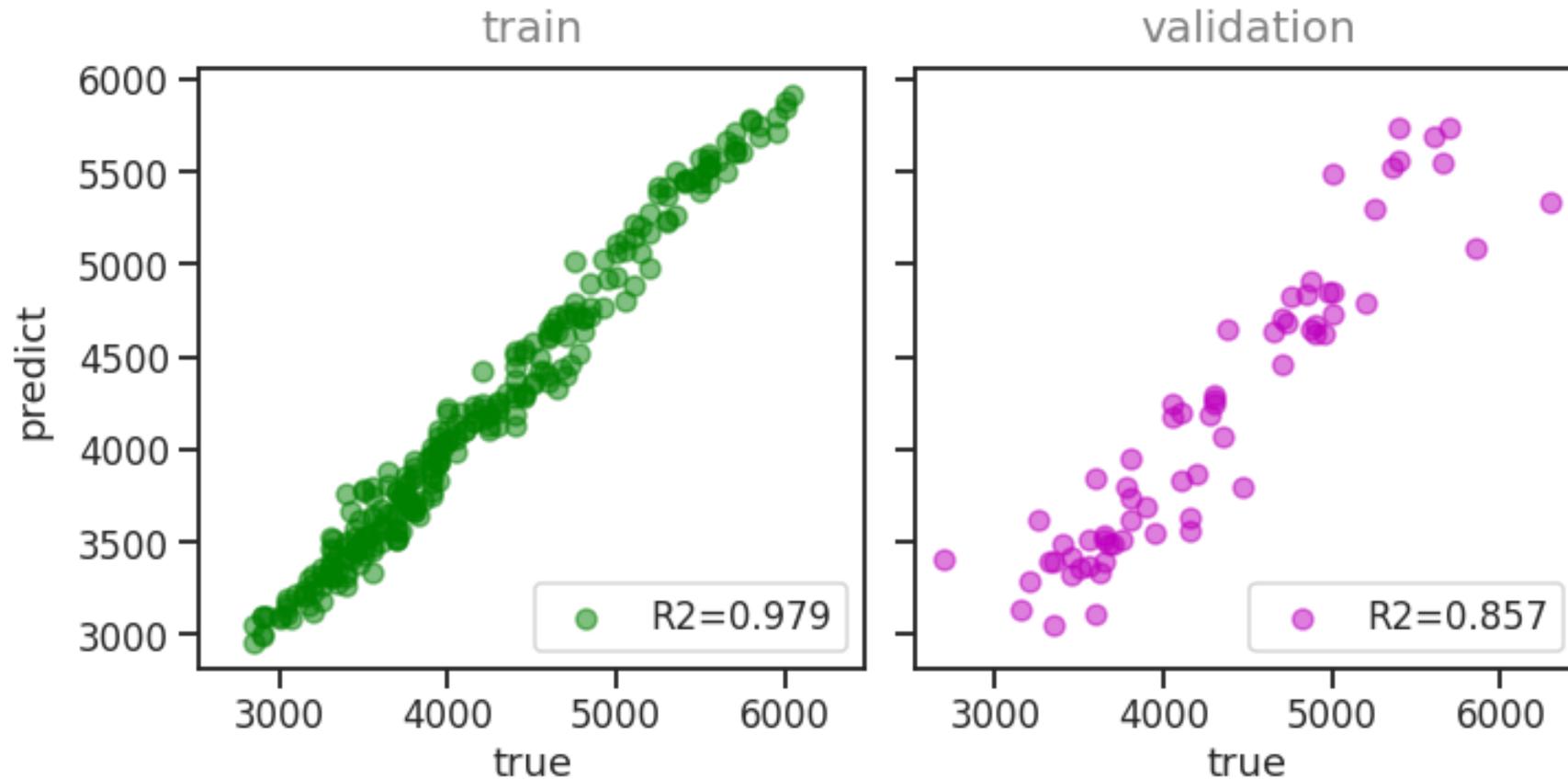
- scikit-learn 제공 데이터 처리 & 머신 러닝 모듈 결합 기능

```
1 from sklearn.compose import ColumnTransformer
2 from sklearn.pipeline import Pipeline
3
4 # 1-1.categorical feature에 one-hot encoding 적용
5 cat_features = ["species", "island", "sex"]
6 cat_transformer = OneHotEncoder()
7
8 # 1-2.numerical feature는 standard scaler 적용
9 num_features = ["bill_length_mm", "bill_depth_mm", "flipper_length_mm"]
10 num_transformer = StandardScaler()
11
12 # 2. 인자 종류별 전처리 적용
13 preprocessor = ColumnTransformer([("cat", cat_transformer, cat_features),
14                                 ("num", num_transformer, num_features)])
15
16 # 3. 전처리 후 랜덤포레스트 적용
17 pipeline = Pipeline(steps=[("preprocessor", preprocessor),
18                             ("rf", RandomForestRegressor())])
```



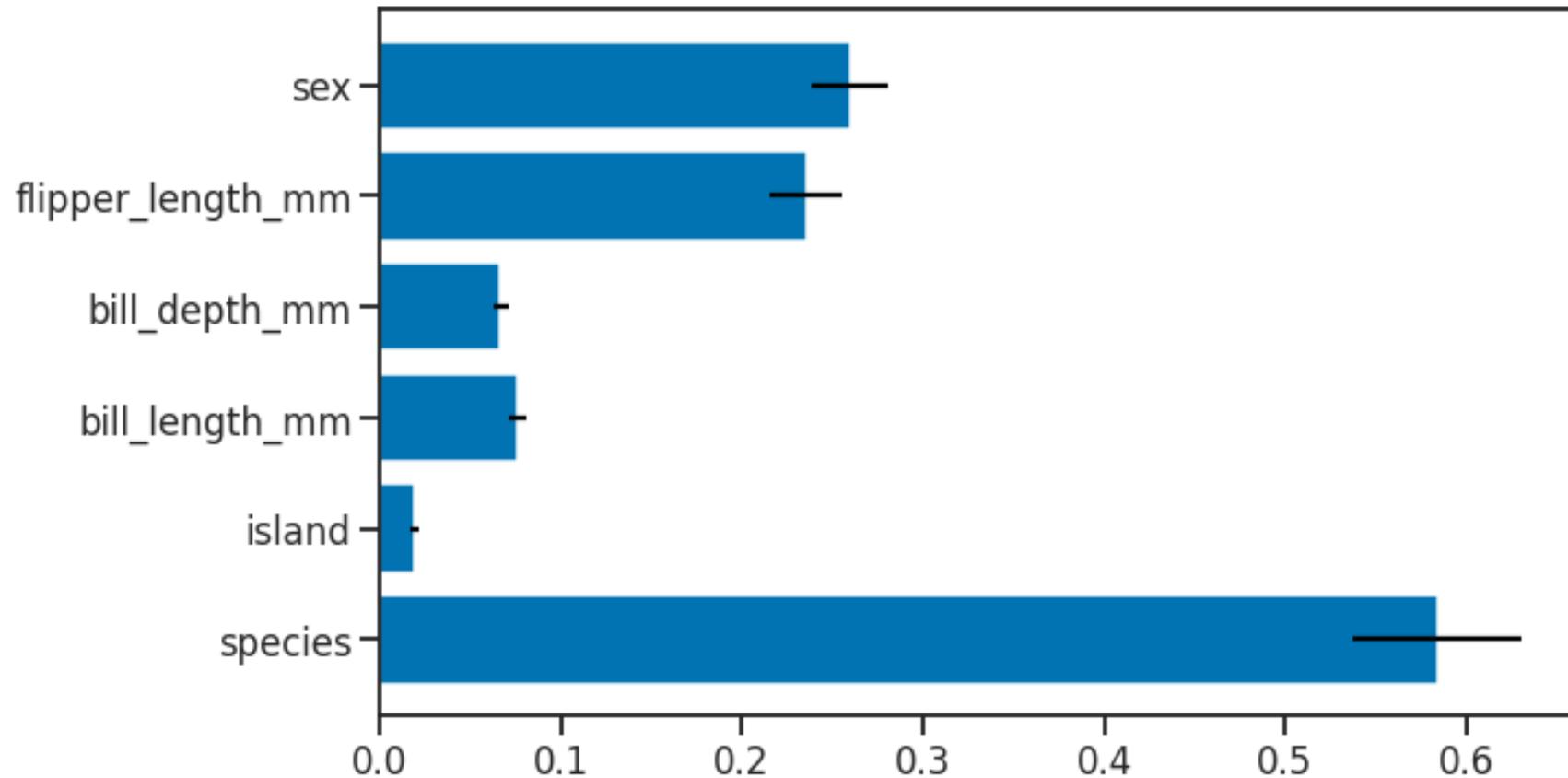
# 머신 러닝 파이프라인

- 성능 확인



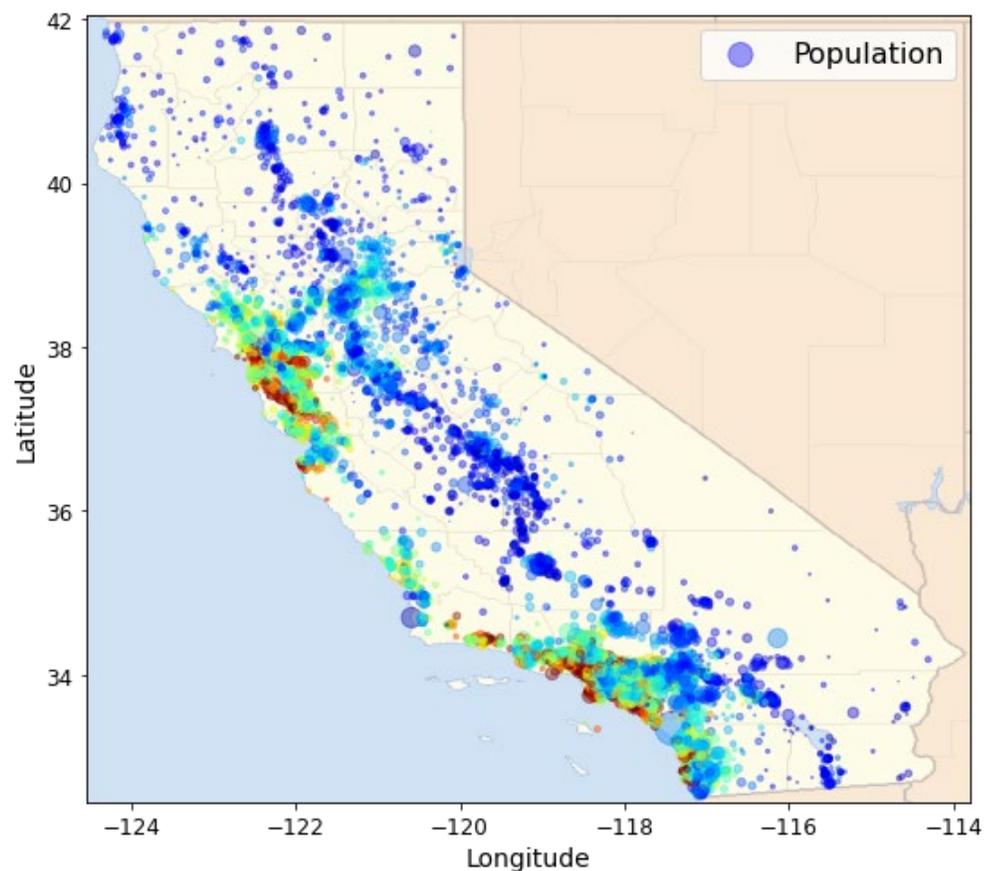
# 머신 러닝 파이프라인

- Permutation importance



# Homework

- 캘리포니아 집값 예측
  - 핸즈온 머신러닝 2판, 한빛미디어, <https://bit.ly/3800a9j>



```
1 from sklearn.datasets import fetch_california_housing
2 housing = fetch_california_housing()
```

```
1 import pandas as pd
2
3 X = pd.DataFrame(data=housing["data"], columns=housing["feature_names"])
4 X
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25
...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24

20640 rows × 8 columns

```
1 y = housing["target"]
2 y
```

```
array([4.526, 3.585, 3.521, ..., 0.923, 0.847, 0.894])
```