

Machine Learning

Machine Learning Project From Start to End

경희대학교 컴퓨터공학과
2019102191 신주영



Main Procedure

1. See big picture
2. Obtain data
3. Explore and visualize to gain insight from data
4. Preprocess data for machine learning algorithms
5. Select and train data
6. Tune the data precisely
7. Present the solution
8. Launch, monitor and maintain the data

Working with physical data

Procedure 1.



1. Working with physical data

- Famous Public Data Repository
 - UC Irvine Machine Learning Repository: [link](#)
 - Kaggle Dataset: [link](#)
 - Amazon AWS Dataset: [link](#)
- Meta Potal
 - Data Portals: [link](#)
 - Open Data Monitor: [link](#)
 - Quandl: [link](#)
- Other pages that list popular public data stores
 - Wiki Machine Learning Dataset List: [link](#)
 - Quora.com: [link](#)
 - Dataset Subreddit: [link](#)

See big picture

Procedure 2.



2. See big picture

- given feature: population, median income, etc...
- given label: median housing price
- to predict: The median price of the zone given different measurement data

2-1. Define problem

1. Define purpose of problem
2. Find solution
3. Select ML algorithm

2-2. Select Performance Metrics

- RMSE (root mean square error) will be used as a performance indicator.

- $RMSE(X, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2}$

- If there are many outliers, a mean absolute error may be considered.

- $MAE(X, h) = \frac{1}{m} \sum_{i=1}^m |h(x^{(i)}) - y^{(i)}|$

2-3. assumption examination

- List and examine assumptions made so far.

Obtain data

Procedure 3.



3-1. Data Download

- create a datasets/housing directory in the current workspace
- download the housing.tgz file, and extract it to the same directory to create a housing.csv file.

```
import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url = HOUSING_URL, housing_path = HOUSING_PATH):
    os.makedirs(housing_path, exist_ok = True) #make directory
    tgz_path = os.path.join(housing_path, "housing.tgz") #cd to directory maded
    urllib.request.urlretrieve(housing_url, tgz_path) #download file
    housing_tgz = tarfile.open(tgz_path) #make extracting instance
    housing_tgz.extractall(path = housing_path) #extract all file
    housing_tgz.close() #close instance maded
```

3-1. Data Download

- Make a pandas dataframe

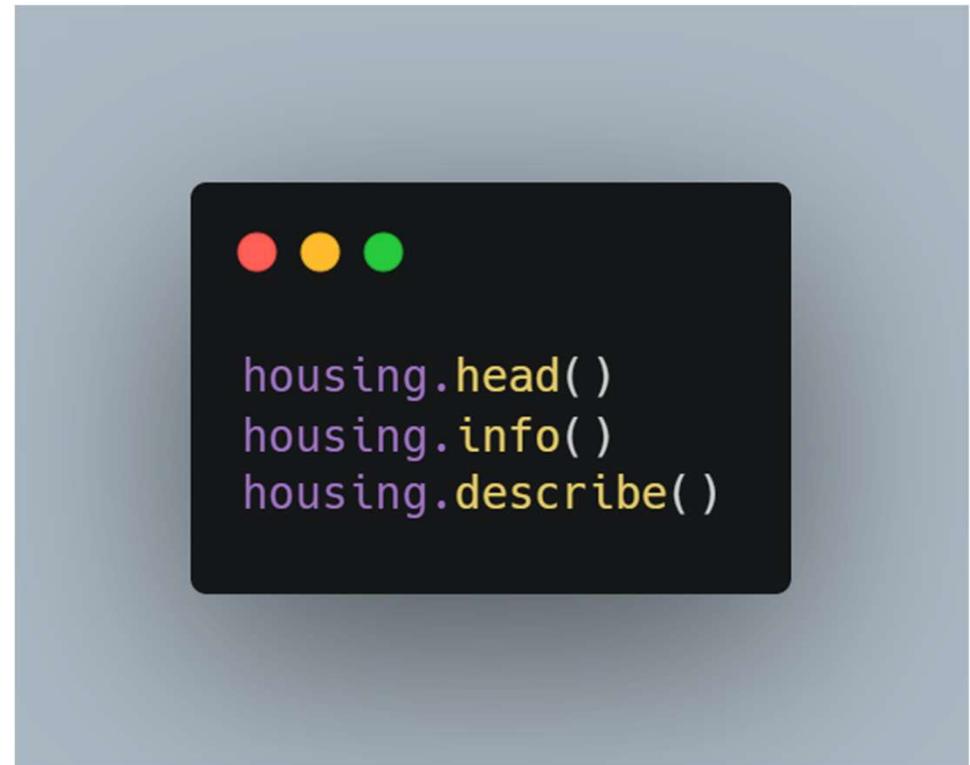


```
import pandas as pd

def load_housing_data(housing_path = HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv") #invoke the csv path
    return pd.read_csv(csv_path) #read csv file
```

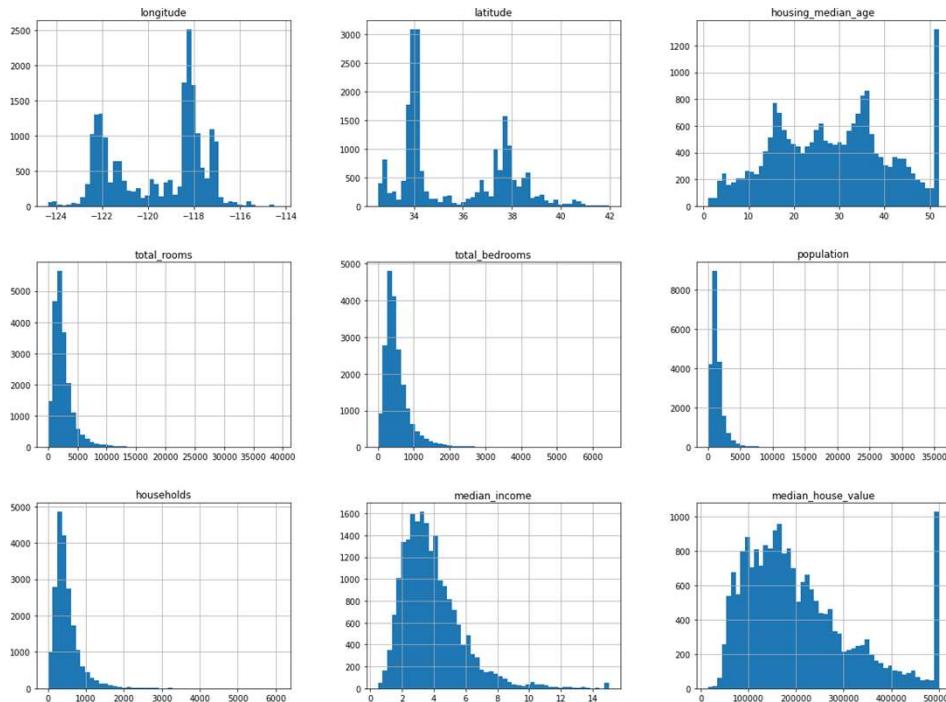
3-2. Data Structure skim through

- head(): Outputs the first five rows of the data frame
- info(): a brief description of the data and the total number of rows, and the number of data values for each attribute and non-NULL data values
- Describe(): shows summary information of numeric characteristics.



경희대학교
KYUNG HEE UNIVERSITY

3-2. Data Structure skim through



```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize = (20, 15))
plt.show()
```

3-3. Create test set

random sampling

- Randomly sample the data

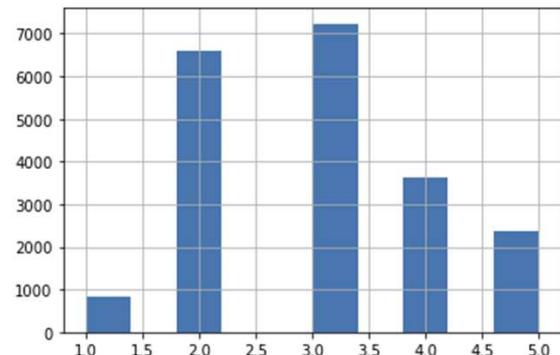
```
● ● ●  
from sklearn.model_selection import train_test_split  
train_set,test_set = train_test_split(housing, test_size = 0.2, random_state = 42)
```

3-3. Create test set stratified sampling

- Sampling data by strata.

```
In [17]: housing["income_cat"].hist()
```

```
Out[17]: <AxesSubplot :>
```



```
from sklearn.model_selection import train_test_split  
train_set, test_set = train_test_split(housing, test_size = 0.2, random_state = 42)
```

3-3. Create test set

stratified sampling

- Sampling data by strata.

parameter name	role
n_splits	int, default = 10 분리할 데이터 셋의 갯수를 지정
test_size	float, default = None 테스트 셋의 비율을 지정
train_size	float, default = None 훈련 셋의 비율을 지정
random_state	int or RandomState instance, default = None 생성된 훈련 및 테스트 셋 난수를 지정

```
● ● ●  
from sklearn.model_selection import StratifiedShuffleSplit  
  
split = StratifiedShuffleSplit(n_splits = 1, test_size = 0.2, random_state = 42)  
for train_index, test_index in split.split(housing, housing["income_cat"]):  
    strat_train_set = housing.loc[train_index] #save training data only in  
    strat_train_set  
    strat_test_set = housing.loc[test_index] #save test data only in strat_test_set
```

Explore and visualize to gain insight from data

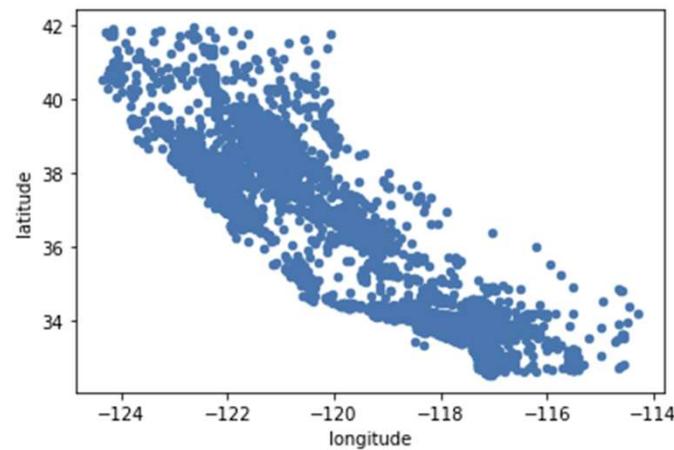
Procedure 4.



4-1. Geographic data visualization

```
In [23]: housing = strat_train_set.copy() #Make and use a copy to avoid damaging the training dataset  
housing.plot(kind = "scatter", x = "longitude", y = "latitude")
```

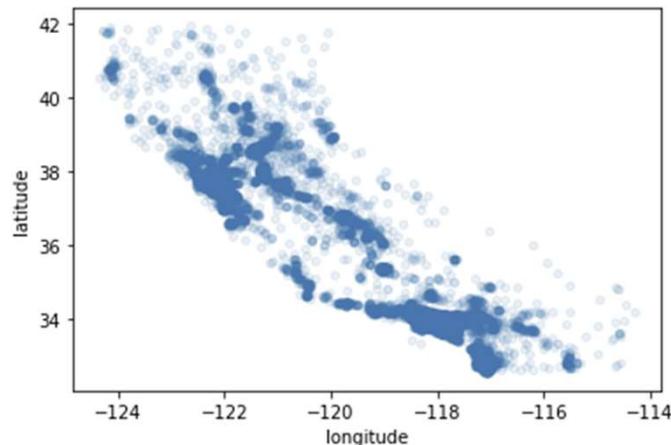
```
Out [23]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>
```



4-1. Geographic data visualization

```
In [24]: housing.plot(kind = "scatter", x = "longitude", y = "latitude", alpha = 0.1)
```

```
Out [24]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>
```

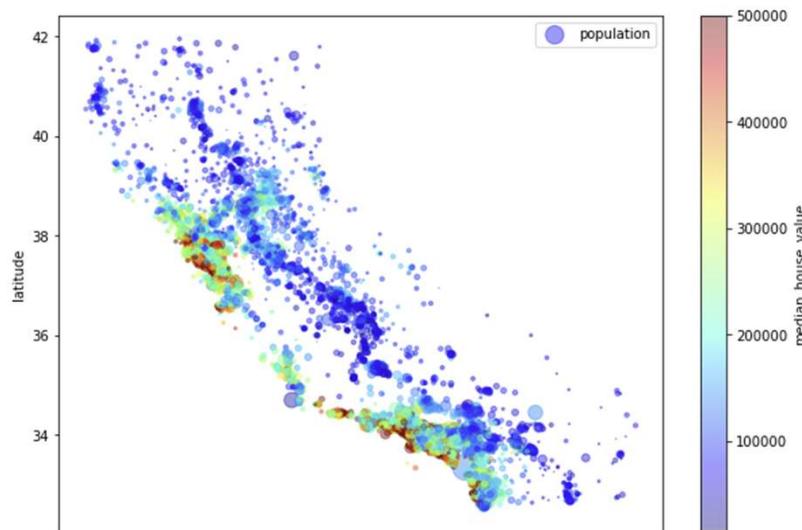


- By specifying the alpha parameter (=0.1), the area where data points are concentrated can be identified.

4-1. Geographic data visualization

```
In [25]: housing.plot(kind = 'scatter', x = 'longitude', y = 'latitude', alpha = 0.4,
                   s = housing["population"]/100, label = "population", figsize = (10, 7),
                   c = "median_house_value", cmap=plt.get_cmap("jet"), colorbar=True)
#color means house value
#size means population
```

```
Out [25]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>
```

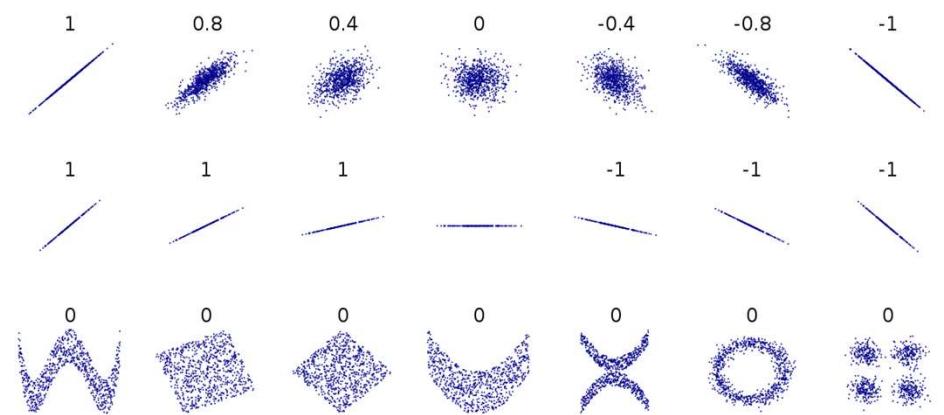


- As expected, housing prices are highly related to region and population density.



4-2. Correlation investigation

- The standard correlation coefficient can be calculated through the corr() method.
- The range of coefficient is -1 to 1
- The closer to 1, the more positive the correlation is.
- The closer to -1, the more negative the correlation is.



4-2. Correlation investigation

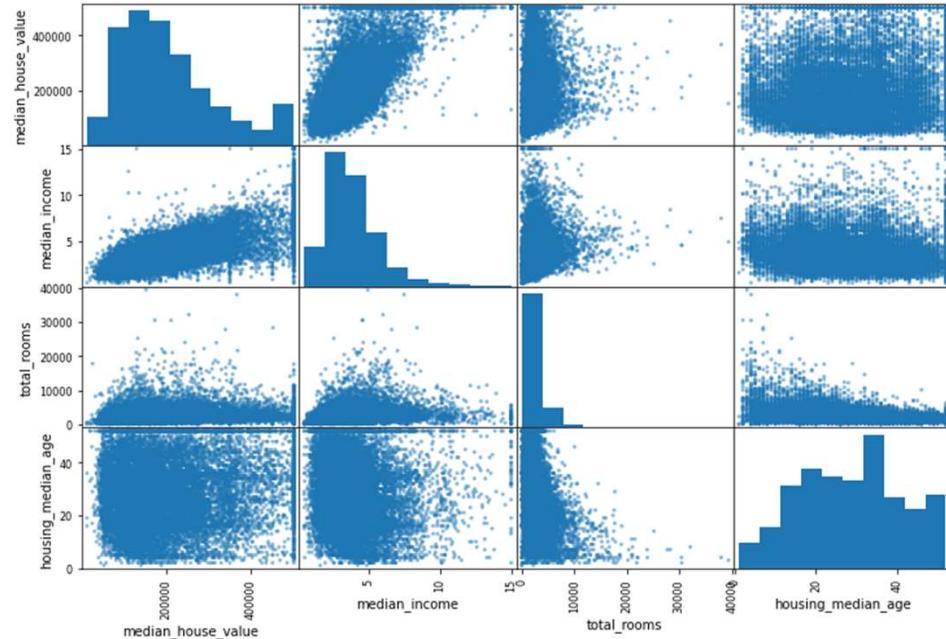
```
In [26]: corr_matrix = housing.corr()  
corr_matrix
```

Out [26]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
longitude	1.000000	-0.924478	-0.105823	0.048909	0.076686	0.108071	0.063146	-0.019615	-0.047466
latitude	-0.924478	1.000000	0.005737	-0.039245	-0.072550	-0.115290	-0.077765	-0.075146	-0.142673
housing_median_age	-0.105823	0.005737	1.000000	-0.364535	-0.325101	-0.298737	-0.306473	-0.111315	0.114146
total_rooms	0.048909	-0.039245	-0.364535	1.000000	0.929391	0.855103	0.918396	0.200133	0.135140
total_bedrooms	0.076686	-0.072550	-0.325101	0.929391	1.000000	0.876324	0.980167	-0.009643	0.047781
population	0.108071	-0.115290	-0.298737	0.855103	0.876324	1.000000	0.904639	0.002421	-0.026882
households	0.063146	-0.077765	-0.306473	0.918396	0.980167	0.904639	1.000000	0.010869	0.064590
median_income	-0.019615	-0.075146	-0.111315	0.200133	-0.009643	0.002421	0.010869	1.000000	0.687151
median_house_value	-0.047466	-0.142673	0.114146	0.135140	0.047781	-0.026882	0.064590	0.687151	1.000000



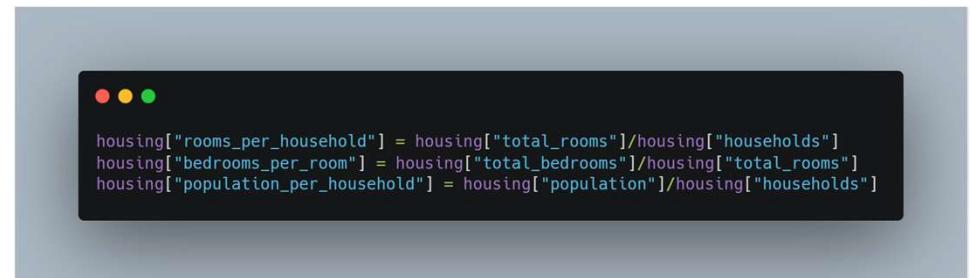
4-2. Correlation investigation



```
● ● ●  
from pandas.plotting import scatter_matrix  
  
attributes = ["median_house_value", "median_income", "total_rooms",  
"housing_median_age"]  
scatter_matrix(housing[attributes], figsize = (12, 8))
```

4-3. Experiment with a combination of properties

- The final process of data processing is to combine several characteristics.



```
rooms_per_household = housing["total_rooms"]/housing["households"]
bedrooms_per_room = housing["total_bedrooms"]/housing["total_rooms"]
population_per_household = housing["population"]/housing["households"]
```

Preprocess data for machine learning algorithms

Procedure 5.



5-1. Data purification

- three methods to purify data
(deleting NULL data)
 - option 1: delete that row -> dropna()
 - option 2: delete that column -> drop()
 - option 3: fill the value(0, mean, median, etc...) -> fillna()



```
housing.dropna(subset=["total_bedrooms"]) #option 1
housing.drop("total_bedrooms", axis = 1) #option 2
median = housing["total_bedrooms"].median() #option 3
housing["total_bedrooms"].fillna(median, inplace=True)
```

5-1. Data purification

using scikit-learn



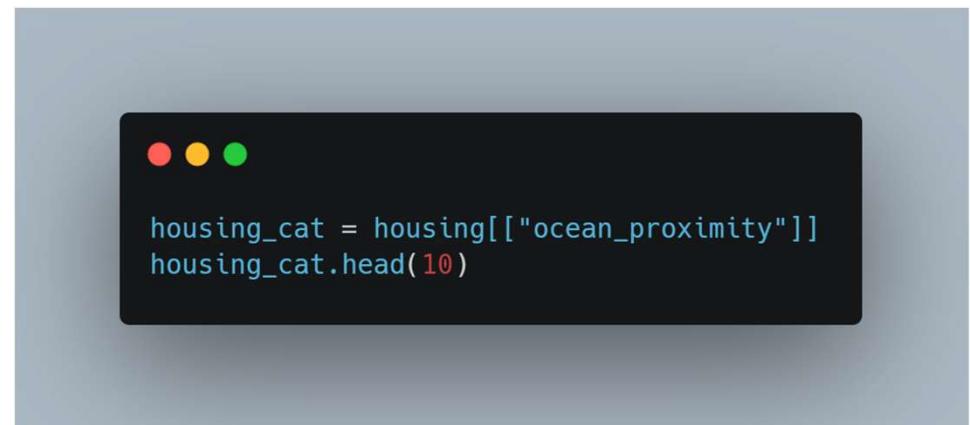
```
from sklearn.impute import SimpleImputer  
imputer = SimpleImputer(strategy = "median")  
housing_num = housing.drop("ocean_proximity", axis = 1)  
imputer.fit(housing_num) #fit data into created instance
```



경희대학교
KYUNG HEE UNIVERSITY

5-2. Dealing with categorical attributes

ocean_proximity	
12655	INLAND
15502	NEAR OCEAN
2908	INLAND
14053	NEAR OCEAN
20496	<1H OCEAN
1481	NEAR BAY
18125	<1H OCEAN
5830	<1H OCEAN
17989	<1H OCEAN
4861	<1H OCEAN



The screenshot shows a Jupyter Notebook cell with three colored dots (red, yellow, green) indicating the status of the code execution. The code itself is:

```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```



5-2. Dealing with categorical attributes

OrdinalEncoder

```
array([[1.],
       [4.],
       [1.],
       [4.],
       [0.],
       [3.],
       [0.],
       [0.],
       [0.],
       [0.]])
```



```
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder() #make OrdinalEncoder() method
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat) #convert categories
from text to numbers
housing_cat_encoded[:10]
```

5-2. Dealing with categorical attributes

OneHotEncoder

```
array([[0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0.],
       ...,
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```



```
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot.toarray()
```

5-3. My own Encoder

- Since scikit-learn supports duck typing, you can create a class that implements the fit(), transform(), and fit_transform() methods.
- The last method(fit_transform()) is automatically generated when TransformerMixin is inherited.
- Inheriting BaseEstimator gives you two additional methods (get_params() and set_params()) required for hyperparameter tuning.

5-3. My own Encoder

```
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin): #class declaration,
    #class name, class to inherit
    def __init__(self, add_bedrooms_per_room = True): #constructor
        self.add_bedrooms_per_room = add_bedrooms_per_room

    def fit(self, X, y = None):
        return self

    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, population_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                       bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room = False)
housing_extra_attribs = attr_adder.transform(housing.values)
```



5-4. Feature Scaling

- min-max scaling(normalization): If the minimum value is subtracted from the data and divided by the difference between the maximum value and the minimum value, it can have a value in the range of 0 to 1.
 - scikit-learn method: MinMaxScaler
 - If you don't want between 0 ~ 1, you can specify a range with the feature_range parameter.
- standardization: Subtract the mean and divide it by the standard deviation so that the variance of the result distribution is 1.
 - scikit-learn method: StandardScaler
 - weakness: There are no upper or lower limits of the range in standardization.
 - strength: Less influenced by outlier.

5-5. Transform Pipeline

```
● ● ●

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy = 'median')), #impute median value to missing
    ('attribs_adder', CombinedAttributesAdder()), #attribute combined data
    ('std_scaler', StandardScaler()) # use scaler as a StandardSclaer()
])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```



5-5. Transform Pipeline

```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num) #column names of housing_num
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs), #numerical columns, method to use, column
name
    ("cat", OneHotEncoder(), cat_attribs) #categorical columns, method to use, column
name
])

housing_prepared = full_pipeline.fit_transform(housing)
```

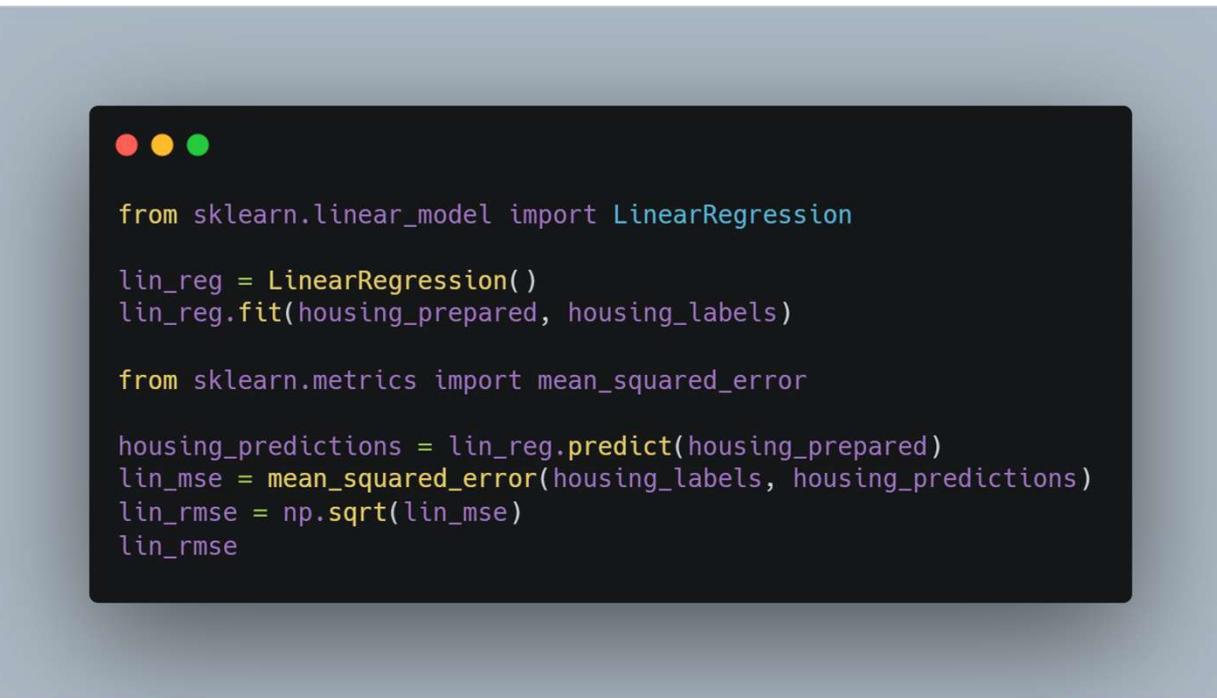


Model selection and Training

Procedure 6.



6-1. Training and evaluation in the training set



```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out [54] : 68898.1375370648

6-1. Training and evaluation in the training set

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)

housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out [56]: 0.0

6-2. Evaluation using cross-validation

K-fold cross-validation

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                        scoring = "neg_mean_squared_error", cv = 10)
tree_rmse_scores = np.sqrt(-scores)

def display_scores(scores):
    print("점수:", scores)
    print("평균:", scores.mean())
    print("표준편차:", scores.std())
```



6-2. Evaluation using cross-validation

K-fold cross-validation (decision tree regression)

```
In [59]: display_scores(tree_rmse_scores) #Decision Tree Regressor Cross Validation
```

```
점수: [69967.3008089 68004.88688671 67430.74076444 68247.0170099  
       66218.82800129 76653.06092979 70273.66771328 71975.20626109  
       69834.44957085 70946.32865859]  
평균: 69955.14866048493
```

```
표준편차: 2776.602809433244
```

6-2. Evaluation using cross-validation

K-fold cross-validation (linear regression)

```
In [60]: lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,  
                                  scoring = "neg_mean_squared_error", cv = 10)  
lin_rmse_scores = np.sqrt(-lin_scores)
```

```
In [61]: display_scores(lin_rmse_scores) #Linear Regression Cross Value Score
```

점수: [72388.64937198 64547.9887793 68170.84810636 69123.05064823
66882.20628954 73022.80595635 70535.54365056 69461.22422101
66950.1167962 70520.49977901]

평균: 69160.29335985384

표준편차: 2479.952095557169



경희대학교
KYUNG HEE UNIVERSITY

6-2. Evaluation using cross-validation

K-fold cross-validation (random forest regression)

```
In [60]: lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,  
                                  scoring = "neg_mean_squared_error", cv = 10)  
lin_rmse_scores = np.sqrt(-lin_scores)
```

```
In [61]: display_scores(lin_rmse_scores) #Linear Regression Cross Value Score
```

점수: [72388.64937198 64547.9887793 68170.84810636 69123.05064823
66882.20628954 73022.80595635 70535.54365056 69461.22422101
66950.1167962 70520.49977901]

평균: 69160.29335985384

표준편차: 2479.952095557169



경희대학교
KYUNG HEE UNIVERSITY

Model detail tuning

Procedure 7.



7-1. Grid Search

- Specify the hyperparameter you want to explore and the value you want to try and evaluate all combinations using cross-validation.

```
● ● ●  
from sklearn.model_selection import GridSearchCV  
  
param_grid = [  
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},  
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]}  
] #hyperparameters to try  
  
forest_reg = RandomForestRegressor()  
  
grid_search = GridSearchCV(forest_reg, param_grid, cv = 5,  
                           scoring = 'neg_mean_squared_error',  
                           return_train_score = True)  
  
grid_search.fit(housing_prepared, housing_labels)
```



```
Out[67]: GridSearchCV(cv=5, estimator=RandomForestRegressor(),  
                      param_grid=[{'max_features': [2, 4, 6, 8],  
                                   'n_estimators': [3, 10, 30]},  
                                   {'bootstrap': [False], 'max_features': [2, 3, 4],  
                                   'n_estimators': [3, 10]}],  
                      return_train_score=True, scoring='neg_mean_squared_error')
```



경희대학교
KYUNG HEE UNIVERSITY

7-2. Random Search

- Grid search is good when exploring a relatively small number of combinations.
- As the hyperparameter search space increases, it is better to use Randomized Search CV.

7-3. Ensemble Search

- Another way to tune a model in detail is to connect the best model.

7-4. Best Model and Error Analysis

- Analyzing the best model often yields good insight into the problem.

```
In [71]: feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

```
Out[71]: array([7.63031915e-02, 7.13582739e-02, 3.96722537e-02, 1.55048455e-02,
   1.49657397e-02, 1.55446577e-02, 1.35897507e-02, 3.24456165e-01,
   4.90514949e-02, 6.60548188e-02, 1.30764725e-01, 9.75393145e-03,
   1.66237347e-01, 6.93386089e-05, 3.99310981e-03, 2.68035737e-03])
```

```
In [72]: extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
cat_encoder = full_pipeline.named_transformers_["cat"]
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
sorted(zip(feature_importances, attributes), reverse = True)
```

```
Out[72]: [(0.3244561646721308, 'median_income'),
(0.16623734681396224, 'INLAND'),
(0.13076472491105126, 'bedrooms_per_room'),
(0.07630319150241188, 'longitude'),
(0.07135827387984, 'latitude'),
(0.06605481882191198, 'pop_per_hhold'),
(0.04905149489931189, 'rooms_per_hhold'),
(0.03967225367970092, 'housing_median_age'),
(0.015544657725079973, 'population'),
(0.015504845471219107, 'total_rooms'),
(0.014965739706786673, 'total_bedrooms'),
(0.013589750673978258, 'households'),
(0.009753931452432318, '<1H OCEAN'),
(0.003993109814244523, 'NEAR BAY'),
(0.0026803573670644686, 'NEAR OCEAN'),
(6.933860887367807e-05, 'ISLAND')]
```

7-5. Evaluate the system with a test set

- Evaluate the final model in the test set.

```
In [73]: final_model = grid_search.best_estimator_
X_test = strat_test_set.drop("median_house_value", axis = 1)
y_test = strat_test_set["median_house_value"].copy()
X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

```
In [74]: from scipy import stats
confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                        loc = squared_errors.mean(),
                        scale = stats.sem(squared_errors)))
```

```
Out[74]: array([45633.32462379, 49459.85948044])
```

Launching, Monitoring, System maintenance

Procedure 8.



8. Launching, Monitoring, System maintenance

- A trained model containing the entire preprocessing pipeline and the prediction pipeline is loaded in a commercial environment and the predict() method is called to make a prediction.
 - joblib
 - REST API
 - Google Cloud AI Platform
- System maintenance
 - Collect and label new data regularly.
 - Train the model and write a script that automatically tunes the hyperparameters in detail.
 - Write one more script to evaluate the new model and the previous model in the updated test set.

Practice Problem

Procedure 9.



1. Find the best SVR model by changing the hyperparameters of the SVR

Fitting 5 folds for each of 50 candidates, totalling 250 fits

```
Out[83]: GridSearchCV(cv=5, estimator=SVR(), n_jobs=-1,
                      param_grid=[{'C': [10.0, 30.0, 100.0, 300.0, 1000.0, 3000.0,
                                      10000.0, 30000.0],
                                   'kernel': ['linear']},
                                  {'C': [1.0, 3.0, 10.0, 30.0, 100.0, 300.0, 1000.0],
                                   'gamma': [0.01, 0.03, 0.1, 0.3, 1.0, 3.0],
                                   'kernel': ['rbf']}],
                      scoring='neg_mean_squared_error', verbose=2)
```

```
In [84]: svm_grid_search.best_params_
```

```
Out[84]: {'C': 300.0, 'kernel': 'linear'}
```

- The best SVR model is "SVR(kernel = 'linear', C = 300)".

```
param_grid = [
    {'kernel': ['linear'], 'C': [10., 30., 100., 300., 1000., 3000.,
                               10000., 30000.0]},
    {'kernel': ['rbf'], 'C': [1.0, 3.0, 10., 30., 100., 300., 1000.0],
     'gamma': [0.01, 0.03, 0.1, 0.3, 1.0, 3.0]}
]

svm_reg = SVR()
svm_grid_search = GridSearchCV(svm_reg, param_grid, cv = 5,
                               scoring = 'neg_mean_squared_error',
                               verbose = 2, n_jobs = -1)
svm_grid_search.fit(housing_prepared, housing_labels)
```

2. Change GridSearchCV to RandomizedSearchCV

```
sv_reg_best_hyperparameter_random = SVR(kernel = "rbf", gamma= 0.1, C = 194700)
sv_reg_best_hyperparameter_random.fit(housing_prepared, housing_labels)

reg_score_best_hyperparameter_random =
cross_val_score(sv_reg_best_hyperparameter_random, housing_prepared, housing_labels,
                           scoring =
"neg_mean_squared_error", cv = 10)
reg_rmse_scores_best_hyperparameter_random = np.sqrt(-
reg_score_best_hyperparameter_random)
display_scores(reg_rmse_scores_best_hyperparameter_random)
```

점수: [54911.97827232 53240.23117989 56831.16972693 57132.12946488
55424.17020292 59463.44284473 52920.31835605 53605.2181824
59146.318853 56179.11354569]
평균: 55885.409062880426
표준편차: 2193.6089873705187

```
from sklearn.model_selection import RandomizedSearchCV

param_random = [{"kernel": ['rbf'], 'C': np.arange(194000.0, 195000.0), 'gamma': np.arange(0.1, 0.5)}]

svm_random_search = RandomizedSearchCV(svm_reg,
                                       param_distributions = param_random,
                                       n_iter = 10,
                                       cv = 5,
                                       scoring = "neg_mean_squared_error",
                                       verbose = 2,
                                       n_jobs = -1)

svm_random_search.fit(housing_prepared, housing_labels)

svm_random_search.best_params_
```

Out [93]: {'kernel': 'rbf', 'gamma': 0.1, 'C': 194999.0}



경희대학교
KYUNG HEE UNIVERSITY

3. Add the transducer that selects the most important characteristics to the staging pipeline.

```
In [95]: def indices_of_top_k(arr, k): #return top k indices
    return np.sort(np.argpartition(np.array(arr), -k)[-k:])
```

```
In [96]: class TopFeatureSelector(BaseEstimator, TransformerMixin): #class declaration, class name, class to inherit
    def __init__(self, feature_importances, k): #constructor, k = number of feature to collect
        self.feature_importances = feature_importances
        self.k = k

    def fit(self, X, y=None):
        self.feature_indices_ = indices_of_top_k(self.feature_importances, self.k) #get feature_importances's top k indices
        return self

    def transform(self, X):
        return X[:, self.feature_indices_] #return only top k importance features
```

```
In [97]: preparation_and_feature_selection_pipeline = Pipeline([
    ('preparation', full_pipeline),
    ('feature_selection', TopFeatureSelector(feature_importances, 5))
])
```

```
In [98]: housing_prepared_top_k_features = preparation_and_feature_selection_pipeline.fit_transform(housing)
```

4. Make the entire data preparation process and the final prediction into one pipeline

```
preparation_and_feature_selection_pipeline = Pipeline([
    ('preparation', preparation_and_feature_selection_pipeline),
    ('svm_reg', SVR(kernel = "rbf", gamma= 0.1, C = 194700))
])
}
```