Ch2. End to End Machine Learning Project

End to End Machine Learning Project

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

01 Look at the big picture

- **Frame this problem**
- **Select a Performance Measure - Regression Problem**

1. RMSE - root mean square error
   Euclidian norm 및 l2 norm

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

2. MAE - Mean Absolute Deviation
   Manhattan norm 및 l1 norm

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

=> Both method measurer the distance between predictor vector and target value vector

=> The higher norm index, the more it focuses on large values and neglect small one

02 Get the data

- **Quick Look at Data Structure**

```python
import os
import tarfile
import urllib.request

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/rickiepark/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):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

	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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 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
dtypes: float64(9), object(1)
memory usage: 1.5+ MB
```

02 Get the data

- **Create Test Set**

Data snooping bias : When you estimate the generalization error using the test set,
your estimate will be too optimistic, and they will not perform as well as expected

1) Random sampling

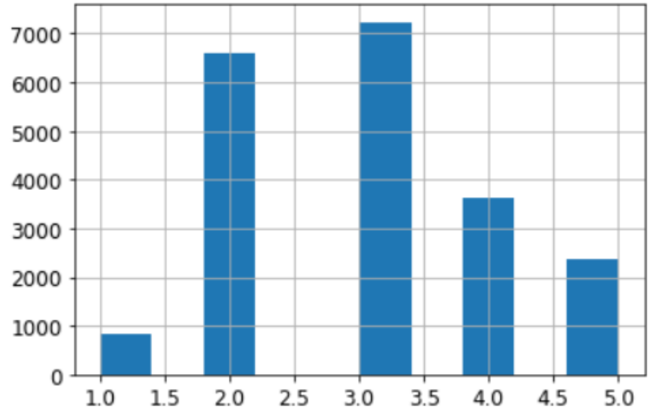
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

```
test_set.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0

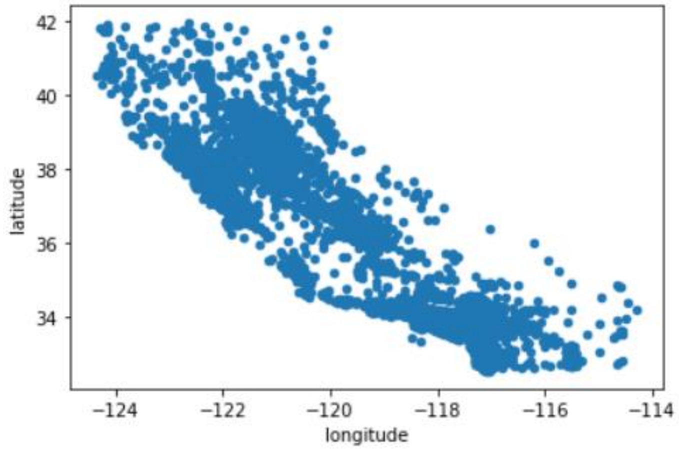
2) Hierarchical Sampling




```
3.0    0.350581
2.0    0.318847
4.0    0.176308
5.0    0.114438
1.0    0.039826
Name: income_cat, dtype: float64
```

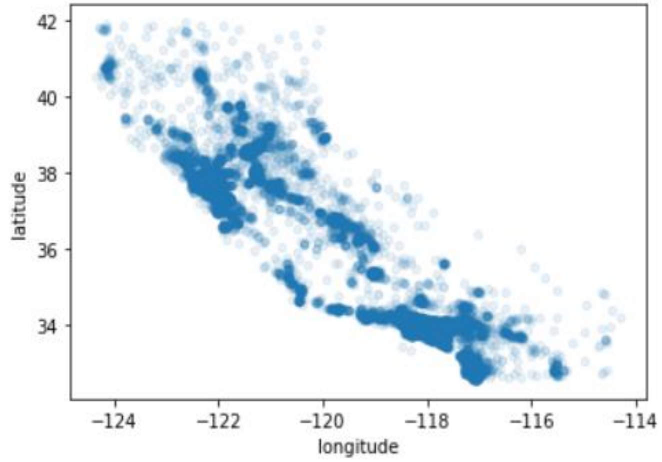
03 Discover and Visualize the Data

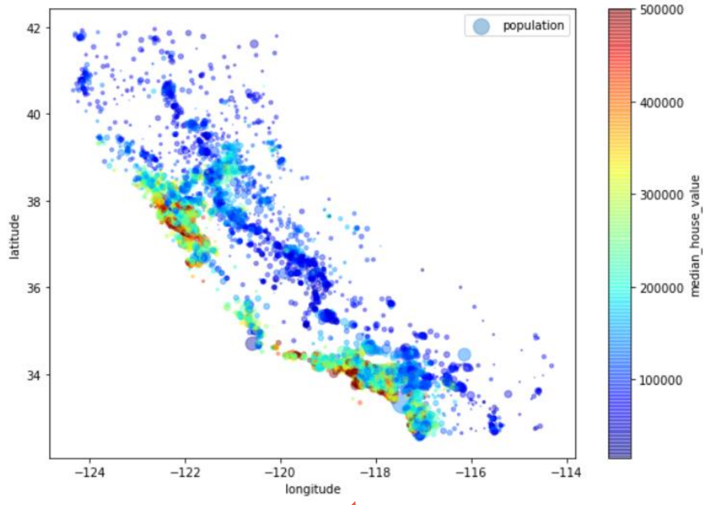
- **Visualize Geographical Data**



housing.plot(kind="scatter",
x="longitude", y="latitude")

alpha=0.1
: Represent
Dense area





s : population / c : price

: Use jet to color out
the range of prices

03 Discover and Visualize the Data

- **Correlations**

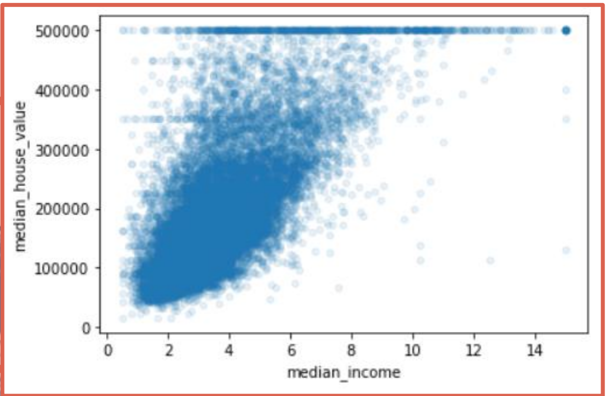
```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

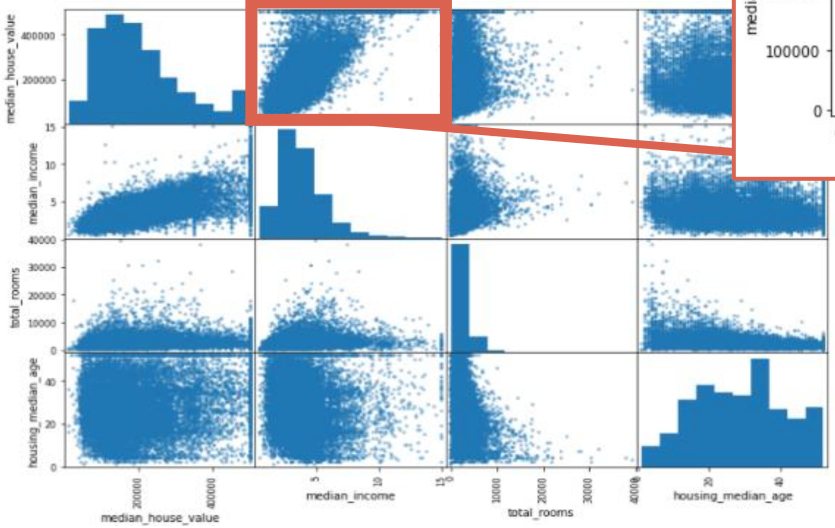
```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64
```

Standard Correlation Coefficient
Pearson's r : corr()

Represening the correlation
between categories
range : -1 to 1

Positive Correlation





Scatter_matrix : Scatter plots between numeric attributes

04 Prepare the Data for Machine Learning Algorithms

- **Data Cleaning**

Handle Null value : dropna(), drop(), fillna() - Imputer

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
imputer.statistics_
```

```
array([-118.51   ,   34.26   ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

```
X = imputer.transform(housing_num)
```

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing_num.index)
```

```
housing_tr.loc[sample_incomplete_rows.index.values]
```

	longitude	latitude	housing_median_age	total_rooms	total_b
4629	-118.30	34.07	18.0	3759.0	
6068	-117.86	34.01	16.0	4632.0	
17923	-121.97	37.35	30.0	1955.0	
13656	-117.30	34.05	6.0	2155.0	
19252	-122.79	38.48	7.0	6837.0	

- **Handling Text & Categorical Attributes**

One Hot Encoder : Transform categorical value into one hot vector

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
housing_cat_1hot.toarray()
```

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

04 Prepare the Data for Machine Learning Algorithms

- **Feature Scaling**

  - **Normalization (Min-max Scaler)**

  : Makes it equal to the range of all attributes scale value from 0 to 1
  -> adjust with 'feature_range'

  - **Standardization (Standard Scaler)**

  : No upper lower bound in range
  -> Less affected by outliers

- **Transformation Pipeline**

  - **Pipeline Class**

  : Preprocess the numerical characteristics

  - **Column Transformer**

  : Transform categorical and numeric columns at once

```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

```
housing_prepared
```

```
array([[-1.15604281,  0.77194962,  0.74333089, ...,  0.        ,
         0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.        ,
         0.        ,  0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, ...,  0.        ,
```

05 Select and Train Model

- **Training and Evaluating on the training set**

- Linear Regression

```
from sklearn.linear_model import LinearRegression

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

```
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

print("예측:", lin_reg.predict(some_data_prepared))
```

```
예측: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

```
print("레이블:", list(some_labels))
```

```
레이블: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

1) RMSE

```
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
```

```
68628.19819848922
```

2) MAE

```
from sklearn.metrics import mean_absolute_error

lin_mae = mean_absolute_error(housing_labels, housing_predictions)
lin_mae
```

```
49439.89599001897
```

05 Select and Train Model

- **Training and Evaluating on the training set**

- Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

```
DecisionTreeRegressor(random_state=42)
```

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

```
0.0
```

- K-fold Cross-Validation

: Split into 10 subsets called fold
and select fold each time to use for evaluation
and use the remaining 9 fold for training

- Random Forest Regressor

: Randomize properties to create many decision trees
and operate by averaging predictions

06 Fine tune your model

- **Grid Search CV**

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    # 12(=3×4)개의 하이퍼파라미터 조합을 시도합니다.
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # bootstrap은 False로 하고 6(=2×3)개의 조합을 시도합니다.
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# 다섯 개의 폴드로 훈련하면 총 (12+6)*5=90번의 훈련이 일어납니다.
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)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
             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')
```

```
grid_search.best_params_
```

```
{'max_features': 8, 'n_estimators': 30}
```

```
grid_search.best_estimator_
```

```
RandomForestRegressor(max_features=8, n_estimators=30, random_state=42)
```

Find the best combination of hyperparameters
=> Ideal for small number of combination navigation

- **Randomized Search CV**

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error', ra
rnd_search.fit(housing_prepared, housing_labels)
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
                   param_distributions={'max_features': <scipy.stats._distn_infrastr
ucture.rv_frozen object at 0x7ff4c8035550>,
                                        'n_estimators': <scipy.stats._distn_infrastr
ucture.rv_frozen object at 0x7ff4c80358d0>},
                   random_state=42, scoring='neg_mean_squared_error')
```

```
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
49150.70756927707 {'max_features': 7, 'n_estimators': 180}
51389.889203389284 {'max_features': 5, 'n_estimators': 15}
50796.155224308866 {'max_features': 3, 'n_estimators': 72}
50835.13360315349 {'max_features': 5, 'n_estimators': 21}
49280.9449827171 {'max_features': 7, 'n_estimators': 122}
50774.90662363929 {'max_features': 3, 'n_estimators': 75}
50682.78888164288 {'max_features': 3, 'n_estimators': 88}
49608.99608105296 {'max_features': 5, 'n_estimators': 100}
50473.61930350219 {'max_features': 3, 'n_estimators': 150}
64429.84143294435 {'max_features': 5, 'n_estimators': 2}
```

Evaluate a specified number of units for each iteration
=> Suitable when hyperparameter navigation is large

06 Fine tune your model

- **Evaluate system on test set**

```
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)
```

```
final_rmse
```

```
47730.22690385927
```

07 Launch, monitor and maintain your system

1) Launch : Connect the input data source and write the test code

2) Monitor : Check system performance at regular intervals
and notify alarm when performance falls

3) Maintain : Model training is required regularly using new data
-> should be automated

THANK YOU